KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS DE NAYER SINT-KATELIJNE-WAVER

Mobile Deep Visual Detection and Recognition

Ondertitel (facultatief)

Thijs VERCAMMEN

Promotor(en): Prof. dr. ir. Toon Goedéme

Co-promotor(en): Ing. Floris De Feyter

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: Elektronica-ICT
ICT

Academiejaar 2021 - 2022

Voorwoord

Het voorwoord vul je persoonlijk in met een appreciatie of dankbetuiging aan de mensen die je hebben bijgestaan tijdens het verwezenlijken van je masterproef en je hebben gesteund tijdens je studie.

Samenvatting

De (korte) samenvatting, toegankelijk voor een breed publiek, wordt in het Nederlands geschreven en bevat **maximum 3500 tekens**. Deze samenvatting moet ook verplicht opgeladen worden in KU Loket.

Abstract

Het extended abstract of de wetenschappelijke samenvatting wordt in het Engels geschreven en bevat **500 tot 1.500 woorden**. Dit abstract moet **niet** in KU Loket opgeladen worden (vanwege de beperkte beschikbare ruimte daar).

Keywords: Voeg een vijftal keywords in (bv: Latex-template, thesis, ...)

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van symbolen

Maak een lijst van de gebruikte symbolen. Geef het symbool, naam en eenheid. Gebruik steeds SI-eenheden en gebruik de symbolen en namen zoals deze voorkomen in de hedendaagse literatuur en normen. De symbolen worden alfabetisch gerangschikt in opeenvolgende lijsten: kleine letters, hoofdletters, Griekse kleine letters, Griekse hoofdletters. Onderstaande tabel geeft het format dat kan ingevuld en uitgebreid worden. Wanneer het symbool een eerste maal in de tekst of in een formule wordt gebruikt, moet het symbool verklaard worden. Verwijder deze tekst wanneer je je thesis maakt.

b	Breedte	$[mm]$
A	Oppervlakte van de dwarsdoorsnede	$[mm^2]$
c	Lichtsnelheid	$[m/s]$

Lijst van afkortingen

CNN RoIs ReLu

Hoofdstuk 1

Situering en doelstelling

1.1 Situering

Tegenwoordig wordt deep learning steeds meer en meer gebruikt om beeldverwerking problemen op te lossen. Via neurale netwerken kunnen we met meer en betere features werken om de afbeeldingen te analiseren. Maar veel van deze modelen hebben behoorlijk wat rekenkracht en geheugen nodig om te werken. Ook is er steeds meer intresse naar real-time toepassingen waarvan het resultaat zo snel mogelijk beschikbaar moet zijn. Dit wordt moeilijk bij veel hedendaagse systemen waarbij de foto eerst genomen moet worden en vervolgens door een computer geanaliseerd moet worden, omdat hedendaagse systemen veel rekenwerk en geheugen vragen. In deze masterproef wordt er onderzocht of de computer kan weggelaten worden en de afbeelding meteen door het mobiel apparaat geanaliseerd kan worden. Dus er moet onderzocht worden hoe een bestaand model aangepast kan worden om efficiënt te werken op een mobiel apparaat. Hierbij moet vooral rekening gehouden worden met de rekenkracht en geheugen van het mobiele apparaat.

1.2 Probleemstelling

Mobiele apparaten zijn kleine toestellen met beperkt geheugen en beperkte rekenkracht. In deze masterproef wordt er onderzocht hoe het rekenwerk beperkt kan worden zodat het resultaat real-time geleverd kan worden. Er gaat ook onderzocht worden hoe alle data efficiënt kan worden opgeslagen op het toestel.

1.3 Doelstellingen

Het uiteindelijke doel van deze masterproef is er voor zorgen dat een bestaand deep learning model aangepast kan worden zodat dit real-time resultaten kan geven op een mobiel apparaat. Dit gebeurt aan de hand van de volgende stappen:

- grondig begrijpen van een deep learning herkenningssysteem

- grondig begrijpen van een deep learning detectiesysteem
- Onderzoeken welke technieken er gebruikt kunnen worden om bestaande systemen op een mobiel apparaat te implementeren.
- onderzoeken voor optimalisaties voor een herkenningssysteem
- onderzoeken voor optimalisaties voor een detectiesysteem
- gevonden technieken testen en analiseren
- werkend prototype applicatie ontwerpen voor een mobiel apparaat

