

3D object detectie voor autonome wagens

Michiel Janssen

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotoren:
Prof. dr. ir. Tinne Tuytelaars
Prof. dr. ir. Hendrik Blockeel

Assessoren:
Prof. dr. ir. Marc Denecker
Dr. Bregt Verreet

Begeleider:
Tom Roussel

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotoren als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail info@cs.kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotoren is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Deze thesis weerspiegelt eerder een continue samenwerking tussen verschillende partijen dan een individuele prestatie. Ten eerste wil ik mijn promotoren, Tinne Tuytelaars en Hendrik Blockeel, van harte bedanken. Zij hebben mij de kans en de vrijheid gegeven om mijn eigen gekozen thesis onderwerp te onderzoeken, in samenwerking met Xenomatix. Van jong af aan ben ik altijd gefascineerd geweest door auto's en computers. De keuze van mijn onderwerp is tot stand gekomen door mijn enorme interesse in Tesla en hun Autopilot software. Ik ben dan ook enorm fier dat ik mijn twee passies heb kunnen combineren in deze thesis. Vervolgens wil ik mijn assistent, Tom Roussel, bedanken om mij doorheen dit proces van ups en downs te begeleiden en te helpen. Onze wekelijkse meetings en discussies hebben enorm veel bijgedragen aan het eindresultaat. Een speciale bedanking gaat uit naar Geert Caenen, van Xenomatix, die altijd de tijd nam om aanwezig te zijn bij onze maandelijkse meetings.

Deze thesis markeert het einde van een vijf jaar durend avontuur aan de KU Leuven, waarin ik het voorrecht had om een semester door te brengen aan de Politecnico di Milano. Ik wil mijn ouders enorm hard bedanken voor alle kansen die zij mij hebben gegeven en voor de steun tijdens mijn traject als student ingenieurswetenschappen.

Michiel Janssen

Inhoudsopgave

| | |
|---|------|
| Voorwoord | i |
| Samenvatting | iv |
| Lijst van figuren | v |
| Lijst van tabellen | viii |
| Lijst van Afkortingen en Symbolen | x |
| 1 Inleiding | 1 |
| 1.1 Onderzoeksvragen | 3 |
| 1.2 Thesis structuur | 3 |
| 2 Literatuurstudie | 5 |
| 2.1 Neurale netwerken | 5 |
| 2.1.1 Trainen | 8 |
| 2.1.2 Convolutionele Neurale Netwerken | 9 |
| 2.2 2D Object Detectie | 11 |
| 2.2.1 <i>Single-stage</i> detectors | 11 |
| 2.2.2 <i>Two-stage</i> detectors | 14 |
| 2.3 3D Object Detectie | 16 |
| 2.3.1 Camera gebaseerde methodes | 16 |
| 2.3.2 Lidar gebaseerde methodes | 20 |
| 2.3.3 Camera en lidar gebaseerde methodes | 23 |
| 3 Datasets | 25 |
| 3.1 KITTI dataset | 25 |
| 3.1.1 Dataset beschrijving | 26 |
| 4 Methodologie | 29 |
| 4.1 2D object detector | 29 |
| 4.2 3D object detector - Camera gebaseerd | 30 |
| 4.2.1 Model architectuur | 30 |
| 4.2.2 Implementatie details | 30 |
| 4.2.3 Toevoegingen | 33 |
| 4.2.4 Finale Architecturen | 34 |
| 4.3 3D object detector - Lidar gebaseerd | 35 |
| 4.3.1 Model architectuur | 35 |
| 4.3.2 Implementatie details | 39 |

| | | |
|---------------------|--|-----------|
| 4.3.3 | Finale Architectuur | 39 |
| 4.4 | 3D object detector - Camera en Lidar gebaseerd | 40 |
| 4.4.1 | Model architectuur | 40 |
| 4.4.2 | Implementatie details | 40 |
| 4.4.3 | Finale Architectuur | 40 |
| 5 | Experimenten en resultaten | 41 |
| 5.1 | Statistieken | 41 |
| 5.2 | Kwantitatieve experimenten | 43 |
| 5.2.1 | 2D detector | 43 |
| 5.2.2 | 3D object detector - Camera gebaseerd | 43 |
| 5.2.3 | 3D object detector - Lidar gebaseerd | 48 |
| 5.2.4 | 3D object detector - Camera en Lidar gebaseerd | 50 |
| 5.3 | Benchmarking | 50 |
| 5.4 | Kwalitatieve experimenten | 53 |
| 5.4.1 | Foutenanalyse | 53 |
| 5.4.2 | Bijkomende resultaten | 57 |
| 6 | Conclusie | 59 |
| 6.1 | Toekomstig werk | 60 |
| 6.1.1 | Camera-lidar fusie | 60 |
| 6.1.2 | Graph-CNN | 60 |
| 6.1.3 | Pseudo-lidar | 61 |
| 6.1.4 | Meer klassen detecteren binnen KITTI | 62 |
| 6.1.5 | Domain Adaptation | 62 |
| A | Hyperparameters | 65 |
| A.1 | GeoLoc | 65 |
| A.2 | LocNN | 65 |
| A.3 | GeoLocNN | 66 |
| A.4 | FrustumPointNet | 67 |
| A.5 | FusionNet | 68 |
| B | Kwalitatieve resultaten | 69 |
| B.1 | ResidualGeoLoc | 69 |
| B.2 | FrustumPointNet | 72 |
| Bibliografie | | 75 |

Samenvatting

Het onderzoek naar zelfrijdende wagens staat meer dan ooit centraal in onze maatschappij. Het perceptieprobleem, dat onder andere bestaat uit het detecteren van 2D en 3D objecten, is een cruciaal onderdeel van een autonome wagen. Hoewel er uitstekende 2D detectors bestaan die heel nauwkeurig de 2D positie van objecten kunnen bepalen, is het voor een autonome wagen ook noodzakelijk om de 3D locatie, positie en oriëntatie van objecten te kennen. Immers is het sturen en controleren van een autonome wagen sterk gelinkt aan de perceptie van de 3D-ruimte. Deze thesis onderzoekt het 3D object detectie probleem. Eerst wordt er onderzocht hoe men op basis van afbeeldingen een 3D object detectie model kan ontwikkelen. Dit is niet vanzelfsprekend aangezien afbeeldingen enkel 2D informatie bevatten. Dan wordt er onderzocht hoe men met een lidar sensor, die puntenwolken genereert, een 3D object detectie model kan ontwikkelen. Het krachtige aan een lidar is dat het zeer accuraat de diepte van objecten kan inschatten. Tot slot wordt er onderzocht hoe men een camera gebaseerde- en lidar gebaseerde methode combineert tot een nieuwe architectuur. De bovenstaande modellen worden geëvalueerd met de bekende KITTI dataset en behaalden competitieve resultaten.

Lijst van figuren

| | | |
|------|--|----|
| 1.1 | De architectuur van een autonoom voertuigsysteem. Bron: [34] | 2 |
| 1.2 | 2D vs 3D object detectie | 2 |
| 2.1 | De vergelijking tussen een biologisch neuron en een artificieel neuron | 5 |
| 2.2 | Een feed forward multilayer perceptron | 7 |
| 2.3 | De werking van een convolutioneel neuraal netwerk. Bron: [46] | 10 |
| 2.4 | <i>Convolutionele</i> operatie op een input matrix | 10 |
| 2.5 | Een <i>max pooling</i> operatie | 11 |
| 2.6 | De werking van een YOLO netwerk. Bron: [41] | 12 |
| 2.7 | VGG-16 architectuur. Bron: [31] | 13 |
| 2.8 | Het concept van <i>residual learning</i> . Bron: [17] | 13 |
| 2.9 | De werking van het R-CNN netwerk. Bron: [14] | 14 |
| 2.10 | De architectuur van het Faster R-CNN netwerk. Bron: [42] | 15 |
| 2.11 | Semantische segmentatie (midden) vs instance segmentatie (rechts). Bron: [8] | 17 |
| 2.12 | De <i>Multi-Bin</i> architectuur. Bron: [30] | 17 |
| 2.13 | De globale oriëntatie van de auto lijkt schijnbaar te veranderen in de bijgesneden afbeeldingen. Bron: [30] | 18 |
| 2.14 | Illustratie van alle oriëntaties in vogelperspectief. Bron: [27] | 18 |
| 2.15 | Het puntenwolk probleem | 21 |
| 2.16 | De PointNet architectuur. Bron: [37] | 21 |
| 2.17 | De VoxelNet architectuur. Bron: [59] | 22 |
| 2.18 | De mogelijke fusies van camera en lidar informatie. Bron: [7] | 24 |
| 2.19 | De MV3D architectuur. Bron: [7] | 24 |
| 3.1 | De karakteristieken van de auto en de sensoren waarmee de KITTI dataset is verzameld. Bron: [12] | 25 |
| 4.1 | De algemene architectuur voor het 2D-3D model | 29 |
| 4.2 | De baseline RGB-architectuur | 30 |
| 4.3 | De locatie bepaling via een ANN | 33 |
| 4.4 | Finetunen van geometrische locatie via een ANN | 34 |
| 4.5 | Schematisch overzicht van de Lidar-architectuur | 35 |
| 4.6 | Het aanmaken van een afgeknotte piramide regio. Bron: [36] | 35 |
| 4.7 | De assenstelsels waarin een puntenwolk kan gedefinieerd worden | 36 |

LIJST VAN FIGUREN

| | |
|--|----|
| 4.8 Schematisch overzicht van het InstanceSeg-Net | 37 |
| 4.9 Schematisch overzicht van het T-Net | 38 |
| 4.10 Schematisch overzicht van het 3DBox-Net | 38 |
| 4.11 Schematisch overzicht van het aangepaste 3DBox-Net | 40 |
| 5.1 GeoLoc: de 3D dimensie error in functie van de afstand tot het object | 44 |
| 5.2 GeoLoc: de 3D locatie error in functie van de afstand tot het object | 44 |
| 5.3 GeoLoc: de oriëntatie error in functie van de afstand tot het object | 44 |
| 5.4 LocNN: de 3D locatie error in functie van de afstand tot het object | 46 |
| 5.5 GeoLocNN: de 3D locatie error in functie van de afstand tot het object | 47 |
| 5.6 FrustumPointNet: de 3D dimensie error in functie van de afstand tot het object | 49 |
| 5.7 FrustumPointNet: de 3D locatie error in functie van de afstand tot het object | 49 |
| 5.8 FrustumPointNet: de oriëntatie error in functie van de afstand tot het object | 49 |
| 5.9 FusionNet: de 3D dimensie error in functie van de afstand tot het object | 51 |
| 5.10 FusionNet: de 3D locatie error in functie van de afstand tot het object | 51 |
| 5.11 FusionNet: de oriëntatie error in functie van de afstand tot het object | 51 |
| 5.12 Visualisatie van voorbeeld 6 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 4 weinig punten bevat en dat noch het FrustumPointNet noch het ResidualGeoLocNN dit object detecteert | 53 |
| 5.13 Visualisatie van voorbeeld 63 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 1 enorm hard wordt geoccludeerd door een muur wat leidt tot een slechte detectie | 54 |
| 5.14 Visualisatie van voorbeeld 8 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 1 deels onzichtbaar is op de afbeelding, wat leidt tot een slechte detectie voor het ResidualGeoLocNN | 55 |
| 5.15 Visualisatie van voorbeeld 373 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 2 wordt gemist door het ResidualGeoLocNN | 56 |
| 5.16 Visualisatie van voorbeeld 2107 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat alle auto's door beide methoden gedetecteerd worden. | 56 |
| 5.17 Visualisatie van voorbeeld 6913 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 3 een foute annotatie is in de <i>ground truth</i> | 57 |
| 5.18 Visualisatie van voorbeeld 3292 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 1 op een helling staat en deze goed gedetecteerd wordt. De puntenwolk wordt als rechterzijaanzicht voorgesteld | 58 |

| | |
|---|----|
| 5.19 Visualisatie van voorbeeld 249 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 6 verbazingwekkend genoeg wordt gedetecteerd | 58 |
| 6.1 Boven: lidar input. Onder: pseudo-lidar input. Op basis van de diepte wordt elke pixel ingekleurd. Blauw is voor nabijgelegen pixels, rood voor verre pixels | 61 |
| B.1 Enkele visualisaties uit de validatie set voor het ResidualGeoLocNN . . | 69 |
| B.2 Enkele visualisaties uit de validatie set voor het ResidualGeoLocNN . . | 70 |
| B.3 Enkele visualisaties uit de validatie set voor het ResidualGeoLocNN . . | 71 |
| B.4 Enkele visualisaties uit de validatie set voor het FrustumPointNet . . . | 72 |
| B.5 Enkele visualisaties uit de validatie set voor het FrustumPointNet . . . | 73 |
| B.6 Enkele visualisaties uit de validatie set voor het FrustumPointNet . . . | 74 |

Lijst van tabellen

| | | |
|-----|--|----|
| 2.1 | Enkele bekende activatie functies | 6 |
| 3.1 | Structuur van de annotaties in de KITTI dataset | 26 |
| 3.2 | Definitie van een Easy, Moderate en Hard KITTI object | 26 |
| 3.3 | Klasse onevenwicht voor de KITTI dataset | 27 |
| 5.1 | De 2D AP voor auto's op de KITTI validatie set | 43 |
| 5.2 | Evaluatie op de validatie set voor de KITTI 3D en BEV objectdetectiebenchmark (auto's). De vetgedrukte blauwe getallen tonen de beste eigengemaakte modellen, de vetgedrukte zwarte cijfers tonen de SOTA architecturen | 52 |
| 6.1 | Eerste testresultaten van het FrustumGraphCNN op de KITTI validatie set (auto's). De vetgedrukte blauwe cijfers zijn een verbetering ten opzichte van het FrustumPointNet | 61 |
| 6.2 | Eerste testresultaten van het FrustumPointNet-pseudoLidar op de KITTI validatie set (auto's) | 62 |
| A.1 | De hyperparameters voor het GeoLoc netwerk | 65 |
| A.2 | De hyperparameters voor het LocNN netwerk | 65 |
| A.3 | De gemiddelde 3D locatie error voor de verschillende configuraties van LocNN, geëvalueerd op de auto's uit de KITTI validatie set | 66 |
| A.4 | De hyperparameters voor het GeoLocNN netwerk | 66 |
| A.5 | De gemiddelde 3D locatie error voor de verschillende configuraties van GeoLocNN, geëvalueerd op de auto's uit de KITTI validatie set | 67 |
| A.6 | De hyperparameters voor het FrustumPointNet netwerk | 67 |
| A.7 | De hyperparameters voor het FusionNet netwerk | 68 |

Lijst van Afkortingen en Symbolen

Afkortingen

| | |
|-------|--|
| 2DBB | 2D Bounding Box |
| 3DBB | 3D Bounding Box |
| 2DOD | 2D Object Detectie |
| 3DOD | 3D Object Detectie |
| ANN | Artificieel Neuraal Netwerk |
| AP | Average Precision |
| BEV | Bird's-Eye-View |
| CNN | Convolutioneel Neuraal Netwerk |
| FC | Fully connected laag |
| FOV | Field Of View |
| FPN | Feature Pyramid Network |
| GCNN | Graph CNN |
| GPS | Global Positioning System |
| GPU | Graphics Processing Unit |
| IDE | Instance Depth Estimation |
| IMU | Inertial Measurement Unit |
| IoU | Intersectie over Unie |
| MAE | Mean Absolute Error |
| MLP | Multilayer Perceptron |
| MSE | Mean Squared Error |
| NMS | Non-Maximum Suppression |
| OFT | Orthographic Feature Transform |
| ReLU | Rectified Linear Unit |
| R-CNN | Regio-gebaseerd Convolutioneel Netwerk |
| RGB | Rood, Groen, Blauw |
| RPN | Region Proposal Netwerk |
| SOTA | State Of The Art |
| SSD | Single Shot Multibox Detector |
| SVM | Support Vector Machine |
| YOLO | You Only Look Once |

Hoofdstuk 1

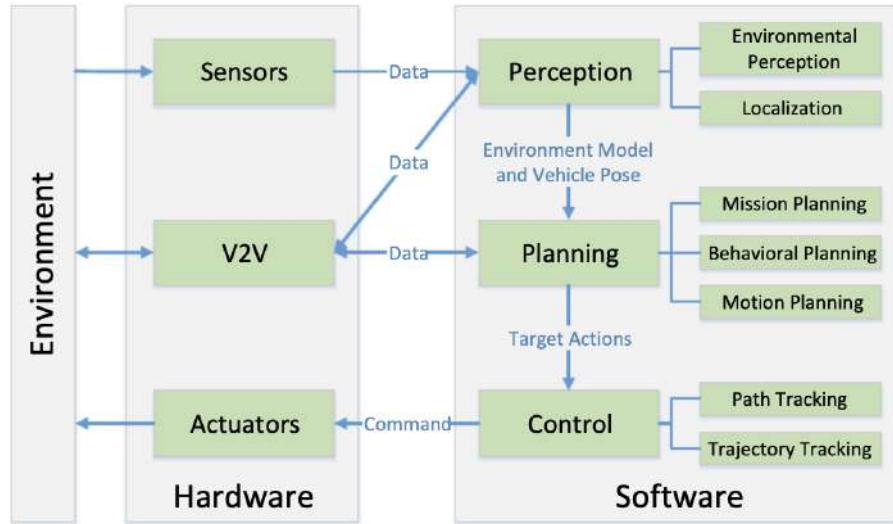
Inleiding

De zelfrijdende auto is wereldwijd in opkomst. Bedrijven zoals Tesla, Uber, Waymo, BMW, Audi en Mercedes investeren enorm veel geld om de race naar autonoom rijden te winnen. Een autonoom voertuigsoftwaresysteem bestaat in het algemeen uit drie modules: perceptie, planning en controle, en de interacties tussen deze modules en de buitenwereld. Perceptie verwijst naar het vermogen om een omgeving waar te nemen en te modelleren. De planning-module gebruikt het perceptie-model om te beslissen welke toekomstige acties er moeten genomen worden en de controle-module zorgt er voor dat deze acties worden uitgevoerd door actuators aan te sturen om te remmen, te sturen, te accelereren, etc. Een weergave van zo een systeem is te zien in Figuur 1.1. Perceptie bestaat in het algemeen uit 2 subtaken. *Environmental Perception* verwijst naar het ontwikkelen van een contextueel begrip van de omgeving waarin een wagen zich bevindt, bv. waar objecten zich in de ruimte bevinden, detectie van verkeersborden, vrije weg detecteren, etc. *Localization* verwijst naar het vermogen van het systeem om de positie van een wagen ten opzichte van zijn omgeving te bepalen [34].

Deze thesis bestudeert 3D object detectie (3DOD), een onderdeel van het perceptieprobleem. 3DOD is een cruciale taak voor autonoom rijden. Veel belangrijke componenten binnen autonoom rijden zoals voorspelling, planning en controle van de beweging van de wagen vereisen doorgaans een perfecte weergave van de 3D ruimte rond het voertuig. In vergelijking met 2D object detectie (2DOD), waarbij een model is getraind om 2D bounding boxes (2DBB's) te detecteren en te tekenen rond objecten in een afbeelding, vereist 3DOD ook een schatting van de grootte, positie en richting van een object in de 3D wereld. Het verschil tussen 2DOD en 3DOD is te zien in Figuur 1.2.

Tegenwoordig maken bedrijven gebruik van drie type sensoren om 3DOD te doen: camera, lidar en radar. Deze thesis behandelt het gebruik van een camera en/of lidar. Het gebruik van een lidar (light detection and ranging) sensor voor 3DOD is de laatste tijd heel populair geworden. Lidar maakt het mogelijk om de afstand tot een object of oppervlak te bepalen door gebruik te maken van laserpulsen [54].

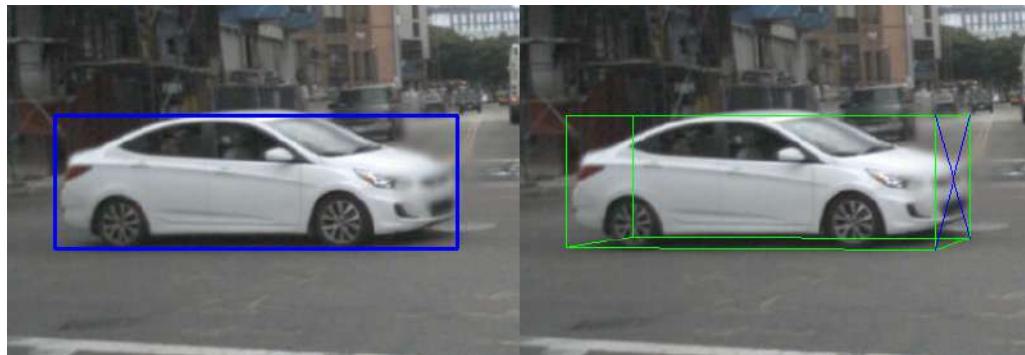
1. INLEIDING



Figuur 1.1: De architectuur van een autonoom voertuigssystemen. Bron: [34].

Het principe is vrij eenvoudig: een laserstraal wordt uitgezonden en zal hopelijk na enige tijd door reflectie terug opgevangen worden. Uit het tijdsverschil tussen zenden en ontvangen kan de afstand worden berekend. In de praktijk worden meerdere laserpulsen uitgezonden op een draaiende spiegel waardoor men een 360° ruimte gevoel krijgt. Het resultaat is een puntenwolk die de 3D wereld zo goed mogelijk probeert weer te geven.

Deze thesis is een samenwerking met Xenomatix, gevestigd in Haasrode. Zij hebben, in tegenstelling tot vele concurrenten, een solid-state (d.w.z. zonder bewegende spiegels) lidar ontwikkeld. Xenomatix heeft zich de laatste jaren voornamelijk gefocust op het ontwikkelen van hun lidar hardware. Echter willen zij nu ook inzetten op software om hun klanten te laten zien wat de mogelijkheden zijn van hun sensor, bv. geavanceerde rijkrijpsystemen en autonome rijtoepassingen.



Figuur 1.2: 2D vs 3D object detectie.

1.1 Onderzoeks vragen

Zoals hierboven vermeld, is het doel van deze thesis het onderzoek naar 3DOD voor autonome wagens. Men gaat in deze thesis onderzoeken welke input leidt tot het beste 3DOD model. Er kunnen drie verschillende modellen getraind worden: één met afbeeldingen als input, één met puntenwolken als input en één dat beide inputs combineert. Voor dit onderzoek gebruikt men de afbeeldingen en puntenwolken van de KITTI dataset [12].

In deze thesis gaat men de volgende onderzoeks vragen beantwoorden:

1. *Als de input van een netwerk RGB afbeeldingen zijn, hoe kan men dit netwerk trainen voor 3DOD en wat is de nauwkeurigheid van dit model?*
2. *Als de input van een netwerk puntenwolken zijn, hoe kan men dit netwerk trainen voor 3DOD en wat is de nauwkeurigheid van dit model?*
3. *Als men zowel RGB afbeeldingen als puntenwolken meegeeft aan een netwerk, verhoogt dit dan de nauwkeurigheid?*

Uit deze drie onderzoeks vragen moet dus blijken welk model het beste is om 3DOD te doen. Daarnaast is het nuttig om te onderzoeken of modellen getraind op de KITTI dataset generaliseerbaar zijn naar andere gelijkaardige datasets, zoals de recente nuScenes dataset [4]. Deze hypothese is niet vanzelfsprekend aangezien er een verschil is tussen de manier waarop deze datasets hun afbeeldingen en puntenwolken verzamelen. Het verschil zit hem vaak in de opstelling waarmee de data wordt verzameld, bv. verschillende cameraconfiguraties, *field of view* (FOV) van de camera, grootte van de afbeelding waarmee getraind is, etc.

Ondanks het feit dat generaliseerbaarheid een groot vraagteken is, zou het gebruik van een getraind model voor andere datasets toch een voordeel kunnen hebben. Het labelen van *ground truth* 3D bounding boxes (3DBB's), een zeer tijdrovend en duur proces, is een strikte vereiste om een goed 3DOD model te maken. Xenomatix die niet beschikt over geannoteerde data kan het beste getrainde model op KITTI inzetten om 3DBB voorstellen te genereren. Mits aanpassingen, bekomt men zo sneller en goedkoper de *ground truth* annotaties.

1.2 Thesis structuur

Deze thesis is als volgt opgebouwd. In hoofdstuk 2 bespreekt men de theorie en literatuur die gerelateerd zijn aan het onderwerp van deze thesis. Hierna zal er in hoofdstuk 3 de gebruikte dataset besproken worden. Vervolgens zullen in hoofdstuk 4 de gebruikte methodes aan bod komen. In hoofdstuk 5 zal men kijken naar de experimenten en resultaten. Tot slot eindigt deze thesis met een conclusie in hoofdstuk 6.

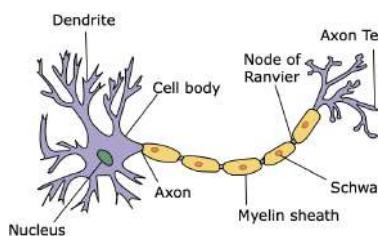
Hoofdstuk 2

Literatuurstudie

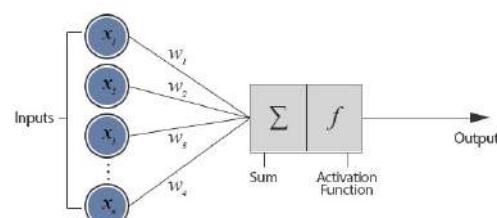
Zoals beschreven in hoofdstuk 1 is 3DOD een cruciale taak voor autonome wagens. Echter is het schatten van zo een 3DBB (grootte, locatie, richting) een complex computervisie probleem. Dit hoofdstuk wordt in drie delen verdeeld. In sectie 2.1 wordt er uitgelegd wat neurale netwerken zijn en hoe deze werken, met de focus op een convolutioneel neuraal netwerk (CNN). Het 3DOD probleem is makkelijker te begrijpen wanneer men begrijpt hoe 2DOD werkt. Vandaar dat sectie 2.2 begint met een analyse over 2DOD. Daarna maakt men in sectie 2.3 de stap naar 3DOD en gaat men bekijken wat er al bestaat in de literatuur rond het 3DOD probleem.

2.1 Neurale netwerken

Een artificieel neuraal netwerk (ANN) is een concept dat in de jaren 40 en 50 al bestudeerd werd [29] [18]. Het is pas sinds de jaren 2000, met de opkomst van deep learning en de *Graphics Processing Unit* (GPU), dat ANN's populariteit vergaarden [33]. Een ANN is gebaseerd op de werking van ons menselijk brein. Net zoals het menselijk brein kunnen ANN's zowel begrippen aanleren als redeneren over bepaalde inputs en hiervoor een output genereren. Dat ons menselijk brein en ANN's nieuwe concepten kunnen aanleren is te danken aan de neuronen die het bevat. Figuur 2.1 toont het verschil tussen een biologisch en artificieel neuron.



(a) Een biologisch neuron. Bron: [53].



(b) Een artificieel neuron. Bron: [35].

Figuur 2.1: De vergelijking tussen een biologisch neuron en een artificieel neuron.

