Sémantisation à la volée de nuages de points 3D acquis par systèmes embarqués

Xavier Roynard

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

Sémantisation à la Volée de Nuages de Points 3D acquis par Systèmes Embarqués

Soutenue par
Xavier Roynard
Le 03 juin 2019

École doctorale n°621
Ingénierie des Systèmes, Matériaux, Mécanique, Energétique

Spécialité
Informatique Temps-Réel, Robotique et Automatique

Composition du jury :

Paul Checchin	Professeur, Université Clermont Auvergne (UCA)	*Rapporteur*
Bruno Vallet	Directeur de Recherche, Institut national de l'information géographique et forestière (IGN)	*Rapporteur*
Martin Weinmann	Assistant Professor, Karlsruhe Institute of Technology (KIT)	*Examinateur*
Beatriz Marcotegui	Professeur, Mines ParisTech	*Présidente*
Jean-Emmanuel Deschaud	Chargé de Recherche, Mines ParisTech	*Examinateur*
François Goulette	Professeur, Mines ParisTech	*Directeur de thèse*

MINES ParisTech | PSL

Remerciements

Je tiens avant tout à remercier mes encadrants de thèse François Goulette et Jean-Emmanuel Deschaud pour m'avoir accompagné durant ces trois ans et pour avoir relu attentivement mon manuscrit de thèse. Et plus généralement, je tiens à remercier toutes les personnes qui ont pu contribuer à cette thèse d'une façon ou d'une autre.

Un grand merci a tous les membres du CAOR où s'est déroulée cette thèse et en particulier les autres doctorants sans qui le temps aurait paru bien trop long. Je me souviendrai entre autres des nombreuses baffes données et reçues lors des parties de jeux de société improvisées en V020 ter, mais aussi des parties de ping-pong aux règles plus que douteuses avec la team Terra, ou encore les diableries surprises de Hugues.

Plus généralement je remercie tous les doctorants rencontrés durant cette thèse dont les Bellifontains et les membres de DOPAMINES. Je me souviendrai incontestablement des voyages au ski réalisés avec eux, des week-ends à Fontainebleau ou encore des différents salons à la porte de Versailles.

Je remercie mes colocs Charles et Paul qui ont rendu cette longue période de rédaction moins solitaire par leurs encouragements et leur présence (pas le choix) quotidienne.

Comment ne pas parler du pub O'Prince ? On s'y sentait comme à la maison dès la porte franchie grâce à Ronan et Claire. Il a été un lieu de décompression, de détente et de joie en toute occasion. Mon seul regret, ne pas avoir pu y fêter ma soutenance de thèse.

Il m'est impossible d'oublier mes amis les plus proches, que je les ai rencontré durant cette thèse, en prépa, au lycée ou bien avant. Ils m'ont toujours soutenu et encouragé. Pour tout cela je les remercie.

Je remercie enfin ceux qui me sont le plus chère, ma famille que j'espère ne pas avoir trop délaissée durant cette thèse. En particulier ma sœur Cécile et mes parents Patrice et Élisabeth.

Table des matières

Liste des figures

Liste des tableaux

Introduction

Le développement des véhicules autonomes a explosé depuis quelques années, en effet ils promettent d'avoir un impact important sur la société, en réduisant le nombre d'accidents sur les routes et en fluidifiant le trafic dans les agglomérations.

Tous les constructeurs « classiques » planchent actuellement sur le sujet, mais également des acteurs plus exotiques qui tirent profits de leurs compétences en traitement de données ou en intelligence artificiels comme Google, Apple, Nvidia ou Uber.

Cette thèse est réalisée au Centre de Robotique de l'école des Mines de Paris, plus précisément dans l'équipe Nuages de Points et Modélisation 3D (NPM3D). Ce centre de recherche a une forte composante autour des véhicules autonomes que ce soit terrestres (avec une chaire autour de la voiture autonome) ou aérien (avec une plateforme LiDAR sur drone voir figure 1).

FIGURE 1 – Plateforme expérimentale LiDAR sur drone pour l'évaluation des méthodes de perception temps-réel en embarqué. Un drone équipé d'une nacelle comportant un LiDAR et un ordinateur embarqué.

Un véhicule autonome est avant tout un robot, qui a entre autres besoin de comprendre son environnement, c'est ce qu'on appelle la « perception ». La perception est réalisée grâce à des capteurs (voir figure 2 page suivante) qui produisent un flux des données. Ces données doivent alors être traitées pour que le véhicule puisse prendre des décisions sur la trajectoire à suivre.

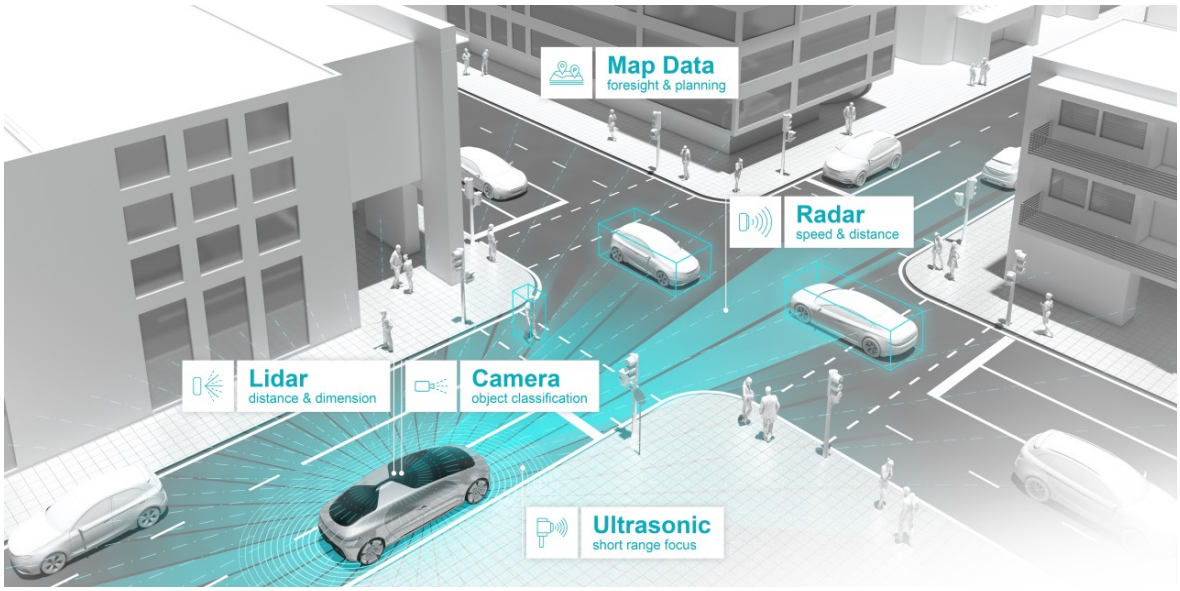


FIGURE 2 – Les capteurs de la voiture autonome.

Les véhicules autonomes peuvent également avoir besoin de cartes des zones dans lesquelles ils circulent, ces cartes contiennent plus d'informations que des cartes routières, en particulier elles contiennent de l'information en trois dimensions et de l'information sémantique utile au véhicule comme la position des voies de circulation, des panneaux de signalisation... Ces cartes qu'on appellera par la suite *HD-Maps* peuvent être produites à partir de nuages de points 3D.

Dans l'équipe NPM3D on s'intéresse particulièrement aux capteurs LiDAR qui produisent des nuages de points 3D (voir figure 3) que ce soit pour la construction de *HD-Maps* ou la perception pour les véhicules autonomes.

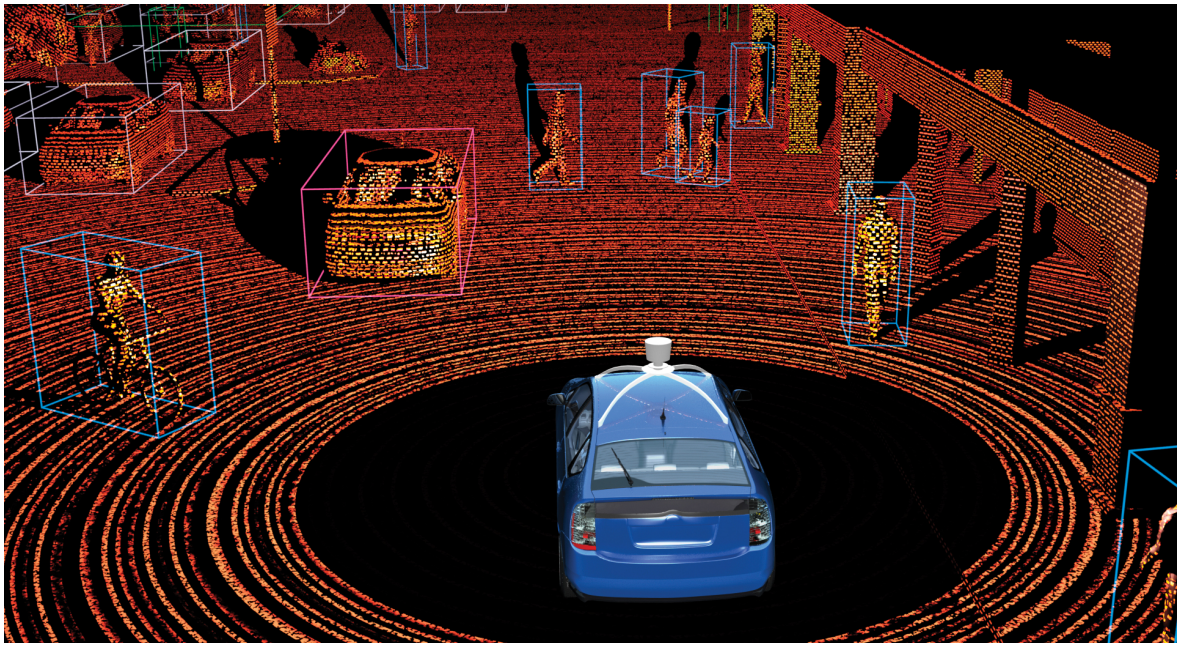


FIGURE 3 – Nuage de point acquis par un LiDAR 3D rotatif sur un véhicule autonome. Les boites englobantes autour de chaque objet sont une des sorties désirées pour la perception du véhicule autonome.

Cette thèse présente des contributions (à la perception pour les véhicules autonomes) sur deux points :

— améliorer la classification en *offline*, pour faire des *HD-Maps*,

— faire de la classification temps-réel en *online*.

Ces tâches requièrent des algorithmes de perception performants capables de traiter de grandes quantités de nuages de points.

Depuis 2012, les méthodes d'apprentissage profond et en particulier de réseaux de neurones convolutionnels ont révolutionné le traitement des images 2D, d'abord pour la classification d'images [KRIZHEVSKY et collab., 2012], puis la segmentation et la détection d'objets sur des images pour se tourner vers des tâches toujours plus complexes.

Cependant, ces méthodes ne produisent pas des résultats aussi spectaculaires sur les nuages de points 3D qui sont des données non denses et non structurées dans l'espace.

De plus pour la segmentation sémantique des scènes complètes de nuages de points 3D, les jeux de données existants présentent plusieurs défauts :

1. ils ne sont pas en quantité ou qualité suffisante pour entraîner de telles réseaux,
2. ils sont très déséquilibrés en nombre d'échantillons par classe (de plusieurs millions pour le sol et les bâtiments à quelques milliers pour les piétons et signalisations),
3. certaines classes possèdent énormément de points redondants, par exemple la plupart des points du sol sont sur une surface plane.

On traitera le point 1 dans le chapitre 1, où après un état de l'art des jeux de données de nuages de points 3D existant, on présente un nouveau jeu de données de nuages de points 3D annotés réalisé par acquisition mobile pour produire suffisamment de données et annoté à la main pour assurer une qualité suffisante de la segmentation.

Les points 2 et 3 seront traités par la première contribution du chapitre 2, qui est une nouvelle méthode d'entraînement de réseaux de neurones profonds sur des scènes de nuages de points 3D annotés.

La deuxième contribution du chapitre 2 traitera du problème du manque de contexte dans la définition d'un voisinage fixe en proposant des architectures de réseaux de neurones convolutionnels capables de tirer profit des voisinages multi-échelles.

Ces méthodes d'apprentissage profond sont certes plus performantes, mais peuvent demander beaucoup de puissance de calculs ce qui nous empêche en l'état de les utiliser pour de la segmentation sémantique de données 3D en temps-réel.

Les contraintes liées aux systèmes embarqués comme les véhicules autonomes (ou robot autonome), sont encore plus grandes avec peu de puissance de calcul et d'énergie embarqués. On a donc besoin de méthodes plus efficaces que l'on développera dans le chapitre 3.

On fera ensuite une Conclusion des travaux réalisés dans cette thèse, et on proposera quelques pistes de développements futurs. On présentera une liste des Publications réalisées pendant cette thèse, puis en Annexe est présenté un État de l'Art de l'Apprentissage Profond en Classification et Segmentation Sémantique d'Images.

Références

KRIZHEVSKY, Alex, Ilya SUTSKEVER et Geoffrey E HINTON (2012). « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems*, p. 1097-1105 (cf. p. 3, 119, 137).

Chapitre 1

Jeux de Données de Nuages de Points 3D

« *It is a capital mistake to theorize before one has data* »

Sherlock Holmes

Sommaire

1.1 Introduction

De nombreux jeux de données sont utilisés pour l'entraînement et la comparaison des algorithmes d'apprentissage machine. Les données fournies permettent d'entraîner des méthodes qui réalisent une tâche donnée, et l'évaluation des performances sur un ensemble de tests permet d'évaluer la qualité des résultats obtenus et de comparer les différentes méthodes en fonction de différentes métriques.

De nombreuses tâches différentes peuvent être apprises, la plus courante étant la classification (par exemple, pour une image, il s'agit de trouver la classe de l'objet principal visible sur l'image). Une autre tâche peut être de segmenter les données en ses parties pertinentes (pour les images, il s'agit de regrouper tous les pixels qui appartiennent au même objet). Il y a de multiples autres tâches qui peuvent être apprises, de l'analyse d'images à la traduction dans le traitement du langage naturel (pour plus d'informations voir [FERRARO et collab., 2015]).

Il existe un grand nombre de jeux de données dans de nombreux domaines. Chaque jeu de données peut être différencié sur plusieurs critères : le type de données (image, son, texte, nuages de points, graphes), la quantité (de centaines à des milliards d'échantillons), la qualité, le nombre de classes (de dizaines à des milliers) et les tâches à apprendre. Parmi les plus utilisés il y a :

— les jeux de données de classification et segmentation d'image : ImageNet [1] [DENG et collab., 2009], MS COCO [2] [LIN, MAIRE et collab., 2014],

— les jeux de stéréo-vision pour l'estimation de carte de profondeur : Middlebury Stereo Datasets [3] [SCHARSTEIN et collab., 2014],

— les jeux de données de vidéos : Youtube-8M [4] [ABU-EL-HAIJA et collab., 2016],

— les jeux de données d'odométrie, de stéréo-vision, de flux optique et de détection d'objets 3D : KITTI [5] [GEIGER, LENZ et URTASUN, 2012],

— les jeux de données de SLAM (*Simultaneous Localisation And Mapping* pour Localisation et Cartographie Simultanée) : Ford Campus Vision and Lidar Data Set [PANDEY et collab., 2011],

— les jeux de données de localisation à long terme : the Oxford Robotcar Dataset [MADDERN et collab., 2017] and the NCLT Dataset [CARLEVARIS-BIANCO et collab., 2016],

— les jeux de données de segmentation d'image de scènes urbaine : The Cityscapes Dataset [CORDTS et collab., 2016], The BDD100K Dataset [F. YU et collab., 2018]

Plus proche de notre domaine de recherche on trouve des jeux de données produit par LiDAR aéroporté (ALS pour *Airborne Laser Scanning* en anglais) comme 3D Semantic Labeling Contest [6] [NIEMEYER et collab., 2014].

Nous présenterons un état de l'art des jeux de données de nuages de points 3D annotés en section 1.2.1 page suivante et un état de l'art des jeux de données utiles pour le véhicule autonome contenant entre autres des nuages de points 3D en section 1.2.2 page 15. On mettra en avant les avantages et inconvénients de chaque jeu de données, en particulier les critères qui nous ont poussé à produire notre propre jeu de données. Il est important de noter que certains jeux de données bien que très intéressant de par leur qualité et la quantité de leurs données ne seront pas présentés ici car ils ne contiennent pas de nuages de points 3D, on fait référence par exemple à : BDD100K, Mapillary Vistas Dataset, Cityscapes.

Puis en section 1.3 page 23 on présentera un jeu de données de nuages de points 3D annoté réalisé pendant cette thèse qui comble certains défauts des jeux de données de l'état de l'art. Enfin en section 1.4 page 31 on présentera des idées, perspectives et lignes directrices pour créer un jeu de données (idéal et complet) pour le véhicule autonome.

1. http://www.image-net.org/
2. http://mscoco.org/
3. http://vision.middlebury.edu/stereo/data/
4. https://research.google.com/youtube8m/
5. http://www.cvlibs.net/datasets/kitti/index.php
6. http://www2.isprs.org/commissions/comm3/wg4/3d-semantic-labeling.html

1.2 Jeux de Données de Nuages de Points 3D Existants

Nous nous intéressons ici plus précisément aux jeux de données de nuages de points 3D représentant des scènes d'intérieur ou d'extérieur acquis par LiDAR.

On peut distinguer trois méthodes d'acquisition :

— par scanner mobile (MLS pour *Mobile Laser Scanning*), avec un LiDAR monté sur véhicle terrestre ou un drone. Pour recaler les nuages, la pose 6D précise du véhicule doit être connue, soit grâce à un système de navigation inertiel et GPS, soit grâce à un algorithme de SLAM (pour *Simultaneous Localisation and Mapping*).

— par scanner fixe (TLS pour *Terrestrial Laser Scanning*), on obtient un nuage à 360° autour d'un point de vue fixe. Le LiDAR doit être déplacé entre chaque acquisition et les nuages doivent être recalés entre eux pour obtenir une scène complète.

— par scanner aéroporté (ALS pour *Airborne Laser Scanning*), ici encore la pose 6D précise du véhicule doit être connue. Cependant cette méthode d'acquisition ne permet pas d'obtenir une densité de points et une précision suffisantes à cause de la distance et de l'angle d'acquisition.

Il existe encore assez peu de jeux de données de scènes de nuages de points 3D annotés. Ces jeux de données sont assez hétérogènes et ont chacun des caractéristiques intéressantes mais aussi des défauts. Nous présenterons donc également un jeu de données réalisé pendant cette thèse appelé **Paris-Lille-3D** et qui en est une contribution. Dans les sections suivantes nous réalisons une comparaison des jeux de données de l'état de l'art tout en identifiant les forces et les faiblesses pour la tâche de segmentation sémantique. Le tableau 1.1 présente un résumé quantitatif de cette comparaison.

Nom	Type de capteur	Surface couverte (longeur)	Nombre de points	Nombre de classes
S3DIS	caméra RGB-D	5944m^2(–)	695,9M	13
ScanNet	caméra RGB-D	34456m^2(–)	5581M	20
Oakland	MLS mono-fibre	2694m^2(1510m)	1,61M	5
Semantic3D	LiDAR fixe	40980m^2(–)	1660M	8
Paris-rue-Madame	MLS multi-fibre	2492m^2(160m)	20M	17
IQmulus	MLS mono-fibre	2517m^2(210m)	12M	22
Paris-Lille-3D	**MLS multi-fibre**	42714m^2(1940m)	143.1M	50

TABLEAU 1.1 – Comparaison des jeux de données de nuages de points 3D annotés. Les deux premiers jeux de données sont des scènes d'intérieur, les autres sont des scènes d'extérieur. Ces jeux de données sont présentés en section 1.2.1 excepté **Paris-Lille-3D** qui est le jeu de données créé dans cette thèse et qui sera présenté en section 1.3 page 23.

1.2.1 Datasets de nuages de points de scènes annotées

1.2.1.1 Oakland 3-D Point Cloud Dataset [Delfina MUNOZ et collab., 2009]

Ce jeu de données a été acquis par un système MLS constitué d'un LiDAR Sick® mono-fibre de telle sorte que son champ de vision permette de voir la route uniquement du côté droit du véhicule. Ce système d'acquisition présente plusieurs défauts (cf figure 1.1 page suivante) :

— le taux de rafraîchissement du LiDAR est assez faible, ce qui donne une densité de points faible à distance moyenne. Pour le premier jeu de données de ce type c'est cependant tout à fait remarquable,

— le LiDAR étant mono-fibre (donc avec un unique angle d'incidence pendant le balayage de la scène), il y a de grosses occlusions sur les données d'arrière plan (comme les bâtiments) quand un objet se trouve au premier plan (un arbre ou un lampadaire par exemple),

— la faible densité ne permet pas de distinguer des objets très fins comme des potelets ou de distinguer le trottoir de la route, par conséquence seul un nombre restreint de classes sont annotées, seulement les classes les plus volumineuses (voitures bâtiment, sol, arbre, lampadaire ...).

— l'intégralité du jeu de données se trouve sur la même route, il y a donc très peu de diversité dans les différentes instances d'objets d'une même classe (les lampadaires sont tous de la même forme).

De plus ce jeu de données contient 44 classes dont une grosse partie a moins de 1000 points annotés. Au final plusieurs classes sont fusionnées et seules 5 classes sont gardées pour entraîner et tester l'algorithme de classification de l'article original.

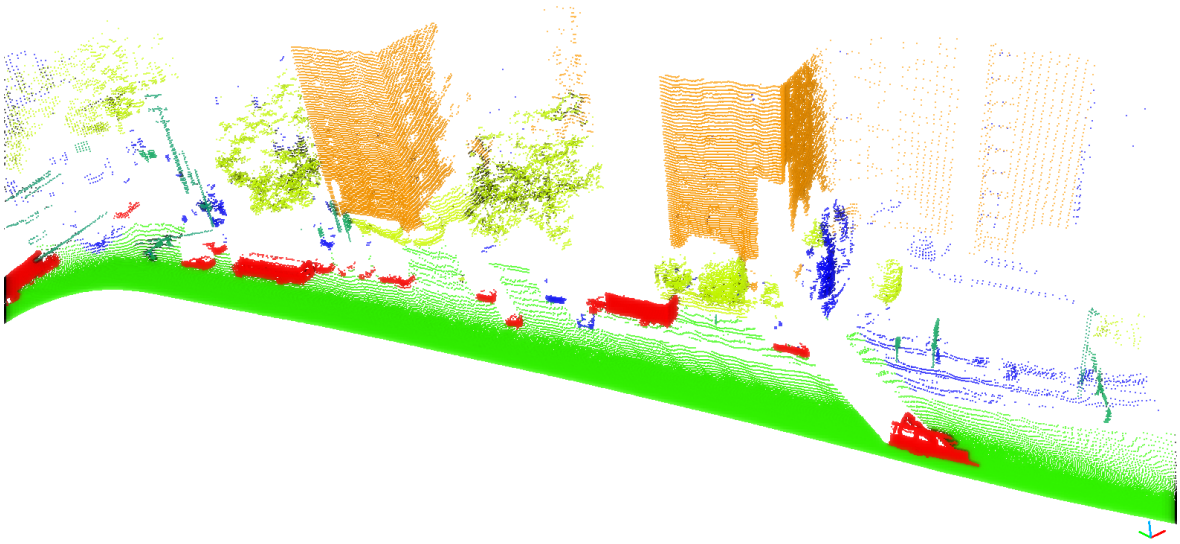


FIGURE 1.1 – Exemple de nuage de points du jeu de données *Oakland*. On observe une densité faible, peu de classe, de grandes occlusions derrières les arbres (dues au LiDAR mono-fibre).

1.2.1.2 Semantic3D [HACKEL, SAVINOV et collab., 2017]

Ce jeu de données a été acquis par scanner LiDAR fixe, il est donc bien plus précis et dense qu'un jeu de données acquis par MLS, mais il présente des inconvénients inhérents au LiDAR statique (cf figure 1.2 page suivante) :

— La densité de points varie considérablement en fonction de la distance entre le LiDAR et la surface scannée, ceci peut mettre en difficulté certaines méthodes, mais peut aussi permettre de tester leur robustesse.

— De la même façon qu'avec un scanner mono-fibre mobile, il y a de nombreuses occlusions. La position fixe du scanner n'arrange pas les choses, même en recalant plusieurs nuages acquis de différents points de vues.

— Les temps d'acquisition sont bien plus importants que par MLS, ce qui empêche d'obtenir des jeux de données très volumineux et de scènes très diversifiées. En particulier cette méthode ne peut pas raisonnablement être utilisée pour scanner l'intégralité des routes d'une ville comme Paris, ce qui serait nécessaire pour créer les HD-Maps indispensables aux véhicules autonomes.

De plus seules des classes suivantes assez grossières sont annotées :

sol artificiel	sol naturel	végétation haute	végétation basse
bâtiments	mobilier urbain	artefacts d'acquisition	voitures

Ainsi toutes les formes de mobilier urbain sont regroupées dans une même classe, et les « objets » mobiles comme les piétons, vélos et voitures (non-garées) sont regroupés dans la classe *artefacts d'acquisition*.

Cependant l'annotation est de très bonne qualité, la frontière des objets est précise ce qui n'est pas le cas de tous les jeux de données, en particulier ceux utilisant des méthodes d'annotation semi-automatique comme on le verra par la suite.

De plus chaque point a une information de réflectance ou intensité du retour laser ainsi qu'une information de couleur obtenu grâce à une caméra RGB. Un avantage du scanner fixe est que l'axe optique de la caméra peut être placé exactement dans la direction du tir laser et donc éviter tout décalage entre les données couleur et la géométrie de la scène comme on peut le voir sur des données acquises en MLS.

Ce jeu de données a également le mérite de proposer un benchmark/challenge publique qui permet d'évaluer et de comparer les méthodes de segmentation sémantique sur des données de test dont l'annotation n'est pas publique.

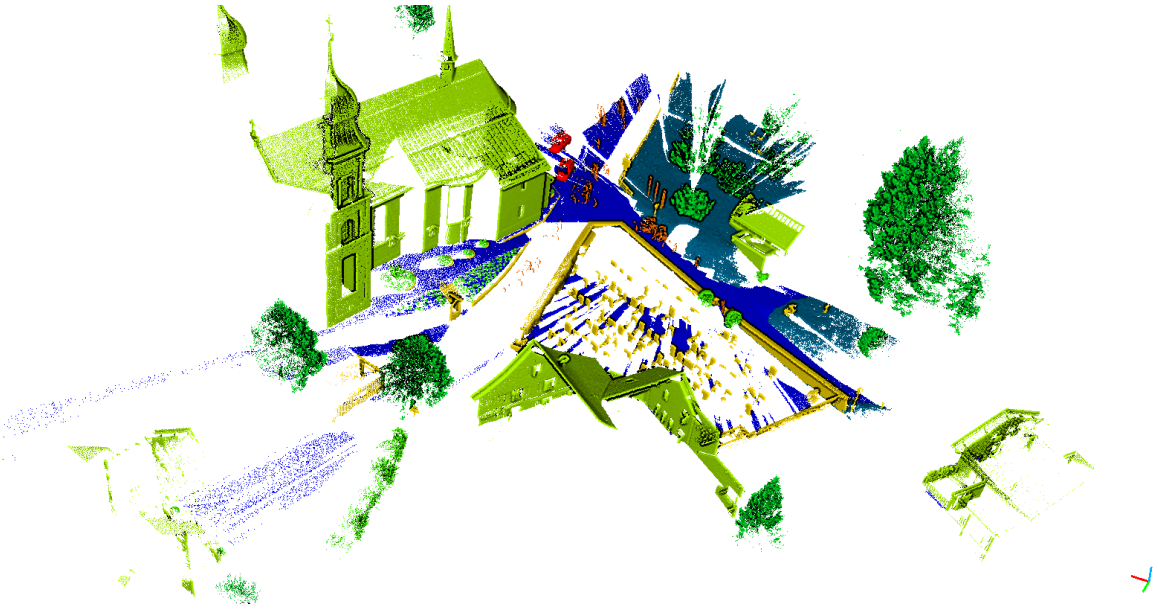


FIGURE 1.2 – Exemple de nuage de points du jeu de données Semantic3D. On peut observer de nombreuses occlusions et la densité dépend de la distance au LiDAR. Chaque classe est représentée par une couleur différente.

1.2.1.3 Paris-rue-Madame Database [SERNA, MARCOTEGUI et collab., 2014]

Ce jeu de données a été acquis grâce à une version plus récente du système MLS expérimental [GOULETTE et collab., 2006] du Centre de Robotique de l'école des Mines de Paris qui est décrit en section 1.3.1 page 23. L'annotation de ce jeu de données a été faite de façon semi-automatique, une pré-segmentation est réalisée par la méthode utilisée dans [SERNA, MARCOTEGUI et collab., 2014], puis l'annotation est affinée à la main par un expert. Mais malgré cette seconde étape il reste des inexactitudes de segmentation sur les contours des objets (cf figure 1.3 page ci-contre), en particulier le bas des objets (par exemple les roues des voitures) est annoté comme appartenant au sol.

Le système d'acquisition utilise un LiDAR multi-fibres Velodyne HDL-32E qui permet en un seul passage de scanner l'avant et l'arrière des objets contrairement à un LiDAR mono-fibre. Mais

le système de localisation est loin d'être parfait, on peut observer un nuage très bruité et déformé en figure 1.3.

Ce jeu de données a l'avantage de contenir de nombreuses classes, mais la plupart des classes ne contiennent que très peu de points. Il est donc difficile d'utiliser ce jeu de données pour entraîner des réseaux de neurones profonds par exemple.

Enfin ce jeu de données ajoute à chaque point un label d'instance d'objet, en plus de la tâche de segmentation sémantique, il peut donc également permettre d'apprendre la tâche de détection d'objet.

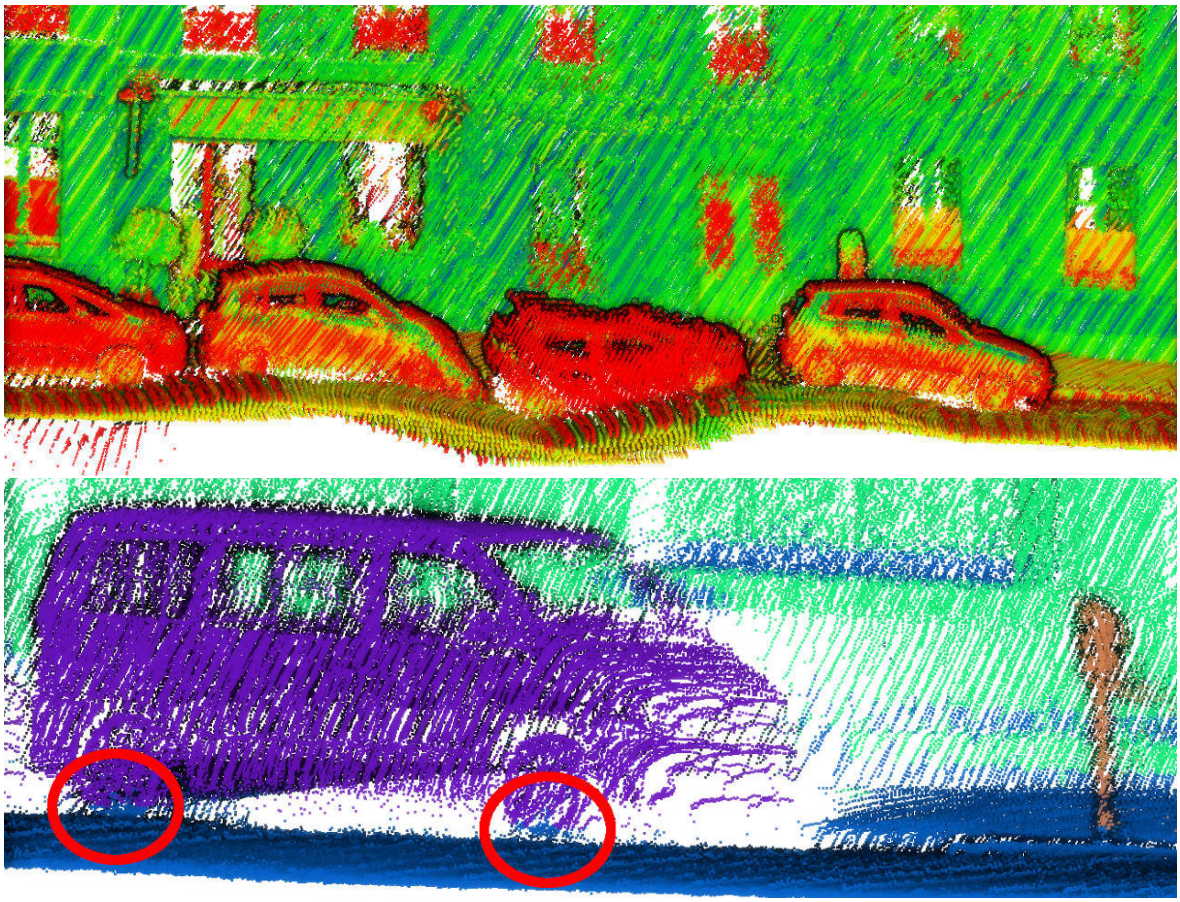


FIGURE 1.3 – Exemple de nuages de points du jeu de données Rue-Madame. On peut observer un nuage de point très bruité (en haut) et des erreurs d'annotation (entourées en rouge en bas).

1.2.1.4 IQmulus & TerraMobilita Contest [VALLET et collab., 2015]

Ce jeu de données a été acquis par un système MLS conçu par l'IGN [PAPARODITIS et collab., 2012] équipé d'un LiDAR monofibre Riegl LMS-Q120i. Il présente donc certains défauts du jeu de données *Oakland* (cf section 1.2.1.1 page 8), mais le LiDAR utilisé à l'avantage d'avoir une meilleure fréquence de tirs laser, ainsi la densité du nuage est suffisante pour observer la géométrie fine de la scène. De plus ce LiDAR est également plus précis qu'un LiDAR multi-fibres comme le Velodyne HDL-32E, il permet donc de produire des nuages beaucoup plus propres et plus fins, mais est bien plus onéreux (de 2 à 3 fois plus à l'époque de l'acquisition, maintenant on est capable de concevoir des système MLS à LiDAR(s) multi-fibres 50 fois moins onéreux).

Pour l'annotation un éditeur web a été utilisé par l'IGN pour rendre accessible cette tâche à des personnes non expertes, au lieu de travailler directement sur le nuage de points, l'annotation a été faite sur des images de réflectance du LiDAR. Chaque ligne de l'image est l'information de réflectance obtenue pendant une révolution du LiDAR, on perd donc totalement l'information de profondeur. Cet outil a permis de faciliter et d'accélérer considérablement l'annotation, mais

a également causé des erreurs d'annotation non-négligeables. En effet l'annotation des contours des objets étant approximative sur une image, une fois re-projetée sur le nuage, on peut observer au bord des occlusions des classes incorrectes (voir figure 1.4).

On notera que sur les 140M points annotés seuls 12M sont publiques, les autres sont utilisés pour faire un benchmark de comparaison des méthodes de segmentation sémantique, ce benchmark a été ouvert jusqu'en 2016 mais n'a eu malheureusement que 2 équipes participantes. Les résultats décrits dans l'article en détail [VALLET et collab., 2015], montrent à l'époque une supériorité de la méthode du CMM-Mines ParisTech (basée sur une segmentation, puis une classification de chaque instance d'objet) sur la méthode du IPF-KIT (basée sur une classification par points). Il aurait été intéressant de pouvoir comparer plus de méthodes, ainsi que l'évolution des algorithmes de l'état de l'art au cours des années. D'autant plus que ce jeu de données aurait également permis d'évaluer la tâche de détection puisque un label d'instance d'objet est disponible pour chaque point, et aucune équipe de recherche ne s'est confrontée à cette tâche avant la fermeture du benchmark.

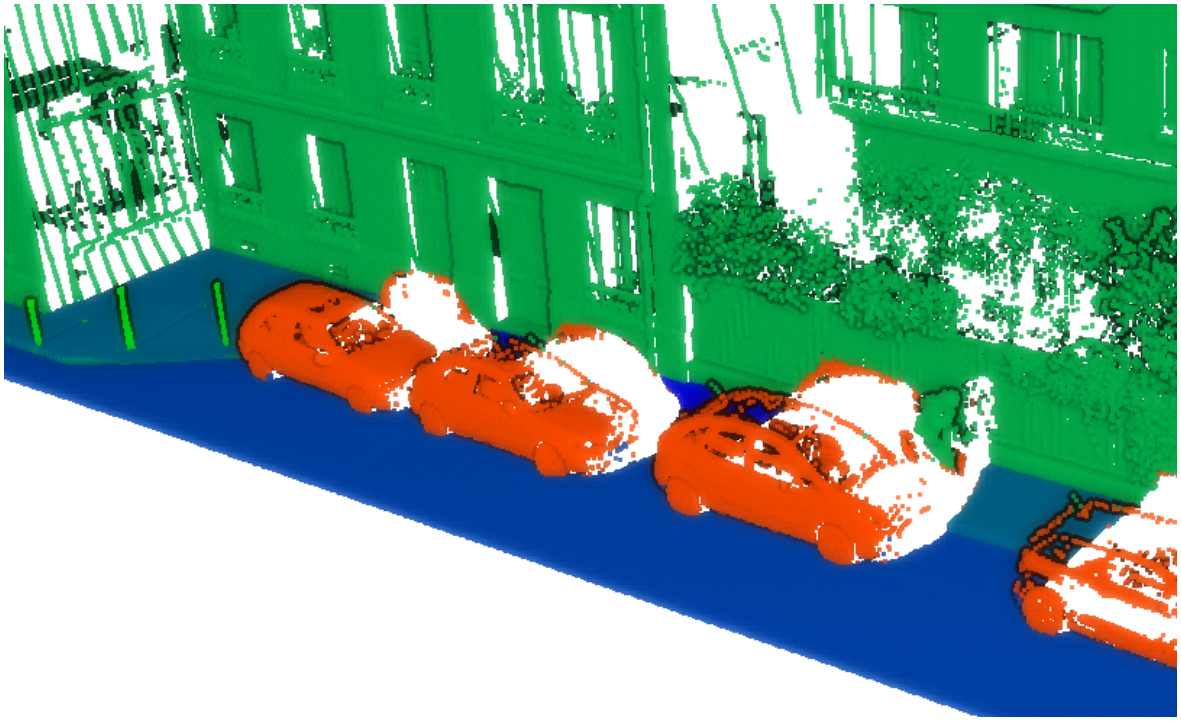


FIGURE 1.4 – Exemple de nuage de points du jeu de données *iQmulus & TerraMobilita*. On peut observer des occlusions derrière les objets de premier plan et des erreurs d'annotations, le mur derrière les voitures est classé en *voiture*.

On présente également deux jeux de données acquis en intérieur, car il semble intéressant de vérifier que des méthodes capables d'apprendre à réaliser la segmentation sémantique d'une scène en extérieur le soient aussi en intérieur et vice versa.

1.2.1.5 S3DIS [ARMENI, SENER et collab., 2016]

Ce jeu de données a été acquis en intérieur grâce à un système Matterport. Ce système combine trois caméras RGB-D (donnant une information de couleur et de profondeur) sur un support rotatif à 360°. Une acquisition consiste en une rotation du système comme pour un scanner LiDAR fixe. Les caméras RGB-D qui sont des capteurs performants à courtes distances (moins de 5m) sont bien adaptées et moins onéreuses pour des acquisitions en intérieur.

Le jeu de données est composé de 6 scènes acquises chacune dans un étage de bâtiment différent. La surface couverte est d'environ 6000m^2 et 215M points ont été acquis, ce qui peut paraître considérable, cependant la variabilité n'est pas très importante puisque les scènes ont toutes été acquises dans le même campus et dans seulement 3 bâtiments différents.

Les points sont classés dans une des classes sémantiques suivantes :

plafond	sol	mur	colonne	poteau
fenêtre	porte	table	chaise	bibliothèque
divan	tableau	artefacts/autres		

Ces classes décrivent bien la scène à "haut niveau", mais ne rentrent pas dans les détails, par exemple toutes les fournitures de bureaux (écrans et ordinateurs, poubelles, claviers, souris...) sont regroupées dans une même classe *artefacts/autres*. Par conséquent plus de 10% des points se retrouvent dans cette classe. Pour un aperçu des nuages du jeu de données voire figure 1.5.

En plus les 270 pièces du jeu de données sont annotées parmi les 11 classes sémantiques suivantes :

bureau	salle de conférence	auditorium	entrée/vestibule
salon	couloir	salle d'impression	salle à manger
espace ouvert	espace de rangement	toilettes	

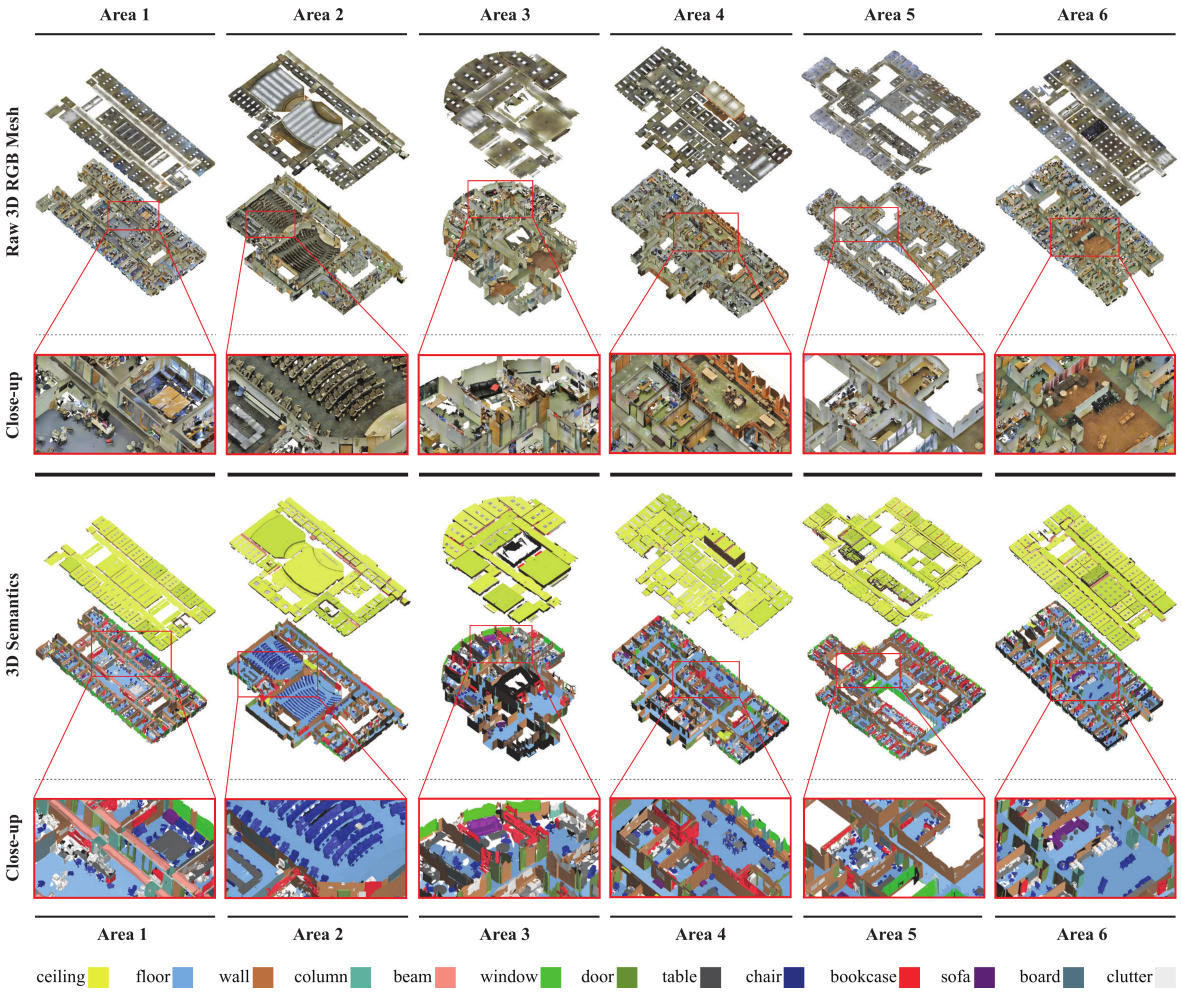


FIGURE 1.5 – Exemple de nuage de point du jeu de données S3DIS. Vue du dessus, en haut le nuage de point colorisé, en bas chaque classe est représentée par une couleur différente. Le toit a été supprimé pour une meilleure visualisation. (image issue de l'article [ARMENI, SENER et collab., 2016])

Mais la plupart des pièces sont classées *bureau* (57.7%), *couloir* (23.6%) ou *espace de rangement* (7.0%), les 8 autres classes se partageant 12.6% des pièces annotées. Il n'est donc pas envisageable d'évaluer des méthodes de reconnaissance de scène sur ce jeu de données.

En 2017 dans [ARMENI, SAX et collab., 2017], le jeu de données a été étendu par l'ajout d'une collection d'images RGB, accompagnées de l'image de profondeur, de normales et l'annotation au pixel près dans les mêmes classes que les nuages de points. De plus un maillage 3D des scènes accompagnés d'une voxellisation des nuages de points est fourni. Un aperçu des données ajoutées est présenté en figure 1.6.

On ne peut que regretter que les résultats des différentes méthodes testées sur ce jeu de données ne soient pas regroupés/collectés/recueillis sur le site internet pour faciliter la comparaison et augmenter la visibilité des méthodes évaluées.

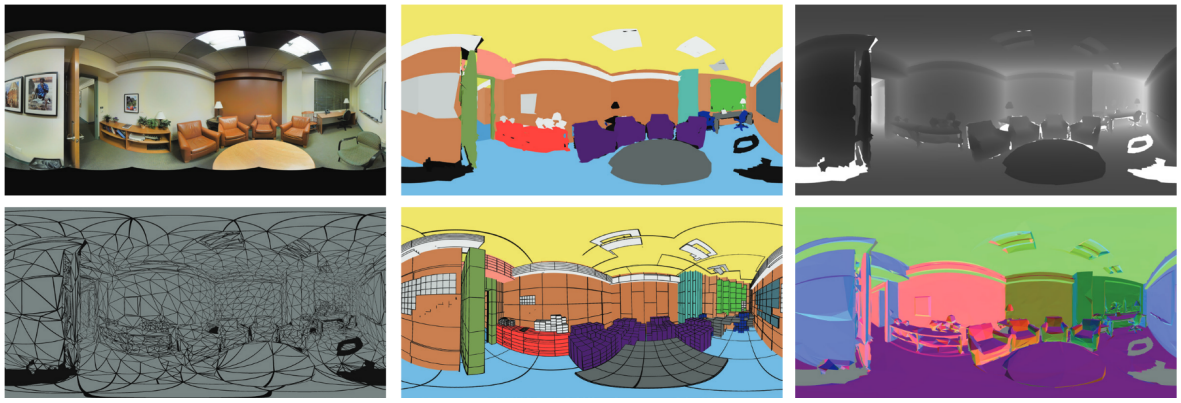


FIGURE 1.6 – Aperçu des données images ajoutées au jeu de données *S3DIS* en 2017. De gauche à droite et de haut en bas : image RGB, image annotée (une couleur par classe), image de profondeur, rendu du maillage 3D en 2D depuis le même point de vu, rendu de l'annotation des voxels 3D, image des normales aux surfaces. (image issue de l'article [ARMENI, SAX et collab., 2017])

1.2.1.6 ScanNet [DAI et collab., 2017]

Ce jeu de données a été acquis en intérieur grâce à un système de caméra RGB-D mobile dans le même principe que le MLS. Les images de profondeurs sont accumulées et recalées pour former un modèle 3D complet d'une scène grâce à un algorithme de type KinectFusion [NEWCOMBE et collab., 2011]. Contrairement au jeu de données S3DIS, une scène consiste en une unique pièce, 707 pièces uniques ont été acquises, mais 1513 scans RGB-D sont fournis dans le jeu de données (une même pièce peut avoir été scannée plusieurs fois). Cette méthode à l'avantage de permettre d'acquérir plus facilement de grandes quantités de données, mais peut générer des scènes avec beaucoup d'occlusions si l'agent réalisant le scan ne fait pas l'effort de passer autour, au-dessus et en-dessous des objets. Pour un aperçu des nuages de points voire figure 1.7 page ci-contre.

Le protocole d'acquisition a été conçu pour que tout puisse être réalisé par des novices de l'acquisition à l'annotation, et donc pour pouvoir réaliser un jeu de données annoté très important. De plus ce protocole a été rendu publique et les implémentations sont open-source pour aider la communauté à créer ses propres jeux de données.

Les nuages de points ont été segmentés en instances d'objets et chaque objet classifié parmi les 20 classes suivantes :

sol	mur	armoire	lit	chaise
sofa	table	porte	fenêtre	bibliothèque
image	comptoir	bureau	rideau	réfrigérateur
baignoire	rideau de douche	toilettes	évier	autre

De plus chaque scène a été classifiée parmi une des 13 classes suivantes :

bibliothèque/librairie	chambre/hotel	appartement	bureau	couloir
salle d'impression	salle de conférence	cuisine	buanderie	salon
salle de bain	cave/garage	divers		

Ceci permet d'évaluer la tâche de reconnaissance de scène, la quantité et la variabilité des scènes étant ici suffisante pour évaluer correctement de telles méthodes. Un benchmark est organisé pour encourager les équipes de recherche à comparer leurs méthodes sur ces tâches de segmentation sémantique 3D/2D, de détection d'objets 3D/2D et de reconnaissance de forme.

De plus, la présence de modèles 3D recalés avec les objets des scènes permet également d'évaluer la tâche de "récupération" d'objets 3D (c'est-à-dire : retrouver un modèle CAO qui représente bien l'objet). Mais pour l'instant aucun benchmark n'est organisé sur cette tâche.

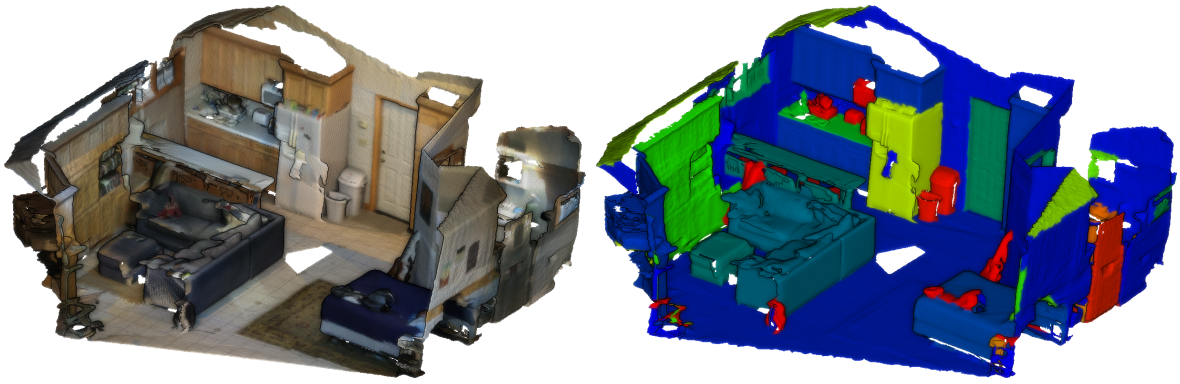


FIGURE 1.7 – Exemple de nuage de points du jeu de données ScanNet. À gauche le nuage de points colorisé, à droite chaque classe est représentée par une couleur différente.

1.2.2 Datasets pour le véhicule autonome

On s'intéresse ici aux jeux de données pour le véhicule autonome qui contiennent des nuages de points 3D. On parle de jeux de données pour le véhicule autonome car ils permettent l'apprentissage ou la validation de tâches utiles au véhicule autonome comme la localisation, la détection de marquages au sol ou la détection des autres usagers de la route, ou encore la segmentation sémantique de scènes urbaines. Ces jeux de données peuvent également contenir d'autres modalités comme des vidéos, des vidéos stéréo, des données GPS et inertielles, des données radar... Le tableau 1.2 page suivante résume les capteurs utilisés pour créer le jeu de données et les tâches pour lesquelles il a été créé.

1.2.2.1 Ford Campus Vision and Lidar Data Set [PANDEY et collab., 2011]

Ce jeu de données est un des tous premiers autour du véhicule autonome équipé d'un LiDAR 3D, il est constitué de deux séquences acquises sur le campus Ford à Dearborn dans le Michigan. Le véhicule d'acquisition est un prototype de véhicule autonome, en particulier il est équipé de :

— une caméra 360° ou sphérique Ladybug3,

— un LiDAR 360° à 64 fibres : Velodyne HDL-64E,

— deux LiDAR mono-fibre Riegl LMS-Q120,

— deux centrales inertielles (une professionnelle et une grand-public) et un GPS.

Un aperçu du jeu de données se trouve en figure 1.8 page 17. Ce jeu de données a été acquis dans l'idée d'évaluer les méthodes d'odométrie/SLAM sur un véhicule autonome, mais aucune vérité terrain n'est fournie, la génération d'une trajectoire à partir des données GPS et inertielles est laissé à l'utilisateur.

Nom	Taille	Type de données	Tâches
Ford Campus	petit	caméra 360°, LiDAR 64 fibres 360°, 2 LIDARs mono-fibre, IMU et GPS	odométrie/SLAM avec fermeture de boucles
KITTI	petit	caméras stéréo RGB et niveaux de gris, LiDAR 64 fibres 360°, IMU et GPS	stéréo-vision, calcul de flux optique, odométrie/SLAM, détection d'objets, pistage d'objets, segmentation sémantique d'images
TorontoCity	grand	LiDAR et caméra aéroportés, LiDAR 64 fibres 360° et caméra 360° sur véhicule terrestre	estimation de hauteur des bâtiments (reconstruction), extraction de courbes et d'axe routier, segmentation d'instance de bâtiments, extraction de contour de bâtiments, segmentation sémantique et classification de type de scène
Robotcar	moyen	caméra stéréo multi-baseline, 3 caméras fisheye, LiDAR 4-fibres frontal, 2 LiDARs mono-fibre, IMU et GPS	localisation à long terme
Appoloscape	grand	2 LiDARs mono-fibres, 2 caméras frontales, IMU et GPS	détection de voitures et de marquages au sol, segmentation sémantique
NCLT Dataset	moyen	caméra 360°, LiDAR 32 fibres 360°, 2 LiDARs mono-fibre, IMU et GPS	localisation à long terme, reconnaissance de scène

TABLEAU 1.2 – Comparaison des datasets pour le véhicule autonome contenant des nuages de points 3D.

Ce jeu de données peut paraître petit par rapport à ceux qui suivent, mais il a le mérite d'être un des premiers jeux de données fournissant les données de tous les capteurs d'un véhicule autonome sur des séquences complètes dont des données LiDAR multi-fibres à 360°. De plus on pourra regretter qu'aucun challenge ne soit organisé pour comparer les méthodes de l'état de l'art contrairement au jeu de données *KITTI* qui suit.

1.2.2.2 KITTI [GEIGER, LENZ et URTASUN, 2012] ; [GEIGER, LENZ, STILLER et collab., 2013]

Le jeu de données KITTI publié en 2012 et acquis en septembre 2011 est un des premiers jeux de données pour la recherche sur le véhicule autonome. La plateforme utilisée se veut comme étant similaire à ce que sera un futur véhicule autonome. En particulier elle est équipée de :

— 2 caméras RGB et 2 caméras niveaux de gris, les deux ensembles sont montés pour faire de la stéréo-vision et tournés vers l'avant,

— un LiDAR 360° à 64 fibres : Velodyne HDL-64E,

— une centrale inertielle et un GPS-RTK.

Une vérité terrain pour les images de profondeur et de flux optique a été obtenue à haute densité en recalant plusieurs scans LiDAR consécutifs par ICP [Paul J BESL et MCKAY, 1992] (un algorithme de recalage de nuages de points, en anglais *Iterative Closest Point*), puis en projetant le nuage

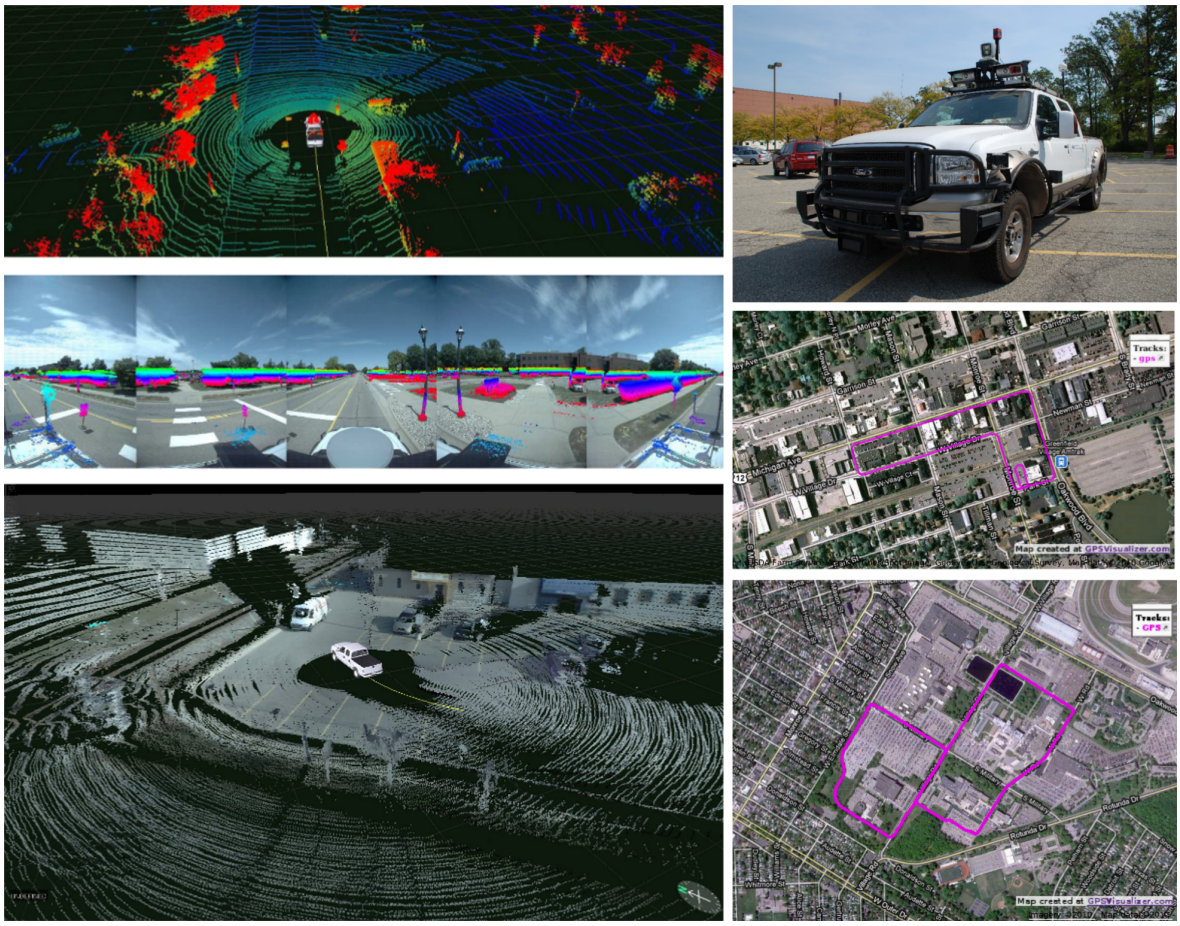


FIGURE 1.8 – Aperçu du jeu de données *Ford Campus*. À gauche : un tour de scan LiDAR coloré par hauteur, un tour de scan projeté sur les images 360° et plusieurs scans colorés grâce aux images 360°. À droite : le véhicule d'acquisition et les trajectoires des deux acquisitions. (image issue de l'article [PANDEY et collab., 2011])

recalé dans le repère caméra. Pour le flux optique, le nuage recalé est projeté sur l'image suivante. On peut avoir un aperçu du jeu de données en figure 1.9 page suivante.

À l'origine, un benchmark est réalisé sur plusieurs tâches :

— la reconstruction de carte de profondeur à partir d'images stéréos,

— la calcul de flux optique,

— l'odométrie/SLAM,

— la détection d'objets en 2D,

— le pistage d'objets,

puis a été étendu à de nombreuses autres tâches au fur et à mesure :

— la détection de route et de marquages au sol en 2013,

— le calcul de flux optique de scène dynamiques en 2015,

— le remplissage (*inpainting* en anglais) d'images de profondeur à partir d'images et de tours de scans de LiDAR,

— la reconstruction de carte de profondeur à partir d'images (non-stéréos) en 2017,

— la détection d'objet en 3D et en vue de dessus de nuages de points (dite *Bird's Eye View*) en 2017,

— la segmentation sémantique d'images en 2017.

Il existe même maintenant un jeu de données synthétique Virtual KITTI [7] [GAIDON et collab., 2016] qui clone plusieurs vidéos de KITTI et génère automatiquement l'annotation des images et les images de flux optique et de profondeur.

On regrettera que les données brut du LiDAR ne soient pas accessibles pour le benchmark d'odométrie/SLAM, seuls les tours complets localisés avec une estimation basée caméra de l'ego-motion du véhicule sont fournis.

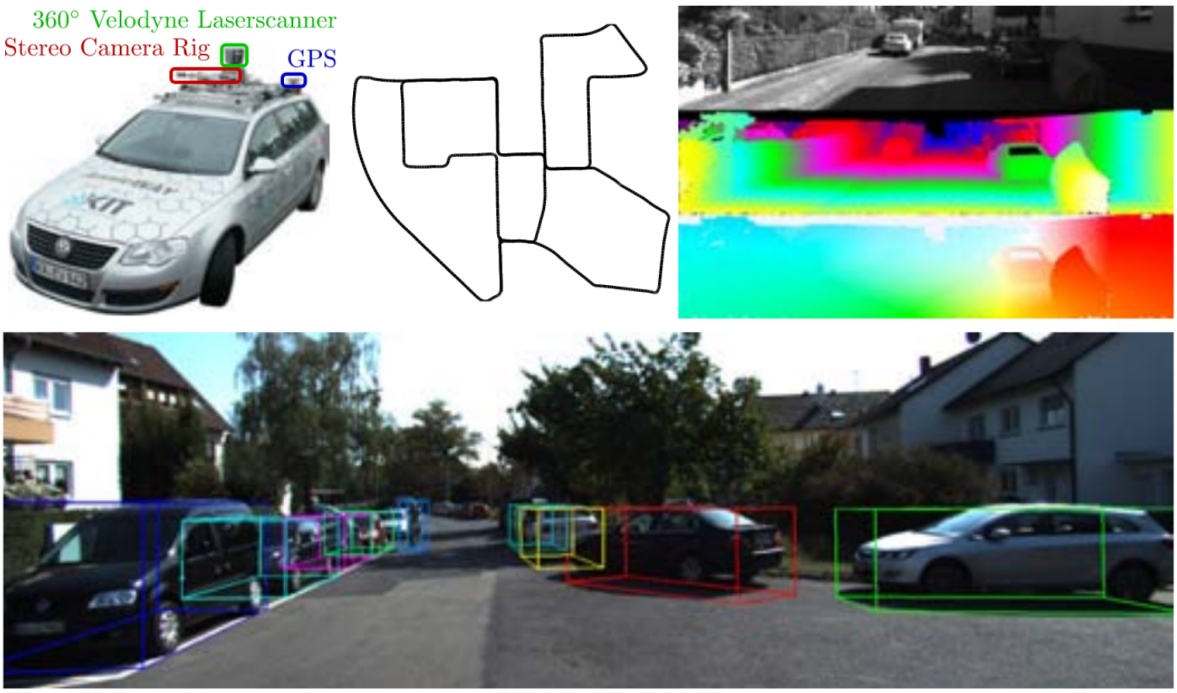


FIGURE 1.9 – En haut de gauche à droite : Plateforme d'acquisition du jeu de données *KITTI*, un exemple de trajectoire, image tirée du jeu de données, la carte de profondeur et le flux optique. En bas : boites englobantes 3D des voitures superposées à une image du jeu de données. (image issue de l'article [GEIGER, LENZ et URTASUN, 2012])

1.2.2.3 TorontoCity [S. WANG et collab., 2017]

Ce jeu de données a été décrit dans l'article [S. WANG et collab., 2017], mais les données n'ont pas encore été publiées. Les informations présentes ici viennent donc uniquement de l'article et des différents sites personnels des auteurs.

Il a été acquis grâce à plusieurs plateformes d'acquisition :

— un avion équipé d'une caméra et d'un LiDAR,

— un véhicule MLS équipé de caméras et LiDAR,

— un drone équipé de caméras.

Peu d'informations sont fournies sur le véhicule terrestre, à part que c'est un véhicule d'acquisition qui est équipé comme pourrait l'être un véhicule autonome. C'est le seul jeu de données qui propose des données acquises par différents vecteurs. Ceci a l'intérêt de permettre d'évaluer des méthodes sur des données très hétérogènes (images et nuages de points) acquis à différents points de vues autour de la même scène. Pour un aperçu du jeu de données voir figure 1.10 page ci-contre.

Il couvre 7112.5km^2, 8439km de route et environ 400000 bâtiments, ce qui en fait sûrement le plus important actuellement publié, mais ces données ne sont pas raisonnablement annotables à la main. Et même avec un protocole d'annotation semi-automatique, la vérification et l'affinement de l'annotation serait très long.

7. http://www.europe.naverlabs.com/Research/Computer-Vision/Proxy-Virtual-Worlds

Les auteurs ont donc choisi une autre approche, l'annotation a été réalisée en alignant des cartes (position des routes et des bâtiments, bâtiments en 3D, équivalent PCRS, cartes des différents réseaux...) avec les données acquises, et en alignant les données acquises entre elles. Il existe donc une segmentation sémantique de la scène sur seulement 3 classes : bâtiments, sol et routes.

Ainsi un benchmark est réalisé dans l'article (peut-être plus tard un benchmark ouvert aux chercheurs du domaine) uniquement sur des tâches qui tournent peu autour de la segmentation sémantique :

— estimation de hauteur des bâtiments (reconstruction),

— extraction d'axe routier,

— segmentation d'instance de bâtiments,

— extraction de contour de bâtiments,

— segmentation sémantique et classification de type de scène.

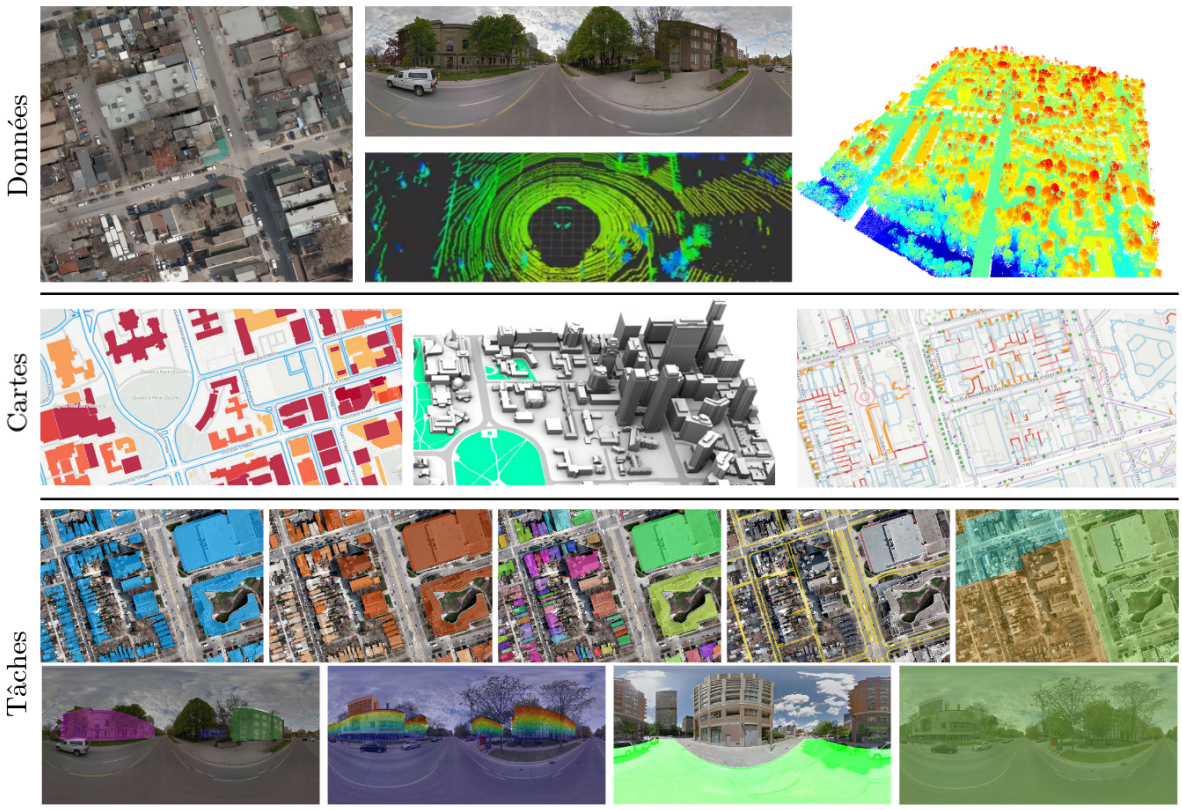


FIGURE 1.10 – Aperçu du jeu de données *TorontoCity*. En haut de gauche à droite : image aérienne, image panoramique et nuage de point terrestre, nuage de point aérien. Au milieu : les cartes. En bas : les tâches (image issue de l'article [S. WANG et collab., 2017]).

1.2.2.4 Robotcar [MADDERN et collab., 2017]

Ce jeu de données est le résultat de plus d'un an d'acquisitions, à raison de deux sessions par semaine, plus de 1000km ont été acquis sur la même route dans toutes les conditions climatiques qui se sont présentées pendant cette durée :

— ensoleillé, pouvant causer l'éblouissement de certains capteurs,

— nuageux, pouvant créer de fortes variations de luminosité,

— (très) pluvieux, peut changer les données captées par le LiDAR,

— enneigé, change complètement l'aspect de l'environnement,

— nocturne, en plus d'une luminosité différente, il y a de nombreuses sources de lumières artificielles.