Hoofdstuk 2

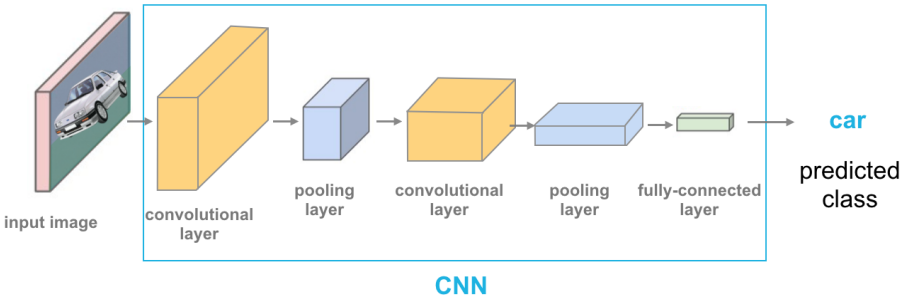
Herkenning en Detectie Algemeen

2.1 Deep learning-gebaseerde herkeningssystemen

Herkeningssystemen systemen voorspellen wat de klasse van een object is in een afbeelding. Dus het herkennen van objecten in digitale afbeeldingen zonder deze te lokaliseren of aan te duiden. Bij herkeningssystemen is er geen of weinig overlap tussen de trainingsafbeeldingen en de inputafbeeldingen. Bijvoorbeeld bij een gezichts herkenningssysteem wordt er een algemeen herkeningssystemen ontworpen dat gezichten herkent, en niet elk individueel gezicht herkent. Voor een herkenningssysteem is er een goed getraind netwerk nodig dat input afbeeldingen omzet in features. Er moet een database zijn met daarin de gegevens van de objecten die men wilt herkennen. Vervolgens hebben is er ook een methode nodig om features van het neuraal netwerk te vergelijken met de gegevens in de database om het juiste object te herkennen.

2.1.1 Herkenning

Wanneer er een getraind CNN is kan er een herkenningssysteem ontwikkeld worden. Als men bepaalde objecten in een afbeelding wil ontdekken gaat men met behulp van een CNN de afbeelding omzetten in een embedding. Embeddings Koehrsen (2018) zijn vector representaties die kunnen worden vergeleken in een embedding space, waar gelijkaardige objecten dichter bij elkaar liggen. De embedding van de input afbeelding wordt vergeleken met de embeddings die zich in een gallerij bevinden. Jiang et al. (2019) geeft aan dat met behulp van een query kunnen er gelijkaardige objecten uit de gallerij gehaald worden om deze te gaan vergelijken in een embedding space. De gallerij is een database/verzameling met gekende embeddings/ID's van de objecten die men wilt herkennen. Een query is een embedding van de input waarvan het label niet gekend is. Gelijkaardige embeddings kunnen gezocht worden via de nearest neighbour techniek, waar we naar de klasse van de dichtsbijzijnde buur gaan kijken.

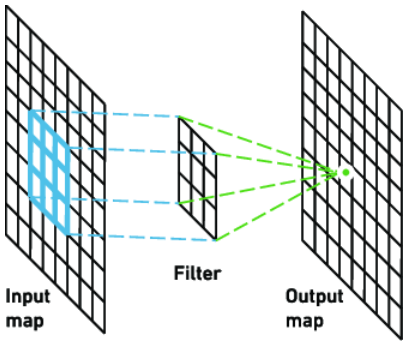


Figuur 2.1: CNN met 2 convolutie lagen en 2 pooling lagen en een fully-connected layer

2.1.2 convolutioneel neuraal netwerk (CNN)

De belankrijkste bouwsteen van een herkenningssysteem is een goed getrainde CNN, in dit hoofdstuk wordt het CNN besproken dat beschreven wordt door Jiang et al. (2019). Het algemeen model van een CNN is weergegeven in figuur 2.1. In tegenstelling tot fully connected netwerken wordt bij een CNN de gewichten gedeeld over verschillende locaties om zo het aantal parameters te verminderen.

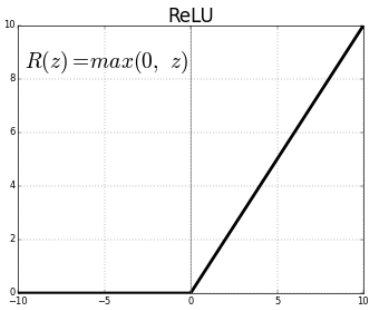
Het belangrijkste deel van een CNN zijn de convolutielagen (figuur 2.2) waarbij men een kernel/filter over de input laat gaan wat als output een feature map genereerd. Een kernel bestaat uit set van gewichten die met de input worden vermenigvuldigd, deze kernel wordt over de input afbeelding geschoven. Al de pixels binnen het veld van de kernel worden gereduceert tot een enkele waarde. CNN leren verschillende features met verschillende kernels in parallel. Waardoor de matrices met feature mappen steeds kleiner worden maar ook dieper worden. Een andere factor van een convolutie laag is de stride, deze waarde geeft aan met hoeveel pixels de kernel telkens moet doorschuiven. Een CNN bestaat uit een opeenvolging van een aantal convolutie lagen die steeds meer high-level features extraheren. Hoe meer convolutielagen een netwerk telt hoe meer features er uit de input worden gehaald, maar hoe trager het netwerk is.



Figuur 2.2: Convolutie laag waarbij een filter wordt herleid tot een output feature.

Elke convolutie laag wordt gevolgd door een niet-lineare activatie functie , de meest gebruikt functie hiervoor is de rectified linear unit (ReLu) (figuur 2.3). De ReLu wordt vaak gebruikt omdat deze eenvoudig is, kan exact 0 weergeven en ziet er lineair uit. Max(0,x) is de ReLu bewerking, dus

er wordt verdergegaan met 0 of de input waarde. Zonder een niet-lineare activatie functie kan het CNN herleid worden tot 1 convolultie laag die geen high-level features kan extraheren. Andere mogelijkheden voor Lineare activatie functies zijn: Sigmoid en Tangens hyperbolicus maar deze functies vragen meer rekenwerk.



Figuur 2.3: ReLu, waarbij het maximum wordt genomen van 0 en de input waarde.