2. LITERATUURSTUDIE

Wanneer een biologisch neuron een voldoende grote zenuwimpuls ontvangt van de dendrieten wordt het neuron geactiveerd en loopt er een signaal van de nucleus naar de axon [20]. Bij een artificieel neuron worden de dendrieten voorgesteld door de x_i 's. Het grijze gedeelte in figuur 2.1(b) stelt de nucleus voor die bestaat uit een sommatie over de inputs gevuld door een activatiefunctie. Zodra deze som boven een bepaalde drempel ligt volgt er een activatie en wordt er een output gegenereerd [32]. Wiskundig gezien wordt een ANN als volgt beschreven:

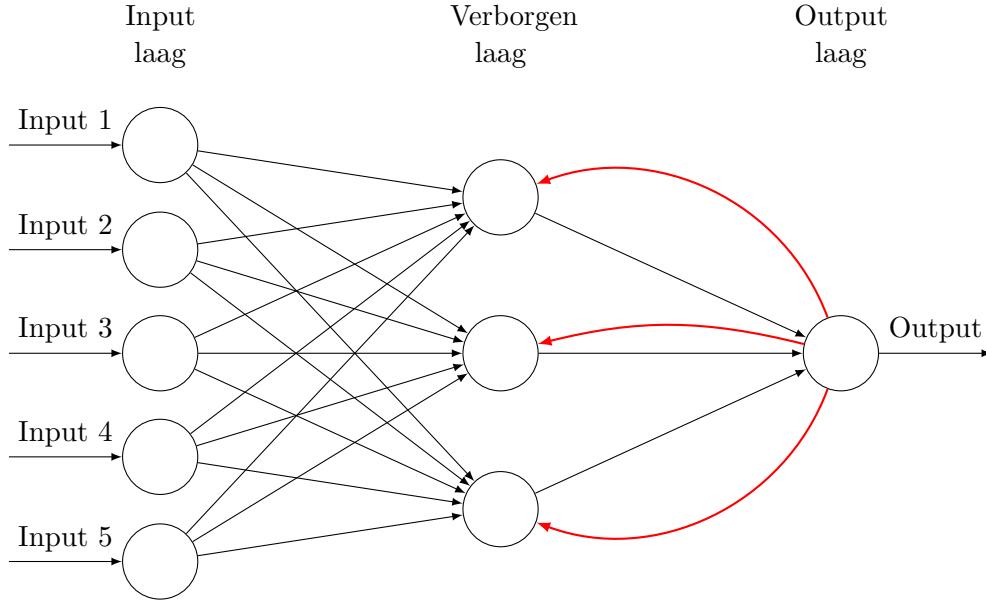
$$y = f(W^T x + b) = f\left(\sum_{i=1}^{n-1} w_i * x_i + b\right) \quad (2.1)$$

Waarbij y de output voorstelt, w_i het gewicht is voor de i -de input x_i , f de activatiefunctie is en b de bias term voorstelt. De bias term is een maat die aangeeft hoe gemakkelijk het is om een neuron te activeren. Voor een grote en kleine bias kan een neuron gemakkelijk respectievelijk moeilijk geactiveerd worden [32]. Deze bias term wordt voor het trainen geïnitialiseerd op een vooraf bepaalde constante. Tabel 2.1 geeft drie activatiefuncties weer die in de praktijk veel gebruikt worden. Omwille van zijn simpelheid wordt een *Rectified Linear Unit* (ReLU) tegenwoordig het meest gebruikt voor het trainen van diepe neurale netwerken.

Het trainen van één artificieel neuron gebeurt door iteratief de gewichten w_i en bias b aan te passen. Het *backpropagation* algoritme (zie sectie 2.1.1) is verantwoordelijk voor deze updates. Hoe en hoeveel de gewichten en bias moeten aangepast worden hangt sterk af van het probleem dat men wilt oplossen.

| Naam | Functie | Afgeleide | Figuur |
|---------|---|--|--------|
| Sigmoid | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)(1 - f(x))$ | |
| tanh | $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ | $f'(x) = 1 - f(x)^2$ | |
| ReLU | $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$ | |

Tabel 2.1: Enkele bekende activatie functies.



Figuur 2.2: Een feed forward multilayer perceptron.

Een ANN is altijd opgebouwd uit een bepaald aantal artificiële neuronen. De eenvoudigste vorm van een ANN is de *multilayer perceptron* (MLP), waarvan een voorbeeld is te zien in figuur 2.2. Elke cirkel op deze figuur stelt een artificieel neuron voor. Dit specifiek netwerk bestaat uit drie lagen: één input laag, één verborgen laag en één output laag. Daarnaast is dit netwerk speciaal omdat elk neuron verbonden is met alle neuronen uit een vorige laag. Zo'n laag noemt men ook wel een *fully connected* (FC) laag [49]. Netwerken waarbij de informatiestroom van links naar rechts verloopt worden *feed forward* netwerken genoemd, zoals de zwarte pijlen in bovenstaande figuur aantonen. Meer specifiek krijgt dit netwerk vijf inputs binnen en kunnen de neuron(en) in de input laag geactiveerd worden. Elk neuron dat geactiveerd wordt zendt op zijn beurt een signaal door naar de connecties met de volgende laag. De verborgen laag dankt zijn naam aan het feit dat deze laag van buitenaf niet zichtbaar is.

De kracht van ANN is dat het in staat is om complexe functies te benaderen. Echter worden ANN's vaak gezien als een *black-box* model met input-, verborgen- en output laag [3]. Het probleem met een *black-box* model is dat men een functie benadert zonder inzicht te krijgen in de vorm van de functie. Er bestaat geen eenvoudig verband tussen de gewichten en de benaderde functie. In sommige toepassingen is het echter noodzakelijk om te begrijpen hoe een netwerk tot een bepaalde output is gekomen. De tak binnen de artificiële intelligentie die deze problematiek bespreekt is Explainable AI maar is verder niet van toepassing binnen deze thesis.

2.1.1 Trainen

ANN's kunnen nieuwe concepten aanleren op voorwaarde dat er voldoende trainingsdata beschikbaar is. Neem bijvoorbeeld het probleem waarbij een ANN getraind wordt om een hond te herkennen in een afbeelding. Hiervoor is er een dataset nodig die bestaat uit (x, y) paren met x de input van een ANN en y de verwachte output van een ANN, ook wel *ground truth* genoemd. In dit voorbeeld bevat x en y de afbeeldingen respectievelijk de informatie of er al dan niet een hond aanwezig is. Vóór het trainen van een ANN start, wordt de gewichtenmatrix W en bias b geïnitialiseerd. Het trainingsproces kan nu beginnen. Dit proces bestaat uit twee delen: de *forward pass* en *backward pass* ook wel *backpropagation* genoemd.

In de *forward pass*, zie stap 3a tot en met 3c in Algoritme 1, wordt alle data meegegeven aan het ANN en zal er een output \hat{y} berekend worden. Vervolgens zal deze geschatte output \hat{y} vergeleken worden met de verwachte output y door gebruik te maken van een *loss* functie \mathcal{L} . De *loss* functie berekent hoe groot de fout is tussen \hat{y} en y . Tijdens het trainen zal een ANN proberen om de output van deze *loss* functie te minimaliseren. In de praktijk zijn er heel wat mogelijke *loss* functies die men kan gebruiken. De keuze van zo'n *loss* functie is enorm belangrijk aangezien deze functie de nauwkeurigheid van het netwerk kan beïnvloeden. Ter illustratie volgen hieronder twee populaire *loss* functies, namelijk de *mean absolute error* (MAE) en *mean squared error* (MSE).

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (2.2)$$

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (2.3)$$

met n het aantal outputs.

Nadat de *forward pass* de *loss* functie heeft berekend gaat de *backward pass*, zie stap 3d tot en met 3e in Algoritme 1, de gradiënten van deze *loss* functie berekenen zodanig dat een kleine aanpassing aan de gewichtenmatrix W leidt tot een kleinere *loss*. Deze gradiëntberekening wordt uitgevoerd door een proces dat men *automatische differentiatie* noemt. Dit proces gebruikt de gradiënten die berekend worden in elke laag en de kettingregel om elke gradiënt van het netwerk efficiënt te kunnen berekenen. Vervolgens worden de gradiënten nog aangepast door een *optimizer* om het vastlopen in lokale minima of numerieke instabiliteit te vermijden. Enkele bekende voorbeelden van *optimizers* zijn Momentum, Adagrad, RMSprop en Adam [44]. Ten slotte wordt de output van de *optimizer* geschaald met de learning rate η , en opgeteld bij W . Het bepalen van gradiënten en updaten van W verloopt in de richting van de rode pijlen in figuur 2.2. Algoritme 1 wordt herhaald voor een aantal *epochs* totdat er aan een stopcriterium is voldaan. Wanneer de trainingsdata eenmaal door het netwerk is gegaan, spreekt men van één *epoch*.

In de praktijk wordt het *backpropagation* algoritme per *batch* uitgevoerd, waarbij een *batch* bestaat uit n inputs. Deze n wordt vooraf bepaald door de gebruiker. Per *batch* worden de gradiënten berekend door een GPU, die geoptimaliseerd is om in parallel te rekenen. De gradiënt is dan een gemiddelde gradiënt van alle inputs aanwezig in deze *batch*. Het gebruik van *batches* maakt het mogelijk om sneller te trainen en leidt in het algemeen tot een betere generalisatie [19]. Generalisatie staat centraal bij veel machine learning problemen. Het verwijst naar hoe goed een model voorspellingen kan maken op nieuwe, gelijkaardige data die niet tijdens het trainen aan bod kwam. Als een model niet in staat is om te generaliseren naar nieuwe data is men mogelijks aan het *overfitten*.

Algorithm 1 ANN trainen. Bron: [44][52]

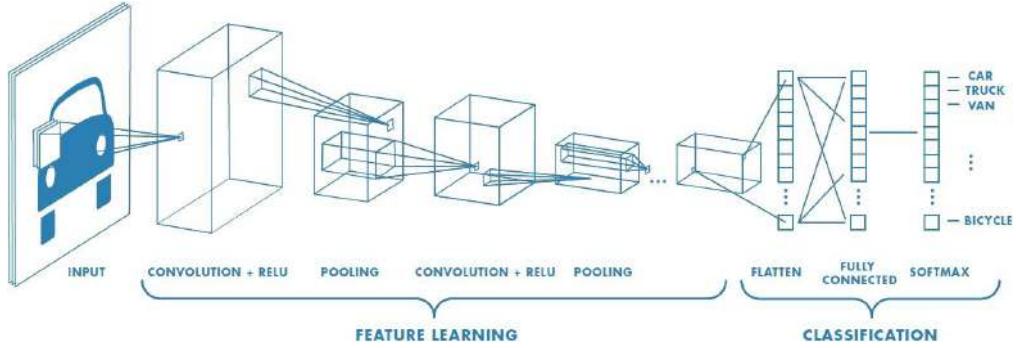
1. Initialiseer de gewichtenmatrix W , biasterm b
 2. Kies een learning rate η en *loss* functie \mathcal{L}
 3. Zolang niet elk voorbeeld uit de dataset 1 keer behandeld is:
 - a) Sample een random voorbeeld uit de dataset: $(x_i, y_i) \in D$
 - b) Bereken de verwachte output: $\hat{y}_i = f(W^T x_i + b)$
 - c) Bereken de *loss*: $\mathcal{L}_i = \mathcal{L}(\hat{y}_i, y_i)$
 - d) Bereken de gradiënten: $\Delta W = -\nabla_W \mathcal{L}_i$
 - e) Update de gewichten: $W = W + \eta \cdot \text{optimizer}(W, \Delta W)$.
 4. Herhaal stap 3 voor bepaald aantal *epochs*
-

2.1.2 Convolutionele Neurale Netwerken

Een 2D CNN ligt aan de basis van een 2DOD model. Figuur 2.3 toont hoe een CNN werkt om objecten te classificeren in een afbeelding. In tegenstelling tot voor de mens is het voor een computer niet vanzelfsprekend om objecten te detecteren in een afbeelding. De reden hiervoor is dat een afbeelding intern gerepresenteerd wordt door een pixel matrix, waarbij elke pixel bestaat uit drie kleurwaarden: rood, groen en blauw.

Een CNN wordt klassiek voorgesteld door *convolutionele* lagen, *pooling* lagen en FC lagen. Wanneer een afbeelding in zijn totaliteit wordt meegegeven aan een ANN, dan zou men voor elke pixel een corresponderend neuron krijgen. Voor een afbeelding van dimensie $(100 \times 100 \times 3)$ resulteert dit in 30.000 neuronen (en bijhorende gewichten), waardoor het trainen van het ANN computationeel niet haalbaar is. Een CNN is wel in staat om informatie uit een afbeelding te halen, door voorafgaande reductie van de pixel matrix.

2. LITERATUURSTUDIE

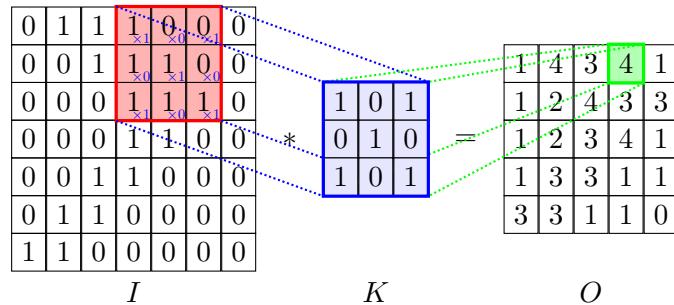


Figuur 2.3: De werking van een convolutioneel neuraal netwerk. Bron: [46].

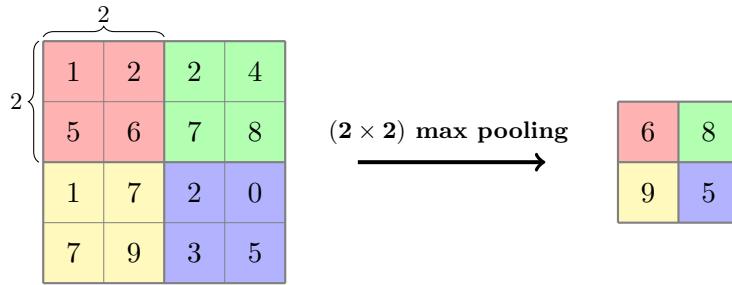
De operatie die de *convolutionele* laag uitvoert is te zien in figuur 2.4. Zij maakt gebruik van een kernel/filter K om te glijden over de pixel matrix I en genereert een output matrix O (activatiemap). Voor een afbeelding met drie kleurwaarden heeft de filter K , in dit voorbeeld, de dimensie $(3 \times 3 \times 3)$. Bij het glijden van de filter over de afbeelding berekent men telkens het inwendig product (convolutie) tussen I en K . Elke $(3 \times 3 \times 3)$ input wordt gereduceerd naar één scalar. Merk op dat de dimensie van een filter K ($m \times m \times 3$) en de specifieke toepassing bepalend is voor de dimensionaliteitsreductie.

Achter elke *convolutionele* laag wordt vaak een *max pooling* laag geplaatst. Deze zorgt opnieuw voor een dimensionaliteitsreductie. *Max pooling* heeft twee grote voordeelen. Enerzijds zal het netwerk minder parameters moeten leren wat computatieve gunstig is en anderzijds zal het netwerk minder snel *overfitten*. De werking van een *max pooling* laag is te zien in figuur 2.5. Van elke (2×2) submatrix wordt als output de maximum waarde teruggegeven.

Binnen een CNN zorgen de eerste *convolutionele* lagen voor *low-level features* zoals randen en kleuren. De diepere *convolutionele* lagen voegen op hun beurt deze *low-level features* samen en creëren een representatie van een afbeelding zoals een mens dat zou doen.



Figuur 2.4: *Convolutionele* operatie op een input matrix.


 Figuur 2.5: Een *max pooling* operatie.

2.2 2D Object Detectie

Het doel van 2DOD is het lokaliseren van objecten in een afbeelding. Een 2DOD model neemt als input een afbeelding en genereert als output alle relevante 2DBB's en hun bijbehorende klasse label. Een 2DBB is een zo klein mogelijke rechthoek die nog het hele object omvat en wordt gekarakteriseerd door vier parameters ($u_{min}, u_{max}, v_{min}, v_{max}$). Hierbij zijn (u_{min}, v_{min}) de coördinaten van de linkerbovenhoek en (u_{max}, v_{max}) de coördinaten van de rechterbenedenhoek. Een voorbeeld is te zien in de linker afbeelding van figuur 1.2.

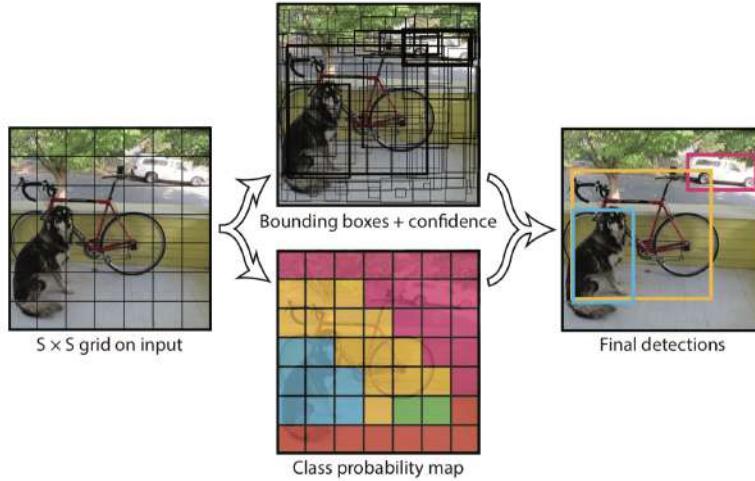
De moderne 2DOD modellen worden opgesplitst in twee groepen: *single-stage* detectors en *two-stage* detectors. Het voorspellen van 2DBB's en hun bijbehorende klasse probabiliteit gebeurt bij *single-stage* detectors in één enkele stap en bij *two-stage* detectors in twee stappen.

2.2.1 Single-stage detectors

Single-stage detectors voorspellen de 2DBB's en bijbehorende klasse probabiliteit met één netwerk. De meest bekende *single-stage* detectors zijn *You Only Look Once* (YOLO) [41], *Single Shot MultiBox Detector* (SSD) [28] en RetinaNet [26]. Ze worden hieronder verduidelijkt.

De afbeelding die aan het YOLO netwerk wordt meegegeven, wordt eerst opgedeeld in een rooster bestaande uit $S \times S$ cellen. Voor elke cel in dit rooster wordt er een bepaald aantal 2DBB's en bijbehorende klasse probabiliteiten voorspeld. Een 2DBB bestaat hier uit vijf componenten: x, y, w, h en een *confidence* score die aantoon hoe zeker het netwerk is dat een 2DBB een object bevat en hoe accuraat deze voorspelde 2DBB is. Hierbij zijn (x, y) en (w, h) het centrum respectievelijk de breedte en hoogte van een 2DBB. Merk op dat (x, y) relatief is bepaald ten opzichte van de grenzen van de cel en dat (w, h) ervoor zorgt dat de 2DBB groter kan zijn dan de cel waarin deze ligt.

2. LITERATUURSTUDIE



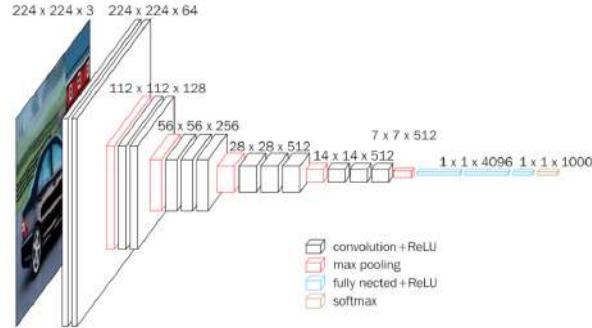
Figuur 2.6: De werking van een YOLO netwerk. Bron [41].

De *confidence* score wordt als volgt gedefinieerd:

$$P(class_i|object) \times P(object) \times IoU_{pred}^{truth} = P(class_i) \times IoU_{pred}^{truth} \quad (2.4)$$

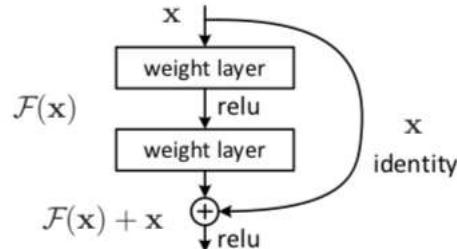
Formule 2.4 wordt gebruikt om aan elke 2DBB juist één klasse toe te kennen. Met de conditionele kans $P(class_i|object)$ wordt een probabiliteitsmap over het rooster opgesteld, zie figuur 2.6. Daarnaast wordt ook de bekende *Intersectie over Unie* (IoU) coëfficiënt gebruikt die meet hoe goed twee 2DBB's overlappen in de afbeelding. Uiteindelijk wordt er per cel juist één klasse voorspeld. Dit is de klasse met de grootste probabiliteit.

De SSD architectuur is complexer dan die van YOLO. Het maakt gebruik van een VGG-16 [48] netwerk om features uit afbeeldingen te halen. Het VGG-16 netwerk is een CNN dat bestaat uit 16 lagen, zoals te zien is op figuur 2.7. Het netwerk neemt als input een afbeelding van $(224 \times 224 \times 3)$ en bestaat uit 13 convolutionele en 3 FC lagen. Daarnaast bevat het ook *max pooling* lagen en een *softmax* laag op het einde. De *softmax* laag is een functie die de output van de laatste FC laag normaliseert tussen 0 en 1, waarbij de som van alle genormaliseerde termen sommeert tot 1. Deze eigenschap wordt gebruikt om een probabiliteitsdistributie over alle mogelijke klassen te definiëren. Het VGG-16 netwerk zal in deze thesis gebruikt worden, zie sectie 4.2. In de SSD architectuur komen achter het VGG-16 netwerk nog enkele convolutionele lagen. Verschillende feature lagen worden gebruikt om op meerdere schaalgroottes objecten te kunnen detecteren. Dit in contrast met YOLO dat slechts objecten op één schaalgrootte kan detecteren.



Figuur 2.7: VGG-16 architectuur. Bron [31].

Single-stage detectors hebben het voordeel dat ze snel voorspellingen kunnen maken en een eenvoudige architectuur hebben. Deze architecturen vinden hun toepassingen in real-time applicaties. Door hun lagere nauwkeurigheid kunnen *single-stage* detectors in het algemeen niet concurreren met de *two-stage* detectors. RetinaNet[26], een *single-stage* detector, is een van de weinige die er in geslaagd is om een *two-stage* detector op nauwkeurigheid te kloppen. RetinaNet maakt gebruik van een ResNet[17] en een Feature Pyramid Network (FPN) [25] als backbone. De FPN architectuur zorgt ervoor dat er op meerdere schaalgroottes features worden geleerd, wat de nauwkeurigheid van het model verhoogt. Naast de *convolutionele* lagen is een ResNet uitgerust met *residual* connecties die de output van bepaalde *convolutionele* lagen verbinden met volgende *convolutionele* lagen. Het mooie aan deze architectuur is dat het niet een originele functie tracht te benaderen maar een *residual* functie, wat men in de literatuur ook wel *residual learning* noemt. Stel dat $\mathcal{H}(x)$ een onderliggende functie is die wordt benaderd door (een deel van) het netwerk, met x de input van een vorige laag. In plaats van $\mathcal{H}(x)$ te benaderen, benadert een ResNet de *residual* functie $\mathcal{F}(x) = \mathcal{H}(x) - x$. De orginele te leren functie $\mathcal{H}(x)$ kan dan geschreven worden als $\mathcal{H}(x) = \mathcal{F}(x) + x$ [17]. Het gebruik van *residual learning* leidt vaak tot betere resultaten en wordt in het verdere onderzoek van deze thesis ook nog gebruikt. In figuur 2.8 is het concept van *residual learning* te zien.


 Figuur 2.8: Het concept van *residual learning*. Bron: [17].

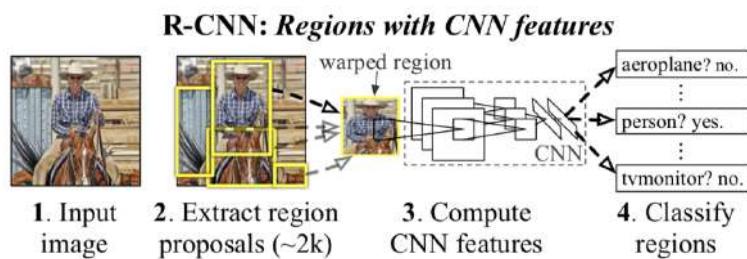
2. LITERATUURSTUDIE

2.2.2 *Two-stage* detectors

Two-stage detectors voorspellen de 2DBB's en bijbehorende klasse probabiliteit in twee stappen. De eerste stap bestaat uit een *region proposal* waaruit meerdere interessante regio's voortvloeien. De tweede stap gaat deze regio's classificeren en mogelijk verfijnen. De meest bekende *two-stage* architecturen zijn het *Regio-gebaseerd Convolutioneel Neuraal Netwerk* (R-CNN)[14] en zijn opvolgers Fast R-CNN[13] en Faster R-CNN[42].

R-CNN maakt gebruik van het selective search algoritme [50] om 2DBB voorstellen te genereren. Vervolgens worden deze voorstellen meegegeven aan een CNN dat een feature vector genereert. Deze vector wordt op zijn beurt meegegeven aan een *Support Vector Machine* (SVM) die als output een *confidence score* berekent voor elke mogelijke klasse. Ten slotte wordt *Non-Maximum Suppression* (NMS) gebruikt om redundante 2DBB's in de afbeelding weg te filteren. Tijdens NMS worden de 2DBB's in afnemende klassenprobabiliteit iteratief doorlopen. Voor elke 2DBB worden alle 2DBB's met een lagere probabiliteit en een IoU groter dan een bepaalde drempelwaarde verwijderd. Het voornaamste probleem met R-CNN is dat het ongeveer 50 seconden duurt om 2DBB's te voorspellen in een afbeelding. De werking van R-CNN is te zien in figuur 2.9.

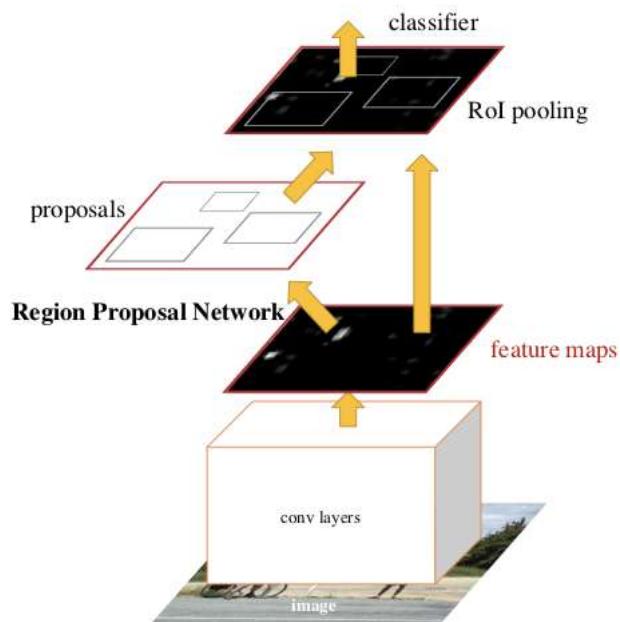
Fast R-CNN loste enkele nadelen van R-CNN op door een sneller 2DOD algoritme te maken. De architectuur is vergelijkbaar met deze van R-CNN maar in plaats van 2DBB voorstellen aan een CNN te geven wordt de oorspronkelijke afbeelding meegegeven. Uit de features van dit CNN worden dan 2DBB voorstellen gemaakt. De reden dat Fast R-CNN veel sneller is dan R-CNN is omdat men niet elke keer ~ 2000 voorstellen aan een CNN moet meegeven. Fast R-CNN gebruikt de convolutiebewerking slechts één keer per afbeelding. De bottleneck van Fast R-CNN blijft nog steeds het genereren van de 2DBB voorstellen.



Figuur 2.9: De werking van het R-CNN netwerk. Bron: [14].

Zowel R-CNN als Fast R-CNN maken gebruik van het selective search algoritme om 2DBB voorstellen te genereren. Dit algoritme is een tijdrovend proces dat de prestaties van het netwerk sterk beïnvloedt. Het Faster R-CNN model is zeer gelijkaardig aan het Fast R-CNN model, waarbij het selective search algoritme vervangen wordt door een *Region Proposal Network* (RPN). In tegenstelling tot het selective search algoritme leert het RPN interessante regio's te genereren, wat het veel sneller maakt. De werking van Faster R-CNN is te zien in figuur 2.10

Als men moet kiezen tussen een *single-stage* detector of een *two-stage* detector moet men altijd een trade-off maken tussen snelheid versus nauwkeurigheid. Daar waar *single-stage* detectors qua snelheid de beste zijn, presteren *two-stage* detectors qua nauwkeurigheid het beste. Aangezien de focus in deze thesis ligt op 3DOD en men een goede 2D detector nodig heeft, is nauwkeurigheid de belangrijkste factor. In deze thesis zal Faster R-CNN gebruikt worden om 2DBB's te genereren voor de KITTI dataset.



Figuur 2.10: De architectuur van het Faster R-CNN netwerk. Bron: [42].