De plus, les différents travaux réalisés sur la route et les bâtiments dans la zone d'expérimentation ajoutent encore un facteur de variabilité sur l'apparence et la géométrie de la scène. Cette variabilité sur une même route au cours d'une année permet d'évaluer la localisation à long terme, plus précisément la localisation sur une carte acquise depuis assez longtemps pour qu'il y ait pu avoir des modifications importantes dans la scène. Un benchmark est prévu sur la tâche de localisation à long-terme, mais les données n'ont pas encore été publiées.

La plateforme d'acquisition est un véhicule équipé de :

— une caméra stéréo multi-baseline à trois caméras : Bumblebee XB3,

— trois caméras à lentilles grand angle (*fisheye*), pour obtenir une couverture à 360° de la scène,

— deux LiDARs mono-fibre SICK LMS-151, un à l'avant et un à l'arrière, pour balayer la scène orthogonalement à la trajectoire du véhicule,

— un LiDAR 3D (SICK LD-MRS 4 fibres) dirigé vers l'avant,

— une centrale inertielle et un GPS.

On peut regretter qu'un véhicule si récent ne soit pas équipé d'un LiDAR 3D plus résolu et avec un plus grand champ de vision, en effet la plupart des constructeurs automobiles prévoient d'équiper leurs véhicules de plusieurs (jusqu'à 6) LiDARs équivalents à des Velodyne VLP-16 déjà bien plus résolus que le LiDAR utilisé ici. Pour un aperçu des données acquises voir figure 1.11.

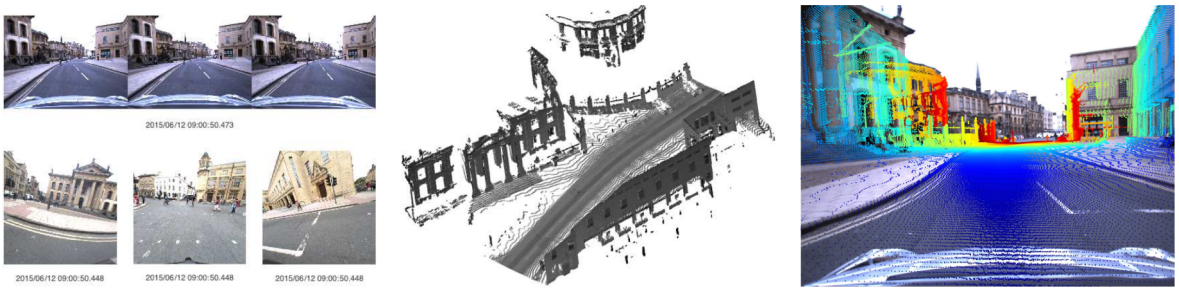


FIGURE 1.11 – Aperçu du jeu de données *Robotcar*. De gauche à droite : images stéréo multi-baseline, nuage de point reconstruit, nuage de point projeté sur une image frontale (image issue de l'article [MADDERN et collab., 2017]).

1.2.2.5 Appoloscape [X. HUANG et collab., 2018]

Ce jeu de données a été acquis dans quatre régions différentes de Chine dans de multiples conditions climatiques (similaires au jeu de données *Robotcar*). Les scènes acquises de complexités différentes, classées en facile, modéré et difficile en fonction du nombre d'objets mobile, d'une dizaine à une centaine. Le véhicule d'acquisition est équipé d'un système MLS RIEGL VMX-1HA intégrant :

— deux LiDARs mono-fibres RIEGL VUX-1HA,

— deux caméras frontales,

— une centrale inertielle et un GPS

Pour un aperçu des données acquises voire figure 1.12 page ci-contre.

Un benchmark est réalisé sur plusieurs tâches :

— détection de marquages au sol,

— détection de voitures,

— segmentation sémantique.

L'annotation a été réalisée sur le nuage de point géo-référencé produit par le système MLS dont les objets potentiellement mobiles ont été supprimés, pour ne garder que la scène statique. Cette annotation a ensuite été projetée sur les images RGB, et l'annotation des objets mobiles a été faite a posteriori. De même les images de profondeur de la scène statique sont produite en projetant le nuage de points en 2D. L'annotation des vidéos est réalisée sur toutes les images, contrairement à tous les autres jeu de données existant, où seules des images prises à certains intervalles de temps sont annotées.

Ce jeu de données semble être à ce jour, le plus important, et le plus prometteur des jeux de données annotés avec information 3D en extérieur. En effet c'est déjà le jeu de données annotées le plus important existant, et il est prévu qu'il soit régulièrement augmenté avec des nouvelles données annotées et de nouvelles tâches à évaluer. Il aurait cependant été intéressant d'avoir des données d'un LiDAR multi-fibres à 360° (dont les certains véhicules autonomes seront probablement équipés).

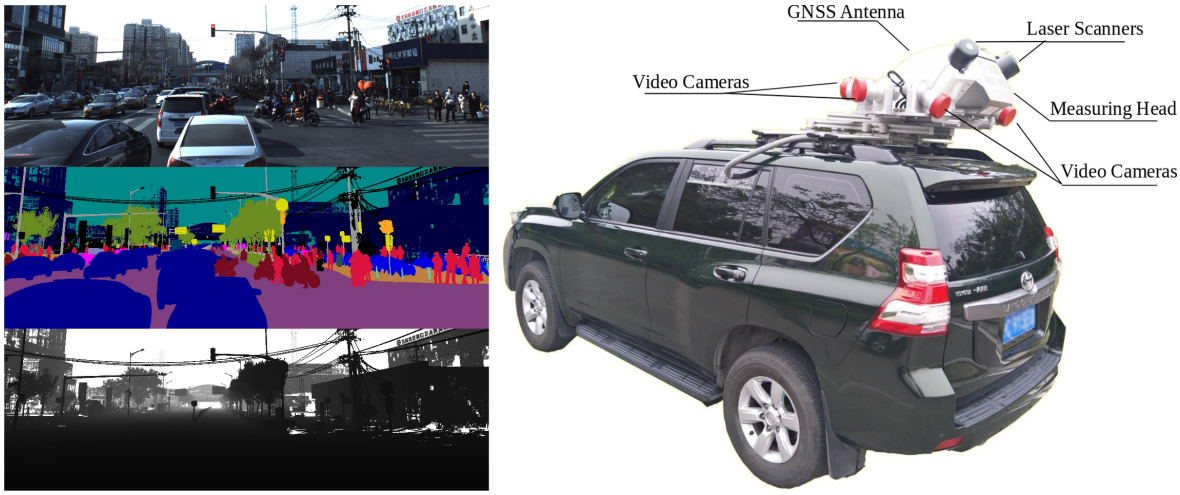


FIGURE 1.12 – Aperçu du jeu de données *Apolloscape*. À gauche : une image RGB, l'annotation correspondante, et l'image de profondeur de l'arrière plan de la scène (sans les objets mobiles). À droite : le véhicule d'acquisition (images issue de l'article [X. HUANG et collab., 2018]).

On ne s'intéresse pas uniquement aux voitures autonomes, mais à tout genre de véhicules autonomes, y compris des robots pouvant se déplacer en intérieur comme en extérieur. On décrit donc également dans la section suivante un jeu de données acquis dans ces deux types d'environnements : NCLT.

1.2.2.6 NCLT Dataset [CARLEVARIS-BIANCO et collab., 2016]

Ce jeu de données a été acquis sur le campus de l'université du Michigan, sur une durée de 15 mois à raison d'une session toutes les deux semaines. Cela a permit de réaliser des acquisitions dans de multiples conditions climatiques (similaires au jeu de données *Robotcar*). La plateforme d'acquisition est un gyropode Segway équipé de :

— une caméra 360° Ladybug3,

— un LiDAR 32 fibres 360° Velodyne HDL-32E,

— 2 LiDARs mono-fibre,

— une centrale inertielle et un GPS-RTK.

Les images et scans LiDAR ne sont pas annotés pour la segmentation sémantique, mais une vérité terrain des trajectoires prises par le véhicule est fournie. La vérité terrain a été obtenue grâce à un algorithme de SLAM LiDAR (par recalage de scans) appliqué à toutes les acquisitions en même

temps pour obtenir toutes les trajectoires dans le même référentiel. De plus, l'erreur de mesure du GPS-RTK par rapport à la vérité terrain est fournie.

Les trajectoires passent à la fois en intérieur et en extérieur, dans des environnements dynamiques à plusieurs échelles temporelles, du piéton ou cycliste qui passe devant les capteurs pendant quelques secondes au bâtiment qui se construit sur plusieurs semaines. Il est donc possible d'évaluer des méthode de localisation à long terme. Un aperçu du jeu de données se trouve en figure 1.13.

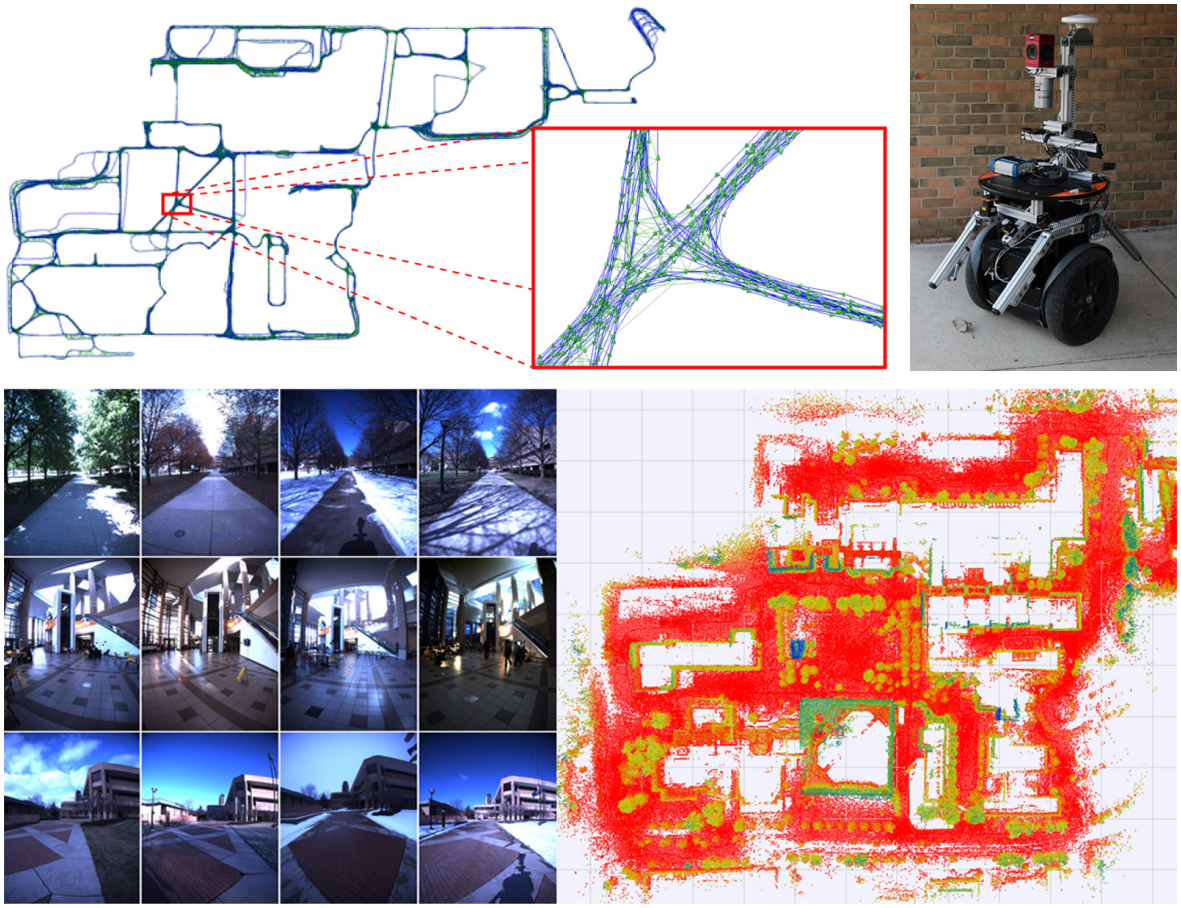


FIGURE 1.13 – Aperçu du jeu de données *NCLT*. En haut à gauche : les trajectoires des différentes acquisitions recalées dans le même référenciel. En haut à droite : la plateforme d'acquisition. En bas à gauche : images de trois scènes prises dans différentes conditions. En bas à droite : une vue du dessus du nuage de point obtenu après une acquisition complète (images issues de l'article [CARLEVARIS-BIANCO et collab., 2016]).

1.3 Un Nouveau Jeu de Données de Nuages de Points 3D Segmentés et Classifiés : Paris-Lille-3D

Les jeux de données de nuages de points annotés présentés précédemment ont tous plusieurs défauts qui ne permettent pas à nos yeux d'entraîner/évaluer des méthodes de segmentation sémantique sur des données réelles.

Cela nous a amenés à la conclusion qu'il était nécessaire de concevoir un jeu de données palliant à la plupart de ces défauts et répondant à certains critères indispensables à un jeu de données permettant d'apprendre la tâche de segmentation sémantique, par exemple pour la conception de HD-Maps de façon automatique.

Le jeu de données que nous allons présenter dans cette section a été publié dans le journal IJRR [ROYNARD, J. DESCHAUD et collab., 2018b] et fait l'objet d'un benchmark publique sur le site http://www.npm3d.fr/paris-lille-3d. Il présente les critères suivants :

— large : équivalent à *Semantic3D* en surface couverte (mais pourrait être plus grand),

— avec de la variabilité : sur deux villes,

— complètement annoté à la main, avec un accent mis sur la qualité de la segmentation entre les objets,

— un nombre de classes élevé (décrit presque tout ce qu'on peut voir dans la rue, bien qu'encore incomplet),

— des données raisonnablement similaires à ce qu'un véhicule d'acquisition pour la production de HD-Maps pourrait acquérir,

— présentant un benchmark publique permettant aux différentes équipes de recherche dans le domaine de comparer leurs résultats sur des critères objectifs.

Cependant ce jeu de données pourrait encore être amélioré, principalement en ajoutant une information de couleur à chaque point.

Le protocole d'acquisition est décrit en section 1.3.1, les nuages obtenus sont décrits en section 1.3.2 page 25, les données annotées sont décrites en section 1.3.3 page 26, un résumé des fichiers est donné en section 1.3.4 page 27, et enfin en section 1.3.5 page 28, les nuages additionnels utilisés pour faire un benchmark publique sont décrits.

1.3.1 Acquisition

Le système d'acquisition choisi est la plateforme MLS expérimentale du Centre de Robotique de l'école des Mines : L3D2 [GOULETTE et collab., 2006] (cf figure 1.14 page suivante). C'est une camionnette Citroën Jumper équipée de :

— un système de navigation et localisation, constitué d'un GPS (Novatel FlexPak6 [8]) et d'une centrale inertielle (Ixsea PHINS [9] en mode LANDINS),

— un capteur extéroceptif : un LiDAR Velodyne HDL-32E [10] monté sur le toit à l'arrière du véhicule avec un angle de 30° entre son axe de rotation et l'horizontale.

Cette plateforme présente deux intérêts majeurs :

— c'est une plateforme MLS, elle permet donc facilement de générer de grandes quantités de données (par exemple d'acquérir une ville moyenne de banlieue parisienne en moins d'une journée), de plus le fait de balayer la scène avec un véhicule mobile roulant à vitesse constante permet d'obtenir une densité quasiment homogène sur tout le jeu de données,

— c'est une plateforme équipée d'un LiDAR multi-fibres, ce qui permet de réduire considérablement les occlusions.

8. https://www.novatel.com/products/gnss-receivers/enclosures/flexpak6/
9. https://www.ixblue.com/products/phins
10. https://velodynelidar.com/hdl-32e.html

FIGURE 1.14 – La plateforme MLS expérimentale de Centre de Robotique : L3D2.

Pour la localisation du véhicule, nous utilisons un GPS RTK à double fréquence L1/L2 cadencé à 1Hz. Une base fixe du réseau RGP [11] (pour *Réseau GNSS Permanent*) de l'IGN est utilisée pour la correction RTK. Les bases RGP utilisées sont : SMNE pour l'acquisition à Paris et LMCU pour les deux acquisitions à Lille. La centrale inertielle est elle cadencée à 100Hz et les données GPS et inertielles et LiDAR sont synchronisées grâce au signal PPS du GPS.

En post-traitement, nous récupérons les données de la base fixe RGP, et générons les trajectoires grâce au logiciel Inertial Explorer [12] prenant en entrée uniquement les données GPS et inertielles. La méthode utilisée est la méthode Tightly Coupled GPS-RTK/INS Kalman Smoothing EKF. Nous obtenons une trajectoire dans le système WGS84 à 100Hz, que nous convertissons en Lambert RGF93. En l'état, aucune méthode de SLAM, de recalage ou d'affinage de nuage ou de fermeture de boucle n'est appliquée pour améliorer la trajectoire ou la qualité du nuage.

Enfin, comme chaque point a son propre horodatage, nous interpolons linéairement la trajectoire pour donner une position adéquate à chaque point. De plus, nous ne gardons que les points mesurés à une distance inférieure à 20m du capteur afin de ne garder que les zones de densité suffisamment élevée. Enfin, nous construisons des nuages pour lesquels chaque point est caractérisé par un vecteur $(x,y,z,x_{origin},y_{origin},z_{origin},t,i)$, où $_{origin}$ indique la position du LiDAR au moment de l'acquisition du point, t est le moment de l'acquisition du point et i est l'intensité du retour laser.

Cette plateforme et ce protocole d'acquisition sont constamment en cours d'amélioration, il y a trois perspectives d'évolutions majeures :

— ajouter une étape de SLAM ou de recalage non-rigide avec fermeture de boucle pour améliorer la qualité des nuages générés, en particuliers là où les signaux GPS sont de mauvaise qualité.

— ajouter d'autres capteurs extéroceptifs, comme une caméra 360° aussi appelée caméra sphérique (Ladybug5 [13] déjà acheté et en cours d'intégration) ou un Velodyne VLP-16 [14] monté

11. http://rgp.ign.fr/
12. https://www.novatel.com/products/software/inertial-explorer/
13. https://www.ptgrey.com/ladybug5-360-degree-usb3-spherical-camera-systems
14. https://velodynelidar.com/vlp-16.html

avec axe de rotation vertical pour faire du SLAM LiDAR, et améliorer encore la localisation du véhicule.

— mettre tous les capteurs et la centrale inertielle sur une plateforme isolée du véhicule pour supprimer les vibrations parasites dues aux imperfections de la route.

1.3.2 Description des nuages de points

Le jeu de données est constitué de trois nuages de points géo-référencés : 2 sections à Lille et une à Paris. La figure 1.15 montre les trajectoires des 3 acquisitions.

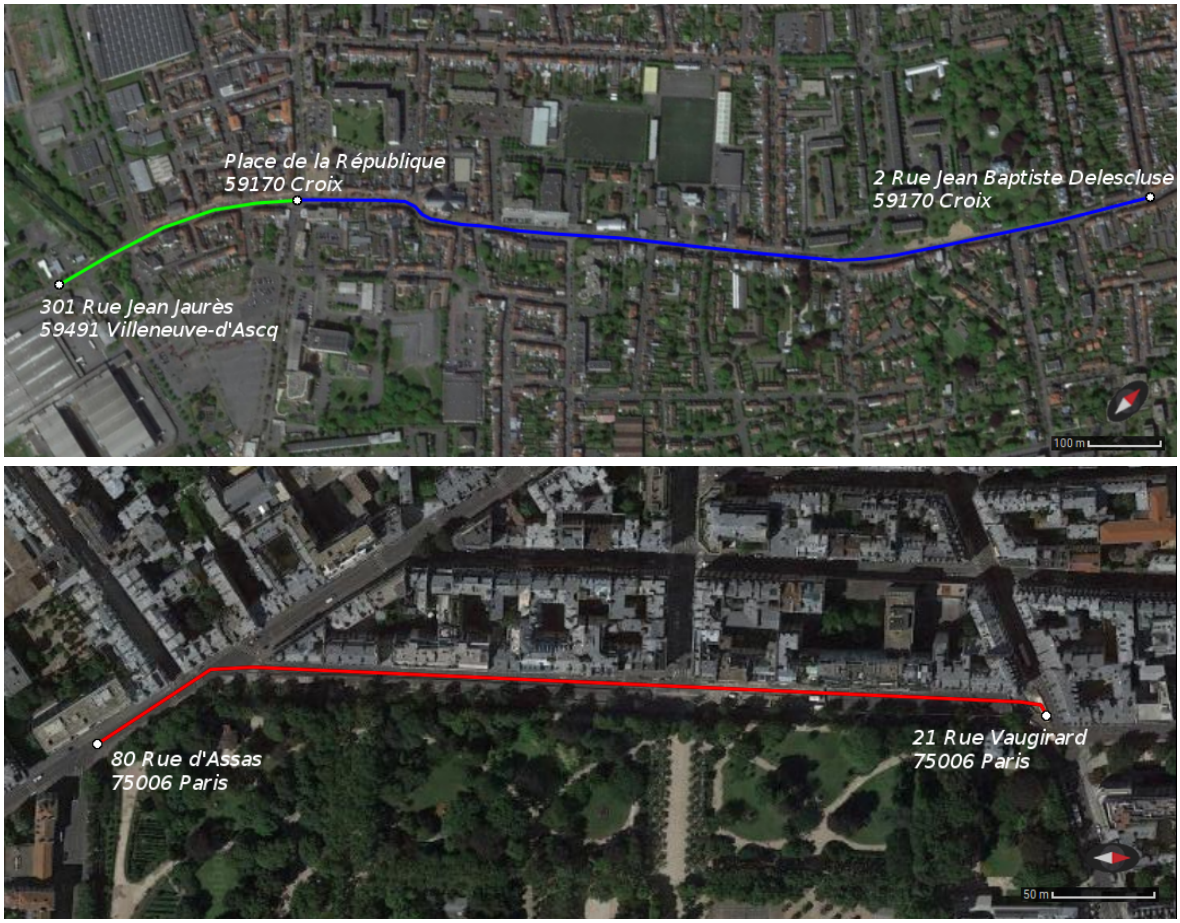


FIGURE 1.15 – Les trajectoires du vehicule pendant les acquisitions. En haut : les 2 trajectoires dans l'agglomeration de Lille sont en bleu (Lille1) et vert (Lille2). En bas : la trajectoire à Paris est en rouge. (Images issues de Google Maps)

Pour pouvoir enregistrer les coordonnées de chaque point en flottant (32 bits) et conserver une précision suffisante, un décalage est soustrait au coordonnées de chaque point dans le plan (x,y). Un résumé des trois sections de nuages est donné en tableau 1.3.

Section	Surface couverte	Longueur	Nombre de points	Décalage RGF93
Lille1	24746m^2	1150m	71.3M	(711164.0m,7064875.0m)
Lille2	7166m^2	340m	26.8M	(711164.0m,7064875.0m)
Paris	10802m^2	450m	45.7M	(650976.0m,6861466.0m)
Total	42715m^2	1940m	143.1M	–

TABLEAU 1.3 – Description des trois sections du jeu de données.

Les nuages ont une densité élevé avec entre 1000 et 2000 points par mètre carré sur le sol,

mais la configuration du système d'acquisition génère des motifs anisotropiques sur le sol (cf figure 1.16).

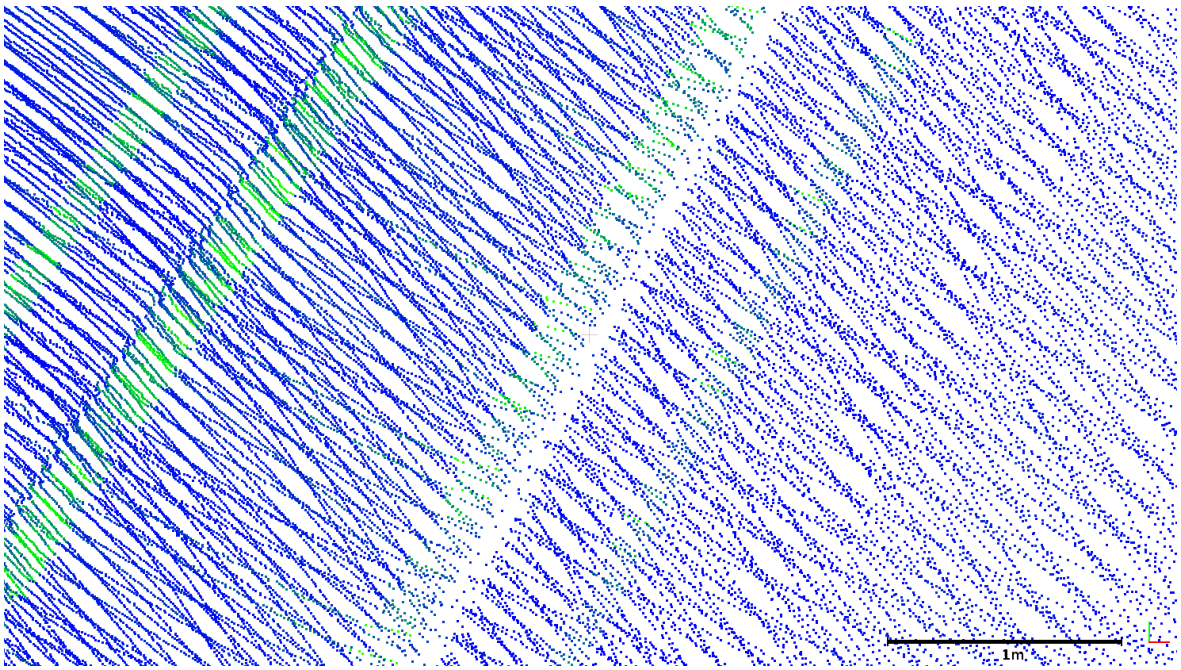


FIGURE 1.16 – Motifs anisotropiques sur le sol (la couleur des points est la réflectance du retour laser).

1.3.3 Description des données annotées

Les nuages obtenus ont été classifiés à la main en utilisant le logiciel CloudCompare. L'annotation des 2km d'acquisition a pris environ un mois et demi à temps plein à un agent expert (l'auteur de cette thèse). L'accent a été mis sur la qualité de l'annotation, en particulier à la frontière entre deux objets. De plus un effort notable a été mis à attribuer une classe à chaque objet, quitte à créer une nouvelle classe si un objet ne rentre dans aucune existante. La figure 1.17 page suivante montre des nuages annotés du jeu de données Paris-Lille-3D.

Nous avons choisi de réutiliser l'arbre de classe [15] du challenge *iQmulus &Terramobilita* qui est déjà très complet. Nous avons seulement changé quelques classes et ajouté les classes manquantes suivantes :

— *borne à vélos* (`id` = 302021200)

— *statue* (`id` = 302021300)

— *boite aux lettres* (`id` = 302040600)

— *console d'éclairage* (`id` = 302040700)

— *moulin* (`id` = 302040800)

Pour une distribution du nombre de points par classe voir le tableau 1.4 page 29.

Nous avons également changé la façon dont les classes de véhicules sont organisées. Pour chaque classe de véhicule (voiture, camion, vélo...), on distingue des sous-classes en fonction de l'*état* du véhicule : garé, à l'arrêt (sur la route) ou en mouvement. Et enfin la classe *terminal Vélib* est remplacée par *terminal à vélo* (`id` = 302021100) qui est plus générique, et ne dépend pas de la ville d'acquisition.

À part les quelques classes mentionnées au-dessus, cet arbre de classe est suffisamment complet pour classer les objets rencontrés dans ce jeu de données. Le fichier décrivant cet arbre de classe est nommée `classes.xml` et est fourni avec le jeu de données. Nous fournissons également

15. http://data.ign.fr/benchmarks/UrbanAnalysis/download/classes.xml

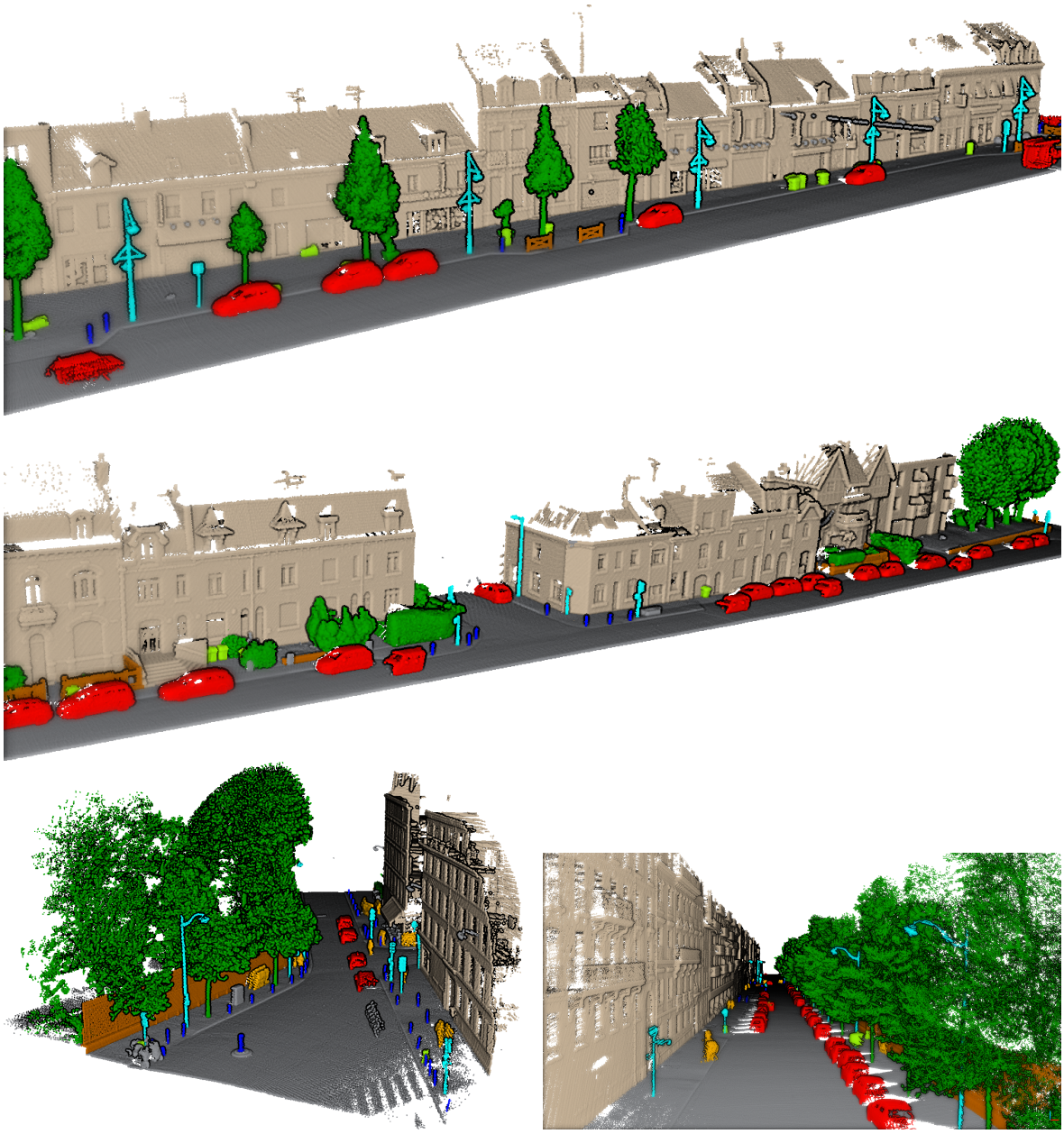


FIGURE 1.17 – Exemples de nuages annotés du jeu de données Paris-Lille-3D.

des fichiers d'annotations pour les instances d'objets qui méritent des informations supplémentaires. Par exemple quand plusieurs arbres sont tellement entrelacés qu'ils ne peuvent pas être séparés à la main, on garde ces arbres en une seule instance, mais une annotation est fournie pour cette instance. Les fichiers d'annotations sont décrits en section 1.3.4.

1.3.4 Description des fichiers

Chaque section du jeu de données est enregistrée dans un fichier au format `.ply` distinct, un résumé de chaque fichier est donné en tableau 1.5 page 29. Chaque point des fichiers `.ply` a 10 attributs :

— `x`, `y`, `z` (float, 32 bits) : la position du point,

— `x_origin`, `y_origin`, `z_origin` (float, 32 bits) : la position du LiDAR au moment de l'acquisition du point,

— `GPS_time` (double, 64 bits) : le moment d'acquisition du point,

— `reflectance` (uint8) : l'intensité du retour laser,

— `label` (uint32) : le label de l'objet auquel le point appartient,

— `class` (uint32) : la classe de l'objet auquel le point appartient.

On fournit également trois fichiers texte contenant des annotation pour les instances d'objets particulières ou sortant de l'ordinaire. Chaque ligne de ces fichiers contient

```
<indentifiant d'instance>,
      <indentifiant de classe>,
            <nom de classe>,
                  <annotation1>,
                        <annotation2>,
                              ...
```

Les annotations les plus courantes sont :

— *plusieurs*, par exemple quand des arbres sont entrelaçés les uns dans les autres et ne peuvent pas être délimités précisément à l'œil nu,

— *renversé*, pour les poubelles couchées sur le sol.

1.3.5 Challenge de segmentation sémantique

Nous avons également annoté trois autres scènes urbaines, deux à Ajaccio et une à Dijon dont l'annotation n'est pas rendue publique. Le but est de créer un challenge de segmentation sémantique similaire au Challenge Semantic3D. Les données sont rendues accessibles à la communauté sur le site npm3d.fr [16]. La figure 1.18 page 30 montre des nuages jeu de test du challenge Paris-Lille-3D.

Pour pouvoir évaluer les méthodes sur un nombre suffisant de points, un sous ensemble de classes plus grossières que les classes du jeu d'entraînement a été choisi pour le jeu de test. Les 9 classes grossières choisies sont les suivantes :

sol	bâtiment	poteaux/signalisation/lampadaires
potelets	poubelles	barrières
piétons	voitures	naturel/végétation

Un fichier donnant les correspondances entres les classes d'origine et les classes grossières est fourni avec le jeu de données. Tous les points ne tombant dans aucune de ces classes sont marqués comme *non-classifiés* mais sont conservés dans les nuages.

Les trois scènes ont été choisies pour être suffisamment représentatives du jeu d'entraînement (zone urbaines avec bâtiments et végétation), et ayant suffisamment de points dans les classes grossières, sans pour autant être acquises dans les mêmes villes, donc les mobilier urbains et les bâtiments ont des formes légèrement différentes. Un résumé des sections du jeu de test se trouve dans le tableau 1.6 page 30.

Comme en plus d'un identifiant de classe, un identifiant d'instance d'objet est donné à chaque point, on peut envisager dans le futur d'ouvrir un benchmark publique de détection d'objets dans les classes grossières, voire un benchmark de segmentation panoptique [KIRILLOV et collab., 2018], qui semble être le but suivant à atteindre dans le domaine de l'analyse de scènes.

16. http://npm3d.fr/paris-lille-3d

	Class	Number of samples (of points)							
		Lille1		Lille2		Paris		Total	
	unclassified	1	(54.88k)	1	(16.47k)	1	(60.79k)	3	(132.1k)
	other	16	(70.15k)	11	(14.10k)	11	(30.82k)	38	(115.1k)
surface	road	1	(34.80M)	1	(11.82M)	1	(19.68M)	3	(66.30M)
surface	sidewalk	18	(8.566M)	8	(4.97M)	5	(4.6M)	31	(16.67M)
surface	island	7	(458.8k)	0	(0)	9	(82.46k)	16	(541.2k)
surface	vegetation	32	(562.2k)	22	(512.4k)	6	(157.1k)	60	(1.232M)
surface	building	34	(18.2M)	8	(7.934M)	5	(9.431M)	47	(35.39M)
object	post	9	(13.51k)	0	(0)	0	(0)	9	(13.51k)
object	bollard	122	(34.80k)	64	(7.982k)	84	(28.33k)	270	(71.10k)
object	floor lamp	72	(232.9k)	13	(23.69k)	21	(113.2k)	106	(369.7k)
object	traffic light	14	(25.82k)	11	(27.41k)	16	(80.76k)	41	(133.10k)
object	traffic sign	82	(113.6k)	39	(69.89k)	17	(30.75k)	138	(214.3k)
object	signboard	20	(68.34k)	12	(43.14k)	3	(13.41k)	35	(124.9k)
object	mailbox	0	(0)	0	(0)	1	(4.739k)	1	(4.739k)
object	trash can	11	(14.72k)	6	(17.43k)	22	(30.35k)	39	(62.50k)
object	meter	0	(0)	0	(0)	2	(2.840k)	2	(2.840k)
object	bicycle terminal	10	(1.736k)	0	(0)	13	(6.451k)	23	(8.187k)
object	bicycle rack	8	(774)	0	(0)	14	(8.665k)	22	(9.439k)
object	statue	2	(4.158k)	0	(0)	0	(0)	2	(4.158k)
object	barrier	45	(83.94k)	5	(9.286k)	7	(56.10k)	57	(149.3k)
object	roasting	6	(831.0k)	1	(20.82k)	4	(1.677M)	11	(2.529M)
object	wire	34	(14.18k)	8	(9.325k)	0	(0)	42	(23.51k)
object	low wall	58	(840.2k)	2	(26.99k)	9	(951.4k)	69	(1.819M)
object	shelter	0	(0)	0	(0)	3	(83.45k)	3	(83.45k)
object	bench	5	(2.911k)	1	(364)	2	(1.391k)	8	(4.666k)
object	distribution box	19	(50.28k)	8	(14.89k)	3	(53.4k)	30	(118.2k)
object	lighting console	78	(7.251k)	60	(9.731k)	9	(4.393k)	147	(21.38k)
object	windmill	1	(10.17k)	0	(0)	0	(0)	1	(10.17k)
object	pedestrian	17	(24.81k)	7	(11.60k)	61	(150.2k)	85	(186.6k)
object	parked bicycle	15	(9.74k)	0	(0)	33	(81.67k)	48	(90.75k)
object	mobile scooter	0	(0)	0	(0)	1	(131)	1	(131)
object	parked scooter	0	(0)	0	(0)	31	(169.1k)	31	(169.1k)
object	mobile motorbike	0	(0)	0	(0)	1	(1.613k)	1	(1.613k)
object	parked motorbike	2	(2.428k)	0	(0)	4	(14.37k)	6	(16.79k)
object	mobile car	21	(175.0k)	4	(40.96k)	5	(66.35k)	30	(282.3k)
object	stopped car	0	(0)	1	(28.27k)	1	(9.375k)	2	(37.64k)
object	parked car	182	(2.266M)	47	(853.7k)	65	(1.610M)	294	(4.730M)
object	mobile van	3	(97.27k)	1	(41.6k)	0	(0)	4	(138.3k)
object	parked van	5	(84.75k)	5	(85.20k)	9	(357.6k)	19	(527.6k)
object	stopped truck	0	(0)	0	(0)	1	(235.7k)	1	(235.7k)
object	parked truck	2	(40.32k)	0	(0)	1	(53.44k)	3	(93.76k)
object	stopped bus	0	(0)	0	(0)	1	(78.41k)	1	(78.41k)
object	parked bus	1	(9.623k)	0	(0)	0	(0)	1	(9.623k)
object	table	0	(0)	0	(0)	2	(576)	2	(576)
object	chair	0	(0)	0	(0)	8	(4.842k)	8	(4.842k)
object	trash can	138	(148.8k)	80	(115.9k)	0	(0)	218	(264.7k)
object	waste	5	(2.307k)	0	(0)	0	(0)	5	(2.307k)
natural	natural	149	(1.233M)	47	(396.9k)	36	(1.610M)	232	(3.240M)
natural	tree	72	(2.310M)	23	(565.10k)	101	(4.755M)	196	(7.631M)
natural	potted plant	32	(72.80k)	5	(26.96k)	0	(0)	37	(99.76k)
	Total	1349	(71.36M)	501	(26.84M)	629	(45.80M)	2479	(143.10M)

TABLEAU 1.4 – Nombre d'échantillons/points par classe (k pour milliers et M pour millions). La classe *trash cans* apparait deux fois, la première fois correspond aux poubelles fixes et la seconde au poubelles individuelles.

Section	Length	Number of points	Number of objects	Number of classes
Lille1	1150m	71.3M	1349	39
Lille2	340m	26.8M	501	29
Paris	450m	45.7M	629	41
Total	1940m	143.1M	2479	50

TABLEAU 1.5 – Chiffres caractéristiques du jeu de données Paris-Lille-3D.

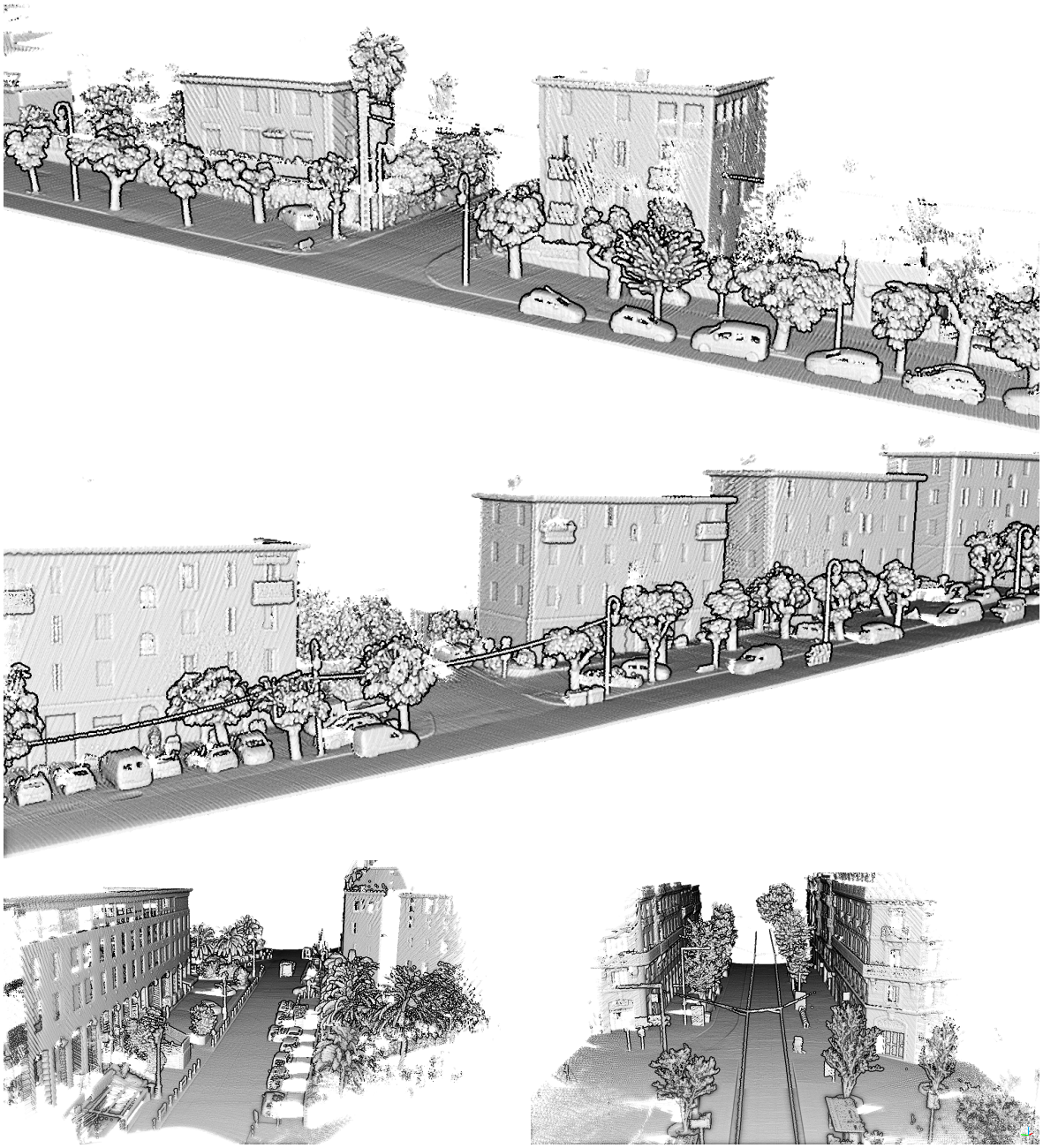


FIGURE 1.18 – Exemples de nuages du jeu de test du challenge Paris-Lille-3D.

Section	Surface couverte	Longueur	Nombre de points	Décalage RGF93
Ajaccio2	2917m^2	120m	10M	(1176570.0m,6108910.0m)
Ajaccio57	2281m^2	100m	10M	(1176570.0m,6108910.0m)
Dijon9	2531m^2	100m	10M	(853310.0m,6693380.0m)
Total	7729m^2	320m	30M	–

TABLEAU 1.6 – Description des trois sections de test du jeu de données.

1.4 Cahier des Charges d'un Jeu de Données pour le Véhicule Autonome

Cette section a pour ambition d'émettre des lignes de conduite pour concevoir ce que serait un jeu de données idéal pour le véhicule autonome.

1.4.1 Contexte

On se met ici du point de vue soit :

— d'un constructeur, qui souhaite concevoir un véhicule autonome dont une ou plusieurs briques algorithmiques nécessitent d'être entraînées,

— d'un législateur (par exemple) ou d'un constructeur, qui veut, avant de laisser circuler de tels véhicules autonomes, s'assurer que les briques logicielles de ces véhicules sont plus performantes / robustes / sûres qu'un humain.

L'un comme l'autre a alors besoin de jeux de données vérifiant certains critères qu'on développera plus loin.

On parle ici en particulier de voiture autonome, puisque c'est un domaine de prédilection du *Centre de Robotique* mais surtout parce que ça donne un bon point d'appui pour développer le raisonnement. Cependant le raisonnement développé ici peut s'appliquer à tout type de robot autonome (drone, bateau, sous-marin, robot humanoïde, de manutention, ...).

Un véhicule autonome est avant tout un robot constitué de capteurs, d'actionneurs, et d'un système informatique (calculateur/ordinateur) traitant toutes les données produites par les capteurs pour prendre une décision quant aux commandes à envoyer aux actionneurs.

Du côté du calculateur embarqué, les algorithmes qui seront finalement utilisés n'étant pas encore connus et les évolutions des technologies d'ordinateurs n'étant pas prévisible, on s'abstiendra de faire toute forme de spéculation dans ce domaine.

Du côté des actionneurs, dans le cadre imaginé actuellement (une voiture classique équipée d'un moyen d'être commandé par ordinateur), les technologies sont bien maîtrisées par les constructeurs et les commandes possibles sont connues et peu nombreuses : tourner les roues avec un certain angle, accélérer, freiner, passer une vitesse, faire marche arrière, actionner le frein à main, actionner un des systèmes lumineux (phares, codes, clignotants, anti-brouillard, marche arrière) ou sonore.

Ainsi, on ne développera pas plus sur les actionneurs, par contre on abordera le sujet des capteurs bien plus en détails en section 1.4.4 page suivante.

1.4.2 Tâches à réaliser

Un véhicule autonome doit comprendre son environnement pour prendre des décisions sur sa façon de conduire. Cette tâche peut se diviser en plusieurs sous-tâches à résoudre en simultané sur les données captées :

— localisation (avec fermeture de boucle)

— cartographie

— détection des zones navigables

— détection des autres objets/véhicules mobiles

— détection de la signalisation

— estimation de la qualité de la chaussée

Après avoir rempli tout ou partie de ces tâches, il doit remplir la tâche *conduire*. C'est-à-dire commander ses actionneurs, pour suivre une trajectoire. Cette tâche peut elle aussi se diviser en plusieurs sous-tâches :

— (re)planification de trajectoire(s),

— évitement d'obstacle,

— conversion de trajectoires en commandes pour les actionneurs (contrôle),

— arrêt d'urgence,

Il y également des tâches qui peuvent relever des deux domaines précédents comme la compréhension de son influence sur son environnement et l'anticipation des réactions des autres usagers de l'espace publique.

1.4.3 Justification

Dans la mesure où la résolution d'une tâche peut aider à la résolution d'une autre, l'idéal serait d'avoir des jeux de données qui permettent de résoudre plusieurs tâches en même temps, voire toutes les tâches.

Cette vision est partagée par exemple par Raquel Urtasun (UBER), elle l'a présenté durant un Keynote à IROS 2018. En effet, pour des raisons d'interprétabilité du comportement, on ne veut pas un véhicule qui fonctionnerait de façon *end-to-end*, c'est-à-dire qui prendrait en entrée l'intégralité des données issues des capteurs et qui produirait les commandes pour les actionneurs, avec au milieu une boite-noire qui aurait été apprise de bout en bout sans aucun contrôle sur les potentielles étapes du processus interne.

On imagine donc plutôt des véhicules autonomes constitués de plusieurs briques logicielles réalisant chacune une tâche bien précise : localisation, détection de zone de circulation, de signalisation, d'obstacles, planification de trajectoire, contrôle... Chaque brique pouvant être apprise indépendamment, mais les briques pouvant également communiquer entre elles, l'ensemble du processus peut également être appris de bout-en-bout. Ce qui mène à un mode d'apprentissage de l'ensemble du processus de bout-en-bout en ajoutant des contraintes pour que chaque brique réalise bien la tâche requise.

Pour cela il suffit d'enregistrer l'intégralité des données acquises par les capteurs ainsi que l'intégralité des actions réalisées par le conducteur du véhicule d'acquisition, le tout synchronisé précisément. Ceci est totalement réalisable avec les technologies actuelles. Il reste alors (mais pas des moindres) à produire la vérité terrain pour chacune des tâches. Tout ceci est bien sûr réalisable en simulation, mais aux vue des technologies actuelles de simulations il n'est pas acceptable d'entraîner et de valider les véhicules autonomes uniquement sur des données synthétiques.

1.4.4 Capteurs et types de données

De plus comme le choix des ensembles de capteurs qui seront installés sur les véhicules autonomes ne sont pas encore définis, l'idéal serait d'utiliser un véhicule sur-équipé en capteurs (sans avoir peur d'être redondant), en ajoutant des capteurs beaucoup plus précis que ce qu'un véhicule pourra embarquer qui permettront d'obtenir une vérité terrain sur certaines tâches. Un tel véhicule serait par exemple un véhicule équipé de :

— Pour la perception du véhicule autonome :
  — un radar frontal (éventuellement des radars latéraux et arrière),
  — des capteurs ultrasonique tout autour de la voiture,
  — des caméras frontale en position stéréo multi-baseline,
  — des caméras grand angle tout autour de la voiture,
  — une caméra 360°
  — des caméras infrarouges, thermiques, multi-spectrales, hyper-spectrales, ...
  — des capteurs sonores ? ...
  — un LiDAR 360° ou plusieurs LiDAR(s) . tous étant des LiDARs multi-fibres,
— Pour la localisation du véhicule autonome :
  — un système GNSS grand publique,
  — une/des centrale(s) inertielle(s) grand publique,

  - des encodeurs de roues,
- Pour produire la vérité terrain :
  - un GPS-RTK : pour la vérité terrain de la trajectoire,
  - une centrale inertielle haut de gamme : pour la vérité terrain de la trajectoire,
  - un système d'acquisition MLS haut de gamme (semblable au RIEGL VMX-1HA) : pour la vérité terrain de la scène 3D.

Ainsi on a un véhicule sur-équipé en capteurs ce qui permet à la fois une redondance et une grande robustesse du système (en particulier dans le cas où un capteur tombe en panne), et de déterminer les capteurs les plus pertinents et utiles dans chaque situation, pour les sélectionner et les traiter uniquement quand nécessaire.

1.4.5 Vérité terrain de qualité

Dans la même idée que le jeu de données Appoloscape, le système d'acquisition MLS haut de gamme permet d'obtenir une carte 3D très précise de l'environnement sous forme de nuage de points 3D. Ceci permet à la fois de produire un équivalent de HD-Maps que le véhicule est censé avoir et utiliser quand il navigue, et de faciliter la phase d'annotation des données par exemple pour séparer les objets mobiles et l'arrière-plan. On peut même imaginer fusionner cette carte avec une acquisition aérienne par avion ou drone pour compléter la carte 3D et s'assurer que quelle que soit la position du véhicule et son point de vue sur la scène, il ne voit pas de *trou*.

1.4.6 Données très diversifiées

Pour rendre un véhicule robuste aux changements à différentes échelles au niveau urbain, le jeu de données doit être à long terme. Il doit même continuer à être acquis indéfiniment, pour être représentatif des éventuels nouveaux comportements et usages ainsi qu'aux modifications réglementations, signalisations... On sait qu'un réseau de neurones profond entraîné à traduire plusieurs langues entre elles sera plus performant qu'un réseau entraîné spécifiquement pour la traduction d'un couple de langues. Il parait donc logique d'imaginer qu'un véhicule autonome entraîné sur plusieurs villes et pays soit plus robuste qu'un véhicule entraîné spécifiquement sur un seul pays. Le réseau apprend en fait des caractéristiques de plus haut niveau (plus abstraites) qu'il peut utiliser dans toutes les conditions. De même pour les différentes conditions climatiques ou périodes de la journée. C'est pourquoi le jeu de données doit être acquis continûment au cours de l'année à toute période de la journée dans toutes les villes et pays de la planète. C'est ce qu'on voit se profiler avec des futures véhicules connectés qui remonteront les moindres informations de changement ou d'anomalie à un jeu de données dans le *cloud*, jeu de données qui permettra de ré-entraîner le véhicule.

1.4.7 Données synthétiques/simulées

Tous les efforts réalisés dans l'acquisition et l'annotation de milliers de kilomètres ne seront pas suffisants pour produire les millions de kilomètres de données nécessaires à la validation. C'est pourquoi il est indispensable d'être capable de créer un double numérique du véhicule autonome, en particulier d'être capable de simuler de façon réaliste tous les capteurs.

Un autre point sur lequel la simulation est indispensable est la diversité des situations de conduites rencontrées par le véhicule pendant la phase d'entraînement. En effet, a priori un jeu de données acquis sur route ouverte ne contiendra que très peu de *cas à risques* ou *cas critiques*, alors qu'un simulateur permettra d'en générer beaucoup plus et donc à la fois d'entraîner le véhicule autonome et d'évaluer comment il réagit dans les pires situations.

1.4.8 Données non-annotées

En plus des données synthétiques/simulées, une autre façon d'augmenter le nombre de kilomètres vus pendant l'entraînement est d'ajouter des données non-annotées qui sont bien plus faciles à produire (il suffit de laisser tourner les véhicules d'acquisition et de les faire parcourir des routes qu'ils n'ont pas encore vues). Il existe en effet de plus en plus de recherches dans les méthodes d'apprentissage dites non-supervisées qui trouve des façon d'apprendre des caractéristiques intéressantes (pour regrouper les objets similaires par exemple). Il reste alors toujours une petite étape supervisée (si on veut faire de la classification par exemple) pour apprendre les noms donnés par les humains à tel ou tel classe/objet, mais cette phase peut alors être presque instantanée.

1.5 Conclusion

Dans ce chapitre, nous avons réalisé un état de l'art des jeux de données de nuages de points 3D annotés et des jeux de données utiles pour le véhicule autonome contenant des nuages de points 3D. Après avoir fait le constat qu'aucun jeu de données de la littérature ne convenait pour entraîner des méthodes d'apprentissage profond pour la segmentation sémantique nécessaire à la création de HD-Maps, nous avons ensuite présenté un nouveau jeu de données de nuages de points 3D urbain acquis par MLS, qui correspond aux données pouvant être utilisées pour produire des HD-Maps. Ce jeu de données, appelé Paris-Lille-3D, a été annoté à la main pour assurer une qualité suffisante de l'annotation et fait l'objet d'un challenge pour comparer les méthodes de segmentation sémantique. Enfin, nous avons donné des lignes de conduite pour concevoir ce que serait un jeu de données complet pour entraîner et valider toutes les briques logicielles d'un véhicule autonome.

Références

ABU-EL-HAIJA, Sami, Nisarg KOTHARI, Joonseok LEE, Paul NATSEV, George TODERICI, Balakrishnan VARADARAJAN et Sudheendra VIJAYANARASIMHAN (2016). « Youtube-8m: A large-scale video classification benchmark ». In : *arXiv preprint arXiv:1609.08675* (cf. p. 7).

ARMENI, Iro, Sasha SAX, Amir R ZAMIR et Silvio SAVARESE (2017). « Joint 2d-3d-semantic data for indoor scene understanding ». In : *arXiv preprint arXiv:1702.01105* (cf. p. 14).

ARMENI, Iro, Ozan SENER, Amir R. ZAMIR, Helen JIANG, Ioannis BRILAKIS, Martin FISCHER et Silvio SAVARESE (2016). « 3D Semantic Parsing of Large-Scale Indoor Spaces ». In : *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* (cf. p. 12, 13, 65).

BESL, Paul J et Neil D MCKAY (1992). « Method for registration of 3-D shapes ». In : *Sensor Fusion IV: Control Paradigms and Data Structures*. T. 1611. International Society for Optics et Photonics, p. 586-607 (cf. p. 16).

CARLEVARIS-BIANCO, Nicholas, Arash K USHANI et Ryan M EUSTICE (2016). « University of Michigan North Campus long-term vision and lidar dataset ». In : *The International Journal of Robotics Research* 35.9, p. 1023-1035 (cf. p. 7, 21, 22).

CORDTS, Marius, Mohamed OMRAN, Sebastian RAMOS, Timo REHFELD, Markus ENZWEILER, Rodrigo BENENSON, Uwe FRANKE, Stefan ROTH et Bernt SCHIELE (2016). « The cityscapes dataset for semantic urban scene understanding ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 3213-3223 (cf. p. 7).

DAI, Angela, Angel X. CHANG, Manolis SAVVA, Maciej HALBER, Thomas FUNKHOUSER et Matthias NIESSNER (2017). « ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes ». In : *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* (cf. p. 14).

DENG, Jia, Wei DONG, Richard SOCHER, Li-Jia LI, Kai LI et Li FEI-FEI (2009). « Imagenet: A large-scale hierarchical image database ». In : *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, p. 248-255 (cf. p. 7, 47).

FERRARO, Francis, Nasrin MOSTAFAZADEH, Ting-Hao HUANG, Lucy VANDERWENDE, Jacob DEVLIN, Michel GALLEY et Margaret MITCHELL (2015). « A survey of current datasets for vision and language research ». In : *arXiv preprint arXiv:1506.06833* (cf. p. 7).

GAIDON, Adrien, Qiao WANG, Yohann CABON et Eleonora VIG (2016). « Virtual worlds as proxy for multi-object tracking analysis ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 4340-4349 (cf. p. 18).

GEIGER, Andreas, Philip LENZ, Christoph STILLER et Raquel URTASUN (2013). « Vision meets robotics: The KITTI dataset ». In : *The International Journal of Robotics Research* 32.11, p. 1231-1237 (cf. p. 16).

GEIGER, Andreas, Philip LENZ et Raquel URTASUN (2012). « Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite ». In : *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, p. 3354-3361 (cf. p. 7, 16, 18).

GOULETTE, F, F NASHASHIBI, I ABUHADROUS, S AMMOUN et C LAURGEAU (2006). « An integrated on-board laser range sensing system for on-the-way city and road modelling ». In : *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 34.Part A, p. 78-83 (cf. p. 10, 23).

HACKEL, Timo, Nikolay SAVINOV, Lubor LADICKY, Jan Dirk WEGNER, Konrad SCHINDLER et Marc POLLEFEYS (2017). « SEMANTIC3D. NET: A new large-scale point cloud classification benchmark ». In : *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. T. 4. ISPRS Foundation, p. 91-98 (cf. p. 9, 65).

HUANG, Xinyu, Xinjing CHENG, Qichuan GENG, Binbin CAO, Dingfu ZHOU, Peng WANG, Yuanqing LIN et Ruigang YANG (2018). « The ApolloScape Dataset for Autonomous Driving ». In : *arXiv: 1803.06184* (cf. p. 20, 21).

KIRILLOV, Alexander, Kaiming HE, Ross GIRSHICK, Carsten ROTHER et Piotr DOLLÁR (2018). « Panoptic Segmentation ». In : *arXiv preprint arXiv:1801.00868* (cf. p. 28, 42).

LIN, Tsung-Yi, Michael MAIRE, Serge BELONGIE, James HAYS, Pietro PERONA, Deva RAMANAN, Piotr DOLLÁR et C Lawrence ZITNICK (2014). « Microsoft coco: Common objects in context ». In : *European Conference on Computer Vision*. Springer, p. 740-755 (cf. p. 7, 47).

MADDERN, Will, Geoff PASCOE, Chris LINEGAR et Paul NEWMAN (2017). « 1 Year, 1000km: The Oxford RobotCar Dataset ». In : *The International Journal of Robotics Research (IJRR)* 36.1, p. 3-15. DOI : 10.1177/0278364916679498. eprint : http://ijr.sagepub.com/content/early/2016/11/28/0278364916679498.full.pdf+html. URL : http://dx.doi.org/10.1177/0278364916679498 (cf. p. 7, 19, 20).

MUNOZ, Delfina, J Andrew BAGNELL, Nicolas VANDAPEL et Martial HEBERT (2009). « Contextual classification with functional max-margin markov networks ». In : *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, p. 975-982 (cf. p. 8).

NEWCOMBE, Richard A, Shahram IZADI, Otmar HILLIGES, David MOLYNEAUX, David KIM, Andrew J DAVISON, Pushmeet KOHI, Jamie SHOTTON, Steve HODGES et Andrew FITZGIBBON (2011). « KinectFusion: Real-time dense surface mapping and tracking ». In : *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, p. 127-136 (cf. p. 14).

NIEMEYER, Joachim, Franz ROTTENSTEINER et Uwe SOERGEL (2014). « Contextual classification of lidar data and building object detection in urban areas ». In : *ISPRS Journal of Photogrammetry and Remote Sensing* 87, p. 152-165 (cf. p. 7).

PANDEY, Gaurav, James R MCBRIDE et Ryan M EUSTICE (2011). « Ford campus vision and lidar data set ». In : *The International Journal of Robotics Research* 30.13, p. 1543-1552 (cf. p. 7, 15, 17).

PAPARODITIS, Nicolas, Jean-Pierre PAPELARD, Bertrand CANNELLE, Alexandre DEVAUX, Bahman SOHEILIAN, Nicolas DAVID et Erwann HOUZAY (2012). « Stereopolis II: A multi-purpose and multi-sensor 3D mobile mapping system for street visualisation and 3D metrology ». In : *Revue française de photogrammétrie et de télédétection* 200.1, p. 69-79 (cf. p. 11).

ROYNARD, Xavier, Jean-Emmanuel DESCHAUD et François GOULETTE (2018b). « Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification ». In : *The International Journal of Robotics Research* 37.6, p. 545-557. DOI :

10.1177/0278364918767506. eprint : https://doi.org/10.1177/0278364918767506. URL : https://doi.org/10.1177/0278364918767506 (cf. p. 23, 115).

SCHARSTEIN, Daniel, Heiko HIRSCHMÜLLER, York KITAJIMA, Greg KRATHWOHL, Nera NEŠIĆ, Xi WANG et Porter WESTLING (2014). « High-resolution stereo datasets with subpixel-accurate ground truth ». In : *German Conference on Pattern Recognition*. Springer, p. 31-42 (cf. p. 7).

SERNA, Andrés, Beatriz MARCOTEGUI, François GOULETTE et Jean-Emmanuel DESCHAUD (2014). « Paris-rue-Madame database: a 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods ». In : *4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014*, p. 819-824 (cf. p. 10).

VALLET, Bruno, Mathieu BRÉDIF, Andrés SERNA, Beatriz MARCOTEGUI et Nicolas PAPARODITIS (2015). « TerraMobilita/iQmulus urban point cloud analysis benchmark ». In : *Computers & Graphics* 49, p. 126-133 (cf. p. 11, 12).

WANG, Shenlong, Min BAI, Gellert MATTYUS, Hang CHU, Wenjie LUO, Bin YANG, Justin LIANG, Joel CHEVERIE, Sanja FIDLER et Raquel URTASUN (2017). « Torontocity: Seeing the world with a million eyes ». In : *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, p. 3028-3036 (cf. p. 18, 19).

YU, Fisher, Wenqi XIAN, Yingying CHEN, Fangchen LIU, Mike LIAO, Vashisht MADHAVAN et Trevor DARRELL (2018). « BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling ». In : *arXiv preprint arXiv:1805.04687* (cf. p. 7).

Chapitre 2

Apprentissage Profond pour la Segmentation Sémantique de Nuages de Points 3D

« *We Need To Go Deeper* »

Leonardo DiCaprio a.k.a.
Dom Cobb (Inception)

Sommaire

2.1 Introduction

La récente disponibilité de capteurs 3D (LiDARs, caméras de profondeur ...) et des systèmes d'acquisition par MLS abordables, permet de créer facilement des cartes 3D précises des réseaux routiers, et en particulier de créer des *HD-Maps* pour les véhicules autonomes.

Ces cartes contiennent des informations nécessaires à la localisation et la planification et prise de décision des véhicules autonomes comme tous les moyens de signalisation (verticaux et au sol), les feux de circulation, les voies de circulation mais aussi l'empreinte des bâtiments ou les places de stationnement.

Cependant les nuages de points acquis par MLS sont vides de sens (ils ne contiennent que l'information géométrique brute et éventuellement une information radiométrique ou colorimétrique par point), on a donc besoin de méthodes de segmentation sémantique pour en extraire des informations utiles.

Les résultats obtenus par les méthodes d'apprentissage profond en image donnent l'espoir d'obtenir des résultats similaires sur les données de nuages de points 3D. Il existe déjà de nombreuses contributions en segmentation sémantique de scènes de nuages de points 3D, mais très peu utilisent l'apprentissage profond et obtiennent de meilleurs résultats qu'avec des méthodes classiques.

En particulier aucun article n'explique comment entraîner un réseau de classification par point sur un jeu de données de scènes de nuages de points annotés. De plus les descripteurs multi-échelles qui offrent les meilleurs résultats parmi les méthodes classiques n'ont pas été adaptés aux réseaux convolutionnels sur grilles voxeliques. On tentera donc de répondre à ces problèmes.

Un état de l'art des méthodes de segmentation sémantique de scènes de nuages de points 3D est présenté en section 2.2 page suivante (en particulier les applications de l'apprentissage profond aux données 3D). Le lecteur trouvera également un état de l'art des méthodes d'apprentissage profond en image en annexe A page 117. On présente ensuite en section 2.3 page 53 deux contributions, à savoir des méthodes d'entraînement de réseaux de neurones profonds sur scènes de nuages de points 3D complètement annotés, et des architectures de réseaux convolutionnels multi-échelles sur grilles voxeliques. Enfin en section 2.5 page 69 on présente les résultats obtenus par ces différentes méthodes comparées entre elles et par rapport à l'état de l'art.

2.2 État de l'Art en Segmentation Sémantique de Scènes de Nuages de Points 3D

2.2.1 Qu'est-ce que la Segmentation Sémantique de Nuages de Points 3D ?

On distingue plusieurs tâches que l'on peut réaliser sur une scène de nuages de points 3D :

— la classification (classifie l'ensemble de la scène comme un tout),

— la détection et le positionnement des objets (trouve des boîtes englobantes des objets),

— la segmentation (chaque point dans une instance d'objet, mais pas de classe),

— la segmentation sémantique (annotation de toute la scène : chaque point dans une classe),

— la segmentation panoptique [KIRILLOV et collab., 2018] (annotation de toute la scène, chaque point dans une classe et une instance d'objet ou une classe background).

Un nuage labellisé par points peut alors servir à créer des cartes précises en 3D des villes (ou de tout environnement intérieur ou extérieur), on pense en particulier aux *HD-Maps* qui sont des cartes contenant les informations utiles aux véhicules autonomes pour leur navigation (localisation et prise de décision), comme les panneaux de signalisation et signalisations au sol, les voies de circulations, les places de parking, la position des bâtiments...

La figure 2.1 montre les étapes pour construire une *HD-Map* à partir d'un nuage brut en passant par le nuage labellisé grâce à une méthode de segmentation sémantique.

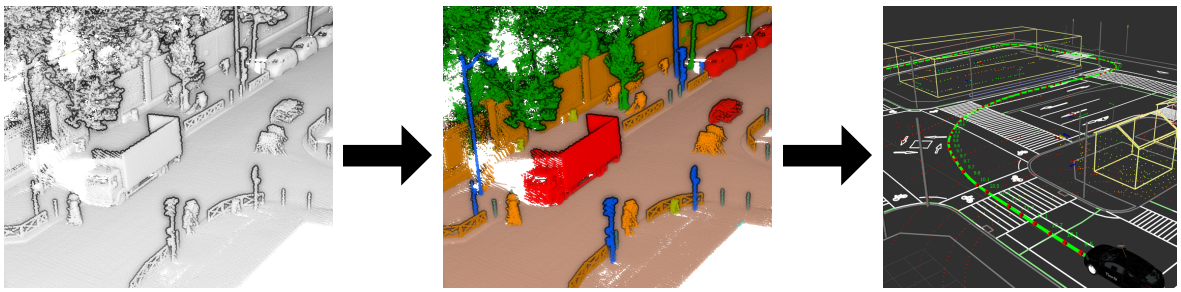


FIGURE 2.1 – Montre une application des méthodes de segmentation sémantique. Le nuage brut (à gauche) est labellisé (au milieu) grâce à une méthode de segmantion sémantique. On peut alors extraire de ce nuage labellisé les informations nécessaires pour construire une *HD-Map* (à droite).

Il existe une grande variété de travaux de recherche sur la segmentation sémantique de scènes de nuages de points 3D et plus généralement sur la classification et segmentation de données 3D. [GRILLI et collab., 2017] et [CHE et collab., 2019] présentent des revues de ces méthodes. On décrit ici les méthodes de l'état de l'art dans l'ordre suivant :

1. les architectures haut niveau de méthodes de segmentation sémantique en section 2.2.2,
2. les différentes approches pour la segmentation en section 2.2.3 page 44,
3. les différentes approches pour la classification en section 2.2.4 page 47.

2.2.2 Architectures des méthodes de segmentation sémantique

Les méthodes de segmentation sémantique de scènes de nuages de points 3D peuvent généralement se classer dans une des deux approches suivantes :

— découper le nuage en objets puis classifier chaque objet, présenté en section 2.2.2.1 page ci-contre,

— classifier chaque point, puis les regrouper en objets, présenté en section 2.2.2.2,

mais il existe également quelques méthodes qui font la segmentation et la classification en même temps, présentées en section 2.2.2.3.