Een volgende bouwsteen is de pooling laag waarbij het aantal samples in de feature map wordt verlaagt. De meest voorkomende methode is max-pooling waarbij er verder wordt gegaan met de maximum waarde in een bepaalde regio. Het doel van een pooling laag is om het aantal parameters te verminderen en zo ook het rekenwerk te verminderen. Er kan ook gebruik gemaakt worden van avarage pooling waarbij er verder wordt gegaan met de gemiddelde waarde van een regio. Er is ook minimal pooling waarbij er verder wordt gegaan met de minimum waarde.

Op het einde van elk CNN volgen er meestal 1 of meerdere fully connected lagen. Deze lagen connecteren elke input van één laag met elke activatie eenheid van de volgende laag. Dit zorgt voor meer parameters en meer rekenwerk waardoor deze lagen een vertragende factor vormen. De fully connected lagen zorgen voor een classificatie op basis van de features van de convolutie lagen.

2.1.3 Trainen van een CNN

Het trainen van een CNN bestaat uit het leveren van veel voorbeelden aan het netwerk. Op basis van het resultaat van deze voorbeelden worden telkens de gewichten van de kernels aangepast, zodat er steeds een beter resultaat wordt geleverd.

De loss functie geeft de error van de voorspelling weer tijdens het trainen van een neuraal netwerk. Op basis van de loss functie gaat men via de stochastic gradiënt decent de gewichten van het netwerken bijstellen zodat bij een volgende trainingsinput de loss functie een beter resultaat geeft. Bij de stochastic gradiënt decent wordt er per batch/group trainingsvoorbeelden de gewichten bijgesteld. De gradienten worden berekend door de loss af te leiden naar de gewichten via de ketting regel, en de gewichten worden bijgesteld volgens de tegengestelde gradiënt.

De learning rate bij het trainen van een CNN beïnvloed de grootte van de stap waarmee de gewichten worden bijgesteld. Hoe kleiner de learning rate hoe langer het trainen van een CNN duurt. Maar als de learning rate te hoog is kan het resultaat een slecht getraind netwerk zijn, omdat de

veranderingen op de gewichten te groot is om een beter resultaat te verkrijgen.

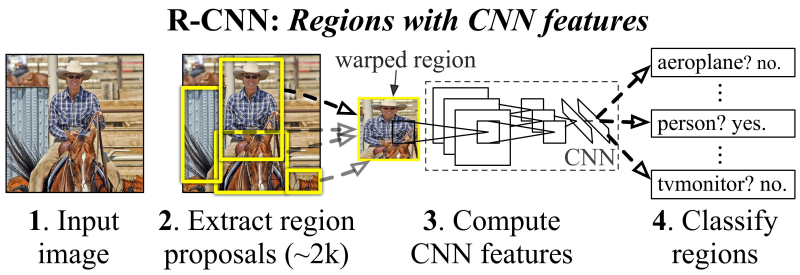
2.2 Deep learning-gebaseerde detector

Object detectie is het lokaliseren en classificeren van objecten in een afbeelding, waarbij de objecten aangeduid worden met een Bounding box. Door gebruik te maken van CNN kunnen er vrij nauwkeurige object detectoren ontworpen worden. Object detectie maakt voornamelijk gebruik van twee methodes: de single-stage detector en de two-stage detector.

2.2.1 Two-stage detector

Zoals de naam zegt bestaat deze methode uit 2 niveaus. Het eerste deel worden er Regions of Intrest (RoIs) gecreëerd, dit is het filteren van regio's waarbij de kans groot is dat deze een object bevatten. Het tweede deel classificeert en verfijnt de lokalisatie van de RoIs die in het eerste deel gecreëerd werden. Dit gebeurt door elk van de RoIs door een CNN te voeren. Region-based Convolutional Neural Network (R-CNN) Girshick (2015) is het basis principe van de two-stage detectoren weergegeven in figuur 2.4. Hierbij wordt met een region proposal algoritme regio's uit de afbeelding gefilterd waar de kans groot is dat er objecten op staan. R-CNN bestaat uit 3 stappen:

1. Via een selective search algoritme Uijlings et al. (2013) worden er ongeveer 2000 mogelijke regio's met objecten geselecteerd.
2. Elke mogelijke regio wordt omgezet naar een feature vector via een CNN.
3. Elke regio wordt vervolgens geclassificeerd met een klas-specifieke SVMs.

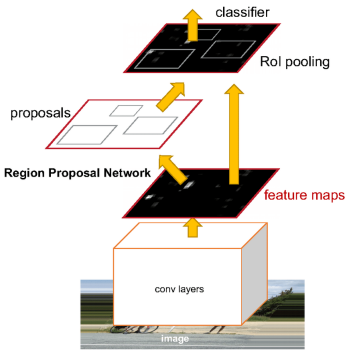


Figuur 2.4: R-CNN

R-CNN is een trage detector vermits elke RoIs door een CNN moet gaan. Deze methode is geëvolueerd tot de veel snellere methode Faster R-CNN Ren et al. (2016). Hierbij wordt de afbeelding door een CNN behandelt en vervolgens wordt er gebruik gemaakt van een Region Proposal Network (RPN) dit is weergegeven in figuur 2.5. Het RPN gaat zoals bij R-CNN regio's uit de afbeelding filteren waar de kans groot is dat er objecten opstaan. Maar het RPN werkt sneller en levert betere resultaten dan het region proposal algoritme.