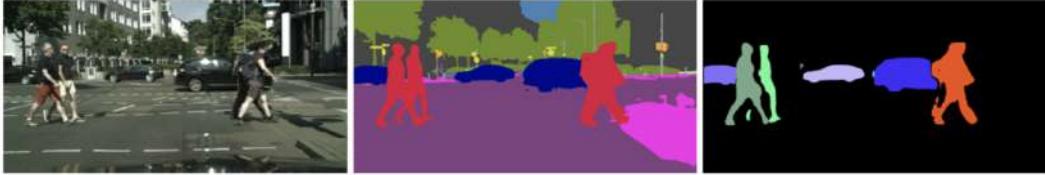
2.3 3D Object Detectie

Hoewel er over 2DOD in de literatuur al enorm veel is onderzocht, blijft 3DOD een open probleem dat nog veel onderzoek vereist. De motivatie om 3DOD te onderzoeken is dat het een noodzakelijke module is voor een autonoom voertuig. Om goed te kunnen reageren op dynamische situaties, moet een dergelijk voertuig kunnen redeneren over welke objecten er zich in de 3D scène bevinden, evenals over hun 3D grootte, locatie en oriëntatie. Een 3DBB is een georiënteerde rechthoekige kubus in een 3D-ruimte die idealiter minimaal moet zijn zodat het het bijbehorende object zo goed mogelijk omvat. Voor autonome rijtoepassingen wordt zo'n 3DBB geparametriseerd door $(x, y, z, h, w, l, \theta)$. Hierbij zijn (x, y, z) de coördinaten van het 3DBB centrum, (h, w, l) respectievelijk de hoogte, breedte en lengte van de 3DBB en θ de yaw hoek van de 3DBB. De yaw hoek wordt gemeten rond de y-as in een rechtsdraaiend assenstelsel. Vanuit het oog van een autonome rijtoepassing worden objecten geacht op de grond te staan. Vandaar dat de *pitch* en *roll* hoek worden verondersteld nul te zijn of van verwaarloosbaar belang.

2.3.1 Camera gebaseerde methodes

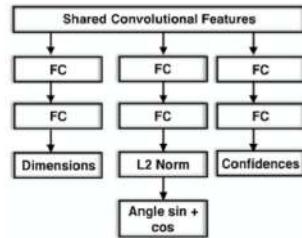
Wat betreft de 3DOD methoden die gebruik maken van een camera, kan men een onderverdeling maken: degene met een enkele camera (monoculair) en die met twee camera's (stereo). Het gebruik van twee camera's heeft als voordeel dat men meer informatie uit een 3D omgeving kan halen, zoals het genereren van een dieptekaart. In wat volgt focust men enkel op het gebruik van één monoculaire camera.

Ondanks het feit dat afbeeldingen enkel 2D informatie bevatten zijn er toch een heel aantal architecturen die trachten om 3DOD op basis van afbeeldingen uit te voeren. Een van de eerste 3DOD papers was Mono3D [6]. Er wordt uit elke afbeelding een aantal features gehaald die bijdragen tot de 3DBB bepaling. De features die bepaald worden zijn: de 2DBB, de semantische segmentatie, de instance segmentatie, de contextuele informatie (pixel informatie rond de 2DBB) en de 3D locatie van het object. Semantische segmentatie is het proces waarbij elke pixel in een afbeelding wordt ingekleurd naar gelang tot welke klasse het behoort. Objecten van dezelfde klasse krijgen dus dezelfde kleur. In tegenstelling tot semantische segmentatie gaat instance segmentatie elk object een andere kleur geven. Objecten van dezelfde klasse krijgen dus een andere kleur. Het verschil tussen de twee is te zien in figuur 2.11. Vervolgens worden deze features gecombineerd en wordt er een score berekend die beslist welke 3DBB voorstellen nuttig zijn.



Figuur 2.11: Semantische segmentatie (midden) vs instance segmentatie (rechts).
Bron: [8].

Een andere methode werd in Deep3DBox [30] gepresenteerd. Ten opzichte van vorige papers had het twee belangrijke contributies. Eerst wordt het 3D centrum van de 3DBB geometrisch bepaald. De paper maakt gebruik van het feit dat de perspectiefprojectie van een 3DBB perfect binnen een 2DBB past. Vervolgens introduceren de auteurs een *Multi-Bin Module* die gebruikt wordt om de dimensie en oriëntatie van een 3DBB te bepalen. Figuur 2.12 toont de werking van deze module. Uit elke afbeelding worden eerst de 2DBB's gesneden en worden dan meegegeven aan een VGG-16 [48] netwerk om features te bepalen.



Figuur 2.12: De *Multi-Bin* architectuur. Bron: [30].

Vervolgens worden uit de FC lagen de dimensie (h, w, l) en globale oriëntatie $\theta_{globaal}$ van de 3DBB bepaald. Er zijn twee oriëntaties in het 3DOD probleem: de globale oriëntatie $\theta_{globaal}$ en de lokale oriëntatie θ_{lokaal} . Op basis van een bijgesneden afbeelding kan men de globale oriëntatie van een object niet berekenen aangezien die in de tijd lijkt te veranderen, zie figuur 2.13. Deze verandering is te danken aan de verandering van de lokale oriëntatie. Men kan het netwerk dus enkel de lokale hoek θ_{lokaal} laten voorspelen. Uit figuur 2.14 haalt men het verband:

$$\theta_{ray} = \theta_{lokaal} + (-\theta_{globaal}) \quad (2.5)$$

De hoek θ_{ray} is de hoek tussen de lijn die door het middelpunt van de bijgesneden afbeelding gaat en de z-as van het camera assenstelsel. De hoek θ_{lokaal} is de hoek tussen diezelfde lijn en de x-as van het lokale auto assenstelsel. De hoek θ_{ray} kan op basis van de bijgesneden afbeelding en de camera intrinsieke matrix berekend worden. De globale oriëntatie $\theta_{globaal}$ volgt dan uit formule 2.5.

2. LITERATUURSTUDIE

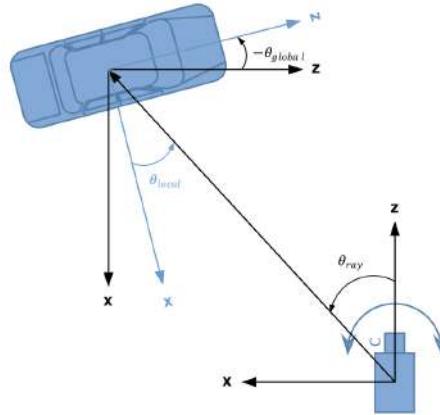


Figuur 2.13: De globale oriëntatie van de auto lijkt schijnbaar te veranderen in de bijgesneden afbeeldingen. Bron: [30].

Om θ_{lokaal} te bepalen wordt een hoek over 360° eerst gediscretiseerd in n overlappende delen, verder *bins* genoemd. Elke bin wordt ook vertegenwoordigd door een 'centrale' hoek θ_{center,bin_i} die het midden van de bin voorstelt. Vervolgens berekent een CNN voor elke bin twee outputs. Eerst wordt er een *confidence* score c_i berekend die aangeeft wat de kans is dat θ_{lokaal} binnen een i^{de} bin ligt. Dan wordt de sinus en cosinus van een *residual* hoek $\Delta\theta_{lokaal,i}$ berekend die moet worden opgeteld bij θ_{center,bin_i} . Voor elke bin i krijgt men een output van de vorm: $(c_i, \sin(\Delta\theta_{lokaal,i}), \cos(\Delta\theta_{lokaal,i}))$. Uiteindelijk leidt de grootste waarde voor c_i tot de bepaling van θ_{lokaal} :

$$\theta_{lokaal} = \arctan(\sin(\Delta\theta_{lokaal,i}), \cos(\Delta\theta_{lokaal,i})) + \theta_{center,bin_i} \quad (2.6)$$

Gegeven de geschatte 3D dimensie (h, w, l) en de hoek $\theta_{globaal}$ kan men via geometrische *constraints* de 3D locatie bepalen. Hoofdstuk 4 behandelt meer in detail deze geometrische *constraints* aangezien deze architectuur aan de basis ligt van de camera gebaseerde methode die uitgewerkt is in deze thesis.



Figuur 2.14: Illustratie van alle oriëntaties in vogelperspectief. Bron: [27].

De bottleneck van de meeste 3DOD methodes is dat het de locatie van de 3DBB slecht kan voorspellen door gebrek aan diepte informatie. De eerste methode die hierop een antwoord wou bieden was MF3D [55]. Het mooie aan de MF3D architectuur is dat het gebruikmaakt van een subnetwerk (gebaseerd op [15]) dat de diepte in een afbeelding leert schatten. Het resultaat is een dieptekaart die vervolgens wordt omgezet in een puntenwolk. Een ander subnetwerk, gebaseerd op Faster R-CNN [42], is verantwoordelijk om 2DBB's te voorspellen en hieruit features te halen. Uiteindelijk worden de features uit de puntenwolken en afbeeldingen samengevoegd om de 3D dimensie, locatie en oriëntatie te bepalen.

Op een gelijkaardige manier voorspellen de auteurs in MonoGRNet [39] hun 3DBB's. Een belangrijke verbetering ten opzicht van MF3D [55] is het *Instance Depth Estimation* (IDE) subnetwerk. Dit IDE netwerk focust enkel op de diepteschatting van objecten binnen een 2DBB. Doordat men enkel focust op de relevante objecten in een afbeelding bekomt men niet enkel een nauwkeurige diepteschatting maar is dit computationeel gezien ook veel efficiënter.

De aanpak van de OFT-Net [43] paper is compleet anders dan zijn voorgangers. In plaats van 3DBB's rechtstreeks uit een afbeelding te voorspellen, laat het het netwerk een *orthographic feature transform* (OFT) leren. Het doel van OFT-Net is dat het orthografische features berekent die vergelijkbaar zijn met wat men in de literatuur *bird's-eye-view* (BEV) of vogelperspectief features noemt. Deze BEV features worden vaak bij lidar gebaseerde 3DOD methodes gebruikt. De onderbouwing van deze architectuur is vrij intuïtief. Objecten die zich ver in de afbeelding bevinden worden gekarakteriseerd door een klein aantal pixels. Als men dit probleem bekijkt vanuit het vogelperspectief nemen objecten van dezelfde klasse gemiddeld gezien even veel plaats in beslag. Door gebruik te maken van de BEV features is een CNN in staat om ook belang te hechten aan objecten ver in een afbeelding. Deze BEV features worden in een laatste fase meegegeven aan een CNN dat als output de dimensie, locatie en oriëntatie berekent van een 3DBB.

Dat het 3DOD probleem ook anders kan aangepakt worden, toont de recent gepubliceerde paper RTM3D [23]. In tegenstelling tot vorige papers vertrekken de auteurs niet vanuit een 2D detector om hieruit 3DBB's te extraheren. Eerst berekent men met een set van FC lagen de negen *keypoints* van een wagen. Deze *keypoints* zijn de 2D projectie van de acht hoekpunten en het centrum van de 3DBB. Door gebruik te maken van het feit dat de (h, w, l) van een 3DBB een constante is kan men geometrische *constraints* opleggen aan deze *keypoints*. In een volgende stap minimaliseert men de projectie error van de negen *keypoints*. Deze paper is op dit moment de *State Of The Art* (SOTA) camera gebaseerde architectuur voor de KITTI dataset die tevens real-time opereert.

2.3.2 Lidar gebaseerde methodes

Lidar is een acroniem voor *Light Detection And Ranging* dat qua functionaliteit sterk lijkt op een radar. Het verschil tussen een lidar en radar sensor is dat lidar gebruikmaakt van laserlicht terwijl radar gebruikmaakt van radiogolven. Een lidar sensor bestaat uit twee belangrijke elementen: een zender en ontvanger. De zender stuurt continu laserlicht uit in de ruimte totdat het een object raakt en het laserlicht (gedeeltelijk) wordt gereflecteerd naar de zender. Dit gereflecteerde laserlicht wordt opgevangen door de ontvanger. Door het tijdsverschil te meten tussen zenden en ontvangen van een laserlicht kan men de afstand tot een object bepalen. Theoretisch gezien kan de afstand met onderstaande formule berekend worden [54]:

$$d = \frac{c \cdot t}{2 \cdot n} \quad (2.7)$$

waarbij d de afstand is in meter, c de lichtsnelheid, t de tijd in seconden en n de brekingsindex.

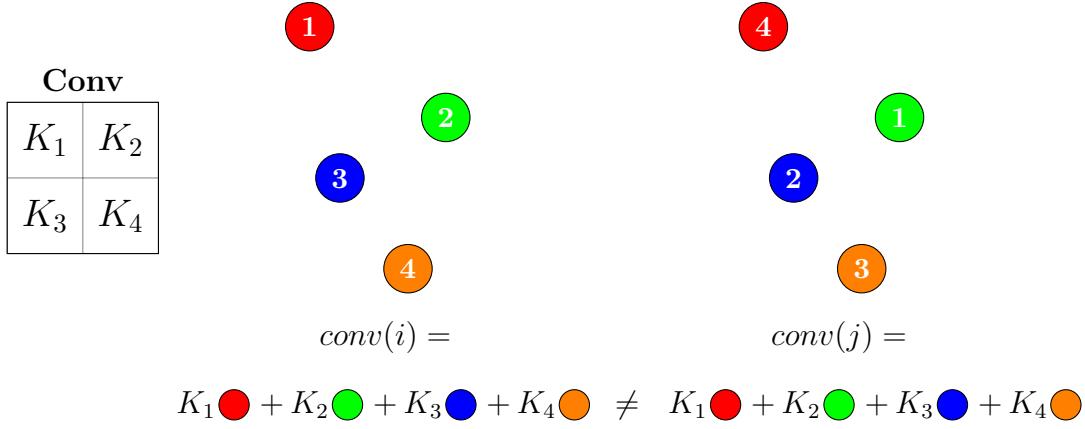
Doorgaans bevat een lidar sensor meerdere zender-ontvanger paren die op een roterende behuizing worden gemonteerd. Elk paar zal dan per omwenteling duizenden afstandsmeetingen per seconde uitvoeren. Zo krijgt men een puntenwolk in 360° die de 3D omgeving zo goed mogelijk beschrijft. Wiskundig gezien wordt een puntenwolk gekarakteriseerd als een set van n punten:

$$P = \{p_1, \dots, p_n\} \subset \mathbb{R}^4 \quad (2.8)$$

waarbij $p_i = (x_i, y_i, z_i, r_i) \in \mathbb{R}^4$ een punt is met 3D coördinaten (x_i, y_i, z_i) en r_i een reflectiewaarde.

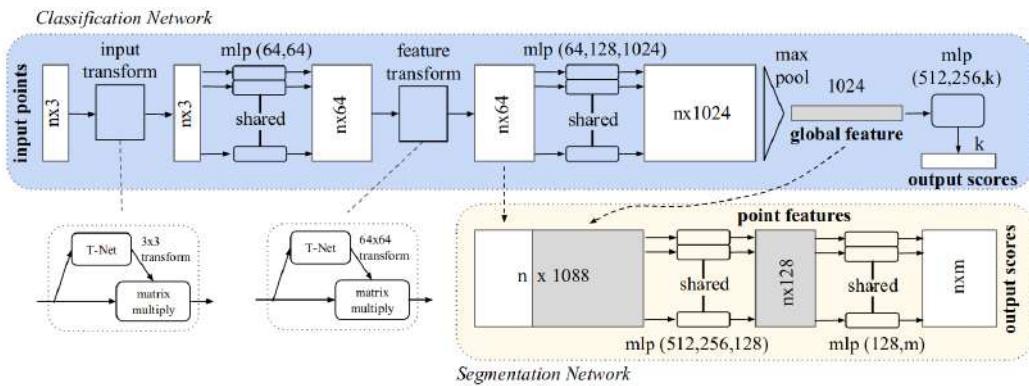
Als men een puntenwolk vergelijkt met een afbeelding zijn er twee grote verschillen. Een puntenwolk is een *sparse* 3D representatie en een afbeelding is een *dense* 2D representatie. Daarnaast is een puntenwolk een ongeordende set 3D-punten dat invariant is voor permutaties. Door deze invariante eigenschap kan een puntenwolk niet zomaar meegegeven worden aan een CNN dat gemaakt is om met afbeeldingen te werken. Beschouw de twee puntenwolken in figuur 2.15. Ondanks dat de puntenwolken slechts een permutatie van elkaar zijn en ze dezelfde structuur voorstellen, geldt er dat $\text{conv}(i) \neq \text{conv}(j)$. Voor men een convolutiebewerking kan toepassen moet men een functie f zoeken met de volgende eigenschappen [40]:

- Invariant voor permutaties, in het voorbeeld betekent dit dat $f(i) = f(j)$
- Invariant voor geometrische transformaties zoals rotaties of translatie van de hele puntenwolk
- Gevoelig voor de lokale structuur van een puntenwolk



Figuur 2.15: Het puntenwolk probleem.

PointNet [37] was een pionier in het ontwikkelen van een neuraal netwerk dat direct puntenwolken kon verwerken. Het netwerk is te zien in figuur 2.16. Het netwerk kan zowel classificatie als segmentatie uitvoeren. Het *classification network* neemt als input een puntenwolk van de vorm $(n \times 3)$, waarbij n het aantal punten is en 3 verwijst naar de (x, y, z) coördinaten. PointNet bevat drie belangrijke modules. Ten eerste lost het het invariant zijn voor permutaties op door een eenvoudige symmetrische functie te gebruiken, de *max pooling* laag. Het *classification network* berekent twee features: de lokale en de globale. Deze globale features kunnen gebruikt worden om een MLP te trainen dat de puntenwolken classificeert. De combinatie van lokale en globale features zorgt ervoor dat het *segmentation network* elk punt kan segmenteren. Om de segmentatie coherent te houden onder rotaties en translaties hanteert men het T-Net. Het T-Net leert een affiene transformatiematrix en past deze transformatie rechtstreeks toe op de puntenwolk die als input wordt meegegeven. In een latere fase vindt deze transformatie ook nog eens plaats op berekende features.

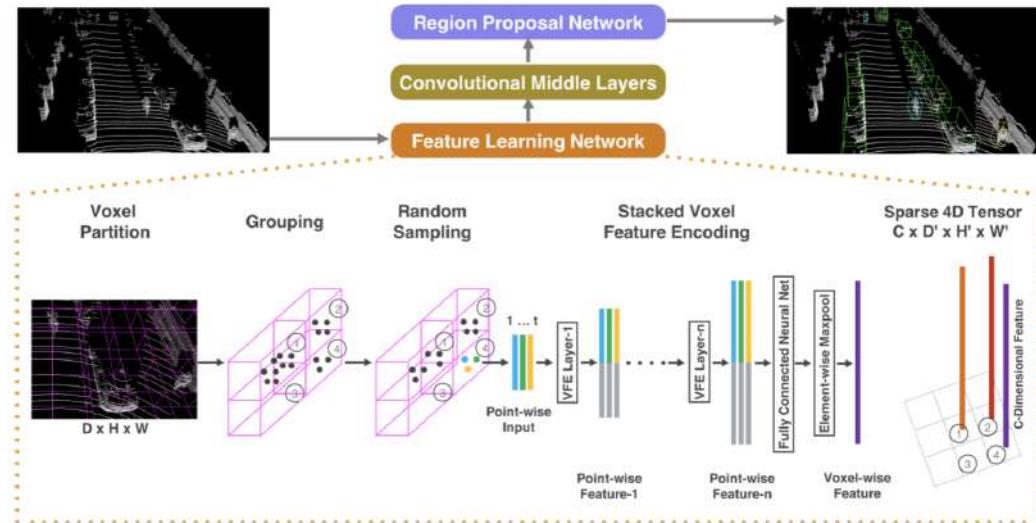


Figuur 2.16: De PointNet architectuur. Bron: [37].

2. LITERATUURSTUDIE

PointNet ligt aan de basis van de Frustum-PointNet [36] architectuur die gebruikt wordt om 3DOD uit te voeren. In hoofdstuk 4 wordt het Frustum-PointNet in detail besproken aangezien deze architectuur wordt geïmplementeerd in deze thesis.

In de literatuur transformeert men punten binnen een puntenwolk ook vaak naar een volumetrische pixel of ook wel voxel genoemd. Een voxel is het driedimensionale equivalent van een pixel in een afbeelding. Vote3Deep [9] was een van de eerste die gebruik maakte van voxels. In deze methode wordt een puntenwolk omgezet naar een *sparse occupancy grid* dat weergeeft welke cellen bezet zijn door punten en hun bijbehorende probabilititeit. Een 3D convolutie berekenen op het hele grid zou computationally niet haalbaar zijn. Om de *sparse* eigenschap ten volle te benutten wordt er een *voting mechanisme* gebruikt dat beslist in welke niet-lege cellen de 3D convolutie moet berekend worden. Uiteindelijk wordt er een CNN gebruikt om 3DBB's te genereren. Een sterk verbeterde architectuur werd gepresenteerd in VoxelNet [59]. Dit netwerk bestaat uit drie belangrijke componenten: een *feature learning netwerk*, *convolutionele lagen* en een RPN [42], zie figuur 2.17. Eerst wordt de 3D-ruimte verdeeld in disjuncte voxels van gelijke grootte. Hierdoor worden punten gegroepeerd op basis van de corresponderende voxel. De niet-lege voxels worden dan meegegeven aan een *Voxel Feature Encoding* netwerk. Dit netwerk, gelijkaardig aan dat van PointNet [37], berekent per voxel een feature. De output hiervan is een *sparse 4D tensor* die vervolgens meegegeven wordt aan een 3D CNN. Het resultaat is een 3D feature map dat gebruikt wordt door het RPN netwerk om 3DBB's te genereren.



Figuur 2.17: De VoxelNet architectuur. Bron: [59].

Een ander alternatief om een puntenwolk te verwerken is door gebruik te maken van een *Graph CNN* (GCNN). Hierbij wordt een puntenwolk gezien als een graaf waarbij de knopen voorgesteld worden door punten. Binnen een bepaalde straal zullen naburige knopen voorgesteld worden door een lijn. Door te werken met een graaf leert men meteen de lokale geometrische eigenschappen van een puntenwolk en kan men hierop een convolutie toepassen. Daar waar PointNet [37] elk punt onafhankelijk behandelt tracht een GNN een lokale geometrische structuur te leren wat leidt tot betere resultaten. De laatste jaren trachten studies zoals [38] en [51] deze GNN toe te passen op classificatie en segmentatie van een puntenwolk. De recent gepubliceerde paper Point-GNN [47] is de eerste paper die een GNN gebruikt voor 3DOD. Deze architectuur staat momenteel in de top 10 best presterende 3DOD architecturen voor de KITTI dataset.

2.3.3 Camera en lidar gebasseerde methodes

In de literatuur vindt men ook veel papers terug die gebruik maken van zowel afbeeldingen als puntenwolken om 3DOD uit te voeren. In het algemeen zijn er drie manieren om deze twee modaliteiten te combineren. De verschillende manieren om de twee modaliteiten te combineren worden hieronder besproken en zijn te zien in figuur 2.18.

- **Early fusion**

Early fusion, ook wel value fusion genoemd, combineert de ruwe data van afbeeldingen en puntenwolken samen tot een gefuseerde vector. Deze vector vormt dan de input voor één classifier. Een voordeel hiervan is dat de classifier meer informatie bevat wat tot potentieel betere resultaten kan leiden.

- **Late Fusion**

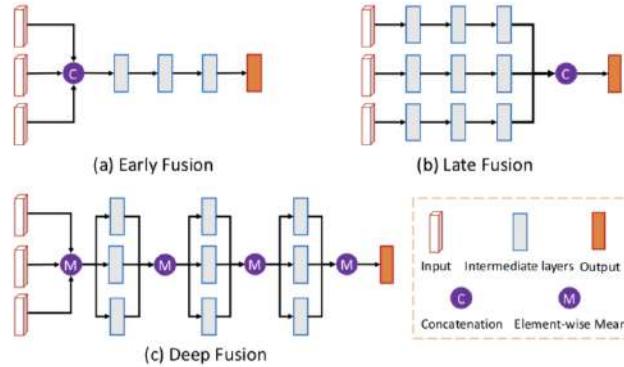
Late fusion, ook wel decision fusion genoemd, maakt de combinatie van een afbeelding classifier en lidar classifier. De twee modaliteiten worden dus apart verwerkt en genereren elk hun eigen features. Het combineren van twee verschillende classifiers in een latere fase kan leiden tot het maken van een nauwkeurigere, betrouwbaardere beslissing.

- **Deep fusion**

Deep fusion is vergelijkbaar met early fusion waarbij feature vectoren meerdere keren worden gefuseerd doorheen het netwerk.

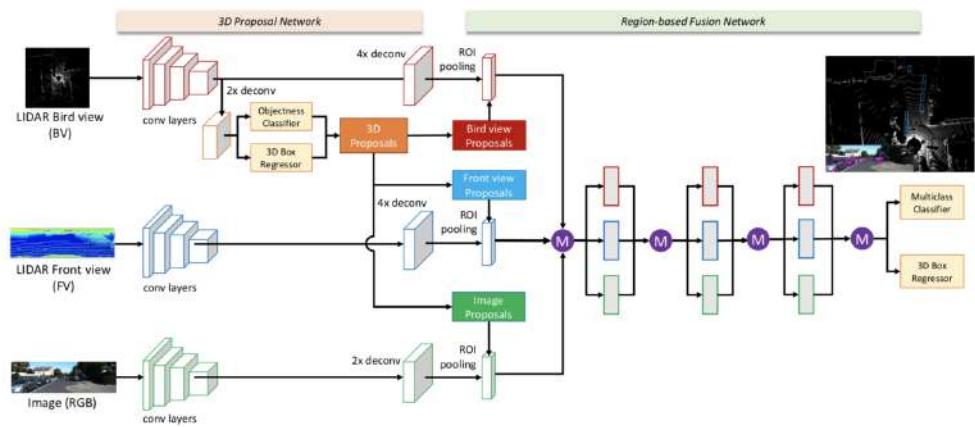
Een van de eerste architecturen die de twee modaliteiten combineerde was PointFusion [56]. PointFusion gebruikt PointNet [37] als backbone netwerk om features uit puntenwolken te halen en een ResNet [17] om features uit afbeeldingen te halen. Deze features worden daarna gecombineerd en meegegeven aan een MLP. Dit netwerk bepaalt uiteindelijk de 3DBB's. PointFusion is een goed voorbeeld van late fusion.

2. LITERATUURSTUDIE



Figuur 2.18: De mogelijke fusies van camera en lidar informatie. Bron: [7].

Een vergelijkbare architectuur werd gepresenteerd in MV3D [7]. Deze architectuur neemt drie inputs: een afbeelding, een lidar puntenwolk in vooraanzicht en een lidar puntenwolk in vogelperspectief. Een overzicht van de architectuur is te zien in figuur 2.19. Binnen de MV3D architectuur gebruikt men een subnetwerk, gebaseerd op een RPN [42], voor het genereren van 3DBB voorstellen. De voorstellen worden gegenereerd vanuit het vogelperspectief en worden dan geprojecteerd op de andere twee modaliteiten. Er valt op te merken dat MV3D gebruikmaakt van deep fusion doorheen hun netwerk. Momenteel is de SOTA architectuur voor de KITTI dataset die gebruikmaakt van zowel afbeelding als lidar een paper van Uber, genaamd MMF [24]. Hierin laat men zien dat 3DOD significant verbetert door het toepassen van wat men in de literatuur *multi-task learning* noemt. Men leert vier taken: mapping (het schatten van de weg geometrie), 2D detectie, 3D detectie en depth completion. Het mooie van *multi-task learning* is dat er kennis kan uitgewisseld worden tussen elke taak. Door zowel gebruik te maken van twee modaliteiten als *multi-task learning* verhoogt de nauwkeurigheid van de voorspelde 3DBB's.



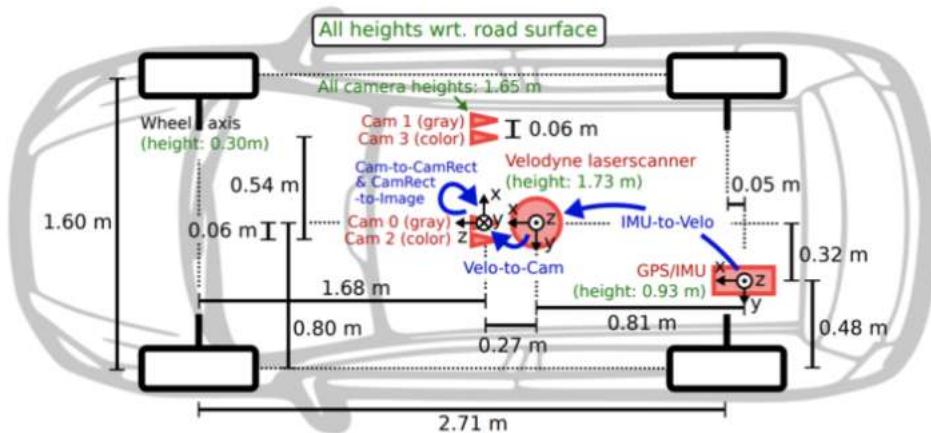
Figuur 2.19: De MV3D architectuur. Bron: [7].