2.2.2.1 Segmenter puis classifier

Cette approche est utilisée par [GOLOVINSKIY, KIM et collab., 2009], qui segmente la scène en deux étapes : d'abord une détection des objets puis une segmentation au point près est réalisée par algorithme de Min-Cut. Pour la classification, des descripteurs globaux sont calculés sur chaque objet, puis l'objet est classifié grâce à un classificateur classique (KNN, SVM, Random Forest...). Pour une illustration de ce pipeline voire figure 2.2.

FIGURE 2.2 – Étapes de la méthode de [GOLOVINSKIY, KIM et collab., 2009] (image tirée de l'article).

Cette approche a été reprise de nombreuse fois, avec des améliorations ou variantes dans chacunes des étapes, mais très peu de changement dans l'organisation du pipeline. En détail, le pipeline suit les étapes :

— extraction du sol (et éventuellement des façades),

— détection des objets (optionnel),

— segmentation des objets,

— calcul des descripteurs globaux sur chaque objet,

— classification de chaque objet grâce à un classificateur.

2.2.2.2 Classifier puis segmenter

L'autre méthode naturelle « Classification puis Segmentation » utilisée par exemple dans [LALONDE et collab., 2006] et [WEINMANN et collab., 2015], est constituée des étapes suivantes :

— calcul de descripteurs locaux sur chaque point,

— classification de chaque point (en une catégorie géométrique ou sémantique),

— régularisation spatiale des classes prédites (optionnel),

— regroupement des points similaires voisins en objets.

[Loïc LANDRIEU et collab., 2017] propose une méthode avec régularisation, le pipeline complet est décrit en figure 2.3 page suivante.

2.2.2.3 Classifier et segmenter en même temps

Il existe également des architectures où la classification et la segmentation sont plus intriquées, ceci pouvant permettre dans certains cas d'être bénéfique à chacune des deux tâches. Certaines de ces méthodes sont :

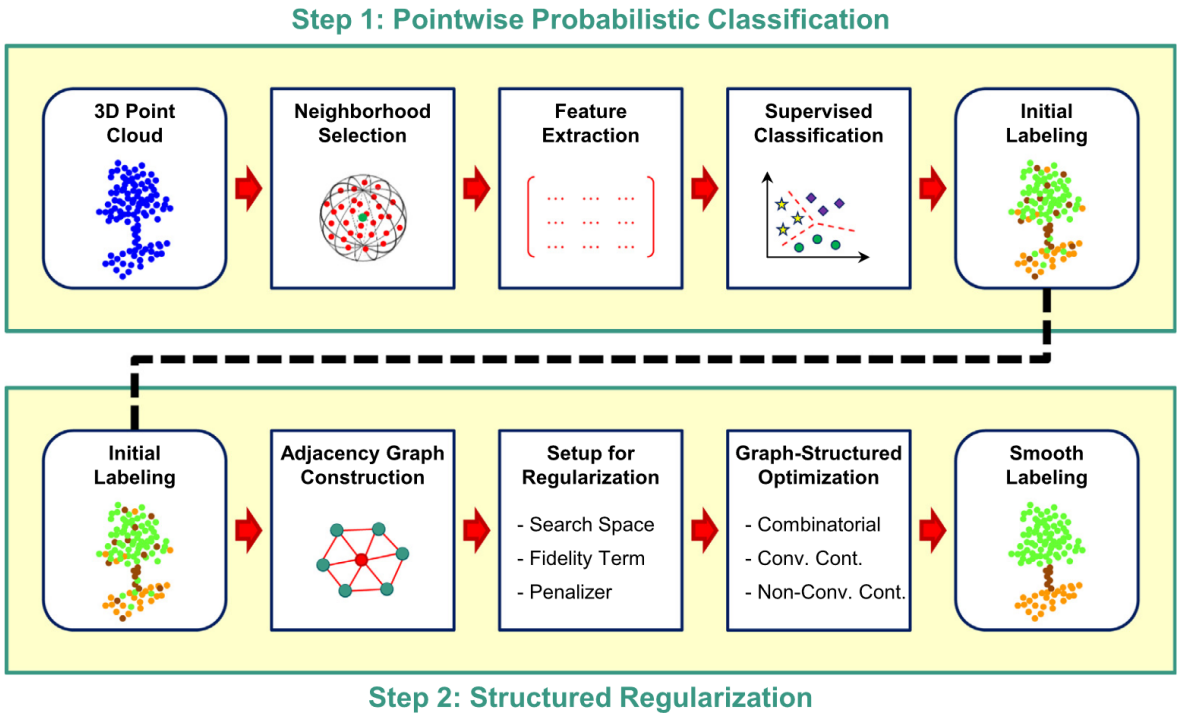


FIGURE 2.3 – Étapes de la méthode de [Loïc LANDRIEU et collab., 2017] (image tirée de l'article).

— l'ajustement de modèles (prédéfinis dans un dictionnaire) à des parties de la scène [JOHNSON et HEBERT, 1999]. Quand un modèle du dictionnaire est effectivement recalé avec une partie de la scène, on a en même temps segmenté une instance de l'objet correspondant au modèle et classifié cette partie grâce à l'information sémantique apportée par le modèle,

— certain réseaux de neurones de détection d'objets (présentés en annexe A.6.5 page 144) permettent, en ajoutant une sortie de classification, d'obtenir la classe de chaque objet détecté. Cependant en l'état ces réseaux ne permettent pas de segmenter toute la scène puisqu'ils servent seulement à détecter certains objets, mais pas toutes les classes ni l'arrière plan ou le sol et les bâtiments.

— [X. WANG et collab., 2019] propose une architecture de réseau de neurones qui combine un réseau de segmentation sémantique et un réseau de détection d'objets pour produire en sortie un nuage dont chaque point appartient à une classe et à une instance d'objet. Pour ce faire, ils introduisent le module *ASIS* qui permet de partager de l'information entre les deux parties du réseau, ce module est décrit en figure 2.4 page ci-contre. Contrairement aux réseaux de détection d'objet mentionnés ci-dessus, les deux tâches peuvent tirer de l'information des résultats obtenus pour l'autre tâche.

2.2.3 Méthodes de Segmentation

Les méthodes de segmentation ont pour but de découper un nuage de points (pré-classifié ou pas) en sous-nuages distincts représentant chacun un objet d'intérêt. Ces méthodes peuvent être séparées entre :

— les méthodes classiques présentées en section 2.2.3.1,

— les méthodes apprises présentées en section 2.2.3.2 page 46.

2.2.3.1 Segmentation classique

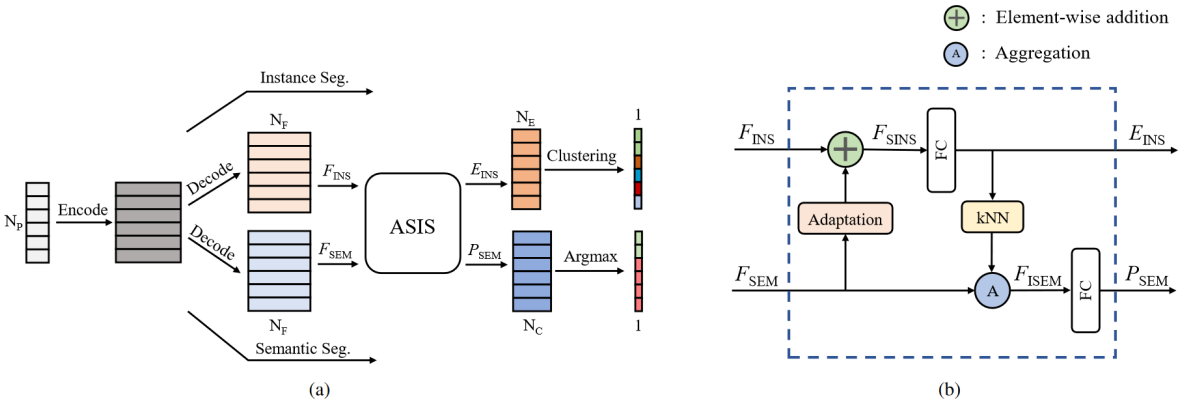


FIGURE 2.4 – (a) architecture complète du réseau qui combine segmentation sémantique et segmentation d'objets, (b) architecture du module *ASIS*. (image tirée de [X. WANG et collab., 2019])

Les méthodes de segmentation les plus classiques se basent généralement sur une des approches suivantes :

— par détection des arêtes [SAPPA et DEVY, 2001] ; [WANI et ARABNIA, 2003] qui sont constituées de deux étapes principales : (1) la détection des arêtes/frontières entre les différents objets, (2) le regroupement des points à l'intérieur des frontières pour déterminer les différents segments. Les arêtes sont détectées comme étant situées aux points où certaines propriétés locales varient au delà d'un certain seuil. Les propriétés locales les plus utilisées sont les normales, les gradients, la courbure ou les dérivées d'ordre supérieur. Ces méthodes détectent souvent des bords déconnectés dans les nuages bruités ou de densité inégale, ce qui rend difficile l'identification des segments fermés sans procédure de remplissage [CASTILLO et collab., 2013].

— par croissance de région [Paul J. BESL et JAIN, 1988] / extraction de composantes connexes. Ces méthodes choisissent une ou plusieurs graines (*seed* en anglais), puis font croître des régions autour. Les points voisins d'un point de la région sont ajoutés s'ils vérifient certains critères de similarité comme un angle suffisamment petit entre les normales aux deux points. Dans [ROYNARD, J. DESCHAUD et collab., 2016] nous utilisons d'abord une croissance de région pour extraire le plus grand plan horizontal puis on utilise plus simplement l'extraction de composante connexes pour obtenir chaque objet de la scène (qui ne sont plus voisins sans le sol). [VOSSELMAN et collab., 2004] introduit des critères colorimétriques en plus des critères géométriques. Ces méthodes sont plus robustes au bruit [Yu LIU et XIONG, 2008], mais sont sensibles au choix des graines et à l'estimation des normales.

— méthodes par votes :

  — la transformée de Hough [HOUGH, 1962], initialement introduite pour détecter les droites dans des images, consiste à transformer l'image dans un espace de paramètres de droites. La transformée de Hough a été généralisée à des formes plus complexes que des droites par [BALLARD, 1981], puis appliquée aux nuages de points par [VOSSELMAN et collab., 2004] pour extraire des plans, des cylindres ou des sphères.

  — le RANSAC pour *Random Sample Consensus* [FISCHLER et BOLLES, 1981]. Pour la détection de plan, on tire aléatoirement 3 points dans le nuage et compte le nombre de points qui tombent dans l'unique plan passant par ces trois points, ce processus est répété plusieurs fois et le plan comptant le plus de votes est gardé. Cette méthode peut être utilisée pour de nombreux autres modèles géométriques comme les sphères, cylindres, cônes et tores [SCHNABEL et collab., 2007].

  Dans le cas où les modèles ont une signification sémantique, alors une telle approche produit également une classification de la scène.

— par découpage de graphes (*Graph-Cut*), ces méthodes consistent à chercher et supprimer

les arêtes de poids le moins élevé d'un graphe. Après avoir détecté une position approximative des objets, [GOLOVINSKIY et FUNKHOUSER, 2009] les segmente par rapport à l'arrière plan grâce à un un algorithme de *Min-Cut*.

— par morphologie mathématique, [SERNA et MARCOTEGUI, 2014] propose d'extraire les objets de la scène par une suite d'opération de morphologie mathématiques effectuées sur une image d'élévation de la scène. Chaque opération a pour but de remplir les trous de l'image 2.5D puis de filtrer ou récupérer des objets de formes ou tailles différentes qui n'avaient pas étés traités pas les opérations précédentes.

— par super-voxelisation [PAPON et collab., 2013] ; [AIJAZI, CHECCHIN et collab., 2013] inspiré des superpixels [X. REN et MALIK, 2003]. Ces méthodes consistent à voxeliser la scène, puis grouper les voxels voisins et ayant des propriétés similaires en « super-voxels ». Les super-voxels peuvent alors être regroupés en objets grâce à d'autres méthodes de segmentation comme le découpage d'un graphe des super-voxels. Ceci a l'avantage de réduire considérablement le nombre d'éléments du graphe et donc d'accélérer les calculs.

Il existe également des méthodes de segmentation hybrides, qui combinent plusieurs des méthodes précédentes, afin d'exploiter la force d'une méthode et de contourner la faiblesse d'une autre méthode. Par exemple [AIJAZI, SERNA et collab., 2015] compare les méthodes par morphologie mathématique de [SERNA et MARCOTEGUI, 2014] et de super-voxelisation de [AIJAZI, CHECCHIN et collab., 2013], et propose une approche simple pour combiner et tirer partie des forces de ces deux approches.

2.2.3.2 Segmentation par apprentissage

Les méthodes de segmentation précédentes sont basées sur des heuristiques, de telles approches peuvent être efficaces sur certains types de données mais s'adapteront difficilement à d'autres types de nuages. Les méthodes suivantes de segmentation par apprentissage ont l'avantage d'apprendre une façon efficace de segmenter les nuages sur un jeu de données et donc de s'adapter aux spécificités des données à traiter. La plupart de ces méthodes sont conçues pour agir après une étape de classification par points.

— les modèles graphiques :
  — les *Markov Random Fields* (MRF) initialement proposés par [KINDERMANN, 1980] modélisent la probabilité jointe qui lie les observations aux labels sur un graphe sous certaines conditions, entre autres de vérifier une propriété de Markov (les variables aléatoires sur les nœuds non-voisins sont indépendantes). Les MRF ont été adaptés aux nuages de points par [ANGUELOV et collab., 2005] ; [Daniel MUNOZ et collab., 2009] ; [SHAPOVALOV et collab., 2010] ; [NAJAFI et collab., 2014] grâce à la structure de graphe qui peut être déduite des relations de voisinage des points,
  — les *Conditional Random Fields* (CRF) initialement proposés par [LAFFERTY et collab., 2001] sont la version déterministe des MRF. Ils permettent, après classification de chaque nœud d'un graphe, de régulariser la classification grâce à la modélisation des interactions entre nœud voisins. Cette approche a été adaptée aux nuages de points par [R. ZHANG et collab., 2015] ; [WEINMANN et collab., 2015], les points sont alors pris comme nœuds du graphe, et les arêtes connectent un point à tous ses voisins dans une certaine sphère.

— les réseaux de neurones profonds pour la détection et segmentation d'objets sur images 2D présentés en annexe A.6.5 page 144, à savoir *Mask-RCNN* [HE, GKIOXARI et collab., 2017], *SSD* [W. LIU et collab., 2016] et *YOLO* [REDMON et collab., 2016] peuvent s'adapter à la 3D en utilisant des grilles voxeliques. Ces réseaux produisent en sortie des boîtes englobantes (ou des masques) des objets présents dans l'image (ou la grille voxelique), elles ont l'avantage de pouvoir produire également une classe pour chaque objet segmenté.

2.2.4 Méthodes de Classification

Les nuages de points 3D ont plusieurs défauts qui compliquent la tâche de classification :

— ils sont non-structurés,

— ils sont non-ordonnées,

— ils n'ont pas de voisinages implicites.

Il est donc plus simple de convertir le nuage de points en une autre représentation ne présentant pas ces défauts ou au moins que l'on sait traiter. On distingue :

— les méthodes projetant le nuage sur une image 2D présentées en section 2.2.4.1

— les méthodes projetant le nuage sur une grille 3D présentées en section 2.2.4.2 page suivante

— les méthodes construisant un maillage à partir du nuage présentées en section 2.2.4.3 page suivante

Il existe tout de même des méthodes utilisant directement les données brutes sous forme de nuages de points 3D, elles seront abordées en section 2.2.4.4 page 49

2.2.4.1 Classification par images

L'utilisation d'images réduit la complexité et donc les temps de calcul par rapport aux nuages de points 3D, et permet d'utiliser des méthodes prouvées et robustes de traitement de l'image [HOOVER et collab., 1996]. De plus en apprentissage profond ces méthodes bénéficient considérablement de l'expertise existante en 2D et des réseaux pré-entraînés sur les jeux de données d'images comme ImageNet [DENG et collab., 2009] et MS COCO [LIN, MAIRE et collab., 2014] qui contiennent bien plus de données que les jeux de nuages de points annotés comme Paris-Lille-3D et Semantic3D ou de modèles 3D comme ModelNet [Z. WU et collab., 2015].

Il existe plusieurs façons de transformer un nuage de point en image(s) utile(s) pour la classification :

— image de réflectance/intensité du LiDAR : [EL-ASHMAWY et collab., 2011] les utilise pour classifier des images aériennes grâce à un classificateur par maximum de vraisemblance,

— image RVB rendue depuis une caméra virtuelle : [BOULCH, SAUX et collab., 2017] les utilise pour appliquer des réseaux de l'état de l'art en segmentation sémantique d'images (comme ceux décrits en annexe A.6.3 page 141), ces réseaux produisent une classe pour chaque pixel qui peut être ensuite re-projetée sur le nuage de point d'origine. Cette méthode est classée 5ème sur le benchmark *reduced-8* de Semantic3D, et 2ème sur le benchmark *semantic-8*,

— image de profondeur, depuis une caméra virtuelle ou directement depuis le capteur : [GORTE, 2007] les utilise pour faire de l'extraction de plans, [ZHU et collab., 2010] les segmente grâce à une méthode de découpage de graphes puis classifie chaque objet grâce à des SVM ou des arbres de décision,

— image de panorama : [SFIKAS et collab., 2017] produit ce type d'images en projetant la surface d'un objet (sous forme de nuage de point ou de maillage) sur un cylindre situé autour de l'objet, ce sont les informations d'orientation de la surface et de distance à l'axe du cylindre qui sont projetées. Un réseau convolutionnel de classification de l'état de l'art est alors appliqué pour obtenir la classe de l'objet,

— image d'élévation : [SERNA et MARCOTEGUI, 2014] les utilise pour segmenter la scène grâce des opérateurs de morphologie mathématique, puis pour calculer des descripteurs statistiques, géométriques et contextuels simples avant de classifier chaque objet grâce à des SVM.

Une des meilleures contributions pour la classification d'objets 3D sur le jeu de données ModelNet est [KANEZAKI et collab., 2016], ils donnent plusieurs vues d'un même objet au réseau et demandent en sortie (en plus de la classe de l'objet) à quelle position se trouve la prise de vue. Ceci aide le réseau à construire des représentations plus utiles pour la classification.

Une des difficultés avec les approches par images pour la segmentation sémantique de scènes complètes est le choix de la position et orientation de la ou des prise(s) de vue. Les variations de densités ou les occlusions complexifient le choix de points de vues générant des images les plus complètes possibles

2.2.4.2 Classification par grilles voxeliques

Une autre façon transformer un nuage de points en une structure plus régulière et qui facilite les traitements est de la projeter sur une grille voxelique. On a déjà vu en section 2.2.3 page 44 que l'utilisation de voxels et super-voxels pouvait aider la segmentation.

Les premiers réseaux convolutionnels profonds utilisés pour classifier des nuages de points 3D datent de 2015 avec *VoxNet* [MATURANA et SCHERER, 2015], ce réseau classifie une instance d'objet en remplissant une grille d'occupation ou de densité puis applique un réseau convolutionnel 3D. Le fait d'utiliser des grilles 3D au lieu d'images 2D ne change que la dimension des noyaux de convolutions, mais toutes les architectures et méthodes développées en images (et décrites en annexe A page 117) peuvent être utilisées en 3D. En particulier les réseaux de segmentation sémantique type *FCNet*, *SegNet* ou *U-Net* présentées en annexe A.6.3 page 141.

Par exemple les meilleurs résultats obtenus sur le jeu de données de classification ModelNet sont réalisés par des réseaux convolutionnels très profonds [BROCK et collab., 2016] basés sur des architectures de type *Inception-ResNet* [SZEGEDY, IOFFE et collab., 2017] et en faisant voter plusieurs réseaux sur plusieurs vues 3D des objets.

Une façon d'utiliser ces réseaux pour la segmentation sémantique de scènes complètes (et plus d'objets 3D seuls) est de les utiliser comme descripteurs du voisinage de chaque point de la même façon que dans [Martin WEINMANN et collab., 2015]. [J. HUANG et YOU, 2016] a appliqué ce type de réseau pour classifier des nuages de points urbains, le réseau prédit alors la classe de chaque point à partir de la grille d'occupation de son voisinage. Cependant cette publication pose plusieurs problèmes :

— les données d'expérimentation n'ont pas été publiées, ce qui empêche de comparer cette méthode aux autres, en particulier de savoir si cette méthode par apprentissage profond est meilleure que les méthodes classiques,

— la méthode d'entraînement du réseau n'est pas décrite, on ne sait pas si chaque point du jeu d'entraînement est vu par le réseau pendant chaque *epoch*.

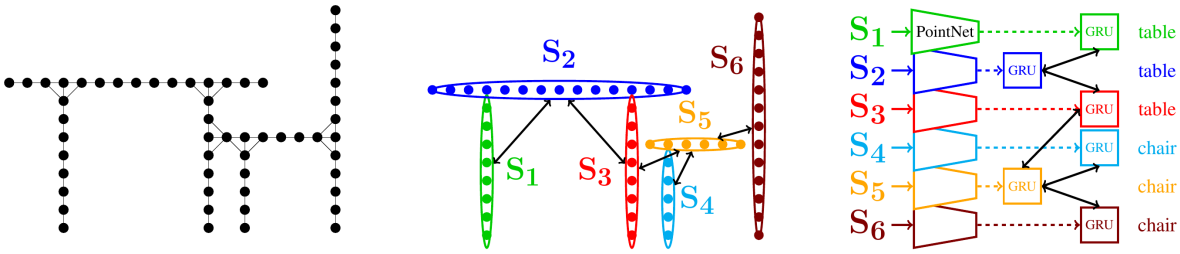
Sur les benchmarks Semantic3D et S3DIS, [TCHAPMI et collab., 2017] utilise des réseaux de segmentation sémantique pour classifier des grilles de voxels entières et plus uniquement le point central, ce qui permet de réduire considérablement le temps d'inférence d'un nuage complet.

Il existe également des méthodes utilisant des grilles 3D implicites comme les *octrees* qui tirent profit du caractère éparse des nuages de points (et donc des grilles sur lesquelles ils sont projetés), ces méthodes sont décrites en détail en section 3.2.2 page 92.

2.2.4.3 Classification par graphes

Une autre approche est d'utiliser des graphes, en effet, le nuage de points à l'état brut n'ayant aucune structure il est très compliqué d'en tirer une information "agrégée"/globale. Alors qu'un graphe donne des relation de voisinages et de distances entre les points et permet par exemple de faire des convolutions, bien ce soit de façon moins évidente que sur des grilles voxeliques.

Les méthodes classiques de CRF ont été adaptées en apprentissage profond pour pouvoir régulariser la segmentation sémantique d'image, et pour pouvoir l'apprendre de bout en bout en



(a) Input point cloud (b) Superpoint graph (c) Network architecture

FIGURE 2.5 – Pipeline de la méthode *SPGraph* [Loic LANDRIEU et SIMONOVSKY, 2018].

sortie d'un réseau convolutionnel, en particulier [KAMNITSAS et collab., 2017] CNN 3D et des deep-CRF pour la segmentation sémantique d'image médicales 3D. Cette approche a ensuite été appliquée par *SEGCloud*[TCHAPMI et collab., 2017] pour la segmentation sémantique de grille voxelique de scènes de nuages de points 3D, en utilisant la structure de graphe sous-jacente à ces grilles voxeliques pour les deep-CRF. Cette méthode est classée 4ème sur le benchmark *reduced-8* du jeu de données Semantic3D.

Après avoir segmenté le nuage en *super-points* par une méthode de sur-segmentation et calculé des descripteurs sur chaque *super-point* grâce à un réseau de type *PointNet*, *SPGraph* [Loic LANDRIEU et SIMONOVSKY, 2018] construit un graphe de *super-points* qui va permettre d'affiner les descripteurs de chaque *super-points* grâce à des méthodes de convolutions par graphes. Ceci est réalisé grâce à des réseaux récurrents (ici des GRU) sur chaque *super-point* qui intègre des informations venant des autres *super-points* à travers les arêtes du graphe de *super-points*. Le pipeline complet de *SPGraph* est décrit en figure 2.5. Cette méthode est classée première sur le benchmark *reduced-8* du jeu de données Semantic3D.

2.2.4.4 Classification par nuages de points

Enfin on peut réaliser la classification des points en utilisant directement le nuage sous sa forme brute, soit en calculant des descripteurs sur les voisinages de chaque point, soit en donnant le nuage en entrée d'un réseau de neurones spécialisé. Ce sont les deux approches qui seront présentées dans cette section.

On peut classifier chaque point en étudiant ses voisinages, ceci soulève le problème du choix des voisinages (et surtout de leur taille). Une solution est apportée dans [DEMANTKÉ et collab., 2011] qui propose de choisir la taille (nombre de voisins ou bien rayon du voisinage) qui minimise une entropie calculée sur des caractéristiques appelées « attributs de dimensionnalité » extraites de la matrice de covariance du voisinage. Cette méthode propose de classer chaque point dans une classe géométrique : linéaire, surfacique ou volumique. Ces attributs de dimensionnalité sont utilisés dans plusieurs méthodes [DOUILLARD et collab., 2011] ; [LALONDE et collab., 2006] ; [Martin WEINMANN et collab., 2015]. Une autre méthode qui permet d'éviter de choisir la taille des voisinages est d'utiliser et de combiner des voisinages à plusieurs échelles [HACKEL, WEGNER et collab., 2016] pour capturer à la fois le contexte (dans les plus grands voisinages) et la forme locale de la surface (dans les plus petits voisinages). Cette approche multi-échelles a été encore améliorée par *RF_MSSF* [THOMAS et collab., 2018] qui utilise des voisinages sphériques au lieu d'un nombre de plus proches voisins et de l'*active-learning* pour choisir les points d'entrainement. Cette dernière méthode est longtemps restée classée 3ème sur le benchmark *reduced-8* du jeu de données Semantic3D derrière la méthode présentée dans cette thèse (2ème) et *SPGraph* (1ère).

Il existe également de nombreux descripteurs locaux comme VFH (*Viewpoint Feature Histogram*) [RUSU et collab., 2010] qui calculent la distribution de normales ou RSD (*Radius-based Shape Descriptor*) [MARTON, PANGERCIC, BLODOW et collab., 2010] qui estime le rayon de courbure localement. Ces derniers peuvent être utilisés pour calculer des descripteurs globaux de nuages de points 3D respectivement CVFH (*Clustered* VFH) [ALDOMA et collab., 2011] et GRSD (*Global* RSD)

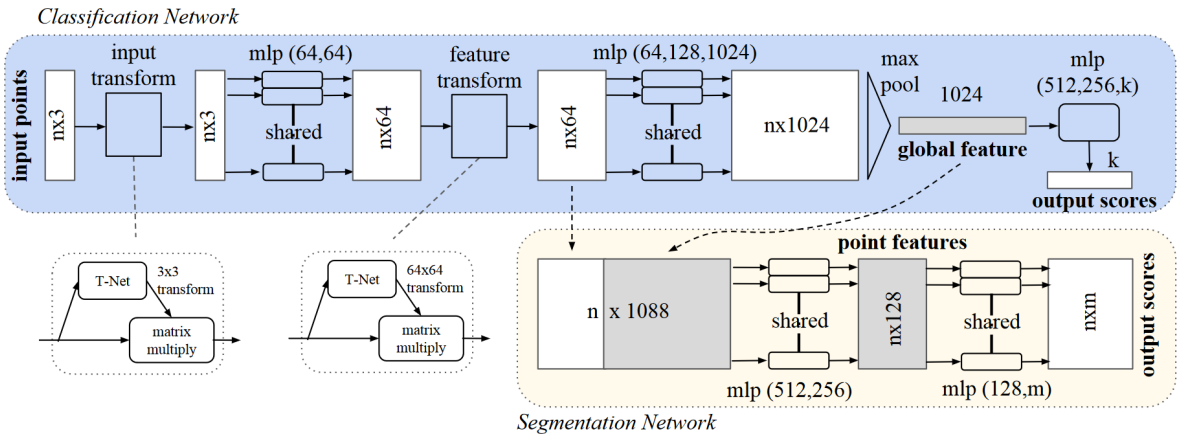


FIGURE 2.6 – Architecture du réseau *PointNet* [Charles R QI et collab., 2017].

[MARTON, PANGERCIC, RUSU et collab., 2010].

Il existe également d'autres descripteurs globaux comme ESF (pour *Ensemble of Shape Functions*) [WOHLKINGER et VINCZE, 2011] qui calcule des histogrammes de distance, d'angle ou de surface entre des groupes de points tirés aléatoirement dans le nuage, et d'autres descripteurs globaux qui décrivent plus simplement géométriquement ou statistiquement le nuage de points comme le volume ou la surface au sol du nuage [SERNA et MARCOTEGUI, 2014], ou simplement un histogramme du nombre de points selon l'axe vertical dans nos travaux [ROYNARD, J. DESCHAUD et collab., 2016].

Au commencement de cette thèse il n'existait aucune méthode par apprentissage profond qui prenne directement en entrée le nuage de points, mais de telles méthodes ont l'intérêt de travailler au plus près des données brutes, quelques unes sont donc apparu dès 2016 [Charles R QI et collab., 2017], et on peut imaginer qu'elles seront dans le futur parmi les plus efficaces puisqu'elles utilisent directement les données brutes.

Les premières méthodes de ce type sont basées sur l'observation qu'un nuage de point est un ensemble et vérifie donc quelques invariants/symétries (invariances par permutation des points ou par ajout de point déjà dans l'ensemble...) et donc se base sur l'utilisation d'opérateurs respectant ces symétries comme le *pooling* global (agréger les caractéristiques sur l'ensemble du nuage pour obtenir un unique vecteur de features), c'est l'approche utilisée par le réseau *PointNet* [Charles R QI et collab., 2017] qui est décrit en figure 2.6. Ce réseau peut être utilisé pour de la classification ou la segmentation sémantique, mais se limite à une nombre restreint de points car il prend beaucoup de place en mémoire pendant l'entrainement.

Mais ces architectures perdent l'aspect hiérarchique des calculs si intéressant pour les *CNN*, [Charles Ruizhongtai QI et collab., 2017] introduit donc le réseau *PointNet++* qui est décrit en figure 2.7 page suivante et qui utilise le *PointNet* pour agréger seulement localement le nuage à l'image d'un noyau de convolution d'un *CNN*. Ce mécanisme d'agrégation locale est réalisé de façon hiérarchique pour progressivement réduire le nombre de points et augmenter la représentativité des features (comme pour les *CNN* en images).

Deux autres approches sont proposées par [ENGELMANN et collab., 2017] qui permettent de mieux prendre en compte le contexte. La première utilise *PointNet* sur des voisinages à plusieurs échelles, la seconde utilise *PointNet* sur des nuages extraits sur une grille 2D et utilise des réseaux récurrents pour partager l'information entre les cases de la grille.

Il existe également des réseaux qui essayent de reproduire plus fidèlement les architectures des CNN en images comme *PointCNN* [Yangyan LI et collab., 2018] en proposant des vraies convolutions sur les points. Toute la difficulté étant de choisir comment sont définis les noyaux de convolution sur un espace continu et comment les implémenter efficacement.

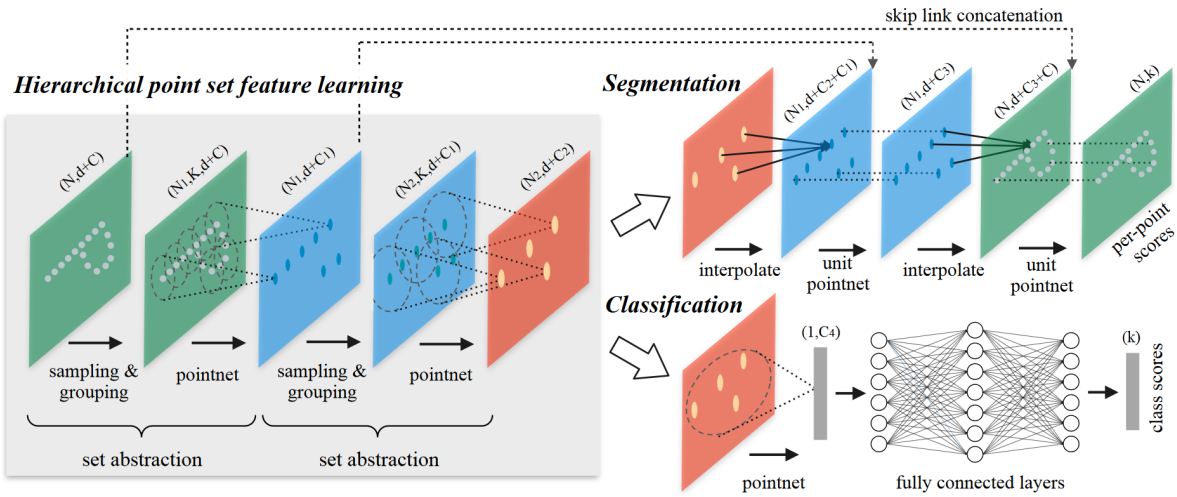


FIGURE 2.7 – Architecture du réseau *PointNet++* [Charles Ruizhongtai QI et collab., 2017] (image tirée de l'article). Ce réseau utilise *PointNet* pour agréger localement et hiérarchiquement les données à la façon des *CNN*.

2.2.5 Apprentissage profond pour d'autres tâches sur les nuages de points 3D

Le nombre d'articles publiés dans les quelques dernières années et la diversité des tâches traités sur des nuages de points 3D montrent l'engouement pour ce domaine de recherche. On citera entre autres :

— Estimation de flot/flux de nuage de points sur des scans Velodyne consécutifs : *FlowNet3D* [X. LIU et collab., 2018],

— Recalage de nuages [ELBAZ et collab., 2017],

— Estimation de normales [BOULCH et MARLET, 2016],

— Calcul de Normales et Courbure basé sur *PointNet*, *PCPNet* [GUERRERO et collab., 2018],

— Consolidation de nuages / détection d'arètes [L. YU et collab., 2018],

— Détection d'objets mobiles [XIAO et collab., 2017],

— Reconstruction 3D à partir de vues 2D d'un objet [YI et collab., 2017] ; [H. FAN et collab., 2017],

— Débruitage / suppression des artefacts basé sur *PointNet* : *PointCleanNet*, [RAKOTOSAONA et collab., 2019],

2.2.6 Problèmes posés et non résolus par l'état de l'art

Avec la disponibilité de capteurs 3D (LiDARs, caméras de profondeur ...) et des systèmes d'acquisition par MLS toujours plus abordables, les données nuages de points 3D deviennent accessibles. La segmentation sémantique des nuages de points 3D est donc indispensable pour tirer une information utile de toutes ces données.

Les résultats obtenus par les méthodes d'apprentissage profond en image donnent l'espoir d'obtenir des résultats similaires sur les données de nuages de points 3D. Il existe déjà quelques contributions qui essayent d'utiliser l'apprentissage profond pour la segmentation sémantique des nuages de points 3D urbains. En particulier les approches les plus similaires à celles qu'on développera par la suite sont :

— [J. HUANG et YOU, 2016] pour l'utilisation de *CNN* sur grilles d'occupation voxeliques pour la segmentation sémantique de nuages de points urbains,

— [HACKEL, WEGNER et collab., 2016] ; [THOMAS et collab., 2018] pour l'utilisation de descripteur multi-échelles.

Mais ces approches posent plusieurs questions non résolues par l'état de l'art :

— il n'est expliqué dans aucun article comment les réseaux sont entraînés sur des jeux de données de scènes de nuages de points complètement annotés. Est-ce que tous les points sont utilisés à chaque *epoch*? Ou bien est-ce que seulement un nombre fixe de points est choisi au début de l'entraînement comme dans [Martin WEINMANN et collab., 2015] ?

— les réseaux convolutionnels 3D de classification à partir de grille d'occupation obtiennent des résultats médiocres sur les scènes urbaines. Comment améliorer ces résultats pour battre l'état de l'art des méthodes non profondes?

— les méthodes multi-échelles classiques n'ont pas été adaptées en apprentissage profond. En particulier quelle est la meilleur façon de fusionner les features obtenues par un réseau convolutionnel à plusieurs échelles?

On propose des solutions à ces questions en section 2.3 page ci-contre.

2.3 Méthode Proposée de Segmentation Sémantique de Scènes de Nuages de Points 3D par Apprentissage Profond

On propose dans cette partie une méthode de segmentation sémantique des scènes de nuages de points 3D en intérieur ou extérieur grâce à un réseau de neurones convolutionnel profond prenant en entrée des grilles voxeliques à plusieurs échelles dont un aperçu est donné en figure 2.8. Les contributions portent en particulier sur deux points :

— Comment entraîner un réseau de neurones sur un jeu de données de nuages de points 3D urbains complètement annoté comme Paris-Lille-3D présenté en section 1.3 page 23 dont les classes sont très déséquilibrées et où la plupart des échantillons n'apportent pas de nouvelles informations ?

— Comment fusionner les données à plusieurs échelles dans un réseau de neurones convolutionnel ?

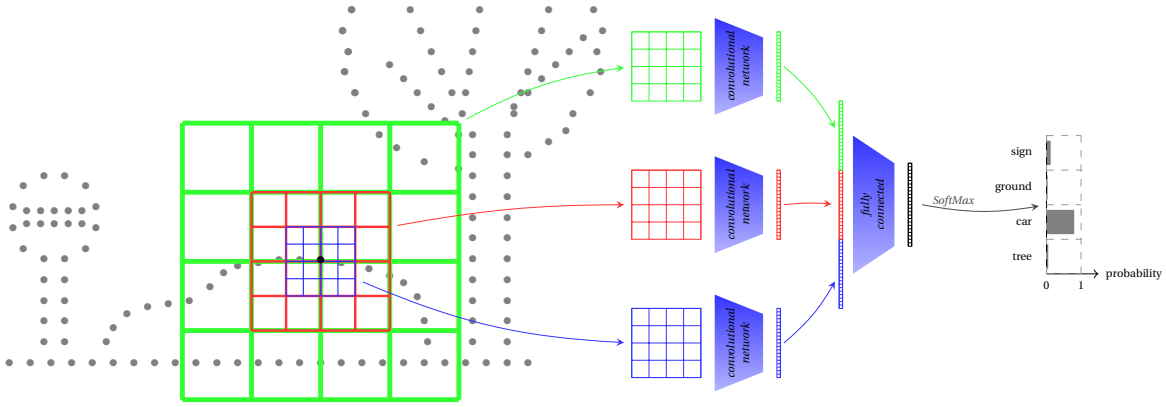


FIGURE 2.8 – Aperçu de la méthode de segmentation sémantique par réseau convolutionnel 3D multi-échelles présentée dans ce chapitre (représenté en 2D pour la simplicité).

Nous allons maintenant progressivement aller des idées les plus naïves aux idées les plus évolués pour présenter ces contributions :

— on expliquera comment entraîner un tel réseau sur des scènes de nuages de points 3D complètement annotées en section 2.3.1 page suivante,

— on décrira rapidement la façon d'adapter un réseau convolutionnel 2D basique (*AlexNet* ou *VGG*) ou reprendre un réseau 3D basique (*VoxNet*) qui servent à la classification d'une grille (2D ou 3D) pour l'utiliser pour classifier un unique point. en section 2.3.2.1 page 56,

— on expliquera comment tirer profit d'entrées multi-échelles pour un réseau convolutionnel :

  — comment fusionner des entrées multi-échelles de façon naïve (fusion-précoce en section 2.3.2.2 page 57 ou fusion-tardive en section 2.3.2.3 page 57),

  — comment fusionner des entrées multi-échelles de façon moins naïve en section 2.3.2.5 page 58,

— sur cette base, on adaptera des architectures classiques de segmentation sémantique d'image (*SegNet, U-Net*) aux données multi-échelles en section 2.3.3 page 60,

— on présentera quelques perspectives d'améliorations en section 2.6 page 77.

Cependant on n'utilisera pas toutes les méthodes présentées en annexe A page 117 pour améliorer nos réseaux jusqu'au niveau de l'état de l'art en classification d'image (*ResNet, Inception*, ...), ceci n'étant pas nécessaire pour valider les méthode développées.

2.3.1 Entraînement sur des Scènes de Nuages de Points 3D

La méthode d'entraînement classique d'un réseau de neurones profond (feed-forward) par back-propagation des gradients consiste à faire voir au réseau une fois chaque échantillon du jeu de donnée dans un ordre aléatoire. Ce processus appelé « *epoch* » est renouvelé autant de fois que nécessaire pour faire converger les paramètres du réseau.

Pour la tâche de segmentation sémantique de nuages de points 3D, chaque point est un échantillon dont on veut inférer la classe. Or les jeux d'apprentissage de tels nuages sont constitués de centaines de millions voire de milliards de points comme on a pu le voir au chapitre 1 page 5. On pourrait envisager d'appliquer la méthode classique « par *epochs* », mais cela pose trois problèmes :

— la recherche du voisinage d'un point est une étape nécessaire qui prend un temps important (avec une recherche de voisinage par point de l'ordre de 0,1 s, il faudrait au moins 14 jours pour faire une *epoch* sur 100 millions de points en utilisant 8 coeurs),

— les jeux de données de nuages de points 3D sont très déséquilibrés en nombre de points par classe, le réseau ne verrait donc presque que des points du sol ou des bâtiments (classes les plus représentées) et très peu les piétons, voitures et mobiliers urbains, voir tableau 2.1 page ci-contre.

— une partie importante des points de ces jeux de donnés ne présente que peu de variabilité géométrique (ce sont globalement des plans sur le sol et les bâtiments, voir figure 2.9), il ne semble donc pas utile de passer sur les millions de points représentés par des plans.

FIGURE 2.9 – Nuages de points annotés des jeux de données Paris-Lille-3D (à gauche) et Semantic3D (à droite). Ces images montrent l'important déséquilibre entre les classes et le fait que beaucoup de points se ressemblent (localement) les endroits variés sont très localisés. On observe en particulier qu'une majorité des points se trouvent sur un plan (pour le sol et les bâtiments).

	Class	Number of samples (of points)							
		Lille1		Lille2		Paris		Total	
surface	unclassified	1	(54.88k)	1	(16.47k)	1	(60.79k)	3	(132.1k)
	other	16	(70.15k)	11	(14.10k)	11	(30.82k)	38	(115.1k)
	road	1	(34.80M)	1	(11.82M)	1	(19.68M)	3	(66.30M)
	sidewalk	18	(8.566M)	8	(4.97M)	5	(4.6M)	31	(16.67M)
	island	7	(458.8k)	0	(0)	9	(82.46k)	16	(541.2k)
	vegetation	32	(562.2k)	22	(512.4k)	6	(157.1k)	60	(1.232M)
	building	34	(18.2M)	8	(7.934M)	5	(9.431M)	47	(35.39M)
	post	9	(13.51k)	0	(0)	0	(0)	9	(13.51k)
static	bollard	122	(34.80k)	64	(7.982k)	84	(28.33k)	270	(71.10k)
	floor lamp	72	(232.9k)	13	(23.69k)	21	(113.2k)	106	(369.7k)
	traffic light	14	(25.82k)	11	(27.41k)	16	(80.76k)	41	(133.10k)
	traffic sign	82	(113.6k)	39	(69.89k)	17	(30.75k)	138	(214.3k)
	signboard	20	(68.34k)	12	(43.14k)	3	(13.41k)	35	(124.9k)
	mailbox	0	(0)	0	(0)	1	(4.739k)	1	(4.739k)
	trash can	11	(14.72k)	6	(17.43k)	22	(30.35k)	39	(62.50k)
	meter	0	(0)	0	(0)	2	(2.840k)	2	(2.840k)
	bicycle terminal	10	(1.736k)	0	(0)	13	(6.451k)	23	(8.187k)
	bicycle rack	8	(774)	0	(0)	14	(8.665k)	22	(9.439k)
	statue	2	(4.158k)	0	(0)	0	(0)	2	(4.158k)
	barrier	45	(83.94k)	5	(9.286k)	7	(56.10k)	57	(149.3k)
	roasting	6	(831.0k)	1	(20.82k)	4	(1.677M)	11	(2.529M)
	wire	34	(14.18k)	8	(9.325k)	0	(0)	42	(23.51k)
	low wall	58	(840.2k)	2	(26.99k)	9	(951.4k)	69	(1.819M)
	shelter	0	(0)	0	(0)	3	(83.45k)	3	(83.45k)
	bench	5	(2.911k)	1	(364)	2	(1.391k)	8	(4.666k)
	distribution box	19	(50.28k)	8	(14.89k)	3	(53.4k)	30	(118.2k)
	lighting console	78	(7.251k)	60	(9.731k)	9	(4.393k)	147	(21.38k)
	windmill	1	(10.17k)	0	(0)	0	(0)	1	(10.17k)
dynamic	pedestrian	17	(24.81k)	7	(11.60k)	61	(150.2k)	85	(186.6k)
	parked bicycle	15	(9.74k)	0	(0)	33	(81.67k)	48	(90.75k)
	mobile scooter	0	(0)	0	(0)	1	(131)	1	(131)
	parked scooter	0	(0)	0	(0)	31	(169.1k)	31	(169.1k)
	mobile motorbike	0	(0)	0	(0)	1	(1.613k)	1	(1.613k)
	parked motorbike	2	(2.428k)	0	(0)	4	(14.37k)	6	(16.79k)
	mobile car	21	(175.0k)	4	(40.96k)	5	(66.35k)	30	(282.3k)
	stopped car	0	(0)	1	(28.27k)	1	(9.375k)	2	(37.64k)
	parked car	182	(2.266M)	47	(853.7k)	65	(1.610M)	294	(4.730M)
	mobile van	3	(97.27k)	1	(41.6k)	0	(0)	4	(138.3k)
	parked van	5	(84.75k)	5	(85.20k)	9	(357.6k)	19	(527.6k)
	stopped truck	0	(0)	0	(0)	1	(235.7k)	1	(235.7k)
	parked truck	2	(40.32k)	0	(0)	1	(53.44k)	3	(93.76k)
	stopped bus	0	(0)	0	(0)	1	(78.41k)	1	(78.41k)
	parked bus	1	(9.623k)	0	(0)	0	(0)	1	(9.623k)
	table	0	(0)	0	(0)	2	(576)	2	(576)
	chair	0	(0)	0	(0)	8	(4.842k)	8	(4.842k)
	trash can	138	(148.8k)	80	(115.9k)	0	(0)	218	(264.7k)
	waste	5	(2.307k)	0	(0)	0	(0)	5	(2.307k)
natural	natural	149	(1.233M)	47	(396.9k)	36	(1.610M)	232	(3.240M)
	tree	72	(2.310M)	23	(565.10k)	101	(4.755M)	196	(7.631M)
	potted plant	32	(72.80k)	5	(26.96k)	0	(0)	37	(99.76k)
	Total	1349	(71.36M)	501	(26.84M)	629	(45.80M)	2479	(143.10M)

TABLEAU 2.1 – Répartition des points entres classes et instances d'objets sur le jeu de données Paris-Lille-3D

Nous proposons trois méthodes qui permettent de résoudre ces problèmes :

— La première est inspirée de ce qui se fait dans la segmentation sémantique (classification par points) de scènes de nuages de points par apprentissage « non-profond ». [Martin WEINMANN et collab., 2015] propose de choisir un certain nombre N de points par classe (par exemple N = 1000) aléatoirement dans le jeu d'entraînement et de ne faire l'apprentissage que sur ces points. La façon naïve d'adapter cette méthode à l'apprentissage « par *epochs* » serait d'effectivement choisir aléatoirement N points par classes au début de l'entraînement et de garder ces mêmes points à chaque *epoch*.

  Une autre façon est de choisir un nouveau jeu de N points dans chaque classe aléatoirement au début de chaque *epoch*.

  Ces deux approches ont l'avantage d'équilibrer le nombre de points par classe vu par le réseau. Le première permet un gain en temps de calcul plus élevé puisque les voisinages

peuvent n'être calculés qu'une seule fois au tout début de l'apprentissage, alors que pour la deuxième ils doivent être calculés à la volée. Cependant ceci peut être fait efficacement grâce à des structures de recherche de voisinage (KdTree ou Octree) qui n'ont besoin d'être calculés qu'une fois au début de l'entraînement, et ces calculs (généralement sur CPU) peuvent être faits en parallèle de la back-propagation des gradients (qui se fait sur GPU) du batch précédent. Ainsi on n'utilisera que cette dernière méthode par tirage d'un nouveau jeu de points d'entrainement au début de chaque *epoch*.

— La seconde méthode est inspirée de l'active learning/core-set selection [SENER et SAVARESE, 2018] ; [S.-J. HUANG et collab., 2018]. Le principe de base est toujours de passer sur N points par classe pendant chaque *epoch*, mais cette fois les points sont choisis comme étant ceux que le réseau a le plus de mal à classifier. En pratique on prend aléatoirement kN points dans chaque classe (k est un paramètre à choisir, qui peut dépendre de la classe), pour lesquels on fait l'inférence et on mesure l'erreur de classification, pour l'*epoch* on ne garde que les N points avec la plus grosse erreur.

— La troisième méthode est en fait hybride entre les deux approches précédentes, il s'agit de renouveler seulement M(< N) points par classes et de garder les N − M points pour lesquels le réseau a fait la plus mauvaise prédiction avant la back-propagation des gradients. On a alors un renouvellement régulier des points qui permet au fur et à mesure au réseau de voir tous les points du jeu d'entraînement et un mécanisme qui permet de revoir plusieurs fois les points qui posent problème.

Ces méthodes sont appelées par la suite : « Aléatoire », « Active » et « Hybride ». Et elles sont comparées entre elles en section 2.5.1 page 69.

La méthode dite « Aléatoire » a été publiée et est celle utilisée dans [ROYNARD, J. DESCHAUD et collab., 2018a] avec N = 1000.

2.3.2 Réseau de Neurones Convolutionnels Multi-échelles

2.3.2.1 Grille voxelique multi-échelles

Pour la classification d'un unique point d'une scène, on a besoin d'une représentation de son voisinage, on choisit ici des voisinages représentés sous forme de grille d'occupation voxelique [1].

Notre but n'étant pas de trouver la meilleure représentation des nuages de points sous forme de grille voxelique, on ne travaillera qu'avec des grilles d'occupation voxeliques, chaque voxel contenant soit un 0 si le voxel est vide, soit un 1 s'il est occupé par au moins un point. Il existe de nombreuses autres façons de remplir une grille voxelique comme une probabilité, une couleur, une réflectance, une densité [MATURANA et SCHERER, 2015], mais le choix de cette représentation n'a pas d'effet sur la conception des méthodes qui vont suivre.

La création de voisinages de grille de voxels multi-échelles consiste à représenter nos voisinages par plusieurs grilles d'occupations créées avec des pas de discrétisation différents et/ou des tailles différentes.

Par la suite on se restreindra au cas de grilles de la même taille [2], et à des pas de discrétisation de la forme $2^n\delta$ où δ est le plus petit pas de discrétisation utilisé.

Par exemple l'image de gauche de la figure 2.10 page 58 montre la subdivision de l'espace (ici en 2D) par grille voxelique multi-échelle autour d'un point central. Ici chaque échelle est deux fois plus grossière que l'échelle plus résolue sous-jacente.

Pour créer ce voisinage multi-échelle sous forme de grille d'occupation, on cherche tous les points de la scène qui sont à une distance inférieure à $\frac{1}{2}\sqrt{3}\cdot 32\cdot 2^{\mathrm{N}_{max}}\cdot\delta$ si la grille est de taille $32\times32\times32$, $2^{\mathrm{N}_{max}}\cdot\delta$ étant le plus grand pas de discrétisation utilisé .

1. c'est une représentation très simple et structurée qui convient très bien aux réseaux de neurones convolutionnels actuels

2. généralement $32\times32\times32$ pour des questions de mémoire et temps de calcul, bien qu'on aimerait avoir des grilles plus résolues.

En pratique, ceci peut prendre beaucoup de temps sur des jeux de données de plusieurs centaines de millions de points [3] et avec des échelles importantes. Une alternative est de sous-échantillonner les nuages du jeu d'entraînement a priori, et de faire les recherches de voisinages dans des nuages échantillonnés différemment pour chaque échelle.

On remarquera qu'ici « multi-échelles » est à nuancer du terme utilisé en images où cela signifie « utiliser la même image à plusieurs résolutions différentes ». Ici cela signifie « prendre des voisinages de tailles différentes », en particulier les plus grands voisinages (les plus grossiers) apportent de l'information qui n'est pas présente dans les plus petits (les plus fins), alors que les images les plus résolues (les plus fines) contiennent toute l'information disponible, on devrait alors parler de multi-résolution.

L'intérêt du multi-échelles dans notre cas est donc tout autre du cas des images :

— Pour les images, le traitement d'une même image peut avoir plusieurs intérêts comme accélérer les calculs en traitant d'abord les moins résolues (plus petites donc plus rapides à traiter) puis capitaliser ces informations pour faire moins de calculs à plus haute résolution [ZHAO et collab., 2017]. Cela peut également permettre de traiter l'information de façon hiérarchique.

— Dans notre cas, les voisinages les plus grands (les plus grossiers) vont chercher de l'information plus loin, par exemple s'il y a un manque de contexte dans les plus petits voisinages, ceci ne peut pas être résolu par des grilles plus grandes (par exemple $64 \times 64 \times 64$ à cause du manque de mémoire et de puissance de calcul).

Ayant des grilles voxeliques de mêmes tailles (prenons par exemple ici $N_{in} \times 32 \times 32 \times 32$ où N_{in} est le nombre de canaux d'entrée) mais dont les voxels représentent des portions d'espace avec des pas de discrétisation Δ différents (par exemple 5 cm, 10 cm et 20 cm), on se pose ici la question de la façon de fusionner ces grilles dans un réseau convolutionnel.

2.3.2.2 Architecture multi-échelles par fusion précoce

L'approche la plus intuitive (qu'on appelera « fusion-précoce » ou « *early-fusion* » en anglais) consiste à voir chaque échelle comme un canal d'entrée (de la même façon que chaque couleur est un canal d'entrée pour les images RVB), il suffit alors de concaténer les grilles voxeliques dans l'espace des caractéristiques pour obtenir une seule grosse grille voxelique avec $N_{in} * N_{sc}$ canaux d'entrée, où N_{sc} est le nombre d'échelles. L'architecture du réseau n'est donc pas changée par rapport à un réseau mono-échelle, seul le nombre de canaux d'entrée est augmenté. Cette méthode de fusion est décrite en image de gauche de la figure 2.13 page 61.

2.3.2.3 Architectures multi-échelles par fusion tardive

Une approche un peu moins naïve (qu'on appelera « fusion-tardive » ou « late-fusion » en anglais) consiste à appliquer la partie convolutionnelle du réseau à chaque échelle séparément, puis à concaténer les vecteurs de caractéristiques obtenues avant de donner le tout à un MLP. L'architecture du réseau est légèrement modifiée, et on peut ici imaginer deux variantes possibles : avec les poids partagés entre les parties convolutionnelles des différentes échelles, ou bien avec des poids indépendants. La version utilisée dans [ROYNARD, J. DESCHAUD et collab., 2018a] est la fusion-tardive avec des poids indépendants entre échelles. Cette méthode de fusion est décrite en image centrale de la figure 2.13 page 61.

2.3.2.4 Défauts des fusions précoce et tardive

Ces deux approches par fusions précoce ou tardive ont l'inconvénient de ne pas prendre en compte la cohérence spatiale des données (bien que ça soit moins marqué pour la fusion tardive).

3. même avec des structures adaptées à la recherche de voisinages comme les *KdTree*.

En effet, si on s'intéresse à un vecteur de caractéristiques à une position 3D donnée, on a un vecteur qui mélange des caractéristiques à plusieurs échelles/tailles de voxels et à des positions qui ne sont pas les mêmes à chaque échelle, voir figure 2.10.

Or les différentes convolutions sont des filtres spatiaux qui mélangent les caractéristiques (en profondeur), et donc vont mélanger de l'information incohérente spatialement.

Cette incohérence peut être mitigée pour la fusion tardive, puisque, au moment de la fusion des caractéristiques aux différentes échelles, l'unique voxel restant représente toute la grille initiale qui est certes de taille différente à chaque échelle mais est centrée et à la même position pour chaque échelle.

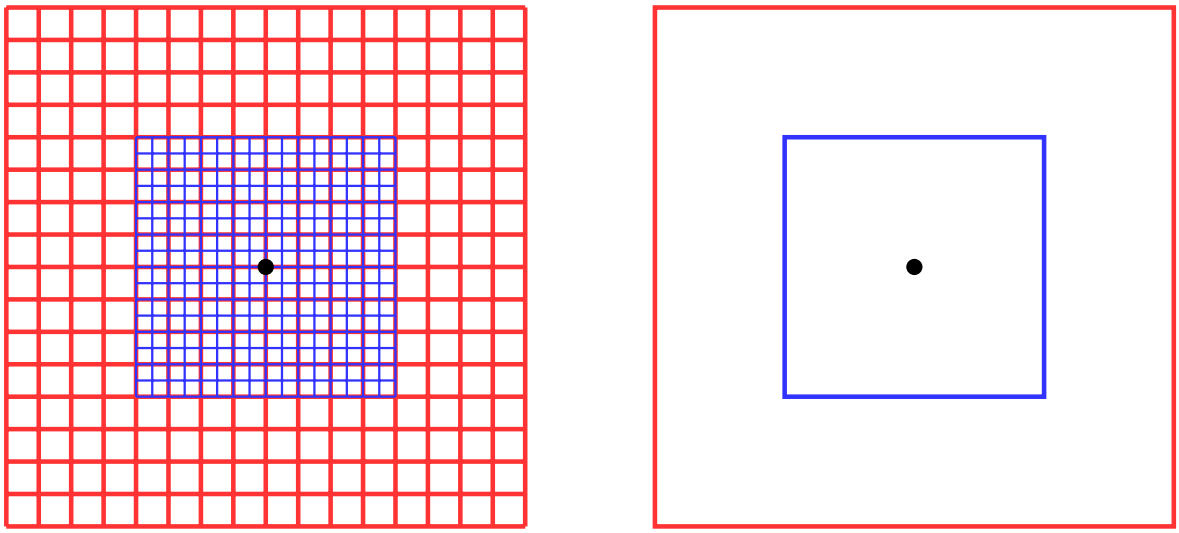


FIGURE 2.10 – Les grilles fusionnées par fusion précoce (à gauche) ou par fusion tardive (à droite). Le • représente le point autour duquel le voisinage est pris. Dans les deux cas, la grille bleue représente une échelle deux fois plus petite que la grille rouge. Pour la fusion tardive les grilles n'ont plus qu'un seul voxel puisqu'elle sont le résultat de la partie convolutionnel du réseau qui agrège spatialement l'information. Ces 2 schémas montrent quelle information est agrégée par le réseau. À gauche, pour la fusion précoce, le voxel en haut à gauche de la grille rouge est fusionné avec celui en haut à gauche de la grille bleue (or ces pixels ne représentent pas une information sur un volume identique, en position et en taille). Et à droite pour la fusion tardive c'est l'information agrégée (par la partie convolutionnelle du réseau) sur les voxels rouges et bleus qui est fusionnée, ici les voxels sont biens centrés à la même position, mais n'ont toujours pas la même taille.

2.3.2.5 Architectures multi-échelles par fusion cohérente

On cherche ici à concevoir des architectures multi-échelles réalisant une fusion cohérente spatialement. C'est-à-dire qu'on ne s'autorise à fusionner deux échelles qu'à une étape où les voxels ont la même taille et sont positionnés au même endroit. Par exemple si on a en entrée deux grilles $32\times32\times32$ à des échelles 5 cm (dite fine) et 10cm (dite grossière) centrées autour du même point et orientées de la même façon, on ne peut pas les fusionner en l'état. Par contre après une opération (qu'on appellera *Downsampling*) de *MaxPooling* ou de *Strided Convolution* (avec stride 2) sur la grille fine (maintenant de taille $16\times16\times16$) il y a « cohérence » entre les voxels de la grille fine et les voxels centraux de la grille grossière (les voxels d'indices $[8:23\times8:23\times8:23]$), voir figure 2.11 page suivante.

On aimerait donc fusionner à ce moment là, c'est-à-dire concaténer dans l'espace des caractéristiques ou sommer les grilles, mais en l'état cela n'est pas possible puisque les grilles ne font pas la même taille. Deux solutions se présentent, voir figure 2.12a page 60 :

— pour l'échelle fine choisir une taille de grille double de celle de l'échelle grossière (cf figure 2.12a page 60),

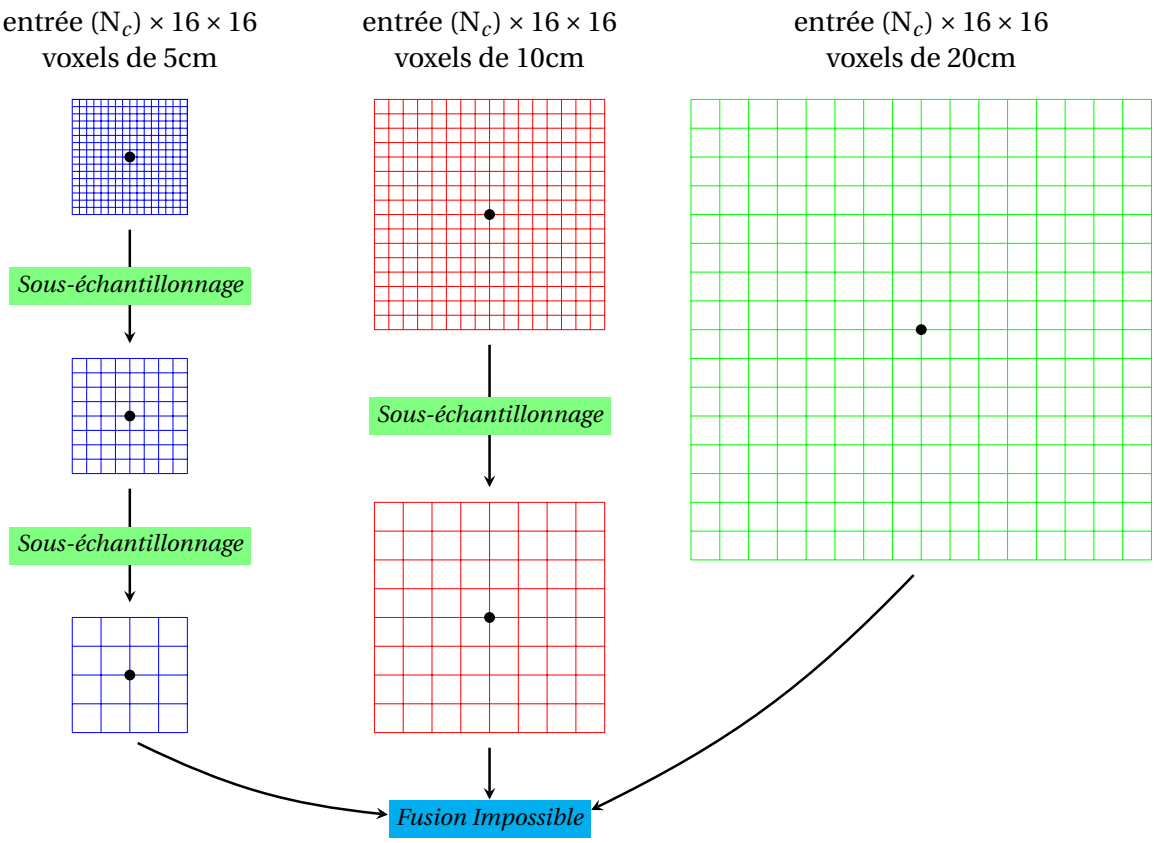


FIGURE 2.11 – L'image montre la cohérence entre les voxels de 2 échelles après *Downsampling* sur une échelle fine. Mais cela ne semble pas possible de fusionner naïvement par sommation ou concaténation puisque les grilles sont de tailles différentes.

— augmenter artificiellement la taille de la grille à l'échelle fine après l'opération de *Downsampling*, par exemple par *Zero-Padding* (cf figure 2.12b page suivante). On appelle cette méthode « fusion cohérente ».

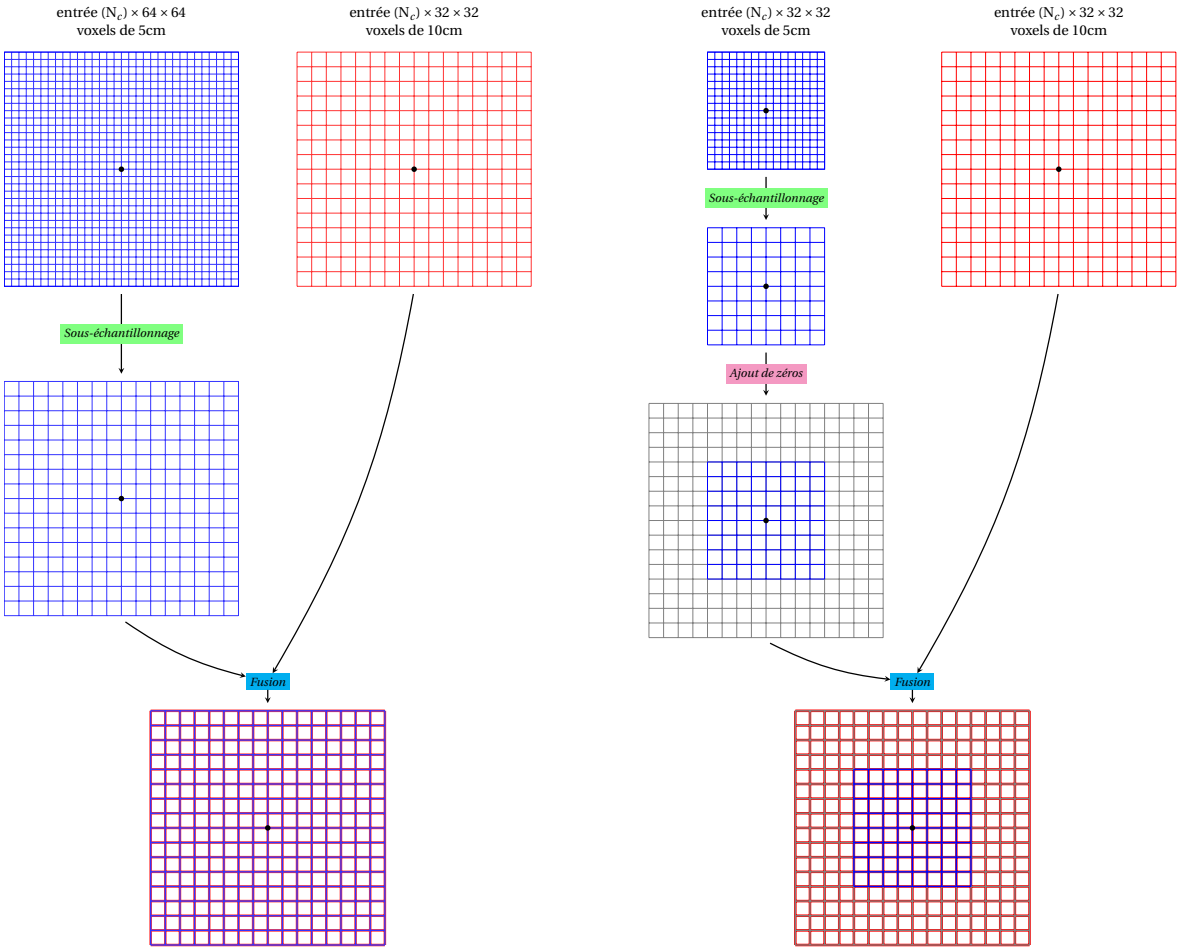
Cependant, la première approche n'est pas envisageable en 3D puisque doubler la taille de la grille selon chaque dimension revient à multiplier par 8 la place prise en mémoire et le nombre d'opérations à réaliser, ce qui n'est pas possible avec des grilles de taille supérieure à $32 \times 32 \times 32$ avec les GPUs actuels.

Une variante moins naïve est de traiter différemment les voxels centraux de ceux de la couronne pour l'échelle grossière. On peut alors fusionner les voxels centraux de l'échelle grossière avec la grille de l'échelle fine après *Downsampling*, sur cette grille fusionnée ou applique une convolution $1 \times 1 \times 1$ pour se ramener au même nombre de cartes de caractéristiques que sur la couronne. En parallèle on effectue également une convolution $1 \times 1 \times 1$ sur la couronne, avant de réassembler la grille complète. Cette variante n'a pas encore été implémentée et évaluée.

Le seul inconvénient qui ressort de cette approche par fusion de grilles cohérentes spatialement est quelle impose un choix restreint de tailles de voxels aux différentes échelles. En effet, si la plus petite échelle est S_{min} toutes les autres échelles doivent être de la forme $2^n \cdot S_{min}$ avec $n \in \mathbb{N}$. Par exemple, on peut choisir [5cm, 10cm, 20cm] mais pas [5cm, 10cm, 15cm].

Quelle que soit la méthode de fusion utilisée, pour construire un réseau complet, il s'agit alors, en partant de l'échelle la plus fine, d'incorporer progressivement les grilles à des échelles de plus en plus grossière.

Une fois la grille la plus grossière fusionnée, il suffit d'appliquer le schéma classique d'agrégation des données par convolutions et *Downsampling*. Pour une description du réseau avec les échelles [5cm, 10cm, 20cm], voir la figure 2.13 page 61. On utilise ici une représentation similaire à celle utilisée pour les *Convolutional Neural Fabrics* [SAXENA et VERBEEK, 2016] décrite en an-



(a) Fusion après *Downsampling* sur une grille fine de définition doublée.

(b) Fusion après *Downsampling* puis ajout de zéros sur une grille fine de résolution doublée. Appelée fusion cohérente.

FIGURE 2.12 – Solutions pour fusionner 2 échelles après *Downsampling* sur l'échelle la plus fine, à gauche avec une définition plus grande pour l'échelle fine et à droite par *Zero-Padding*

nexe A.6.4 page 143.