Het RPN is een deep fully convolutional network dat per input een set van regio's als output geeft. Elk van deze regio's heeft een objectness score wat een maat is voor het object t.o.v. de achtergrond in de afbeelding. Om een region proposal te genereren wordt het RPN over de feature map geschoven die gegenereerd is door het voorgaande CNN. Op elke sliding window locatie worden er meerdere regio voorspellingen gedaan. Deze voorspelling wordt gedaan door verschillende anchor boxen in een sliding window te evalueren.

Vervolgens worden RoIs omgezet naar een feature vector met vaste lengte door RoI pooling. Elk van deze features gaat door een set van fully-connected lagen die 2 lagen als output heeft. een softmax laag die de klasse voorspelt, en een bounding box regressie laag die de bounding box voorspelt.



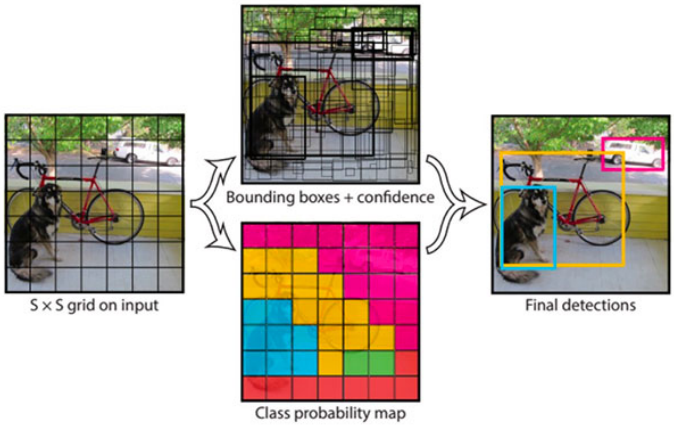
Figuur 2.5: Faster R-CNN

2.2.2 One-stage detector

Bij one-stage detectoren gebeurt object detectie in één keer met één neuraal netwerk. Dus er is geen region proposal niveau meer zoals bij de two-stage detector. Deze detector gebruiken minder geheugen en rekenkracht t.o.v. two-stage detectoren. One-stage detectoren zijn sneller dan two-stage detectoren omdat ze alles in één keer doen, maar kunnen wat in nauwkeurigheid verliezen t.o.v. two-stage detectoren. De twee veel gebruikte technieken van one-stage detectie zijn: You Only Look Once (YOLO) en Single Shot Detection (SSD).

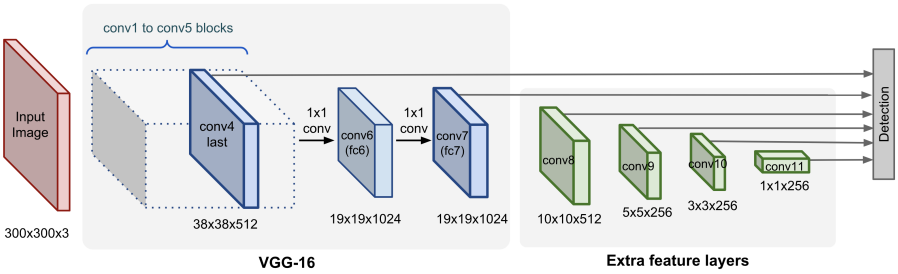
YOLO Redmon et al. (2016) verdeelt de afbeelding in een S x S rooster zoals in figuur 2.6 te zien is. De cel waarin het middelpunt van het object valt is verantwoordelijk voor de object detectie. Elke cel voorspelt K aantal bounding boxes en een score die aangeeft hoe zeker het model is dat een bepaalde bounding box een object bevat. Deze score wordt bepaalt met de Intersection Of Union (IOU) tussen de voorspelde box en de ground truth box. Vervolgens is er nog een methode nodig om de overbodige bounding boxen te verwijderen. Een eerste mogelijkheid is door enkel bounding boxen te tekenen waarvan de voorspelling boven een treshold ligt. Een andere methode is non-maxima supression deze methode zorgt ervoor dat elk object maar één bounding box heeft. Deze techniek houdt enkel de bounding box over met de beste voorspelling en onderdrukt de rest van de bounding boxen.

SSD Liu et al. (2016) is een one-stage detector (figuur 2.7) waarbij een afbeelding door verschil-



Figuur 2.6: YOLO waarbij de input is opgedeeld in een S x S rooster. En waarbij bounding box voorspellingen zijn gedaan.

lende convolutielagen gaat, wat als resultaat feature mappen op verschillende schalen oplevert. Op elke locatie van deze feature mappen van verschillende schalen wordt een vaste set van bounding boxen geëvalueerd. Voor elk van deze boxen wordt de zekerheid dat het een object bevat voorspelt. Op het einde wordt non maximum suppression gebruikt om de finale voorspelling te maken. Het netwerk van een SSD bestaat uit een basis netwerk dat gevormd wordt door een standaard classificatie netwerk zonder de fully-connected lagen. Vervolgens worden er extra convolutie lagen toegevoegd, wat het model toelaat om voorspellingen te doen op verschillende schalen.