Hoofdstuk 3

Datasets

3.1 KITTI dataset

De KITTI Vision Benchmark Suite [11] is een dataset die dateert uit 2012 en gemaakt is door het Karlsruhe Institute of Technology en het Toyota Technological Institute in Chicago. De dataset is verzameld met een auto die is uitgerust met twee kleuren- en twee grijswaardencamera's, een Velodyne lidar en een GPS/IMU-eenheid, zie figuur 3.1. Daarnaast organiseren de auteurs een online benchmark competitie voor taken zoals diepte schatting, 2D, 3D en BEV object detectie. Vandaar dat deze dataset de voorbije jaren enorm veel gebruikt is geweest in de literatuur. In deze thesis focust men op de 3DOD benchmark competitie.



Figuur 3.1: De karakteristieken van de auto en de sensoren waarmee de KITTI dataset is verzameld. Bron: [12].

3. DATASETS

3.1.1 Dataset beschrijving

De 3DOD dataset bevat 7481 gelabelde frames, opgesplitst in een train en validation set. Van de 7841 gelabelde frames vormen 3712 frames de train set en 3769 frames de validation set. Daarnaast zijn er ook nog eens 7518 niet-gelabelde frames die gebruikt worden als test set. Elk frame bevat op zijn beurt een RGB-afbeelding en een lidar puntenwolk. Daarnaast is er aan elk frame een tekstbestand gekoppeld dat alle objecten in de scène beschrijft. Elk beschrijving van een object bestaat uit 15 parameters, die te zien zijn in tabel 3.1.

| Nr. | Informatie | Waarden |
|-------|---|---|
| 1 | Type van het object | 'Car', 'Pedestrian', 'Van', 'Truck', 'Cyclist', 'Person_sitting', 'Tram', 'Misc', 'DontCare' |
| 2 | Informatie die aangeeft of een object truncated (mate waarin een object zich buiten het beeld bevindt) is of niet | Een waarde van 0.00 (niet-truncated) tot 1.00 ('truncated'), hoe hoger de waarde hoe minder zichtbaar het object is |
| 3 | Informatie die aangeeft of een object geoccludeerd (mate waarin objecten overlappen) is of niet | '0' - volledig zichtbaar '1' - gedeeltelijke occlusie '2' - grote occlusie '3' - onbekend |
| 4 | Locale hoek θ_{local} van het object | $\theta_{local} \in [-\pi, \dots, \pi]$ |
| 5-8 | 2DBB van het object | $(u_{min}, u_{max}, v_{min}, v_{max})$ |
| 9-11 | 3DBB dimensie | hoogte, breedte, lengte (in meter) |
| 12-14 | 3DBB locatie | x, y, z in camera coördinaten (in meter) |
| 15 | Rotatie θ_{global} van het object rond de Y-as | $\theta_{global} \in [-\pi, \dots, \pi]$ (in camera coördinaten) |

Tabel 3.1: Structuur van de annotaties in de KITTI dataset.

Bovendien wordt de dataset ook opgesplitst in drie categorieën naargelang hoe moeilijk het object te detecteren valt. De moeilijkheid neemt toe naarmate het object zich verder in de ruimte bevindt. Dit gaat immers gepaard met een kleiner aantal pixels en lidar punten. De drie categorieën zijn te zien in tabel 3.2. Ten slotte bevat elk frame ook een *camera kalibratie* tekstbestand. Hierin vindt men allerlei matrices terug, zoals bv. de camera intrinsieke matrix.

| Moeilijkheid | Min. 2DBB hoogte | Max. occlusie | Max. truncation |
|--------------|------------------|--------------------|-----------------|
| Easy | 40 pixels | volledig zichtbaar | 15% |
| Moderate | 25 pixels | deels zichtbaar | 30% |
| Hard | 25 pixels | moeilijk te zien | 50% |

Tabel 3.2: Definitie van een Easy, Moderate en Hard KITTI object.

3.1. KITTI dataset

Tabel 3.3 geeft het aantal objecten per klasse weer, verdeeld over de train en validatie set. Er valt op te merken dat er een groot klasse onevenwicht is binnen de KITTI dataset. Gezien het grote verschil tussen het aantal objecten per klasse is er gekozen om in het verdere verloop enkel te focussen op de auto's.

| Klasse | Train | Validatie | Totaal |
|------------|-------|-----------|--------|
| Auto | 14357 | 14385 | 28742 |
| Voetganger | 2207 | 2280 | 4487 |
| Fietser | 734 | 893 | 1627 |

Tabel 3.3: Klasse onevenwicht voor de KITTI dataset.

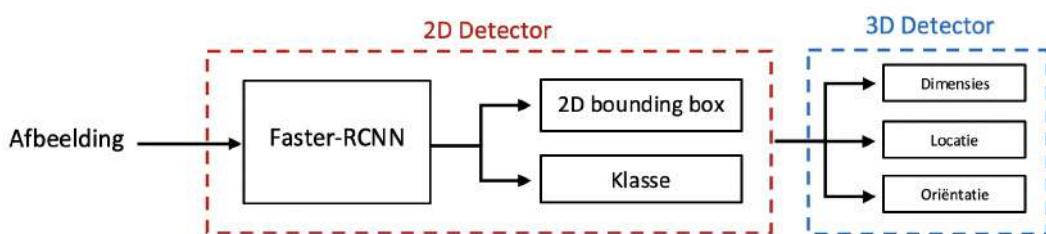
Hoofdstuk 4

Methodologie

In dit hoofdstuk worden de architecturen uitgelegd die in deze thesis zijn geïmplementeerd. Sectie 4.1 bespreekt de 2D detector, sectie 4.2 bespreekt de architectuur die gebruikmaakt van afbeeldingen, sectie 4.3 bespreekt de architectuur die gebruikmaakt van lidar en sectie 4.4 bespreekt de architectuur die gebruikmaakt van afbeeldingen en lidar.

4.1 2D object detector

Zoals besproken in de literatuurstudie zijn two-stage 2D object detectors in het algemeen de beste qua nauwkeurigheid. In deze thesis is er gekozen om een Faster-RCNN [42] te gebruiken voor het genereren van 2DBB's. Een ResNet-101 [17] werd als backbone CNN gekozen om features te genereren. Dit netwerk is pretrained op ImageNet [45] en gefinetuned op de KITTI dataset. In de eerste lagen van een pretrained netwerk ontstaan er meer generieke features, die minder afhankelijk zijn van een bepaalde dataset. Vandaar dat men vaak de laatste laag van een pretrained netwerk hertraint op de specifieke dataset. In dit geval wordt er gefinetuned op de KITTI dataset. Figuur 4.1 geeft de algemene architectuur weer die in het verdere verloop van deze thesis aan bod komt. Het bestaat uit twee modules: een 2D detector en een 3D detector. De 3D detector wordt in de volgende secties meer in detail besproken.

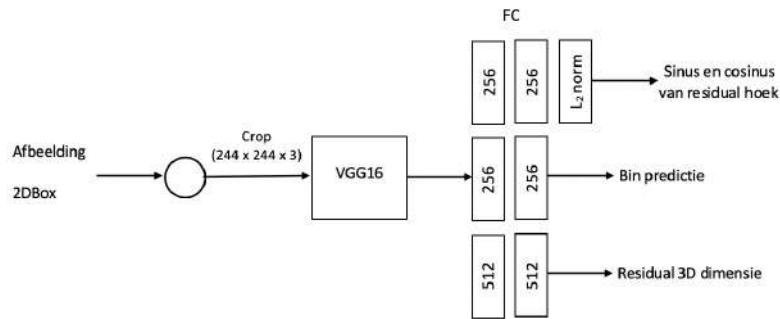


Figuur 4.1: De algemene architectuur voor het 2D-3D model.

4.2 3D object detector - Camera gebaseerd

4.2.1 Model architectuur

Het model dat 3DBB's op basis van afbeeldingen genereert, is gebaseerd op de Deep3DBox [30] architectuur. Een overzicht van de architectuur is te vinden in figuur 4.2. Deze architectuur werd al uitgebreid besproken in sectie 2.3.1. Uit elke afbeelding worden de 2DBB's bijgesneden en meegegeven aan een VGG16 netwerk dat features berekent. Vervolgens worden deze features gegeven aan drie FC takken. De eerste tak bevat twee FC lagen van dimensie 256 (met L2-normalisatie) en geeft een output van dimensie $2 * N_H$, met N_H het aantal *bins* dat is gebruikt. De tweede tak bevat twee FC lagen van dimensie 256 en geeft een output van dimensie N_H . De derde tak bevat twee FC lagen van dimensie 512 en geeft een output van dimensie 3. Daarnaast gebruiken alle FC lagen een ReLU activatiefunctie.



Figuur 4.2: De baseline RGB-architectuur.

Merk op dat men in deze architectuur de *residual 3D dimensie* schat en dat deze later moeten opgeteld worden bij de gemiddelde dimensie van een auto. Het bepalen van $\theta_{globaal}$ met het gebruik van *bins* werd reeds in de literatuurstudie uitgelegd, zie formule 2.6. De volgende paragraaf bespreekt in detail hoe de locatie geometrisch kan bepaald worden.

4.2.2 Implementatie details

Om de globale oriëntatie $\theta_{globaal}$ te bepalen wordt een hoek tussen $[-\pi, \dots, \pi]$ gediscretiseerd in N_H *bins*. In deze implementatie is er gekozen voor $N_H = 4$, met $\{0, \frac{\pi}{2}, -\pi, -\frac{\pi}{2}\}$ de centrale hoeken van elke bin. De volgende formule definieert het domein van elke bin:

$$\begin{cases} \theta \in bin_1 \iff \theta \in [-\frac{\pi}{4}, \frac{\pi}{4}] \\ \theta \in bin_2 \iff \theta \in [\frac{\pi}{4}, \frac{3\pi}{4}] \\ \theta \in bin_3 \iff \theta \in [\frac{3\pi}{4}, \pi] \cup [-\pi, -\frac{3\pi}{4}] \\ \theta \in bin_4 \iff \theta \in [-\frac{3\pi}{4}, -\frac{\pi}{4}] \end{cases} \quad (4.1)$$

Geometrische locatie bepaling

De 3D locatie wordt vervolgens geometrisch bepaald. Men vertrekt van de aannname dat de perspectiefprojectie van een 3DBB perfect binnen een 2DBB past. Een 3DBB is gekarakteriseerd door zijn centrum $T = [t_x, t_y, t_z]$, dimensie $D = [d_x, d_y, d_z]$ en globale oriëntatie $\theta_{globaal}$. Veronderstel nu dat de oorsprong van het assenstelsel van het object in het midden ligt van de 3DBB. Door gebruik te maken van de dimensie D kan men de acht hoekpunten van de 3DBB in dit assenstelsel als volgt beschrijven: $P_1 = [d_x/2, d_y/2, d_z/2]$, $P_2 = [-d_x/2, d_y/2, d_z/2]$, ..., $P_8 = [-d_x/2, -d_y/2, -d_z/2]$. De *constraint* dat een 3DBB perfect past binnen zijn 2DBB vereist dat er op elke zijde van 2DBB minstens één 3D hoekpunt wordt geprojecteerd.

Dit levert de volgende 4 *constraints* op:

$$\begin{aligned} x_{min} &= \left(K_{3 \times 3} [R_{3 \times 3} | T_{3 \times 1}] \begin{bmatrix} P_{3 \times 1}^{(1)} \\ 1 \end{bmatrix} \right)_x \\ x_{max} &= \left(K_{3 \times 3} [R_{3 \times 3} | T_{3 \times 1}] \begin{bmatrix} P_{3 \times 1}^{(2)} \\ 1 \end{bmatrix} \right)_x \\ y_{min} &= \left(K_{3 \times 3} [R_{3 \times 3} | T_{3 \times 1}] \begin{bmatrix} P_{3 \times 1}^{(3)} \\ 1 \end{bmatrix} \right)_y \\ y_{max} &= \left(K_{3 \times 3} [R_{3 \times 3} | T_{3 \times 1}] \begin{bmatrix} P_{3 \times 1}^{(4)} \\ 1 \end{bmatrix} \right)_y \end{aligned} \quad (4.2)$$

Hierbij refereert $(.)_x$ naar het x-coördinaat van de perspectiefprojectie, $(.)_y$ naar het y-coördinaat van de perspectiefprojectie, K naar de camera instrieke matrix, R naar de rotatie matrix en $P^{(1)}$ tot $P^{(4)}$ naar de perspectiefprojectie van de vier hoekpunten die op de 2DBB liggen.

Het resultaat is een overgedetermineerd stelsel met 4 vergelijkingen en 3 onbekenden, namelijk $T = [t_x, t_y, t_z]$. In theorie kan elk hoekpunt van de 3DBB geprojecteerd worden op elke zijde van de 2DBB, wat resulteert in $8^4 = 4096$ mogelijke configuraties. In de KITTI dataset zijn de *pitch* en de *roll* hoek verwaarloosbaar klein. Met deze kennis kan men het aantal configuraties verlagen naar 64. Eerst kan men aannemen dat projecties op de verticale 2DBB zijden enkel komen van verticale 3DBB zijden. Analoog kunnen projecties op de horizontale 2DBB zijden enkel komen van horizontale 3DBB zijden. Dit resulteert in $4^4 = 256$ mogelijke configuraties. Op basis van de kennis van θ_{lokaal} kan men het aantal configuraties nog verder verlagen. Het resultaat is: maximaal 2 hoekpunten op de verticale zijden, maximaal 4 hoekpunten op de horizontale zijden. Dit leidt tot $4 * 4 * 2 * 2 = 64$ configuraties.

4. METHODOLOGIE

Elke vergelijking in formule 4.2 kan herschreven worden zodanig dat men het overgedetermineerd stelsel makkelijker kan oplossen. Hieronder volgt een illustratie voor x_{min} .

$$\begin{aligned}
x_{min} &= \left(K_{3 \times 3} [R_{3 \times 3} | T_{3 \times 1}] \begin{bmatrix} P_{3 \times 1}^{(1)} \\ 1 \end{bmatrix} \right)_x \\
&= \left(K_{3 \times 3} \begin{bmatrix} I_{3 \times 3} & R_{3 \times 3} \times P_{3 \times 1}^{(1)} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} T_{3 \times 1} \\ 1 \end{bmatrix} \right)_x \\
&= \left(M_{3 \times 4} \begin{bmatrix} T_{3 \times 1} \\ 1 \end{bmatrix} \right)_x \\
&= \begin{bmatrix} M[1, 1 : 4]T_{3 \times 1} + M[1, 4] \\ M[2, 1 : 4]T_{3 \times 1} + M[2, 4] \\ M[3, 1 : 4]T_{3 \times 1} + M[3, 4] \end{bmatrix}_x \\
&= \frac{M[1, 1 : 4]T_{3 \times 1} + M[1, 4]}{M[3, 1 : 4]T_{3 \times 1} + M[3, 4]} \\
&\Rightarrow (M[1, 1 : 4] - x_{min}M[3, 1 : 4])T_{3 \times 3} = M[3, 4]x_{min} - M[1, 4] \\
&\Rightarrow A_{1 \times 3}T_{3 \times 1} = b_{1 \times 1}
\end{aligned} \tag{4.3}$$

In de stap tussen de vierde en vijfde gelijkheid transformeert men een 3D punt naar een 2D punt waarbij de x - en y -coördinaat gedeeld wordt door de z -coördinaat. Door vervolgens $(.)_x$ toepassen behoudt men enkel de x -coördinaat.

Als men formule 4.3 voor x_{max} , y_{min} en y_{max} herhaalt en deze samenvoegt tot een matrix kan men de vergelijking hieronder oplossen naar T :

$$\begin{aligned}
A_{4 \times 3}T_{3 \times 1} &= b_{4 \times 1} \\
\Rightarrow T_{3 \times 1} &= (A^T A)^{-1} A^T b
\end{aligned} \tag{4.4}$$

Het overgedetermineerd stelsel wordt iteratief opgelost door voor de 64 mogelijke configuraties de kleinste-kwadratenmethode toe te passen, zie formule 4.4. De configuratie met de kleinste waarde voor $\|AT - b\|^2$ wordt gekozen als de locatie van de 3DBB.

Loss functies

Het VGG16 netwerk en de FC lagen worden tegelijk geoptimaliseerd met behulp van een *multi-task loss* functie:

$$\mathcal{L} = \alpha \mathcal{L}_{dim} + \beta \mathcal{L}_{bin} + \gamma \mathcal{L}_{orient} \tag{4.5}$$

met \mathcal{L}_{dim} de dimensie *loss*, \mathcal{L}_{bin} de bin classificatie *loss* en \mathcal{L}_{orient} de oriëntatie *loss*. De gewichten krijgen volgende waarden: $\alpha = 0.5$, $\beta = 0.1$ en $\gamma = 1.0$.

Voor het berekenen van \mathcal{L}_{dim} wordt de Huber *loss* gebruikt, zie formule 4.6.

$$\mathcal{L}_{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2}, & |y - \hat{y}| \geq 1 \end{cases} \quad (4.6)$$

\mathcal{L}_{bin} wordt berekend met de cross-entropy *loss*. Deze *loss* functie combineert een *softmax* operatie gevolgd door een *negative log-likelihood* operatie.

$$\mathcal{L}_{cross-entropy}(y, \hat{y}) = - \sum_{i=1}^N y_i \log(\hat{y}_i) \quad (4.7)$$

hierbij is y de *ground truth* en \hat{y} de waarde na de *softmax* operatie.

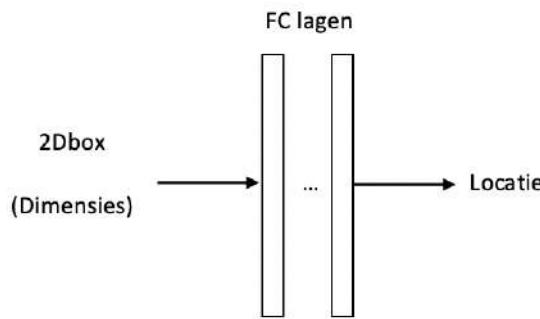
\mathcal{L}_{orient} probeert het verschil tussen de geschatte hoek $\hat{\theta}_{lokaal}$ en de *ground truth* θ_{lokaal} te minimaliseren, zie formule 4.8.

$$\mathcal{L}_{orient} = \cos(\theta_{lokaal} - \theta_{center,bin_i} - \Delta\theta_{lokaal,i}) \quad (4.8)$$

hierbij is θ_{lokaal} de *ground truth* lokale oriëntatie, θ_{center,bin_i} de hoek die zich in het midden van de bin met de grootste *confidence* score c_i bevindt en $\Delta\theta_{lokaal,i}$ de *residual* hoek die door het netwerk ten opzichte van θ_{center,bin_i} wordt voorspeld.

4.2.3 Toevoegingen

Een eerste aanpassing die gemaakt werd ten opzichte van bovenstaande architectuur heeft betrekking tot de locatie voorspelling. Het probleem met de locatie geometrisch te bepalen is dat men een harde *constraint* legt op het feit dat elke 3DBB perfect binnen zijn 2DBB past. Dit is echter niet altijd het geval. Het grote probleem bij het schatten van een 3D locatie is de diepteschatting, of de z-coördinaat van de 3D locatie. De volgende aanpassing is gebaseerd op de paper MonoGRNet [39]. In deze paper genereert men diepteschattingen voor elke 2DBB en kan men daarna de 3D locatie bepalen. Het netwerk om de locatie te bepalen is te zien in figuur 4.3.

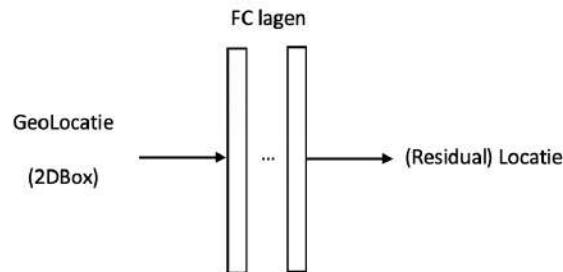


Figuur 4.3: De locatie bepaling via een ANN.

4. METHODOLOGIE

Dit netwerk bestaat uit een aantal FC lagen die de 3D locatie bepaalt aan de hand van een 2DBB en eventueel de *ground truth* 3DBB dimensie. Hoofdstuk 5 behandelt de experimenten met betrekking tot het bepalen van het optimaal aantal FC lagen en het al dan niet gebruiken van 3D dimensie als extra input. Elk van deze FC lagen bestaat uit 20 neuronen met uitzondering van de laatste die er slechts 15 heeft en elke laag gebruikt ReLU als activatiefunctie. De logica achter deze architectuur is vrij intuïtief. Het probeert op een naïeve wijze de geometrische *constraints* na te bootsen.

Een tweede aanpassing die gemaakt werd ten opzichte van de baseline architectuur heeft ook betrekking tot de locatie voorspelling. Daar waar een geometrische locatie bepaling niet altijd optimaal is kan men dit gebruiken als een initiële gok. Deze initiële gok kan dan vervolgens door een netwerk gefinetuned worden. Dit is exact wat er gebeurt in figuur 4.4. Het netwerk neemt als input de geometrische locatie, bepaald door formule 4.4, en eventueel ook de 2DBB. Het resultaat is een (*residual*) 3D locatie. In het geval van de *residual* 3D locatie benadert men niet de *ground truth* 3D locatie, maar het verschil tussen de *ground truth* en de geometrisch bepaalde 3D locatie. De architecturen in figuur 4.3 en 4.4 worden beide getraind met de Huber *loss*, gedefinieerd in formule 4.6.



Figuur 4.4: Finetunen van geometrische locatie via een ANN.

Hoofdstuk 5 behandelt de experimenten met betrekking tot het berekenen van de (*residual*) locatie en het al dan niet gebruiken van de extra 2DBB informatie.

4.2.4 Finale Architecturen

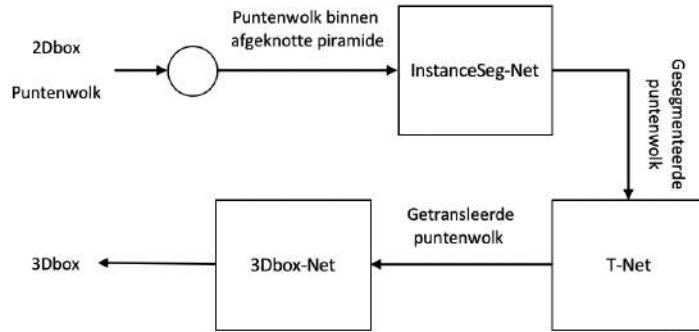
Er worden in totaal drie architecturen gedefinieerd. In hoofdstuk 5 wordt er voor de experimenten naar de volgende namen gerefereerd:

- GeoLoc: 3D dimensie en oriëntatie uit figuur 4.2, 3D locatie geometrisch bepaald met formule 4.4.
- LocNN: 3D dimensie en oriëntatie uit figuur 4.2, 3D locatie uit figuur 4.3.
- GeoLocNN: 3D dimensie en oriëntatie uit figuur 4.2, 3D locatie uit figuur 4.4.

4.3 3D object detector - Lidar gebaseerd

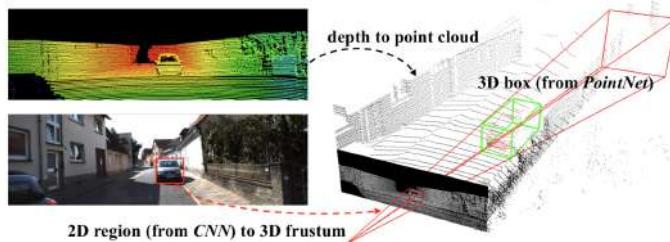
4.3.1 Model architectuur

Het model dat 3DBB's genereert enkel op basis van puntenwolken is gebaseerd op de Frustum-PointNet [36] architectuur. Een overzicht van de architectuur is te vinden in figuur 4.5.



Figuur 4.5: Schematisch overzicht van de Lidar-architectuur.

De architectuur neemt 2DBB's en puntenwolken mee als input¹. Op basis van deze twee inputs en de camera intrinsieke matrix kan men een afgeknotte piramide genereren waarin zich puntenwolken bevinden die mogelijk tot het te detecteren object kunnen behoren. Het resultaat is een puntenwolk binnen een afgeknotte piramide, afgebakend door de rode lijnen in figuur 4.6. Vervolgens gaat het InstanceSeg-Net voor elk punt bepalen of het al dan niet tot het object behoort. Zo houdt men enkel punten over die tot het object horen. Dan wordt deze gesegmenteerde puntenwolk meegegeven aan het T-Net. Het T-Net gaat de *residual* 3D locatie (x, y, z) ten opzichte van het zwaartepunt van de gesegmenteerde puntenwolk voorspellen. Ten slotte berekent het 3Dbox-Net de zeven parameters (x, y, z, h, w, l, θ) die de 3DBB box karakteriseert. De volgende secties bespreken deze architectuur meer in detail.



Figuur 4.6: Het aanmaken van een afgeknotte piramide regio. Bron: [36].

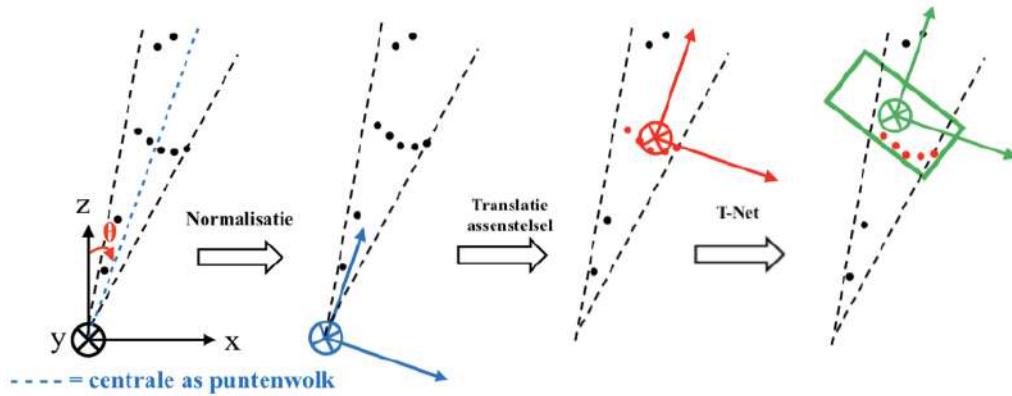
¹Merk hierbij op dat, hoewel een 2DBB meegegeven wordt, de 3DBB enkel met lidar features wordt bepaald.

Puntenwolken binnen afgeknotte piramide

Voor men de puntenwolken binnen de afgeknotte piramide kan bepalen wordt elk punt $p_i^{3D} = (x_i, y_i, z_i, r_i)$ van de lidar input geprojecteerd op de afbeelding en verkrijgt men het corresponderende 2D coördinaat $p_i^{2D} = (u_i, v_i)$. Op basis van een 2DBB ($u_{min}, u_{max}, v_{min}, v_{max}$) kan men met de volgende formule alle puntenwolken behouden die binnen deze 2DBB vallen:

$$\begin{cases} u_{min} \leq u_i \leq u_{max} \\ v_{min} \leq v_i \leq v_{max} \end{cases} \quad (4.9)$$

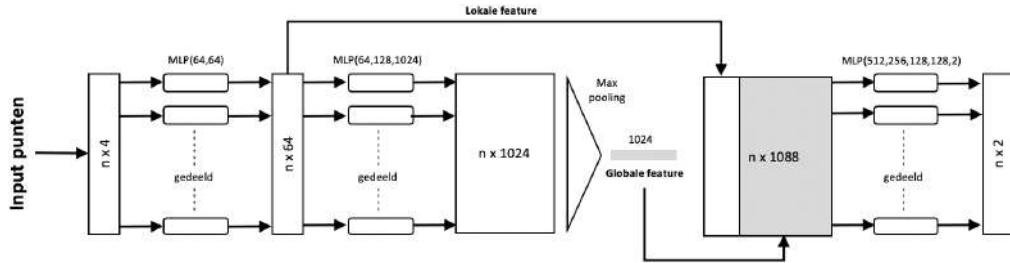
Om ervoor te zorgen dat het netwerk altijd hetzelfde aantal punten meekrijgt, worden er 1024 punten willekeurig gekozen. Als er zich te weinig punten binnen de afgeknotte piramide bevinden, worden een aantal punten geduplicateerd zodanig dat men altijd 1024 punten kan meegeven aan de volgende subnetwerken. Voordat men de puntenwolken binnen de afgeknotte piramide meegeeft aan de volgende subnetwerken worden deze eerst nog genormaliseerd. Het camera assenstelsel wordt zodanig geroteerd dat de centrale as van de puntenwolk orthogonaal staat ten opzichte van het camera assenstelsel. Men gaat van het zwarte camera assenstelsel over naar het blauwe assenstelsel, wat te zien is in figuur 4.7. De normalisatie komt tot stand door eerst een punt te vinden dat geprojecteerd wordt op het midden van een 2DBB. Echter kunnen enorm veel verschillende punten aan deze eisen voldoen. Door gebruik te maken van de Moore–Penrose inverse (pseudo inverse) van de camera projectiematrix kan men het punt vinden dat leidt tot de kleinste-kwadraten oplossing. De hoek θ die de centrale as maakt in het xz -cameravlak kan vervolgens gebruiken om de normalisatie door te voeren. Bovendien zorgt de normalisatie voor een verbetering van de rotatie-invariantie, een belangrijke eigenschap van een puntenwolk.