Grâce à la représentation sous forme de *Convolutional Neural Fabrics*, on peut envisager des architectures encore plus complexes, avec un flux d'information connecté à plusieurs autres couches à chaque échelle. Similaire au réseau décrit au milieu de la figure 2.13 page suivante, mais au lieu d'une fusion en fin de réseau, la fusion se ferait à de multiples endroits grâce à des fusions cohérentes (flèches vers la gauche) à chaque échelle. Nous n'avons pas eu le temps d'investiguer plus dans cette voie.

2.3.3 Réseaux Multi-échelles pour la Segmentation Sémantique

On s'intéresse ici à la façon d'intégrer un réseau multi-échelles dans une architecture de segmentation sémantique du type *SegNet* ou *U-Net* (pour rappel sur leurs architectures voir figure 2.14 page ci-contre et l'annexe A.6.3 page 141). Ces méthodes n'ayant pas encore été implémentées, nous ne présenterons aucune expérimentations les évaluant.

A priori les flux d'information des échantillons de différentes échelles ne se connectent pas, comme sur l'image de gauche de la figure 2.15 page 62 sans la fusion des grilles 1×1.

On propose d'adapter les méthodes de fusion tardive et cohérente déjà étudiées (la fusion précoce n'ayant pas de sens ici en sortie du réseau) de la façon la plus simple possible :

— Pour la fusion tardive, comme précédemment on fusionne les sorties de la partie décrois-

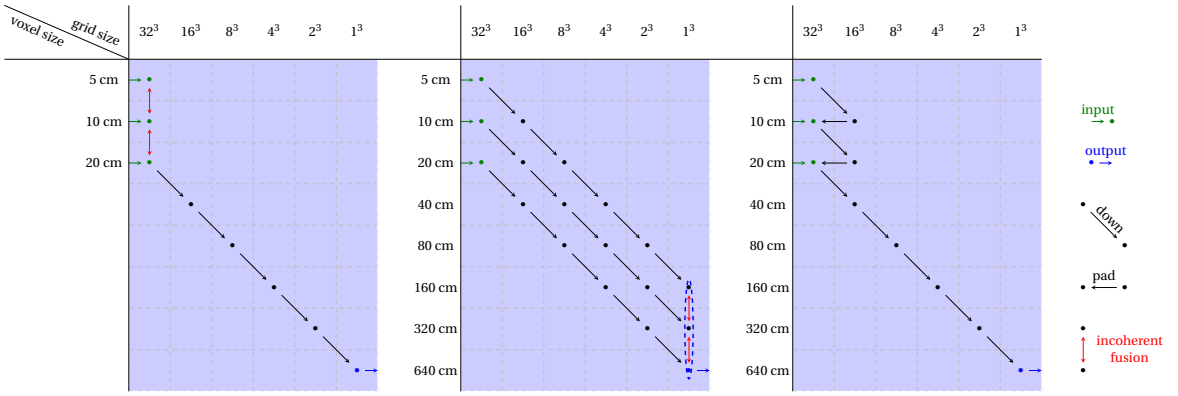


FIGURE 2.13 – Comparaison des réseaux de classification multi-échelle. À gauche la fusion précoce, au milieu la fusion tardive et à droite une possibilité de fusion cohérente. Chaque • peut être accompagné de convolution(s) qui conservent la dimension des couches de caractéristiques. Pour la fusion précoce, les échelles de taille de voxel ne sont plus indiquées après fusion car elles n'auraient aucun sens.

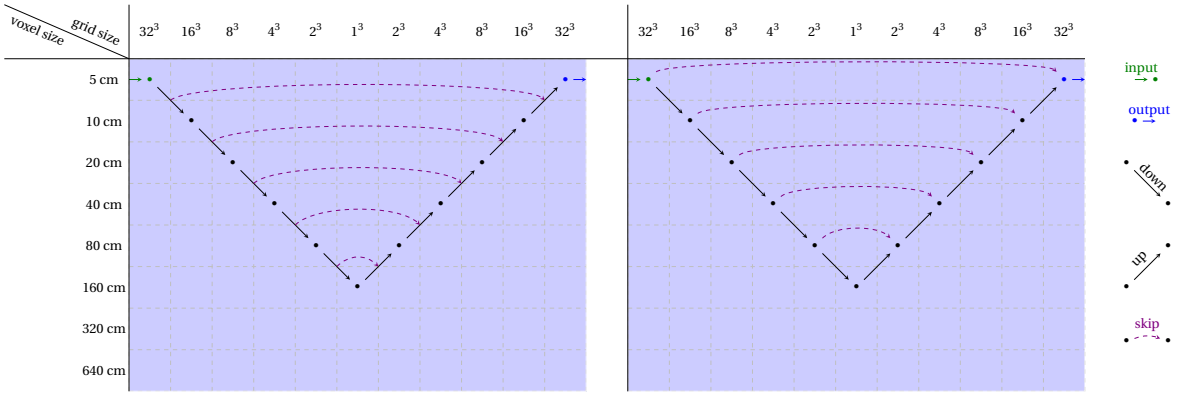


FIGURE 2.14 – Rappel des architectures des résaux *SegNet* (à gauche) et *U-Net* (à droite) présentés plus en détails en annexe A.6.3 page 141. C'est la nature des *skip connections* qui diffère entre les deux architectures. Chaque • peut être accompagné de convolution(s) qui conservent la dimension des cartes de caractéristiques.

sante du réseau (partie convolutionnelle), puis on donne à chaque partie croissante (déconvolutionnelle) la même entrée : le produit de la fusion.

— Pour la fusion cohérente, comme précédemment on fusionne tous les flux en un seul flux décroissant, puis en un seul flux croissant. À la sortie, le résultat est produit des couches les plus grossières vers les plus fines, et pour ne jamais dépasser la taille de grille initiale on utilise l'opération inverse du *Zero-Padding* : le rognage (*crop*). C'est en fait très similaire à un réseau classique (mono-échelle) de segmentation, toute la partie multi-échelle est gérée intégralement à l'entrée et à la sortie.

Ces deux architectures sont décrites plus en détails en figure 2.15 page suivante.

Nous n'avons pas eu le temps d'investiguer plus en profondeur cette voie, en particulier il serait intéressant d'étudier l'endroit où les fusions apportent le plus de gain par rapport à une architecture mono-échelle, ou bien de montrer que toutes les connexions sont utiles.

Un problème qui peut être rencontré pendant l'entraînement de réseaux de segmentation sur des grilles d'occupation (occupées de façon éparses) est le risque de déséquilibrage entre les voxels annotés minoritaires et les voxels vides largement majoritaires, une solution de l'état de l'art est d'utiliser une fonction de coût spéciale comme la *Focal Loss* [LIN, GOYAL et collab., 2017] qui donne plus d'importance aux voxels annotés. Cependant cette méthode requiert de choisir encore de nouveaux hyper paramètres.

Une approche simple que l'on peut utiliser dans notre cas (pas auto-encodeur) est d'appliquer en sortie le masque des voxels occupés en entrée, ainsi la fonction de coût est calculée uniquement

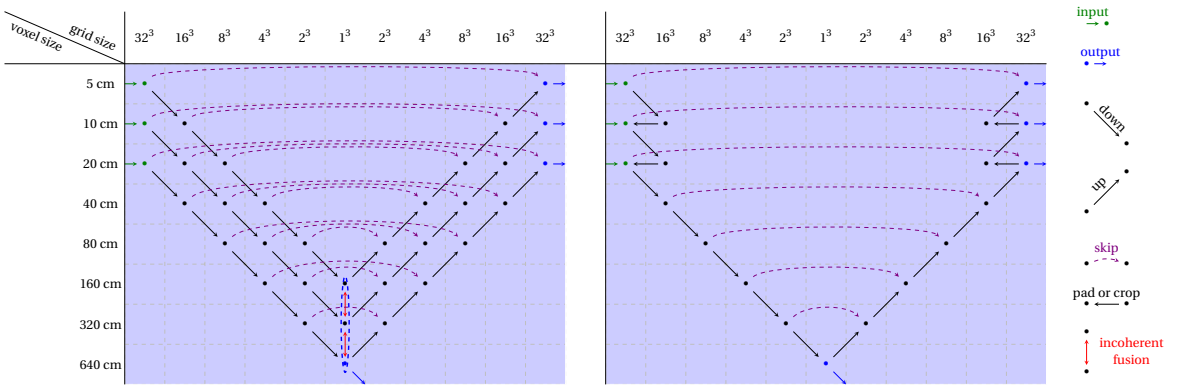


FIGURE 2.15 – Proposition d'architecture de résaux de segmentation à entrée multi-échelles, ici type *U-Net*. À gauche avec fusion tardive, et à droite avec fusion cohérente à l'entrée du réseau. Les flèches horizontales représentent du *padding* sur la partie décroissante du réseau et du rognage (*crop*) sur la partie croissante. Chaque • peut être accompagné de convolution(s) qui conservent la dimension des cartes de caractéristiques.

sur les voxels occupés, et les gradients ne sont propagés que par les neurones qui permettent de calculer la classe sur ces voxels.

2.4 Protocole Expérimental

Nous décrivons ici le protocole utilisé pour l'évaluation des méthodes proposées en section 2.3 page 53. L'architecture convolutionnelle de base est très semblable à celle décrite en figure 2.16, et ne change pas pour les différentes expérimentations pour pouvoir comparer les différents résultats sur la même base.

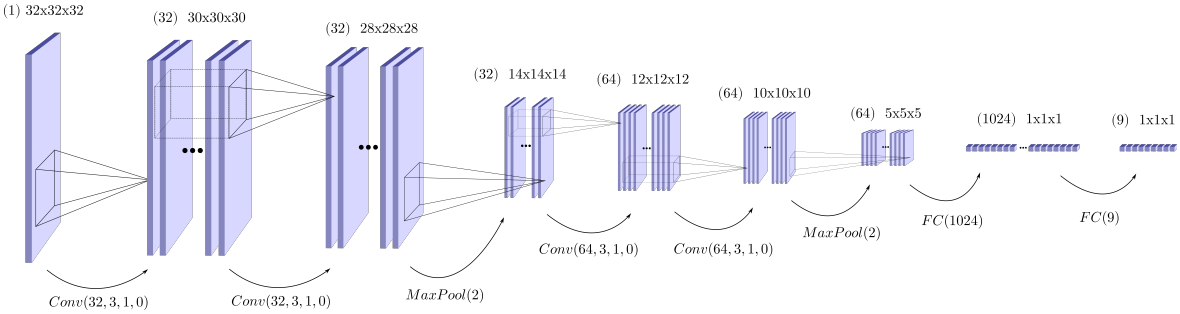


FIGURE 2.16 – L'architecture de réseau convolutionnel de base utilisée sur tous les réseaux testés (toutes les cartes de caractéristiques sont représentées en 2D au lieu de 3D pour la simplicité).

2.4.1 Construction des entrées du réseau

Une fois un échantillon (un point p à classifier) choisi, on obtient une grille de voxels à donner au réseau convolutionnel en construisant une grille d'occupation centrée sur p dont les voxels vides contiennent 0 et les voxels occupés contiennent 1. Nous n'utilisons que des grilles cubiques $n \times n \times n$ où n est pair, et nous n'utilisons que des pas de discrétisation de l'espace Δ isotrope. Donc par exemple pour $n = 32$ et $\Delta = 10$cm, on a une grille cubique de côté 3.20m et on doit chercher un voisinage de rayon $r = \sqrt{3} \times 1.60\text{m} \approx 2.77\text{m}$ autour de p. De plus, pour un nuage de points qui contient également des informations par points (couleur, réflectance...), cette grille d'occupation peut avoir des canaux supplémentaires comme la réflectance (les voxels vides contiennent toujours 0 et les voxels occupés contiennent une moyenne de la réflectance sur tous les points qui tombent dans le voxel).

2.4.2 Augmentation de données

On réalise quelques procédures d'augmentation de données classiques sur le nuage de points 3D avant de le projeter dans une grille d'occupation :

— Échanger les axes x et y, avec une probabilité de 0.5

— Effectuer une rotation aléatoire autour de l'axe vertical

— Changer l'échelle, entre 95% et 105%,

— Ajouter des occlusions aléatoires (aléatoirement retirer des points), jusqu'à 5% de suppression,

— Ajouter des artefacts aléatoires (aléatoirement ajouter des points), jusqu'à 5% d'ajout,

— Ajouter un bruit à la position des points, le bruit suit une distribution normale centrée en 0 avec un écart-type de 0.01m

Pour un nuage de point d'entraînement qui contient également des informations par points (couleur, réflectance...), on peut également faire de l'augmentation sur ces informations :

— en ajoutant du bruit

— en éteignant aléatoirement une entrée, par exemple si la réflectance est présente on peut la binariser (0 quand le voxel est vide, 1 quand il contient un point), pour forcer le réseau à apprendre également une information géométrique

2.4.3 Entraînement

Les paramètres d'entraînement sont les suivants :

— fonction de coût : entropie croisée, qui combine un *LogSoftMax* et une fonction de coût *Negative Likelyhood*,

— optimiseur : ADAM avec taux d'apprentissage de 0.001 et $\epsilon = 1e-8$, qui sont les paramètres par défaut dans la plupart des bibliothèques d'apprentissage profond,

— une décroissance exponentielle du taux d'apprentissage, pour arriver à un taux d'apprentissage final de 10^{-6},

— nombre d'*epochs* : 100,

— nombre de points (renouvelés aléatoirement avant chaque *epoch*) par classe : 100,

Ces paramètres sont assez classiques pour l'entraînement de réseaux convolutionnels, et on a remarqué qu'ils donnaient des résultats satisfaisants, ils seront donc utilisés pour la plupart des expérimentations (sauf indication contraire).

2.4.4 Inférence d'une scène complète

Pour labelliser une scène de nuage de points complète, la méthode naïve est de parcourir tous les points du nuage, et pour chaque point de :

— chercher tous les points voisins qui tombent dans la grille d'occupation,

— créer la grille d'occupation,

— inférer la classe du point grâce au réseau convolutionnel pré-entraîné.

Or deux points très proches l'un de l'autre vont avoir la même grille d'occupation du voisinage et donc le réseau va prédire la même classe. Une méthode plus rapide de test est donc de sous-échantillonner le nuage à tester, ce qui a deux effets bénéfiques :

— réduit le nombre d'inférences et de recherches de voisinage,

— chaque recherche de voisinage prend moins de temps,

Pour inférer la classe des points du nuage initial, il suffit de donner à chaque point la classe du point le plus proche dans le nuage sous-échantillonné. Si le sous-échantillonnage est fait grâce à une grille 3D, on peut également utiliser une interpolation trilinéaire des *probabilités* (sorties de la dernière couche *SoftMax*) obtenues sur les 8 sommets du voxel auquel appartient le point comme réalisé par [TCHAPMI et collab., 2017], mais cette méthode ne joue que très peu sur les résultats.

Cette méthode permettra de classifier une scène complète grâce à un réseau de classification, mais les temps de calculs sont encore considérables (de l'ordre d'une journée pour un nuage de 10M points sur un ordinateur de bureau haut de gamme).

Un moyen d'accélérer l'inférence d'une scène complète est d'utiliser des réseaux de segmentation sémantique qui classifient l'ensemble des voxels de la grille d'occupation donnée en entrée (ou de façon équivalente l'ensemble des points qui ont permis de construire cette grille d'occupation). On n'a alors pas besoin de construire une grille d'occupation de voisinage pour chaque point et d'y appliquer une réseau convolutionnel.

Pour des méthodes plus évoluées et plus efficaces d'inférence d'un réseau convolutionnel sur une scène complète, le lecteur est renvoyé au chapitre 3 page 85 et plus particulièrement à la section 3.3.1 page 97.

2.4.5 Métriques et protocole d'évaluation

Les métriques utilisées pour évaluer les performances des nos différents réseaux et méthodes d'entraînement sont les suivantes :

$$
\begin{aligned}
\mathrm{P}_c &= \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c} \\
\mathrm{R}_c &= \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c} \\
\mathrm{F1}_c &= \frac{2\mathrm{TP}_c}{2\mathrm{TP}_c + \mathrm{FP}_c + \mathrm{FN}_c} = 2\frac{\mathrm{P}_c \mathrm{R}_c}{\mathrm{P}_c + \mathrm{R}_c} \\
\mathrm{Acc}_c &= \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c} \\
\mathrm{IoU}_c &= \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c + \mathrm{FN}_c}
\end{aligned}
$$

Où P_c, R_c, $\mathrm{F1}_c$, Acc_c et IoU_c représentent respectivement la Précision, le Rappel, le F1-score, l'Accuracy (qui se traduit en français par Précision) et l'Indice de Jaccard (noté IoU pour *Intersection-over-Union* en anglais) de la classe c.

Et TP_c, TN_c, FP_c et FN_c sont respectivement le nombre de Vrais-Positifs, Vrais-Negatifs, Faux-Positifs et Faux-Negatifs de la classe c.

Chaque dataset sur lequel on teste nos méthodes est découpé en plusieurs nuages ou parties (*folds* en anglais) pour réaliser une évaluation par validation croisée. Consécutivement, on prend chaque nuage comme jeu de validation (ou de test) et tous les autres comme jeu d'entraînement. À la fin de ce processus, on moyenne les métriques de validation sur l'ensemble des nuages, pour obtenir des métriques globales sur le jeu de données en question et la méthode testée.

Pour augmenter la fiabilité des résultats obtenus et réduire la dépendance aux différents tirages aléatoires (des points d'entraînement au début de chaque *epoch*, poids initiaux des réseaux) apparaissant pendant l'entraînement, on réalise ce processus plusieurs fois (ici 10 fois) avant de moyenner les résultats.

Comme cela prendrait trop de temps de mesurer l'erreur de validation sur l'ensemble du nuage de validation à chaque *epoch*, au début de l'entraînement, on choisit aléatoirement N points dans chaque classe sur le nuage de validation. Ces points permettent de suivre les différentes métriques au cours de l'entraînement. N est quand même choisi assez grand (supérieur au nombre de points choisis par classe sur le jeu d'entraînement) pour obtenir des métriques fiables.

De plus quand on compare différentes méthodes sur un même jeu de données, on s'assure que les points de validation choisis dans le nuage de validation sont les mêmes.

2.4.6 Jeux de données d'évaluation

Pour réaliser nos expérimentations nous avons choisi les 3 jeux de données de scènes 3D qui nous semblent les plus pertinents pour entraîner des méthodes d'apprentissage profond :

— le jeu de données présenté en chapitre 1 page 5 : Paris-Lille-3D [ROYNARD, J. DESCHAUD et collab., 2017] ,

— Semantic3D [HACKEL, SAVINOV et collab., 2017].

— S3DIS [ARMENI, SENER et collab., 2016],

Parmi les jeux de données de scènes 3D, ce sont ceux couvrant le plus de surface et possédant le plus de variabilité, ces données sont résumées dans le tableau 2.2 page suivante. La surface couverte est obtenue en projetant chaque nuage sur un plan horizontal dans des pixels de taille 10cm × 10cm, puis en sommant la surface de tous les pixels occupés

Pour confirmer l'intérêt des méthodes développée en section 2.3 page 53, nous appliquons la méthodologie de validation croisée décrite en section 2.4.5 sur les trois jeux de données Paris-Lille-3D, Semantic3D et S3DIS. De plus sur le challenge *reduced-8* de Semantic3D et S3DIS nous

comparons certains résultats avec ceux de la littérature. On décrit ici rapidement ces 3 jeux de données, pour plus de détails sur ces jeux de données le lecteur est renvoyé au chapitre 1 page 5.

Nom	type de LiDAR	Aire couverte	Nombre de points (sous-échantillonné)	Nombre de classes
Paris-Lille-3D	MLS multi-fibres	55000 m^2	143.1 M (44.0 M)	9
Semantic3D	LiDAR static	110000 m^2	1660 M (79.5 M)	8
S3DIS	MatterPort	6020 m^2	695.9 M (36.9 M)	13

TABLEAU 2.2 – Comparaison des jeux de données de nuages de points 3D. Paris-Lille-3D contient 50 classes mais pour nos expérimentations nous gardons seulement 9 classes plus grossières. Entre parenthèses est indiqué le nombre de points après sous-échantillonnage à 2 cm.

Les figures 2.17, 2.18 page ci-contre et 2.19 page suivante montrent respectivement des exemples de nuages de points des jeux de données Paris-Lille-3D, Semantic3D et S3DIS.

FIGURE 2.17 – Exemple de nuage annoté du jeu de données Paris-Lille-3D.

2.4.6.1 Paris-Lille-3D

Le jeu de données Paris-Lille-3D a déjà été décrit dans le chapitre 1 page 5 mais nous en rappelons ici les principales caractéristiques. Il est constitué de 2km de nuages de points 3D acquis par *Mobile Laser Scanning* (MLS) grâce à un Velodyne HDL-32e monté sur une camionnette. Les nuages sont géo-référencés uniquement grâce à une IMU et un GPS-RTK, aucune méthode de recalage ou de SLAM n'est utilisée ce qui engendre un léger bruit. Comme le balayage de la scène se fait à-peu-près à vitesse constante, la densité de points est à-peu-près régulière. Le jeu de données Paris-Lille-3D est constitué de 3 fichiers, un acquis à Paris et deux acquis à Lille dont $Lille1.ply$ beaucoup plus gros que $Lille2.ply$. Pour faire la validation de nos architectures par méthode de K-*fold*, on découpe donc $Lille1.ply$ en deux *folds* contenant le même nombre de points. De plus, ce jeu de données contient 50 classes, dont certaines qui n'apparaissent que dans certains *folds* et avec très peu de points. Nous décidons donc de supprimer et regrouper certaines classes pour ne garder que 9 classes plus grossières :

sol	bâtiments	potelets
poteaux/signalisation	poubelles	barrières
piétons	voitures	naturel/végétation

2.4.6.2 Semantic3D

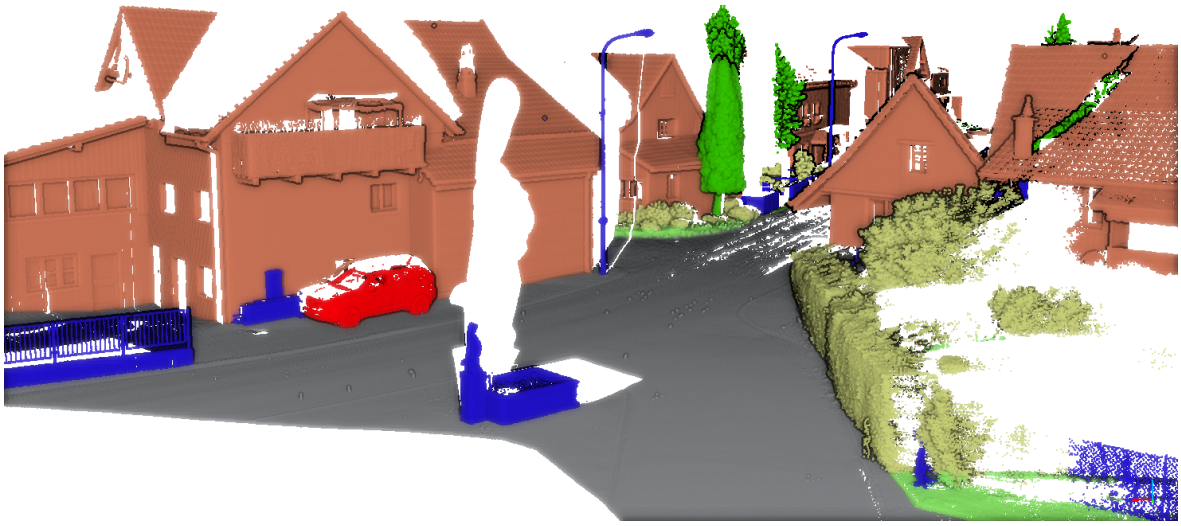


FIGURE 2.18 – Exemple de nuage annoté du jeu de données Semantic3D.

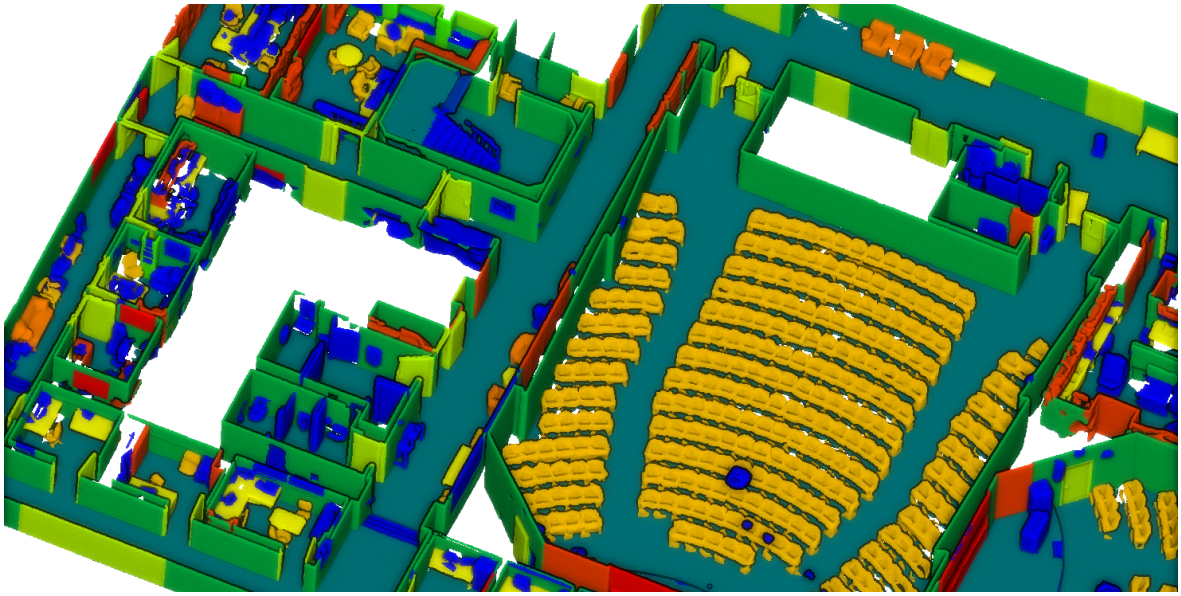


FIGURE 2.19 – Exemple de nuage annoté du jeu de données S3DIS.

Le jeu de données Semantic3D a été acquis par LiDAR statique, il est donc plus dense que les jeux de données acquis par MLS comme Paris-Lille-3D, mais la densité du nuage varie considérablement en fonction de la distance au capteur. Et il y a de nombreuses occlusions dues au fait que le capteur (étant fixe), ne peut pas tourner autour des objets pour les voir sous plusieurs points de vue. Pour atténuer un peu le problème de densité très variable, on sous-échantillonne les nuages d'entraînement à 2 cm. Ceci permet d'obtenir une densité plus uniforme au moins proche du capteur, et d'éviter d'avoir des points redondants.

2.4.6.3 S3DIS

Le jeu de données S3DIS est constitué de 6 scènes de nuages de points 3D RGB acquis dans 3 bâtiments différents et contenant 13 classes. Sur ce jeu de données les nuages sont sous-échantillonnés à 2 cm de la même façon que sur Semantic3D. Après sous-échantillonnage l'ensemble du jeu de données comporte 36.868M points.Comme dans [TCHAPMI et collab., 2017] la comparaison des résultats est faite seulement sur le *fold* 5, qui est un bâtiment différent des 5 autres *folds*.

2.4.7 Détails matériel et logiciel

Toutes les expérimentations sont réalisées sur un ordinateur de bureau haut de gamme (dit « de *gamer* ») équipé de :

— un CPU Intel® Core™ i7-4770K, avec 8 threads à 3.5GHz,

— un GPU Nvidia® GTX 1080 Ti, avec 11 Go de mémoire GDDR5X,

— 32 Go de RAM DDR4

De plus, pour l'apprentissage profond la bibliothèque `Pytorch` en langage `Python` est utilisée. Et les recherche de voisinages dans le jeux de données pour créer les échantillons pendant l'entraînement sont réalisées grâce à la bibliothèques `nanoflann` ou au *kd-tree* de la bibliothèque `scikit-learn`.

2.5 Résultats de Segmentation Sémantique

2.5.1 Étude comparative des différentes méthodes d'entraînement

On compare ici les trois méthodes d'entraînement des réseaux de neurones sur des scènes de nuages de points 3D complètement annotées. Ces méthodes sont appelées : « Aléatoire », « Hybride » et « Active », respectivement sur les graphiques *random*, *hybrid* et *active*.

Pour rappel, la méthode « Aléatoire » tire un nouveau jeu de N points d'entraînement dans chaque classe au début de chaque *epoch*. La méthode « Hybride » conserve les points pour lesquels le réseaux a fait une erreur de classification à l'*epoch* précédente, on complète alors avec des points tirés aléatoirement pour bien avoir N points par classe. La méthode « Active » tire avant chaque *epoch* $k \times$ N par classe (avec $k > 1$, typiquement $k = 10$), la classe de ces points est inférée grâce au réseau, et seuls N points par classe parmi les points où il y a eu une erreur de classification sont gardés.

Ici seule la méthode d'échantillonnage des points à chaque *epoch* varie, tous les autres paramètres sont identiques, en particulier le réseau utilisé est exactement le même, c'est un réseau de classification à une échelle similaire à celui décrit en figure 2.16 page 63

On observe logiquement et sans surprise que l'erreur d'entraînement est plus faible que l'erreur de validation pour toutes les méthodes (voir graphique 2.20 et tableau 2.3 page suivante). On observe en particulier que :

— la méthode dite « hybride », apporte deux améliorations par rapport à la méthode totalement aléatoire :
  — Une erreur de validation moins élevée (de 0.9 points),
  — Une erreur d'entraînement plus élevée (de 2.5 points) et donc plus proche de l'erreur de validation. Ceci permet d'avoir une meilleur estimation de l'erreur quand on entraîne un réseau sur l'ensemble des données (sans jeu de validation).
— étonnamment, la méthode dite « active » montre des performance très en-deçà des deux autres méthodes en validation (> 20%) alors que l'erreur de classification en entraînement est similaire (< 10%).

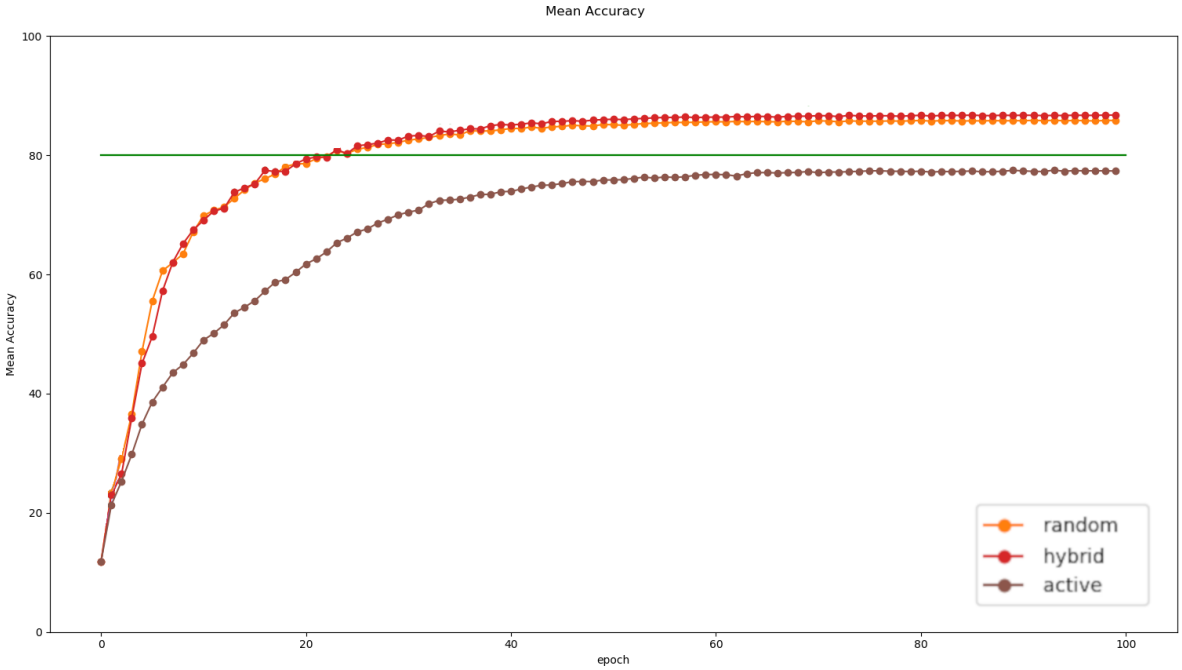


FIGURE 2.20 – Graphe d'*accuracy* en validation au cours de l'apprentissage pour les différentes méthodes de choix des échantillons pendant chaque *epoch*. La droite horizontale verte correspond au seuil de 80% d'accuracy (20% d'erreur de classification).

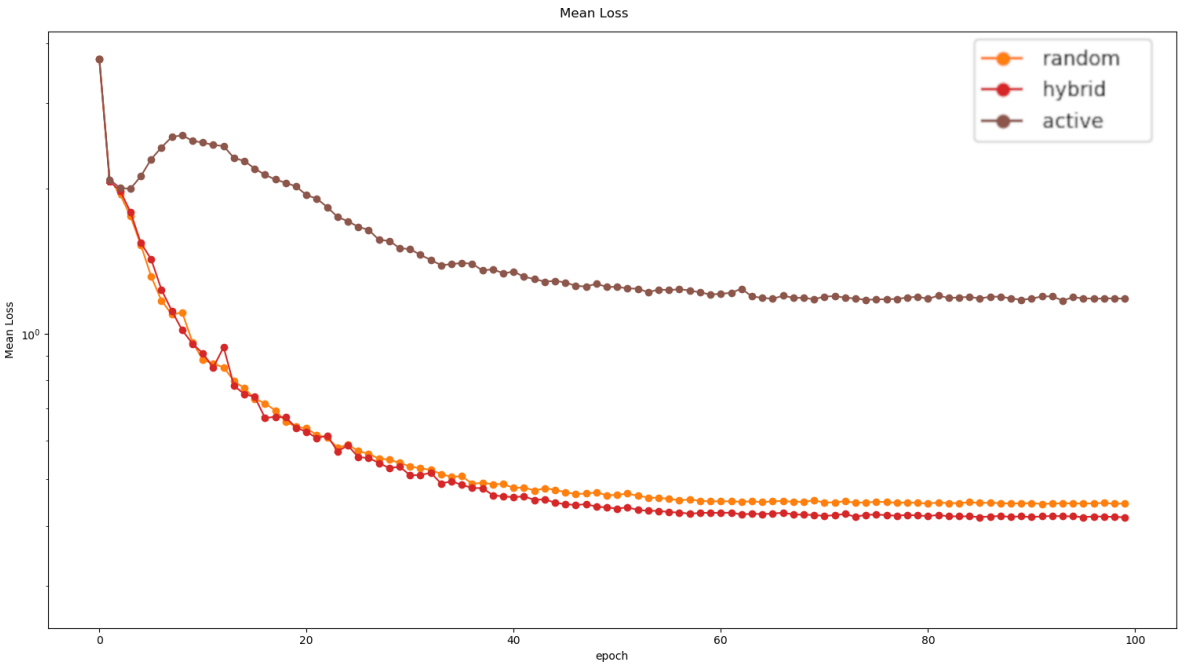


FIGURE 2.21 – Graphe de fonction de coût en validation au cours de l'apprentissage pour les différentes méthodes de choix des échantillons pendant chaque *epoch*. La droite horizontale verte correspond au seuil de 20% d'erreur de classification.

Méthode	Erreur d'entraînement Finale	Erreur de validation Finale	Première epoch sous les 20% d'erreur (Entraînement/Validation)	Durée moyenne pour 100 epochs
Aléatoire	**8.28%**	14.14%	**15/23**	**8512** s
Hybride	10.80%	**13.25%**	20/**23**	8528 s (+0.18%)
Active	8.78%	22.49%	17/Jamais	14312 s (+68.14%)

TABLEAU 2.3 – Les valeurs en **gras** indiquent les meilleurs résultats de la colonne. Les pourcentages entre parenthèses dans la dernière colonne indiquent le surplus de temps d'entraînement par rapport à la méthode complètement aléatoire.

On observe en figure 2.21 que la fonction de coût en validation pour la méthode « active » commence par augmenter avant de descendre et converger vers une valeur bien plus haute que pour les deux autres méthodes, alors qu'en entraînement on observe un comportement similaire aux deux autres méthodes.

Ceci peut s'expliquer par le fait que la méthode « active » ne fait voir au réseau que de « mauvais » échantillons du jeu d'entraînement (qui intuitivement vont contenir entre autre des valeurs aberrantes), ce qui n'aide pas à apprendre une distribution générale, qui ne représenterait pas uniquement le jeu d'entraînement, mais toutes les données qu'on aimerait classifier. Ceci semble donc être un cas flagrant de sur-apprentissage (ou *over-fitting*).

On peut donc en conclure que la méthode « hybride » est un bon compromis entre les méthodes complètement aléatoire et « active ».

2.5.2 Étude comparative des différentes architectures multi-échelles

Pour comparer les trois méthodes de fusion multi-échelles « précoce », « tardive » et « cohérente », on applique le même protocole que précédemment, avec la méthode d'échantillonnage complètement aléatoire et avec les trois échelles 5 cm, 10 cm et 20 cm. La partie convolutionnelle est la même pour les trois réseaux, les fusions « précoce » ou « cohérente » sont appliquées avant la partie convolutionnelle, alors que la fusion « tardive » est appliquée après.

Les figure 2.22 page suivante et tableau 2.4 page ci-contre résument les résultats obtenus.

Comme on pouvait s'y attendre la fusion tardive produit de meilleurs résultats que la fusion

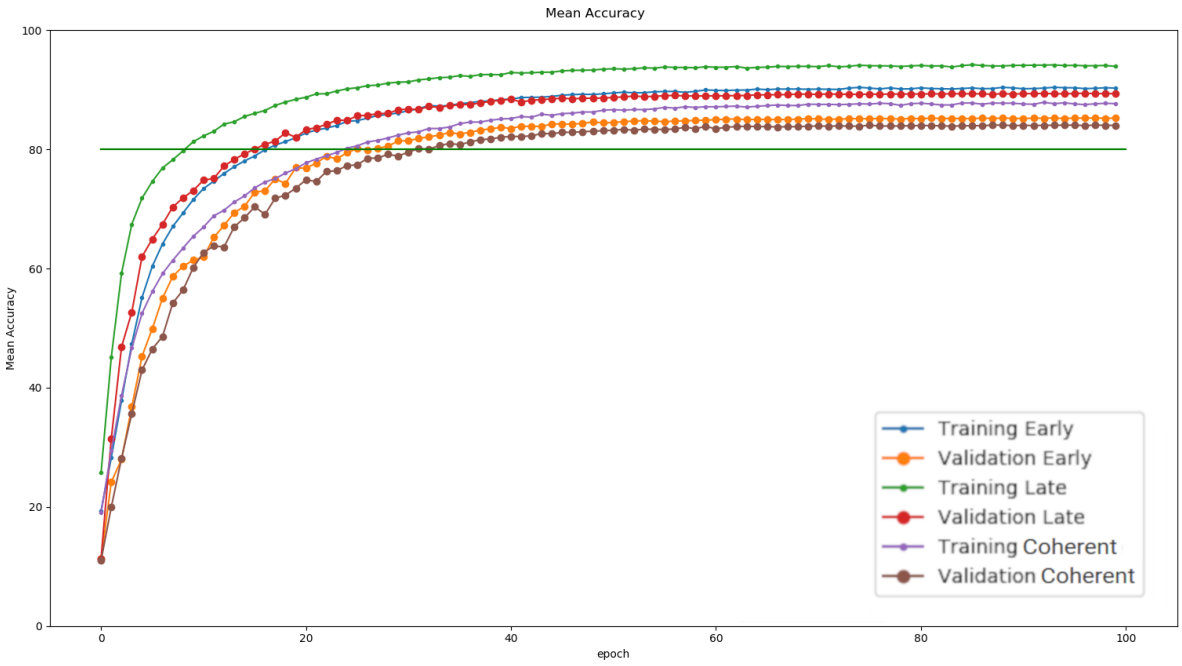


FIGURE 2.22 – Graphe d'*accuracy* et fonction de coût en validation et en entraînement au cours de l'apprentissage pour les différentes méthodes de fusion des échelles.

Fusion	Erreur d'entraînement Finale	Erreur de validation Finale	Première epoch sous les 20% d'erreur (Entraînement/Validation)	Durée moyenne pour 100 epochs
Précoce	9.72%	14.77%	16/25	**10350** s
Tardive	**6.07**%	**10.64**%	**9/15**	13467 s (+30.12%)
Cohérente	12.33%	15.97%	24/30	13570 s (+31.11%)

TABLEAU 2.4 – Les valeurs en **gras** indiquent les meilleurs résultats de la colonne. Les pourcentages entre parenthèses dans la dernière colonne indiquent le surplus de temps d'entraînement par rapport à la méthode fusion précoce.

précoce, puisque les données fusionnées ne représentent certes pas des portions d'espace de même taille, mais sont au moins centrées sur le même point aux différentes échelles (ce qui n'est pas la cas pour la fusion précoce, pour rappel la figure 2.10 page 58)

Ensuite, la fusion cohérente obtient d'encore moins bon résultats que la fusion précoce, malgré l'intérêt théorique derrière une fusion cohérente des voxels à différentes échelles. Ceci peut s'expliquer de deux façons :

— les cartes fusionnées ne représentent pas exactement le voisinage du point, en effet lors d'une fusion, la carte de plus basse résolution est complètée par des 0 pour avoir la même taille que la carte de plus haute résolution, alors que théoriquement à ces endroits comblés il peut y avoir des points donc de l'information non-nulle.

— ici la fusion cohérente est utilisée uniquement au début du réseau (avant la partie convolutionnelle), c'est peut-être trop tôt pour fusionner les échelles, il est peut-être plus facile de fusionner deux échelles après quelques traitements au milieu de la partie convolutionnelle, ou bien même à la toute fin. Il est encore possible qu'il faille fusionner les échelles après chaque sous-échantillonnage de la partie convolutionnelle.

En l'état, la méthode par fusion tardive semble la meilleure façon de fusionner des grilles de voxels multi-échelles, et on verra en section 2.5.4 page 73 qu'elle permet d'atteindre l'état de l'art en classification par points.

2.5.3 Étude de l'apport de l'architecture multi-échelles par rapport à une seule échelle (méthode de fusion tardive)

Nous avons dans un premier temps trouvé le meilleur pas de discrétisation dans le cas mono-échelle par méthodes de validation croisées, on a ensuite ajouté des échelles plus petites pour avoir une information local autour du point plus précise et des échelles plus grandes pour avoir plus d'information de contexte dans la scène. Pour cette expérimentation, les réseaux multi-échelles utilisent la fusion tardive, et la méthode d'échantillonnages des points d'entraînement est « aléatoire ».

Le pas de discrétisation offrant les meilleurs performances de classification en mono-échelle est 10 cm pour les jeux de données de nuages de points urbains. Nous avons donc testé les combinaisons d'échelles à 5 cm, 10 cm et 20 cm. Les résultats sont résumés en figure 2.23 et tableau 2.5.

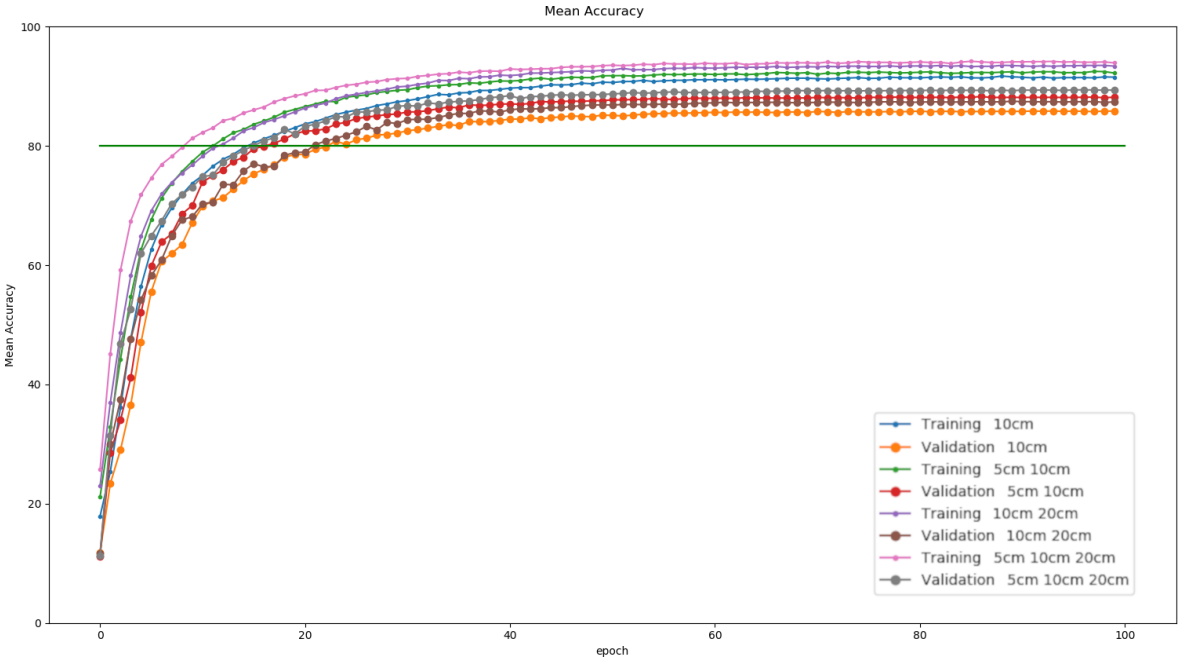


FIGURE 2.23 – Graphe d'*accuracy* et fonction de coût en validation et en entraînement au cours de l'apprentissage pour les différentes combinaisons d'échelles choisies.

Échelles	Erreur d'entraînement Finale	Erreur de validation Finale	Première epoch sous les 20% d'erreur (Entraînement/Validation)	Durée d'inférence d'un batch de taille 20
10 cm	8.47%	14.16%	15/23	30.5 ms
5 et 10 cm	7.78%	11.82%	11/17	57.2 ms (+87.5%)
10 et 20 cm	6.62%	12.61%	12/21	57.2 ms (+87.5%)
5, 10 et 20 cm	**6.07%**	**10.64%**	**9/15**	87.4 ms (+186%)

TABLEAU 2.5 – Comparaison des résultats en foncion des échelles choisies. La méthode utilisée est la fusion tardive pour toutes les configurations et la méthode d'entraînement est par échantillonnage complètement aléatoire. Les valeurs en **gras** indiquent les meilleurs résultats de la colonne. Les pourcentages entre parenthèses dans la dernière colonne indiquent le surplus de temps d'entraînement par rapport à la méthode mono-échelle.

On observe deux phénomènes sur les résultats de classification :

— comme attendu, l'ajout d'une ou plusieurs échelles, quelles soient plus grandes ou plus petites, améliore de façon non-négligeable les performances de classification,

— l'ajout d'une échelle plus petite est plus bénéfique que l'ajout d'une échelle plus grande, mais c'est bien l'ajout des deux échelles qui obtient les meilleurs résultats.

— les temps de calcul sont de l'ordre de 2× (respectivement 3×) plus important pour 2 (respectivement 3) échelles que pour un réseau mono-échelles, mais légèrement inférieurs car

ce n'est pas l'intégralité du réseau qui est dupliqué pour chaque échelle. La partie *fully-connected* reste unique à la fin de réseau pour agréger les features obtenues des différentes échelles.

2.5.4 Comparaison avec l'état de l'art

Nous comparons également nos méthodes sur deux benchmarks publiques de scènes de nuages de points 3D extérieur :

— Semantic3D, pour une visualisation des résultats voir tableau 2.6

— Paris-Lille-3D, pour une visualisation des résultats voir figure 2.24 page suivante et tableau 2.7 page suivante

Et sur un jeu de données de scènes intérieures, S3DIS : même si comme on va le voir les résultats semblent moins pertinents/convaincant sur ce jeu de données en intérieur. Pour une visualisation rapide des résultats voir figure 2.25 page 75 tableau 2.8 page 76

2.5.4.1 Résultats sur Semantic3D

Les résultats soumis au benchmark *reduced-8* de Semantic3D ont été obtenus avec les 3 échelles 5 cm, 10 cm et 15 cm en entraînant les réseau pendant 1000 *epochs* avec 100 points par classe et par *epoch* choisis par méthode d'échantillonnage complètement aléatoire. Les résultats comparés à l'état de l'art sont résumés en tableau 2.6 (uniquement les méthodes publiées soumises au moment de la soumission de [ROYNARD, J. DESCHAUD et collab., 2018a] sont présentées).

Rang	Méthode	IoU Moyen	IoU par classe							
			man-made terrain	natural terrain	high vegetation	low vegetation	buildings	hard scape	scanning artefacts	cars
1	SPGraph [Loic Landrieu et Simonovsky, 2018]	**73.2%**	**97.4%**	**92.6%**	**87.9%**	**44.0%**	**93.2%**	31.0%	**63.5%**	76.2%
2	MS3_DVS [Roynard, J. Deschaud et collab., 2018a]	65.3%	83.0%	67.2%	83.8%	36.7%	92.4%	**31.3%**	50.0%	**78.2%**
3	RF_MSSF [Thomas et collab., 2018]	62.7%	87.6%	80.3%	81.8%	36.4%	92.2%	24.1%	42.6%	56.6%
4	SegCloud [Tchapmi et collab., 2017]	61.3%	83.9%	66.0%	86.0%	40.5%	91.1%	30.9%	27.5%	64.3%
5	SnapNet_ [Boulch, Saux et collab., 2017]	59.1%	82.0%	77.3%	79.7%	22.9%	91.1%	18.4%	37.3%	64.4%
9	MS1_DVS [Roynard, J. Deschaud et collab., 2018a]	57.1%	82.7%	53.1%	83.8%	28.7%	89.9%	23.6%	29.8%	65.0%

TABLEAU 2.6 – Les 5 meilleurs resultats sur le benchmarck publique *Semantic3D reduced-8*. MS3_DVS est notre réseau avec 3 échelles de tailles 5 cm, 10 cm et 15 cm par fusion tardive et entraîné par méthode d'échantillonnage complètement aléatoire, tandis que MS1_DVS est notre réseau avec une seule échelle de taille 10 cm (ajouté pour la comparaison avec un réseau non multi-échelles).

L'intérêt d'utiliser un réseau multi-échelle est confirmé ici, puisque un réseau à 3 échelles (MS3_DVS) atteint la deuxième place du benchmark, alors qu'un réseau à une seule échelle mais avec la même architecture de partie convolutionnelle se retrouve dans les dernières du classement (9ème) avec un *IoU* moyen plus petit de 8.2 points. L'amélioration se fait sur toutes les classes sans exception et est même très importante sur la plupart des classes.

En comparaison à la méthode RF_MSSF qui est également multi-échelle mais qui utilise des descripteurs conçus à la main (de type attributs de dimensionnalité) sur chaque voisinage, notre méthode est meilleure sur toutes les classes excepté les classes de sol (terrain artificiel et terrain naturel).

2.5.4.2 Résultats sur Paris-Lille-3D

Les résultats soumis au benchmark *Paris-Lille-3D* ont été obtenus avec les 3 échelles 5 cm, 10 cm et 15 cm par fusion tardive en entraînant le réseau pendant 1000 *epochs* avec 100 points par classe et par *epoch* choisis par méthode d'échantillonnage complètement aléatoire. Les résultats comparés à l'état de l'art sont résumés en tableau 2.7 et quelques résultats qualitatifs sont présentés en figure 2.24.

			IoU par classe								
Rang	Méthode	IoU Moyen	ground	building	pole	bollard	trash can	barrier	pedestrian	car	natural
1	MS3_DVS [Roynard, J. Deschaud et collab., 2018a]	**66.89%**	99.03%	**94.76%**	**52.40%**	38.13%	**36.02%**	**49.27%**	**52.56%**	**91.3%**	**88.58%**
2	RF_MSSF [Thomas et collab., 2018]	56.28%	**99.25%**	88.63%	47.75%	**67.27%**	2.31%	27.09%	20.61%	74.79%	78.83%

TABLEAU 2.7 – Resultats sur le benchmarck publique *Paris-Lille-3D*. MS3_DVS est notre réseau MS3_DeepVoxScene avec des voxels de tailles 5 cm, 10 cm et 15 cm. Les valeurs en **gras** indiquent les meilleurs résultats de chaque colonne.

La méthode RF_MSSF [THOMAS et collab., 2018] est également une méthode multi-échelle, mais au lieu d'apprendre des descripteurs par apprentissage profond, les descripteurs sont réalisés à la main, et beaucoup plus d'échelles sont utilisées (8 échelles allant de voisinages de taille 0.1 m à 12.8 m), puis la classification est faite grâce à un classifieur de type *Random-Forest*.

Les résultats par classe sont meilleurs, excepté pour la classe poteaux (*bollard*), qui tire sans-doute ses meilleurs performances des plus grandes échelles fournis par RF_MSSF apportant plus de contexte.

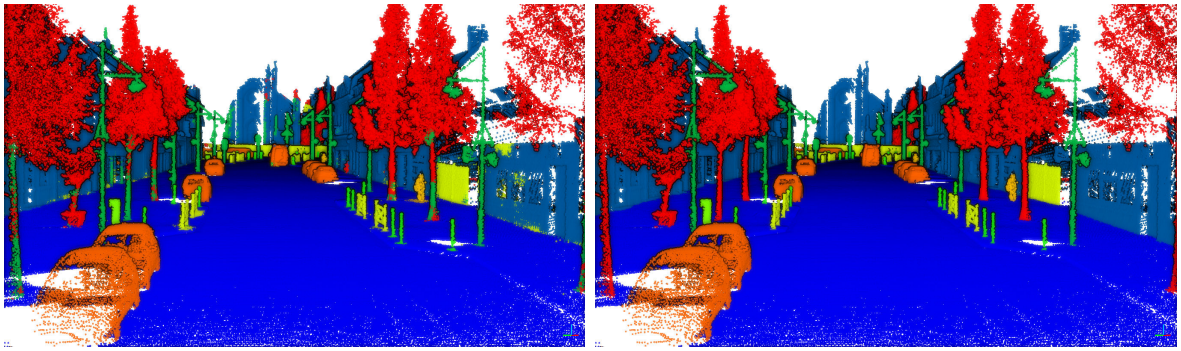


FIGURE 2.24 – Exemple de nuage de points classifié du jeu de données Paris-Lille-3D. À gauche : classifié grâce à MS3_DVS [ROYNARD, J. DESCHAUD et collab., 2018a] à trois échelles de tailles 5 cm, 10 cm et 15 cm, à droite : la vérité terrain (bleu : sol, bleu azur : bâtiments, vert foncé : potelets, vert : poteaux, vert clair : poubelles, jaune : barrières, jaune foncé : piétons, orange : voitures, red : végétation).

On observe que certains troncs d'arbres sont classés comme potelets, il semblerait donc que le contexte éloigné ne soit pas assez pris en compte, bien qu'avec le plus grand pas de discrétisation de 15 cm la grille ait une taille de 4m80. Ceci pourrait sûrement être corrigé en ajoutant des échelles plus grandes à 20 cm ou 30 cm par exemple.

De plus, le sol autour des objets posés dessus est classé comme appartenant au dit objet (sauf pour les voitures). Ceci montre que le réseau n'a pas réussi à apprendre correctement les frontières entre certaines classes comme les potelets et le sol. Cela peut s'expliquer par le désequilibre de répartition des points de chaque classe, en effet tout point d'un potelet est à petite distance de

la frontière avec le sol (donc cette frontière apparait dans tous les voisinages de tels points), alors que très peu de points du sol sont proche d'un potelet. Ceci peut se reformuler en disant que la frontière potelet/sol n'est vu pendant l'apprentissage que très peu du côté sol et beaucoup du côté potelet, c'est pour cette raison que les points proches de la frontière sont classifiés comme potelet.

Pour remédier à ce problème, on peut imaginer au moins deux solutions :

— changer la méthode d'échantillonnage des points d'entraînement, en favorisant les points aux frontières des objets,

— ajouter une échelle plus fine pour mieux capturer la position exacte de la frontière.

2.5.4.3 Résultats sur S3DIS

Les mêmes réseaux ont été testés sur le jeu de données d'intérieur S3DIS, pour évaluer l'intérêt du multi-échelles sur des scènes d'intérieur. La figure 2.25 présente des résultats qualitatifs sur S3DIS, et le tableau 2.8 page suivante présente la comparaison quantitative avec l'état de l'art.

On observe avant tout qu'il y a une grande confusion entre la classe « autre » (*clutter* en anglais) et les autres classes, ce qui peut s'expliquer par le fait que cette classe contient des objets de toutes formes et tailles qui peuvent parfois ressembler aux objets d'une classe existante.

De plus, comme sur l'ensemble de données Paris-Lille-3D, le sol autour des objets tels que les chaises est classé comme appartenant à l'objet et non au sol. On peut imaginer qu'au cours de l'entraînement, le réseau n'a vu que peu de points du sol proches de la frontière avec une chaise. Ici encore, il faudrait améliorer la méthode d'échantillonnage des points et ajouter une échelle plus fine.

On peut également remarquer en figure 1.6 page 14 que géométriquement, certaines classes comme les tableaux ne se distinguent pas ou très peu des murs, alors que nos réseaux ne prennent en entrée qu'une grille d'occupation qui ne contient pas d'information colorimétrique. Il faudrait donc ajouter des canaux de couleur à la grille d'occupation en entrée du réseau pour mieux prendre en compte cet aspect.

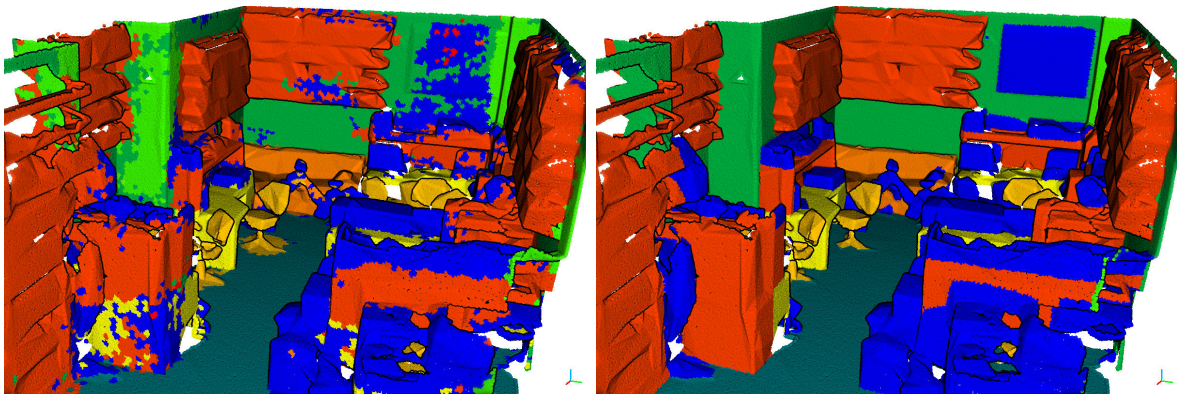


FIGURE 2.25 – Exemple de nuage de points classifié du jeu de données S3DIS. À gauche : classifié grâce à MS3_DVS [ROYNARD, J. DESCHAUD et collab., 2018a] notre réseau multi-échelles de tailles 5 cm, 10 cm et 15 cm, à droite : la vérité terrain (bleu : fouillis, bleu azur : sol, vert foncé : murs, vert : colonnes, jaune foncé : chaise, jaune clair : table, orange foncé : bibliothèque, orange clair : canapé).

Méthode	IoU Moyen	Accuracy Moyenne	IoU par classe (en %)												
			ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
SPG [Loic Landrieu et Simonovsky, 2018]	**54.67%**	**61.75%**	**91.49**	**97.89**	**75.89**	0.00	14.25	**51.34**	**52.29**	**86.35**	**77.40**	**65.49**	40.38	7.23	**50.67**
SEGCloud [Tchapmi et collab., 2017]	48.92%	57.35%	90.06	96.05	69.86	0.00	18.37	38.35	23.12	75.89	70.40	58.42	40.88	12.96	41.60
MS3_DVS [Roynard, J. Deschaud et collab., 2018a]	46.32%	57.93%	79.03	88.07	53.55	0.00	**20.47**	29.01	37.29	68.84	63.72	47.44	**61.62**	16.50	36.64
PointNet [Charles R Qi et collab., 2017]	41.09%	48.98%	88.80	97.33	69.80	**0.05**	3.92	46.26	10.76	52.61	58.93	40.28	5.85	**26.38**	33.22

TABLEAU 2.8 – Resultats sur le 5$^{\text{ième}}$ *fold* du jeu de données *S3DIS*. MS3_DVS [ROYNARD, J. DESCHAUD et collab., 2018a] est notre réseau multi-échelles de tailles 5 cm, 10 cm et 15 cm. Les valeurs en **gras** indiquent les meilleurs résultats de chaque colonne.

2.6 Perspectives et Conclusion

Il y a de nombreuses perspectives d'améliorations qui ressortent du travail présenté, par ordre d'importance croissante (des moins aux plus prometteuses) :

— Faire de l'ensemble sur plusieurs réseaux (faire voter plusieurs réseaux initialisés et/ou entraînés différemment) ou sur plusieurs rotations d'un même nuage,

— ajouter en entrée des *features* géométriques (type attributs de dimensionnalité de [Martin WEINMANN et collab., 2015]) calculées à l'avance sur le nuage de points,

— calibrer la réflectance ou ajouter en entrée des *features* qui permettent de calibrer la réflectance (distance au capteur, angle d'incidence du laser, ...),

— pré-entraîner les branches convolutionnelles en mode mono-échelle avant de faire du multi-échelle, de même entraîner la partie encodeur en mode classification de point avant de faire du U-Net. Combiner les deux pour le U-Net multi-échelles,

— faire du multi-tâches / multi-dataset en mettant en sortie plusieurs *softmax* + *loss*, une pour chaque jeu de donnée (ou tâche),

— tester les réseaux multi-échelles avec des réseaux utilisant des convolutions par points au lieu de voxels,

— mettre en entrée du réseau la TSDF (à chaque échelle) et en sortie dans le cas d'un auto-encodeur, et la carte de distance à chaque classe [AUDEBERT et collab., 2018].

Dans ce chapitre, après un état de l'art des méthodes d'apprentissage profond en classification et segmentation d'image puis des méthodes qui utilisent l'apprentissage profond en 3D, nous avons présenté de nouvelles méthodes d'entraînement de réseaux de neurones sur des scènes de nuages de points 3D complètement annotées, ainsi que de nouvelles architectures de réseau de neurones prenant en compte une information multi-échelle. On a enfin validé et comparé ces méthodes sur des jeux de données de la littérature et sur des benchmarks publiques.

On remarquera en particulier que :

— les méthodes d'entraînement développées ici, bien que utilisées uniquement pour de la classification de points par réseaux de neurones convolutionnels 3D, peuvent être utilisées pour apprendre d'autres tâches avec d'autres types de réseaux (ou algorithmes entraînés par « *epochs* »)

— les architecture multi-échelles développées ici, bien que testées uniquement sur des réseaux à entrée sous forme de grilles d'occupation voxelliques, peuvent être appliquées à d'autres types de données, en particulier des nuages de points (avec les réseaux à convolutions par points).

Références

AIJAZI, Ahmad Kamal, Paul CHECCHIN et Laurent TRASSOUDAINE (2013). « Segmentation based classification of 3D urban point clouds: A super-voxel based approach with evaluation ». In : *Remote Sensing* 5.4, p. 1624-1650 (cf. p. 46).

AIJAZI, Ahmad Kamal, Andrés SERNA, Beatriz MARCOTEGUI, Paul CHECCHIN et L TRASSOUDAINE (2015). « Segmentation and Classification of 3D Urban Point Clouds: Comparison and Combination of Two Approaches ». In : *10th Conference on Field and Service Robotics* (cf. p. 46).

ALDOMA, Aitor, Markus VINCZE, Nico BLODOW, David GOSSOW, Suat GEDIKLI, Radu Bogdan RUSU et Gary BRADSKI (2011). « CAD-model recognition and 6DOF pose estimation using 3D cues ». In : *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, p. 585-592 (cf. p. 49).

ANGUELOV, Dragomir, B TASKARF, Vassil CHATALBASHEV, Daphne KOLLER, Dinkar GUPTA, Geremy HEITZ et Andrew NG (2005). « Discriminative learning of markov random fields for segmentation of 3d scan data ». In : *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. T. 2. IEEE, p. 169-176 (cf. p. 46).

ARMENI, Iro, Ozan SENER, Amir R. ZAMIR, Helen JIANG, Ioannis BRILAKIS, Martin FISCHER et Silvio SAVARESE (2016). « 3D Semantic Parsing of Large-Scale Indoor Spaces ». In : *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* (cf. p. 12, 13, 65).

EL-ASHMAWY, N., A. SHAKER et W. YAN (2011). « PIXEL VS OBJECT-BASED IMAGE CLASSIFICATION TECHNIQUES FOR LIDAR INTENSITY DATA ». In : *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XXXVIII-5/W12, p. 43-48. DOI : 10.5194/isprsarchives-XXXVIII-5-W12-43-2011. URL : https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXVIII-5-W12/43/2011/ (cf. p. 47).

AUDEBERT, Nicolas, Alexandre BOULCH, Bertrand LE SAUX et Sébastien LEFÈVRE (2018). « Segmentation sémantique profonde par régression sur cartes de distances signées ». In : *Reconnaissance des Formes, Image, Apprentissage et Perception (RFIAP)* (cf. p. 77).

BALLARD, Dana H (1981). « Generalizing the Hough transform to detect arbitrary shapes ». In : *Pattern recognition* 13.2, p. 111-122 (cf. p. 45).

BESL, Paul J. et Ramesh C JAIN (1988). « Segmentation through variable-order surface fitting ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.2, p. 167-192 (cf. p. 45).

BOULCH, Alexandre et Renaud MARLET (2016). « Deep learning for robust normal estimation in unstructured point clouds ». In : *Computer Graphics Forum*. T. 35. 5. Wiley Online Library, p. 281-290 (cf. p. 51).

BOULCH, Alexandre, Bertrand Le SAUX et Nicolas AUDEBERT (2017). « Unstructured point cloud semantic labeling using deep segmentation networks ». In : *Eurographics Workshop on 3D Object Retrieval*. T. 2, p. 1 (cf. p. 47, 73).

BROCK, Andrew, Theodore LIM, JM RITCHIE et Nick WESTON (2016). « Generative and Discriminative Voxel Modeling with Convolutional Neural Networks ». In : *arXiv preprint arXiv:1608.04236* (cf. p. 48).

CASTILLO, Edward, Jian LIANG et Hongkai ZHAO (2013). « Point cloud segmentation and denoising via constrained nonlinear least squares normal estimates ». In : *Innovations for Shape Analysis*. Springer, p. 283-299 (cf. p. 45).

CHE, Erzhuo, Jaehoon JUNG et Michael J OLSEN (2019). « Object Recognition, Segmentation, and Classification of Mobile Laser Scanning Point Clouds: A State of the Art Review ». In : *Sensors* 19.4, p. 810 (cf. p. 42).

DEMANTKÉ, Jerome, Clément MALLET, Nicolas DAVID et Bruno VALLET (2011). « Dimensionality based scale selection in 3D lidar point clouds ». In : *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38.Part 5, W12 (cf. p. 49).

DENG, Jia, Wei DONG, Richard SOCHER, Li-Jia LI, Kai LI et Li FEI-FEI (2009). « Imagenet: A large-scale hierarchical image database ». In : *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, p. 248-255 (cf. p. 7, 47).

DOUILLARD, Bertrand, James UNDERWOOD, Noah KUNTZ, Vsevolod VLASKINE, Alastair QUADROS, Peter MORTON et Alon FRENKEL (2011). « On the segmentation of 3D LIDAR point clouds ». In : *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, p. 2798-2805 (cf. p. 49).

ELBAZ, Gil, Tamar AVRAHAM et Anath FISCHER (2017). « 3d point cloud registration for localization using a deep neural network auto-encoder ». In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, p. 2472-2481 (cf. p. 51).

ENGELMANN, Francis, Theodora KONTOGIANNI, Alexander HERMANS et Bastian LEIBE (2017). « Exploring spatial context for 3d semantic segmentation of point clouds ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 716-724 (cf. p. 50).

FAN, Haoqiang, Hao SU et Leonidas GUIBAS (2017). « A point set generation network for 3d object reconstruction from a single image ». In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, p. 2463-2471 (cf. p. 51).

FISCHLER, Martin A et Robert C BOLLES (1981). « Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography ». In : *Communications of the ACM* 24.6, p. 381-395 (cf. p. 45).

GOLOVINSKIY, Aleksey et Thomas FUNKHOUSER (2009). « Min-cut based segmentation of point clouds ». In : *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE, p. 39-46 (cf. p. 46).

GOLOVINSKIY, Aleksey, Vladimir G KIM et Thomas FUNKHOUSER (2009). « Shape-based recognition of 3D point clouds in urban environments ». In : *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, p. 2154-2161 (cf. p. 43).

GORTE, Ben (2007). « Planar feature extraction in terrestrial laser scans using gradient based range image segmentation ». In : *ISPRS Workshop on Laser Scanning*, p. 173-177 (cf. p. 47).

GRILLI, E, F MENNA et F REMONDINO (2017). « A REVIEW OF POINT CLOUDS SEGMENTATION AND CLASSIFICATION ALGORITHMS ». In : *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 42.2/W3 (cf. p. 42).

GUERRERO, Paul, Yanir KLEIMAN, Maks OVSJANIKOV et Niloy J MITRA (2018). « PCPNet Learning Local Shape Properties from Raw Point Clouds ». In : *Computer Graphics Forum*. T. 37. 2. Wiley Online Library, p. 75-85 (cf. p. 51).

HACKEL, Timo, Nikolay SAVINOV, Lubor LADICKY, Jan Dirk WEGNER, Konrad SCHINDLER et Marc POLLEFEYS (2017). « SEMANTIC3D. NET: A new large-scale point cloud classification benchmark ». In : *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. T. 4. ISPRS Foundation, p. 91-98 (cf. p. 9, 65).

HACKEL, Timo, Jan D WEGNER et Konrad SCHINDLER (2016). « Fast semantic segmentation of 3D point clouds with strongly varying density ». In : *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic* 3, p. 177-184 (cf. p. 49, 51).