Figuur 2.7: One-stage detector met VGG net backbone

Hoofdstuk 3

Herkenning en detectie implementatie op mobiel platform

Dit hoofdstuk zal gaan over het implementeren van deep learning herkeningssystemen en detectiesystemen op een mobiel platform. In dit hoofdstuk wordt er besproken welke technologiën er gebruikt kunnen worden om een CNN te implementeren op een mobiel platform. Dit gaan een aantal frameworks zijn die de programeur in staat stelt om een bestaand model te implementeren op een mobiel apparaat. Deze frameworks gaan vergeleken worden om te kijken van welk framework het best gebruik wordt gemaakt voor een bepaalde toepassing. Er wordt ook onderzocht hoe een bestaand herkeningssystemen en detectiesystemen geöptimalisserd kan worden zodat dit gebruikt kan worden op een mobiel platform. Bij het uitvoeren van een neuraal netwerk op een mobiel apparaat zal men rekening moeten houden met de volgende zaken: gelimiteerde rekenkracht en beschikbaar geheugen. Ook moet er rekening gehouden worden met een beperkte batterij, want CNN's gebruiken veel bandbreedte en voeren veel berekeningen uit wat meer energie verbruikt. Dus er zal onderzocht moeten worden welke methodes men kan gebruiken om het geheugen van het model, het aantal bewerkingen en het energieverbruik te kunnen verbeteren.

3.1 Van object detectie model naar een mobiele Implementatie

Om machine learning modelen te ontwerpen en te trainen kan er gebruik gemaakt worden van frameworks. Deze frameworks geven de programeur een set van tools die hun in staat stelt om op een overzichtelijke en flexibele manier machine learning modellen te ontwerpen en trainen. In deze paragraaf worden enkele van de meest gebruikte frameworks besproken. Dit zijn enkele van de meest gebruikte frameworks, maar zeker niet allemaal.

3.2 Frameworks

TensorFlow en PyTorch zijn de 2 voornaamste frameworks die gebruikt worden om neurale netwerken te ontwerpen en trainen. Veel van de tools en bibliotheken die gebruikt worden om herkeningssystemen en detectiesystemen te ontwerpen worden bovenop deze frameworks gebruikt.

3.2.1 TensorFlow

TensorFlow (Abadi et al. (2016)) is ontworpen door Google en is een open source library voor machine learning implementaties. TensorFlow focust op het trainen en het deployen van neurale netwerken. Ook ondersteund TensorFlow meerdere programeer talen zoals: Python, Java en C. Door de introductie van de Keras API is TensorFlow meer gebruiksvriendelijk geworden. Keras is een framework dat bovenop TensorFlow gebruikt kan worden, waarmee machine learning modellen kunnen ontworpen worden op een overzichtelijke manier. Zo heeft TensorFlow een gebruiksvriendelijk API voor eenvoudige projecten en meer uitgebreide tools voor complexe projecten. Door gebruik te maken van TensorBoard kan data op een flexibele manier gevisualiseerd worden tijdens runtime. TensorFlow biedt ook goede ondersteuning op gebied van deployment van een productie model. Nog een voordeel van TensorFlow is dat het een grootte community achter zich heeft, omdat dit een veel gebruikt framework is.

3.2.2 PyTorch

PyTorch (Li et al. (2020)) is ontworpen door facebook en wordt zoals TensorFlow ook gebruikt voor machine learning implementaties. Dit is een python gebaseerd framework dat focust op flexibiliteit, maar deze extra flexibiliteit zorgt voor meer lijnen code. Door zijn flexibiliteit is het gemakkelijk om nieuwe functionaliteiten toe te voegen door bestaande code aan te passen of nieuwe code toe te voegen. PyTorch maakt gebruik van externe tools zoals TensorBoard om data te visualiseren. Dit framework wordt gebruikt om een machine learning model te ontwerpen en trainen. Het model dat is ontworpen kan vervolgens gebruikt worden gebruikt om een applicatie te ontwerpen.

PyTorch wordt voornamelijk gebruikt om te experimenteren met neurale netwerken. Bedrijven en onderzoekers gebruiken dit framework vooral om een CNN experimenteel op te bouwen en te trainen. TensorFlow wordt voornamelijk gebruikt om het model effectief in gebruik te nemen.

3.3 Object detectie bibliotheken

Er zijn heel wat bibliotheken die de programeur kan importeren. Deze bibliotheken geven de programeur extra tools en hulpmiddelen om een detectiesysteem te ontwerpen en te trainen.

3.3.1 MMDetection

MMDetection maakt deel uit van OpenMMLab en is een open source object detectie toolbox gebaseerd op PyTorch. De toolbox bevat gewichten van meer dan 200 voorgetrainde modellen. Via modulair ontwerp kan het detectie framework opgesplitst worden in verschillende componenten. Met de verschillende componenten kan een eigen detectie model worden gemaakt door de verschillende componenten te combineren. MMDetection ondersteund ook 48 verschillende detectie methodes zoals: YOLO, Faster R-CNN, etc. Al de bounding box en masker operaties worden uitgevoerd op GPU's dit geeft MMDetectie een grootte trainingssnelheid. MMDetection biedt geen ondersteuning om een bestaand model te optimaliseren naar een model voor mobiele toepassingen. Dus dit model zal geconverteerd moeten worden naar een framework waarbij het optimaliseren naar een mobiel model wel mogelijk is.