Figuur 4.7: De assenstelsels waarin een puntenwolk kan gedefinieerd worden.

InstanceSeg-Net

Het InstanceSeg-Net neemt de genormaliseerde puntenwolk mee als input en geeft als output twee classificatie scores voor elk punt n . De classificatie scores weerspiegelen de probabilititeit dat een bepaald punt al dan niet tot een object behoort. Het InstanceSeg-Net is gelijkaardig aan het segmentatie netwerk van de PointNet [37] architectuur maar met twee outputs in plaats van m outputs, zie figuur 2.16. Een overzicht van de InstanceSeg-Net architectuur is te zien in figuur 4.8.



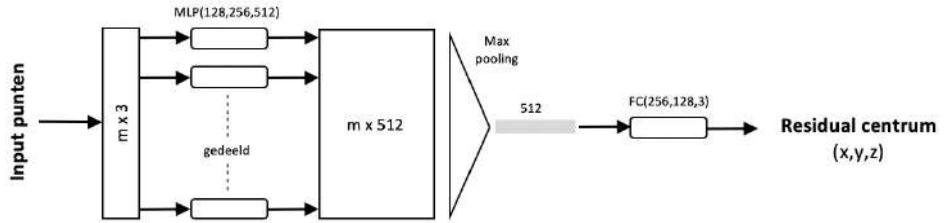
Figuur 4.8: Schematisch overzicht van het InstanceSeg-Net.

Voor men elk punt meegeeft aan het InstanceSeg-Net wordt elk punt nogmaals genormaliseerd door de hele puntenwolk te verminderen met zijn zwaartepunt. Deze bewerking komt overeen met de translatie van het blauwe naar het rode assenstelsel, zie figuur 4.7. Eerst wordt elk punt $p_i = (x_i, y_i, z_i, r_i)$ meegegeven aan een gedeeld MLP die bestaat uit twee lagen met ReLU activatie. Het resultaat is een lokale feature $f_i^{lokaal} \in \mathbb{R}^{64}$. Vervolgens wordt deze lokale feature meegegeven aan een tweede gedeelde MLP dat bestaat uit drie lagen met ReLU activatie. Het resultaat is een feature $f_i \in \mathbb{R}^{1024}$. *Max pooling* wordt toegepast op f_1, \dots, f_n en men krijgt zo een globale feature $f^{globaal} \in \mathbb{R}^{1024}$. Om een betere segmentatie te verkrijgen wordt in een volgende stap de lokale feature f_i^{lokaal} gecombineerd met de globale feature $f^{globaal}$ en vormt het een nieuwe feature $g_i \in \mathbb{R}^{1088}$. Tot slot wordt g_i meegegeven aan een laatste gedeelde MLP met drie lagen. De eerste twee lagen bevatten een ReLU activatie en de laatste laag een *softmax* functie zodanig dat men twee classificatie scores krijgt voor elk punt.

T-Net

Nadat het InstanceSeg-Net heeft bepaald welke punten tot een object horen, worden deze punten meegegeven aan het T-Net. De punten die overblijven zijn in figuur 4.7 rood gekleurd. Vervolgens gaat het T-Net een *residual* centrum (x, y, z) berekenen ten opzichte van het zwaartepunt van de rode puntenwolk. Merk op dat elk punt dat meegegeven wordt aan het T-Net driedimensionaal is, men geeft de reflectiewaarde r_i niet meer mee. Het T-Net is gelijkaardig aan het classificatie netwerk van de PointNet [37] architectuur, zie figuur 2.16. Een overzicht van de T-Net architectuur is te zien in figuur 4.9.

4. METHODOLOGIE

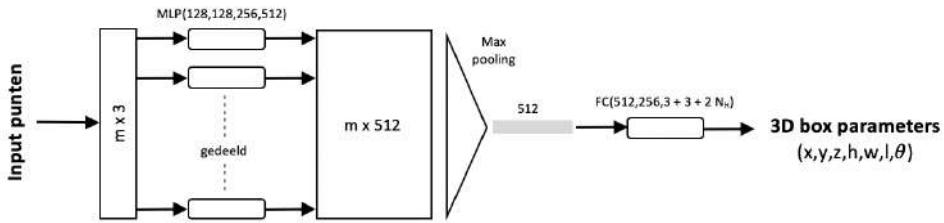


Figuur 4.9: Schematisch overzicht van het T-Net.

Elk punt wordt meegegeven aan een gedeeld MLP dat bestaat uit drie lagen met ReLU activatie. Het resultaat is een feature $f_i \in \mathbb{R}^{512}$. Vervolgens wordt er *max pooling* toegepast op f_1, \dots, f_m en verkrijgt men een globale feature $f_{globaal} \in \mathbb{R}^{512}$. Tot slot wordt $f_{globaal}$ meegegeven aan drie FC lagen waarvan de eerste twee lagen een ReLU activatie bevatten. Dit netwerk geeft als output het *residual centrum*.

3Dbox-Net

Voor men elk punt meegeeft aan het 3Dbox-Net gaat men van elk punt het *residual centrum*, berekend door het T-Net, aftrekken. Deze bewerking komt overeen met de translatie van het rode naar het groene assenstelsel, zie figuur 4.7. Vervolgens gaat het 3Dbox-Net de zeven parameters $(x, y, z, h, w, l, \theta)$ van een 3DBB bepalen. Merk op dat hier ook de reflectiewaarde r_i niet meer wordt meegegeven. Qua architectuur is het 3Dbox-Net heel gelijkaardig aan dat van het T-Net, met wat verschillen in de MLP en FC lagen. Een overzicht van de 3Dbox-Net architectuur is te zien in figuur 4.10.



Figuur 4.10: Schematisch overzicht van het 3DBox-Net.

Zoals men kan zien genereert het 3Dbox-net een output van dimensie $(3 + 3 + 2 * N_H)$. De eerste 3 waarden zijn de *residuals* ten opzichte van het centrum (x, y, z) bepaald door T-Net, de volgende 3 waarden zijn de *residuals* ten opzicht van de gemiddelde dimensie van een auto en de laatste $2 * N_H$ waarden worden gebruikt om de hoek $\theta_{globaal}$ te bepalen. De N_H waarde verwijst naar het aantal *bins* dat men gebruikt om een hoek in op te splitsen. Aangezien men in deze architectuur niet eerst een θ_{lokaal} moet bepalen, kan formule 2.6 direct gebruikt worden om $\theta_{globaal}$ te berekenen.

4.3.2 Implementatie details

Naar analogie met sectie 4.2.2 werd er gekozen voor $N_H = 4$, met $\{0, \frac{\pi}{2} - \pi, -\frac{\pi}{2}\}$ de centrale hoeken van elke bin. Formule 4.1 definieert het domein van de *bins*.

Loss functies

De drie bovenstaande subnetwerken worden tegelijk getraind door gebruik te maken van de volgende *multi-task loss*:

$$\mathcal{L}_{multi-task} = \mathcal{L}_{InstanceSeg-Net} + \alpha(\mathcal{L}_{T-Net} + \mathcal{L}_{3Dbox-center} + \mathcal{L}_{3Dbox-size} + \mathcal{L}_{3Dbox-orient}^{cls} + \mathcal{L}_{3Dbox-orient}^{reg} + \beta\mathcal{L}_{corner}) \quad (4.10)$$

met $\alpha = 1$, $\beta = 10$ en:

- $\mathcal{L}_{InstanceSeg-Net}$ de negatieve log-likelihood loss voor de segmentatie van de puntenwolken bepaald door het InstanceSeg-Net
- \mathcal{L}_{T-Net} de Huber loss voor de regressie van het 3DBB centrum bepaald door het T-Net
- $\mathcal{L}_{3Dbox-center}$ de Huber loss voor de regressie van het 3DBB centrum bepaald door het 3DBox-Net
- $\mathcal{L}_{3Dbox-size}$ de Huber loss voor de regressie van de 3DBB dimensie bepaald door het 3DBox-Net
- $\mathcal{L}_{3Dbox-orient}^{cls}$ de negatieve log-likelihood loss voor de classificatie van de *bins* bepaald door het 3DBox-Net
- $\mathcal{L}_{3Dbox-orient}^{reg}$ de Huber loss voor de residual hoek $\Delta\theta_{globaal}$ bepaald door het 3DBox-Net
- \mathcal{L}_{corner} de Huber loss voor de regressie van de acht hoekpunten van een 3DBB

Er kunnen zich situaties voordoen waar bijvoorbeeld het centrum en de dimensie van een 3DBB goed voorspeld worden maar dat er een grote error zit op de oriëntatie schatting. Hierdoor zal de 3D IoU error met de *ground truth* 3DBB gedomineerd worden door de fout op de oriëntatie. Idealiter moet het 3D centrum, de 3D dimensie en de 3D oriëntatie gezamenlijk geoptimaliseerd worden. Om dit probleem op te lossen wordt een extra *loss* \mathcal{L}_{corner} toegevoegd. Deze *loss* meet het verschil tussen de voorspelde acht hoekpunten en de *ground truth* hoekpunten.

4.3.3 Finale Architectuur

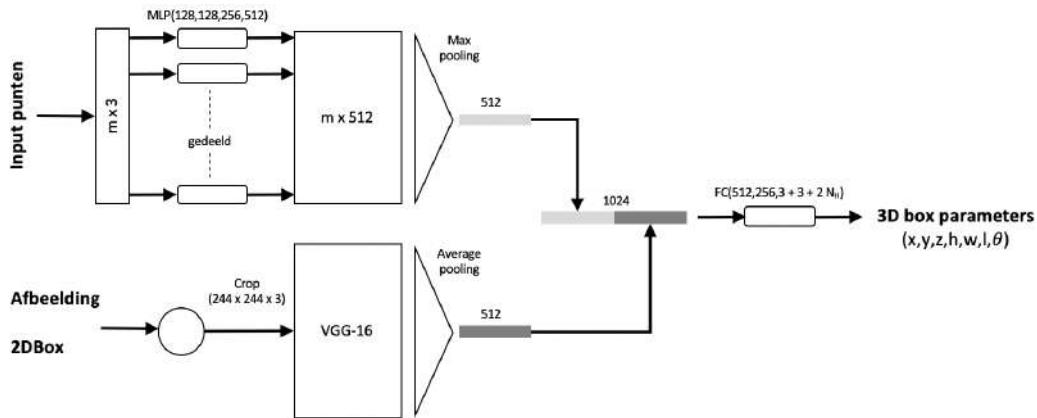
In hoofdstuk 5 wordt er voor de experimenten naar de volgende naam gerefereerd:

- FrustumPointNet: 3D dimensie, 3D locatie en oriëntatie uit figuur 4.5

4.4 3D object detector - Camera en Lidar gebaseerd

4.4.1 Model architectuur

Het model dat 3DBB's genereert op basis van afbeeldingen en puntenwolken is geïnspireerd op wat de paper PointFusion [56] doet. Deze paper was een van de eerste die aantoonde dat de fusie van de features uit de afbeeldingen en de lidar leidde tot een beter model. Als men naar figuur 4.2 en 4.5 kijkt zijn er niet veel mogelijkheden om beide modaliteiten te fusioneren. Vandaar is er gekozen om in de laatste fase, in het 3Dbox-Net, de fusie te maken tussen de afbeelding en lidar features. Dit model behoort dus tot een late fusion architectuur. Het camera en lidar gebaseerde model lijkt sterk op figuur 4.5 met enkel een aanpassing aan het 3Dbox-Net. Deze aangepaste versie van 3Dbox-Net is te zien in figuur 4.11.



Figuur 4.11: Schematisch overzicht van het aangepaste 3DBox-Net.

Men fuseert in dit model de feature $f_{img} \in \mathbb{R}^{512}$ met de feature $f_{lidar} \in \mathbb{R}^{512}$ tot een nieuwe gecombineerde feature $f_{fusie} \in \mathbb{R}^{1024}$. Deze feature f_{fusie} wordt vervolgens aan drie FC lagen meegegeven die de 3DBB parameters bepaald.

4.4.2 Implementatie details

De implementatie details en *loss* functies zijn dezelfde als die vermeld worden in sectie 4.3.2.

4.4.3 Finale Architectuur

In hoofdstuk 5 wordt er voor de experimenten naar de volgende naam gerefereerd:

- FusionNet: 3D dimensie, 3D locatie en oriëntatie uit figuur 4.11

Hoofdstuk 5

Experimenten en resultaten

Dit hoofdstuk bespreekt de experimenten voor de architecturen die besproken werden in hoofdstuk 4. Sectie 5.1 bespreekt de statistieken die gebruikt worden om de nauwkeurigheid van een model te testen. In sectie 5.2 worden de kwantitatieve resultaten besproken. In sectie 5.3 worden de 3DOD modellen die in deze thesis gemaakt zijn vergeleken met de (SOTA) architecturen uit de literatuur. Tot slot laat sectie 5.4 de kwalitatieve resultaten zien.

5.1 Statistieken

In het algemeen genereert elk getraind 3DOD model zeven parameters voor elke 3DBB. In wat volgt zal de schatting van de 3D locatie (x, y, z), de 3D dimensie (h, w, l) en de oriëntatie θ_{global} apart geëvalueerd worden. De locatie error zal bepaald worden door de euclidische afstand, de dimensie error en oriëntatie error door de MAE.

Daarnaast zal men ook de IoU statistiek gebruiken, die algemeen wordt gebruikt om de nauwkeurigheid van een 2DOD of 3DOD model te evalueren. De IoU statistiek meet hoe goed een voorspelde bounding box de *ground truth* bounding box benadert. De IoU formule berekent het quotiënt tussen de doorsnede van de *ground truth* en de voorspelde bounding box en de unie van de *ground truth* en de voorspelde bounding box. De IoU formule om een 2DOD en 3DOD model te evalueren is te zien in formule 5.1 respectievelijk 5.2.

$$IoU_{2D}(x, x*) = \frac{\text{oppervlakte}(x \cap x*)}{\text{oppervlakte}(x \cup x*)} \quad (5.1)$$

$$IoU_{3D}(x, x*) = \frac{\text{volume}(x \cap x*)}{\text{volume}(x \cup x*)} \quad (5.2)$$

met x de voorspelde bounding box en $x*$ de *ground truth* bounding box. Hoe hoger de IoU waarde hoe beter de gedetecteerde 2DBB's of 3DBB's aansluiten bij de werkelijkheid.

5. EXPERIMENTEN EN RESULTATEN

Tijdens het evalueren zijn de begrippen *precision* en *recall* ook van enorm belang. *Precision* is gedefinieerd in formule 5.3 en *recall* in formule 5.4.

$$Precision = \frac{TP}{TP + FP} \quad (5.3)$$

$$Recall = \frac{TP}{TP + FN} \quad (5.4)$$

met TP de *true positives*, FP de *false positives*, FN de *false negatives*.

Precision meet hoe nauwkeurig de voorspellingen zijn, dit wil zeggen het percentage van voorspellingen dat correct is. *Recall* meet hoe goed men alle relevante objecten kan vinden. Een gedetecteerde object wordt beschouwd als een TP wanneer zowel de IoU waarde als de klasprobabiliteit boven een bepaalde drempel ligt. De drempels voor IOU en de geschatte waarschijnlijkheid zullen dus een grote invloed hebben op de *precision* en *recall* waarde.

De KITTI Vision Benchmark Suite [11] heeft allerlei statistieken gedefinieerd waarvan in deze thesis de *average precision* (AP) wordt gebruikt. AP wordt in de literatuur enorm veel gebruikt om de nauwkeurigheid van een 2DOD of 3DOD model te testen. De AP wordt berekend volgens de PASCAL VOC benchmark [10] mits het aanpassen naar 40 recall interpolatiepunten.

De AP wordt berekend door de volgende formule:

$$AP|_R = \frac{1}{|R|} \sum_{r \in R} \rho_{interp}(r) \quad (5.5)$$

met $\rho_{interp}(r) = \max_{r': r' \geq r} \rho(r')$ en $\rho(r)$ de *precision* op *recall* waarde r , $r \in R_{40} = \{1/40, 2/40, \dots, 1\}$.

Om de nauwkeurigheid van de geïmplementeerde 3DOD modellen te evalueren, gaat men de *3D-average precision* (3D-AP) gebruiken. Door de klasse onbalans in de KITTI dataset focussen we in wat volgt enkel op de detectie van auto's. KITTI definieert voor auto's een 3D-IoU drempel van 0.7. Daarnaast gebruikt men ook de vogelperspectief *average precision* (BEV-AP) waarbij de IoU in vogelperspectief wordt berekend. Het gebruik van de 3D-AP en de BEV-AP is in de literatuur de standaard geworden om een 3DOD model te evalueren.

De KITTI Vision Benchmark Suite bevat een *development kit* met C++ code die zowel de 3D-AP als de BEV-AP kan berekenen, opgesplitst voor de drie moeilijkheidsgraden.

5.2 Kwantitatieve experimenten

Voor het trainen en testen van alle netwerken is er een privé computer gebruikt met volgende eigenschappen:

- AMD 6-core CPU, NVIDIA GeForce RTX 2070 SUPER GPU, 32GB RAM

5.2.1 2D detector

Zoals vermeld in sectie 4.1 wordt er een Faster-RCNN netwerk gebruikt om 2DBB's te genereren. Wanneer men de nauwkeurigheid van een 3DOD model op de validatie set wil evalueren, moet men gebruik maken van deze 2DBB's. De nauwkeurigheid van het Faster-RCNN netwerk op de KITTI validatie set is te zien in tabel 5.1.

| 2D detector | Easy | Moderate | Hard |
|-------------|--------|----------|--------|
| Faster-RCNN | 92.50% | 92.36% | 90.59% |

Tabel 5.1: De 2D AP voor auto's op de KITTI validatie set.

Ondanks de hoge 2D AP behoort dit 2DOD model niet tot de SOTA architectuur voor de KITTI 2D-benchmark. Aangezien de focus van deze thesis niet ligt op het maken van een SOTA 2D detector is er toch gekozen om dit model te gebruiken voor de 3D detectors. Dit model is een pretrained Faster-RCNN gefinetuned op KITTI van de Tensorflow Object Detection API [16].

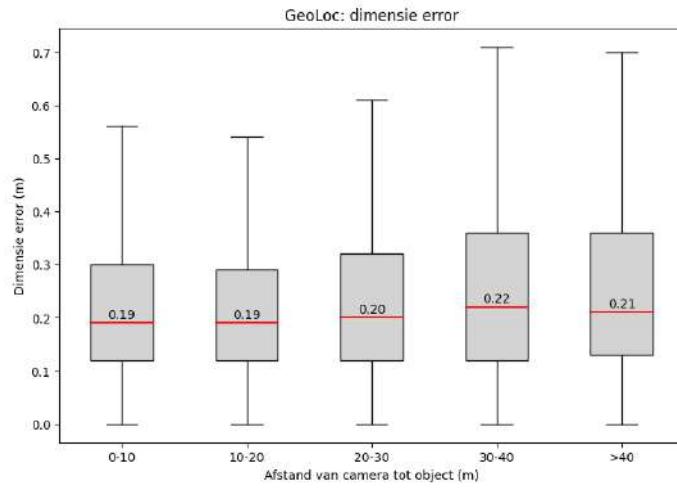
5.2.2 3D object detector - Camera gebaseerd

GeoLoc

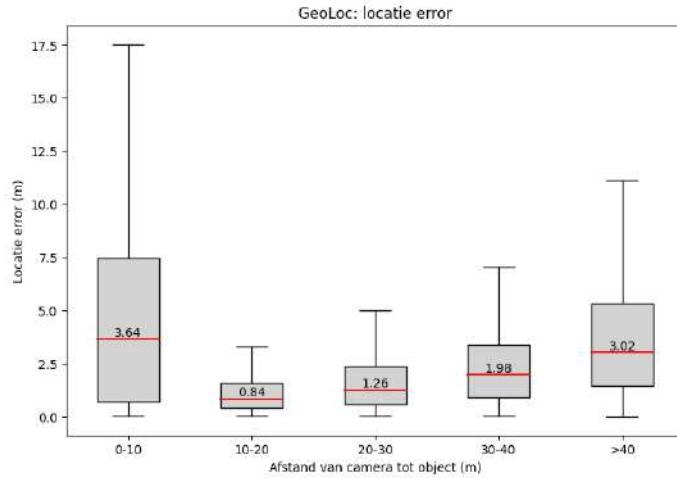
Tijdens het trainen van het GeoLoc netwerk is er gevarieerd met de keuze van de hyperparameters en de gewichten voor de *loss* functies in formule 4.5. Meer bepaald is de learning rate gevarieerd tussen 10^{-2} en 10^{-5} , de gewichten van de *loss* functie tussen 0.1 en 1.0 en is er getest met Adam en SGB als *optimizer*. Uiteindelijk is er gekozen voor de setting in tabel A.1. Deze setting leidde tot de beste kwantitatieve resultaten. Het beste model werd op basis van de validation *loss* gekozen. Figuur 5.1, 5.2 en 5.3 toont boxplots voor de drie belangrijke errors. Per figuur werden er vijf intervallen gedefinieerd op basis van de afstand van een object tot de camera. Verder is elke boxplot geannoteerd door zijn mediaan en bijhorende waarde.

In figuur 5.1 ziet men dat de dimensie error in functie van de afstand nagenoeg constant blijft. Dit is geen verrassing aangezien bij het berekenen van de 3D dimensie men een crop van een afbeelding meegeeft. Hierdoor elimineert men de afstand tot het object en lijkt elk object voor het netwerk even ver te zijn. Bijgevolg is er geen correlatie tussen de dimensie error en de afstand tot een object.

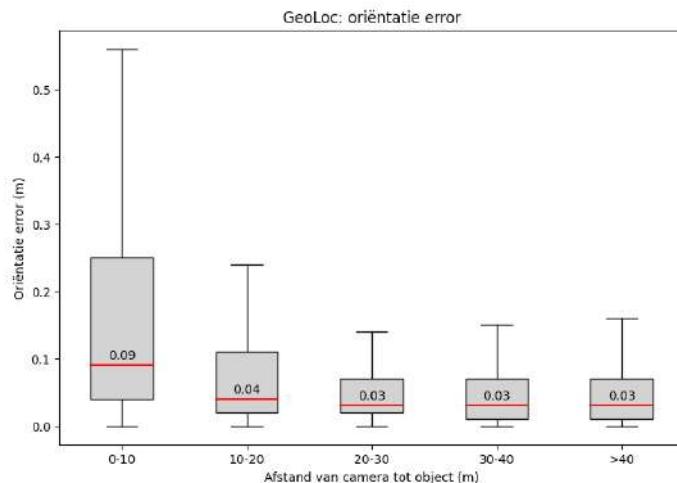
5. EXPERIMENTEN EN RESULTATEN



Figuur 5.1: GeoLoc: de 3D dimensie error in functie van de afstand tot het object.



Figuur 5.2: GeoLoc: de 3D locatie error in functie van de afstand tot het object.



Figuur 5.3: GeoLoc: de oriëntatie error in functie van de afstand tot het object.

Wat betreft de 3D locatie error zou men intuïtief kunnen verwachten dat de error lineair toeneemt met de afstand. Als men figuur 5.2 bekijkt, ziet men enkel lineariteit voor objecten die verder liggen dan 10 meter. Het zijn de objecten die het dichtst bij de camera liggen die de grootste fout ondervinden. Dit fenomeen is te danken aan het *truncation* niveau van een object, dat aangeeft in welke mate een object onzichtbaar is op de afbeelding. Objecten die dicht bij de camera staan gaan vaak gepaard met een hoog *truncation* niveau, en zijn bijgevolg grotendeels onzichtbaar in een afbeelding. De bottleneck van een 2D detector, voor *truncated* objecten, is dat het niet in staat is om een 2DBB te tekenen rond het hele object. Als slechts een deel van een object wordt meegegeven aan het GeoLoc netwerk is het dan ook niet onlogisch dat hiervoor de grootste fout wordt verwacht.

Tot slot laat figuur 5.3 de oriëntatie error zien. Voor de oriëntatie bepaling wordt er ook gewerkt met een crop van een afbeelding. Men zou dus verwachten dat de error nagenoeg constant blijft door de eliminatie van de afstand. Echter valt er op dat enkel de objecten die dicht bij de camera staan afwijken van deze constante. Zoals bij de 3D locatie error zal de oriëntatie bepaling moeilijkheden ondervinden door het hoge *truncation* niveau van nabij gelegen objecten.

LocNN

Daar waar de locatie geometrisch bepaald wordt in het GeoLoc netwerk, wordt de locatie in het LocNN netwerk bepaald door een neuraal netwerk. De dimensie en oriëntatie bepaling blijven dezelfde als in het GeoLoc netwerk. Tijdens het trainen van het LocNN netwerk is er gevarieerd met de keuze van de hyperparameters. De learning rate werd gevarieerd tussen 10^{-2} en 10^{-5} en er werd getest met een Adam en SGD *optimizer*. Uiteindelijk is er gekozen voor de setting in tabel A.2. Deze setting leidde tot de beste kwantitatieve resultaten.

Het LocNN tracht de geometrische locatiebepaling te vervangen door een neuraal netwerk. In de zoektocht naar de beste configuratie is er met twee parameters gevarieerd:

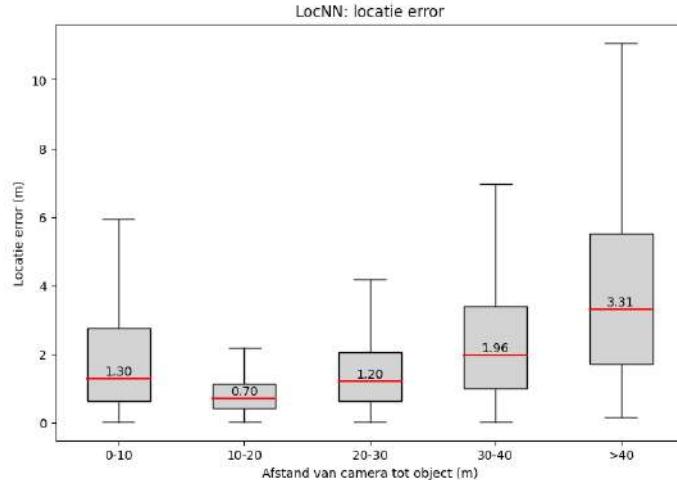
- Het aantal FC lagen: 2 tot 7 lagen
- De input: een 2DBB en eventueel een 3D dimensie

Elke configuratie is getraind voor 100 *epochs*. De resultaten van de verschillende configuraties zijn te vinden in tabel A.3. Deze tabel toont de gemiddelde euclidische afstand tussen de *ground truth* en de voorspelde 3D locatie, zowel absoluut als per dimensie. Elk vinkje representeert de input die wordt meegegeven aan het netwerk. Er valt op te merken dat de inputgegevens tijdens het trainen van het LocNN model de *ground truth* 3D dimensies en 2DBB's waren. Tijdens het valideren werkt men met voorspelde dimensies van GeoLoc en de voorspelde 2DBB's van Faster-RCNN. Hierdoor verwacht men een grotere fout op de 3D locatie.

5. EXPERIMENTEN EN RESULTATEN

Naarmate het aantal lagen toeneemt en de complexiteit van het model groeit, kan men zien dat de 3D locatie schatting verbetert. Deze grote verbeteringen lijken te verminderen rond de 6 of 7 FC lagen. Uit tabel A.3 kan men een duidelijke conclusie trekken: het toevoegen van 3DBB dimensies leidt tot een betere 3D locatiebepaling. Door het gebrek aan 3D informatie in een afbeelding, kan men waarnemen dat de grootste fout in de diepteschatting $ZLOC_{error}$ zit. Deze is gemiddeld gezien dubbel zo groot dan de som van de $XLOC_{error}$ en de $YLOC_{error}$. De configuratie met de kleinste error is die met 6 FC lagen en 3DBB dimensies als extra input. Dit model wordt gebruikt om te evalueren op de KITTI validatie set.