HE, Kaiming, Georgia GKIOXARI, Piotr DOLLÁR et Ross GIRSHICK (2017). « Mask r-cnn ». In : *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, p. 2980-2988 (cf. p. 46, 146, 147).

HOOVER, Adam, Gillian JEAN-BAPTISTE, Xiaoyi JIANG, Patrick J FLYNN, Horst BUNKE, Dmitry B GOLDGOF, Kevin BOWYER, David W EGGERT, Andrew FITZGIBBON et Robert B FISHER (1996). « An experimental comparison of range image segmentation algorithms ». In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 18.7, p. 673-689 (cf. p. 47).

HOUGH, Paul VC (déc. 1962). *Method and means for recognizing complex patterns*. US Patent 3,069,654 (cf. p. 45).

HUANG, Jing et Suya YOU (2016). « Point cloud labeling using 3d convolutional neural network ». In : *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE, p. 2670-2675 (cf. p. 48, 51).

HUANG, Sheng-Jun, Jia-Wei ZHAO et Zhao-Yang LIU (2018). « Cost-Effective Training of Deep CNNs with Active Model Adaptation ». In : *arXiv preprint arXiv:1802.05394* (cf. p. 56).

JOHNSON, Andrew E et Martial HEBERT (1999). « Using spin images for efficient object recognition in cluttered 3D scenes ». In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.5, p. 433-449 (cf. p. 44).

KAMNITSAS, Konstantinos, Christian LEDIG, Virginia FJ NEWCOMBE, Joanna P SIMPSON, Andrew D KANE, David K MENON, Daniel RUECKERT et Ben GLOCKER (2017). « Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation ». In : *Medical image analysis* 36, p. 61-78 (cf. p. 49).

KANEZAKI, Asako, Yasuyuki MATSUSHITA et Yoshifumi NISHIDA (2016). « RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints ». In : *arXiv preprint arXiv:1603.06208* (cf. p. 48).

KINDERMANN, Ross (1980). « Markov random fields and their applications ». In : *American mathematical society* (cf. p. 46).

KIRILLOV, Alexander, Kaiming HE, Ross GIRSHICK, Carsten ROTHER et Piotr DOLLÁR (2018). « Panoptic Segmentation ». In : *arXiv preprint arXiv:1801.00868* (cf. p. 28, 42).

LAFFERTY, John D, Andrew MCCALLUM et Fernando CN PEREIRA (2001). « Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data ». In : *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., p. 282-289 (cf. p. 46).

LALONDE, Jean-Francois, Nicolas VANDAPEL, Daniel F HUBER et Martial HEBERT (2006). « Natural terrain classification using three-dimensional ladar data for ground robot mobility ». In : *Journal of field robotics* 23.10, p. 839-862 (cf. p. 43, 49).

LANDRIEU, Loïc, Hugo RAGUET, Bruno VALLET, Clément MALLET et Martin WEINMANN (2017). « A structured regularization framework for spatially smoothing semantic labelings of 3D point clouds ». In : *ISPRS Journal of Photogrammetry and Remote Sensing* 132, p. 102-118. DOI : https://doi.org/10.1016/j.isprsjprs.2017.08.010 (cf. p. 43, 44).

LANDRIEU, Loic et Martin SIMONOVSKY (2018). « Large-scale point cloud semantic segmentation with superpoint graphs ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 4558-4567 (cf. p. 49, 73, 76).

LI, Yangyan, Rui BU, Mingchao SUN et Baoquan CHEN (2018). « PointCNN ». In : *arXiv preprint arXiv:1801.07791* (cf. p. 50).

LIN, Tsung-Yi, Priya GOYAL, Ross GIRSHICK, Kaiming HE et Piotr DOLLAR (2017). « Focal Loss for Dense Object Detection ». In : *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, p. 2999-3007 (cf. p. 61).

LIN, Tsung-Yi, Michael MAIRE, Serge BELONGIE, James HAYS, Pietro PERONA, Deva RAMANAN, Piotr DOLLÁR et C Lawrence ZITNICK (2014). « Microsoft coco: Common objects in context ». In : *European Conference on Computer Vision*. Springer, p. 740-755 (cf. p. 7, 47).

LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU et Alexander C BERG (2016). « Ssd: Single shot multibox detector ». In : *European conference on computer vision*. Springer, p. 21-37 (cf. p. 46, 145-147).

LIU, Xingyu, Charles R QI et Leonidas J GUIBAS (2018). « Learning Scene Flow in 3D Point Clouds ». In : *arXiv preprint arXiv:1806.01411* (cf. p. 51).

LIU, Yu et Youlun XIONG (2008). « Automatic segmentation of unorganized noisy point clouds based on the Gaussian map ». In : *Computer-Aided Design* 40.5, p. 576-594 (cf. p. 45).

MARTON, Zoltan-Csaba, Dejan PANGERCIC, Nico BLODOW, Jonathan KLEINEHELLEFORT et Michael BEETZ (2010). « General 3D modelling of novel objects from a single view ». In : *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, p. 3700-3705 (cf. p. 49).

MARTON, Zoltan-Csaba, Dejan PANGERCIC, Radu Bogdan RUSU, Andreas HOLZBACH et Michael BEETZ (2010). « Hierarchical object geometric categorization and appearance classification for mobile manipulation ». In : *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE, p. 365-370 (cf. p. 50).

MATURANA, Daniel et Sebastian SCHERER (2015). « VoxNet: A 3D convolutional neural network for real-time object recognition ». In : *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, p. 922-928 (cf. p. 48, 56).

MUNOZ, Daniel, Nicolas VANDAPEL et Martial HEBERT (2009). « Onboard contextual classification of 3-d point clouds with learned high-order markov random fields ». In : *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE (cf. p. 46).

NAJAFI, Mohammad, Sarah Taghavi NAMIN, Mathieu SALZMANN et Lars PETERSSON (2014). « Non-associative higher-order markov networks for point cloud classification ». In : *European Conference on Computer Vision*. Springer, p. 500-515 (cf. p. 46).

PAPON, Jeremie, Alexey ABRAMOV, Markus SCHOELER et Florentin WORGOTTER (2013). « Voxel cloud connectivity segmentation-supervoxels for point clouds ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2027-2034 (cf. p. 46).

QI, Charles R, Hao SU, Kaichun MO et Leonidas J GUIBAS (2017). « PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 652-660 (cf. p. 50, 76).

QI, Charles Ruizhongtai, Li YI, Hao SU et Leonidas J GUIBAS (2017). « Pointnet++: Deep hierarchical feature learning on point sets in a metric space ». In : *Advances in Neural Information Processing Systems*, p. 5105-5114 (cf. p. 50, 51).

RAKOTOSAONA, Marie-Julie, Vittorio LA BARBERA, Paul GUERRERO, Niloy J. MITRA et Maks OVSJANIKOV (jan. 2019). « POINTCLEANNET: Learning to Denoise and Remove Outliers from Dense Point Clouds ». In : *arXiv e-prints*, arXiv:1901.01060, arXiv:1901.01060. arXiv : 1901.01060 [cs.GR] (cf. p. 51).

REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK et Ali FARHADI (juin 2016). « You Only Look Once: Unified, Real-Time Object Detection ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 46, 145, 146).

REN, Xiaofeng et Jitendra MALIK (2003). « Learning a classification model for segmentation ». In : *null*. IEEE, p. 10 (cf. p. 46).

ROYNARD, Xavier, Jean-Emmanuel DESCHAUD et François GOULETTE (2016). « Fast and Robust Segmentation and Classification for Change Detection in Urban Point Clouds ». In : *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B3, p. 693-699. DOI : 10.5194/isprs-archives-XLI-B3-693-2016. URL : https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B3/693/2016/ (cf. p. 45, 50, 115).

ROYNARD, Xavier, Jean-Emmanuel DESCHAUD et François GOULETTE (nov. 2017). « Paris-Lille-3D: a large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification ». In : *ArXiv e-prints*. arXiv : 1712.00032 [cs.LG] (cf. p. 65).

ROYNARD, Xavier, Jean-Emmanuel DESCHAUD et François GOULETTE (avr. 2018a). « Classification of Point Cloud Scenes with Multiscale Voxel Deep Network ». In : *arXiv preprint arXiv:1804.03583* (cf. p. 56, 57, 73-76, 115).

RUSU, Radu Bogdan, Gary BRADSKI, Romain THIBAUX et John HSU (2010). « Fast 3d recognition and pose using the viewpoint feature histogram ». In : *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, p. 2155-2162 (cf. p. 49).

SAPPA, Angel Domingo et Michel DEVY (2001). « Fast range image segmentation by an edge detection strategy ». In : *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. IEEE, p. 292-299 (cf. p. 45).

SAXENA, Shreyas et Jakob VERBEEK (2016). « Convolutional neural fabrics ». In : *Advances in Neural Information Processing Systems*, p. 4053-4061 (cf. p. 59, 144, 145).

SCHNABEL, Ruwen, Roland WAHL et Reinhard KLEIN (2007). « Efficient RANSAC for point-cloud shape detection ». In : *Computer graphics forum*. T. 26. 2. Wiley Online Library, p. 214-226 (cf. p. 45).

SENER, Ozan et Silvio SAVARESE (2018). « Active learning for convolutional neural networks: A core-set approach ». In : *stat* 1050, p. 21 (cf. p. 56).

SERNA, Andrés et Beatriz MARCOTEGUI (2014). « Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning ». In : *ISPRS Journal of Photogrammetry and Remote Sensing* 93, p. 243-255 (cf. p. 46, 47, 50).

SFIKAS, Konstantinos, Ioannis PRATIKAKIS et Theoharis THEOHARIS (2017). « Ensemble of PANORAMA-based convolutional neural networks for 3D model classification and retrieval ». In : *Computers & Graphics*. DOI : https://doi.org/10.1016/j.cag.2017.12.001. URL : http://www.sciencedirect.com/science/article/pii/S0097849317301978 (cf. p. 47).

SHAPOVALOV, Roman, Er VELIZHEV et Olga BARINOVA (2010). « Non-associative markov networks for 3d point cloud classification. the ». In : *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVIII, Part 3A* (cf. p. 46).

SZEGEDY, Christian, Sergey IOFFE, Vincent VANHOUCKE et Alexander A ALEMI (2017). « Inception-v4, inception-resnet and the impact of residual connections on learning. » In : *AAAI*. T. 4, p. 12 (cf. p. 48, 140).

TCHAPMI, Lyne, Christopher CHOY, Iro ARMENI, JunYoung GWAK et Silvio SAVARESE (2017). « Segcloud: Semantic segmentation of 3d point clouds ». In : *3D Vision (3DV), 2017 International Conference on*. IEEE, p. 537-547 (cf. p. 48, 49, 64, 67, 73, 76).

THOMAS, Hugues, François GOULETTE, Jean-Emmanuel DESCHAUD et Beatriz MARCOTEGUI (2018). « Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods ». In : *2018 International Conference on 3D Vision (3DV)*. IEEE, p. 390-398 (cf. p. 49, 51, 73, 74).

VOSSELMAN, G, BGH GORTE, G SITHOLE, T RABBANI, M THIES, B KOCH, H SPIECKER et H WEINACHER (2004). « Recognising structure in laser scanning point clouds ». In : *ISPRS working group VIII/2* (cf. p. 45).

WANG, Xinlong, Shu LIU, Xiaoyong SHEN, Chunhua SHEN et Jiaya JIA (fév. 2019). « Associatively Segmenting Instances and Semantics in Point Clouds ». In : *arXiv e-prints*, arXiv:1902.09852, arXiv:1902.09852. arXiv : 1902.09852 [cs.CV] (cf. p. 44, 45).

WANI, M Arif et Hamid R ARABNIA (2003). « Parallel edge-region-based segmentation algorithm targeted at reconfigurable multiring network ». In : *The Journal of Supercomputing* 25.1, p. 43-62 (cf. p. 45).

WEINMANN, M, A SCHMIDT, C MALLET, S HINZ, F ROTTENSTEINER et B JUTZI (2015). « Contextual Classification of Point Cloud Data by Exploiting Individual 3d Neigbourhoods ». In : *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.3, p. 271 (cf. p. 43, 46).

WEINMANN, Martin, Boris JUTZI, Stefan HINZ et Clément MALLET (2015). « Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers ». In : *ISPRS Journal of Photogrammetry and Remote Sensing* 105, p. 286-304 (cf. p. 48, 49, 52, 55, 77).

WOHLKINGER, Walter et Markus VINCZE (2011). « Ensemble of shape functions for 3d object classification ». In : *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE, p. 2987-2992 (cf. p. 50).

WU, Zhirong, Shuran SONG, Aditya KHOSLA, Fisher YU, Linguang ZHANG, Xiaoou TANG et Jianxiong XIAO (2015). « 3d shapenets: A deep representation for volumetric shapes ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1912-1920 (cf. p. 47).

XIAO, W, B VALLET, Y XIAO, J MILLS et N PAPARODITIS (2017). « Occupancy Modelling for Moving Object Detection from LIDAR Point Clouds: a Comparative Study ». In : *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, p. 171-178 (cf. p. 51).

YI, Li, Hao SU, Lin SHAO, Manolis SAVVA, Haibin HUANG, Yang ZHOU, Benjamin GRAHAM, Martin ENGELCKE, Roman KLOKOV, Victor LEMPITSKY et collab. (2017). « Large-Scale 3D Shape Reconstruction and Segmentation from ShapeNet Core55 ». In : *arXiv preprint arXiv:1710.06104* (cf. p. 51).

YU, Lequan, Xianzhi LI, Chi-Wing FU, Daniel COHEN-OR et Pheng-Ann HENG (2018). « EC-Net: an Edge-aware Point set Consolidation Network ». In : *European Conference on Computer Vision*. Springer, p. 398-414 (cf. p. 51).

ZHANG, Richard, Stefan A CANDRA, Kai VETTER et Avideh ZAKHOR (2015). « Sensor fusion for semantic segmentation of urban scenes ». In : *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, p. 1850-1857 (cf. p. 46).

ZHAO, Hengshuang, Xiaojuan QI, Xiaoyong SHEN, Jianping SHI et Jiaya JIA (2017). « Icnet for real-time semantic segmentation on high-resolution images ». In : *arXiv preprint arXiv:1704.08545* (cf. p. 57, 143, 144).

ZHU, Xiaolong, Huijing ZHAO, Yiming LIU, Yipu ZHAO et Hongbin ZHA (2010). « Segmentation and classification of range image from an intelligent vehicle in urban environment ». In : *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, p. 1457-1462 (cf. p. 47).

Chapitre 3

Vers la Segmentation Sémantique en Temps-Réel sur Système Embarqué

« *Gotta Go Faster* »

Sonic the Hedgehog

Sommaire

3.1 Introduction

Les algorithmes de perception des futurs véhicules autonomes, demandent de traiter de grandes quantités de données (plusieurs dizaines de capteurs de types différents pouvant générer plusieurs Go par secondes).

Le fait de traiter les données directement où elles sont générées plutôt que de les transférer vers un ordinateur plus puissant sur le *cloud* s'appelle l'*edge computing* [1].

L'*edge computing* a plusieurs intérêts :

— éviter de transférer des quantités phénoménales de données, donc à la fois de saturer les réseaux de communication et de consommer énormément d'énergie,

— éviter d'être dépendant d'un réseau (qui peut tomber), et donc rend le système plus robuste,

— éviter d'avoir à faire confiance à tout un tas d'agents potentiellement hostiles (problèmes de cyber-sécurité), puisqu'il permet de fermer complètement le système à toute intrusion non-physique.

Bien que les GPUs permettent d'accélérer fortement les calculs, les plus gros réseaux de neurones sont encore loin de tourner en embarqué en temps-réel sur les centaines d'images hautes résolutions et millions de points générés par les LiDARs chaque seconde sur un véhicule autonome.

Il faut donc faire un effort particulier pour améliorer les algorithmes de perception utilisés et mieux les adapter aux matériels de calculs embarqués sur ces véhicules.

On présentera d'abord en section 3.2 page 89 un état de l'art des méthodes qui peuvent permettre aux méthodes développées au chapitre 2 page 39 d'approcher l'inférence en temps réel. Puis on proposera une nouvelle approche pour appliquer un réseau convolutionnel 3D de segmentation sémantique sur une structure d'*octree* ainsi que sur une nouvelle structure similaire à l'*octree*, qui permet de n'effectuer les convolutions que là où se trouve effectivement de l'information à traiter (contrairement aux grilles denses utilisées dans le chapitre 2 page 39).

On présentera ensuite quelques résultats de temps d'inférence sur des nuages points urbains grâce à ces structures en section 3.4 page 102.

Enfin nous présenterons en section 3.5 page 105 une plateforme expérimentale LiDAR aérienne pour l'évaluation en conditions réelles des algorithmes de perception temps-réel en embarqué. La figure 3.1 page suivante montre la plateforme en vol.

1. https://www.lebigdata.fr/edge-computing-definition

FIGURE 3.1 – Le système expérimental LiDAR aérien pour l'évaluation des algorithmes de perception temps-réel en embarqué.

3.2 État de l'Art en Segmentation Sémantique Temps-Réel par Apprentissage Profond

3.2.1 Deep-Learning en temps-réel

On présente ici différentes méthodes utilisées pour réduire les temps de calculs des réseaux de neurones convolutionnels et donc s'approcher du temps réel. On s'intéressera dans l'ordre :

— au matériel spécialisé en section 3.2.1.1

— à la conception de convolutions plus efficaces en section 3.2.1.2 page suivante

— aux différentes approximations de réseaux pré-entrainés en section 3.2.1.3 page 91

[ALYAMKIN et collab., 2018] présente un challenge (le *Low-Power Image Recognition Challenge*) et une étude comparative de méthodes de classification d'images par Deep-Learning en temps-réel sur matériel de calcul destiné à l'embarqué. Ce challenge montre l'effort fait du côté de la recherche pour appliquer les méthodes de l'état de l'art en Deep-Learning à des problèmes du monde réel.

3.2.1.1 Matériel spécialisé

Pour ce qui est des réseaux de neurones (qu'on aborde longuement en annexe A page 117) l'utilisation de GPU [CHETLUR et collab., 2014] pour les opérations de convolution accélère grandement les calculs avec des speed-up qui peuvent aller de $\times 10$ à $\times 200$ par rapport à un CPU standard. On peut encore accélérer les calculs en utilisant des puces FPGA [OVTCHAROV et collab., 2015].

Les GPUs sont des unités de calcul de type SIMD (*Single Instruction on Multiple Data* pour Instruction Unique, Données Multiples), ce qui signifie qu'ils effectuent la même opération sur plusieurs données en même temps. Ceci permet par exemple de réduire considérablement les temps d'exécution d'une couche de convolution ou d'une couche d'activation dans un réseau de neurone. En effet, une couche d'activation consiste à appliquer une même fonction (par exemple un `ReLU`) à tous les voxels d'une grille 3D. Une convolution $3 \times 3 \times 3$ consiste à appliquer un même filtre de taille $3 \times 3 \times 3$ à chaque voxel et son voisinage direct.

Actuellement toutes les bibliothèques de *Deep-Learning* permettent de faire les calculs sur GPU de façon transparente pour l'utilisateur. Il suffit en général de quelques lignes pour que les données et modèles soient envoyés sur le GPU, et que tous les calculs coûteux y soient alors effectués. Ces mêmes bibliothèques permettent même souvent de faire les calculs sur plusieurs GPUs sur un même nœud de calcul, parfois en une ligne, c'est alors la bibliothèque qui gère intégralement les communications de données entre les GPUs. Un exemple de code avec la bibliothèque `PyTorch` :

```
## On met les donnees sur un GPU
  inputs = inputs.cuda()
  labels = labels.cuda()
  model = model.cuda()

## On demande a la bibliotheque de faire les calculs sur les GPUs 0 et 1
  model = DataParallel(model, devices=[0,1])

## Calculs s'executant sur les GPUs (ne differe pas de l'implementation sur
                                     CPU)
  outputs = model(inputs)
  loss = criterion(outputs,labels)
  # retro-propagation des gradients
  loss.backward()
  # pas d'optimisation par descente de gradient
  model.parameters() += learning_rate * model.parameters().grad
```

L'approche adoptée par `DataParallel` est la suivante, pour chaque *batch* :

— une réplique du réseau complet est envoyé sur chaque GPU,

— le *batch* d'entrée `inputs` est coupé en deux sur la dimension de *batch*, et chaque partie est envoyée sur un GPU différent

— l'inférence est faite sur chaque GPU avec son sous-batch,

— le calcul des gradients est fait sur chaque GPU,

— tous les gradients sont sommés sur un seul GPU pour pouvoir faire le pas d'optimisation par descente de gradient.

Cependant les GPUs, bien qu'au moins un ordre de grandeur plus efficace que les CPUs, ont encore des défauts pour l'inférence des réseaux de neurones convolutionnels en temps-réel sur systèmes embarqués. En particulier le fait d'être spécialisé dans les opérations sur `float` 32 bits alors que les réseaux profonds n'ont pas besoin d'une telle précision.

Parmi les architectures de calculs plus spécialisées on citera les FPGAs (pour *Field-Programmable Gate Array*) qui sont utilisés par [OVTCHAROV et collab., 2015] ; [C. WANG et collab., 2017], les ASICs (pour *Application-Specific Integrated Circuit*) [NURVITADHI et collab., 2016], ou encore par exemple les TPUs (pour *Tensor Processing Unit*) qui ont été conçues par Google spécialement pour l'inférence de réseaux de neurones profonds.

De plus [NURVITADHI et collab., 2016] présente une étude comparative des différentes unités de calcul pour l'inférence de réseaux de neurones binarisés (comme ceux décrit en section 3.2.1.3 page ci-contre), ils en arrivent à la conclusion que bien que les ASICs sont les architectures de calculs les plus performantes, les FPGAs offrent des performances assez proches sans avoir à se restreindre à une architecture fixé par un ASIC.

Le niveau ultime d'*edge computing* est actuellement atteint par la caméra SCAMP [DUDEK et HICKS, 2005] ; [CAREY et collab., 2013] ; [Jianing CHEN et collab., 2018] dont chaque pixel est accompagné localement d'une unité de calcul très légère mais capable de faire des opérations simples localement sur l'image, par exemple calculer un filtre de Sobel pour faire de l'identification et suivi d'objets.

Dans notre cas on peut imaginer que les premières couches convolutionnelles d'un réseau soient calculées sur la caméra. Ceci permettrait également de faire plus d'opérations à résolution complète, alors que les réseaux actuels de l'état de l'art en image commencent tous par des *strided-convolutions* pour réduire rapidement la taille des images.

3.2.1.2 Convolutions plus efficaces et factorisation

La plupart des contributions présentées dans cette section consistent à factoriser les noyaux de convolution le bon nombre de fois et selon les bonnes dimensions.

Le réseau VGG [SIMONYAN et ZISSERMAN, 2014], remplace les convolutions 5×5 par 2 convolutions 3×3 consécutives et les convolutions 7×7 par 3 convolutions 3×3. Les convolutions étant toujours entrecoupées de non-linéarités et éventuelles couches de normalisation. Ceci a l'intérêt de conserver le même champ réceptif tout en réduisant le nombre d'opérations et la place mémoire. Un effet secondaire de cette factorisation est la réduction du nombre de paramètres à apprendre et l'augmentation du nombre de non-linéarité (donc théoriquement du nombre de niveaux d'abstraction possibles du réseau). Pour une convolution 5×5 remplacée par 2 convolutions 3×3, on gagne 28% en place mémoire et en temps de calcul. Et 44,9% pour une convolution 7×7 remplacée par 3 convolutions 3×3.

Les réseaux *Xception* [CHOLLET, 2017] et *ResNeXt* [S. XIE et collab., 2017] utilisent des convolutions par groupes combinées avec des convolutions 1×1 (pour *Xception* il y a autant de groupes que de caractéristiques en entrée, on parle alors de *depthwize separable convolutions*, on rentre plus dans les détails de ce type de convolutions en annexe A.6.1 page 137), ceci a l'avantage de séparer les dimensions d'espace du noyau de la dimension de profondeur, et donc réduire considérablement le nombre de paramètres et d'opérations tout en gardant une précision très élevée puisque ces réseaux sont au niveau de l'état de l'art.

Les réseaux *MobileNetv1* [A. G. HOWARD et collab., 2017] et *MobileNetv2* [SANDLER et collab., 2018], sont également basés sur les *depthwize separable convolutions* mais ajoutent en plus deux paramètres qui permettent de jouer sur le compromis entre temps d'inférence et *accuracy* du réseau. Le but étant de pouvoir adapter un réseau à n'importe quel matériel (y compris des smartphones d'où le nom de *MobileNet*). Les deux paramètres à ajuster sont :

— un coefficient multiplicateur de largeur $\alpha \in]0,1]$: ce coefficient permet de raffiner encore plus le modèle en multipliant le nombre de cartes de caractéristiques en entrée et sortie de chaque couches par α, il réduit approximativement le nombre d'opération et de paramètres du modèle par α^2

— un coefficient multiplicateur de résolution $\rho \in]0,1]$: il est appliqué à la résolution de l'image d'entrée, ce qui a pour effet de réduire par ρ^2 le nombre d'opérations à chaque couche.

IGCNet [T. ZHANG et collab., 2017] et ses variantes *IGCv2* [G. XIE et collab., 2018] et *IGCv3* [K. SUN et collab., 2018] reprennent les *depthwize separable convolutions*, mais factorise également la convolution 1×1 en plusieurs convolutions 1×1 par groupes intercalées par des matrices de permutations qui permettent de mélanger les features entre groupes. Le lecteur est renvoyé à [K. SUN et collab., 2018] pour plus de détails sur les différences entres les différentes versions de *MobileNet* et *IGCNet*.

Les réseaux *ShuffleNet* v1 [X. ZHANG et collab., 2017] et v2 [MA et collab., 2018], ne réalisent que des convolutions par groupes ce qui permet de considérablement réduire la complexité du réseau, et pour que les groupes ne restent pas indépendants les uns des autres une opération de mélange des cartes de caractéristiques est ajoutée (au lieu de la convolution 1×1 utilisée dans les *depthwize separable convolutions*). Il s'agit simplement de changer l'ordre des cartes de caractéristiques pour que les groupes de la convolution suivante soient chacun un mélange des groupes précédents.

Le réseau *EffNet* [FREEMAN et collab., 2018], résout certains problèmes posés par *MobileNet* et *SuffleNet* :

— ces méthodes appliquées à des modèles plus légers causent une trop grande perte en précision,

— elles présentent des taux de compression (réduction du nombre de cartes de caractéristiques entre deux couches) trop drastiques ce qui limite la capacité d'apprentissage du réseau [SZEGEDY, VANHOUCKE et collab., 2016].

Pour ce faire, *EffNet* est composé uniquement de convolutions en profondeur 1×1, et de convolutions en espace 3×1 ou 1×3, de telle sorte qu'à aucun moment il n'y ait de taux de compression supérieur à 2.

[G. HUANG, S. LIU et collab., 2018] propose le réseau *CondenseNet* qui utilise l'architecture *DenseNet* avec un nouveau module appelé *learned group convolution*. L'architecture *DenseNet* permet la réutilisation de toutes les features créées en amont, alors que les *learned group convolutions* suppriment les connexions entre couches qui n'apportent pas ou peu d'information ce qui permet de réduire le nombres d'opérations. Les *learned group convolutions* sont apprises en 3 étapes :

— l'étape de condensation, le réseau est entrainé avec une régularisation qui pousse les connexions à être éparses,

— les connexions avec une magnitude faible sont alors supprimées,

— le réseau est alors entrainé jusqu'à convergence.

Les deux premières étapes peuvent être appliquées plusieurs fois consécutives pour supprimer plus de connexions.

3.2.1.3 Approximations et réduction de réseaux

[RASTEGARI et collab., 2016] propose les réseaux *Binary-Weight-Networks* et *XNOR-Networks*, les deux utilisent des poids binarisés dans $-1,1$. Ceci permet de supprimer le produit dans l'opération de convolution pour les *Binary-Weight-Networks* et d'utiliser uniquement des sommes et

soustractions. Ceci permet un gain de 32× en place mémoire et 2× en temps de calcul sans perte de précision sur le réseau *AlexNet*.

Les entrées des convolutions des *XNOR-Networks* sont également binarisées ce qui permet de remplacer totalement les opérations de convolutions (qui sont des produits puis une somme) par de simples opérations binaires (ici XNOR et bitcount). Les gains de 32× en place mémoire et 58× en temps de calcul sont ici encore plus important que pour les *Binary-Weight-Networks*, mais au prix d'une perte de plus de 10 points de précision avec le réseau *AlexNet* sur *ImageNet*.

[CINTRA et collab., 2018] ne va pas aussi loin dans la quantification des poids et les activations par des ensembles d'entiers dyadiques, ce qui permet de supprimer les multiplications et de ne faire que des additions et des décalages de bits. Les poids approximés en entiers dyadiques sont obtenus grâce à une méthode de programmation en nombre entiers pour minimiser la différence avec les poids en nombre flottants. Ces approximations ne causent qu'une perte presque négligeable en précision de classification.

[HAN et collab., 2015] propose une approche en 3 étapes pour compresser un réseau déjà entrainé :

— suppression de connexions inutiles,

— quantification des poids sur 5 bits au lieu de 32,

— le codage de Huffman.

Chaque étape permet de réduire la place prise par le réseau en mémoire sans perte de précision de classification. La figure 3.2 explique plus précisément la méthode de compression.

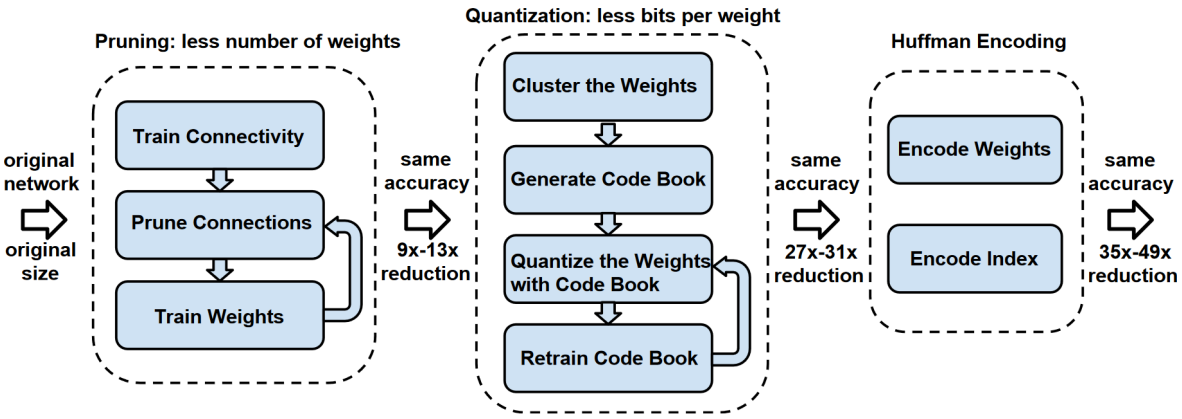


FIGURE 3.2 – Les trois étapes de compression utilisées par [HAN et collab., 2015]. (image tirée de l'article [HAN et collab., 2015])

3.2.2 Structures 3D éparses temps-réel

Notre contribution pour améliorer le temps de calcul est basée sur l'utilisation d'une structure d'*octree*. Nous verrons tout d'abord les contributions similaires sur la structure d'*octree* que ce soit en Deep-Learning (en section 3.2.2.2 page 94) ou pas (en section 3.2.2.1). On présentera quand même quelques méthodes sur d'autres structures qui ont un intérêt en section 3.2.2.3 page 95.

3.2.2.1 Méthodes classiques de traitement de nuages de points 3D sur *octree*

Un *octree* est une structure d'arbre hiérarchique de subdivision d'un espace cubique ou pavé en 3D (équivalent à un arbre binaire en 1D et un *quadtree* en 2D). La racine représente l'espace complet, et chaque nœud interne (la racine comprise) est divisé en 2 selon chaque dimension d'espace et a donc exactement 8 fils qui représentent chacun un huitième de l'espace représenté par le nœud parent. L'intérêt de l'*octree* est que les fils d'un nœud interne ne sont instanciés que s'ils contiennent de l'information (par exemple au moins un point pour un *octree* sensé représenter un nuage de points). La figure 3.3 montre l'espace représenté par les premiers étages d'un *octree*.

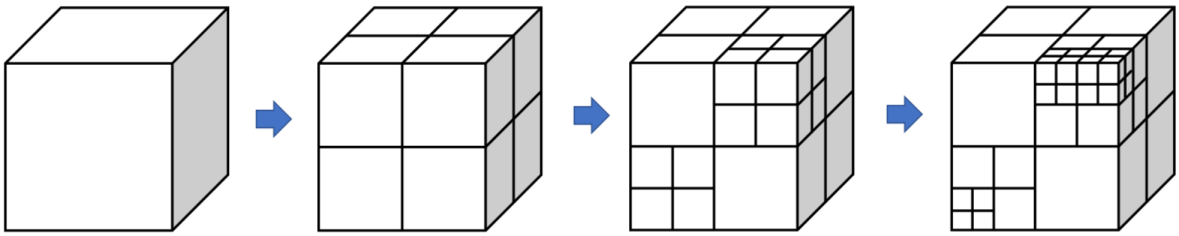


FIGURE 3.3 – Montre de gauche à droite les espaces représentés par les 4 premiers étages d'un *octree*. (image tirée de l'article [Jichao CHEN et collab., 2018])

[WURM et collab., 2010] utilise une structure d'*octree* pour garder à jour une carte 3D probabiliste des zones occupées, libres et inconnues (non-encore explorées), cette structure est appelée *OctoMap*. À chaque fois qu'un scan ou un simple point est ajouté à la carte, l'état (la probabilité d'être occupé) de chaque nœud traversé par le(s) rayon(s) laser est mis à jour. Un nœud peut donc être supprimé s'il n'est plus occupé (ce que notre contribution ne fait pas en l'état). Dans les faits, une probabilité proche de 1 indique un nœud occupé, une probabilité proche de 0 indique un nœud vide et proche de 0,5 indique un nœud à l'état inconnu ou traversé par aucun rayon laser. Cette structure permet également de faire de la compression de cartes en remplaçant la probabilité sur chaque nœud par son état libre/occupé/inconnu (codé sur 2 bits) et en ne conservant que la structure de l'arbre.

[SENGUPTA et STURGESS, 2015] utilise un MRF (pour *Markov Random Field*) sur la structure d'arbre d'un *octree*, les feuilles contenant les potentiels unitaires du MRF et les nœuds internes contenant les potentiels d'ordre plus élevé. Un algorithme de Graphe-Cut permet alors à la fois d'inférer l'occupation ou non de chaque voxel, et en cas d'occupation d'inférer la classe du voxel. Ceci permet en même temps de réaliser la reconstruction et la classification de la géométrie discrétisée par l'*octree*.

[CURA et collab., 2018] propose un descripteur à base d'attributs de dimensionnalité, mais ils ne sont pas calculés à partir des valeurs propres de la matrice de covariance locale à plusieurs échelles, ils sont calculés à partir du taux d'occupations de nœuds d'un *octree* à plusieurs profondeurs. Ceci permet d'être plus robuste aux structures linéaires mais courbées par exemple. La figure 3.4 montre un cas où l'attribut de dimensionnalité serait "surfacique", alors que le nuage est clairement linéaire.

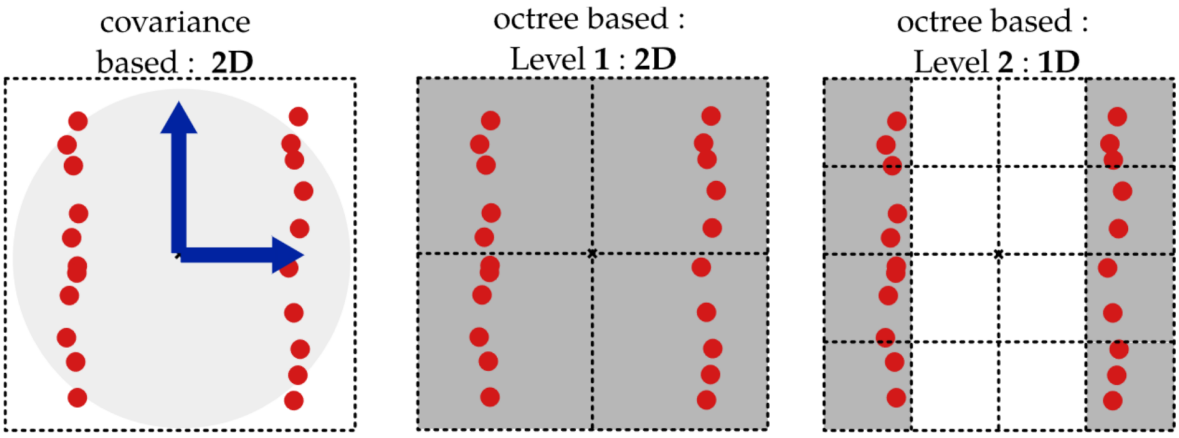


FIGURE 3.4 – Montre la robustesse des attributs de dimensionnalité basés sur le taux d'occupation d'un *octree*. À gauche les attributs de dimensionnalité basés sur la covariance classent le nuage en « surfacique » (2D), alors qu'à droite pour un *octree* assez profond, le nuage est classé en « linéaire » (1D). (image tirée de l'article [CURA et collab., 2018])

3.2.2.2 Deep-Learning sur *octree*

Les méthodes de réseaux convolutionnels s'adaptent très bien aux grilles denses en 3D, mais cette structure n'est pas optimale pour représenter des données éparses comme les nuages de points. Plusieurs chercheurs ont donc essayé d'adapter les réseaux convolutionnels à des structures plus adaptées aux nuages de points comme les *octree*.

[RIEGLER et collab., 2017] propose d'utiliser une structure hybride constituée d'une grille d'*octree* non-profonds (de profondeur 3), cette structure est appelée *OctNet*. L'image de droite de la figure 3.5 explique comment sont réalisées les convolutions sur les nœuds internes de l'*octree*.

Cette structure permet d'utiliser des grilles de taille $256 \times 256 \times 256$ au lieu de se limiter $32 \times 32 \times 32$ (ou à $64 \times 64 \times 64$ avec les plus GPUs les plus récents avec le plus de mémoire), ce qui est utile soit pour discrétiser des scènes plus grandes soit pour discrétiser plus finement (avec des voxels plus petits) certaines scènes. Cependant cette structure ne semble pas permettre de monter au dessus de cette résolution, ce qui peut être préjudiciable pour classifier une scène urbaine complète.

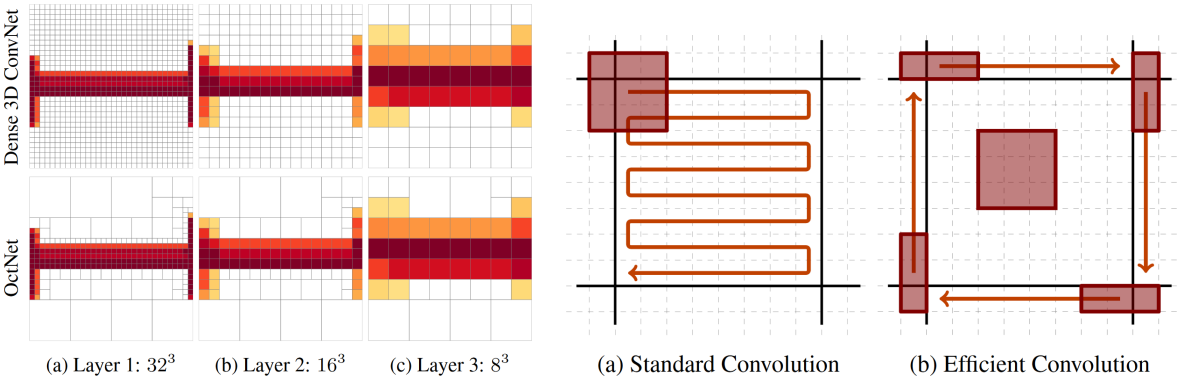


FIGURE 3.5 – L'image de gauche montre la différence du nombre de voxels instanciés entre une grille dense et un *octree*. L'image de droite montre comment les convolutions sont réalisées sur un nœud interne de l'arbre, ici un noyau 3×3 est appliqué une fois au centre et aux frontières du nœud plutôt que de balayer tout l'espace. (représenté en 2D pour la simplicité, images tirées de l'article [RIEGLER et collab., 2017])

[TATARCHENKO et collab., 2017] propose un decodeur appelé *OGN* (pour *Octree Generating Networks*) qui produit à la fois la structure d'*octree* et la valeur du signal à chaque nœud de l'*octree*. Ce décodeur est constitué de couches de convolution transposées (ou *up-convolutions*) appelées *OGNConv* qui à chaque nœud occupé produit des features à chaque enfant du nœud. Puis une couche *OGNProp* ne propage que les features des nœuds fils qui doivent effectivement être construits. L'architecture de *OGN* est décrite en figure 3.6 page suivante.

[P.-S. WANG, Yang LIU et collab., 2017] utilise un *octree* pour réaliser des opérations de convolutions uniquement aux endroits où il y a de l'information, c'est à dire uniquement aux nœuds non-vides de l'*octree*. Cette méthode est appelée *O-CNN*. Pour les convolutions avec un noyau autre que $2 \times 2 \times 2$ à pas 2 qui requièrent les features situées sur nœuds non-compris dans les voisinages implicites définis par la structure de l'*octree* (c'est-à-dire : les enfants d'un même nœud sont voisins), une structure de *Hash-Map* est utilisée pour garder en mémoire les voisinages manquants. On peut noter que chaque feuille occupée contient en plus une information d'orientation sous la forme d'une normale à la surface.

Initialement le signal (les features) n'existent qu'aux feuilles de l'*octree*. Lorsque le pas (*stride*) de la couche de convolution est de 1, le signal est traité avec une résolution inchangée et il reste au niveau de l'*octree* actuel. Lorsque le pas est supérieur à 1, le signal est condensé et est propagé à travers l'*octree* à l'étage suivant des feuilles vers la racine. L'image du haut de la figure 3.7 page ci-contre illustre le flux de données dans l'*octree*.

[P.-S. WANG, C.-Y. SUN et collab., 2018] propose *Adaptive O-CNN* qui reprend l'architecture de *O-CNN* en changeant l'*octree* par un *octree* adaptatif, c'est-à-dire dont chaque branche ne descend pas forcément jusqu'à la résolution la plus fine quand des nœuds plus fins n'apporteraient pas d'information supplémentaire, ceci permet de gagner de la place mémoire et des opérations de

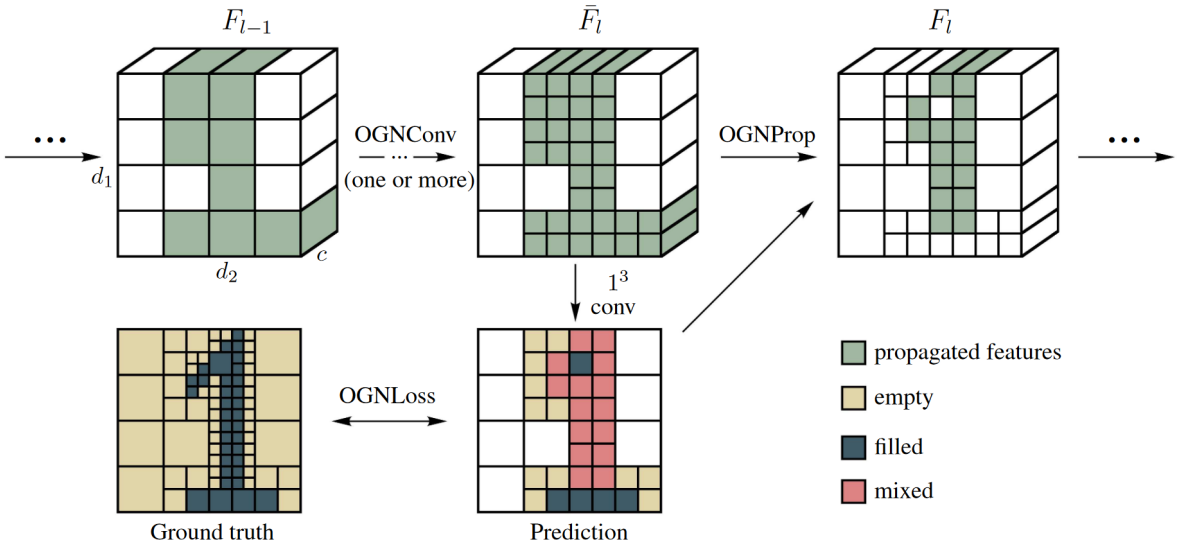


FIGURE 3.6 – Architecture du réseau *OGN*. (image tirée de l'article [TATARCHENKO et collab., 2017])

calcul. L'image du bas de la figure 3.7 décrit comment les différents étages de l'*octree* sont intégrés successivement au flux de calculs.

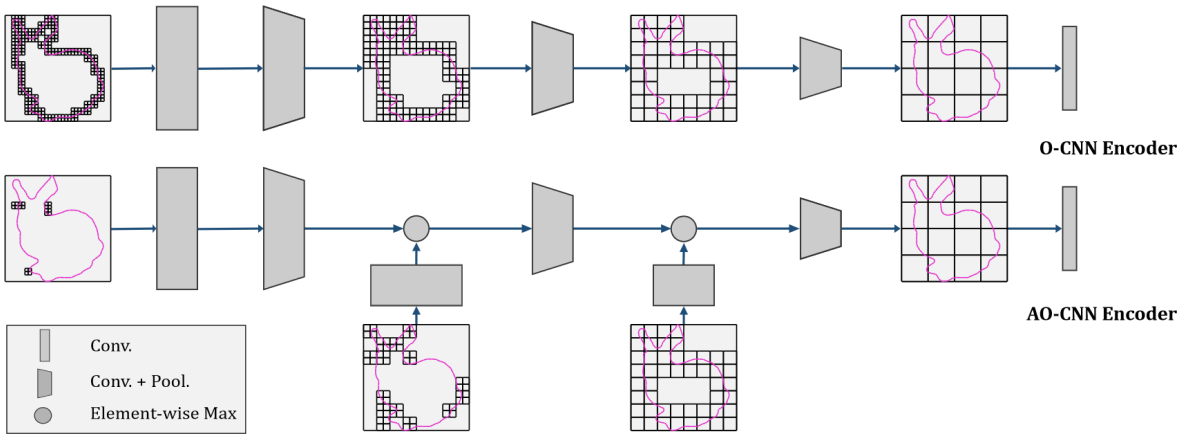


FIGURE 3.7 – Architectures de *O-CNN* (en haut) et *Adaptive O-CNN* (en bas). (image tirée de l'article [P.-S. WANG, C.-Y. SUN et collab., 2018])

La méthode *O-CNN* est la contribution la plus proche de celle présentée dans ce chapitre. Mais, la méthode utilisée pour conserver les voisinages ne semble pas adaptée, en l'état au cas dynamique où on ajoute des nuages plusieurs fois par seconde, ces nuages pouvant sortir de l'espace initialement discrétisé par l'*octree*.

[LEI et collab., 2019] utilise une structure d'*octree* pour guider les calculs d'un réseau convolutionnel (comme *O-CNN*), cependant la structure n'est pas le support des features, ce sont ici les points 3D qui conservent le support. Ce ne sont donc pas des convolutions sur grilles denses qui sont appliquées, mais des convolutions sur les points grâce à des noyaux sphériques décrits en figure 3.8 page suivante. La structure d'*octree* sert en fait à chercher le voisinage d'un point pour lui appliquer un filtre de convolution, et à gérer les niveaux de sous-échantillonnage hiérarchiques.

3.2.2.3 Deep-Learning sur d'autres structures éparses

[KLOKOV et LEMPITSKY, 2017] utilise un *kdtree* pour construire le "graphe de calcul" (*computational graphs*) et pour gérer le partage des poids du réseau appelé *Kd-Network*. La figure 3.9 page suivante décrit très bien ce processus. Par exemple pour obtenir les features au nœud 7 de l'arbre, on concatène les features des nœuds 14 et 15 et on applique un opérateur linéaire et une

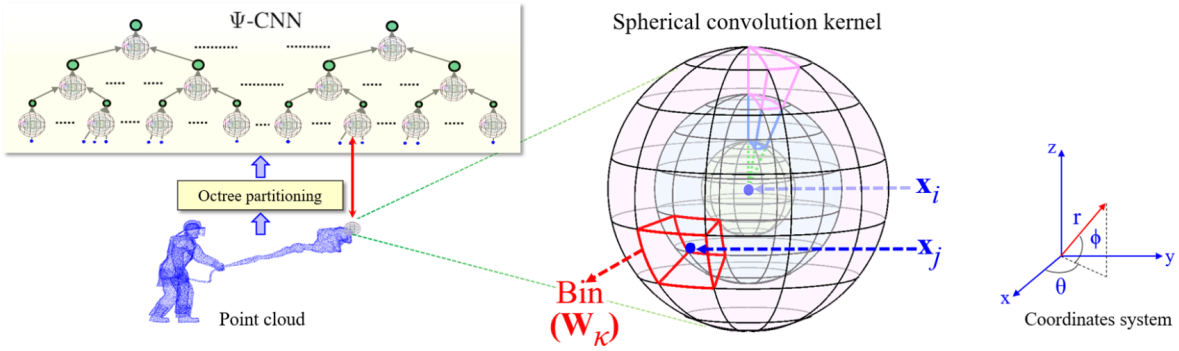


FIGURE 3.8 – Décrit comment l'*octree* est utilisé pour guider les calculs (à gauche) et comment est défini le noyau sphérique de convolution sur les points (à droite). (image tirée de l'article [LEI et collab., 2019])

non-linéarité. À une même profondeur dans le réseau/l'arbre et pour une division selon la même dimension (x, y ou z en 3D), les poids sont partagés, ainsi on applique le même opérateur linéaire pour obtenir les features aux nœuds 4, 6 et 7.

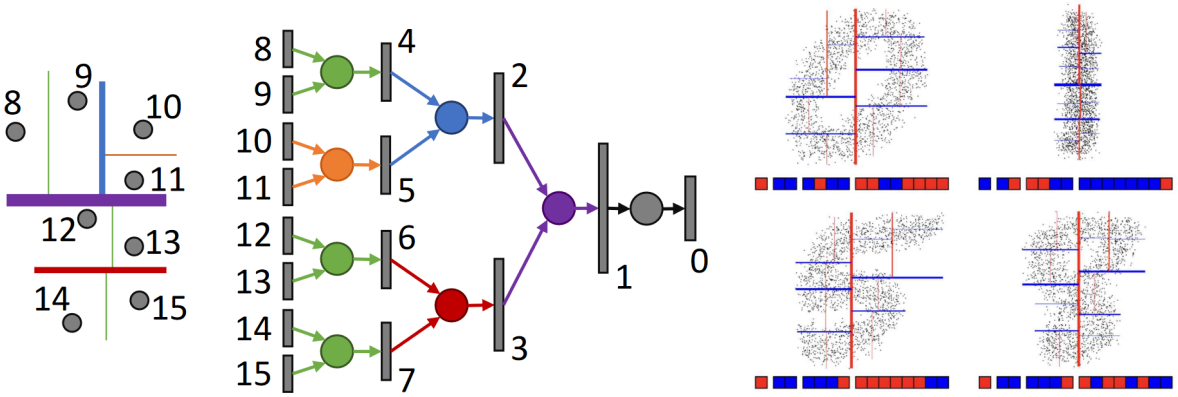


FIGURE 3.9 – À gauche, chaque point numéroté de 8 à 15 est une feuille du *Kd-Network* (représenté au milieu), deux opérations de la même couleur représentent des poids partagés. À droite, des exemples de nuages de points 2D et le *kdtree* associé, avec en rouge les divisions selon la première dimension et en bleu selon la deuxième. (images tirées de l'article [KLOKOV et LEMPITSKY, 2017])

3.3 Méthode Proposée d'Inférence en Temps-Réel de Réseaux Convolutionnels 3D sur Données Éparses

3.3.1 Un *octree* pour l'inférence en temps réel de réseaux convolutionnels 3D

On développe ici une structure de données qui permet, à un robot équipé d'un capteur 3D (LiDAR ou caméra de profondeur), de créer une carte 3D sémantique de son environnement en temps réel. On suppose que l'étape d'odométrie ou de localisation est bien maîtrisée et tourne en temps réel. Cette dernière étape produit des nuages de points acquis par le capteur 3D et recalés dans un repère fixe (géo-référencé ou dans le répère de la position initiale du robot).

La structure que nous développons ici doit vérifier plusieurs critères que l'on estime indispensables pour la perception des véhicules autonomes :

— dynamique : intègre les nuages recalés au fur et à mesure de l'acquisition, même s'ils sortent d'une zone initiale

— temps-réel pour intégrer un nouveau nuage à la carte : prend moins de temps à intégrer un nuage que le temps qu'il a fallu pour l'acquérir ou qu'il faut pour acquérir le suivant (typiquement pour des *frames* de Velodyne HDL-32e, jusqu'à 70000 points à ajouter toutes les 0,1 s),

— temps-réel pour classifier la partie modifiée de la carte : permet d'évaluer un réseau de neurones convolutionnel 3D pour classifier la carte à la volée, c'est-à-dire qu'au moment où un nouveau nuage est intégré à la carte, la partie de la carte modifiée doit être classifiée avant qu'on veuille y intégrer un nouveau nuage,

— utiliser une grande gamme de réseaux de neurones convolutionnels 3D (on verra en fait qu'on peut y arriver en se limitant aux réseaux dont les convolutions ont uniquement des noyaux de taille $k \times l \times m$ avec $k,l,m \leq 3$)

Pour remplir ces critères, les solutions retenues dans la littérature s'appuient sur des structures capables de gérer l'information 3D éparse (*octree*, *kdtree*...) comme on l'a vu en section 3.2.2 page 92. L'avantage de l'*octree* venant de la structure de grille régulière dense sous-jacente et implicite et du fait qu'il permet de garder cette information de façon hiérarchique et éparse.

Cependant les structures développées dans l'état de l'art ne vérifient pas tous les critères énoncés plus haut, entre autre *O-CNN* utilise une structure de voisinage qui n'autorise pas l'intégration de nouveaux points, en particulier des points hors de l'espace initialement discrétisé par l'octree. De plus les *octree* utilisés dans *O-CNN* sont de profondeur fixée ce qui empêche le caractère dynamique de la structure.

En s'intéressant à la structure d'un réseau convolutionnel composé uniquement de convolutions $2 \times 2 \times 2$ à pas 2, on remarque que le graphe de calcul du réseau (le flux de données à travers ses couches) représente exactement la structure d'arbre de l'*octree* construit sur la grille 3D dense donnée en entrée du réseau.

En effet, une convolution $2 \times 2 \times 2$ à pas 2 est une opération de sous-échantillonnage ou d'agrégation qui agrège l'information sur 8 voxels voisins de tailles $\Delta \times \Delta \times \Delta$, en un plus gros voxel les contenant de taille $2\Delta \times 2\Delta \times 2\Delta$. Ainsi cette opération revient à propager les features dans l'*octree* des fils d'une profondeur d à leurs parents à profondeur $d+1$. Par le même raisonnement, une opération de convolution transposée $2 \times 2 \times 2$ à pas 2 revient à propager des features depuis les parents à profondeurs d vers leurs fils à profondeurs $d-1$. Ce phénomène est expliqué en figure 3.10 page suivante.

On prend ici la convention de définir la profondeur d'un nœud comme étant la distance de ce nœud à ses feuilles, ainsi les feuilles sont toujours de profondeur 0. Ceci se justifie car on travaille avec des *octrees* dynamiques (dont la racine peut potentiellement être modifiée pour prendre en compte une plus grande zone, et donc augmenter sa distance aux feuilles), alors que les feuilles sont fixées en taille.

Initialement le signal (les features) n'existent qu'aux feuilles de l'*octree*, ce sont soit un vecteur de taille 1 contenant un 1, soit un vecteur contenant des attributs propres aux points ajoutés dans

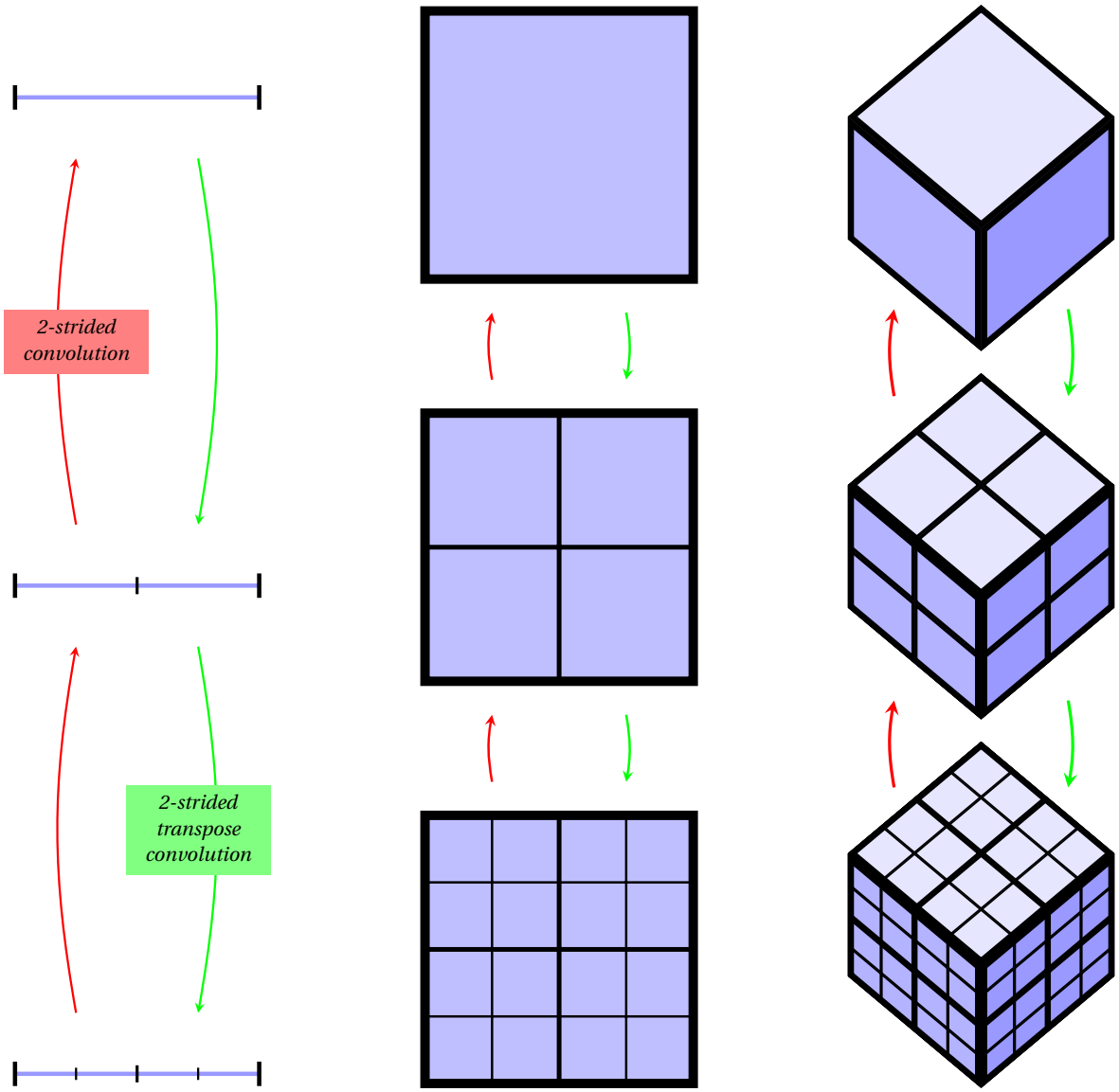


FIGURE 3.10 – Représentation de la subdivision de l'espace en 1D (à gauche), 2D (milieu) et 3D (à droite). La première ligne représente l'intégralité de l'espace (la racine de l'arbre), et à chaque ligne inférieure les portions d'espace de la ligne supérieure sont divisées en 2 selon chaque dimension.

la feuille comme la réflectance ou une couleur RGB.

Pour calculer le vecteur de caractéristiques x_d de taille N_d au nœud $\mathcal{N}$ à profondeur d, on récupère les 8 vecteurs de caractéristiques de taille N_{d-1} des 8 fils du nœud (à profondeur $d-1$), on les regroupe en un échantillon x_{d-1} de taille $N_{d-1} \times 2 \times 2 \times 2$. Si un des fils n'existe pas, on remplace son vecteur de caractéristiques par un vecteur de même taille rempli de 0. On applique alors l'opération de convolution à pas `Conv(`N_d`,(`$2 \times 2 \times 2$`),s=2,p=0)`, puis on peut appliquer autant de convolutions $1 \times 1 \times 1$ que l'on veut (ces opérations n'utilisant pas d'autres informations que celles locales au nœud en question).

Ainsi le signal se propage des feuilles jusqu'à la profondeur maximale où des opérations sont définies par le réseau. Cette profondeur est égale au nombre de convolutions à pas du réseau (c'est-à-dire au nombre de couches de sous-échantillonnage/agrégation). Une fois cette profondeur atteinte, on propage l'information vers les feuilles grâce à la partie « déconvolutionnelle » du réseau (constituée de convolutions transposées).

La figure 3.11 page ci-contre représente un *octree* vu en 1D pour la simplicité.

Par exemple supposons que l'*octree* en figure 3.11 page suivante soit déjà construit et sémantisé grâce à un réseau de segmentation composé de 2 convolutions $2 \times 2 \times 2$ à pas 2 comme celui

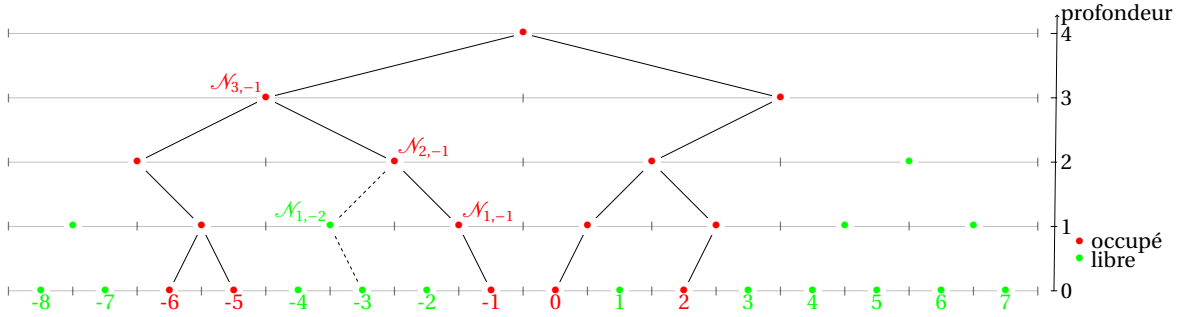


FIGURE 3.11 – Structure de l'arbre d'un *octree* vu en 1D pour la simplicité. Ici les quatre feuilles -6, -5, -1, 0 et 2, sont occupées (contiennent au moins un point). Les • représentent les éventuels nœuds de l'arbre et les segments horizontaux centrés autour d'un • représentent l'espace que le nœud correspondant discrétise. Seuls les • sont effectivement instanciés car contiennent au moins un point. Les branches en pointillés représentent les connexions qui seraient ajoutées si on ajoutait un point dans la feuille -3.

décrit en figure 3.12. Il y a donc des features dans tous les nœuds aux profondeurs 0, 1 et 2.

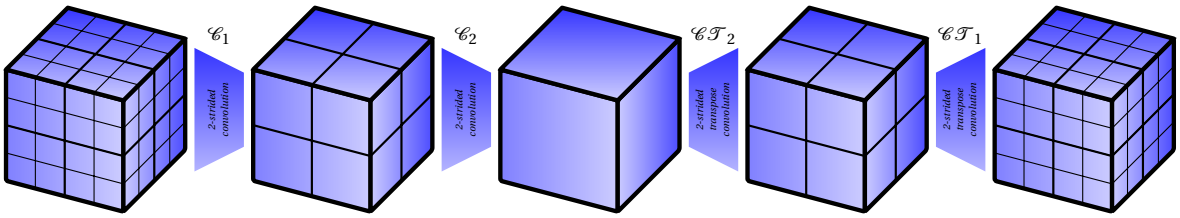


FIGURE 3.12 – Un réseau convolutionnel 3D de segmentation composé de 2 convolutions $2 \times 2 \times 2$ à pas 2 et de 2 convolutions transposées $2 \times 2 \times 2$ à pas 2.

Décrivons les opérations réalisées pour ajouter un point dans la feuille -3 et pour mettre à jour la classification de la partie de l'arbre modifiée, successivement :

— on vérifie que le point appartient à la racine, sinon on augmente la racine de profondeur (et donc de taille) autant de fois que nécessaire pour qu'elle contienne le point,

— on ajoute le point à la racine, qui va le propager à son unique fils qui le contient (ici le nœud $\mathcal{N}_{3,-1}$), récursivement jusqu'à ce que le point arrive dans une feuille. Ici le chemin existe jusqu'à la profondeur 2 (jusqu'au nœud $\mathcal{N}_{2,-1}$), après les nœuds $\mathcal{N}_{1,-2}$ et la feuille -3 sont crées,

— dans la feuille -3, un vecteur de features est créé en fonction du type d'information prise en entrée du réseau (binaire, reflectance, RGB...) et apportée par le point qu'on ajoute,

— puis le vecteur de features du nœud parent $\mathcal{N}_{1,-2}$ à profondeur 1 est calculé en appliquant la convolution $\mathscr{C}_1$ (la première convolution $2 \times 2 \times 2$ à pas 2 du réseau décrit en figure 3.12) au tenseur composé des features des feuilles -3 et -4.

— le même processus est appliqué pour monter jusqu'à la profondeur maximale du réseau (ici 2), donc on utilise la convolution $\mathscr{C}_2$ pour obtenir les features au nœud $\mathcal{N}_{2,-1}$,

— il reste alors à propager les features vers les feuilles en utilisant cette fois des convolutions transposées $2 \times 2 \times 2$ à pas 2, jusqu'à arriver aux feuilles. La convolution transposée $\mathscr{CT}_2$ propage l'information du nœud $\mathcal{N}_{2,-1}$, vers les nœuds $\mathcal{N}_{1,-2}$ et $\mathcal{N}_{1,-1}$. Puis la convolution transposée $\mathscr{CT}_1$ propage l'information du nœud $\mathcal{N}_{1,-2}$ (respectivement $\mathcal{N}_{1,-1}$), vers la feuille -3 (et respectivement la feuille -1),

— enfin pour inférer la classe de chaque feuille, une convolution $1 \times 1 \times 1$ (ici équivalent d'une couche FC) est appliquée aux features (ici on infère donc la classe des feuilles -3 et -1).

Cette approche permet donc de mettre à jour la sémantique de la carte dès qu'un point est ajouté à la structure

Cette structure a également l'avantage d'être dynamique, elle permet d'ajouter des points à la volée et de calculer à la volée les nouveaux vecteurs de features dans chaque nœud modifié par l'ajout. Elle permet également d'ajouter des points qui ne tombent pas dans l'espace représenté par la racine de l'arbre, il suffit alors de sur-élever autant de fois que nécessaire la racine (c'est à dire de créer un nœud représentant un espace deux fois plus grand dont un des fils est l'ancienne racine).

Cette structure permet au choix :

— d'utiliser un réseau convolutionnel 3D entrainé sur des grilles denses,

— d'entrainer un réseau directement sur la structure d'octree.

Cette structure laisse également la possibilité de réaliser l'inférence de réseaux de segmentation multi-échelles comme décrits en chapitre 2 page 39. De tels réseaux ne partent alors pas uniquement des feuilles, mais peuvent commencer déjà à une profondeur ≤ 1 et permettent alors de s'approcher plus de la racine dans l'arbre pour ajouter plus de contexte. Mais cette approche devra être développée dans de futurs travaux.

3.3.2 Une variante de l'*octree* qui permet d'adapter des réseaux convolutionnels plus variés

La structure décrite en section précédente n'autorise pas de convolutions $3\times3\times3$ ou de convolutions $2\times2\times2$ à pas 1, or ces convolutions permettent d'augmenter le champ réceptif effectif d'un réseau, alors qu'en l'état le vecteur de features calculé en un nœud est obtenu uniquement avec des informations provenant de ses sous-nœuds et donc de l'espace que ce nœud représente, mais ne permet pas d'agréger plus d'information contextuelle venant des nœuds voisins.

En l'état, pour réaliser des convolutions $3\times3\times3$ et les convolutions $2\times2\times2$ à pas 1 il faut aller chercher les features chez des nœuds voisins, ce qui peut nécessiter de remonter tout en haut de l'arbre (cf point en -1 et 0 qui sont voisins sur la figure 3.11 page précédente).

Une solution serait de stocker à chaque nœud un pointeur vers chaque nœud voisin (nécessaire pour faire une convolution $3\times3\times3$) qui n'est pas un fils direct, cela demande donc :

— en 1D : stocker un seul pointeur,

— en 2D : stocker 5 pointeurs,

— en 3D : stocker 19 pointeurs,

En plus de prendre 3 fois plus de place en mémoire, ceci pose le problème d'aller chercher ces nœuds voisins. En effet à la création d'un nœud le seul moyen de savoir si les nœuds voisins existent est de remonter dans l'arbre (potentiellement jusqu'à la racine).

Dans l'état de l'art pour *O-CNN* [P.-S. WANG, Yang LIU et collab., 2017] utilise une *hash-map* pour conserver les voisinages, mais cette approche n'autorise pas la structure à être dynamique.

On propose donc une structure légèrement modifiée de l'*octree*. Elle ne respecte plus l'inclusion des fils dans les parents et prend un peu plus de place en mémoire, mais permet de réaliser des convolutions $3\times3\times3$ et les convolutions $2\times2\times2$ à pas 1 aussi simplement que les convolutions $2\times2\times2$ à pas 2 pour la structure d'*octree* définie en section précédente.