3.3.2 Detectron2

Detectron2 is een bibliotheek van Facebook die segmentatie en detectie algoritmes ondersteunt. Zoals MMDetection werkt Detectron2 bovenop Pytorch en kan het netwerk getraind worden op 1 of meerdere GPU's. Via Modular, extensible design kan Detectron2 specifieke modules toevoegen aan bijna elk deel van een object detectiesysteem. Detectron2 bevat meer dan 80 voorgetrainde modellen waarop de programeur verder kan bouwen. Voor object detectie ondersteund Detectron2 6 verschillende standaard modellen.

3.3.3 Darkflow

3.3.4 ImageAI

ImageAI is makelijk te gebruiken Python bibliotheek die de programeur in staat stelt om State-of-the-art AI feautres te implementeren. ImageAI ondersteund object detectie door gebruik te maken van RetinaNet, YOLOv3 en TinyYOLOv3 getraind met de COCO dataset. Deze bibliotheek werkt sinds juni 2021 bovenop PyTorch, hiervoor werkte ImageAI bovenop TensorFlow. Ook deze bibliotheek geeft de programeur de mogelijkheid om nieuwe modellen te trainen om specifieke objecten te detecteren.

3.4 Frameworks voor mobiele implementatie

Er zijn een aantal frameworks die de programeur de mogelijkheid geven om een model te optilmaliseren voor een mobiel platform. Er zal voornamelijk gefocust worden op Android implementaties. Niet elke framework voor mobiele implementatie optimaliseert het model op dezelfde manier. Zo zullen sommige modellen na het optilmaliseren in een bepaald framework een kleinere bestands grootte hebben dan bij andere frameworks. Of sommige frameworks zullen beter optimaliseren op gebied van latency of accuraatheid dan andere frameworks.

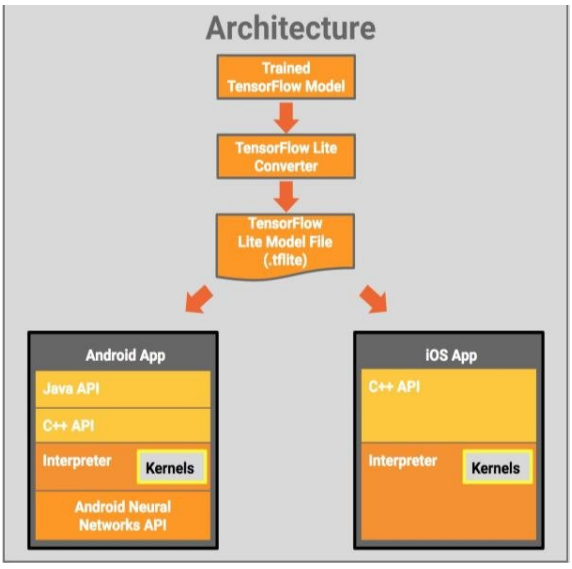
3.4.1 CoreML

Core ML is het Apple framework om machine learning tools te integreren in een applicatie. Dit kan een model zijn van Create ML het machine learning framework van Apple zelf. Maar Core ML biedt ondersteuning om modellen te converteren van TensorFlow, PyTorch en ONNX naar Core ML. Uiteraard is dit framework enkel van toepassing voor Apple, en in deze masterproef wordt er vooral gefocust op een Android implementatie.

3.4.2 Mobile AI Compute Engine (MACE)

Het MACE framework is ontworpen door Xaomi en dient specifiek voor mobiele toepassingen van neurale netwerken op Android, IOS, Linux en Windows. MACE biedt ondersteuning voor verschillende frameworks zoals: TensorFlow, Caffe en ONNX. Het model kan geïmplementeerd worden op Android, IOS en Linux. Het MACE framework bespaart geheugen door de core library zo klein mogelijk te maken door het aantal externe dependencies te minimaliseren. Het Winograd algoritme ... wordt gebruikt om convolutie bewerkingen te versnellen en verbeterd op deze manier de latency van het CNN model.

3.4.3 TensorFlow Lite



Figuur 3.1: Implementatie flow van een getraind TensorFlow model naar de applicatie

TensorFlow Lite is het antwoord van TensorFlow om CNN modellen te optimaliseren voor mobiel gebruik (figuur: 3.1). TensorFlow Lite zorgt ervoor dat het model een lage latency heeft en een kleine binaire grootte. Het converteren kan makkelijk worden uitgevoerd via de volgende lijnen code in Python.

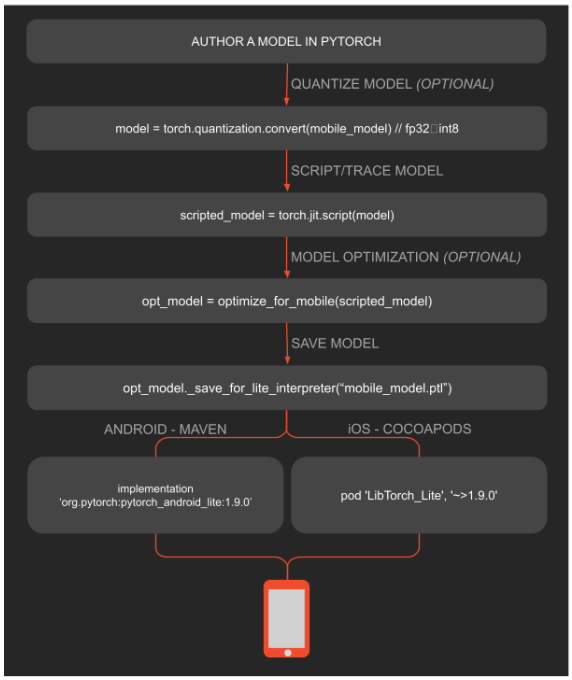
Listing 3.1: Converteren van TensorFlow naar een TensorFlow Lite model

```
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
tflite_model = converter.convert()
```