Voor het LocNN netwerk zijn de dimensie en oriëntatie error dezelfde als GeoLoc. Figuur 5.4 toont de 3D locatie error in functie van de afstand tot een object voor het LocNN netwerk. Analoog aan figuur 5.2 ziet men voor nabije objecten een grote error die vervolgens afneemt en dan lineair stijgt met de afstand. De grootste verbetering ten opzicht van de locatiebepaling in GeoLoc is te zien voor objecten dichtbij de camera. Voor die objecten ziet men de mediaan en de interkwartielafstand drastisch dalen, met een factor van ongeveer 3. De reden dat het LocNN netwerk de 3D locatie beter kan voorspellen is omwille van het feit dat de locatie met een neuraal netwerk berekend wordt in plaats van geometrisch. Op de categorie met de verste objecten na, presteert het LocNN netwerk in het algemeen beter dan het GeoLoc netwerk.



Figuur 5.4: LocNN: de 3D locatie error in functie van de afstand tot het object.

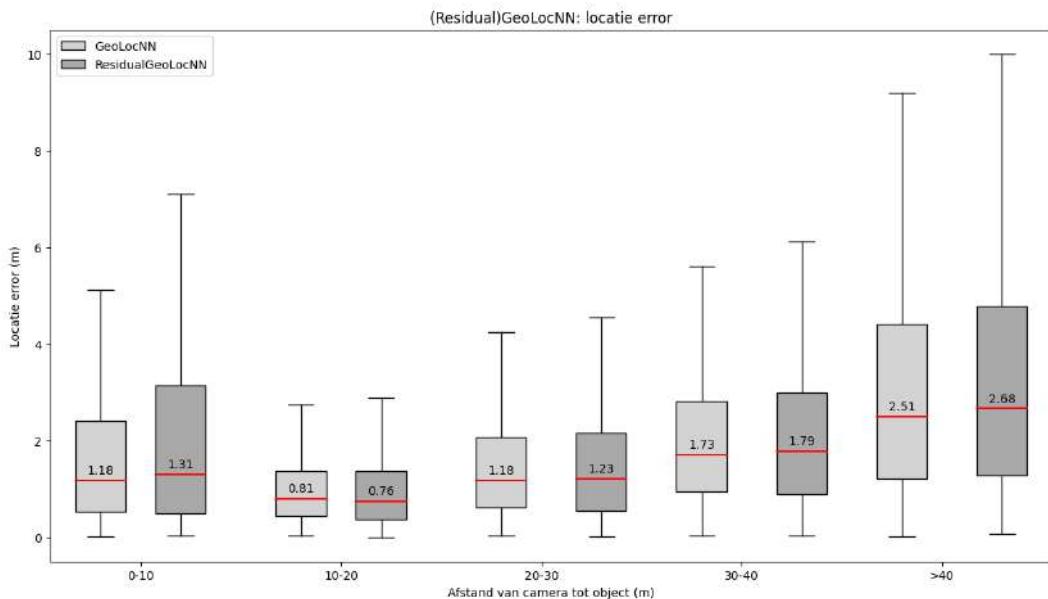
GeoLocNN

Het GeoLocNN tracht de locatie die geometrisch bepaald is door GeoLoc nog verder te finetunen door een neuraal netwerk. Men gebruikt dus de geometrisch geschatte dimensie als initiële schatting. De dimensie en oriëntatie bepaling blijft hetzelfde als in het GeoLoc netwerk.

Aangezien de GeoLocNN architectuur sterk lijkt op de LocNN architectuur zijn hiervan de hyperparameters overgenomen, zie tabel A.4. Analoog aan LocNN is er ook hier gezocht naar de beste configuratie door met drie variabelen te variëren:

- Het aantal FC lagen: 2 tot 7 lagen
- De input: geometrische locatie en eventueel een 2DBB
- De output: locatie vs *residual* locatie

Elke configuratie is getraind voor 100 *epochs*. De resultaten van de verschillende configuraties zijn te vinden in tabel A.5. In deze tabel kan men, zoals in LocNN, duidelijk zien dat het toevoegen van een extra input leidt tot een kleinere euclidische afstand tussen de *ground truth* en voorspelde 3D locatie. Uiteindelijk werd de configuratie met 4 FC lagen, de geometrische locatie en de 2DBB als input gekozen als beste model aangezien deze de laagste 3DLOC_{error} heeft.



Figuur 5.5: GeoLocNN: de 3D locatie error in functie van de afstand tot het object.

Voor het GeoLocNN netwerk zijn de dimensie en oriëntatie error dezelfde als GeoLoc. Figuur 5.5 toont de 3D locatie error in functie van de afstand tot een object voor het GeoLocNN en ResidualGeoLocNN netwerk. Het ResidualGeoLocNN netwerk is qua architectuur dezelfde als het GeoLocNN netwerk maar men benadert een *residual* locatie in plaats van de absolute locatie. Op basis van figuur 5.5 lijkt de *residual* benadering tot slechtere resultaten te leiden. Vreemd genoeg leidt het ResidualGeoLocNN netwerk wel tot een veel betere 3D-AP en BEV-AP statistiek in vergelijking met GeoLocNN, zie tabel 5.2.

5. EXPERIMENTEN EN RESULTATEN

Als men figuur 5.2, 5.4, 5.5 vergelijkt kan men het volgende concluderen:

- Het gebruik van een neurale netwerk om de 3D locatie te benaderen is meer performant in vergelijking met de geometrische locatiebepaling
- De geometrische locatie als initiële schatter voor het neurale netwerk zorgt voor betere resultaten

5.2.3 3D object detector - Lidar gebaseerd

FrustumPointNet

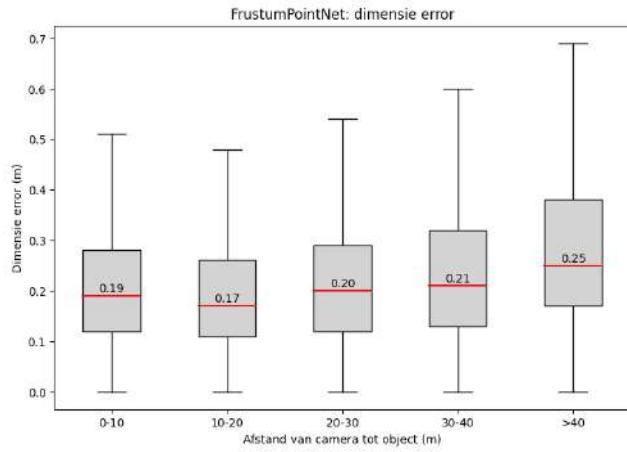
Tijdens het trainen van het FrustumPointNet netwerk is er gevarieerd met de keuze van de hyperparameters en de gewichten voor de *loss* functies in formule 4.10. Uiteindelijk is er gekozen voor de setting in tabel A.6. Figuur 5.6 toont de 3D dimensie error, figuur 5.7 de 3D locatie error en figuur 5.8 de oriëntatie error voor het FrustumPointNet.

Uit deze drie figuren kan men afleiden dat een lidar moeilijkheden ondervindt met het detecteren van verre objecten. Dit valt te verklaren door het feit dat verre objecten gekarakteriseerd worden door een laag aantal punten. Het is namelijk zo dat de hoeveelheid laserpulsen die terug worden gereflecteerd afneemt met de afstand van het object.

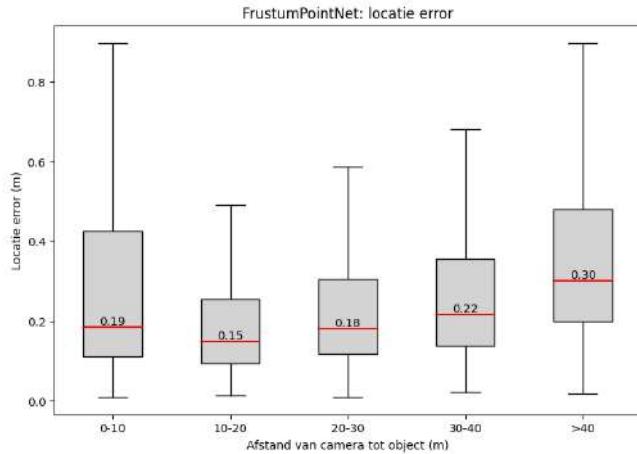
Als men de errors van het FrustumPointNet vergelijkt met de errors van GeoLoc, LocNN en GeoLocNN kan men het volgende concluderen:

- De 3D dimensie error voor de camera en de lidar is van dezelfde grootteorde
- De 3D locatie error voor de lidar is enorm veel kleiner in vergelijking met de camera, meer bepaald 5 tot 10 keer zo klein
- De oriëntatie error voor de camera en de lidar is van dezelfde grootteorde

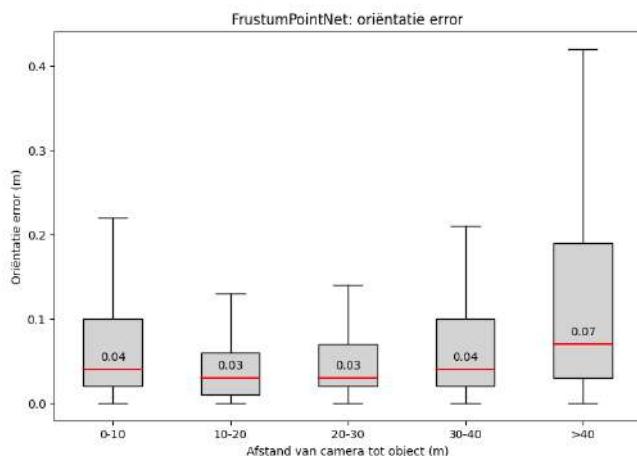
Het grote verschil voor de 3D locatie error tussen een camera versus lidar gebaseerde methode is eenvoudig te verklaren. Een lidar ziet de ruimte namelijk in 3D en een camera in 2D. Echter is de locatiebepaling net het cruciale en tevens moeilijkste deel van het hele 3D detectie probleem. De voordelen van het gebruik van een lidar worden hiermee dus beantwoord.



Figuur 5.6: FrustumPointNet: de 3D dimensie error in functie van de afstand tot het object.



Figuur 5.7: FrustumPointNet: de 3D locatie error in functie van de afstand tot het object.



Figuur 5.8: FrustumPointNet: de oriëntatie error in functie van de afstand tot het object.

5.2.4 3D object detector - Camera en Lidar gebaseerd

FusionNet

Voor het FusionNet netwerk zijn dezelfde hyperparameters, op de *batch* grootte na, genomen als deze voor het FrustumPointNet, zie tabel A.7.

In de zoektocht naar de beste configuratie voor de FusionNet architectuur is er gevarieerd met het aantal lidar en afbeelding features die in het 3Bbox-Net worden samengevoegd. Figuur 5.9, 5.10 en 5.11 tonen de errors met betrekking tot de verschillende FusionNet architecturen. Elke figuur toont drie configuraties, die de volgende eigenschappen hebben:

- FusionNet: $f_{img} \in \mathbb{R}^{512}$ en $f_{lidar} \in \mathbb{R}^{512}$
- FusionNet-moreImage: $f_{img} \in \mathbb{R}^{1024}$ en $f_{lidar} \in \mathbb{R}^{512}$
- FusionNet-moreLidar: $f_{img} \in \mathbb{R}^{512}$ en $f_{lidar} \in \mathbb{R}^{1024}$

Als men de resultaten van deze configuraties bekijkt kan men twee conclusies trekken.

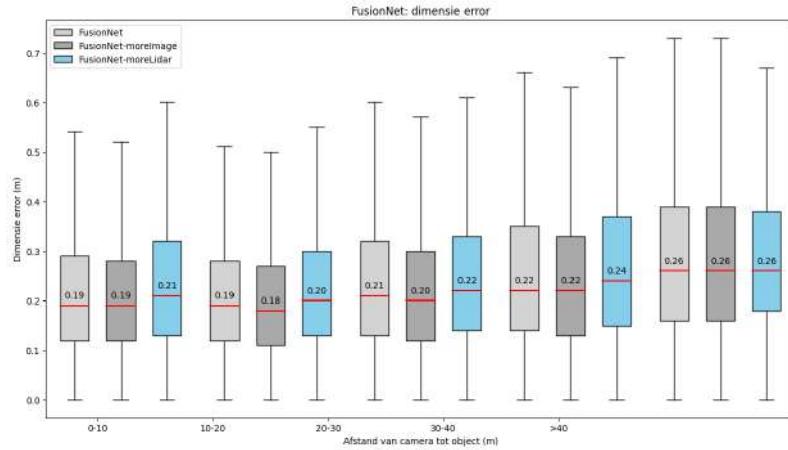
1. Men ziet dat er geen significante verbetering is wanneer men de camera- en lidar features combineert. Architecturen zoals [56] [7] [24], die er in geslaagd zijn om betere resultaten te bekomen door camera- en lidar features te combineren, zijn ontworpen met het oog op de combinatie van deze twee modaliteiten. Het is dus niet vanzelfsprekend hoe men FusionNet, die twee bestaande architecturen tracht te combineren, moet designen om betere resultaten te bekomen.

2. Men kan op basis van de boxplots een conclusie trekken wat betreft het percentage camera- en lidar features. In deze setting neemt de performantie van het FusionNet af met een stijgend percentage lidar features, zoals FusionNet-moreLidar toont. Verrassend genoeg presteert het FusionNet-moreImage netwerk, dat een groter percentage camera features bevat, het beste. Dat meer lidar features toevoegen leidt tot slechtere resultaten kan liggen aan het feit dat de oorspronkelijke lidar feature map van dimensie 512 is en dat men een extra laag in de architectuur moet plaatsen om te schalen naar 1024 features. De lidar architectuur complexer maken heeft hier dus nadelige gevolgen. Het is veel intuïtiever om de oorspronkelijke camera feature map die van dimensie 25088 is, te verkleinen naar 1024 of 512.

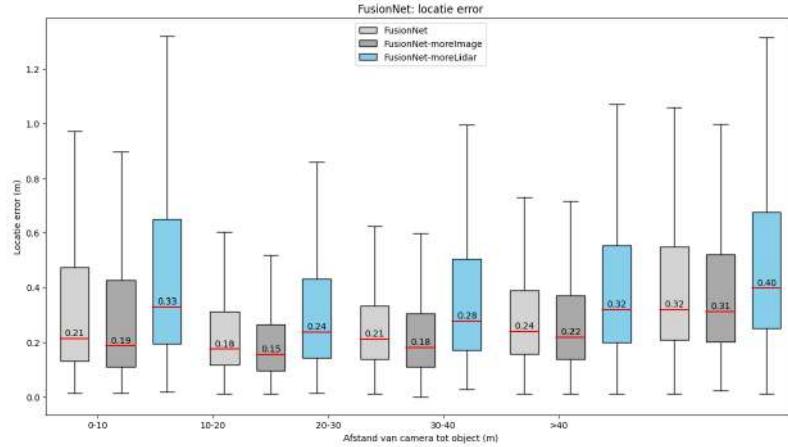
5.3 Benchmarking

Om de 3DOD modellen die in deze thesis gemaakt werden te vergelijken met de literatuur gebruikt men de 3D-AP en BEV-AP statistiek, vermeld in sectie 5.1. Tabel 5.2 toont de 3D-AP en BEV-AP statistiek voor zowel de netwerken geïmplementeerd in deze thesis als gepubliceerde (SOTA) architecturen uit de literatuur. De vetgedrukte blauwe getallen in de tabel tonen het beste eigengemaakte model per modaliteit, de vetgedrukte zwarte cijfers tonen de SOTA architectuur per modaliteit.

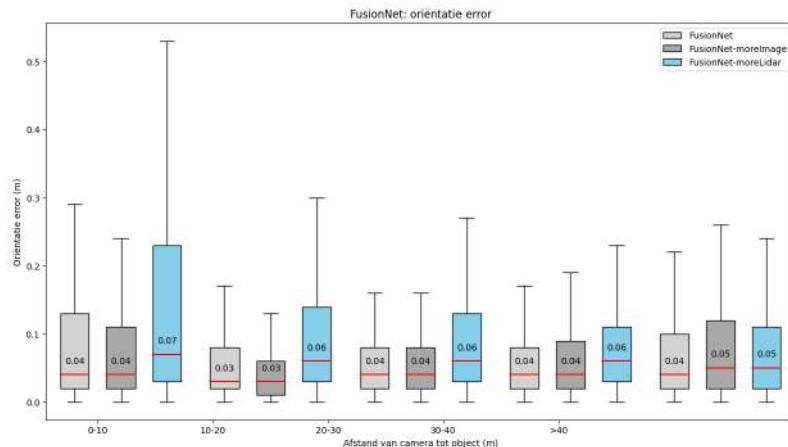
5.3. Benchmarking



Figuur 5.9: FusionNet: de 3D dimensie error in functie van de afstand tot het object.



Figuur 5.10: FusionNet: de 3D locatie error in functie van de afstand tot het object.



Figuur 5.11: FusionNet: de oriëntatie error in functie van de afstand tot het object.

5. EXPERIMENTEN EN RESULTATEN

Als men de eigengemaakte modellen vergelijkt met eerder gepubliceerde architecturen (aangeduid met een cijfer achter hun naam), kan men per modaliteit het volgende concluderen:

- Het eigengemaakte ResidualGeoLocNN netwerk presteert vrij goed in vergelijking met andere camera gebaseerde architecturen.
- Veel gepubliceerde lidar architecturen kennen een serieuze drop in nauwkeurigheid naarmate het object moeilijker te detecteren is. Het FrustumPointNet (thesis) is hier minder gevoelig voor wat leidt tot een competitief voordeel.
- Ook gepubliceerde camera-lidar architecturen kennen een drop in nauwkeurigheid naarmate het object moeilijker te detecteren is. Het FusionNet-moreImage is hier minder gevoelig voor wat leidt tot een competitief voordeel.

| Model | Input data | | 3D-AP (%) | | | BEV-AP (%) | | |
|--------------------------|------------|-------|--------------|--------------|--------------|--------------|--------------|--------------|
| | Camera | Lidar | easy | moderate | hard | easy | moderate | hard |
| Mono3D [6] | ✓ | | 2.53 | 2.31 | 2.31 | 5.22 | 5.19 | 4.13 |
| Deep3DBox [30] | ✓ | | 5.85 | 4.10 | 3.84 | 9.99 | 7.71 | 5.30 |
| OFT-Net [43] | ✓ | | 4.07 | 3.27 | 3.29 | 11.06 | 8.79 | 8.91 |
| MF3D [55] | ✓ | | 10.53 | 5.69 | 5.39 | 22.03 | 13.63 | 11.60 |
| M3D-RPN [2] | ✓ | | 20.27 | 17.06 | 15.21 | 25.94 | 21.18 | 17.90 |
| MonoGRNet [39] | ✓ | | 13.88 | 10.19 | 7.62 | 24.97 | 19.44 | 16.30 |
| RTM3D [23] | ✓ | | 20.77 | 16.86 | 16.63 | 25.56 | 22.12 | 20.91 |
| GeoLoc | ✓ | | 9.86 | 9.80 | 10.88 | 12.78 | 12.70 | 14.09 |
| LocNN | ✓ | | 6.22 | 5.21 | 5.70 | 13.43 | 12.58 | 13.13 |
| GeoLocNN | ✓ | | 7.80 | 6.96 | 6.90 | 15.38 | 14.40 | 14.66 |
| ResidualGeoLocNN | ✓ | | 13.40 | 12.45 | 12.70 | 16.69 | 15.72 | 16.18 |
| FrustumPointNet [36] | | ✓ | 80.62 | 64.70 | 56.07 | 87.28 | 77.09 | 67.90 |
| VoxelNet [59] | | ✓ | 77.47 | 65.11 | 57.73 | 89.35 | 79.26 | 77.39 |
| SECOND [57] | | ✓ | 83.13 | 73.66 | 66.20 | 88.07 | 79.37 | 77.95 |
| PointPillars [22] | | ✓ | 79.05 | 74.99 | 68.30 | 88.35 | 86.10 | 79.83 |
| STD [58] | | ✓ | 86.61 | 77.63 | 76.06 | 89.66 | 87.76 | 86.89 |
| Point-GNN [47] | | ✓ | 88.33 | 79.47 | 72.29 | 93.11 | 89.17 | 83.9 |
| FrustumPointNet (thesis) | | ✓ | 74.00 | 77.98 | 74.3 | 76.42 | 80.09 | 78.19 |
| PointFusion [56] | ✓ | ✓ | 77.92 | 63.00 | 53.27 | / | / | / |
| AVOD [21] | ✓ | ✓ | 81.94 | 71.88 | 66.38 | 88.53 | 83.79 | 77.90 |
| UberATG-MMF [24] | ✓ | ✓ | 87.90 | 77.86 | 75.57 | 96.66 | 88.22 | 79.60 |
| FusionNet | ✓ | ✓ | 70.93 | 73.92 | 70.52 | 74.2 | 78.67 | 74.82 |
| FusionNet-moreLidar | ✓ | ✓ | 59.14 | 64.28 | 60.03 | 64.48 | 71.55 | 68.50 |
| FusionNet-moreImage | ✓ | ✓ | 72.45 | 77.04 | 73.58 | 75.25 | 79.58 | 77.80 |

Tabel 5.2: Evaluatie op de validatie set voor de KITTI 3D en BEV objectdetectiebenchmark (auto's). De vetgedrukte blauwe getallen tonen de beste eigengemaakte modellen, de vetgedrukte zwarte cijfers tonen de SOTA architecturen.

5.4 Kwalitatieve experimenten

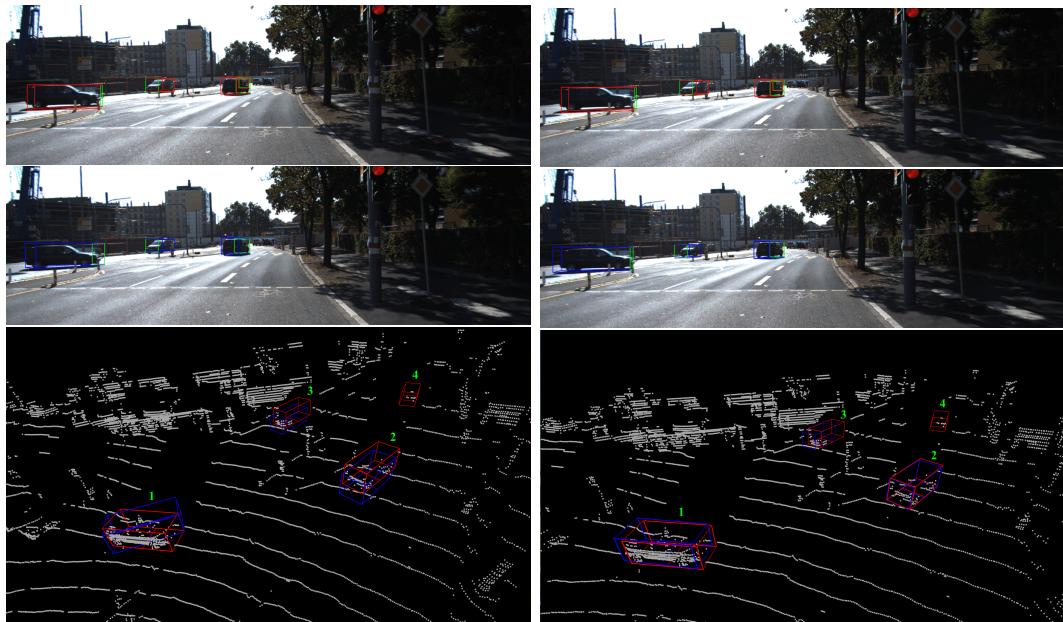
In deze sectie vergelijkt men de kwalitatieve resultaten van het ResidualGeoLocNN en FrustumPointNet. Uit de kwantitatieve resultaten blijkt dat het FusionNet niet beter presteert dan het FrustumPointNet dus wordt daarom achterwege gelaten.

5.4.1 Foutenanalyse

Voor zowel het ResidualGeoLocNN en FrustumPointNet zijn er situaties waarin het model fouten maakt. In wat volgt zal men telkens de resultaten van de twee modellen naast elkaar zetten en vergelijken. Links zullen de resultaten van het ResidualGeoLocNN netwerk te zien zijn, rechts die van het FrustumPointNet. Voor elk voorbeeld zijn er drie foto's te zien, in volgorde: een foto met geannoteerde *ground truth* 3DBB's (rood), een foto met geannoteerde voorspelde 3DBB's (blauw) en beide 3DBB's gevisualiseerd in 3D met behulp van de puntenwolken.

Weinig punten binnen een object

Objecten die ver weg liggen zijn gekarakteriseerd door een laag aantal punten, wat het moeilijker maakt om het object te detecteren. Zoals te zien in figuur 5.12 wordt auto nummer 4 niet gedetecteerd door het FrustumPointNet en ResidualGeoLocNN.

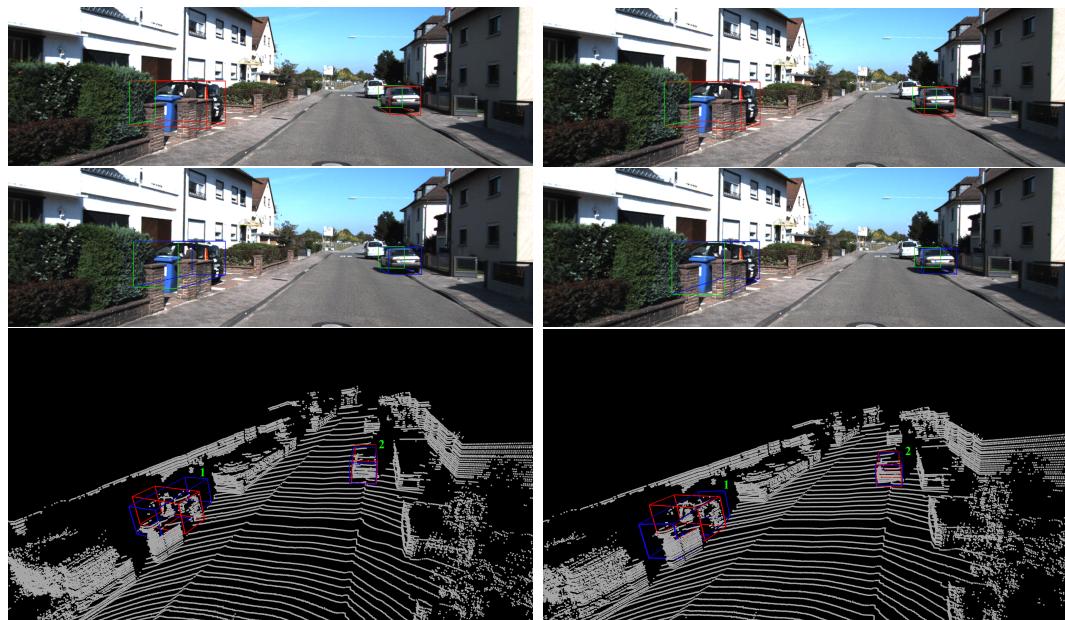


Figuur 5.12: Visualisatie van voorbeeld 6 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 4 weinig punten bevat en dat noch het FrustumPointNet noch het ResidualGeoLocNN dit object detecteert.

5. EXPERIMENTEN EN RESULTATEN

Occlusie van een object

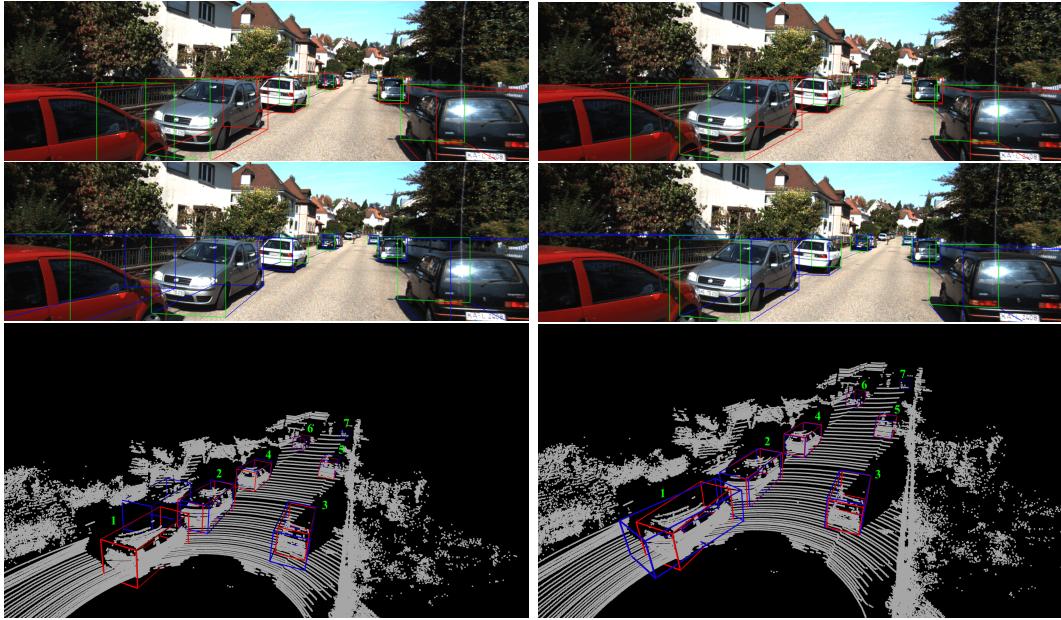
Een andere veel voorkomende fout is dat een object door andere objecten wordt geoccludeerd. Zoals te zien in figuur 5.13, wordt auto 1 achter de muur heel slecht gedetecteerd door zowel het ResidualGeoLocNN als het FrustumPointNet. Voor het ResidualGeoLocNN is een occlusie nadelig aangezien de kans groot is dat een 2D detector het object niet/slecht detecteert, wat leidt tot een foute 3DBB. Als men kijkt naar de puntenwolken lijkt het dat de punten van de muur deel uitmaken van de auto. Bijgevolg segmenteert en classificeert het FrustumPointNet de punten fout wat leidt tot een foute voorspelling.