Chaque nœud a 27 fils en 3D (3 fils en 1D, 9 fils en 2D) au lieu de 8 pour un *octree* (2 en 1D et 4 en 2D). Contrairement à l'*octree*, les fils d'un nœud ne sont pas exclusifs (sauf le fils central), ils sont partagés avec les nœuds voisins (cf figure 3.13 page ci-contre) :

— en 1D, le fils central est exclusif, et les fils de gauche et de droite sont partagés respectivement avec les nœuds voisins (à profondeur d) de gauche et de droite,

— en 2D, le fils central est exclusif, les fils aux arètes sont partagés avec 1 nœud voisin à profondeur d, et les fils aux coins/sommets sont partagés avec 3 nœuds voisins à profondeur d,

— en 3D, le nœud a un fils central qui lui est exclusif, il a un fils à chacune de ses faces, ces fils sont partagés avec 1 nœud voisin à profondeur d, les fils aux arètes sont partagés avec

3 nœuds voisins à profondeur d, et les fils aux coins/sommets sont partagés avec 7 nœuds voisins à profondeur d,

Une façon de se représenter cela est d'imaginer que la grille à l'échelle d est décalée par rapport à la grille à l'échelle $d-1$ d'un demi voxel selon chaque dimension. La figure 3.13 représente en 1D la structure de l'arbre ainsi que l'espace discrétisé par chaque nœud.

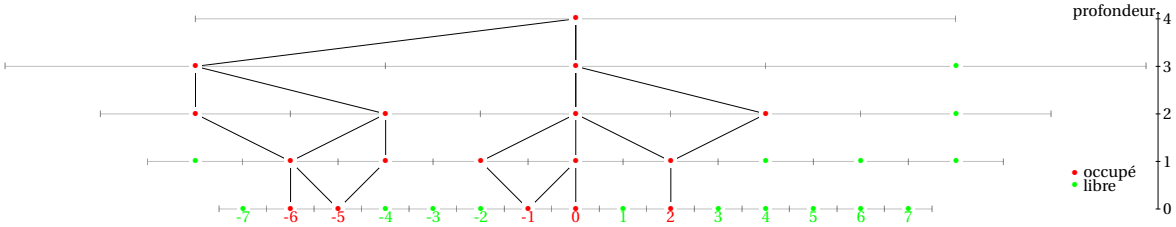


FIGURE 3.13 – Structure de *ConvTree* vu en 1D pour la simplicité. Ici les trois feuilles -6, -5, -1, 0 et 2 sont occupées (contiennent au moins un point). Les • représentent les éventuels nœuds de l'arbre et les segments horizontaux centrés autour d'un • représentent l'espace que le nœud correspondant discrétise. Seuls les • sont effectivement instanciés car contiennent au moins un point.

Pour réaliser un convolution $3\times3\times3$ à pas 1 en un nœud $\mathcal{N}$ de profondeur d par exemple, il suffit de "demander" au(x) nœud(s) parent(s) (de profondeur $d+1$) les vecteurs de features situés dans chacun des nœuds voisins de $\mathcal{N}$ de profondeur d ces derniers étant tous fils d'un des parents de $\mathcal{N}$.

Ainsi, pour réaliser une convolution $3\times3\times3$ en un nœud, il ne faut jamais aller chercher d'information à une distance supérieure à 2, contrairement au cas de l'*octree* où le temps d'accès à l'information nécessaire ne serait pas déterministe (dépendant de la structure de l'arbre localement, et pouvant aller jusqu'à deux fois la profondeur de l'arbre).

Ici il faut faire attention à faire un parcours en largeur de l'arbre (pas en profondeur) pour s'assurer avant de passer à la profondeur suivante ($d+1$ en phase de *downsampling* et $d-1$ en phase de *upsampling*) que tous les vecteurs de features à profondeur d ont bien étés mis à jour.

On propose d'appeler cette structure *ConvTree* puisque c'est une structure d'arbre qui permet de faire tout type de convolutions de noyau de taille $k\times l\times m$ avec $k,l,m\leq3$ et de pas 1 ou 2, en particulier les convolutions $3\times3\times3$ qui sont de loin les plus utilisées dans les réseaux convolutionnels de l'état de l'art.

La structure de *ConvTree* est une amélioration potentielle de la structure d'*octree*, mais nous n'avons pas encore eu le temps de l'implémenter et de la tester. Nous ne testerons donc que la structure d'*octree* en section 3.4 page suivante.

3.4 Résultats de Segmentation Sémantique en Temps-Réel

On présente ici les résultats d'amélioration de temps de calculs obtenus grâce à la méthode proposée en 3.3 page 97. Comme la classification est la même, nous ne ferons pas de comparaison des taux d'erreurs de classification.

Pour ces expérimentations, on entraine un réseau convolutionnel 3D mono-échelle de segmentation sur Paris-Lille-3D avec le même protocole que celui décrit en section 2.4

L'architecture choisie est une succession de convolutions $2 \times 2 \times 2$ à pas 2 et de convolutions $1 \times 1 \times 1$ dans la phase descendante, puis une succession de convolutions transposées $2 \times 2 \times 2$ à pas 2 et de convolutions $1 \times 1 \times 1$ dans la phase montante. La figure 3.14 décrit plus précisément l'architecture du réseau utilisé.

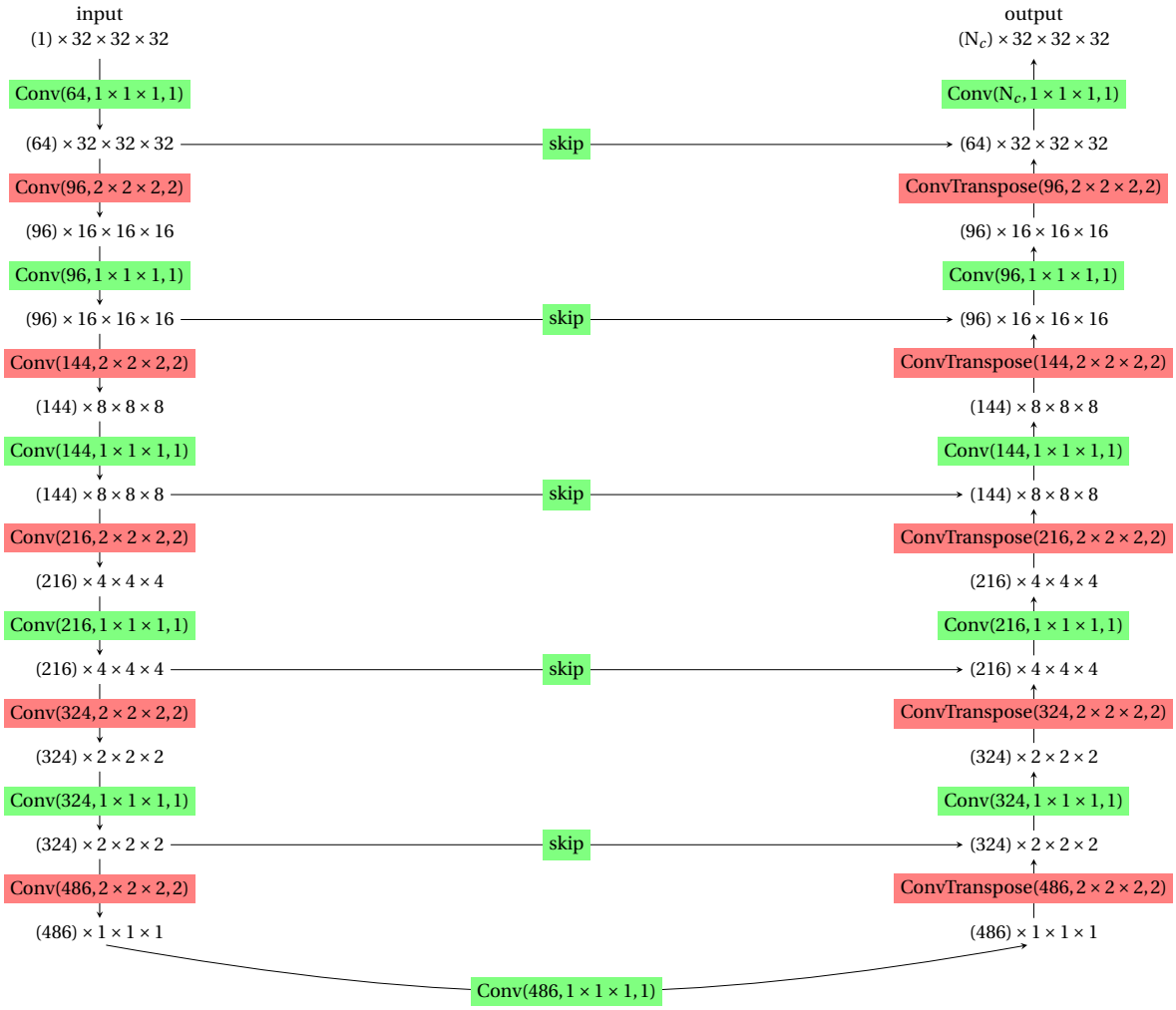


FIGURE 3.14 – Architecture du réseau dense utilisé sur un *octree*.

3.4.1 Inférence à la volée

On veut ici évaluer la capacité de la structure d'*octree* à intégrer et à classifier à la volée des nuages acquis par un robot ou un véhicule autonome, en particulier, on mesure le temps d'intégration d'une frame de Velodyne HDL-32e (environ 70000 points toutes les 0,1 s, un exemple de *frame* est montré en figure 3.15 page suivante). Comme le *timestamp* de chaque point est fourni dans le jeu de données Paris-Lille-3D, on évalue cette tâche sur les nuages de test qui contiennent 10M points chacun, mais cette fois en intégrant successivement les portions de nuages acquises toutes les 0,1 s. Les résultats sont donnés en tableau 3.1 page ci-contre. On évalue la méthode avec

les convolutions réalisées sur CPU ou sur GPU, la structure de l'*octree* étant elle toujours gérée par le CPU.

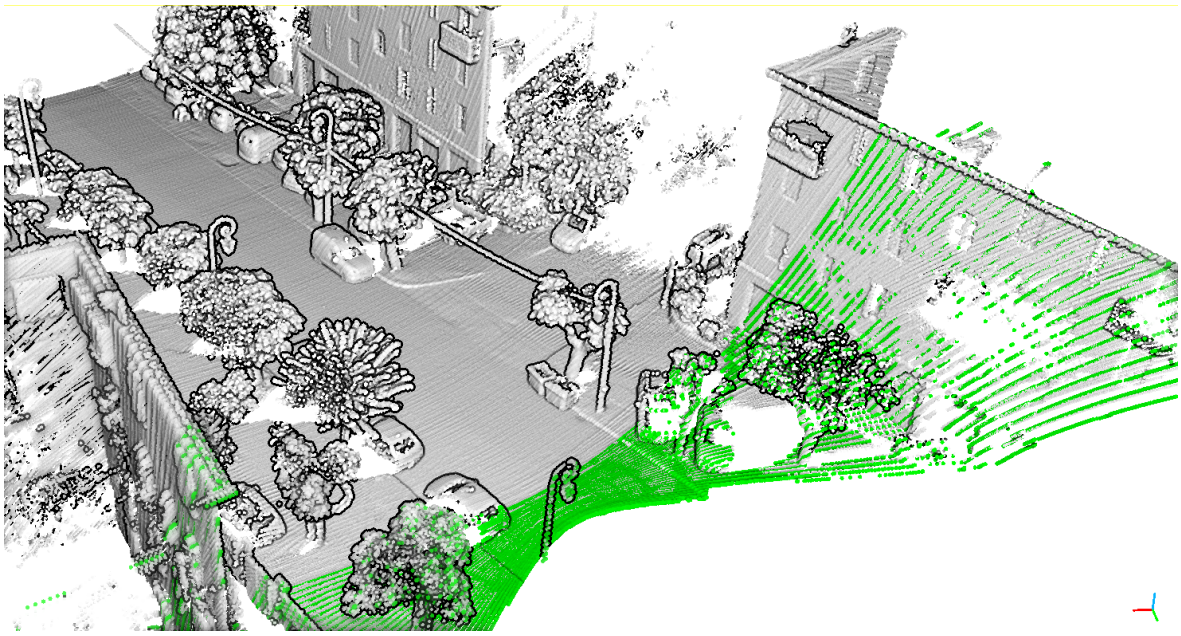


FIGURE 3.15 – Nuage de points acquis par MLS. Les points verts représentent les points de la dernière *frame* ajoutés à la carte.

Pas de discrétisation	réseau dense	octree (sur CPU)	octree (sur GPU)
voxels de 5 cm	2.04 s	39.5 s	68.6 s
voxels de 10 cm	0.79 s	16.3 s	28.8 s
voxels de 20 cm	0.16 s	6.24 s	11.2 s

TABLEAU 3.1 – Comparaison de temps d'ajout et d'inférence d'une *frame* de Velodyne entre un réseau de segementation dense et sur un *octree*. Les temps obtenus pour le réseau dense ne comprennent que le temps d'inférence du réseau, mais pas le temps de reconstruction de la structure de recherche de voisinages indipensable pour générer les échantillons.

On observe déjà que la méthode classique n'atteint pas le temps réel (à 0,1 s par frame) et est à un ordre de grandeur de l'atteindre pour des voxels de 10 cm (qui est le pas de discrétisation optimal pour les performances de classification d'un réseau mono-échelle).

Pour notre approche basée *octree*, on observe que :

— les temps de calculs sont meilleurs sur CPU, alors que le GPU est en théorie plus adapté à faire des convolutions, ceci s'explique par deux raisons :

  — quand on crée une feuille, il faut transférer l'information portée par les points (réflectance, couleur...) de la mémoire CPU à la mémoire GPU, ce qui prend un temps non négligeable,

  — les implémentations de convolutions sur GPU sont optimisées pour être réalisées sur des images ou grilles entières (ici on le fait uniquement sur de très petites portions d'une grille de taille $2 \times 2 \times 2$) et sur des *batchs*,

— on est encore loin du temps réel, environs deux ordres de grandeurs trop lent. Notre implémentation requiert encore d'être optimisée.

3.4.2 Inférence d'une scène complète

Notre approche par structure d'*octree* peut également être utilisée pour faire la segmentation sémantique d'une scène complète en donnant directement l'intégralité du nuage de points. On compare ici les temps d'inférence avec la méthode classique qui consiste à parcourir tout le nuage et d'inférer la classe des points grâce à un réseau dense. Pour ce faire on mesure et moyenne les temps sur les trois nuages de test du benchmark Paris-Lille-3D qui contiennent chacun 10 millions de points. Les résultats sont présentés dans le tableau 3.2.

Taille des voxels	réseau sur grilles dense	sur *octree*
5 × 5 × 5 cm	448 s	4080 s (×9.1)
10 × 10 × 10 cm	173 s	1320 s (×7.6)
20 × 20 × 20 cm	36.1 s	150 s (×4.2)

TABLEAU 3.2 – Comparaison de temps d'inférence moyen d'une scène complète (de 10 millions de points) entre un réseau de segmentation sur grille dense et sur un *octree*. Les temps réalisés par le réseau dense ne comprennent pas les temps de construction de la structure de recherche de voisinages (ici un *kdtree*).

On observe que la méthode classique est encore meilleure d'un peu moins d'un ordre de grandeur par rapport à notre approche par *octree*. De plus aucune des deux méthodes n'est proche du temps réel puisque ces scènes ont été acquises en environ 22 s.

Les temps de calculs sur structure d'*octree* réalisés par les méthodes similaires de l'état de l'art sont proches de ceux que l'on veut obtenir, c'est-à-dire le temps-réel. En particulier, [LEI et collab., 2019] présente des temps construction de l'*octree* et d'inférence de 34.1 ms pour 10k points. Pour comparaison, notre implémentation prend 5.50 s pour réaliser la même tâche.

Ceci laisse penser qu'une implémentation optimisée de notre méthode obtiendrait des temps de calculs au moins aussi bons qu'un réseau sur grille dense.

3.5 Platforme d'Expérimentation LiDAR Aérienne

Nous présentons dans cette section une plateforme expérimentale de LiDAR porté par drone pour l'évaluation d'algorithmes de perception temps-réel sur système embarqué.

3.5.1 Enjeu et Contraintes

Le système conçu doit être capable d'estimer sa localisation et de construire la carte de son environnement par exemple grâce à un algorithme de SLAM en temps-réel et en embarqué. Pour cela il doit être équipé de tous les capteurs proprioceptifs (comme une centrale inertielle) et extérioceptifs (comme un LiDAR) nécessaires à ces tâches, mais il doit aussi être équipé de la puissance de calcul qui permet de les réaliser. En particulier, la sémantisation qui utilise des méthodes d'apprentissage profond dans notre cas peut demander beaucoup de puissance de calcul, c'est pourquoi on utilisera un ordinateur embarqué équipé d'un petit GPU.

3.5.2 Scénario

Le but de ces expérimentations est de créer des données de test pour les algorithmes développés en section 3.3 page 97 pour cette thèse.

Le but étant que le système (drone ou véhicule terrestre) soit capable de comprendre son environnement en temps réel à partir des données nuage de points 3D acquis par LiDAR. L'application la plus évidente est le véhicule autonome.

Dans ce but, le LiDAR doit être placé de telle sorte à avoir le meilleur champ de vision pour anticiper et identifier les potentiels obstacles sur sa trajectoire ainsi que les autres individus, objets, utilisateurs ou véhicules dans son espace de navigation. Ce point est abordé en section 3.5.4 page suivante. Pour atteindre ce but, et dans l'état actuel des technologies LiDAR multi-fibres rotatifs à 360°, il faudrait dans l'idéal plusieurs LiDARs (au moins 3) positionnés de telle sorte à "voir" tout autour de soi.

3.5.3 Le Système d'Acquisition Aérien

Pour remplir ces contraintes nous avons choisi un drone professionnel capable de porter plusieurs kilogrammes, puisque le LiDAR (un Velodyne HDL-32e ou VLP-16) pèse déjà 1 kg. De plus il faut porter la batterie permettant d'alimenter le LiDAR et l'ordinateur embarqué. Pour assurer la légèreté et la rigidité de la nacelle nous avons utilisé des plaques de carbone et le dessin de la nacelle a été réalisé par Joël Senporauca du *Centre de Robotique*.

La nacelle contient les composants :

— un ordinateur embarqué Nvidia Jetson TX1 avec un SSD pour l'acquisition et le traitement à la volée des données,

— un LiDAR Velodyne HDL-32e pour la première version, puis un Velodyne VLP-16 plus léger pour la deuxième version,

— une centrale inertielle de milieu de gamme,

— une caméra pour coloriser les nuages ou pour faire de l'odométrie visuelle,

La figure 3.16 page suivante présente les première et deuxième versions de la nacelle.

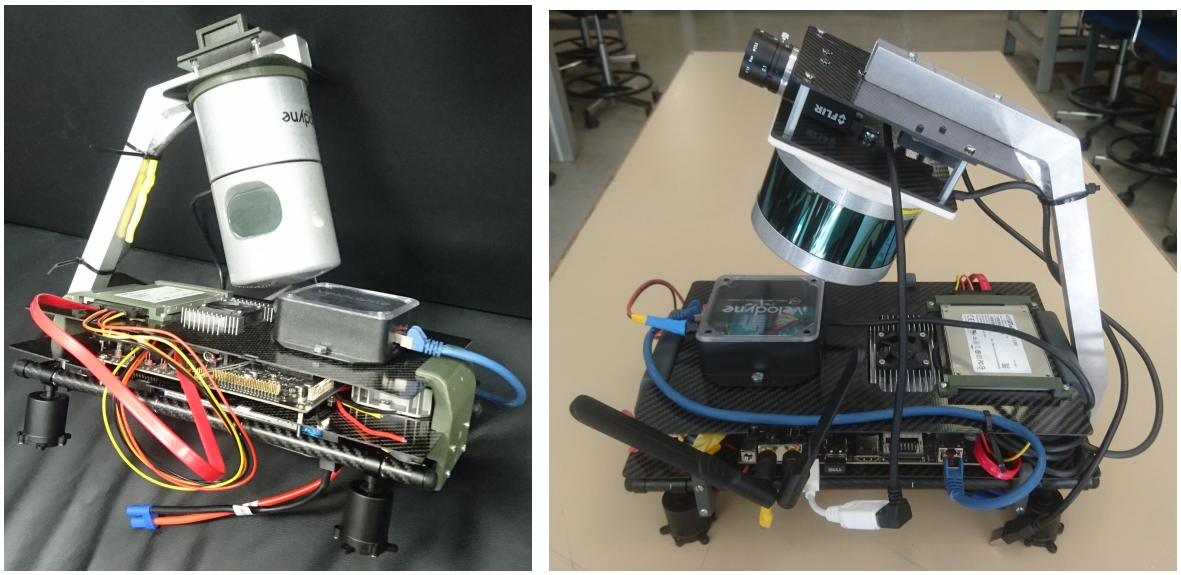


FIGURE 3.16 – Les première (à gauche) et deuxième (à droite) versions de la nacelle.

La nacelle complète pèse environ 3 kg, ce qui est assez léger pour le drone choisi qui peut porter jusqu'à 5 kg.

3.5.4 Positionnnement de la Nacelle

La nacelle a pour vocation à pouvoir être montée sur notre drone (DJI M600) ou sur notre véhicule d'acquisition expérimental (L3D2), voir 3.17.

FIGURE 3.17 – Le drone du centre de robotique : un DJI M600 (à gauche), le véhicule expérimental : L3D2 (à droite)

La configuration qui semble optimale pour nos cas d'expérimentation est celle décrite en figure 3.18 page ci-contre. Le LiDAR est penché dans la direction du mouvement de 20°. C'est un compromis entre deux configurations :

— le LiDAR n'est pas penché (axe de rotation vertical), alors le champs de vision est optimal pour un véhicule terrestre qui a besoin de voir devant lui,

— le LiDAR est penché à 90° dans la direction du mouvement (axe de rotation horizontal), le champs de vision est optimal pour voir le sol et balayer l'environnement donc plutôt utilisé pour faire de la cartographie.

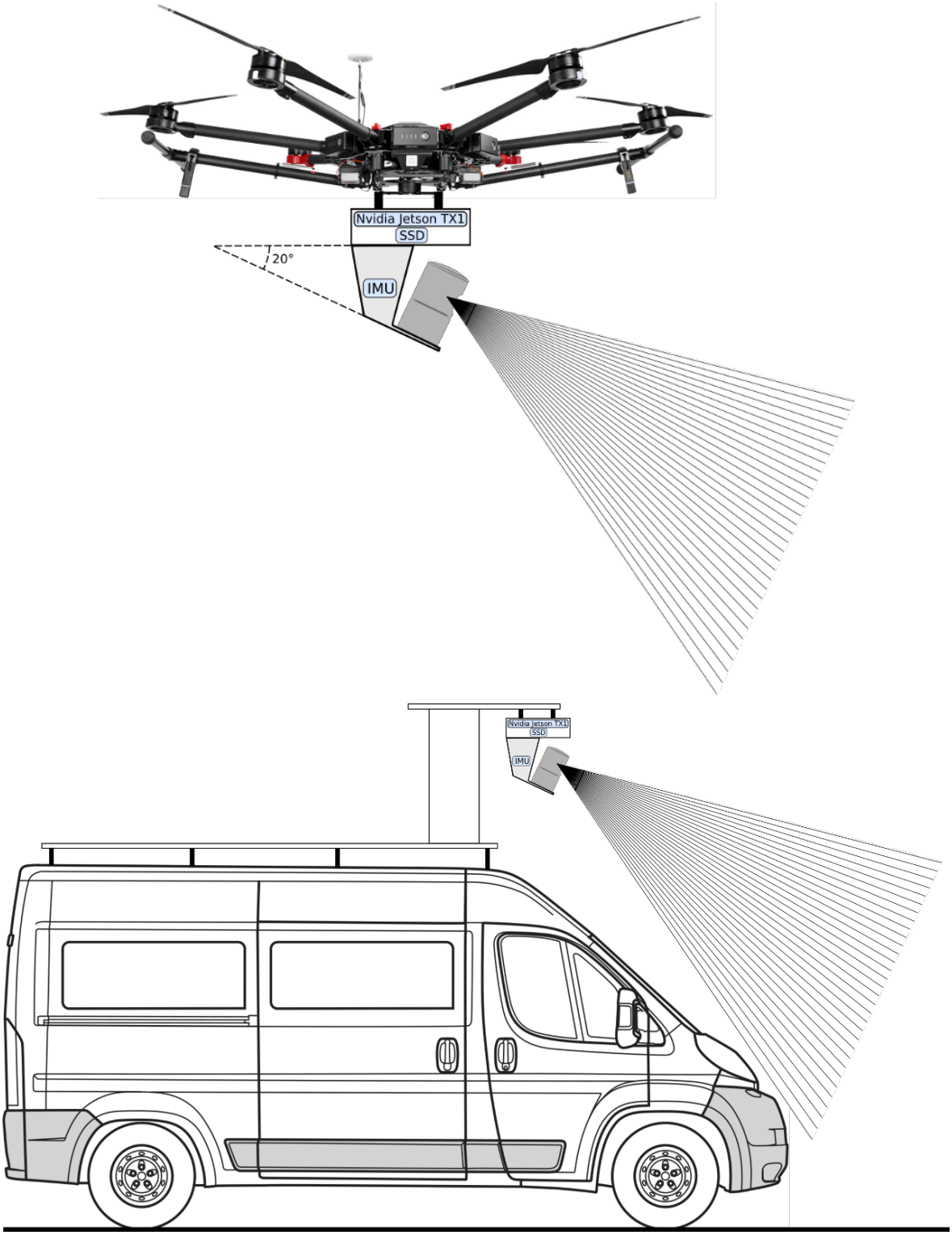


FIGURE 3.18 – Configuration de la nacelle sur le drone (en haut), sur le L3D2 (en bas)

En figure 3.19 page suivante, on peut visualiser le champ de vision du LiDAR pour les configurations où le drone vol à 10m de hauteur, ou bien placé sur le L3D2. On remarque qu'un angle moins important aurait permis à la configuration "L3D2" de voir plus en avant, mais aurait empêché au drone de voir suffisament le sol.

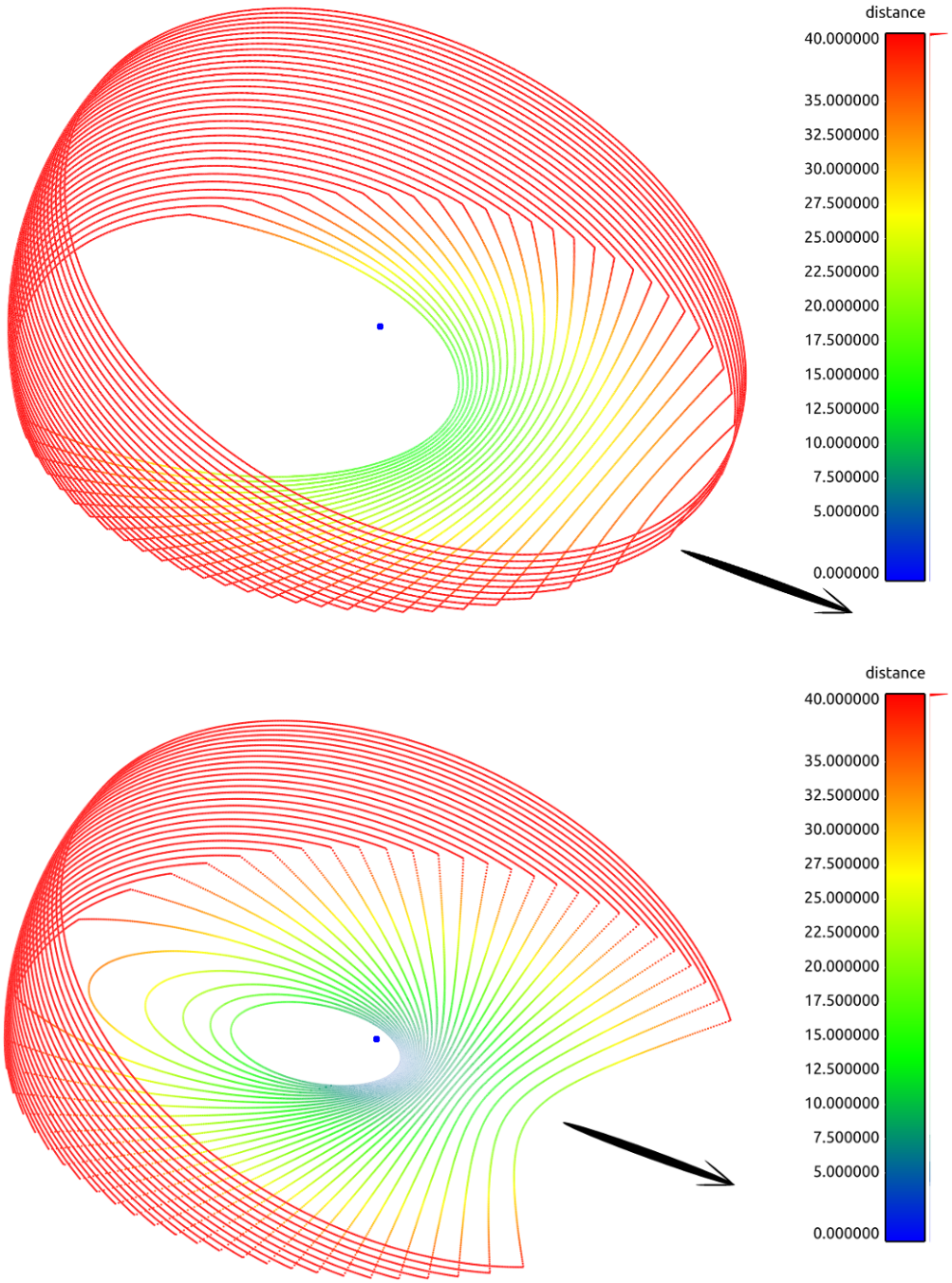


FIGURE 3.19 – Champ de vision du capteur, sur le drone à 10m de hauteur (en haut), sur le L3D2 (en bas). Dans les deux cas le point bleu représente la position du LiDAR.

3.5.5 Données acquises

La législation française n'autorisant pas le survol de zones habitées, il a été difficile de mettre en place un scénario expérimental. Nous avons réalisé les deux seuls vols expérimentaux au-dessus d'une ferme familiale près d'Angers.

Les algorithmes présentés en section 3.3 page 97 n'étant pas encore implémentés au moment des expérimentations, nous n'avons pas pu les tester en conditions réelles. Cependant les acquisitions ont été enregistrées avec ROS ce qui permet de les rejouer pour évaluer nos algorithmes.

Les nuages présentés en figure 3.20 page suivante on été obtenus en recalant des *frames* de Velodyne consécutifs par méthode d'IMLS SLAM [J.-E. DESCHAUD, 2018]. Les nuages obtenus sont similaires à des nuages obtenus par MLS terrestre. Ils présentent une densité similaire et on peut

distinguer les objets aussi facilement. Cependant, le point de vue étant plus élevé les objets sont complets sur leur partie haute, par exemple les toitures sont presque complètes contrairement à une scène scannée par MLS terrestre.

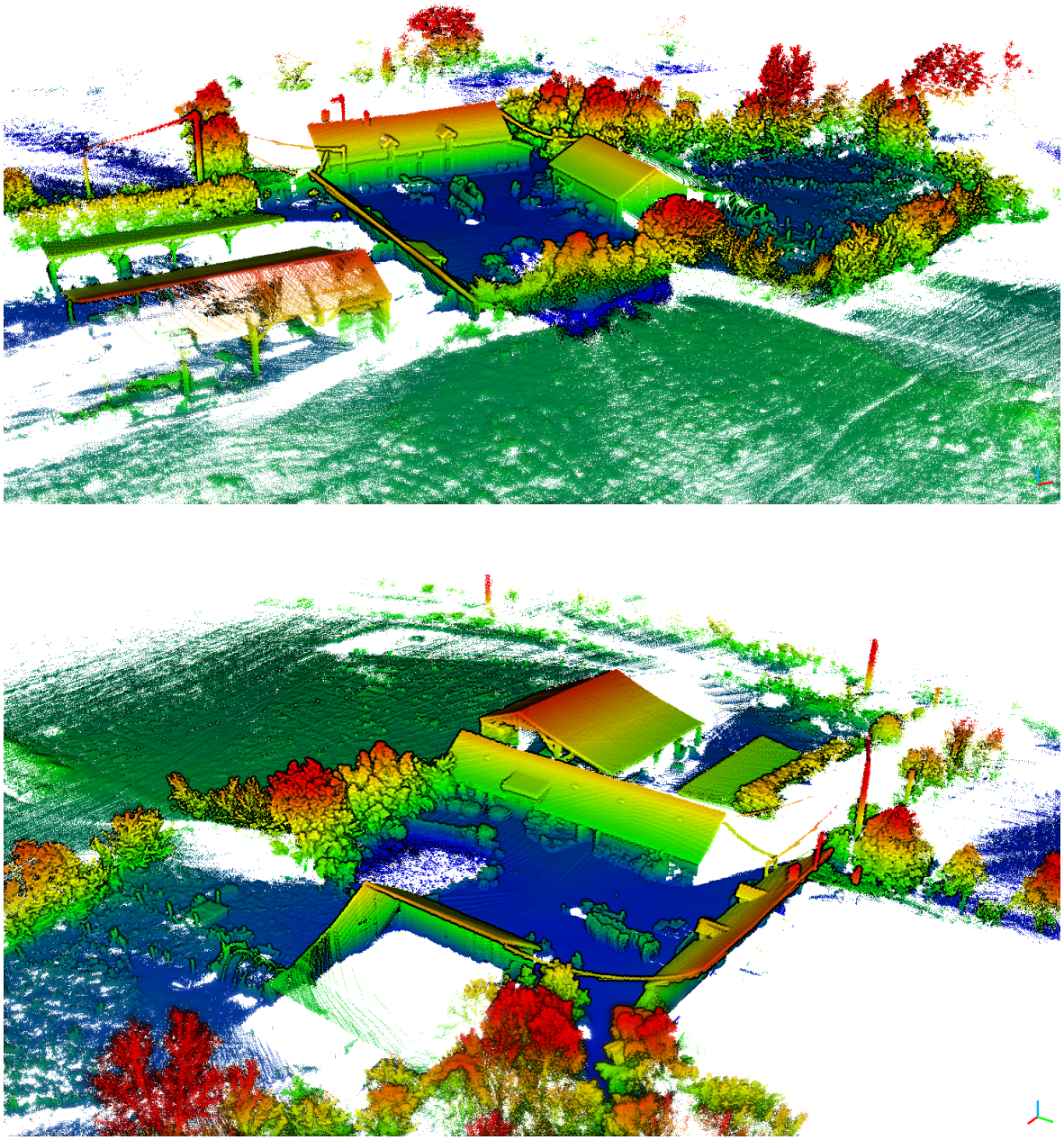


FIGURE 3.20 – Nuage acquis grâce à la plateforme expérimentale aérienne (colorisé par élévation verticale).

3.6 Perspectives et Conclusion

On a donc développé dans ce chapitre une structure d'*octree* ainsi qu'une variante permettant l'inférence de réseaux convolutionnels 3D sur des données éparses comme des nuages de points 3D.

Les structures développées dans ce chapitre offrent plusieurs perspectives d'amélioration et d'adaptation à d'autres tâches :

— on peut imaginer mettre à jour l'état des nœuds entre *occupé*, *libre* ou *inconnu*, ce qui est fait dans *OctoMap* [WURM et collab., 2010] en fonction du trajet du rayon laser. Ceci pourrait être appris grâce à un classificateur en fonction des features locales au nœud,

— on pourrait utiliser des réseaux récurrents à chaque noeud pour mettre à jour les features en fonction certes des enfants ou parents, mais aussi en fonction des features déjà calculées précédemment au nœud. Cette approche serait similiare à *Recurrent-OctoMap* [L. SUN et collab., 2018],

— ces structures ne sont utilisées ici que pour faire de la segmentation sémantique, mais on peut par exemple utiliser les descripteurs locaux à chaque nœud pour faire du *matching* (à plus ou moins haut niveau dans l'arbre), ce qui peut être utile pour de la fermeture de boucle en *SLAM* par exemple.

Cependant pour obtenir des résultats expérimentaux qui montrent que l'on peut réellement atteindre le temps-réel, il faudra faire un effort sur l'implémentation de l'*octree* qui, par faute de temps, n'a pas pu être optimisé. D'un autre côté, l'implémentation des réseaux denses utilise des librairies de convolutions très optimisées. On remarquera que l'implémentation actuelle peut être améliorée sur plusieurs points : 1. l'utilisation d'un langage compilé (`C++`) au lieu d'un langage interprété (`Python`), 2. la suppression des nœuds (zones de la carte) trop éloignés du robot, 3. tirer parti de l'inférence optimisée par *batch* des bibliothèques d'apprentissage profond.

On a également présenté une plateforme d'expérimentation avec LiDAR porté par drone pour l'évaluation en conditions réelles des méthodes de segmentation sémantique en temps-réel sur système embarqué.

Références

ALYAMKIN, Sergei, Matthew ARDI, Achille BRIGHTON, Alexander C BERG, Yiran CHEN, Hsin-Pai CHENG, Bo CHEN, Zichen FAN, Chen FENG, Bo FU et collab. (2018). « 2018 Low-Power Image Recognition Challenge ». In : *arXiv preprint arXiv:1810.01732* (cf. p. 89).

CAREY, Stephen J, Alexey LOPICH, David RW BARR, Bin WANG et Piotr DUDEK (2013). « A 100,000 fps vision sensor with embedded 535GOPS/W 256× 256 SIMD processor array ». In : *VLSI Circuits (VLSIC), 2013 Symposium on*. IEEE, p. C182-C183 (cf. p. 90).

CHEN, Jianing, Stephen J CAREY et Piotr DUDEK (2018). « Scamp5d Vision System and Development Framework ». In : *Proceedings of the 12th International Conference on Distributed Smart Cameras*. ACM, p. 23 (cf. p. 90).

CHEN, Jichao, Yijie ZENG, Siqi WANG, Soh Ling MIN et Guang-Bin HUANG (2018). « Octree-based Convolutional Autoencoder Extreme Learning Machine for 3D Shape Classification ». In : *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, p. 1-7 (cf. p. 93).

CHETLUR, Sharan, Cliff WOOLLEY, Philippe VANDERMERSCH, Jonathan COHEN, John TRAN, Bryan CATANZARO et Evan SHELHAMER (2014). « cudnn: Efficient primitives for deep learning ». In : *arXiv preprint arXiv:1410.0759* (cf. p. 89).

CHOLLET, Francois (juil. 2017). « Xception: Deep Learning With Depthwise Separable Convolutions ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 90, 138).

CINTRA, Renato J, Stefan DUFFNER, Christophe GARCIA et André LEITE (2018). « Low-Complexity Approximate Convolutional Neural Networks ». In : *IEEE Transactions on Neural Networks and Learning Systems* (cf. p. 92).

CURA, Remi, Julien PERRET et Nicolas PAPARODITIS (2018). « An octree cells occupancy geometric dimensionality descriptor for massive on-server point cloud visualisation and classification ». In : *arXiv preprint arXiv:1801.05038* (cf. p. 93).

DESCHAUD, Jean-Emmanuel (2018). « IMLS-SLAM: scan-to-model matching based on 3D data ». In : *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, p. 2480-2485 (cf. p. 108).

DUDEK, Piotr et Peter J HICKS (2005). « A general-purpose processor-per-pixel analog SIMD vision chip ». In : *IEEE Transactions on Circuits and Systems I: Regular Papers* 52.1, p. 13-20 (cf. p. 90).

FREEMAN, Ido, Lutz ROESE-KOERNER et Anton KUMMERT (2018). « EffNet: An Efficient Structure for Convolutional Neural Networks ». In : *arXiv preprint arXiv:1801.06434* (cf. p. 91).

HAN, Song, Huizi MAO et William J DALLY (2015). « Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding ». In : *arXiv preprint arXiv:1510.00149* (cf. p. 92).

HOWARD, Andrew G, Menglong ZHU, Bo CHEN, Dmitry KALENICHENKO, Weijun WANG, Tobias WEYAND, Marco ANDREETTO et Hartwig ADAM (2017). « Mobilenets: Efficient convolutional neural networks for mobile vision applications ». In : *arXiv preprint arXiv:1704.04861* (cf. p. 91).

HUANG, Gao, Shichen LIU, Laurens VAN DER MAATEN et Kilian Q WEINBERGER (2018). « Condensenet: An efficient densenet using learned group convolutions ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 2752-2761 (cf. p. 91).

KLOKOV, Roman et Victor LEMPITSKY (2017). « Escape From Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models ». In : *Proceedings of the IEEE International Conference on Computer Vision*, p. 863-872 (cf. p. 95, 96).

LEI, Huan, Naveed AKHTAR et Ajmal MIAN (fév. 2019). « Octree guided CNN with Spherical Kernels for 3D Point Clouds ». In : *arXiv e-prints*, arXiv:1903.00343, arXiv:1903.00343. arXiv : 1903.00343 [cs.CV] (cf. p. 95, 96, 104).

MA, Ningning, Xiangyu ZHANG, Hai-Tao ZHENG et Jian SUN (2018). « Shufflenet v2: Practical guidelines for efficient cnn architecture design ». In : *Proceedings of the European Conference on Computer Vision (ECCV)*, p. 116-131 (cf. p. 91).

NURVITADHI, Eriko, David SHEFFIELD, Jaewoong SIM, Asit MISHRA, Ganesh VENKATESH et Debbie MARR (2016). « Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC ». In : *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE, p. 77-84 (cf. p. 90).

OVTCHAROV, Kalin, Olatunji RUWASE, Joo-Young KIM, Jeremy FOWERS, Karin STRAUSS et Eric S CHUNG (2015). « Accelerating deep convolutional neural networks using specialized hardware ». In : *Microsoft Research Whitepaper* 2 (cf. p. 89, 90).

RASTEGARI, Mohammad, Vicente ORDONEZ, Joseph REDMON et Ali FARHADI (2016). « Xnor-net: Imagenet classification using binary convolutional neural networks ». In : *European Conference on Computer Vision*. Springer, p. 525-542 (cf. p. 91).

RIEGLER, Gernot, Ali Osman ULUSOY et Andreas GEIGER (2017). « OctNet: Learning Deep 3D Representations at High Resolutions ». In : *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, p. 6620-6629 (cf. p. 94).

SANDLER, Mark, Andrew HOWARD, Menglong ZHU, Andrey ZHMOGINOV et Liang-Chieh CHEN (jan. 2018). « MobileNetV2: Inverted Residuals and Linear Bottlenecks ». In : *arXiv e-prints*, arXiv:1801.04381, arXiv:1801.04381. arXiv : 1801.04381 [cs.CV] (cf. p. 91).

SENGUPTA, Sunando et Paul STURGESS (2015). « Semantic octree: Unifying recognition, reconstruction and representation via an octree constrained higher order MRF ». In : *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, p. 1874-1879 (cf. p. 93).

SIMONYAN, Karen et Andrew ZISSERMAN (sept. 2014). « Very Deep Convolutional Networks for Large-Scale Image Recognition ». In : *ArXiv e-prints*, arXiv:1409.1556, arXiv:1409.1556. arXiv : 1409.1556 [cs.CV] (cf. p. 90, 120).

SUN, Ke, Mingjie LI, Dong LIU et Jingdong WANG (2018). « IGCV3: Interleaved Low-Rank Group Convolutions for Efficient Deep Neural Networks ». In : *arXiv preprint arXiv:1806.00178* (cf. p. 91).

SUN, Li, Zhi YAN, Anestis ZAGANIDIS, Cheng ZHAO et Tom DUCKETT (2018). « Recurrent-OctoMap: Learning State-Based Map Refinement for Long-Term Semantic Mapping With 3-D-Lidar Data ». In : *IEEE Robotics and Automation Letters* 3.4, p. 3749-3756 (cf. p. 110).

SZEGEDY, Christian, Vincent VANHOUCKE, Sergey IOFFE, Jon SHLENS et Zbigniew WOJNA (2016). « Rethinking the inception architecture for computer vision ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2818-2826 (cf. p. 91).

TATARCHENKO, Maxim, Alexey DOSOVITSKIY et Thomas BROX (2017). « Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs ». In : *Proceedings of the IEEE International Conference on Computer Vision*, p. 2088-2096 (cf. p. 94, 95).

WANG, Chao, Lei GONG, Qi YU, Xi LI, Yuan XIE et Xuehai ZHOU (2017). « DLAU: A scalable deep learning accelerator unit on FPGA ». In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.3, p. 513-517 (cf. p. 90).

WANG, Peng-Shuai, Yang LIU, Yu-Xiao GUO, Chun-Yu SUN et Xin TONG (2017). « O-cnn: Octree-based convolutional neural networks for 3d shape analysis ». In : *ACM Transactions on Graphics (TOG)* 36.4, p. 72 (cf. p. 94, 100).

WANG, Peng-Shuai, Chun-Yu SUN, Yang LIU et Xin TONG (2018). « Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes ». In : *arXiv preprint arXiv:1809.07917* (cf. p. 94, 95).

WURM, Kai M, Armin HORNUNG, Maren BENNEWITZ, Cyrill STACHNISS et Wolfram BURGARD (2010). « OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems ». In : *In Proc. of the ICRA 2010 workshop*. Citeseer (cf. p. 93, 110).

XIE, Guotian, Jingdong WANG, Ting ZHANG, Jianhuang LAI, Richang HONG et Guo-Jun QI (2018). « IGCV 2: Interleaved Structured Sparse Convolutional Neural Networks ». In : *arXiv preprint arXiv:1804.06202* (cf. p. 91).

XIE, Saining, Ross GIRSHICK, Piotr DOLLÁR, Zhuowen TU et Kaiming HE (2017). « Aggregated residual transformations for deep neural networks ». In : *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, p. 5987-5995 (cf. p. 90, 138).

ZHANG, Ting, Guo-Jun QI, Bin XIAO et Jingdong WANG (2017). « Interleaved Group Convolutions ». In : *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, p. 4383-4392 (cf. p. 91).

ZHANG, Xiangyu, Xinyu ZHOU, Mengxiao LIN et Jian SUN (juil. 2017). « ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices ». In : *arXiv e-prints*, arXiv:1707.01083, arXiv:1707.01083. arXiv : 1707.01083 [cs.CV] (cf. p. 91).

Conclusion

Le domaine de la segmentation sémantique de nuages points 3D est en pleine effervescence entre d'un côté l'apparition de méthodes d'apprentissage profond qui battent l'état de l'art de la segmentation sémantique en image et offrent la promesse de réaliser des prouesses similaires en 3D, et d'un autre côté le développement des véhicules autonomes de plus en plus équipés de capteurs apportant une information 3D sous forme de nuages de points (LiDAR, caméra de profondeur...). Ces véhicules autonomes ont un besoin indispensable de méthodes de segmentation sémantique pour comprendre leur environnement.

Nous avons donc dans cette thèse essayé d'apporter des contributions aux méthodes de segmentation sémantiques de nuages de points 3D par apprentissage profond, d'abord sur des scènes complètes en *offline* dans le chapitre 2 page 39 puis pour la segmentation sémantique en temps réel en embarqué dans le chapitre 3 page 85. Mais comme toute méthode d'apprentissage profond requiert des jeux d'entraînement importants nous avons avant tout créé un jeu de données appelé Paris-Lille-3D permettant d'entraîner ces méthodes et présenté dans le chapitre 1 page 5.

Dans une première partie, après avoir étudié les différents jeux de données de scènes de nuages de points annotés existants, il nous est apparu qu'aucun jeu de données ne contenait assez de données ou des données suffisamment bien annotées. Nous avons donc réalisé et publié un jeu de données correspondant à des critères essentiels pour entraîner des méthodes de segmentation sémantique par apprentissage profond.

Dans une seconde partie, nous avons étudié les méthodes déjà existantes d'apprentissage profond pour la segmentation sémantique de scènes de nuages de points 3D. Au début de cette thèse il en existait encore très peu et avaient des résultats mitigés (n'ayant rien à voir avec les résultats de l'apprentissage profond en images). Mais ces méthodes n'explicitaient pas la façon d'entraîner des réseaux de neurones profonds sur des jeux de données de scènes complètement annotées. De plus ces méthodes n'utilisaient pas encore certaines améliorations connues en segmentation sémantique classiques comme l'utilisation de descripteurs multi-échelles. Nous avons donc proposé des solutions à ces problèmes :

— une méthode d'entraînement, qui diffère de la méthode standard où l'on parcourt l'intégralité du jeu de données à chaque *epoch*, et qui pallie au déséquilibre du nombre de points par classe inhérent à ce type de jeu de données,

— plusieurs méthodes permettant de fusionner des descripteurs à plusieurs échelles dans un réseau de neurones convolutionnel qui peut être appris de bout en bout.

Les méthodes proposées dans cette partie battent l'état de l'art de segmentation sémantique de scènes de nuages de points 3D par méthode de classification point par point. Ces méthodes, utilisées ici sur grilles voxeliques denses, peuvent également être utilisées avec des opérations de convolutions sur points.

Dans une troisième et dernière partie, dans le but de réduire les temps de calculs et de s'approcher de la segmentation sémantique en temps réel, nous avons proposé une méthode tirant profit du caractère éparse des nuages de points grâce à une structure d'*octree*, qui permet de réaliser les convolutions uniquement là où il y a de l'information contrairement aux réseaux convolutionnels sur grilles denses. Ces méthodes offrent des perspectives intéressantes pour réussir à réaliser la perception en temps-réel sur un ordinateur embarqué.

Plusieurs questions se posent encore sur la façon d'adapter des réseaux multi-échelles (bien

plus performants) pour pouvoir les utiliser sur un *octree*, ou comment implémenter efficacement un *ConvTree* pour être aussi efficace qu'un *octree*.

On peut imaginer que des structures similaires soient également utilisées pour réaliser des fermetures de boucles en SLAM, puisque l'information sémantique calculée à chaque nœud de l'*octree* peut être utilisée pour détecter des endroits similaires d'une carte.

Publications

Cette thèse a mené à plusieurs publications :

— Dans une revue à comité de lecture :

  — Xavier ROYNARD, Jean-Emmanuel DESCHAUD et collab. (2018b). « Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification ». In : *The International Journal of Robotics Research* 37.6, p. 545-557. DOI : 10.1177/0278364918767506. eprint : https://doi.org/10.1177/0278364918767506. URL : https://doi.org/10.1177/0278364918767506.

— Dans des conférences internationales :

  — Xavier ROYNARD, Jean-Emmanuel DESCHAUD et collab. (2016). « Fast and Robust Segmentation and Classification for Change Detection in Urban Point Clouds ». In : *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B3, p. 693-699. DOI : 10.5194/isprs-archives-XLI-B3-693-2016. URL : https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B3/693/2016/,

  — Xavier ROYNARD, Jean-Emmanuel DESCHAUD et collab. (juin 2018c). « Paris-Lille-3D: A Point Cloud Dataset for Urban Scene Segmentation and Classification ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*,

  — Xavier ROYNARD, Jean-Emmanuel DESCHAUD et collab. (oct. 2018b). « Classification of Point Cloud for Road Scene Understanding with Multiscale Voxel Deep Network ». In : *10th Workshop on Planning, Perception and Navigation for Intelligent Vehicles, PPNIV'18*,

  — Marie-Anne MITTET et collab. (2016). « Experimental Assessment of the Quanergy M8 LiDAR Sensor ». In : *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B5, p. 527-531. DOI : 10.5194/isprs-archives-XLI-B5-527-2016. URL : https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B5/527/2016/.

— Dans une conférence nationale :

  — Xavier ROYNARD, Jean-Emmanuel DESCHAUD et collab. (2018a). « Classification de Scènes de Nuages de Points 3D par Réseau Convolutionnel Profond Voxelique Multi-échelles ». In : *RFIAP et CFPT 2018*

— Une version plus complète de l'article sur les réseaux multi-échelles est disponible sur arxiv, avec des expérimentations sur données d'intérieur :

  — Xavier ROYNARD, Jean-Emmanuel DESCHAUD et collab. (avr. 2018a). « Classification of Point Cloud Scenes with Multiscale Voxel Deep Network ». In : *arXiv preprint arXiv:1804.03583*,

— Ainsi que la création d'un benchmark publique de segmentation sémantique de scènes de nuages de points 3D acquis par MLS sur le site http://npm3d.fr/paris-lille-3d, dont le jeu d'entraînement est le jeu de données Paris-Lille-3D présenté en section 1.3 page 23.

Annexe A

État de l'Art de l'Apprentissage Profond en Classification et Segmentation Sémantique d'Images

Sommaire

A.1 Organisation de l'état de l'art

On réalise ici un état de l'art des méthodes de Deep-Learning pour la classification et segmentation d'images en s'intéressant dans l'ordre :

— aux définitions et notations des couches de base qui composent un réseau de neurones convolutionnel en section A.2 page suivante,

— aux premiers réseaux qui ont permis de battre l'état de l'art en classification d'images en section A.3 page 119

— aux méthodes qui ont permis par la suite des améliorations significatives de performances pour la tâche de classification, découpées en deux types de méthodes :

  - les méthodes applicables partout, qui peuvent être vues comme des bonnes pratiques dans la conception des blocs de base d'un réseau (non-linéarités ReLU, normalisation, initialisation, dropout...), cf sections A.4.3 page 127, A.4.2 page 125, A.5.1 page 131, A.5.2 page 134, A.5.3 page 135 et A.4.5 page 130.
  - les différentes architectures de réseaux (comment les connexions s'organisent, Inception/ResNet, Réseau dans réseaux, attention, DenseNet, SENet, Dual-Path-Network, NasNet, ...), cf sections A.6.2 page 139, A.6.1 page 137 et A.6.4 page 143.
- à certains types de tâches que l'on peut remplir, et les méthodes qui permettent de les accomplir, cf sections A.6.3 page 141 et A.6.5 page 144

A.2 Définitions et Notations

Un réseau de neurones convolutionnel profond est constitué d'une succession de couches qui transforment progressivement l'entrée du réseau (une image ou une grille voxelique) en images de moins en moins résolues mais contenant des informations à plus haut niveau d'abstraction, jusqu'à produire la sortie du réseau qui pour un réseau de classification est représentée par un vecteur dont la dimension est le nombre de classes, et qui représente, pour chaque classe, la probabilité qu'a l'entrée d'appartenir à cette classe.

Les entrées ou sorties de chaque couche du réseau sont constituées de trois types de dimensions :

- les dimensions d'espaces qui représentent une distribution spatiale des données (en image il y en a deux : la dimension des colonnes et celle des lignes de l'image)
- la dimension de profondeur, qui représente le nombre de (cartes de) caractéristiques présentes sur l'image (appelée *feature maps* en anglais), par exemple une image RGB possède 3 cartes de caractéristique, une pour chaque couleur : rouge, vert et bleu.
- la dimension de *batch* (un *batch* est un lot d'échantillons, on explique pourquoi les échantillons sont pris en *batchs* en section A.4.2 page 125),

On note alors la taille d'un *batch* : $N_b \times N_c \times W \times H \times D$, où N_b représente le nombre d'échantillons dans le *batch*, N_c représente le nombre de cartes de caractéristiques et W, $W \times H$ ou $W \times H \times D$ les dimensions d'espace.

En général, plus on avance profondément dans le réseau, plus les dimensions d'espace diminuent pour agréger toutes les informations présentes sur l'entrée et la dimension de profondeur augmente. Ce mécanisme d'agrégation, (appelé *pooling* en anglais) est généralement réalisé par des couches d'agrégation par maximum (`MaxPooling`), par moyenne (`AvgPooling`) ou de convolution à pas (*strided convolution*).

Les réseaux convolutionnels (et réseaux de neurones artificiels en général) sont toujours conçus comme une succession de transformations linéaires et de non-linéarités appelées couches. La fait de mettre une non-linéarité après chaque transformation linéaire s'explique car sinon le réseau pourrait se résumer à une seule grosse transformation linéaire comme composée de telles transformations. Or on ne peut approximer que très peu de fonctions par des fonctions linéaires, alors qu'on a le résultat suivant : l'espace des réseaux de neurones profonds à une seule couche cachée (couche autre que les entrées et sorties) est dense dans l'espace des fonctions continues à support compact [CYBENKO, 1989]. Ce qui se traduit par la possibilité d'approcher toute fonction continue à support compact par un réseau de neurone (même avec une seule couche cachée).

On notera par la suite (sauf indications contraires, ces couches peuvent être utilisée en 1D, 2D ou 3D) :

- `Conv(n,(w,h,d),s,p)` une couche convolutionnelle qui transforme les cartes de caractéristiques en sortie de la couche précédente en `n` nouvelles cartes de caractéristiques, avec

un noyau de taille `w` × `h` × `d`, appliqué avec un pas de `s` (*stride* en anglais) et en ajoutant `p` zéros de chaque coté de la grille (*padding* en anglais). Une convolution à pas `s` ≤ 2 est appelée *strided-convolution*.

— `DeConv(n,(w,h,d),s,p)` (aussi notées `UpConv` ou `ConvTranspose`) une couche de convolution transposée qui transforme les cartes de caractéristiques en sortie de la couche précédente en `n` nouvelles cartes de caractéristiques, avec un noyau de taille `w` × `h` × `d`, un pas de `s` et en ajoutant `p` zéros de chaque coté de la grille.

— `FC(n)` une couche complètement-connectée (ou linéaire) qui applique un opérateur linéaire au vecteur d'entrée et qui produit un vecteur de sortie de taille `n`. C'est équivalent à une couche `Conv(n,(1,1,1),1,0)` sur une grille de taille 1 × 1 × 1.

— `MaxPool((w,h,d),s)` (respectivement `AvgPool((w,h,d),s)`) une couche qui agrège / sous-échantillonne la grille à une échelle plus grossière avec un pas *s*, sur chaque carte de caractéristiques chaque groupe de `w` × `h` × `d` voxel est agrégé en un seul voxel en gardant le maximum (respectivement la moyenne).

— `MaxUnPool((w,h,d),s)` une couche qui *inverse* un `MaxPool((w,h,d),s)` en ajoutant des 0 pour sur-échantillonner la grille, plus précisément chaque voxel est divisé en 8 sous-voxels de mêmes tailles, ils prennent tous la valeur 0 sauf un qui prend la valeur avant sous-division.

— `Sigmoid`, `Tanh`, `ReLU`, `LeakyReLU` et `PReLU` sont des non-linéarités classiques utilisées après les couches linéaires comme `Conv`, `DeConv` et `FC` (cf section A.5.1 page 131).

— `SoftMax` une non-linéarité qui redimensionne les valeurs d'un vecteur dans l'intervalle [0,1] avec somme 1 (cf section A.5.1 page 131).

— `BatchNorm` une couche qui normalise les échantillons sur un *batch* (cf section A.5.2 page 134).

— `DropOut(p)` une couche qui pendant la phase d'entraînement met aléatoirement à 0 certaines valeurs du vecteur d'entrée avec une probabilité `p` ∈ [0,1] (cf section A.5.3 page 135).

Pour une description plus visuelle du champ-réceptif des convolutions et convolutions transposées on renvoie le lecteur vers l'article de blog très pédagogique [1] (en anglais et qui introduit également les convolutions dilatées).

A.3 Petit historique des réseaux de classification d'images

On présente ici une sélection de quelques réseaux convolutionnels qui ont (dans l'ordre chronologique) fait avancer significativement les performances en classification d'image. Un aperçu des architectures de certains de ces réseaux est présenté en figure A.1 page 122.

— le réseau *LeNet* [LeCun et collab., 1998] est le premier réseau de neurones convolutionnel utilisé pour classifier des images qui bat les autres méthodes de l'état de l'art en particulier les réseaux de neurones denses (sans couches convolutionnelles). Cependant ça reste un petit réseau constitué seulement de deux `Conv`, deux `MaxPool` et deux `FC` qui prend en entrée des images 32 × 32.

— le réseau *AlexNet* (classé premier au challenge de classification d'image ILSVRC 2012) : [Krizhevsky et collab., 2012], est le premier réseau convolutionnel profond qui a pu être entraîné suffisamment efficacement pour battre les méthodes utilisant des descripteurs développés par des experts sur la classification d'images réelles. Cette percée est attribuée à 3 verrous qui ont sauté :

  — la puissance de calcul des GPUs, particulièrement adaptés aux convolutions (cf section 3.2.1.1 page 89 pour plus de détails sur les architectures de calculs spécialisées pour l'apprentissage profond),

1. https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

  - la disponibilité des données, le challenge ILSVRC (couramment appelé ImageNet) dispose de plus d'un million d'images annotées parmi 1000 classes,
  - la maturité des algorithmes d'apprentissage de réseaux convolutionnels profond.
- le réseau *Clarifai* (classé premier au challenge de classification d'image ILSVRC 2013) [Matthew D ZEILER et FERGUS, 2014], fait l'observation du nombre élevé d'hyper-paramètres à régler pour optimiser un réseau de neurones et propose quelques méthodes de visualisation permettant d'obtenir des intuitions sur la façon de modifier ces hyper-paramètres. Ils utilisent cette méthode pour modifier légèrement les paramètres du réseau AlexNet et améliorer ses performances.

  De plus, ces méthodes de visualisation permettent de confirmer l'intuition que dans les premières couches le réseau apprend à détecter des caractéristiques locales très simples (des points, des arêtes, des coins...), puis plus la profondeur est importante, plus il combine des caractéristiques locales pour pouvoir détecter des morceaux d'objets (ou des concept avec un niveau d'abstraction plus élevé), pour enfin détecter des objets entiers dans les dernières couches.
- le réseau *VGG* (classé second au challenge de classification d'image ILSVRC 2014) [SIMONYAN et ZISSERMAN, 2014] remplace les blocs de convolutions de 5×5 par deux blocs de convolutions 3×3, ce qui apporte plusieurs améliorations théoriques vérifiées en pratique :
  - réduit de façon non-négligeable la place mémoire et le nombre d'opérations nécessaires au réseau (de $1 - \frac{2 \times 3^2}{5^2} = 28\%$)
  - double le nombre de blocs convolutionnels donc d'activations pour le même champ réceptif,
  - d'une certaine façon, factorise les convolutions 5×5 en convolutions 3×3, ce qui rend l'apprentissage plus facile

  Ceci permet d'entraîner des réseaux plus profonds (jusqu'à 19 blocs convolutionnels) sans difficultés.

VGG est alors un réseau qui allie une grande simplicité de conception et de bonne performances. C'est pourquoi il est encore beaucoup utilisé comme base de réseau pour des tâches plus compliquées. Cependant c'est un réseau très lourd en mémoire avec plus de 100M paramètres à apprendre. Suite à ça les réseaux et méthodes d'apprentissage vont commencer à grandement se complexifier pour pouvoir entraîner plus facilement des réseaux plus profonds :

- le réseau *GoogLeNet* (classé premier au challenge de classification d'image ILSVRC 2014) : [SZEGEDY, W. LIU et collab., 2015], introduit le module Inception, décrit en figure A.2 page 123, ce module peut être vu comme un bloc convolutionnel évolué (cf section A.6.1 page 137) qui permet à la fois de réduire le nombre de poids à apprendre et qui combine des noyaux de tailles différentes.
- le réseau *FCN* pour *Fully Convolutionnal Network* [LONG et collab., 2015], remplace les `Pooling` par des convolutions-à-pas > 1 (*strided convolutions* en anglais) et remplace les blocs `FC` à la fin en blocs convolutionnels de noyaux 1×1 d'où le nom de *complètement convolutionnel*. Ceci permet de supprimer les couches qui demandent une taille fixe en entrée (couches `FC`), et ainsi autorise à inférer la classe d'une image de taille quelconque plus grande que 224×224 (taille utilisée pour l'entraînement sur le jeu de données *ImageNet*).

  De plus ce réseau est utilisé pour faire de la segmentation d'image, après chaque convolution-à-pas les cartes de caractéristique sont récupérées pour inférer une classe par pixel au lieu d'une classe pour l'image complète.
- le réseau *ResNet* [HE, X. ZHANG et collab., 2016], introduit la connexion résiduelle (décrite en figure A.3 page 123) qui permet de propager les gradient plus *vite* jusqu'à la racine pendant la back-propagation (cf section A.6.2 page 139). Il a permis d'entraîner des réseaux bien plus profonds, jusqu'à 1200 couches convolutionnelles sans grande difficulté, bien que les résultats de classifications sur ImageNet cessent de s'améliorer avec plus de 150 couches.

Ce réseau a également été amélioré par [ZAGORUYKO et KOMODAKIS, 2016] qui montrent qu'augmenter le nombre de cartes de caractéristiques en sortie de chaque couche peut être plus efficace que d'augmenter le nombre de couches.

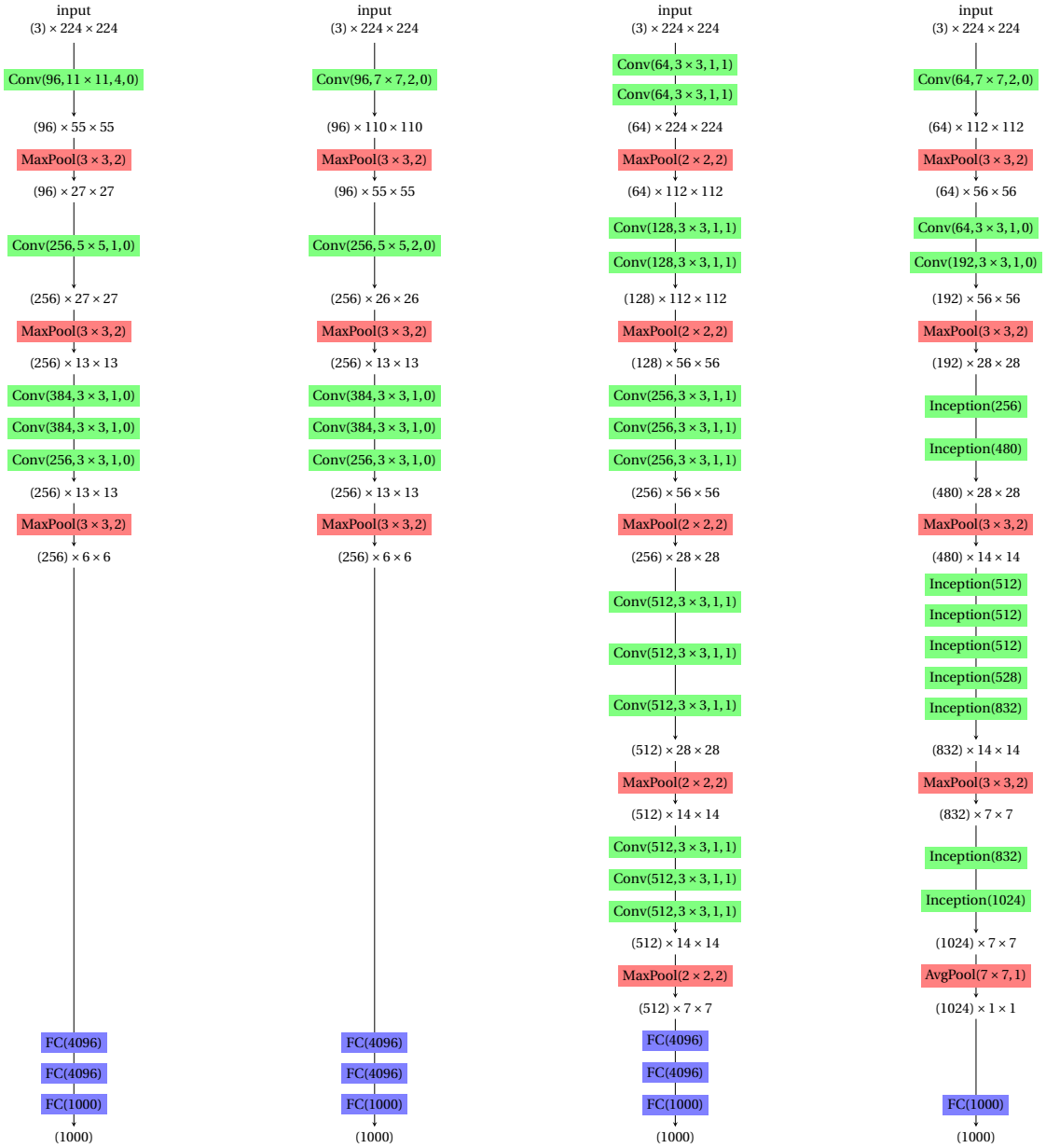


FIGURE A.1 – Architectures des réseaux, de gauche à droite : *Alexnet*, *Clarifai*, *VGG* et *GoogLeNet*. La différences entre *AlexNet* et *Clarifai* se trouvent uniquement dans les deux premières couches de convolutions. Le module Inception peut s'apparenter à une convolution évoluée (cf section A.6.1 page 137), il est décrit en figure A.2 page ci-contre.