Het TensorFlow Lite framework geeft ook de mogelijkheid om verdere optimalisaties zoals pruning en quantisatie uit te voeren na het converteren. Het TensorFlow Lite model is ook compatibel met android en IOS. Bij Android kan TensorFlow gebruik maken van de Neural Network API (NNAPI) ... die beschikbaar is vanaf Android 8.1. Deze API kan gebruikt worden om netwerk modellen te versnellen met de GPU, DSP en NPU.

3.4.4 PyTorch Mobile



Figuur 3.2: Implementatie flow van een getraind PyTorch model naar de applicatie met code

PyTorch biedt zoals TensorFlow ook de mogelijk om het model te optimaliseren naar PyTorch Mobile. In figuur: 3.2 is te zien welke lijnen code er moeten worden uitgevoerd om een PyTorch model te converteren naar een PyTorch mobile model. PyTorch mobile zit nog in zijn beta fase, dus hierbij kunnen er onverwachte complicaties optreden. Een ander framework is Caffe2 dit framework focust ook op mobiele implementaties, maar het framework is ondertussen geïntrigeerd met PyTorch. Ook is het PyTorch framework compatibel met android en IOS.

In de paper geschreven door Luo et al. (2020) wordt PyTorch Mobile vergeleken met Tenserfolow Lite voor verschillende netwerk architecturen en een model getraind met dezelfde data. Voor alle netwerk architecturen in deze paper geeft de optimalisatie naar TensorFlow Lite de kleinste bestand grootte van het model. Uit deze paper is ook af te leiden dat de optimalisatie naar een mobiel

model niet alleen afhankelijk is van het framework maar ook van de netwerk architectuur. Zo geeft TensorFlow Lite volgens Luo et al. (2020) betere latency resultaten voor de zwaardere netwerken (ResNet50, InceptionV3, DenseNet121) dan PyTorch Mobile. Maar PyTorch Mobile heeft op zijn beurt wel een betere latency voor SqueezeNet en MobileNetV2. Dus uit deze paper kunnen we afleiden dat TensorFlow Lite het beste de bestands grootte verkleint, maar dat de netwerk architectuur ook een rol speelt.

Febvay (2020) vergelijkt TensorFlow Lite met MACE voor verschillende neurale netwerken (SqueezeNet, MobileNetV1/V2). Hierbij geeft TensorFlow Lite het beste resultaat, TensorFlow Lite gaf een Top-1 resultaat van 69,19% en MACE gaf een top-1 resultaat van 66.84% bij MobileNetV1. Ook voor de latency gaf TensorFlow Lite in de meeste gevallen de beste resultaten buiten bij het gebruik van 4 of 6 CPU cores, dan gaf MACE betere resultaten.

3.5 Converteren naar framework voor mobiele implementatie

Het doel van deze masterproef is om een bestaand netwerk op een mobiel platform te krijgen. Dus zullen de object detectoren die ontworpen zijn met bovenstaande tools (Detectron2, MMDetection, ImagAI en Darkflow), geconverteerd moeten worden naar een framework voor mobiele implementatie. Object detectoren zijn complexe systemen waarbij het converteren naar een ander framework complex of zelfs niet mogelijk zal worden. In deze paragraaf zal per objectdetector bibliotheek gekeken worden wat de mogelijkheden zijn om van een detector model naar een mobiele implementatie te gaan. De eerste stap zal zijn om te kijken welke mogelijkheden er zijn zonder te converteren naar een ander framework. Een tweede stap is via Open Neural Network Exchange (ONNX) het huidige detectie model converteren naar een andere framework. Een derde stap is verder zoeken naar een alternatief als de eerste twee methodes niet lukken.

3.5.1 ONNX

ONNX (2017) biedt de mogelijkheid om verschillende tools/frameworks samen te laten werken. Hierbij wordt een bestaand model geconverteerd naar een ONNX model, en dit model kan op zijn beurt geconverteerd worden naar het gewilde framework. ONNX ondersteund niet elk machine learning framework, maar toch wel de meest bekende. Het ONNX framework biedt zelf ook de mogelijkheid om een model te deployen en te optimaliseren voor mobiel gebruik. ONNX dient goed als overkoepelend framework dat er voor zorgt dat verschillende frameworks compatibel zijn met elkaar. Volgens de website zijn er 23 frameworks die naar het ONNX framework geconverteerd kunnen worden en een CNN kunnen ontwerpen en trainen. Om dan geconverteerd te worden naar een framework dat het model kan omzetten naar een mobile versie. Een ander manieren om een model te gebruiken over verschillende frameworks in plaats van ONNX hangt af van de compatibiliteit tussen frameworks. Bepaalde frameworks zullen zelf een methode hebben om modellen van andere frameworks in te laden.