Figuur 5.13: Visualisatie van voorbeeld 63 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 1 enorm hard wordt geoccludeerd door een muur wat leidt tot een slechte detectie.

Object deels buiten een afbeelding

Een andere veel voorkomende fout is dat een object maar voor een deel op de afbeelding staat. Zoals te zien in figuur 5.14, ligt auto 1 deels buiten de afbeelding. Bijgevolg detecteert een 2D detector slechts een deel van het object waardoor de 3D detectie compleet misloopt. Het gebruik van een lidar sensor toont hier zijn voordeelen. De lidar kan het object helemaal vastleggen waardoor het FrustumPointNet een redelijke voorspelling kan maken. Alhoewel auto 7 zich op ongeveer 51m van de camera/lidar bevindt, kunnen zowel het ResidualGeoLocNN en FrustumPointNet deze auto detecteren. Merk op dat deze auto niet geannoteerd was in de *ground truth*.



Figuur 5.14: Visualisatie van voorbeeld 8 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 1 deels onzichtbaar is op de afbeelding, wat leidt tot een slechte detectie voor het ResidualGeoLocNN.

Hoge concentratie aan objecten van dezelfde klasse

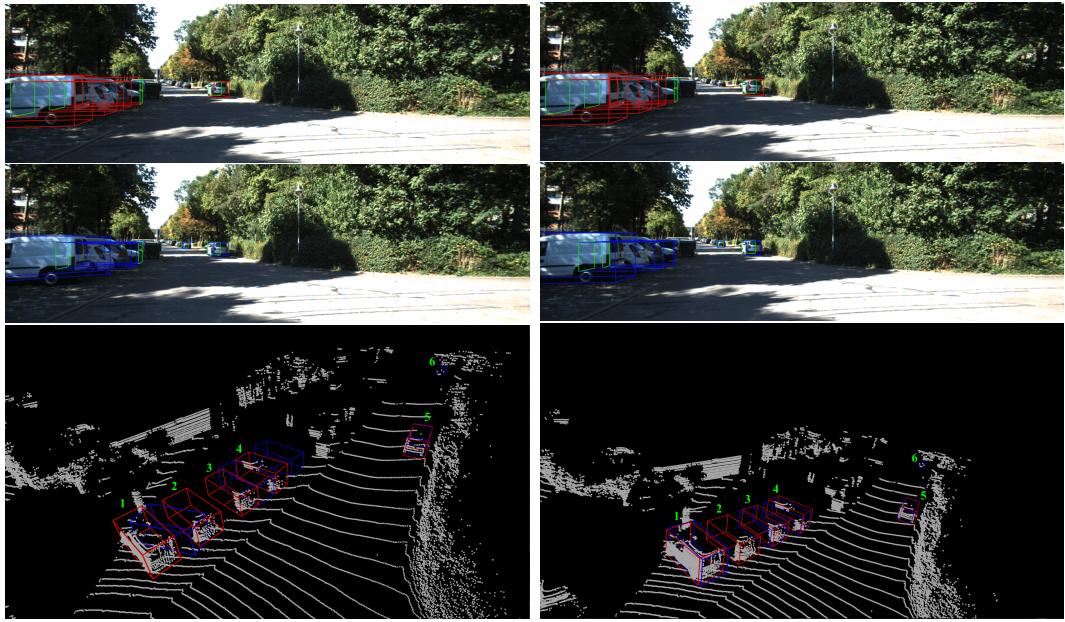
Wanneer auto's redelijk dicht op elkaar langs de weg staan geparkeerd, kan de detectie ook mislukken. Zoals te zien in figuur 5.15 wordt auto 2 gemist en auto 1, 3 en 4 slecht voorspeld door het ResidualGeoLocNN. Ook het FrustumPointNet mist auto 2 maar voorspeld auto 1, 3 en 4 wel goed. De nabij gelegen objecten maken segmentatie moeilijk wat leidt tot een mislukte classificatie en voorspelling van de 3DBB. Merk op dat beide methoden ook een auto 6 voorspellen die niet geannoteerd is in de *ground truth*.

Het voorspellen van veel objecten kan ook goed verlopen. Zoals te zien in figuur 5.16 detecteert zowel het ResidualGeoLocNN als het FrustumPointNet de 10 auto's die geannoteerd zijn in de *ground truth*. Daar waar het ResidualGeoLocNN het moeilijk krijgt om heel verre auto's (7, 8, 9 en 10) te detecteren, kan het FrustumPointNet dit wel goed.

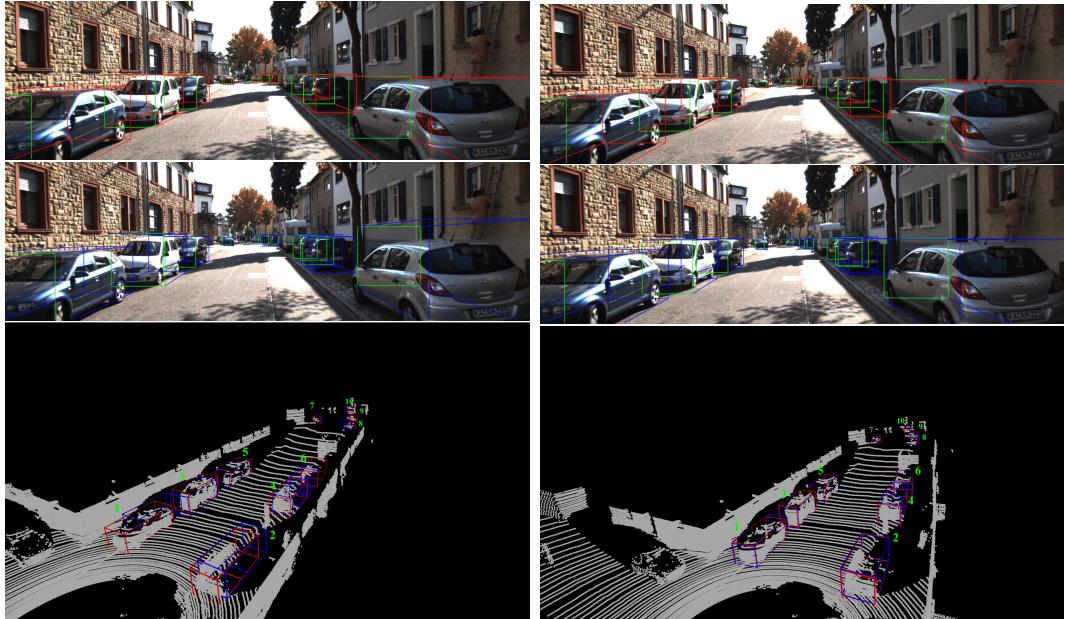
Fout in de *ground truth* annotatie

In sommige situaties moet men de *ground truth* annotaties kritisch beoordelen. Men ziet in figuur 5.17 dat er 3 geannoteerde auto's moeten voorspeld worden. Nog in de 2D afbeelding noch in de 3D puntenwolk kan met auto 3 herkennen. Dit is dus fout in de *ground truth*.

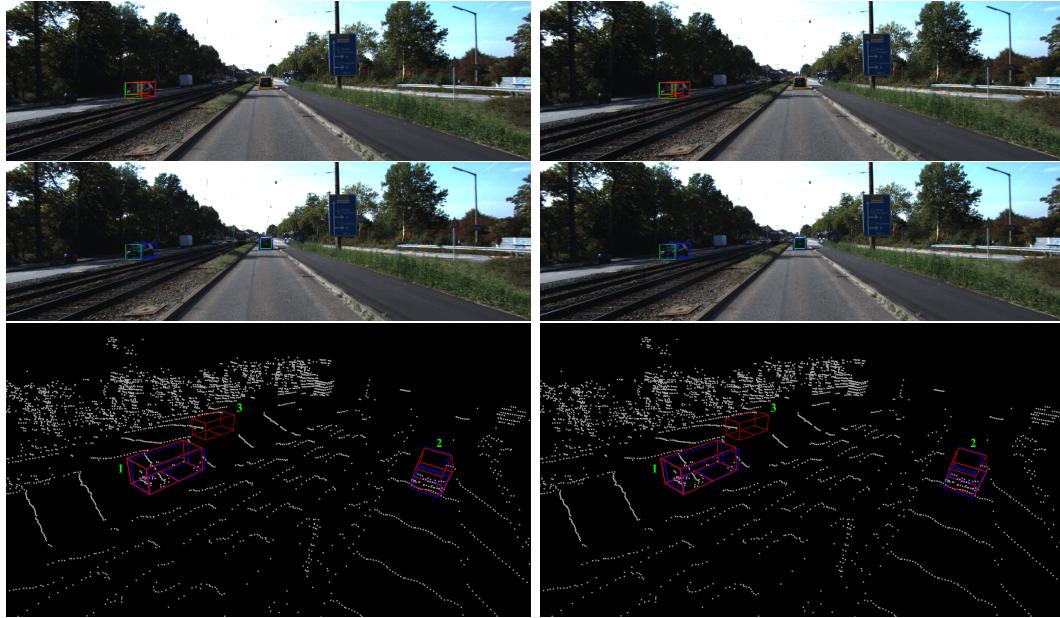
5. EXPERIMENTEN EN RESULTATEN



Figuur 5.15: Visualisatie van voorbeeld 373 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 2 wordt gemist door het ResidualGeoLocNN.



Figuur 5.16: Visualisatie van voorbeeld 2107 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat alle auto's door beide methoden gedetecteerd worden..



Figuur 5.17: Visualisatie van voorbeeld 6913 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 3 een foute annotatie is in de *ground truth*.

Detecties op een helling

Er doen zich ook situaties voor waar een auto op een helling moet gedetecteerd worden. Dit is niet vanzelfsprekend aangezien de train en validatie set veel meer situaties bevat waarin de auto's op een horizontaal vlak staan. Het is dus een uitdaging voor de camera en de lidar om met dit soort situaties om te gaan. Figuur 5.18 toont de resultaten van een situatie met een auto op een hellend vlak. Verbazingwekkend genoeg kan zowel het ResidualGeoLocNN en FrustumPointNet de helling van de wagen goed voorspellen.

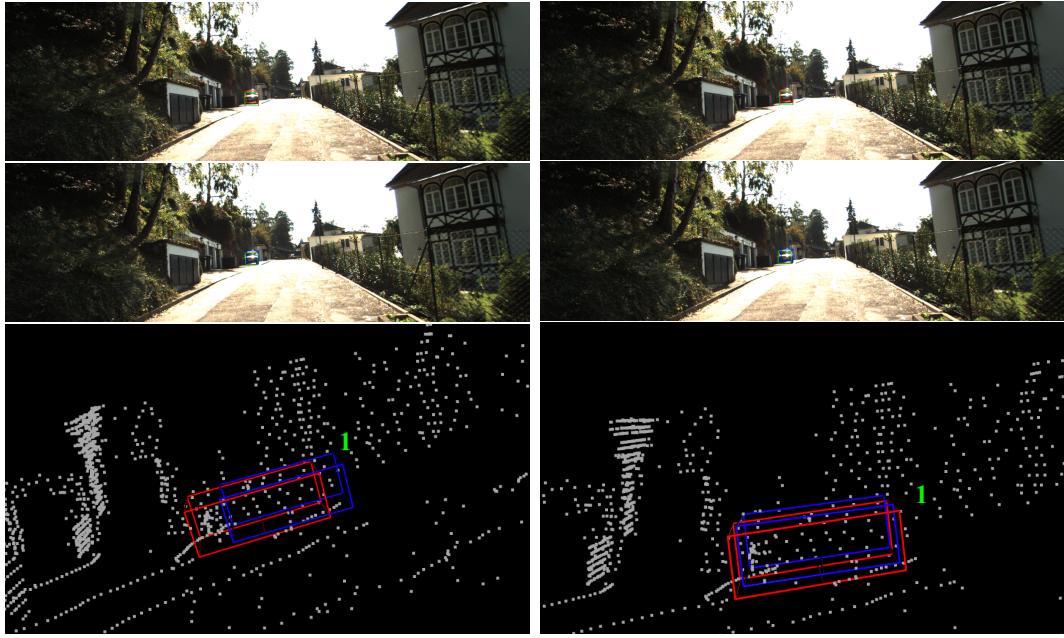
Onverwachte voorspelling

Desondanks dat auto 6, op figuur 5.19, aan de andere kant van de berm rijdt en visueel bijna niet zichtbaar is, wordt hij toch gedetecteerd door het ResidualGeoLocNN en het FrustumPointNet. Dankzij een goede 2D detector kan men hier een goede 3DBB voorspellen.

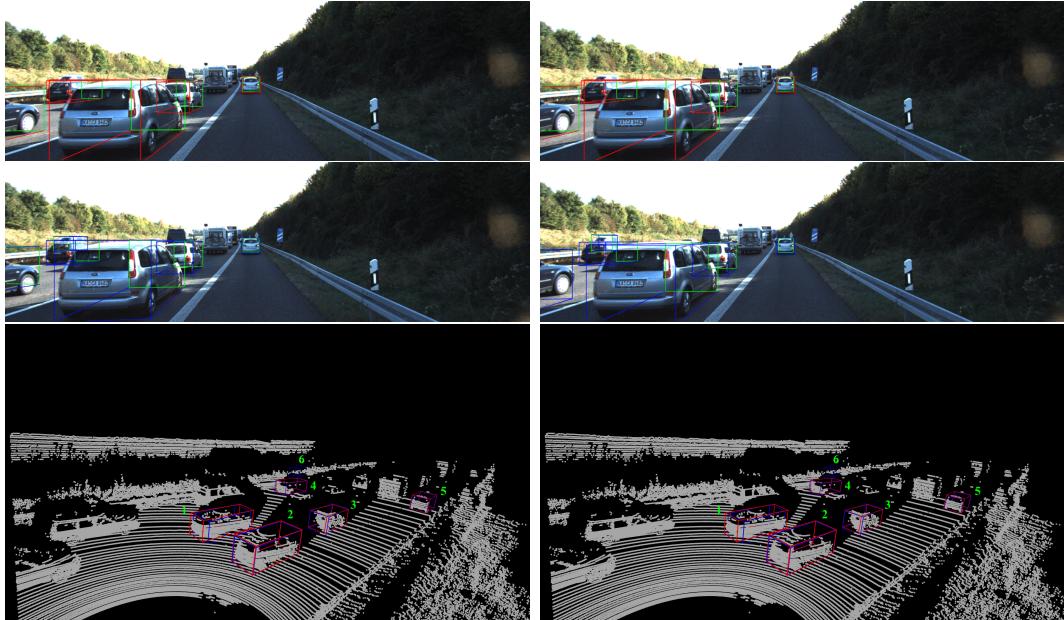
5.4.2 Bijkomende resultaten

Voor meer visualisaties van het ResidualGeoLocNN en het FrustumPointNet kan men gaan kijken naar figuur B.1 tot en met B.6.

5. EXPERIMENTEN EN RESULTATEN



Figuur 5.18: Visualisatie van voorbeeld 3292 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 1 op een helling staat en deze goed gedetecteerd wordt. De puntenwolk wordt als rechterzij-aanzicht voorgesteld.



Figuur 5.19: Visualisatie van voorbeeld 249 uit de validatie set voor het ResidualGeoLocNN (links) en FrustumPointNet (rechts). Merk op dat auto 6 verbazingwekkend genoeg wordt gedetecteerd.

Hoofdstuk 6

Conclusie

In deze thesis werd het 3D object detectie probleem bestudeerd. Meer bepaald werd een camera, lidar en camera-lidar gebaseerde methode onderzocht. Voor de camera-gebaseerde methode designde men drie nieuwe architecturen (LocNN, GeoLocNN en ResidualGeoLocNN) geïnspireerd op Deep3DBox [30]. Uit de kwantitatieve resultaten in tabel 5.2 kan men besluiten dat het ResidualGeoLocNN vrij goed scoort en behoorlijk competitief is in vergelijking met de SOTA camera-gebaseerde 3DOD modellen. Ondanks de goede resultaten kunnen deze niet tippen aan de resultaten van een lidar gebaseerd model.

Voor het lidar gebaseerde model implementeerde men het FrustumPointNet. Uit 5.2 kan men besluiten dat onze implementatie de resultaten uit de originele Frustum-PointNet [36] evenaart. Sterker nog, de 3D-AP en BEV-AP van onze architectuur is meer robuust voor de *moderate* en *hard* klasse. Daar waar de originele architectuur een enorme drop kent naarmate het te detecteren object moeilijker wordt, is onze architectuur daar bijna niet gevoelig aan. Een tekortkoming van onze lidar-gebaseerde methode is dat het vertrouwt op een 2D detector: dit wil zeggen dat er geen 3D detectie is wanneer er geen 2D detectie is. Wanneer men een performante 2D detector gebruikt zorgt dit niet voor grote problemen maar het kan potentieel de maximale prestaties van het model en de algemene bruikbaarheid van de architectuur beperken.

Voor het camera-lidar gebaseerde model ontwikkelde men drie netwerken op basis van het percentage camera vs lidar features. Uit de kwantitatieve resultaten in 5.2 presteerde het FusionNet-moreImage merkwaardig genoeg het beste. Echter is men er niet in geslaagd om de 3D-AP en BEV-AP te verhogen door het FrustumPointNet uit te breiden met features uit afbeeldingen. Dit was eerder een verrassend resultaat omdat men intuïtief zou verwachten dat het gebruik van extra informatie zou leiden tot betere resultaten. Desalniettemin is het gebleken dat het een uitdaging is hoe en waar men deze twee feature sets moet combineren.

6.1 Toekomstig werk

Deze sectie bespreekt de mogelijke toekomstige ideeën die van toepassing zijn op het 3DOD probleem.

6.1.1 Camera-lidar fusie

Ondanks dat er gevarieerd is met het percentage camera- en lidar features, is men in deze thesis er niet in geslaagd om betere resultaten te bekomen in vergelijking met een lidar gebaseerde methode. Het eerste dat men kan doen is nog verder onderzoeken hoe men de features uit het VGG-16 netwerk het beste kan combineren met het FrustumPointNet:

- Bestaat er een bepaald percentage camera- en lidar features zodanig dat men wel betere resultaten?
- Kan men de camera features op een andere locatie inbrengen in het FrustumPointNet, bv. early of deep fusion?

Een bedenking die men bij het lezen van deze vragen moet maken is dat het kan zijn dat de FrustumPointNet architectuur niet echt gemaakt is om deze te fusioneren met camera features. Het kan dus mogelijk zijn dat men stuit op de limieten van het FrustumPointNet en dat men nooit betere resultaten zal bekomen door extra camera features toe te voegen. Als dit het geval is, kan men opzoek gaan naar een totaal nieuwe architectuur die vertrekt vanuit het gedacht om voorspellingen te maken op basis van zowel camera als lidar features, zoals gedaan in [56] [21] [24].

6.1.2 Graph-CNN

In sectie 2.3.2 werd er reeds vermeld dat de meest recente lidar gebaseerde architecturen soms gebruik maken van een GCNN. Tevens kan men in tabel 5.2 zien dat het Point-GNN netwerk [47] behoort tot de SOTA lidar gebaseerde methoden. Aangezien een GCNN doorgaans beter is in het classificeren en segmenteren van puntenwolken zou een GCNN in het FrustumPointNet verwerkt kunnen worden op het niveau van het InstanceSeg-Net, zie figuur 4.8.

Om een idee te krijgen wat een GCNN potentieel kan betekenen voor het FrustumPointNet heeft men het InstanceSeg-Net eens gesubstitueerd door een GCNN instance segmentatie netwerk. Als GCNN werd er gekozen om de paper DGCNN [51] te implementeren.

Tabel 6.1 toont de eerste resultaten wanneer men een GCNN gebruikt binnnen het FrustumPointNet. Merk hierbij op dat er een stijging is in de 3D-AP en de BEV-AP voor de klasse *easy* in vergelijking met het FrustumPointNet. Vreemd genoeg kent de *moderate* en *hard* klasse een enorme drop in nauwkeurigheid. Het feit dat men een verbetering ziet voor de *easy* klasse is beloftevol. Verder onderzoek moet uitmaken hoe men een verbetering kan krijgen over alle moeilijkheidscategorieën.

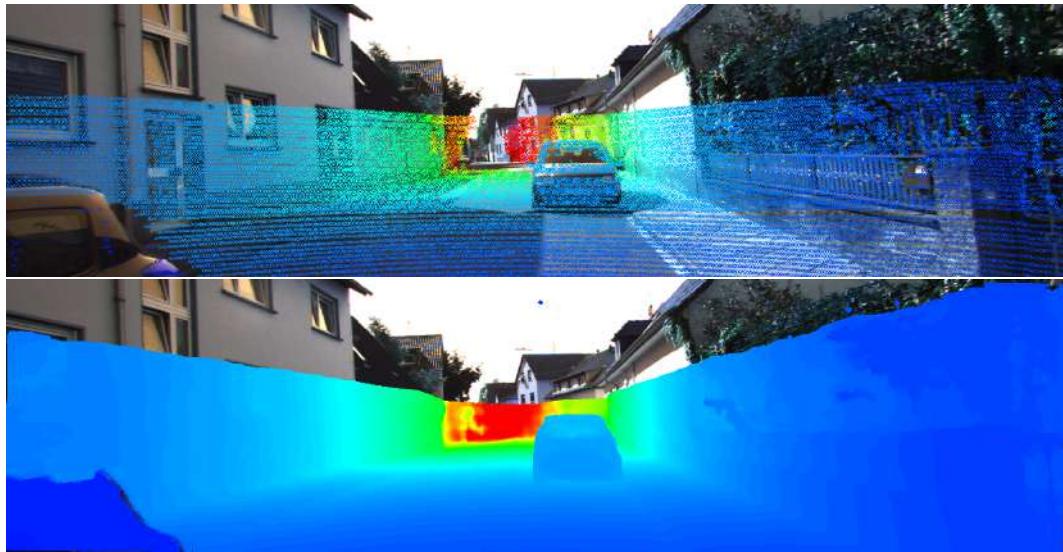
| Model | Input data | | 3D-AP (%) | | | BEV-AP (%) | | |
|-----------------|------------|-------|--------------|----------|-------|--------------|----------|-------|
| | Camera | Lidar | easy | moderate | hard | easy | moderate | hard |
| FrustumPointNet | | ✓ | 74.00 | 77.98 | 74.3 | 76.42 | 80.09 | 78.19 |
| FrustumGraphCNN | | ✓ | 76.40 | 59.68 | 43.03 | 77.40 | 60.19 | 43.39 |

Tabel 6.1: Eerste testresultaten van het FrustumGraphCNN op de KITTI validatie set (auto's). De vetgedrukte blauwe cijfers zijn een verbetering ten opzichte van het FrustumPointNet.

6.1.3 Pseudo-lidar

Uit de kwantitatieve en kwalitatieve experimenten die in deze thesis werden gemaakt kan men concluderen dat een lidar gebaseerde methode superieur is ten opzicht van een camera gebaseerde methode. Desondanks is een lidar vanuit een economisch standpunt een hele dure sensor. Ter vergelijking: de camera en lidar die gebruikt wordt voor de KITTI dataset kost \$1000 respectievelijk \$75.000.

Er wordt in de literatuur ook getracht om een dieptekaart te genereren op basis van een stereofoto, bestaande uit een linker en rechter afbeelding. Deze dieptekaart wordt op zijn beurt omgevormd naar puntenwolken. Zo verkrijgt men een pseudo-lidar input die gemaakt is door een goedkope camera. Het verschil tussen een lidar- en pseudo-lidar input is te zien in figuur 6.1. De pseudo-lidar is in tegenstelling tot lidar een *dense* input aangezien er voor elke pixel in de afbeelding een diepte wordt geschat. Bovendien ziet men op deze figuur dat de pseudo lidar de echte lidar vrij goed benadert.



Figuur 6.1: Boven: lidar input. Onder: pseudo-lidar input. Op basis van de diepte wordt elke pixel ingekleurd. Blauw is voor nabijgelegen pixels, rood voor verre pixels.

6. CONCLUSIE

Het mooie aan de pseudo-lidar data is dat het als input gebruikt kan worden voor een lidar gebaseerde methode, zonder tussenkomst van een echte fysieke lidar sensor.

Ook hier werd al een initiële test gedaan met het gebruik van pseudo-lidar. Een pre-trained netwerk, PSMNet [5], werd gebruikt voor het genereren van een dieptekaart op basis van een linker en rechter afbeelding. Na de transformatie naar pseudo-lidar werd dit gebruikt als input voor het FrustumPointNet en werd alles hertraind.

Tabel 6.2 toont de resultaten van het FrustumPointNet-pseudoLidar netwerk in vergelijking met het beste camera gebaseerde model van deze thesis. Merk op dat, hoewel het FrustumPointNet lidar data als input verwacht, het FrustumPointNet-pseudoLidar een camera gebaseerde methode is aangezien de pseudo-lidar voortkomt uit afbeeldingen. Als men de resultaten van het FrustumPointNet-pseudoLidar netwerk vergelijkt met het ResidualGeoLocNN netwerk zijn deze verbazingwekkend goed en veelbelovend. Dit stimuleert het verdere onderzoek binnen het pseudo-lidar domein.

| Model | Input data | | 3D-AP (%) | | | BEV-AP (%) | | |
|-----------------------------|------------|-------|-----------|----------|-------|------------|----------|-------|
| | Camera | Lidar | easy | moderate | hard | easy | moderate | hard |
| ResidualGeoLocNN | ✓ | | 13.40 | 12.45 | 12.70 | 16.69 | 15.72 | 16.18 |
| FrustumPointNet-pseudoLidar | ✓ | | 55.98 | 53.64 | 50.67 | 59.53 | 58.70 | 57.18 |

Tabel 6.2: Eerste testresultaten van het FrustumPointNet-pseudoLidar op de KITTI validatie set (auto's).

6.1.4 Meer klassen detecteren binnen KITTI

Door de klasse onevenwicht van de KITTI dataset, zie tabel 3.3, werd er enkel getraind op de auto's. Een logische uitbreiding van deze thesis is de uitbreiding naar de overige klassen uit de KITTI dataset, zoals voetgangers en fietsers. Dit brengt natuurlijk de nodige uitdagingen met zich mee aangezien er veel minder gelabelde voetgangers en fietsers aanwezig zijn in de dataset.

6.1.5 Domain Adaptation

Een andere tak die interessant is om te onderzoeken is de domain adaptation. Men zou kunnen onderzoeken of een model getraind op KITTI al dan niet generaliseerbaar is naar een andere gelijkaardige dataset, bv. de nuScenes dataset [1]. Als blijkt dat een model getraind op KITTI generaliseerbaar is naar nuScenes, kan het voor een bedrijf zoals Xenomatix nuttig zijn om dit model te gebruiken voor het genereren van *ground truth* annotaties op een semi-automatische manier.

Bijlagen

Bijlage A

Hyperparameters

A.1 GeoLoc

| | |
|----------------------|--------|
| Batch grootte | 32 |
| Epochs | 100 |
| Learning rate | 0.0001 |
| Optimizer | Adam |
| α | 0.5 |
| β | 0.1 |
| γ | 1.0 |

Tabel A.1: De hyperparameters voor het GeoLoc netwerk.

A.2 LocNN

| | |
|----------------------|------|
| Batch grootte | 128 |
| Epochs | 100 |
| Learning rate | 0.01 |
| Optimizer | Adam |

Tabel A.2: De hyperparameters voor het LocNN netwerk.