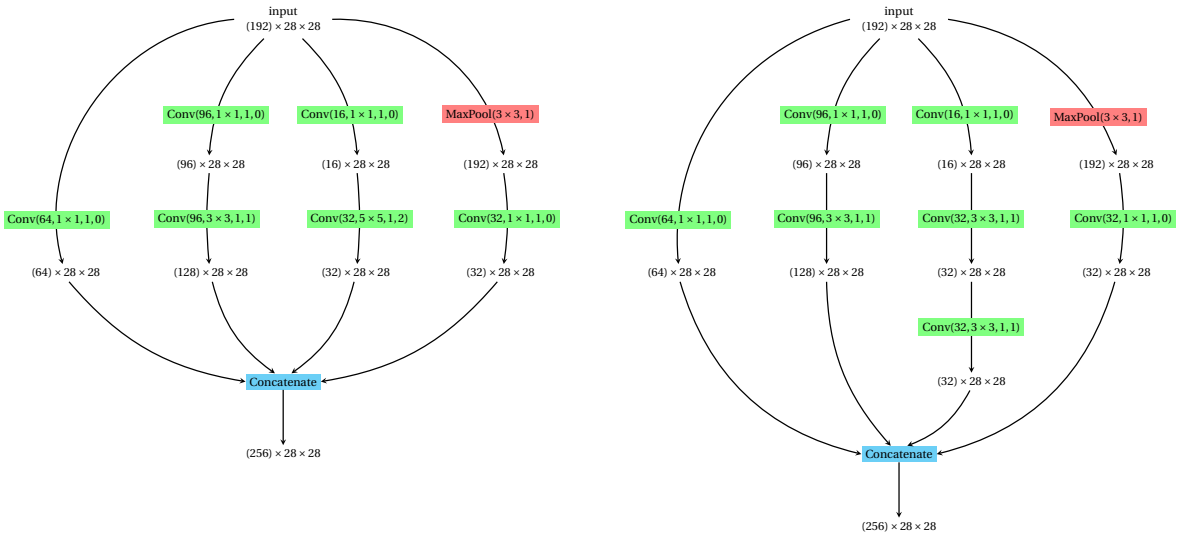


FIGURE A.2 – Module Inception d'origine (à gauche), revu façon VGG en remplaçant la convolution 5 × 5 par deux convolutions 3 × 3 (à droite). (les nombres correspondent au premier module inception rencontré dans le réseau GoogLeNet)

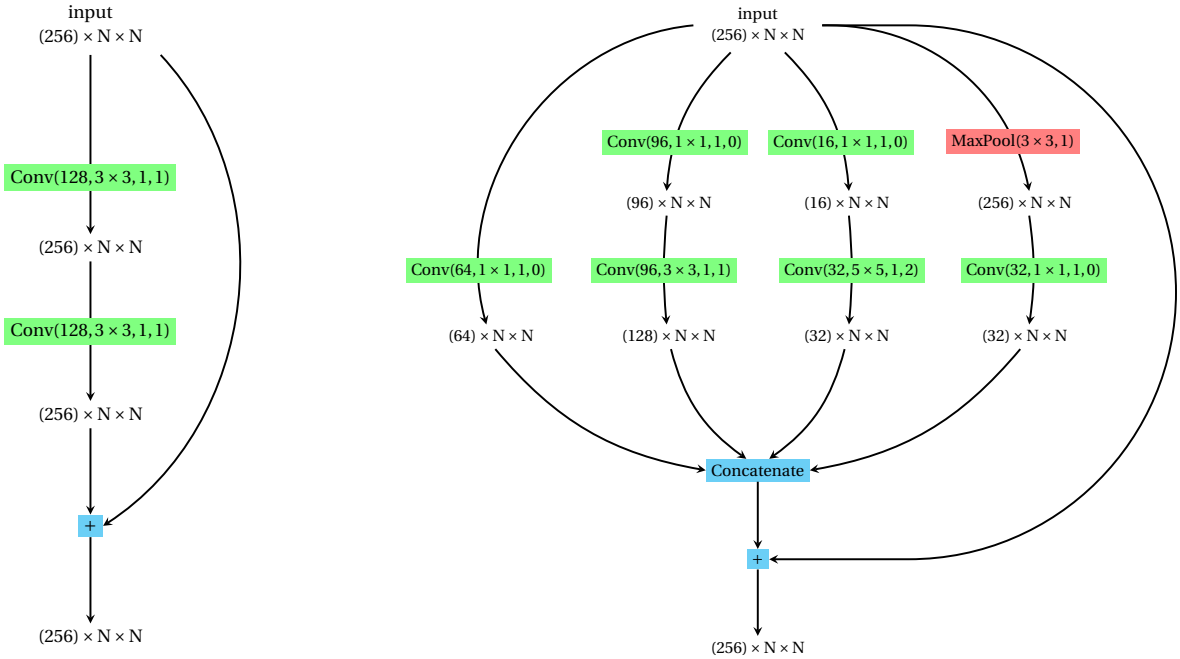


FIGURE A.3 – Connexion résiduelle d'origine (à gauche), et connexion résiduelle sur un module Inception (à droite).

A.4 Méthodes d'apprentissage de réseaux de neurones

L'entraînement d'un réseau de neurones profond consiste à trouver les meilleurs paramètres de ses différentes couches, plus précisément c'est un problème d'optimisation. On cherche à minimiser l'erreur de classification de notre réseau sur un jeu de données de test qui est censé représenter la distribution réelle des données. Pour ce faire, on a uniquement accès à un jeu de données d'entraînement dont la distribution n'est pas forcément exactement la même que le jeu de test. Les réseaux ont donc tendance à être très bons sur les données d'entraînement mais pas particulièrement sur les données de test, on appel ce problème le défaut de généralisation ou le sur-apprentissage (*over-fitting* en anglais). Entre autres des méthodes dites d'augmentation de données sont utilisées pour pallier ce genre de problème, voir le paragraphe A.4.1.

La plupart des algorithmes d'optimisation en apprentissage profond sont basés sur la descente de gradient. Si on note $\theta_t \in \mathbb{R}^d$ les paramètres de notre réseau à l'itération $t \in \mathbb{N}$, il s'agit de mettre à jour itérativement les paramètres du réseau grâce à l'équation :

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta C(\theta_t) \tag{A.1}$$

où η est le taux d'apprentissage et $\nabla_\theta C$ représente le gradient de la fonctionnelle à minimiser C (l'erreur de classification) en θ (les poids des couches linéaires du réseaux). Pour le choix et l'administration du taux d'apprentissage η le lecteur est renvoyé au paragraphe A.4.3 page 127. Les algorithmes d'optimisation de réseau sont des variantes plus ou moins complexes de cette règle, le lecteur est renvoyé au paragraphe A.4.2 page ci-contre pour une description détaillée des différents algorithmes.

Le calcul du gradient $\nabla_\theta C(\theta_t)$ se fait par rétro-propagation des gradients (*backpropagation* en anglais) grâce à le règle de la chaîne :

$$h = g \circ f \Rightarrow \forall x, \nabla h(x) = \nabla g(f(x)) \cdot \nabla f(x) \tag{A.2}$$

Cette formule permet de calculer successivement le gradient par rapport aux paramètres des couches de moins en moins profondes (de plus en plus proches de l'entrée du réseau).

A.4.1 Augmentation de données

Un moyen d'apprendre sur une distribution plus large que sur le jeu d'entraînement à purement parler, est de faire de l'augmentation de données. Cela consiste à transformer légèrement les données en entrée du réseau, tout en gardant le même label à prédire en sortie. On transforme les entrées de telle sorte à prendre en compte les possibles variations qui ne seraient pas présentes dans le jeu d'entraînement, mais qui devrait l'être (d'après des experts du domaine par exemple). Les méthodes d'augmentation classiques en image sont (avec les paramètres de [D. CIREŞAN et collab., 2012]) :

— translater l'image aléatoirement (de maximum 5%)

— ré-échelonner/redimensionner aléatoirement l'image (entre ±15%)

— pivoter/tourner l'image d'un angle aléatoire (entre −5° et 5°)

[SIMARD et collab., 2003] proposent une autre méthode d'augmentation, par déformation élastique de l'image, c'est à dire par déformation non-linéaire. La figure A.4 page suivante montre quelques exemples de transformation élastique d'une image de chiffre manuscrit.

L'augmentation de données peut également être vue comme un moyen de forcer le réseau à apprendre des invariances (qui ne sont pas forcément imposées par l'architecture du réseau). Par exemple pour la classification de chiffres manuscrits, on aimerait, quelle que soit la place du chiffre dans l'image (au centre ou dans un coin), tant qu'il est toujours entier sur l'image, qu'il soit toujours classifié de la même façon, d'où le choix de l'augmentation par translation. Avec des

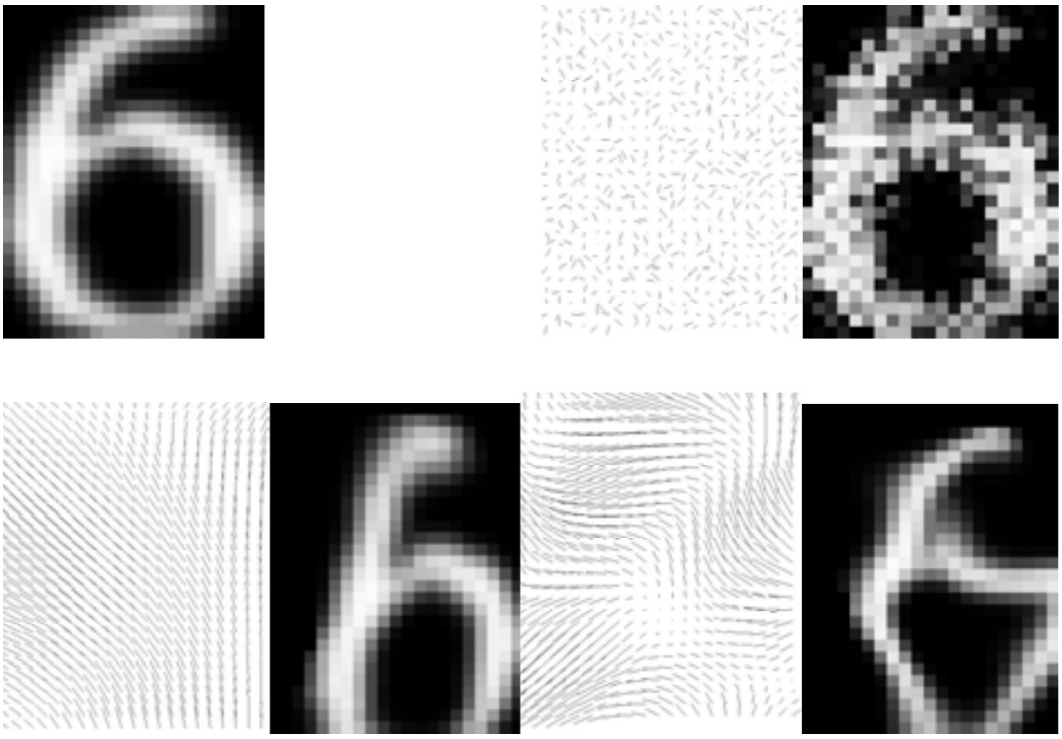


FIGURE A.4 – Exemple d'augmentation de données par déformation élastique sur des images de chiffres manuscrits.

données plus complexes, le choix des bonnes transformations peut être plus difficile et doit être choisi comme étant les transformations pour lesquelles la distribution est invariante ce qui peut demander les compétences d'un expert du domaine.

Cependant il semble que le même style d'augmentation utilisé sur des images stéréo d'objets 3D (jeu de données NORB) ait un effet négatif sur les résultats de généralisation [D. C. CIREŞAN et collab., 2011].

[DEVRIES et TAYLOR, 2017] proposent la méthode d'augmentation *CutOut* qui consiste à mettre un patch uniforme à une position aléatoire sur chaque image pendant l'entraînement (à chaque fois que la même image est vue, le patch est mis à une position différente), ceci pousse le réseau à découvrir plus de caractéristiques complémentaires que les caractéristiques évidentes pour décrire un objet.

A.4.2 Optimisation

Les algorithmes d'entraînement des réseaux de neurones profonds sont basés sur la descente de gradient stochastique (*SGD* pour *Stochastic Gradient Descent* en anglais). En effet, avec le nombre de paramètres à apprendre et le nombre d'échantillons dans les jeux de données, il ne serait pas envisageable de faire une descente de gradient « complète » (où on évalue la fonction de coût C sur l'ensemble du jeu d'entraînement). On fait à la place une minimisation du coût sur un échantillon (ou un lot d'échantillons appelé *batch* en anglais) tiré aléatoirement dans le jeu d'entraînement.

On appel époque (ou *epoch* en anglais) le fait de parcourir une fois l'intégralité du jeu d'entraînement par *SGD* (que ça soit par échantillons ou par *batchs*). Il faut en général plusieurs *epochs* pour réussir à faire converger le réseau vers un minimum (qu'il soit local ou global).

Pour la descente de gradient on doit calculer $\nabla_\theta C(\theta_t)$ avec $C(\theta_t) = \sum_{i=0}^{N} L(z_i, \theta_t)$ où $(z_i)_{i \leq N} = (x_i, y_i)_{i \leq N}$ représente le jeu de données d'entraînement et chaque $z_i = (x_i, y_i)$ est une paire (échantillon,label), et L est une fonction de perte (*loss* en anglais). Un tel calcul peut être impossible pour des jeux de données très grand (ou potentiellement infinis avec l'augmentation de données). Pour la descente de gradient stochastique on calcule donc plutôt $\nabla_\theta C(\theta_t)$ avec $C(\theta_t) =$

$\sum_{i=0}^{N_b} L(z_{p(i+j\cdot N_b)},\theta_t)$ où p est une permutation aléatoire de $[\![1,N]\!]$, $N_b \geq 1$ est la taille d'un *batch* choisie petite devant N et $j \in \left[\!\left[0, \left\lfloor \frac{N}{N_b} \right\rfloor \right]\!\right]$ est le numéro du *batch*. On réalise alors une *epoch* en parcourant tout le jeu d'entraînement, et on tire aléatoirement une nouvelle permutation p de $[\![1,N]\!]$ au début de chaque *epoch*.

La *SGD* a été utilisée pour la première fois en apprentissage profond pour entraîner des modèles en très grande dimension sur beaucoup de données par [Bottou et Le Cun, 2005], mais cette méthode était déjà étudiée depuis [Benveniste et collab., 1990] et bénéficie déjà d'améliorations comme l'inertie (décrite en équation A.3) ou l'inertie de Nesterov [Nesterov, 1983] (décrite en équation A.4) (*momentum* et *Nesterov momentum* en anglais) qui modifient légèrement l'équation A.1 page 124 pour ajouter une inertie des gradients, ceci permet d'éviter un comportement trop stochastique des poids du réseau.

Les deux inerties sont équivalentes pour un taux d'apprentissage η assez petit, et pour un taux d'apprentissage élevé l'inertie de Nesterov permet d'utiliser un coefficient d'inertie μ plus élevé [Sutskever et collab., 2013].

Inertie :

$$\begin{aligned} v_{t+1} &= \mu \cdot v_t - \eta \cdot \nabla_\theta C(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{A.3}$$

Inertie de Nesterov :

$$\begin{aligned} v_{t+1} &= \mu \cdot v_t - \eta \cdot \nabla_\theta C(\theta_t + \mu \cdot v_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{A.4}$$

où η est le taux d'apprentissage, $\mu \in [0,1]$ est le coefficient d'inertie et $\nabla_\theta C(\theta_t)$ est le gradient en θ_t

Le taux d'apprentissage η est un hyper-paramètre qui influence considérablement la dynamique d'apprentissage comme on le verra en section A.4.3 page suivante, de nombreuses méthodes dites à « taux d'apprentissage adaptatif » ont donc été développées pour réduire (voire) supprimer l'influence du taux d'apprentissage. [Ruder, 2016] fait une revue des plus utilisées de ces méthodes.

AdaGrad [Duchi et collab., 2011] adapte le taux d'apprentissage aux poids du réseau, il est plus élevé pour les poids qui sont mis à jour peu fréquemment. Pour cette raison, *AdaGrad* fonctionne bien avec des données éparses. La règle de mise à jour est la suivante :

$$\begin{aligned} n_{t+1} &= n_t + \nabla_\theta C(\theta_t)^2 \\ v_{t+1} &= -\frac{\eta}{\sqrt{n_{t+1}+\epsilon}} \cdot \nabla_\theta C(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{A.5}$$

Une faiblesse de *AdaGrad* est que le terme n_t ne cesse de croître pour finalement empêcher toute mise à jour des poids du réseau.

AdaDelta [Matthew D. Zeiler, 2012] est une variante qui vise à réduire l'agressivité de la décroissance du taux d'apprentissage de *AdaGrad*, ceci est réalisé en ne gardant qu'une moyenne glissante des carrés des gradients par la formule suivante :

$$\begin{aligned} n_{t+1} &= \beta \cdot n_t + (1-\beta) \cdot \nabla_\theta C(\theta_t)^2 \\ m_{t+1} &= \beta \cdot m_t + (1-\beta) \cdot v_t^2 \\ v_{t+1} &= -\frac{\sqrt{m_{t+1}+\epsilon}}{\sqrt{n_{t+1}+\epsilon}} \cdot \nabla_\theta C(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{A.6}$$

RMSProp [TIELEMAN et G. HINTON, 2012] est une méthode à taux d'apprentissage adaptatif, qui en plus d'une moyenne glissante des gradients (inertie) calcul une moyenne glissante des gradients au carré.

$$\begin{aligned} g_{t+1} &= \beta \cdot g_t - (1-\beta) \cdot \nabla_\theta C(\theta_t) \\ n_{t+1} &= \beta \cdot n_t - (1-\beta) \cdot \nabla_\theta C(\theta_t)^2 \\ v_{t+1} &= \mu \cdot v_t - \frac{\eta}{\sqrt{n_t - g_t^2 + \epsilon}} \cdot \nabla_\theta C(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{A.7}$$

ADAM [KINGMA et J. BA, 2014] est également une méthode à taux d'apprentissage adaptatif, qui conserve des moyennes glissantes des gradients et des gradients au carré :

$$\begin{aligned} g_{t+1} &= \beta_1 \cdot g_t - (1-\beta_1) \cdot \nabla_\theta C(\theta_t) \\ n_{t+1} &= \beta_2 \cdot n_t - (1-\beta_2) \cdot \nabla_\theta C(\theta_t)^2 \\ \hat{g}_{t+1} &= \frac{g_t}{(1-\beta_1^t)} \\ \hat{n}_{t+1} &= \frac{n_t}{(1-\beta_2^t)} \\ v_{t+1} &= -\frac{\eta}{\sqrt{\hat{n}_{t+1} + \epsilon}} \cdot \hat{g}_{t+1} \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{A.8}$$

Il existe encore de nombreuses autres variantes qui se veulent plus ou moins performantes en fonction du type de fonctionnelle à minimiser. Cependant les deux méthodes qui semblent les plus utilisées dans les articles de recherche semblent encore être la *SGD* avec *momentum* et *ADAM*.

Plutôt que de concevoir une méthode d'optimisation la mieux adaptée à un problème, [BELLO et collab., 2017] proposent d'entraîner un « contrôleur » à générer des lois de mise à jour des poids qui minimise l'erreur des réseaux entraînés. Le « contrôleur » est un réseau de neurones récurrent (*RNN*) entraîné par *Reinforcement Learning* (plus de détails dans l'article). C'est une approche similaire à celles utilisées en section A.6.4 page 143 pour trouver des architectures de réseaux optimales.

Cette méthode a permis de découvrir les lois de mise à jour *AddSign* et *PowerSign* décrites respectivement par des formules des formes suivantes :

$$\begin{aligned} \theta_{t+1} &= \theta_t + \left(\alpha + f(t) \cdot \text{sign}(\nabla_\theta C(\theta_t)) \cdot \text{sign}(m)\right) \cdot \nabla_\theta C(\theta_t) \\ \theta_{t+1} &= \theta_t + \alpha^{\left(f(t) \cdot \text{sign}(\nabla_\theta C(\theta_t)) \cdot \text{sign}(m)\right)} \cdot \nabla_\theta C(\theta_t) \end{aligned} \tag{A.9}$$

où m représente une moyenne glissante des gradients et f est soit 1 soit une fonction de décroissance vers 0. On voit en particulier apparaître le terme $\text{sign}(\nabla_\theta C(\theta_t)) \cdot \text{sign}(m)$ qui change de sens si le gradient et la moyenne glissante concordent.

A.4.3 Taux d'apprentissage

Le choix du taux d'apprentissage η initial et de sa méthode de décroissance au cours des *epochs* est un des hyper-paramètres les plus importants pour arriver à faire converger un réseau. Un taux d'apprentissage trop faible rend la convergence très (trop) lente, alors qu'un taux trop élevé fait diverger les paramètres du réseau.

Il existe plusieurs méthodes classiques pour faire décroître le taux d'apprentissage η_t au cours des *epochs* t :

— décroissance en $\eta_t = \eta_0/(1+\gamma t)$,
— décroissance exponentielle : $\eta_t = \eta_0 e^{-\gamma t}$,
— décroissance exponentielle à pas ou à pas multiples, revient à une décroissance exponentielle où on ne réduit le taux d'apprentissage qu'à certains pas de temps avec un γ plus élevé,
— décroissance cosinus : $\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + cos\left(\pi \frac{t}{t_{max}}\right)\right)$,
— décroissance sur les plateaux, comme son nom l'indique il s'agit de décroître le taux d'apprentissage quand l'erreur de validation stagne depuis un certains temps.

La figure A.5 montre graphiquement l'évolution du taux d'apprentissage par ces différentes méthodes de décroissance.

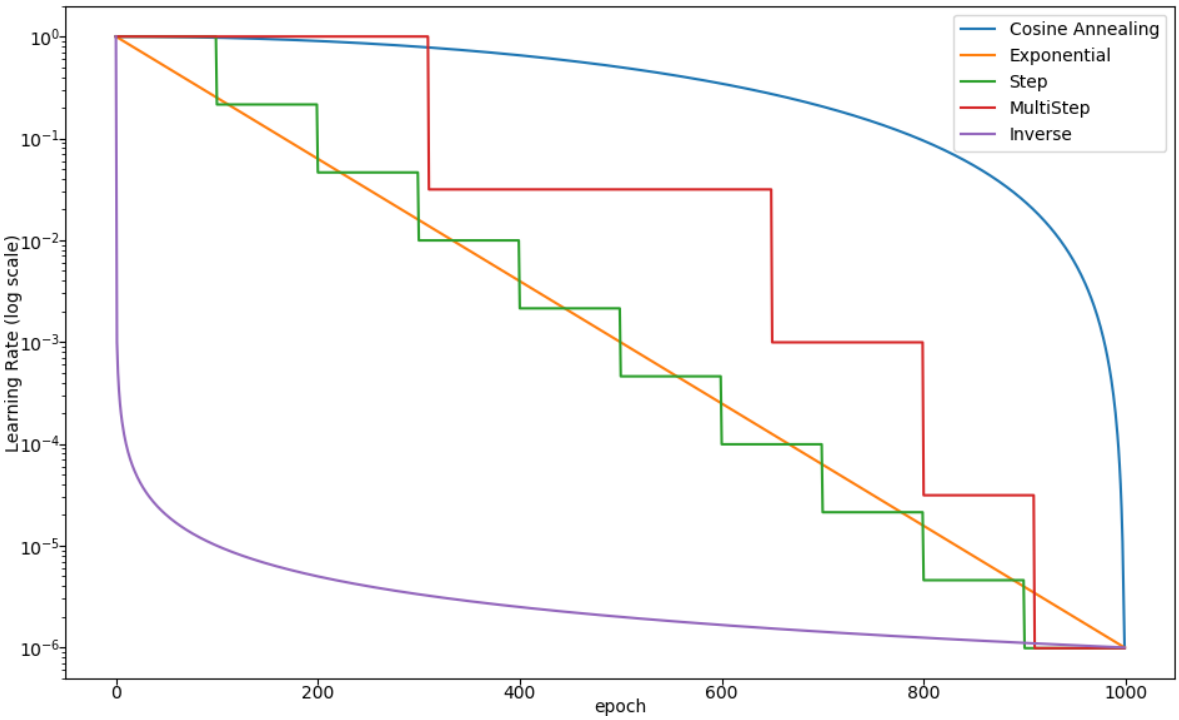


FIGURE A.5 – Évolution du taux d'apprentissage au cours des itérations pour différentes méthodes classiques (décroissantes).

Devant la difficulté rencontrée avec le choix du taux d'apprentissage, plusieurs méthodes moins classiques ont été développées. Contre-intuitivement ces méthodes ne sont pas décroissantes. Elle se basent sur le fait qu'il existe un intervalle de taux d'apprentissage qui permet de faire converger un réseau, le bas de l'intervalle permettant de faire converger les paramètres finement vers un minimum local, alors que le haut de l'intervalle permet de s'extraire d'un minimum local pour se diriger éventuellement vers un meilleur minimum local.

La méthode de *Cycling Learning Rate* [SMITH, 2017] propose donc de faire croître puis décroître le taux d'apprentissage à des intervalles de pas constants tout en réduisant la taille de l'intervalle de valeurs que peut prendre le taux d'apprentissage (cf figure A.6 page ci-contre).

Pour la même raison que le *Cycling Learning Rate*, la méthode *SGDR* (pour *SGD with Restarts*) [LOSHCHILOV et HUTTER, 2016] ré-initialise le taux d'apprentissage à un certain intervalle de temps pour sortir d'un minimum local. Cependant l'intervalle de temps entre chaque ré-initialisation est multiplié à chaque fois par un facteur ≤ 1 de telle sorte que le réseau final soit le résultat de l'intervalle le plus long (cf figure A.7 page suivante)

De plus la figure A.8 page 130 décrit bien intuitivement ce qu'il se passe avec les méthodes cycliques ou à ré-initialisation du taux d'apprentissage, en particulier l'idée convergence vers un minimum local, puis à chaque ré-initialisation, un saut pour ressortir de ce minimum local.

Il existe également des méthodes à taux d'apprentissage différent dans chaque couche du réseau [J. HOWARD et RUDER, 2018]. Ceci peut être utile pour le *Fine-Tunning* de réseau en *Transfer*

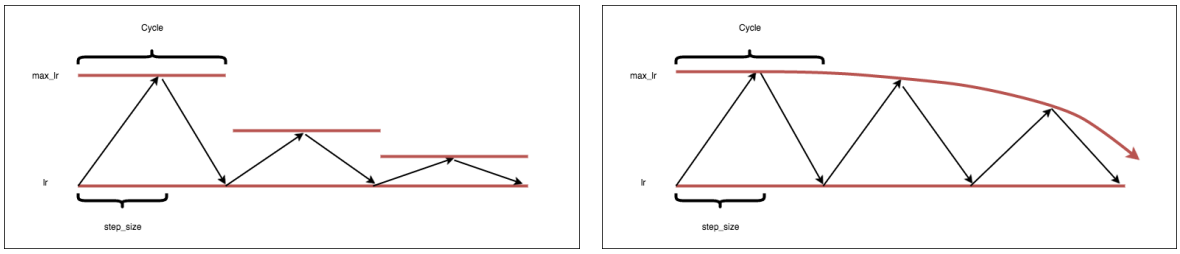


FIGURE A.6 – Évolution du taux d'apprentissage cyclique. À gauche la taille de l'intervalle est divisée par deux à chaque cycle, à droite la taille de l'intervalle décroît exponentiellement (images issues du github)

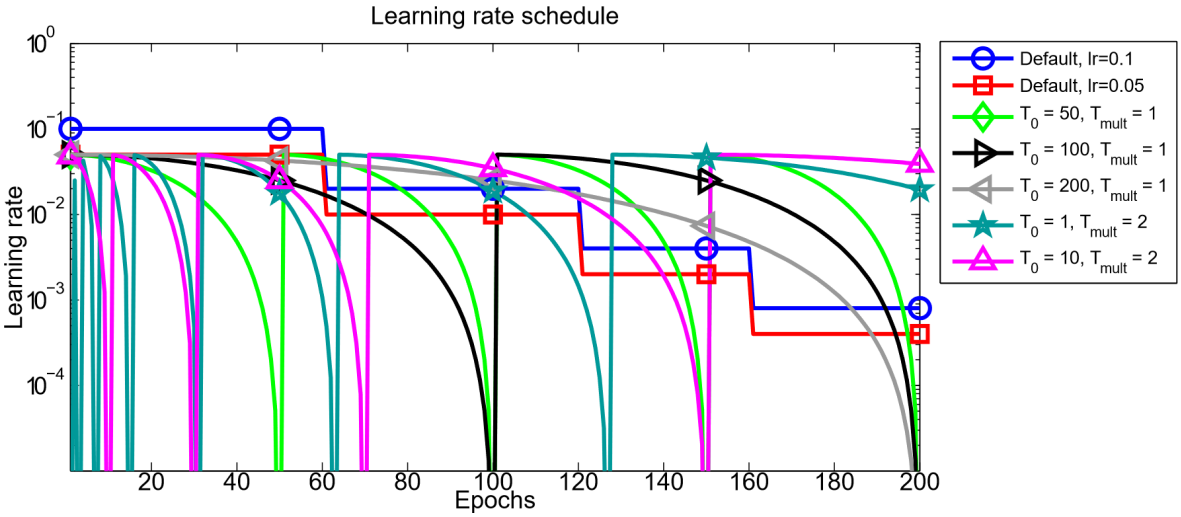


FIGURE A.7 – Évolution du taux d'apprentissage au cours des itérations pour différents méthodes. *Default* est pour la méthode utilisé par [ZAGORUYKO et KOMODAKIS, 2016] par plateaux. T_0 est le nombre d'*epochs* avant la première ré-initialisation et T_{mult} est le facteur multiplicateur. (image issue de [LOSHCHILOV et HUTTER, 2016])

Learning, en effet les premières couches extraient des caractéristiques géométriques fines (des arètes, des coins) qui n'ont pas spécialement besoin d'être changées, alors que les couches plus profondes extraient des caractéristiques plus abstraites et qui doivent donc mieux s'adapter à la nouvelle tâche à apprendre.

A.4.4 Ensembles de réseaux

Les méthodes d'ensembles de réseaux sont des méthodes qui d'une façon ou d'une autre tirent parti de l'information fournie par plusieurs réseaux pour obtenir un résultat final.

Il existe d'abord des méthodes de création d'ensembles explicites, en inférant la classe d'un même échantillon de plusieurs façons, puis en votant pour la classe la plus représentée, ou en moyennant les probabilités sorties pour chaque classe :

— simplement en entraînant plusieurs réseaux avec des poids différents initialisés aléatoirement. Si on a suffisamment de modèles entraînes, on peut même sélectionner les plus utiles pour maximiser telle ou telle métrique [CARUANA et collab., 2004],

— en inférant la classe avec un seul réseau, mais en lui donnant des variations de l'échantillon (par exemple tourné à 0°, 45°, 90° et 135°),

Créer un ensemble de réseaux peut être très coûteux si un seul réseau est déjà long à entraîner, il existe donc des méthodes qui cherchent à créer des ensembles « gratuitement ». La méthode d'ensemble d'instantannées [G. HUANG, Yixuan LI et collab., 2017] tire parti de méthode comme *SGDR* ou *Cycling Learning Rate* qui converge vers plusieurs minima locaux pendant l'entraîne-

ment, il suffit donc de garder les m derniers réseaux convergés (juste avant ré-initialisation du taux d'apprentissage) et d'en faire un ensemble. Pour être sûr qu'à chaque cycle ou ré-initialisation le réseau se déplace vers un autre minimum local, l'intervalle de valeurs du taux d'apprentissage n'est pas diminué (cf figure A.8).

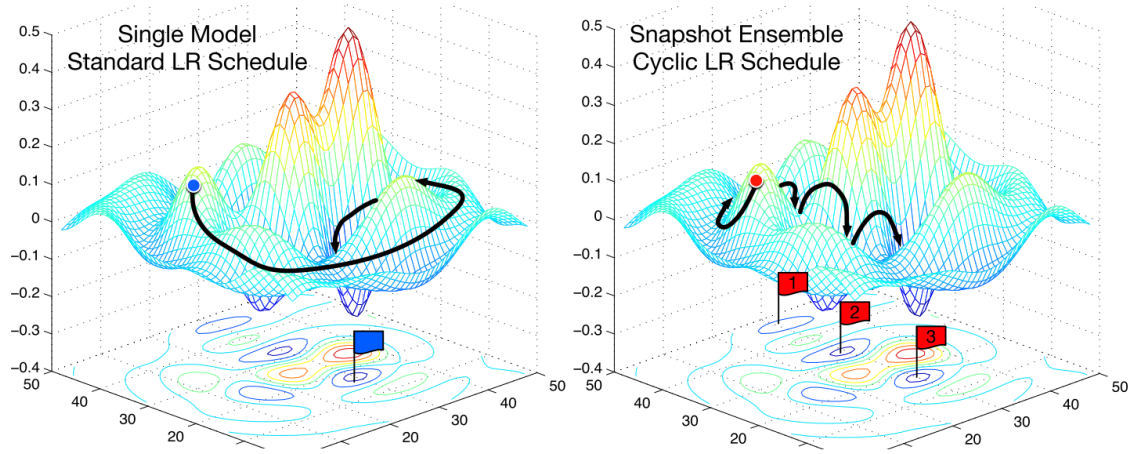


FIGURE A.8 – Description du mécanisme d'optimisation de la méthode de création d'ensemble d'instantannés (*Snapshot Ensembles* en anglais). Le taux d'apprentissage est réinitialisé (remis à sa valeur initiale élévée) pour pousser le réseau à sortir d'un minimum local pour en explorer d'autre, le taux d'apprentisage est alors rapidement diminué pour faire converger le réseau vers un autre minimum local (on garde alors un instantanné) avant de réinitialiser à nouveau le taux d'apprentissage. (image tirée de [G. HUANG, Yixuan LI et collab., 2017])

Il existe également des méthodes de créations d'ensembles implicites, qui entraînent plusieurs réseaux comme des sous-parties d'un seul gros réseau, et qui utilisent le gros réseau entier pour l'inférence en test. Ces méthodes sont appelées DropOut, DropConnect, Stochastic Depth, SwapOut et on rentrera plus en détail sur ces méthodes en section A.5.3 page 135.

A.4.5 Initialisation

Les algorithmes d'optimisation font converger les poids du réseau vers un optimal local par descente de gradient. Mais pour que cette descente de gradients fonctionne (et ne soit pas trop lente), il faut se trouver dans une région pas trop loin d'un minimum local et où les gradients peuvent être calculés (sans que la rétro-propagation gradients ne subisse un phénomène d'extinction qu'on abordera plus tard).

[G. E. HINTON et SALAKHUTDINOV, 2006] proposent une méthode d'initialisation par pré entraînement non-supervisé de chaque couche par algorithme glouton. Successivement chaque couche est entraînée pour représenter les principaux éléments de variabilité existants dans les données (plus de détails dans l'article). [BENGIO et collab., 2007] confirment que cette méthode initialise les poids du réseau près d'un bon minimum local.

Cependant cette méthode d'initialisation ajoute du temps de calcul, pour la plupart, les méthodes suivantes ne font qu'échantillonner les poids des différentes couches selon des distributions de probabilité avec des paramètres bien choisis, ce qui prend un temps négligeable par rapport au pré-entraînement.

En étudiant la propagation de l'information dans un réseau de neurones, que ce soit pendant l'inférence en phase de test ou pendant la rétro-propagation des gradients en phase d'entraînement, on observe qu'il peut y avoir des phénomènes d'atténuation ou même d'extinction de l'information [GLOROT et BENGIO, 2010]. Pour éviter ce type de problème, il faut que les réseaux soient conçus pour la variance en sortie d'une couche soit la même qu'en entrée, sinon le flux d'information s'atténuera soit en inférence soit en rétro-propagation. [GLOROT et BENGIO, 2010] propose

donc une méthode pour initialiser les poids d'un réseau, il s'agit de tirer aléatoirement les poids d'une couche selon une loi normale centrée de variance $Var(\omega^i) = \frac{2}{n_i+n_{i+1}}$, où ω_i représente les poids de la couche i et n_i et n_{i+1} sont le nombre de variables respectivement en entrée et sortie de la couche i. Cette méthode est appelée initialisation « Xavier » ou « Glorot » du nom de l'auteur de l'article.

[He, X. Zhang et collab., 2015] reprend l'approche développée pour l'initialisation « Xavier », et l'adapte au cas des non-linéarités *ReLU*. Le fait que cette non-linéarité ne soit pas de moyenne nulle autour de 0 aboutit à une règle d'intialisation légèrement différente : il faut tirer aléatoirement les poids d'une couche selon une loi normale centrée de variance $Var(\omega^i) = \frac{2}{n_i}$. Cette méthode est appelée initialisation « Kaiming » ou « MSRA »

[Saxe et collab., 2013] étudie la dynamique d'apprentissage des réseaux de neurones profonds linéaires qui présentent des phénomènes d'apprentissage similaires aux réseaux non-linéaires, comme entre autre une convergence accélérée quand les poids du réseau sont initialisés par l'algorithme glouton comme [G. E. Hinton et Salakhutdinov, 2006] ; [Bengio et collab., 2007]. Grâce à l'étude menée, [Saxe et collab., 2013] propose une méthode d'initialisation des poids par matrices orthogonales et prouve que l'initialisation par algorithme glouton et la méthode proposée permettent un entraînement avec une vitesse d'apprentissage qui ne dépend pas de la profondeur du réseau, propriété non vérifiée par l'initialisation avec distribution normale.

Cependant les méthodes décrites plus haut ne sont que des études théoriques sur la façon dont l'information se propage dans des réseaux assez simplistes par rapport aux réseaux actuellement utilisés dans la littérature (des non-linéarités exotiques, des couches de normalisation, des connexions dans tous les sens...).

[Mishkin et Matas, 2015] propose une méthode appelée LSUV pour *Layer-Sequential Unit Variance*, qui fait varier les poids du réseau progressivement pour que la sortie de chaque couche soit effectivement de variance unitaire sur les données d'entraînement. L'algorithme est assez simple, après avoir initialisé les poids par matrices orthogonales comme dans [Saxe et collab., 2013], dans l'ordre des couches L les moins profondes vers les plus profondes, on calcule la sortie y_L d'un *batch* de données du jeu d'entraînement, si la variance de cette sortie est trop éloignée de 1, les poids ω_L de la couche L sont redimensionnés par la variance $Var(y_L)$: $\omega_L \leftarrow \omega_L / Var(y_L)$. Et on itère tant que la variance obtenue en sortie de la couche n'est pas assez proche de 1.

A.5 Bonnes pratiques dans le choix des différentes couches

Il existe de multiples façons de choisir les couches qui composent un réseau de neurones profond, cependant on peut les classer dans différentes catégories : les non-linéarités abordées en section A.5.1, les couches de normalisation abordées en section A.5.2 page 134, les couches de *DropOut* abordées en section A.5.1 et les convolutions qui seront abordées plus loin en section A.6.1 page 137.

A.5.1 Non-Linéarités

Intuitivement les non-linéarités donnent l'activation d'un neurone, proche de 1 le neurone est activé, proche de 0 (ou de -1) le neurone est éteint, c'est pourquoi les premiers réseaux de neurones ont été conçus avec les non-linéarités suivantes (représentation graphique en figure A.9 page suivante) :

— *Sigmoid* : $\sigma(x) = \frac{1}{1+e^{-x}}$

— *Tanh* : $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

— *Hard-Sigmoid* : $\sigma(x) = \max(0, \min(1, 0.25x + 0.5))$ proposée par [Gulcehre et collab., 2016] comme *Hard-Tanh*, qui ont l'intérêt d'assurer un comportement linéaire proche de 0, et de produire une décision nette dans les zones de saturation,

— *Hard-Tanh* : $\sigma(x) = \max(-1, \min(1, x))$

— *SoftSign* [GLOROT et BENGIO, 2010] : $\sigma(x) = \frac{x}{1+|x|}$, permet de minimiser la saturation des neurones car cette fonction converge plus lentement vers ses asymptotes.

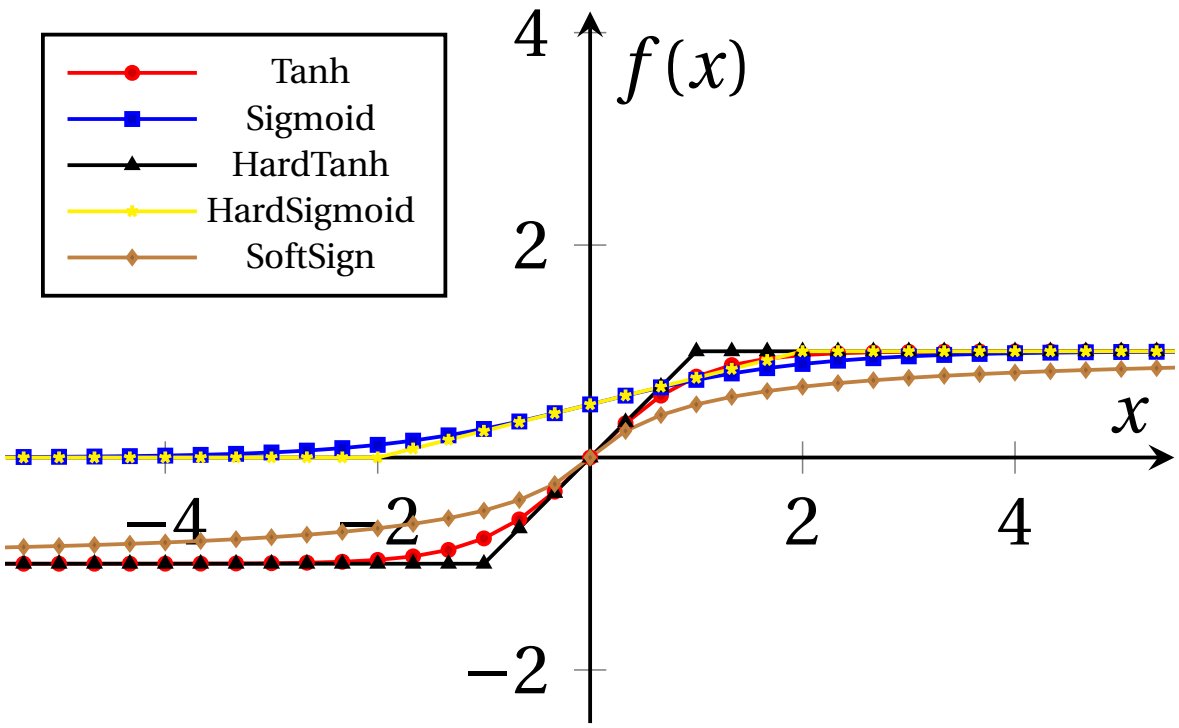


FIGURE A.9 – Graphiques des différentes non-linéarités utilisées dans les réseaux de neurones anciens.

Cependant ces non-linéarités ont un gros inconvénient, elles ont une dérivée (presque) nulle sur une bonne partie de leur espace de définition, ainsi pendant la phase de rétro-propagation des gradients par règle de la chaine, aucune information n'est transmise dans les neurones « saturés » (là où la dérivée est nulle). L'entraînement est donc ralenti, parfois tellement que le réseau ne converge jamais.

Une méthode a été proposée pour faire converger les réseaux équipés de non-linéarités de ce type. Il s'agit d'ajouter du bruit pendant l'entraînement sur les parties saturées $]-\infty,-1]$ et $[1,+\infty[$ pour éviter que les gradients soient nuls [GULCEHRE et collab., 2016].

Cependant cette méthode ne permet pas d'obtenir des résultats aussi bons qu'avec les réseaux de neurones modernes qui utilisent tous des non-linéarités non saturées (*ReLU* [NAIR et G. E. HINTON, 2010] ; [GLOROT, BORDES et collab., 2011] et ses variantes). En opposition aux linéarités saturées comme *Sigmoid* ou *Tanh* utilisées ultérieurement, ces non-linéarités ont l'avantage de ne propager intégralement l'information sur une partie de leur espace de définition : $\mathbb{R}^+$ Cependant la non-linéarité *ReLU* a encore deux inconvénients que ses variantes tentent de corriger :

— une dérivée nulle sur $\mathbb{R}^-$,

— une moyenne non-nulle autour de 0.

les variantes corrigent le premier inconvénient en remplaçant la partie sur $\mathbb{R}^-$ par une fonction décroissante à dérivée non-nulle (mais toujours de dérivée de norme < 1), et le second inconvénient est corrigé par des fonctions qui gardent un comportement semblable à l'identité sur les valeurs négatives proches de 0.

Les non-linéarités *ReLU* et ses variantes sont définies comme suit (représentation graphique en figure A.10 page ci-contre) :

— *ReLU* [NAIR et G. E. HINTON, 2010] ; [GLOROT, BORDES et collab., 2011] : $\sigma(x) = \max(0, x)$

— *LeakyReLU* [MAAS et collab., 2013] : $\sigma(x) = \max(sx, x)$ où $0 < s < 1$ est la pente sur $\mathbb{R}^-$ qui est fixée à 0.01 dans l'article d'origine,

— *P(arametric)ReLU* [He, X. Zhang et collab., 2015], est une non-linéarité *LeakyReLU* avec une pente s apprenable,

— [Xu et collab., 2015] propose une étude des différents ReLU et propose *RReLU* qui est une non-linéarité *LeakyReLU* avec une pente s qui est tirée aléatoirement pendant l'entraînement dans un intervalle fixé à [0.125,0.333] dans l'article. Et pendant l'inférence en test, la pente est fixée à $s = 0.229$ (la moyenne des bornes de l'intervalle),

— *E(xponential)LU* [Clevert et collab., 2015] : $\sigma(x) = \max(0,x) + \min(0,\alpha(e^x - 1))$ est localement de moyenne nulle autour de 0,

— *PELU* [Trottier et collab., 2017], est une non-linéarité *ELU* avec un paramètre α apprenable,

— *CELU* [Barron, 2017] : $\sigma(x) = \max(0,x) + \min(0,\alpha(e^{x/\alpha} - 1))$, cette variante de *ELU* est dérivable partout (même en 0) pour toutes les valeurs de α,

— *SELU* [Klambauer et collab., 2017] : $\sigma(x) = s(\max(0,x) + \min(0,\alpha(e^x - 1)))$ avec $s = 1.0507009873554804934193349852946$ et $\alpha = 1.6732632423543772848170429916717$, c'est valeurs ont été choisies pour vérifier des propriétés de normalisation de la sortie (plus de détails en section A.5.2 page suivante).

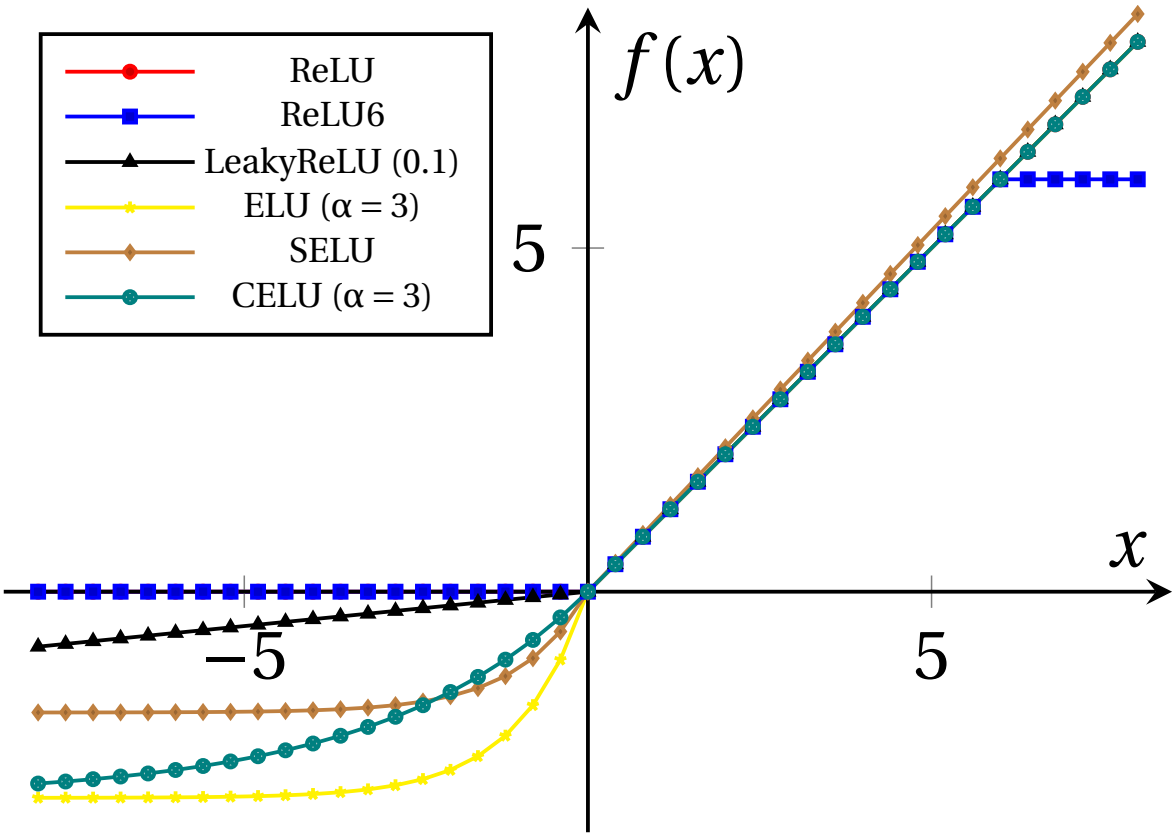


FIGURE A.10 – Graphiques des différentes non-linéarités utilisées dans les réseaux de neurones modernes. On observe en particulier que ReLU, LeakyReLU, ELU (pour $\alpha \neq 1$) et SELU ne sont pas dérivables en 0 contrairement à CELU qui a été conçu pour être dérivable partout.

Enfin, la non-linéarité *SoftMax* est utilisée en sortie d'un réseau de classfication pour transformer un vecteur (de taille le nombre de classes) en un vecteur de même taille qui représente par classe la probabilité d'appartenir à la classe, elle est définie pour la classe i par $\sigma(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$. Elle retourne donc bien un vecteur de valeurs positives et de somme 1, et la classe avec la plus haute probabilité est celle dont la valeur était la plus importante.

A.5.2 Normalisation

Les observations faites en section A.4.5 page 130 sur la propagation de l'information dans un réseau, et en particulier sur la conservation de la variance à travers les différentes couches, peuvent être traîtées en ajoutant une couche spécifique à l'entrée ou à la sortie de chaque couche linéaire (`Conv` ou `FC`). Ces couches sont dites de « Normalisation ».

La première couche de ce type est la normalisation par *batch* [IOFFE et SZEGEDY, 2015] (couramment appelée *BatchNorm* ou BN). Pendant l'entraînement, elle calcule des moyenne et variance glissante de tous les échantillons qu'on lui donne, ces statistiques sont calculées par *batch* et seront utilisées pour normaliser les *batchs* pendant l'inférence en phase de test. Pendant l'entraînement le *batch* est normalisé pour que sa moyenne et sa variance correspondent aux moyennes et variances glissantes.

Ceci a l'intérêt de réduire ce qui est appelé *Internal Covariate Shift* qui correspond à la variation de la distribution des activations en sortie de chaque couche au cours de l'entraînement. En effet quand les poids d'une couche sont modifiés, cela modifie la distribution des sorties de la couche ainsi les poids de la couche suivante ne sont plus adaptés. Alors qu'en conservant la variance et la moyenne à la sortie de chaque couche (par *BatchNorm*) les poids des différentes couches interfèrent beaucoup moins entre eux.

Cependant cette méthode fonctionnne mal quand les *batchs* sont petits (pour des très gros réseaux) ou que les échantillons d'un même *batch* ne sont pas indépendants (pour images de séquences vidéos par exemple). Suite à l'introduction de cette couche, quelques variantes ont donc été créées pour remédier aux différents inconvénients qu'elle présente dans certains contextes (les différences sont représentées en figure A.11) :

— [IOFFE, 2017] (*BatchReNormalization*) permet de gérer les petits *batchs* sur lesquels il est difficile d'obtenir des statistiques fiables, ceci est réalisé en corrigeant les moyennes et variances du *batch*, qui permet d'entraîner en normalisant les *batchs* avec les moyennes et variances glissantes,

— la normalisation par couche [J. L. BA et collab., 2016] (*LayerNorm*), supprime la dépendance à la taille du *batch* en calculant les statistiques sur la profondeur (les cartes de caractéristique) plutôt que sur le *batch*,

— la normalisation par instance [ULYANOV et collab., 2016] (*InstanceNorm*), supprime également la dépendance à la taille du *batch*, cette fois en calculant les statistiques sur les dimensions d'espace,

— la normalisation par groupe de cartes de caractéristiques [Y. WU et HE, 2018] (*GroupNorm*) est un compromis entre *InstanceNorm* et *LayerNorm*, les cartes de caractéristiques sont regrouppées pour calculer les statistiques (cf figure A.11).

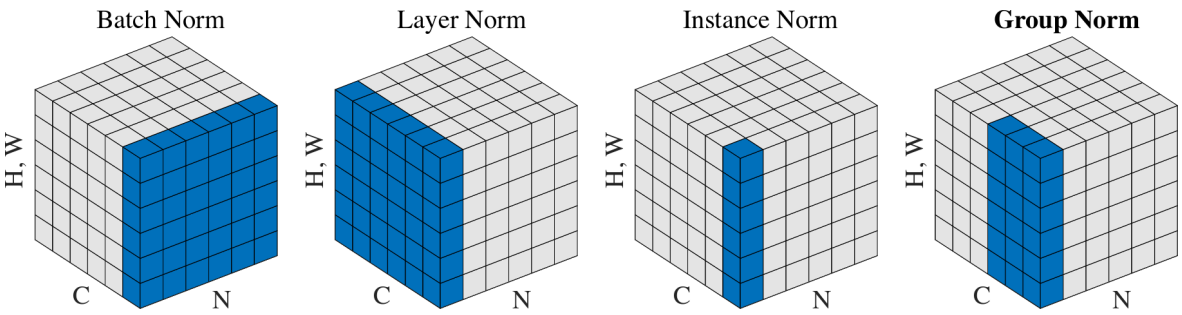


FIGURE A.11 – Description des différents types de normalisation, la partie bleu représente l'espace sur lequel l'échantillon est normalisé. La dimension N représente le *batch*, la dimension C représente les cartes de caractéristques et H,W représentent les dimensions d'espace (image tirée de l'article [Y. WU et HE, 2018]).

Il existe également des méthodes qui visent à normaliser de façon implicite (sans ajouter de couche dédiée) :

— [KLAMBAUER et collab., 2017] propose des réseaux SN-NN (pour *Self-Normalizing Neural Network*) qui ne nécessitent pas de couche de normalisation car l'initialisation, les couches de non-linéarité *SELU* (cf section A.5.1 page 131) et de *dropout* (*AlphaDropOut* cf section A.5.1 page 131) sont conçues de telle sorte que la sortie de chaque couche soit de moyenne nulle et de variance 1. De plus ces couches ont une propriété contractante, même un échantillon qui n'est pas de moyenne nulle et de variance 1, va progressivement à travers le réseau tendre vers un signal de moyenne nulle et de variance 1,

— La normalisation des poids (*WeightNorm*) [SALIMANS et KINGMA, 2016], normalise les poids des filtres de convolutions pendant l'entraînement en les représentant sous une autre forme, $\omega = n_\omega u_\omega = |\omega| \frac{\omega}{|\omega|}$ les gradients sont alors calculés sur les deux variables n_ω et u_ω de la décomposition comme si c'étaient deux poids indépendants du réseau ainsi que le pas de descente de gradient, avant que ω ne soit recalculé pour traiter un nouveau *batch*. Ceci a l'avantage d'améliorer le conditionnement de la Hessienne de problème d'optimisation et donc d'accélérer la convergence de la descente de gradient.

A.5.3 DropOut

[N. SRIVASTAVA et collab., 2014] observe un phénomène de « co-adaptation » des neurones, c'est-à-dire que le réseau arrive à combiner de nombreux neurones pour classifier une donnée. Mais ceci n'est pas forcément désirable car de cette façon le réseau peut apprendre des corrélations très fines présentes uniquement dans le jeu d'entraînement et donc mener à du sur-apprentissage et donc avoir des performances dégradées sur un jeu de test.

Pour réduire ce défaut de généralisation [N. SRIVASTAVA et collab., 2014] propose d'ajouter une couche de *DropOut* qui pendant l'entraînement éteint aléatoirement une certaine proportion $p \in [0,1]$ des neurones en entrée (remplace l'entrée par 0) c'est à dire qu'à chaque *batch* les mêmes combinaisons de neurones ne peuvent pas forcément être utilisées pour classifier telle ou telle classe, mais le réseau est obligé d'apprendre à combiner de l'information venant de différentes sources pour prendre une décision. La couche de *DropOut* est expliquée graphiquement en figure A.12.

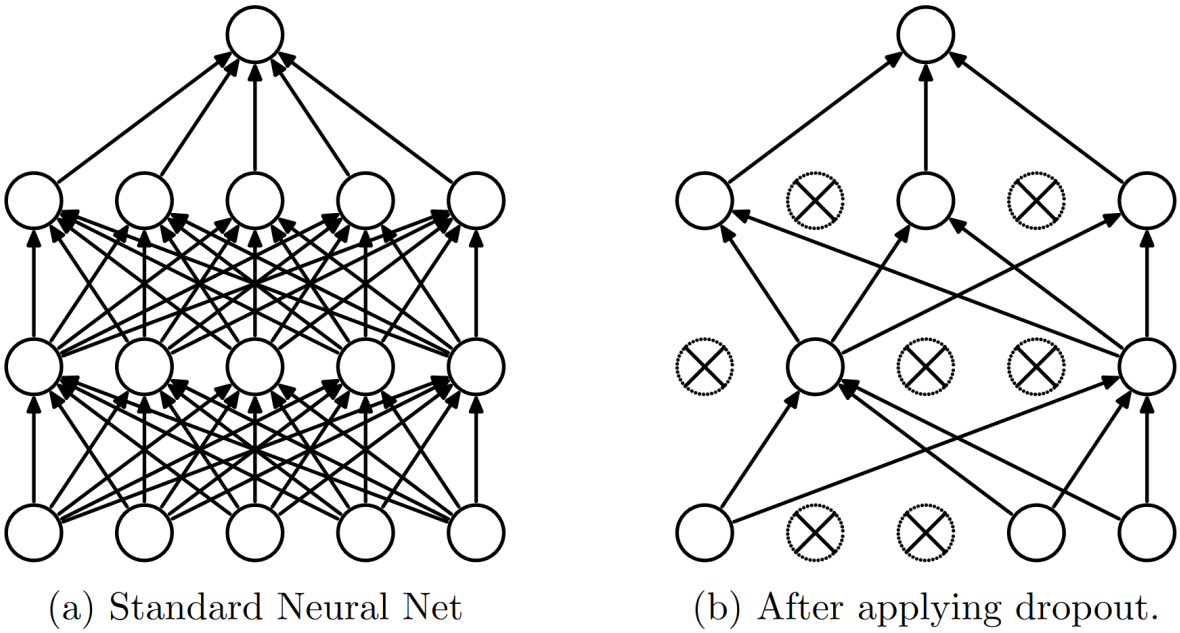


FIGURE A.12 – Mécanisme du *DropOut* pendant la phase d'entraînement (image tirée de l'article [N. SRIVASTAVA et collab., 2014]).

Pendant l'inférence en phase de test tous les neurones sont activés et redimensionnés à la

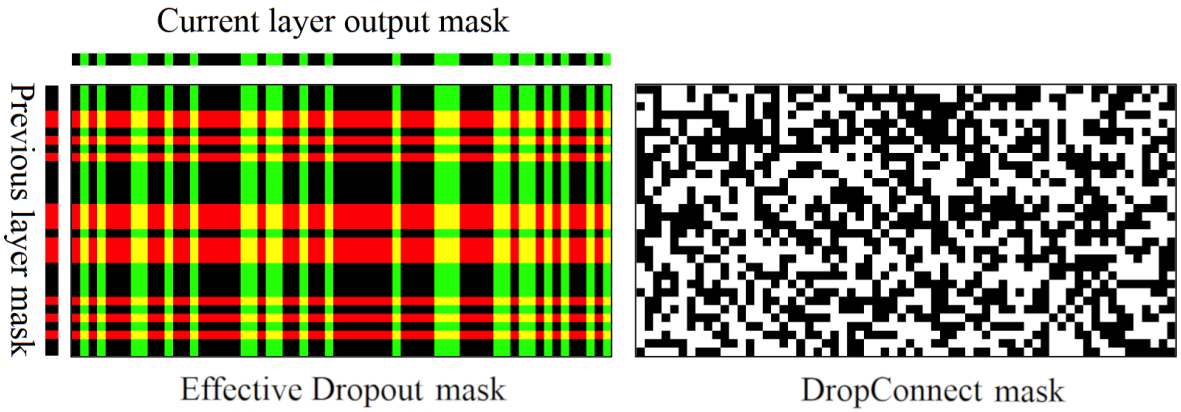
proportion d'activation pendant l'entraînement p, pour que la variance en sortie soit conservée entre l'entraînement et le test.

La couche *DropOut* est particulièrement adaptée aux réseaux de neurones quelconques (non-convolutionnels), la première façon de l'adapter aux réseaux convolutionnels est d'ajouter cette couche uniquement aux couches *FC* de classification à la fin du réseau.

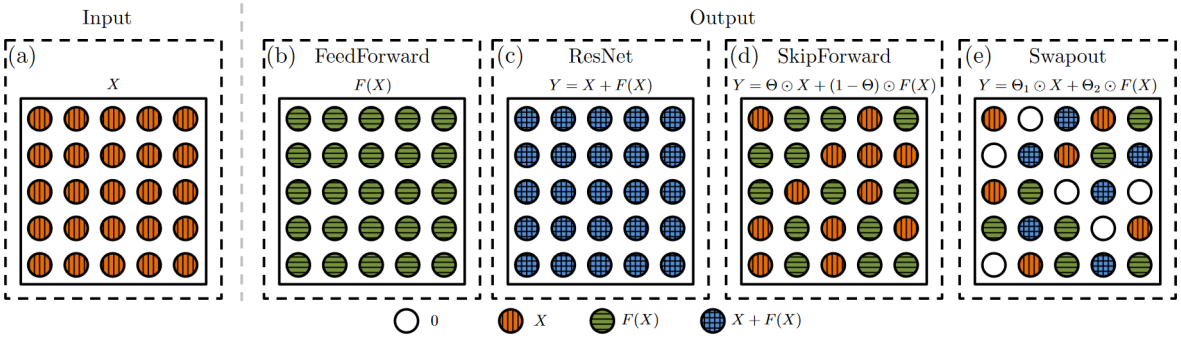
Mais l'adaptation naïve de DropOut aux couches convolutionnelles plus proches de l'entrée n'empêche pas le sur-apprentissage [TOMPSON et collab., 2015]. En effet, les neurones adjacents d'une même carte de caractéristiques sont fortement corrélés (ou non-indépendants), en éteindre un n'oblige donc pas le réseau à apprendre à ne pas utiliser l'information qu'il apporte puisque ses voisins apportent une information similaire. [TOMPSON et collab., 2015] propose donc le *Spatial-DropOut* qui, au lieu d'éteindre des neurones aléatoirement dans chaque carte de caractéristique, éteint aléatoirement chaque carte de caractéristique entièrement.

DropConnect [WAN et collab., 2013] généralise le *DropOut* en éteignant les connections entre neurones (les coefficients de la matrice de l'opérateur linéaire) plutôt que les sorties de la couche précédente (cf figure A.13a), ceci permet d'augmenter le nombre possible de combinaisons et donne de meilleurs résultats sur tous les jeux de données testés.

[G. HUANG, Y. SUN et collab., 2016] propose d'apprendre des réseaux de type *ResNet* avec une profondeur aléatoire, ceci a le même effet que le *DropOut* mais agit sur la profondeur du réseau plutôt que sur les combinaisons entre cartes de caractéristiques (plus de détails en section A.6.2 page 139). Enfin *SwapOut* proposé par [SINGH et collab., 2016], combine les méthodes de *DropOut* et de profondeur aléatoire sur *ResNet*. À chaque couche une partie aléatoire des connexions est activée, une autre partie aléatoire est éteinte et une dernière partie est transformée en identité (transmet la valeur de l'entrée sans modifications), pour augmenter encore le nombre de combinaisons possibles (cf figure A.13b).



(a) *DropConnect* (image issue de [WAN et collab., 2013])



(b) *SwapOut* (image issue de [SINGH et collab., 2016]).

FIGURE A.13 – Différentes couches qui généralisent le *DropOut*.

La couche *AlphaDropout* [KLAMBAUER et collab., 2017] est très similaire au *DropOut* mais au lieu d'éteindre certains neurones elle les active avec une valeur négative fixée $\lim_{x \longrightarrow -\infty} SELU(x)$,

adaptée aux non-linéarités des *Self Normalizing Neural Networks* pour conserver les moyennes et variances en sortie de la non-linéarité *SELU* également proposée par [KLAMBAUER et collab., 2017]

De plus, on peut interpréter la technique de *DropOut* comme une méthode de création d'ensemble implicite comme évoqué en section A.4.4 page 129. En effet, à chaque *batch* un nouveau sous-réseau est tiré parmi les 2^n sous-réseaux possibles (si n est le nombre de neurones), de cette façon chaque réseau ne sera peut-être vu et entraîné qu'une fois. Pendant la phase de test tous ces sous-réseaux (peu entraîné et donc peu apte au sur-apprentissage) sont moyennés.

La couche de *DropOut* (ou variante) peut également être activée pendant la phase de test, alors en mesurant la variance des sorties produites par les différents sous-réseaux on mesure une sorte d'incertitude du réseau pendant l'inférence [KENDALL, BADRINARAYANAN et collab., 2015] ; [GAL et GHAHRAMANI, 2016] ; [KENDALL et GAL, 2017]. Ceci est une capacité très recherchée pour l'explicabilité des méthodes d'apprentissage, en effet peu de méthodes peuvent produire ce genre de remontées d'information. En particulier pour les réseaux qui produisent une image, on obtient une image d'incertitude des résultats [KENDALL, BADRINARAYANAN et collab., 2015] ; [CARVALHO et collab., 2018], ce qui peut être utilisé par exemple pour focaliser un réseau plus précis aux endroits incertains.

A.6 Architectures

Après avoir choisi les couches les plus adaptées à notre tâche il faut encore concevoir l'architecture du réseau. On va voir que le choix de cette architecture peut en effet faciliter grandement l'entraînement du réseau ou améliorer la descriptivité des caractéristiques apprises.

A.6.1 *Inception, Group Convolution* et *Depthwise Separable Convolutions*

En image, les convolutions « standard » ont des noyaux à 3 dimensions : les deux dimensions d'espace et une dimension de profondeur (celle des cartes de caractéristiques). Une couches `Conv(128,(3 × 3))` qui prend en entrée 64 cartes de caractéristiques a donc 128 noyaux de convolutions de dimension $64 \times 3 \times 3$, ou de façon équivalente un noyau W de dimension $128 \times 64 \times 3 \times 3$. Alors la sortie de la convolution est définie de la façon suivante, au pixel (i, j) et sur la carte de caractéristique $1 \leq k \leq 128$:

$$y_{kij} = \sum_{l=1}^{64} \sum_{\hat{i}=i-1}^{i+1} \sum_{\hat{j}=j-1}^{j+1} \mathrm{W}_{kl\hat{i}\hat{j}} x_{l\hat{i}\hat{j}} \tag{A.10}$$

Les convolutions par groupes de cartes de caractéristiques, consistent à ne pas prendre toutes les caractéristiques pour chaque noyau, mais à utiliser des noyaux différents différents sous-groupes de caractéristiques (de tailles égales en général). Ce mécanisme est décrit en figure A.14 page suivante au milieu les trois blocs de convolutions 3×3 et la concaténation peuvent être vues comme une convolution par groupes (avec 3 groupes). À droite de la figure A.14 page suivante il y a le cas extrême où il y a autant de groupes que de cartes de caractéristiques.

Les convolutions par groupes ont été introduites pour la première fois dans le réseau AlexNet [KRIZHEVSKY et collab., 2012], pour pouvoir entraîner le réseau sur plusieurs GPUs en parallèle, en effet à l'époque les GPUs ne possédaient pas assez de mémoire pour réaliser l'entraînement sur une seule carte, alors chaque groupe de convolution était traité par un GPU différent en parallèle.

Mais les convolutions par groupes ont surtout l'intérêt de réduire considérablement le nombre de poids à apprendre et le nombre d'opérations à réaliser. En effet, si on a un bloc convolutionnel de noyau $k \times k$ qui prend des entrées de taille $\mathrm{C}_{in} \times \mathrm{W} \times \mathrm{H}$ et produit des sorties de taille $\mathrm{C}_{out} \times \mathrm{W} \times \mathrm{H}$, et si on note N_g le nombre de groupes (qui doit diviser C_{in} et C_{out}), alors si on suppose $\mathrm{C}_{in} = \mathrm{C}_{out} = \mathrm{C}$:

— le nombre de poids à apprendre est : $\mathrm{C}^2 k^2$

— le nombre d'opérations est : $C^2 k^2 WH$

alors que pour une convolution par groupes avec N_g groupes :

— le nombre de poids à apprendre est : $N_g \frac{C}{N_g} \frac{C}{N_g} k^2 = \frac{C^2 k^2}{N_g}$, donc un taux de réduction de $\frac{1}{N_g}$,

— le nombre d'opérations est : $N_g \frac{C}{N_g} \frac{C}{N_g} k^2 WH = \frac{C^2 k^2}{N_g} WH$, donc un taux de réduction de $\frac{1}{N_g}$,

Des variantes de convolutions par groupes ont été introduites pour réduire la taille des réseaux et le nombre d'opérations à réaliser tout en conservant des performances de classification presque aussi bonnes.

— Le module *Inception* a été introduit dans le réseau GoogLeNet [SZEGEDY, W. LIU et collab., 2015] (cf figure A.2 page 123), c'est un module qui sous certaines hypothèses est équivalent à une convolution 1×1 suivie d'une non-linéarité puis d'une convolution 3×3 par groupes avec 3 goupes (cf figure A.14).

— Les premières *Depthwise Separable Convolutions* ont étées introduites par [MAMALET et GARCIA, 2012] puis reprises dans le réseau *Xception* [CHOLLET, 2017] qui est un des réseaux les plus légers de l'état de l'art sur *ImageNet*. Elles consistent à prendre le nombre de groupes maximal possible $N_g = C$, ce qui signifie que chaque convolution d'espace agit sur une seule carte de caractéristiques. Ceci s'apparente en fait à décomposer le noyau de convolution classique en deux composantes : une sur la profondeur (les caractéristiques) et une sur la dimension d'espace, sauf qu'ici des non-linéarités sont introduites entres les deux composantes. De plus des connexions résiduelles comme décrites en section A.6.2 page ci-contre sont utilisées pour améliorer la vitesse d'entraînement.

La figure A.14 montre le parallèle entre les modules de convolution par groupes, *Inception* et *Depthwise Separable Convolutions*.