A. HYPERPARAMETERS

| FC Lagen | 2DBB | 3D dimensie | 3DLOC _{error} | XLOC _{error} | YLOC _{error} | ZLOC _{error} |
|----------|------|-------------|------------------------|-----------------------|-----------------------|-----------------------|
| 2 | ✓ | | 3.32 | 1.11 | 0.30 | 2.92 |
| 2 | ✓ | ✓ | 2.79 | 0.92 | 0.27 | 2.44 |
| 3 | ✓ | | 3.03 | 0.98 | 0.24 | 2.69 |
| 3 | ✓ | ✓ | 2.65 | 0.89 | 0.24 | 2.31 |
| 4 | ✓ | | 2.69 | 0.82 | 0.26 | 2.41 |
| 4 | ✓ | ✓ | 2.59 | 0.81 | 0.25 | 2.3 |
| 5 | ✓ | | 2.59 | 0.72 | 0.32 | 2.33 |
| 5 | ✓ | ✓ | 2.51 | 0.80 | 0.21 | 2.23 |
| 6 | ✓ | | 2.53 | 0.77 | 0.25 | 2.25 |
| 6 | ✓ | ✓ | 2.43 | 0.69 | 0.24 | 2.19 |
| 7 | ✓ | | 2.48 | 0.79 | 0.25 | 2.19 |
| 7 | ✓ | ✓ | 2.48 | 0.73 | 0.25 | 2.22 |

Tabel A.3: De gemiddelde 3D locatie error voor de verschillende configuraties van LocNN, geëvalueerd op de auto's uit de KITTI validatie set.

A.3 GeoLocNN

| | |
|----------------------|------|
| Batch grootte | 128 |
| Epochs | 100 |
| Learning rate | 0.01 |
| Optimizer | Adam |

Tabel A.4: De hyperparameters voor het GeoLocNN netwerk.

| FC Lagen | GeoLoc | 2DBB | 3DLOC_{error} | XLOC_{error} | YLOC_{error} | ZLOC_{error} |
|-----------------|---------------|-------------|------------------------------|-----------------------------|-----------------------------|-----------------------------|
| 2 | ✓ | | 2.47 | 0.68 | 0.11 | 2.30 |
| 2 | ✓ | ✓ | 2.42 | 0.7 | 0.20 | 2.2 |
| 3 | ✓ | | 2.47 | 0.70 | 0.15 | 2.28 |
| 3 | ✓ | ✓ | 2.35 | 0.66 | 0.21 | 2.14 |
| 4 | ✓ | | 2.49 | 0.68 | 0.12 | 2.31 |
| 4 | ✓ | ✓ | 2.30 | 0.64 | 0.20 | 2.09 |
| 5 | ✓ | | 2.45 | 0.69 | 0.14 | 2.26 |
| 5 | ✓ | ✓ | 2.33 | 0.68 | 0.24 | 2.09 |
| 6 | ✓ | | 2.46 | 0.66 | 0.25 | 2.25 |
| 6 | ✓ | ✓ | 2.31 | 0.68 | 0.25 | 2.07 |
| 7 | ✓ | | 2.48 | 0.71 | 0.12 | 2.29 |
| 7 | ✓ | ✓ | 2.31 | 0.63 | 0.25 | 2.10 |

Tabel A.5: De gemiddelde 3D locatie error voor de verschillende configuraties van GeoLocNN, geëvalueerd op de auto's uit de KITTI validatie set.

A.4 FrustumPointNet

| | |
|----------------------|-------|
| Batch grootte | 64 |
| Epochs | 200 |
| Learning rate | 0.001 |
| Optimizer | Adam |
| α | 1 |
| β | 10 |

Tabel A.6: De hyperparameters voor het FrustumPointNet netwerk.

A. HYPERPARAMETERS

A.5 FusionNet

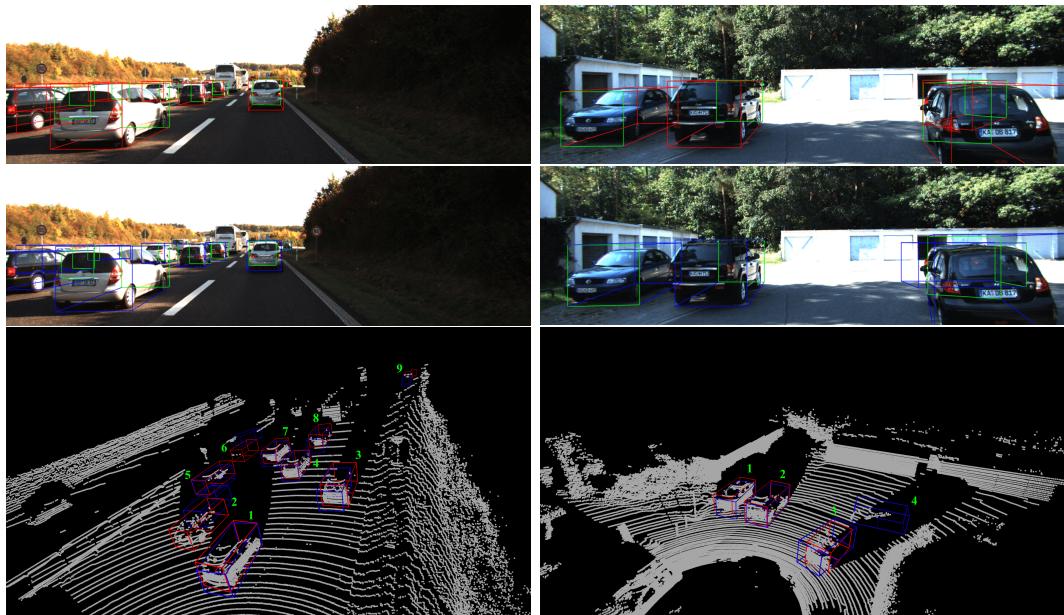
| | |
|----------------------|-------|
| Batch grootte | 16 |
| Epochs | 200 |
| Learning rate | 0.001 |
| Optimizer | Adam |
| α | 1 |
| β | 10 |

Tabel A.7: De hyperparameters voor het FusionNet netwerk.

Bijlage B

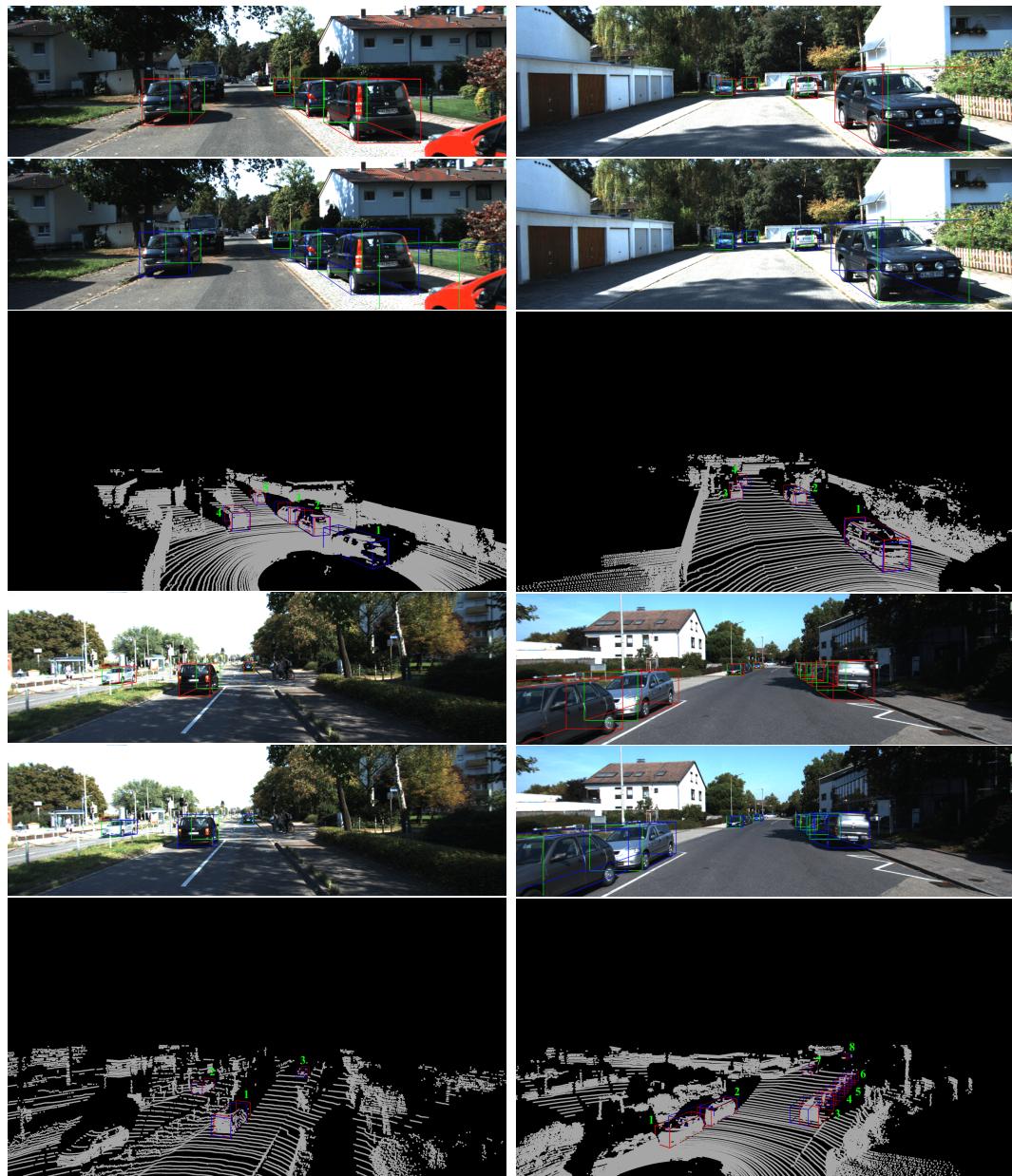
Kwalitatieve resultaten

B.1 ResidualGeoLoc

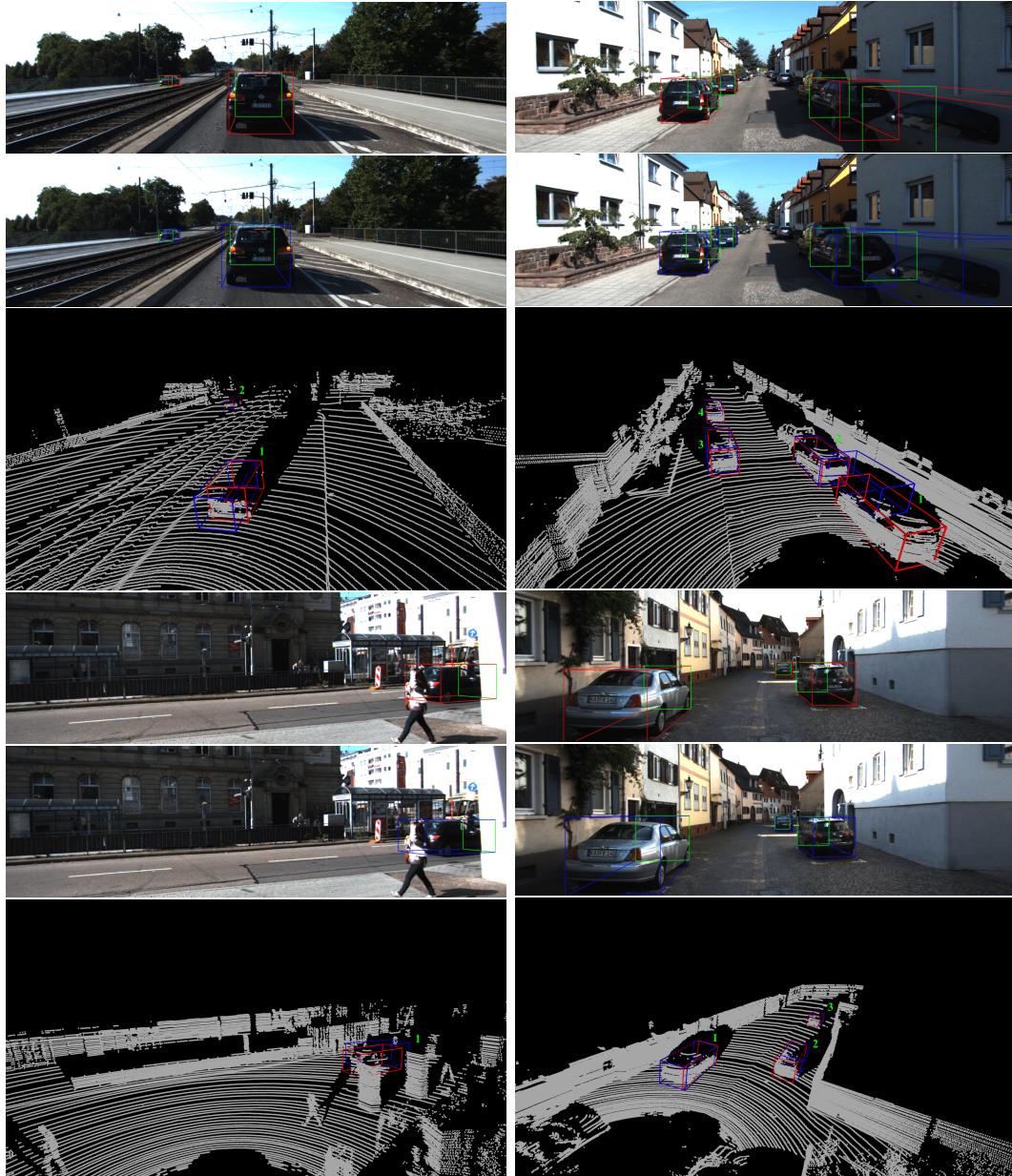


Figuur B.1: Enkele visualisaties uit de validatie set voor het ResidualGeoLocNN.

B. KWALITATIEVE RESULTATEN



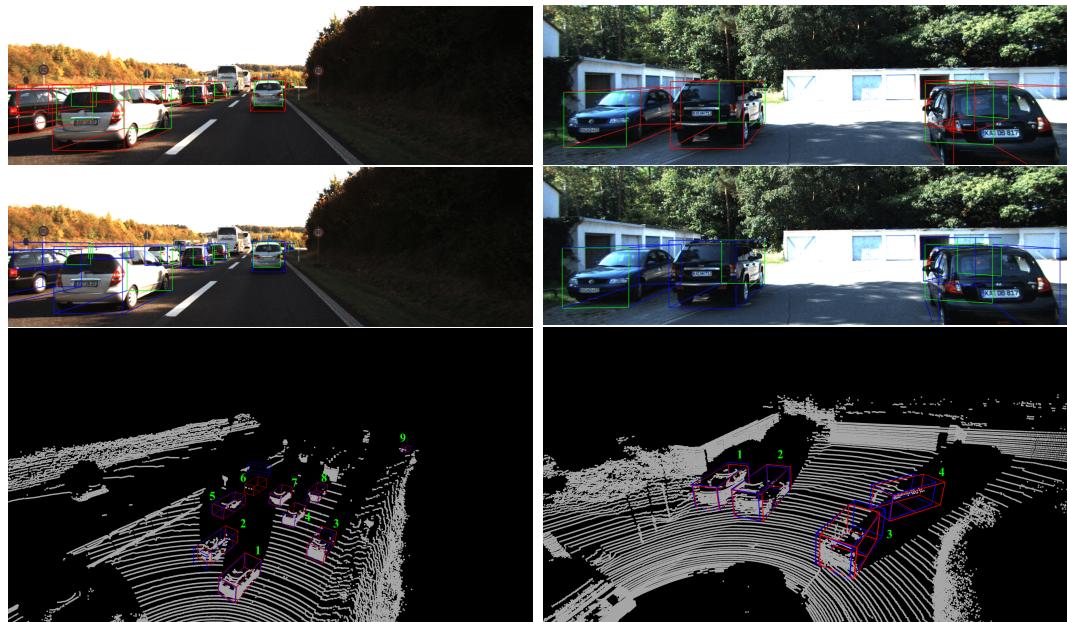
Figuur B.2: Enkele visualisaties uit de validatie set voor het ResidualGeoLocNN.



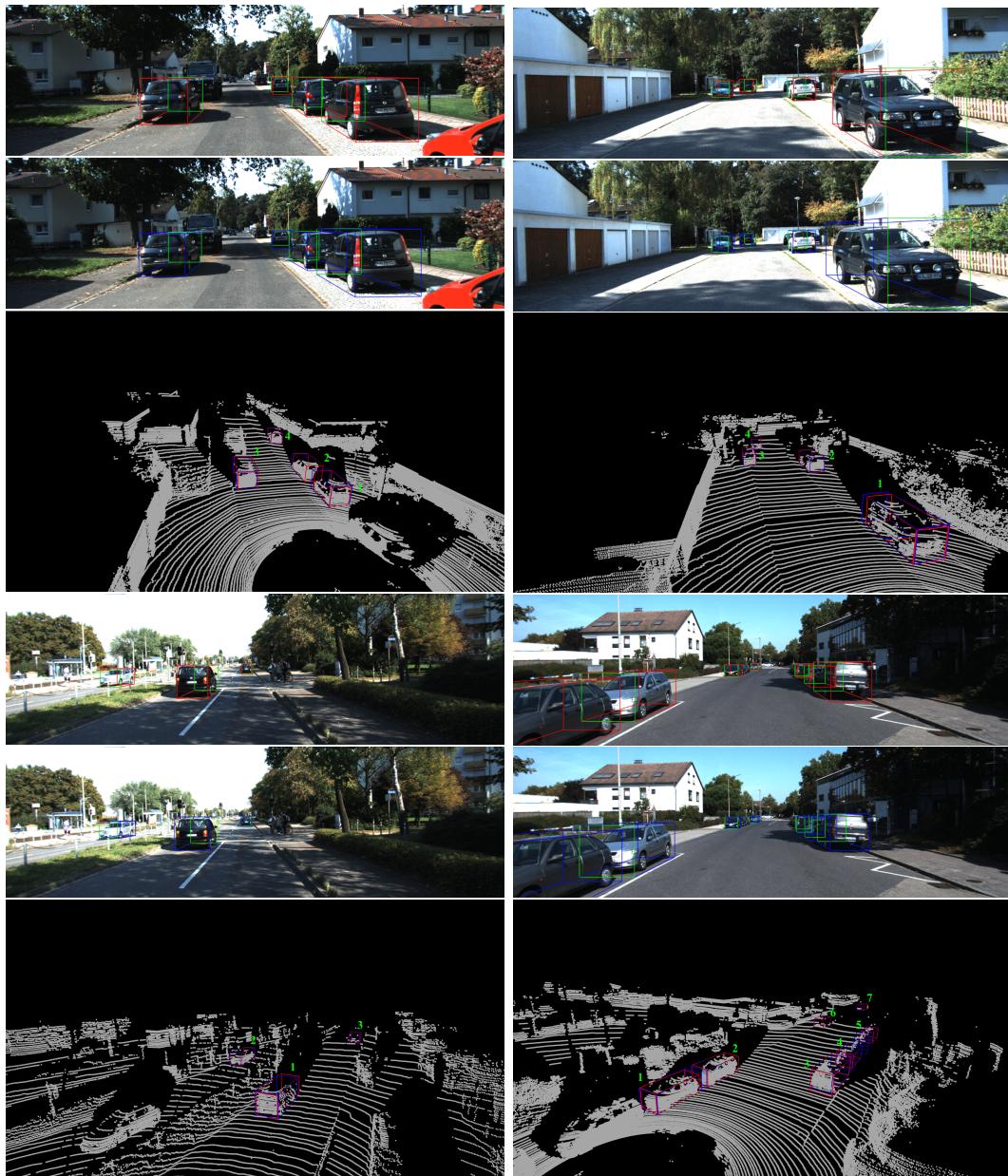
Figuur B.3: Enkele visualisaties uit de validatie set voor het ResidualGeoLocNN.

B. Kwalitatieve resultaten

B.2 FrustumPointNet

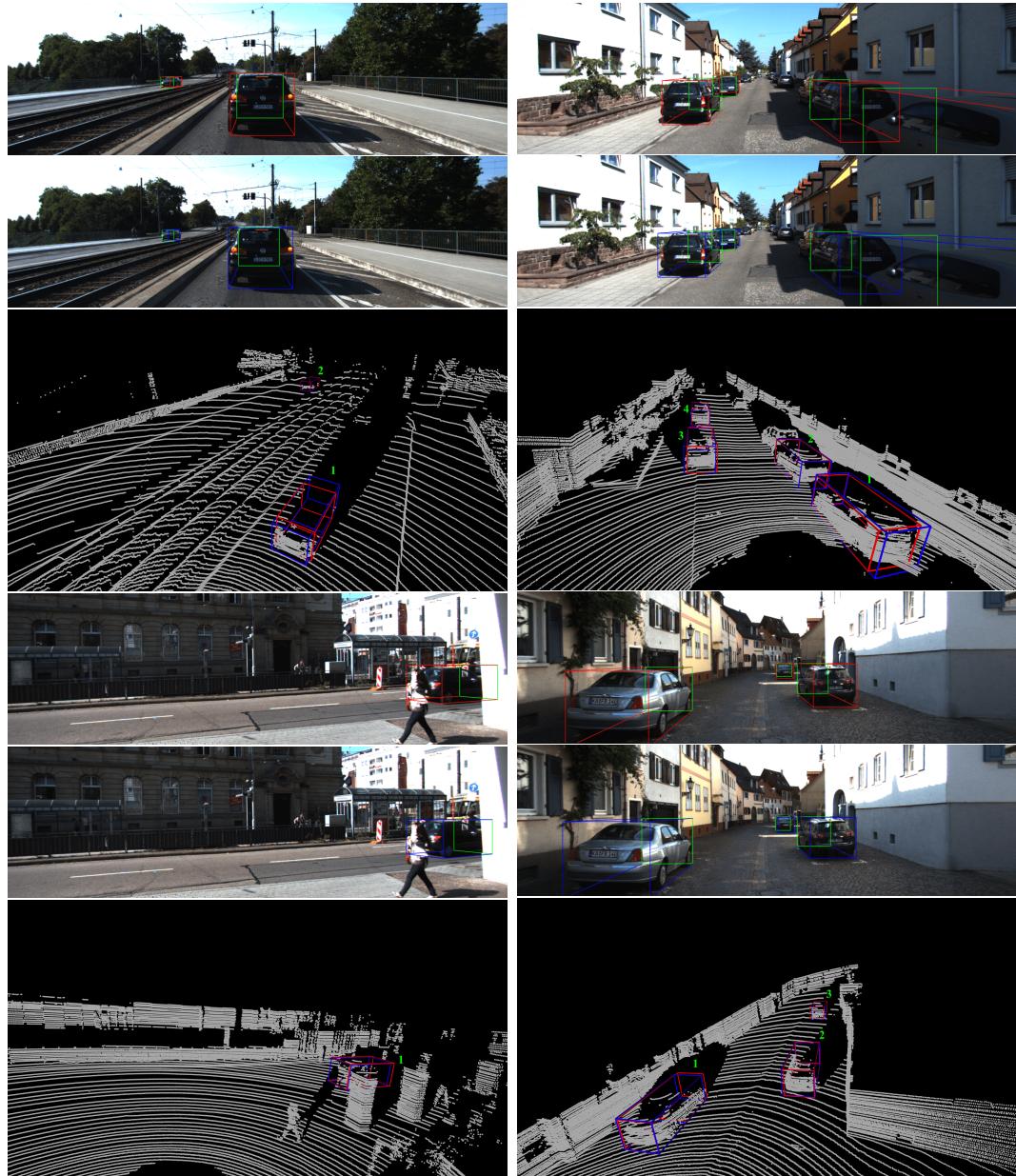


Figuur B.4: Enkele visualisaties uit de validatie set voor het FrustumPointNet.



Figuur B.5: Enkele visualisaties uit de validatie set voor het FrustumPointNet.

B. KWALITATIEVE RESULTATEN



Figuur B.6: Enkele visualisaties uit de validatie set voor het FrustumPointNet.

Bibliografie

- [1] APTIV. The nuscenes dataset. <https://www.nuscenes.org>.
- [2] G. Brazil and X. Liu. M3D-RPN: monocular 3d region proposal network for object detection. *CoRR*, abs/1907.06038, 2019.
- [3] V. Buhrmester, D. Münch, and M. Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey. *ArXiv*, abs/1911.12116, 2019.
- [4] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CoRR*, abs/1903.11027, 2019.
- [5] J. Chang and Y. Chen. Pyramid stereo matching network. *CoRR*, abs/1803.08669, 2018.
- [6] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. *CVPR*, 2016.
- [7] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.
- [8] A. Costea, A. Petrovai, and S. Nedevschi. Fusion scheme for semantic and instance-level segmentation. pages 3469–3475, 11 2018.
- [9] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. *CoRR*, abs/1609.06666, 2016.
- [10] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015.
- [11] A. Geiger. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR ’12, page 3354–3361, USA, 2012. IEEE Computer Society.

BIBLIOGRAFIE

- [12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *I. J. Robotic Res.*, 32(11):1231–1237, 2013.
- [13] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.
- [14] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.
- [15] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CoRR*, abs/1609.03677, 2016.
- [16] Google. Tensorflow object detection api. https://github.com/tensorflow/models/tree/master/research/object_detection.
- [17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [18] D. Hebb. The organization of behavior; a neuropsychological theory. *Wiley*, 1949.
- [19] E. Hoffer, T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry. Augment your batch: better training with larger batches. *CoRR*, abs/1901.09335, 2019.
- [20] A. G. Hoffmann. Artificial and natural computation. In J. D. Wright, editor, *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)*, pages 27 – 31. Elsevier, second edition edition, 2015.
- [21] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander. Joint 3d proposal generation and object detection from view aggregation. *CoRR*, abs/1712.02294, 2017.
- [22] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *CoRR*, abs/1812.05784, 2018.
- [23] P. Li, H. Zhao, P. Liu, and F. Cao. Rtm3d: Real-time monocular 3d detection from object keypoints for autonomous driving. 01 2020.
- [24] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [25] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.
- [26] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

- [27] L. Liu, J. Lu, C. Xu, Q. Tian, and J. Zhou. Deep fitting degree scoring network for monocular 3d object detection. *CoRR*, abs/1904.12681, 2019.
- [28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.
- [29] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943.
- [30] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. 3d bounding box estimation using deep learning and geometry. *CoRR*, abs/1612.00496, 2016.
- [31] Neurohive. Vgg16 – convolutional network for classification and detection. <https://neurohive.io/en/popular-networks/vgg16/>.
- [32] M. Nielsen. Neural networks and deep learning. <http://neuralnetworksanddeeplearning.com>.
- [33] Nvidia. The intelligent industrial revolution. <https://blogs.nvidia.com/blog/2016/10/24/intelligent-industrial-revolution/>.
- [34] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Jr. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5:6, 02 2017.
- [35] H. Pokharna. Artifical neuron. <https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb>.
- [36] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017.
- [37] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.
- [38] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for rgbd semantic segmentation. pages 5209–5218, 10 2017.
- [39] Z. Qin, J. Wang, and Y. Lu. Monogrnet: A geometric reasoning network for monocular 3d object localization. *CoRR*, abs/1811.10247, 2018.
- [40] Qwertee. Deep learning with point clouds. <https://www.qwertee.io/blog/deep-learning-with-point-clouds/>.
- [41] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [42] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

- [43] T. Roddick, A. Kendall, and R. Cipolla. Orthographic feature transform for monocular 3d object detection. *CoRR*, abs/1811.08188, 2018.
- [44] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.
- [45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.
- [46] S. Saha. Convolutional neural networks. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- [47] W. Shi and R. R. Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. 2020. cite arxiv:2003.01251.
- [48] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. cite arxiv:1409.1556.
- [49] Stanford. Cs231n: Convolutional neural networks for visual recognition. <https://cs231n.github.io>.
- [50] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.
- [51] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.
- [52] Wikipedia. Backpropagation. <https://en.wikipedia.org/wiki/Backpropagation>.
- [53] Wikipedia. Biological neuron. <https://simple.wikipedia.org/wiki/Neuron>.
- [54] Wikipedia. Lidar. <https://nl.wikipedia.org/wiki/Lidar>.
- [55] B. Xu and Z. Chen. Multi-level fusion based 3d object detection from monocular images. pages 2345–2353, 06 2018.
- [56] D. Xu, D. Anguelov, and A. Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. *CoRR*, abs/1711.10871, 2017.
- [57] Y. Yan, Y. Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18:3337, 10 2018.
- [58] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia. STD: sparse-to-dense 3d object detector for point cloud. *CoRR*, abs/1907.10471, 2019.
- [59] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.