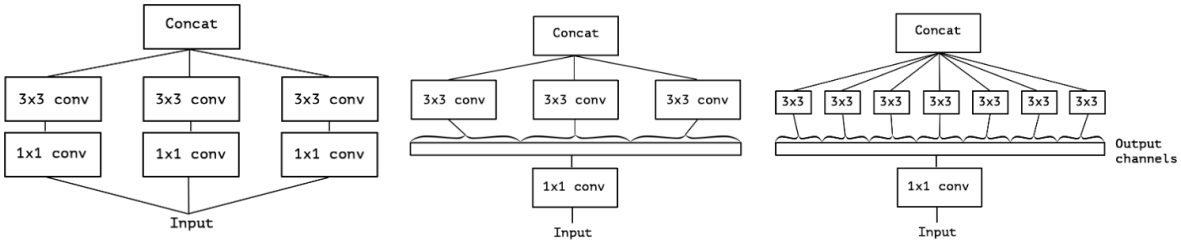


FIGURE A.14 – Interprétation des modules Inception comme *depthwize separable convolutions*. À gauche un module *Inception* simplifié, au milieu une reformulation équivalente avec une convolution par groupes, à droite le cas extrême où chaque convolution 3×3 est appliquée à une unique carte de caractéristiques, ce dernier cas correspond aux *Depthwise Separable Convolutions* (images tirées de l'article [CHOLLET, 2017]).

pour le cas extrême où $N_g = C$, on a :

— le nombre de poids à apprendre est : $N_g \frac{C}{N_g} \frac{C}{N_g} k^2 = Ck^2$, donc un taux de réduction de $\frac{1}{C}$,

— le nombre d'opérations est : $\left(N_g \frac{C}{N_g} \frac{C}{N_g} k^2\right) WH = Ck^2 WH$, donc un taux de réduction de $\frac{1}{C}$,

Ainsi pour les *depthwize separable convolutions* qui ajoute une convolution 1×1, on a :

— le nombre de poids à apprendre est : $N_g \frac{C}{N_g} \frac{C}{N_g} k^2 + CC = C(k^2 + C)$, donc un taux de réduction de $\frac{C(k^2+C)}{C^2 k^2} = \frac{1}{C} + \frac{1}{k^2}$,

— le nombre d'opérations est : $\left(N_g \frac{C}{N_g} \frac{C}{N_g} k^2 + CC\right) WH = C(k^2 + C)WH$, donc un taux de réduction de $\frac{C(k^2+C)}{C^2 k^2} = \frac{1}{C} + \frac{1}{k^2}$,

Donc dans le cas le moins favorable qu'on trouve généralement dans la littérature $C = 32$ et $k = 3$, le taux de réduction est $\frac{1}{C} + \frac{1}{k^2} \equiv 14{,}2\%$. Et il tend vers 11.1% quand C grandit.

Une autre architecture utilisant des blocs de convolutions équivalents aux *depthwize separable convolutions* est *ResNeXt* [S. XIE et collab., 2017], la seule différence est que la convolution en profondeur (noyau 1×1) est faite une fois avant et une fois après les convolutions en espace. De plus

comme *Xception* des connexions résiduelles sont utilisées et des non-linéarités sont introduites entres les convolutions sur les différentes dimensions.

[JIN et collab., 2014] pousse la décomposition à l'extrême en décomposant également les convolutions en espace selon chaque dimension d'espace, cela revient à supposer que le noyau de convolution est de rang 1 (cf figure A.15). De plus contrairement aux architectures *ResNeXt* et *Xception* aucune non-linéarité n'est ajoutée entres les différentes parties de la convolution décomposée, c'est réellement équivalent à une seule convolution de rang 1. Cependant ils observent une baisse de performance avec de telles décompositions, ils proposent donc d'en mettre deux à la suite (cf figure A.15), ce qui leur permet d'obtenir des performances équivalentes à des noyaux de convolution classiques.

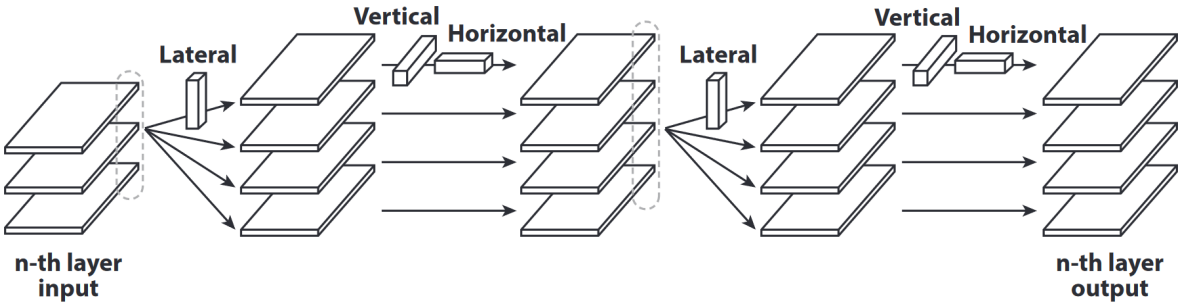


FIGURE A.15 – *flattened convolutions. Lateral* correspond à la dimension de caractéritiques et *Vertical* et *Horizontal* aux dimensions d'espace (image tirée de l'article [JIN et collab., 2014]).

A.6.2 *ResNet, DenseNet* et *Dual-Path-Network*

Le problème d'atténuation de l'information au cours du parcours du réseau, a essayé d'être reglé par initialisation des poids du réseau (section A.4.5 page 130), des non-linéarités appropriées (section A.5.1 page 131) des couches de normalisation (section A.5.2 page 134)). Une autre approche consiste à créer une architecture de réseau qui permet à l'information de se propager plus facilement dans le réseau, ceci est réalisé généralement en ajoutant des connexions qui propagent les caractéristiques dans des couches plus profondes sans les transformer en parallèle de connexions qui transforment les caractéristiques.

Les *Highway Networks* [R. K. SRIVASTAVA et collab., 2015] décrits en figure A.16a page suivante, utilisent un système de portes pour fusionner l'information transformée et l'information propagée sans transformation. La sortie y d'un tel bloc avec entrée x est donc obtenue par la formule suivante :

$$y = H_\theta(x)T_\theta(x) + x(1 - T_\theta(x)) \tag{A.11}$$

où $H_\theta(x)$ et $T_\theta(x)$ sont deux fonctions qui sont apprises pendant l'entraînement. La sortie de $T_\theta(x)$ est une couche *Sigmoid* de telle sorte que pour chaque composante de x, $T_\theta(x)$ indique s'il faut garder l'information non-transformée x ou l'information transformée $H_\theta(x)$.

Les *Residual Networks* ou *ResNets* [HE, X. ZHANG et collab., 2016] introduisent les connexions résiduelles, pour chaque groupe de deux convolutions successives il s'agit de sommer l'entrée avec la sortie de la deuxième convolution (cf figure A.3 page 123). Il est donc nécessaire que les convolutions conservent toutes les dimensions de l'entrée. La sortie y d'un tel bloc avec entrée x est donc obtenue par la formule suivante :

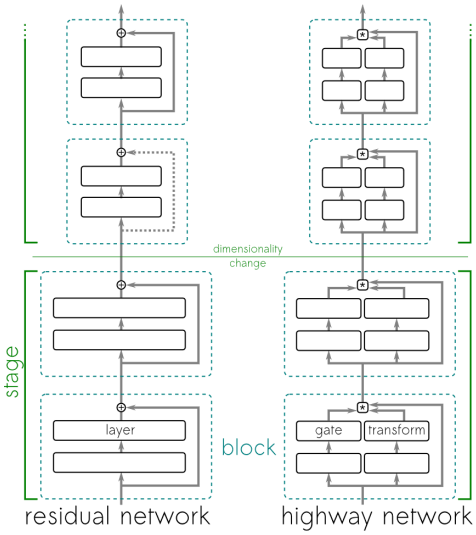
$$y = F_\theta(x) + x \tag{A.12}$$

où F_θ est la fonction constituée des deux blocs convolutionnels consécutifs. En plus de rétropropager les gradients à des poids plus proches de l'entrée sans atténuation, [HE, X. ZHANG et collab., 2016] affirme qu'apprendre les résidus est plus simple que d'apprendre une fonction proche de l'identité.

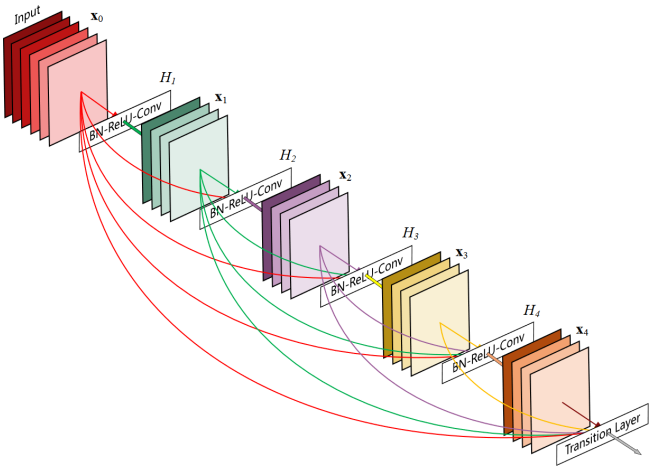
Ces connexions résiduelles ont été étendues aux blocs *Inception* au lieu de simples convolutions dans le réseau *Inception-ResNet* [SZEGEDY, IOFFE et collab., 2017]. Et grâce aux connexions résiduelles [G. HUANG, Y. SUN et collab., 2016] proposent d'apprendre des réseaux de taille aléatoire. Pendant l'entraînement les deux blocs convolutionnels entourés par une connexion résiduelle peuvent être désactivées aléatoirement pendant l'inférence de chaque *batch*, alors que pendant la phase de test tous les blocs convolutionnels sont activés. C'est une approche similaire au *DropOut* qui agit sur la profondeur du réseau et donc tente d'éviter la « co-adaptation » des couches consécutives plutôt que celle des neurones d'une même couche.

On peut remarquer que l'architecture *ResNet* est équivalente au *Highway Networks* si on pose $F_\theta(x) = T_\theta(x)(H_\theta(x) - x)$, ou encore qu'une connexion résiduelle est un système de porte similaire aux *Highway Networks* avec des portes constamment ouvertes à 100%. Ceci se confirme par l'observation faite par [GREFF et collab., 2016] que en supprimant un bloc avec connexion résiduelle ou *Highway*, l'erreur de classification n'augmente que très peu, contrairement aux réseau VGG pour lequel l'erreur monte à 90% presque aléatoire [VEIT et collab., 2016]. [GREFF et collab., 2016] remet donc en cause la vision de l'apprentissage profond qui consiste à penser que chaque nouvelle couche crée des caractéristiques totalement nouvelles et plus abstraites à partir des précédentes, mais estime que les blocs *Highway* et résiduels consécutifs rafines progressivement des caractéristiques (un peu comme un réseau récurrent déroulé).

L'architecture *DenseNet* [G. HUANG, Z. LIU et collab., 2017] reprend l'idée de propager une information non-transformée plus loin dans le réseau, mais la pousse à l'extrême en communiquant la sortie de chaque couche à toutes les couches suivantes (cf figure A.16b). Et au lieu de fusionner les données transformées et non-transformées en les sommant (ce qui requiert le même nombre de cartes de caractéristiques et de mêmes dimensions), elles sont fusionnées par concaténation dans l'espace des caractéristiques. En pratique cette approche n'est utilisée que dans des blocs denses situés entre les couches de sous-échantillonages (convolutions à pas ou *pooling*).



(a) Architecture de *ResNet* (à gauche) et de *Highway Network* (à droite) (image issue de [GREFF et collab., 2016])



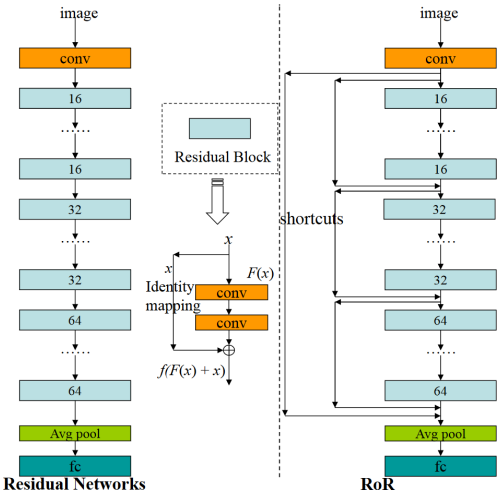
(b) Architecture de *DenseNet* (image issue de [G. HUANG, Z. LIU et collab., 2017]).

FIGURE A.16 – Différentes architectures qui permettent de propager parallèlement de l'information plus ou moins transformée.

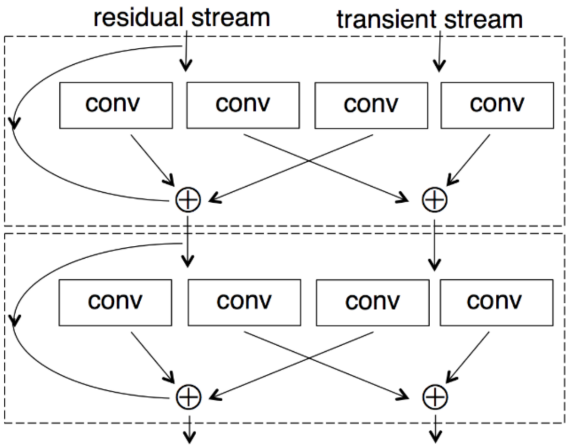
Les connexions résiduelles ne propagent le flux d'information que deux couches plus loin, ainsi plusieurs architecture ont tenté de généraliser les connexions résiduelles :

— *ResNet of ResNet* [K. ZHANG et collab., 2018] imbrique les connexions résiduelles les unes à l'intérieur des autres (cf figure A.17a page ci-contre) pour propager l'information plus loin dans le réseau. À tel point que la connexion résiduelle de plus haut niveau fait communiquer l'entrée du réseau avec la sortie des couches convolutionnels (juste avant les couches FC),

— l'architecture *ResNet in ResNet* [TARG et collab., 2016] gère deux flux de convolutions (un avec connexions résiduelles et un sans), ces deux flux étant intriqués, mais le réseau est conçu de telle façon qu'il puisse apprendre comment connecter les deux flux (cf figure A.17b)



(a) L'architecture *ResNet of ResNet* (image issue de [K. ZHANG et collab., 2018]).



(b) L'architecture *ResNet in ResNet* (image issue de [TARG et collab., 2016]).

FIGURE A.17 – Architectures proposées pour généraliser les connexions résiduelles

[Yunpeng CHEN et collab., 2017] propose les *Dual Path Networks*, une architecture qui combine la réutilisation de caractéristiques grâce aux connexions résiduelles des *ResNets* et l'exploration de nouvelles caractéristiques grâce aux *DenseNet* (cf figure A.18).

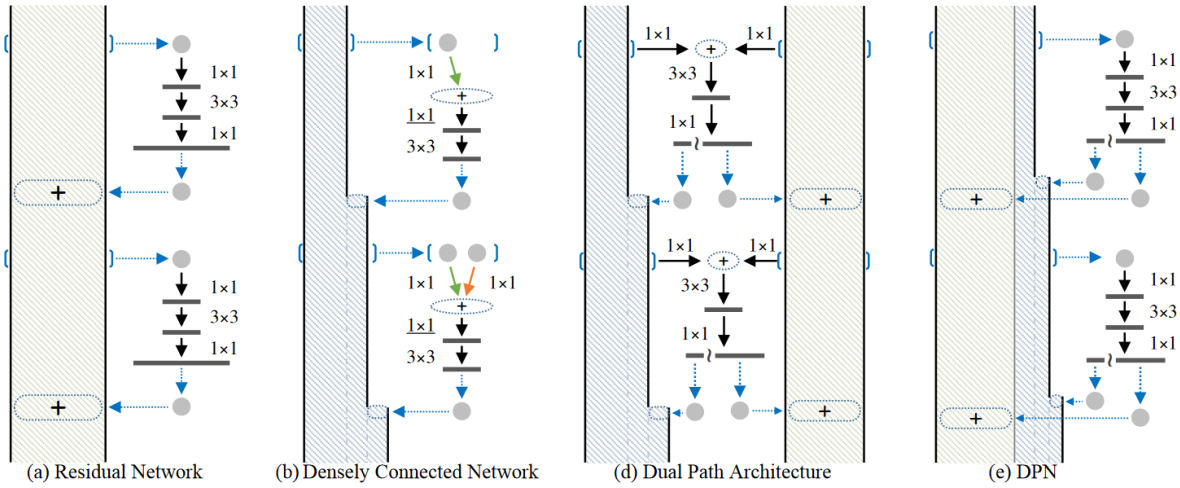


FIGURE A.18 – Architecture du *Dual Path Networks* qui allie réutilisation de caractéritiques grâce aux connexions résiduelles et exploration de nouvelles caractéristiques grâce à l'architecture de *DenseNet* (image issue de [Yunpeng CHEN et collab., 2017]).

A.6.3 Architectures pour la segmentation sémantique

Les réseaux de classification d'image agrègent progressivement l'information en réduisant la taille de cartes de caractéristiques (en sous-échantillonnant) jusqu'à avoir une information agrégée sur toute l'image, pour obtenir une information par pixel il faut « dés-agréger » (sur-échantillonner) l'information pour revenir à une image de résolution identique à celle de l'image initiale. Alors que le sous-échantillonnage est réalisé par des couches `Pooling` ou `Conv` de *stride* > 1, le sur-

échantillonnage est réalisé de façon symétrique par des couches `UnPooling` ou `ConvTranspose` de *stride* > 1.

Le réseau FCN (pour *Fully Convolutionnal Network*) [LONG et collab., 2015] est un des premiers réseaux convolutionnels pour la segmentation d'images. Il reprend n'importe quelle architecture de réseau convolutionnel de classification d'image, et le transforme en réseau de segmentation d'image en 4 étapes (cf figure A.19) :

— transforme les couches `FC` en couches `Conv` à noyaux 1×1 (avec les mêmes poids),

— récupère la sortie de la dernière couche du réseau et applique une `ConvTranspose` de *stride* suffisante pour revenir à la résolution initiale de l'image,

— applique une fonction de coût par pixel de telle sorte que le réseau puisse être entraîné de bout en bout,

— éventuellement raffine le résultat obtenu en fusionnant avec des sorties de couches plus résolues (plus proches de l'entrée), la fusion est réalisée avant le sur-échantillonnage par `ConvTranspose` en sommant les prédictions sur chaque pixel (FCN-16s et FCN-8s sur la figure A.19).

Le gros inconvénient du réseau *FCN* est que dans ses dernières couches il sur-échantillonne très brutalement ce qui empêche :

— de propager et désagréger progressivement les caractéristiques abstraites obtenues à résolution faibles,

— d'obtenir des frontières fines des objets (aussi fines que sur l'image en entrée),

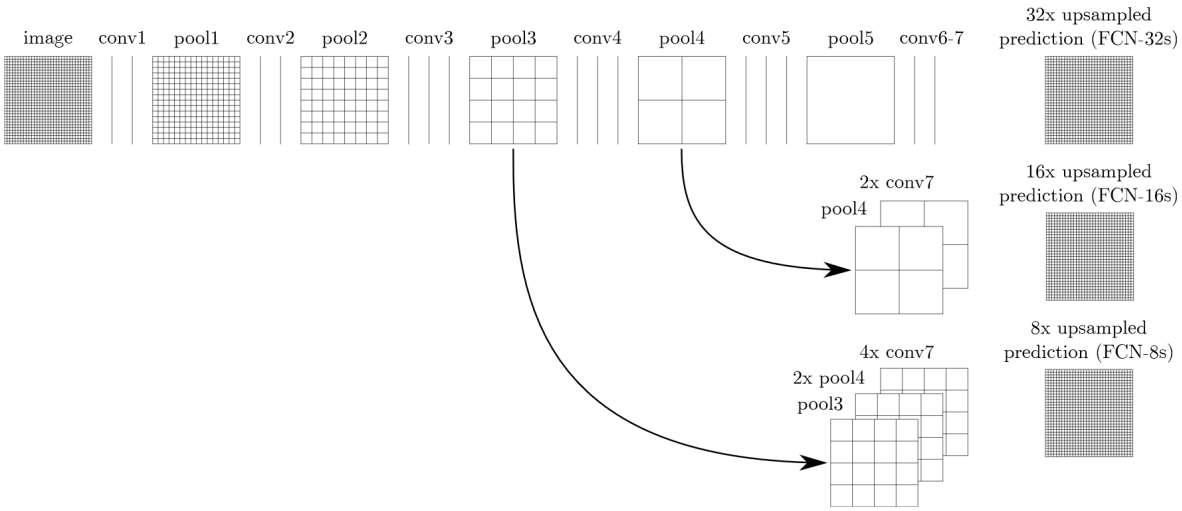


FIGURE A.19 – Architecture du réseau FCN. (images tirées de [LONG et collab., 2015])

Les deux architectures *SegNet* [BADRINARAYANAN et collab., 2017] et *U-Net* [RONNEBERGER et collab., 2015] résolvent ces inconvénients. Le premier est résolu en ajoutant une partie déconvolutionnelle progressive symétrique de la partie convolutionnelle, le second est résolu en propageant d'une certaine façon la localisation où les activations sont « les plus importantes » de la partie convolutionnelle vers la partie déconvolutionnelle grâce à des connexions appelées *skip connections* en anglais.

— l'architecture *SegNet* [BADRINARAYANAN et collab., 2017] est décrite en figure A.20 page suivante, ses *skip connections* consistent à transmettre la position des pixels qui ont été conservés dans les couches de `MaxPooling` aux couches de `UnPooling` correspondantes (symétriques). Les couches de `UnPooling` produisent alors une image de résolution doublée en positionnant les pixels de l'image basse résolution grâce aux positions utilisées par le `MaxPooling` et en mettant des 0 sur tous les autres pixels. L'ensemble de l'image est alors remplie par des convolutions.

— l'architecture *U-Net* [RONNEBERGER et collab., 2015] est décrite en figure A.21, ses *skip connections* consistent à transmettre entièrement les entrées des couches de sous-échantillonnages et à les concaténer avec les sorties des couches de sur-échantillonnage symétriques, pour communiquer la « finesse » des activations qui a pu être perdue par les convolutions successives.

L'architecture *U-Net* a l'avantage de propager plus d'information (l'ensemble des cartes de caractéristiques) via ses *skip connections* que juste la position de certaines caractéristiques.

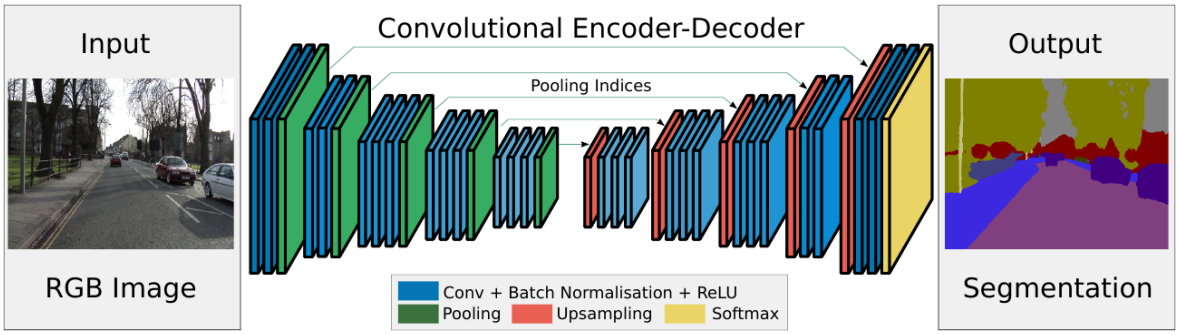


FIGURE A.20 – Architecture du réseau *SegNet.* (images tirées de [BADRINARAYANAN et collab., 2017])

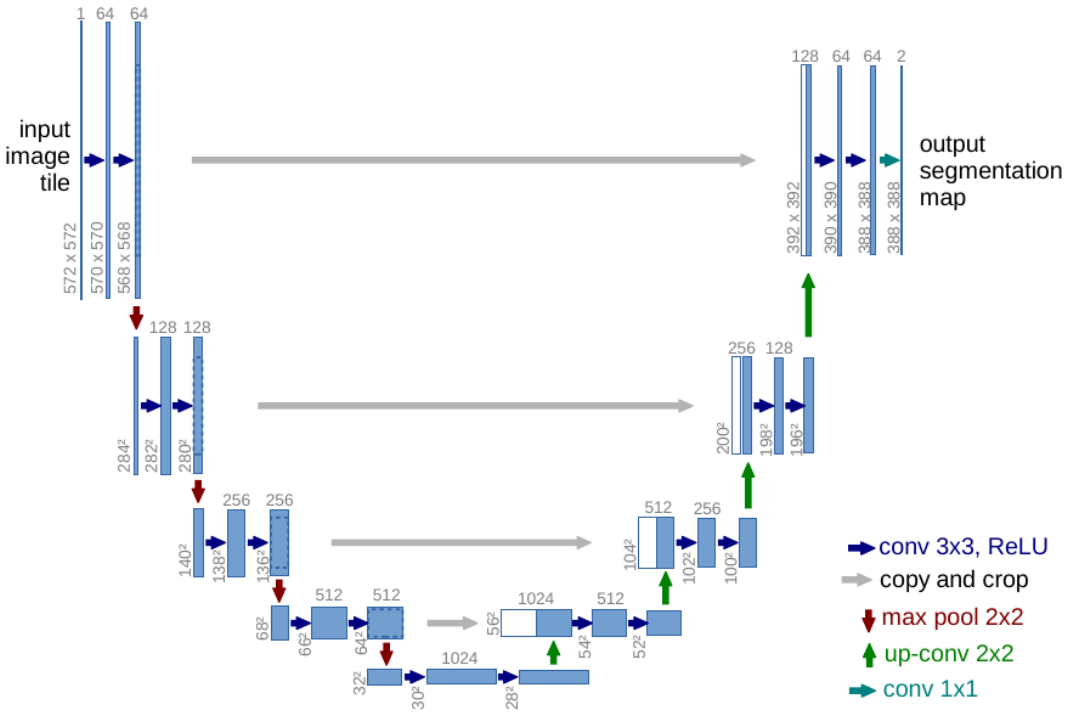


FIGURE A.21 – Architecture du réseau *U-Net.* (image tirée de [RONNEBERGER et collab., 2015])

Il existe également des méthodes qui prennent en entrée une cascade multi-échelles d'entrées, le but étant de commencer à inférer des classes sur des versions basse résolution de l'image, puis de progressivement remonter à la résolution initiale à moindre coût que si on traitait directement l'image initiale (cf figure A.22 page suivante). Cette approche permet à *ICNet* [ZHAO et collab., 2017] de tourner en temps réel sur des images haute résolution.

A.6.4 Conception d'architectures

On a pu voir qu'il existait une quantité phénoménale de possibilités pour connecter les différentes couches d'un réseau de neurones convolutionnel, ainsi que pour choisir les différents

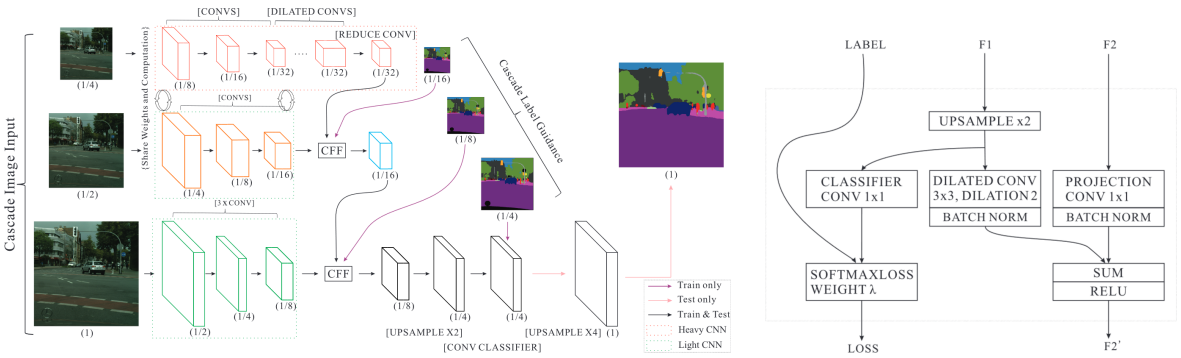


FIGURE A.22 – Architecture du réseau ICNet. CFF signifie *Cascade Feature Fusion*, ce module est décrit en image de droite (image tirée de [ZHAO et collab., 2017]).

hyper-paramètres de chaque couche. Il y a donc de plus en plus de recherche sur la façon de trouver des architectures optimales selon certains critères. Il y a différentes approches :

— L'architecture globale du réseau est fixée comme une suite de blocs convolutionnels et de blocs de sous-échantillonnage et la méthode cherche les blocs génériques optimaux en combinant tout un bestiaire de couches possibles (différents type de convolutions, inception, residuals, normalisation, pooling...). C'est l'approche utilisée pour obtenir le réseau *NasNet* [ZOPH et collab., 2017], les différents blocs sont générés par un réseau de neurones récurrent (*RNN*) appelé « contrôleur ». Le réseau échantillonné est alors entraîné jusqu'à convergence et ses performances sont calculées sur un jeu de validation. Le contrôleur est entraîné (par les mêmes méthodes de descente de gradient que les réseaux convolutionnels) pour minimiser l'erreur obtenue par le réseau échantillonné sur le jeu de validation.

— La méthode recherche plutôt l'architecture globale optimale sans trop se focaliser sur le type de blocs convolutionnels, mais pas sur la façon dont l'information se propage entre les différents sous-échantillonnages et sur l'utilisation ou pas de flux parallèles et comment les connecter. C'est l'approche utilisée par les *Convolutional neural fabrics* de [SAXENA et VERBEEK, 2016] et [VENIAT et DENOYER, 2017]

Les réseaux générés pas [ZOPH et collab., 2017] peuvent nécessiter plusieurs jours de calculs sur des centaines de GPUs, [PHAM et collab., 2018] améliore cette méthode en conservant les poids des réseaux entraînés, ce qui évite d'avoir à ré-entraîner chaque réseau à partir de zéro, et permet un taux de réduction de ×1000. Ceci est réalisé en plongeant chaque « petit » réseau entraîné dans un « gros » réseau les englobant tous, ainsi quand un petit réseau est généré par le contrôleur (*RNN*) comme sous-réseau, si certaines de ses couches ont déjà été utilisées par un autre petit réseau, les poids déjà entraînés sont réutilisés.

De plus [SAXENA et VERBEEK, 2016] utilise une visualisation des réseaux (cf figure A.23 page ci-contre) qui permet de se représenter facilement l'évolution de la forme des échantillons à travers les couches d'un réseau. On utilisera une variante de cette visualisation en section 2.3 page 53 pour décrire nos réseaux multi-échelles.

[VENIAT et DENOYER, 2017] trouvent des architectures parmi celles décrites par une variante de *Convolutional neural fabrics*. Certaines contraintes sont ajoutées, mais pas seulement de performances de classification, par exemple de nombre d'opérations maximal, de place mémoire maximale ou de capacité de passage (cf figure A.24 page ci-contre pour différentes architectures découvertes par [VENIAT et DENOYER, 2017]). Ici ce n'est pas un contrôleur qui génère une architecture, mais les différentes couches et connexions sont tirées aléatoirement selon des distributions de probabilités sur chaque couche, arête ou nœud de la *Convolutional neural fabrics*, ce sont les paramètres de ces distributions qui sont appris par descente de gradients.

A.6.5 Architectures pour la détection

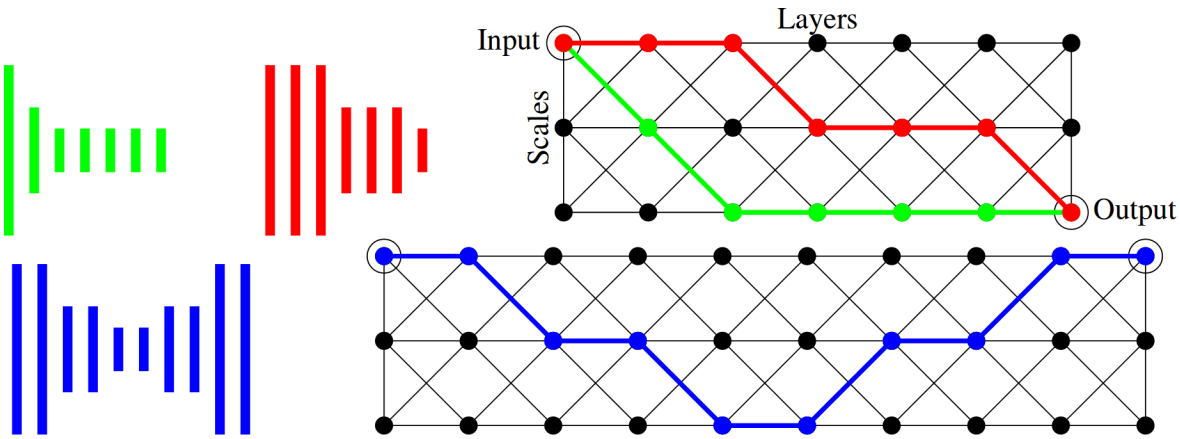
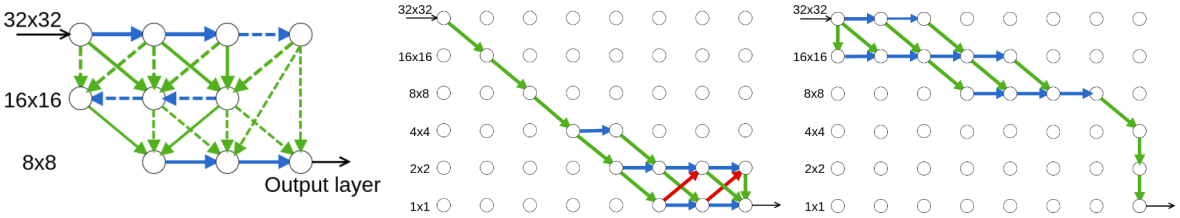


FIGURE A.23 – Visualisation des *Convolutional neural fabrics*. En haut deux réseaux de classification et en bas un réseau de segmentation. La troisième dimension représentant le nombre de couches en parallèle qui peuvent se combiner par exemple en sommant ou concaténant leurs sorties (image tirée de [SAXENA et VERBEEK, 2016])



(a) B-ResNet (b) B-CNF & computation cost (c) B-CNF & memory consumption cost

FIGURE A.24 – Visualisation des meilleures architectures trouvées par [VENIAT et DENOYER, 2017] (image tirée de l'article).

Parmi les architectures qui découlent directement des réseaux de neurones convolutionnels pour la classification, il y a également les réseaux de détection et classification d'images. Leur but n'est pas de donner une classe à chaque pixel, mais de détecter chaque objet de la scène (au moins les plus importants), c'est-à-dire trouver une boite englobante de l'objet (ou même un masque de pixels), et de classifier l'objet détecté.

Toutes les architectures qui réalisent cette tâche utilisent des réseaux de l'état de l'art en classification d'image comme une sous-partie de l'architecture. Et ces méthodes peuvent être divisée en deux approches :

— les architectures composées d'une première partie qui propose des régions, puis d'une seconde qui classifie chaque région, comme *R-CNN* [GIRSHICK et collab., 2014] et ses variantes,

— les architectures en une seule partie (qui ressemblent beaucoup plus à des réseau de segmentation sémantique) qui pour chaque pixel prédit à la fois la présence d'un objet et la boite englobante à laquelle il appartient, comme *SSD* [W. LIU et collab., 2016] et *YOLO* [REDMON et collab., 2016].

Dans *R-CNN* [GIRSHICK et collab., 2014] un algorithme de recherche de région [UIJLINGS et collab., 2013] propose environ 2000 régions (des rectangles uniquement), chaque région est alors redimensionnée sous forme d'une image carrée de taille 227×227, cette image est donnée à un réseau de classification de l'état de l'art. Cependant on ne récupère pas la sortie de la dernière couche du réseau (un `SoftMax`), mais la sortie de la partie convolutionnelle qui est un vecteur de taille 4096. À partir de ce descripteur de l'image, un classificateur de type *Support Vector Machine* (*SVM*) est utilisé pour classifier chaque région, ainsi que des corrections pour améliorer les frontières de la région. Cette méthode est décrite en figure A.25 page suivante.

R-CNN peut prendre jusqu'à une minute de calcul pour détecter les objets d'une image ce qui

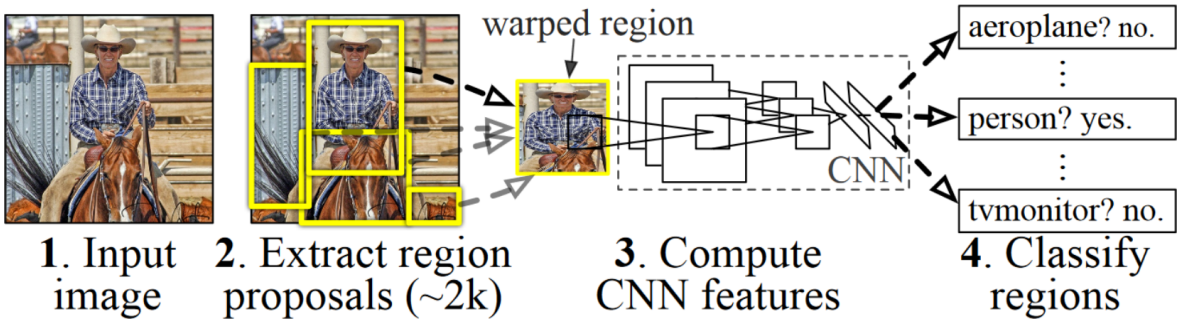


FIGURE A.25 – Architecture du réseau *R-CNN*. (image tirée de [GIRSHICK et collab., 2014])

est très loin du temps réel, ceci est du au grand nombre de régions données au réseau convolutionnel. Or pour les régions qui se chevauchent, les calculs de convolutions sont les mêmes. Il est donc beaucoup plus efficace d'appliquer le réseau convolutionnel directement à l'image entière, puis récupérer les caractéristiques de chaque région en sortie du réseau. C'est ce qui est fait dans *Fast-RCNN* [GIRSHICK, 2015], c'est décrit en figure A.26.

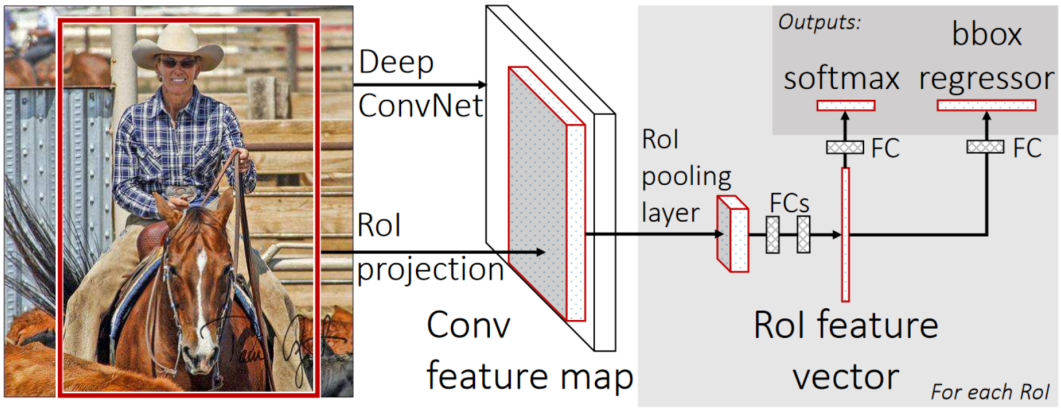


FIGURE A.26 – Architecture du réseau *Fast-RCNN* sans la partie qui propose les régions (image tirée de [GIRSHICK, 2015])

Un inconvénient de *R-CNN* et *Fast-RCNN* est que l'étape de proposition des régions reste « non-apprise » et donc très dépendante de l'algorithme de recherche de région de [UIJLINGS et collab., 2013] qui peut avoir ses défauts, entre autres de proposer un nombre fixe de régions (environ 2000 ce qui est assez élevé). *Faster-RCNN* [S. REN et collab., 2015] utilise plutôt un *Region Proposal Network* (*RPN*), c'est un réseau qui prend en entrée la sortie du réseau convolutionnel pour proposer des régions où il peut y avoir un objet. Les boites englobantes sont alors utilisées pour extraire des caractéristiques de l'image et classifier l'objet comme pour *Fast-RCNN*. Cette méthode est décrite en figure A.27 page suivante. Le réseau peut alors être entraîné intégralement de bout-en-bout.

Un inconvénient de ces méthodes est qu'elles ne produisent que la boite englobante contenant l'objet, mais pas précisément les frontières de l'objet. *Mask-RCNN* [HE, GKIOXARI et collab., 2017] résout ce problème en ajoutant une sortie (entraînée simultanément du reste du réseau) produisant un masque des pixels appartenant à l'objet à l'intérieur de sa boite englobante. Cette architecture est d'écrite en figure A.28 page ci-contre.

Les deux méthodes suivantes agissent en une seule étape, et produisent simultanément les classes et les boîtes englobantes des objets présents sur l'image, elle sont décrites en détail en figure A.29 page suivante.

— *YOLO* (pour *You Only Look Once*) [REDMON et collab., 2016],

— *SSD* (pour *Single Shot Detector*) [W. LIU et collab., 2016],

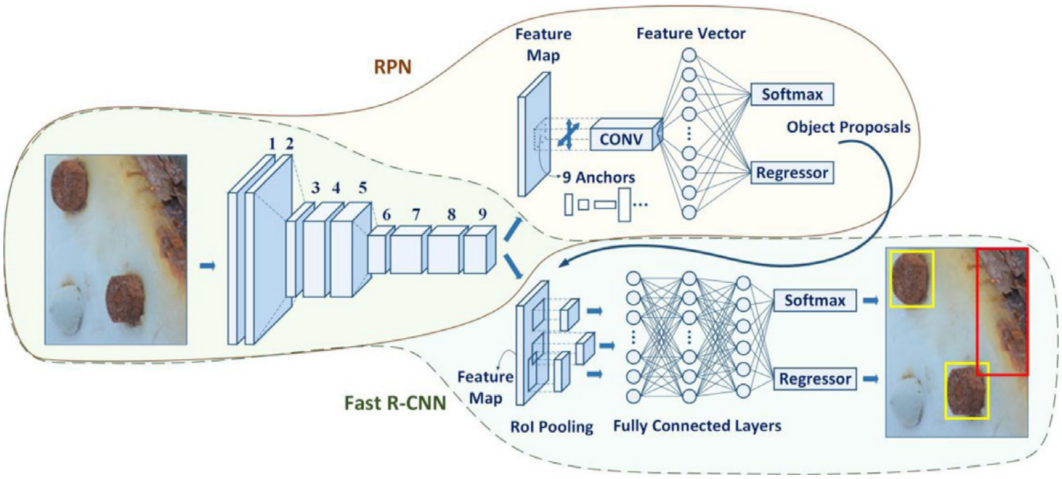


FIGURE A.27 – Architecture du *Region Proposal Network* (*RPN*) utilisé par *Faster-RCNN* (image tirée de [CHA et collab., 2018]).

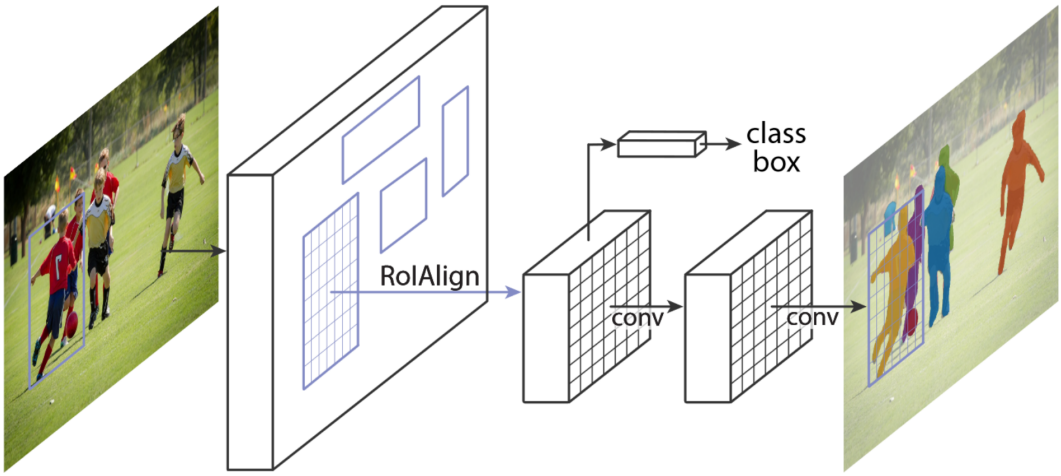


FIGURE A.28 – Architecture du réseau *Mask-RCNN*. (image tirée de [HE, GKIOXARI et collab., 2017])

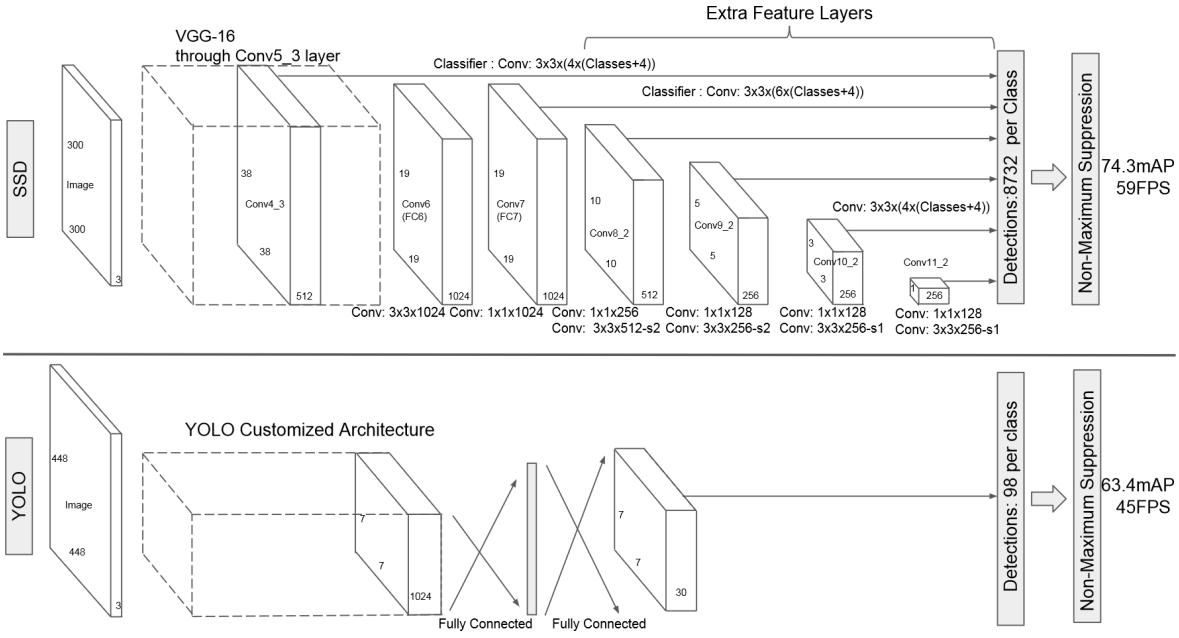


FIGURE A.29 – Architectures des réseaux *SSD* (en haut) et *YOLO* (en bas). (image tirée de [W. LIU et collab., 2016])

Références

BA, Jimmy Lei, Jamie Ryan KIROS et Geoffrey E HINTON (2016). « Layer normalization ». In : *arXiv preprint arXiv:1607.06450* (cf. p. 134).

BADRINARAYANAN, Vijay, Alex KENDALL et Roberto CIPOLLA (2017). « SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation ». In : *IEEE Transactions on Pattern Analysis & Machine Intelligence* 12, p. 2481-2495 (cf. p. 142, 143).

BARRON, Jonathan T (2017). « Continuously Differentiable Exponential Linear Units ». In : *arXiv preprint arXiv:1704.07483* (cf. p. 133).

BELLO, Irwan, Barret ZOPH, Vijay VASUDEVAN et Quoc V LE (2017). « Neural optimizer search with reinforcement learning ». In : *arXiv preprint arXiv:1709.07417* (cf. p. 127).

BENGIO, Yoshua, Pascal LAMBLIN, Dan POPOVICI et Hugo LAROCHELLE (2007). « Greedy layer-wise training of deep networks ». In : *Advances in neural information processing systems*, p. 153-160 (cf. p. 130, 131).

BENVENISTE, Albert, Pierre PRIOURET et Michel MÉTIVIER (1990). « Adaptive algorithms and stochastic approximations ». In : (cf. p. 126).

BOTTOU, Léon et Yann LE CUN (2005). « On-line learning for very large data sets ». In : *Applied stochastic models in business and industry* 21.2, p. 137-151 (cf. p. 126).

CARUANA, Rich, Alexandru NICULESCU-MIZIL, Geoff CREW et Alex KSIKES (2004). « Ensemble selection from libraries of models ». In : *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 18 (cf. p. 129).

CARVALHO, Marcela, Bertrand Le SAUX, Pauline TROUVÉ-PELOUX, Andrés ALMANSA et Frédéric CHAMPAGNAT (2018). « Deep Depth from Defocus: how can defocus blur improve 3D estimation using dense neural networks? » In : *arXiv preprint arXiv:1809.01567* (cf. p. 137).

CHA, Young-Jin, Wooram CHOI, Gahyun SUH, Sadegh MAHMOUDKHANI et Oral BÜYÜKÖZTÜRK (2018). « Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types ». In : *Computer-Aided Civil and Infrastructure Engineering* 33.9, p. 731-747 (cf. p. 147).

CHEN, Yunpeng, Jianan LI, Huaxin XIAO, Xiaojie JIN, Shuicheng YAN et Jiashi FENG (2017). « Dual Path Networks ». In : *Advances in Neural Information Processing Systems 30*. Sous la dir. d'I. GUYON, U. V. LUXBURG, S. BENGIO, H. WALLACH, R. FERGUS, S. VISHWANATHAN et R. GARNETT. Curran Associates, Inc., p. 4467-4475. URL : http://papers.nips.cc/paper/7033-dual-path-networks.pdf (cf. p. 141).

CHOLLET, Francois (juil. 2017). « Xception: Deep Learning With Depthwise Separable Convolutions ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 90, 138).

CIREŞAN, Dan C, Ueli MEIER, Jonathan MASCI, Luca M GAMBARDELLA et Jürgen SCHMIDHUBER (2011). « High-performance neural networks for visual object classification ». In : *arXiv preprint arXiv:1102.0183* (cf. p. 125).

CIREŞAN, Dan, Ueli MEIER et Jürgen SCHMIDHUBER (2012). « Multi-column deep neural networks for image classification ». In : *arXiv preprint arXiv:1202.2745* (cf. p. 124).

CLEVERT, Djork-Arné, Thomas UNTERTHINER et Sepp HOCHREITER (2015). « Fast and accurate deep network learning by exponential linear units (elus) ». In : *arXiv preprint arXiv:1511.07289* (cf. p. 133).

CYBENKO, George (1989). « Approximation by superpositions of a sigmoidal function ». In : *Mathematics of control, signals and systems* 2.4, p. 303-314 (cf. p. 118).

DEVRIES, Terrance et Graham W. TAYLOR (août 2017). « Improved Regularization of Convolutional Neural Networks with Cutout ». In : *arXiv e-prints*, arXiv:1708.04552, arXiv:1708.04552. arXiv : 1708.04552 [cs.CV] (cf. p. 125).

DUCHI, John, Elad HAZAN et Yoram SINGER (2011). « Adaptive subgradient methods for online learning and stochastic optimization ». In : *Journal of Machine Learning Research* 12.Jul, p. 2121-2159 (cf. p. 126).

GAL, Yarin et Zoubin GHAHRAMANI (2016). « Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning ». In : *International Conference on Machine Learning*, p. 1050-1059 (cf. p. 137).

GIRSHICK, Ross (déc. 2015). « Fast R-CNN ». In : *The IEEE International Conference on Computer Vision (ICCV)* (cf. p. 146).

GIRSHICK, Ross, Jeff DONAHUE, Trevor DARRELL et Jitendra MALIK (juin 2014). « Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 145, 146).

GLOROT, Xavier et Yoshua BENGIO (2010). « Understanding the difficulty of training deep feedforward neural networks ». In : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, p. 249-256 (cf. p. 130, 132).

GLOROT, Xavier, Antoine BORDES et Yoshua BENGIO (2011). « Deep sparse rectifier neural networks ». In : *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, p. 315-323 (cf. p. 132).

GREFF, Klaus, Rupesh K SRIVASTAVA et Jürgen SCHMIDHUBER (2016). « Highway and residual networks learn unrolled iterative estimation ». In : *arXiv preprint arXiv:1612.07771* (cf. p. 140).

GULCEHRE, Caglar, Marcin MOCZULSKI, Misha DENIL et Yoshua BENGIO (2016). « Noisy activation functions ». In : *International Conference on Machine Learning*, p. 3059-3068 (cf. p. 131, 132).

HE, Kaiming, Georgia GKIOXARI, Piotr DOLLÁR et Ross GIRSHICK (2017). « Mask r-cnn ». In : *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, p. 2980-2988 (cf. p. 46, 146, 147).

HE, Kaiming, Xiangyu ZHANG, Shaoqing REN et Jian SUN (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. DOI : 10.1109/ICCV.2015.123. arXiv : 1502.01852. URL : http://arxiv.org/abs/1502.01852%20http://ieeexplore.ieee.org/document/7410480/ (cf. p. 131, 133).

HE, Kaiming, Xiangyu ZHANG, Shaoqing REN et Jian SUN (juin 2016). « Deep Residual Learning for Image Recognition ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 120, 139).

HINTON, Geoffrey E et Ruslan R SALAKHUTDINOV (2006). « Reducing the dimensionality of data with neural networks ». In : *science* 313.5786, p. 504-507 (cf. p. 130, 131).

HOWARD, Jeremy et Sebastian RUDER (2018). « Fine-tuned Language Models for Text Classification ». In : *arXiv preprint arXiv:1801.06146* (cf. p. 128).

HUANG, Gao, Yixuan LI, Geoff PLEISS, Zhuang LIU, John E HOPCROFT et Kilian Q WEINBERGER (2017). « Snapshot ensembles: Train 1, get M for free ». In : *arXiv preprint arXiv:1704.00109* (cf. p. 129, 130).

HUANG, Gao, Zhuang LIU, Laurens van der MAATEN et Kilian Q. WEINBERGER (juil. 2017). « Densely Connected Convolutional Networks ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 140).

HUANG, Gao, Yu SUN, Zhuang LIU, Daniel SEDRA et Kilian Q WEINBERGER (2016). « Deep networks with stochastic depth ». In : *European Conference on Computer Vision*. Springer, p. 646-661 (cf. p. 136, 140).

IOFFE, Sergey (2017). « Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models ». In : *Advances in Neural Information Processing Systems 30*. Sous la dir. d'I. GUYON, U. V. LUXBURG, S. BENGIO, H. WALLACH, R. FERGUS, S. VISHWANATHAN et R. GARNETT. Curran Associates, Inc., p. 1945-1953. URL : http://papers.nips.cc/paper/6790-batch-renormalization-towards-reducing-minibatch-dependence-in-batch-normalized-models.pdf (cf. p. 134).

IOFFE, Sergey et Christian SZEGEDY (2015). « Batch normalization: Accelerating deep network training by reducing internal covariate shift ». In : *arXiv preprint arXiv:1502.03167* (cf. p. 134).

JIN, Jonghoon, Aysegul DUNDAR et Eugenio CULURCIELLO (2014). « Flattened convolutional neural networks for feedforward acceleration ». In : *arXiv preprint arXiv:1412.5474* (cf. p. 139).

KENDALL, Alex, Vijay BADRINARAYANAN et Roberto CIPOLLA (2015). « Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding ». In : *arXiv preprint arXiv:1511.02680* (cf. p. 137).

KENDALL, Alex et Yarin GAL (2017). « What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? » In : *Advances in Neural Information Processing Systems 30*. Sous la dir. d'I. GUYON, U. V. LUXBURG, S. BENGIO, H. WALLACH, R. FERGUS, S. VISHWANATHAN et R. GARNETT. Curran Associates, Inc., p. 5574-5584. URL : http://papers.nips.cc/paper/7141-what-uncertainties-do-we-need-in-bayesian-deep-learning-for-computer-vision.pdf (cf. p. 137).

KINGMA, Diederik P et Jimmy BA (2014). « Adam: A method for stochastic optimization ». In : *arXiv preprint arXiv:1412.6980* (cf. p. 127).

KLAMBAUER, Günter, Thomas UNTERTHINER, Andreas MAYR et Sepp HOCHREITER (2017). « Self-Normalizing Neural Networks ». In : *Advances in Neural Information Processing Systems 30*. Sous la dir. d'I. GUYON, U. V. LUXBURG, S. BENGIO, H. WALLACH, R. FERGUS, S. VISHWANATHAN et R. GARNETT. Curran Associates, Inc., p. 971-980. URL : http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf (cf. p. 133, 135-137).

KRIZHEVSKY, Alex, Ilya SUTSKEVER et Geoffrey E HINTON (2012). « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems*, p. 1097-1105 (cf. p. 3, 119, 137).

LECUN, Yann, Léon BOTTOU, Yoshua BENGIO et Patrick HAFFNER (1998). « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11, p. 2278-2324 (cf. p. 119).

LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU et Alexander C BERG (2016). « Ssd: Single shot multibox detector ». In : *European conference on computer vision*. Springer, p. 21-37 (cf. p. 46, 145-147).

LONG, Jonathan, Evan SHELHAMER et Trevor DARRELL (juin 2015). « Fully Convolutional Networks for Semantic Segmentation ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 120, 142).

LOSHCHILOV, Ilya et Frank HUTTER (2016). « Sgdr: Stochastic gradient descent with warm restarts ». In : *arXiv preprint arXiv:1608.03983* (cf. p. 128, 129).

MAAS, Andrew L, Awni Y HANNUN et Andrew Y NG (2013). « Rectifier nonlinearities improve neural network acoustic models ». In : *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Citeseer (cf. p. 132).

MAMALET, Franck et Christophe GARCIA (2012). « Simplifying convnets for fast learning ». In : *International Conference on Artificial Neural Networks*. Springer, p. 58-65 (cf. p. 138).

MISHKIN, Dmytro et Jiri MATAS (nov. 2015). « All you need is a good init ». In : *ArXiv e-prints*, arXiv:1511.06422, arXiv:1511.06422. arXiv : 1511.06422 [cs.LG] (cf. p. 131).

NAIR, Vinod et Geoffrey E HINTON (2010). « Rectified linear units improve restricted boltzmann machines ». In : *Proceedings of the 27th international conference on machine learning (ICML-10)*, p. 807-814 (cf. p. 132).

NESTEROV, Yurii E (1983). « A method for solving the convex programming problem with convergence rate O (1/kˆ 2) ». In : *Dokl. Akad. Nauk SSSR*. T. 269, p. 543-547 (cf. p. 126).

PHAM, Hieu, Melody Y. GUAN, Barret ZOPH, Quoc V. LE et Jeff DEAN (fév. 2018). « Efficient Neural Architecture Search via Parameter Sharing ». In : *arXiv e-prints*, arXiv:1802.03268, arXiv:1802.03268. arXiv : 1802.03268 [cs.LG] (cf. p. 144).

REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK et Ali FARHADI (juin 2016). « You Only Look Once: Unified, Real-Time Object Detection ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 46, 145, 146).

REN, Shaoqing, Kaiming HE, Ross GIRSHICK et Jian SUN (2015). « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks ». In : *Advances in Neural Information Processing Systems 28*. Sous la dir. de C. CORTES, N. D. LAWRENCE, D. D. LEE, M. SUGIYAMA et R. GARNETT. Curran Associates, Inc., p. 91-99. URL : http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf (cf. p. 146).

RONNEBERGER, Olaf, Philipp FISCHER et Thomas BROX (2015). « U-net: Convolutional networks for biomedical image segmentation ». In : *International Conference on Medical image computing and computer-assisted intervention*. Springer, p. 234-241 (cf. p. 142, 143).

RUDER, Sebastian (2016). « An overview of gradient descent optimization algorithms ». In : *arXiv preprint arXiv:1609.04747* (cf. p. 126).

SALIMANS, Tim et Diederik P KINGMA (2016). « Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks ». In : *Advances in Neural Information Processing Systems 29*. Sous la dir. de D. D. LEE, M. SUGIYAMA, U. V. LUXBURG, I. GUYON et R. GARNETT. Curran Associates, Inc., p. 901-909. URL : http://papers.nips.cc/paper/6114-weight-normalization-a-simple-reparameterization-to-accelerate-training-of-deep-neural-networks.pdf (cf. p. 135).

SAXE, Andrew M., James L. MCCLELLAND et Surya GANGULI (déc. 2013). « Exact solutions to the nonlinear dynamics of learning in deep linear neural networks ». In : *ArXiv e-prints*, arXiv:1312.6120, arXiv:1312.6120. arXiv : 1312.6120 [cs.NE] (cf. p. 131).

SAXENA, Shreyas et Jakob VERBEEK (2016). « Convolutional neural fabrics ». In : *Advances in Neural Information Processing Systems*, p. 4053-4061 (cf. p. 59, 144, 145).

SIMARD, Patrice Y, Dave STEINKRAUS et John C PLATT (2003). « Best practices for convolutional neural networks applied to visual document analysis ». In : *null*. IEEE, p. 958 (cf. p. 124).

SIMONYAN, Karen et Andrew ZISSERMAN (sept. 2014). « Very Deep Convolutional Networks for Large-Scale Image Recognition ». In : *ArXiv e-prints*, arXiv:1409.1556, arXiv:1409.1556. arXiv : 1409.1556 [cs.CV] (cf. p. 90, 120).

SINGH, Saurabh, Derek HOIEM et David FORSYTH (2016). « Swapout: Learning an ensemble of deep architectures ». In : *Advances in Neural Information Processing Systems 29*. Sous la dir. de D. D. LEE, M. SUGIYAMA, U. V. LUXBURG, I. GUYON et R. GARNETT. Curran Associates, Inc., p. 28-36. URL : http://papers.nips.cc/paper/6205-swapout-learning-an-ensemble-of-deep-architectures.pdf (cf. p. 136).

SMITH, Leslie N (2017). « Cyclical learning rates for training neural networks ». In : *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, p. 464-472 (cf. p. 128).

SRIVASTAVA, Nitish, Geoffrey HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER et Ruslan SALAKHUTDINOV (2014). « Dropout: a simple way to prevent neural networks from overfitting ». In : *The Journal of Machine Learning Research* 15.1, p. 1929-1958. URL : http://jmlr.org/papers/v15/srivastava14a.html (cf. p. 135).

SRIVASTAVA, Rupesh Kumar, Klaus GREFF et Jürgen SCHMIDHUBER (2015). « Highway networks ». In : *arXiv preprint arXiv:1505.00387* (cf. p. 139).

SUTSKEVER, Ilya, James MARTENS, George DAHL et Geoffrey HINTON (2013). « On the importance of initialization and momentum in deep learning ». In : *International conference on machine learning*, p. 1139-1147 (cf. p. 126).

SZEGEDY, Christian, Sergey IOFFE, Vincent VANHOUCKE et Alexander A ALEMI (2017). « Inception-v4, inception-resnet and the impact of residual connections on learning. » In : *AAAI*. T. 4, p. 12 (cf. p. 48, 140).

SZEGEDY, Christian, Wei LIU, Yangqing JIA, Pierre SERMANET, Scott REED, Dragomir ANGUELOV, Dumitru ERHAN, Vincent VANHOUCKE et Andrew RABINOVICH (2015). « Going deeper with convolutions ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1-9 (cf. p. 120, 138).

TARG, Sasha, Diogo ALMEIDA et Kevin LYMAN (2016). « Resnet in Resnet: generalizing residual architectures ». In : *arXiv preprint arXiv:1603.08029* (cf. p. 141).

TIELEMAN, Tijmen et Geoffrey HINTON (2012). « Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning ». In : *University of Toronto, Technical Report* (cf. p. 127).

TOMPSON, Jonathan, Ross GOROSHIN, Arjun JAIN, Yann LECUN et Christoph BREGLER (juin 2015). « Efficient Object Localization Using Convolutional Networks ». In : *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cf. p. 136).

TROTTIER, Ludovic, Philippe GIGU, Brahim CHAIB-DRAA et collab. (2017). « Parametric exponential linear unit for deep convolutional neural networks ». In : *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE, p. 207-214 (cf. p. 133).

UIJLINGS, Jasper RR, Koen EA VAN DE SANDE, Theo GEVERS et Arnold WM SMEULDERS (2013). « Selective search for object recognition ». In : *International journal of computer vision* 104.2, p. 154-171 (cf. p. 145, 146).

ULYANOV, D., A. VEDALDI et V. LEMPITSKY (juil. 2016). « Instance Normalization: The Missing Ingredient for Fast Stylization ». In : *ArXiv e-prints*. arXiv : 1607.08022 [cs.CV] (cf. p. 134).

VEIT, Andreas, Michael J WILBER et Serge BELONGIE (2016). « Residual networks behave like ensembles of relatively shallow networks ». In : *Advances in Neural Information Processing Systems*, p. 550-558 (cf. p. 140).

VENIAT, Tom et Ludovic DENOYER (2017). « Learning Time/Memory-Efficient Deep Architectures with Budgeted Super Networks ». In : *arXiv preprint arXiv:1706.00046* (cf. p. 144, 145).

WAN, Li, Matthew ZEILER, Sixin ZHANG, Yann Le CUN et Rob FERGUS (17–19 Jun 2013). « Regularization of Neural Networks using DropConnect ». In : *Proceedings of the 30th International*

Conference on Machine Learning. Sous la dir. de Sanjoy DASGUPTA et David MCALLESTER. T. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA : PMLR, p. 1058-1066. URL : http://proceedings.mlr.press/v28/wan13.html (cf. p. 136).

WU, Yuxin et Kaiming HE (2018). « Group normalization ». In : *arXiv preprint arXiv:1803.08494* (cf. p. 134).

XIE, Saining, Ross GIRSHICK, Piotr DOLLÁR, Zhuowen TU et Kaiming HE (2017). « Aggregated residual transformations for deep neural networks ». In : *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, p. 5987-5995 (cf. p. 90, 138).

XU, Bing, Naiyan WANG, Tianqi CHEN et Mu LI (2015). « Empirical evaluation of rectified activations in convolutional network ». In : *arXiv preprint arXiv:1505.00853* (cf. p. 133).

ZAGORUYKO, Sergey et Nikos KOMODAKIS (2016). « Wide residual networks ». In : *arXiv preprint arXiv:1605.07146* (cf. p. 121, 129).

ZEILER, Matthew D. (déc. 2012). « ADADELTA: An Adaptive Learning Rate Method ». In : *arXiv e-prints*, arXiv:1212.5701, arXiv:1212.5701. arXiv : 1212.5701 [cs.LG] (cf. p. 126).

ZEILER, Matthew D et Rob FERGUS (2014). « Visualizing and understanding convolutional networks ». In : *European conference on computer vision*. Springer, p. 818-833 (cf. p. 120).

ZHANG, Ke, Miao SUN, Tony X HAN, Xingfang YUAN, Liru GUO et Tao LIU (2018). « Residual networks of residual networks: Multilevel residual networks ». In : *IEEE Transactions on Circuits and Systems for Video Technology* 28.6, p. 1303-1314 (cf. p. 140, 141).

ZHAO, Hengshuang, Xiaojuan QI, Xiaoyong SHEN, Jianping SHI et Jiaya JIA (2017). « Icnet for real-time semantic segmentation on high-resolution images ». In : *arXiv preprint arXiv:1704.08545* (cf. p. 57, 143, 144).

ZOPH, Barret, Vijay VASUDEVAN, Jonathon SHLENS et Quoc V LE (2017). « Learning Transferable Architectures for Scalable Image Recognition ». In : *arXiv preprint arXiv:1707.07012* (cf. p. 144).

RÉSUMÉ

Cette thèse se trouve à la confluence de deux mondes en pleine explosion : la voiture autonome et l'intelligence artificielle (particulièrement l'apprentissage profond). Le premier tirant profit du deuxième, les véhicules autonomes utilisent de plus en plus de méthodes d'apprentissage profond pour analyser les données produites par ses différents capteurs (dont les LiDARs) et pour prendre des décisions.
Alors que les méthodes d'apprentissage profond ont révolutionné l'analyse des images (en classification et segmentation par exemple), elles ne produisent pas des résultats aussi spectaculaires sur les nuages de points 3D. En particulier parce que les jeux de scènes données de nuages de points 3D annotés sont rares et de qualité moyenne. On présente donc dans cette thèse un nouveau jeu de données réalisé par acquisition mobile pour produire suffisamment de données et annoté à la main pour assurer une bonne qualité de segmentation.
De plus ces jeux de données sont par nature déséquilibrés en nombre d'échantillons par classe et contiennent beaucoup d'échantillons redondants, on propose donc une méthode d'échantillonnage adaptée à ces jeux de données.
Un autre problème rencontré quand on essaye de classifier un point à partir de son voisinage sous forme de grille voxelique est le compromis entre un pas de discrétisation fin (pour décrire précisément la surface voisine du point) et une grille de taille élevée (pour aller chercher du contexte un peu plus loin). On propose donc également des méthodes de réseaux tirant profit de voisinages multi-échelles. Ces méthodes atteignent l'état de l'art des méthodes de classification par point sur des benchmarks publics.
Enfin pour respecter les contraintes imposées par les systèmes embarqués (traitement en temps réel et peu de puissance de calcul), on présente une méthode qui permet de n'appliquer les couches convolutionnelles que là où il y a de l'information à traiter.

MOTS CLÉS

Nuage de points, Classification, Segmentation, Apprentissage Profond, Calcul Temps-Réel Embarqué

ABSTRACT

This thesis is at the confluence of two worlds in rapid growth: autonomous cars and artificial intelligence (especially deep learning). As the first takes advantage of the second, autonomous vehicles are increasingly using deep learning methods to analyze the data produced by its various sensors (including LiDARs) and to make decisions.
While deep learning methods have revolutionized image analysis (in classification and segmentation, for example), they do not produce such spectacular results on 3D point clouds. This is particularly true because the datasets of annotated 3D point clouds are rare and of moderate quality. This thesis therefore presents a new dataset developed by mobile acquisition to produce enough data and annotated by hand to ensure a good quality of segmentation. In addition, these datasets are inherently unbalanced in number of samples per class and contain many redundant samples, so a sampling method adapted to these datasets is proposed.
Another problem encountered when trying to classify a point from its neighbourhood as a voxel grid is the compromise between a fine discretization step (for accurately describing the surface adjacent to the point) and a large grid (to look for context a little further away). We therefore also propose network methods that take advantage of multi-scale neighbourhoods. These methods achieve the state of the art of point classification methods on public benchmarks.
Finally, to respect the constraints imposed by embedded systems (real-time processing and low computing power), we present a method that allows convolutional layers to be applied only where there is information to be processed.

KEYWORDS

Point clouds, Classification, Segmentation, Deep-Learning, Embedded Real-Time Processing