3.5.2 Van MMDetection naar detectie model voor mobile implementatie

Voor het testen van MMDetection nemen we de Kitty dataset ... waarmee we een detector trainen via transfer learning, hiervoor gebruiken we de Mask-rcnn detector. Vermits MMDetection bovenop PyTorch werkt gaan we eerst proberen om met PyTorch Mobile te werken, dit gaat via de volgende lijnen code.

Listing 3.2: Converteren van MMDetection naar een PyTorch Mobile

```
import torch
import torchvision
from torch.utils.mobile_optimizer import optimize_for_mobile

model.eval()
example = torch.rand(1, 3, 224, 224)
traced_script_module = torch.jit.trace(model, example)
traced_script_module_optimized = optimize_for_mobile(traced_script_module)
traced_script_module_optimized._save_for_lite_interpreter("model.ptl")
```

Voordat het model kan geoptimaliseerd worden moet het python afhankelijk model worden omgezet in TorchScript. Deze TorchScript module kan dan verder geoptimaliseerd worden voor mobiel gebruik. Het omzetten naar de scriptmodule geeft een TypeError fout waardoor het optimaliseren voor mobiel niet lukt.

Een andere manier om naar een mobiele implementatie te gaan is door het model eerst om te zetten naar het .onnx formaat. MMDetection ondersteund de conversie naar ONNX, maar dit zit nog in zijn experimentele fase. In de documentatie van MMDetection kan er een lijst gevonden worden met detectiemodellen die ondersteuning hebben voor het exporteren naar ONNX ... . Ook heeft ONNX 'opsets' met daarin de operaties die ONNX ondersteund, niet elk framework past dezelfde opset versie toe wat later voor problemen kan zorgen. Elke bibliotheek/framework moet dezelfde 'opset' implementeren anders zullen er operaties zijn die niet compatibel zijn met het gewilde framework voor mobiele implementatie. Via de volgende lijn code is het mogelijk om het MMDetection model om te zetten naar een ONNX model.

Listing 3.3: Converteren van MMDetection naar een onnx bestand

```
!python ./tools/deployment/pytorch2onnx.py <config_file> <checkpiont_file>
        --output-file <output file>
```

pytorch2onnx.py is het MMDetection script om een model te converteren naar ONNX formaat. De config file is het bestand dat het neuraal netwerk beschrijft. En de checkpoint file is een file die tijdens het trainen wordt aangemaakt als checkpoint. Het finale model is vaak terug te vinden als latest.pth, dit is het laatste checkpoint dat na het trainen wordt aangemaakt. Op het einde van deze lijn code is het mogelijk om nog extra opties toe te voegen die in de MMDetection documentatie ... terug te vinden zijn. Bovenstaande lijn code converteert het MMDetection model succesvol naar een ONNX model. Wel moet er bij vermeld worden dat dit MMDetection model een éénvoudig

model is waarbij geen speciale aanpassingen zijn gedaan. Dus bij complexere modellen zou het resultaat anders kunnen zijn.

Het gegenereerde .onnx model is vervolgens geconverteerd naar een TensorFlow Lite model via de volgende lijnen code.

Listing 3.4: Converteren van ONNX bestand naar een TensorFlow Lite model

```
import tensorflow as tf
import onnx
from onnx_tf.backend import prepare

#ONNX model inladen
onnx_model = onnx.load("model.onnx")  # inladen onnx model
output = prepare(onnx_model)
output.export_graph('tf_model.pb') # model exporteren naar TensorFlow model

#Ingeladen model omzetten naar TensorFlow Lite model
converter = tf.lite.TFLiteConverter.from_saved_model('tf_model.pb')
converter.target_spec.supported_ops = [
        tf.lite.OpsSet.TFLITE_BUILTINS, # enable TensorFlow Lite ops.
        tf.lite.OpsSet.SELECT_TF_OPS # enable TensorFlow ops.
]
tflite_model = converter.convert() # converteer model
```

Eerst moet het ONNX model ingeladen worden als een standaard TensorFlow model. Vervolgens moest bij het converteren naar een TensorFlow Lite model eerst vermeld worden welke opsets er ondersteund moeten worden. Het effectief converteren naar een TensorFlow Lite model duurt een hele tijd .... .

Om een model te converteren naar CoreML zijn er 3 mogelijkheden. Eerst is rechtstreeks converteren vanuit PyTorch, daarvoor moeten we eerst ons model omzetten in een torchscript. Maar dan stoot men op hetzelfde probleem als bij de PyTorch Mobile implementatie, waarbij het omzetten naar torchscript een TypeError geeft. De tweede manier is via ONNX maar dat raad Apple af omdat dit in de volgende versie van CoreML niet meer ondersteund wordt, CoreML biedt ook maar ondersteuning tot en met opset versie 10. Vanuit ONNX converteren gaat via de volgende lijnen code, maar vermits dit in later versies niet meer ondersteund wordt is deze methode niet uitgetest.

Listing 3.5: Converteren van ONNX bestand naar een CoreML model

```
import coremltools as ct

model  = ct.converters.onnx.convert(model='my_model.onnx')
```

De derde manier is rechtstreeks via TensorFlow convert, maar dit is een vrij omslachtige manier omdat we dan de volgende converties moeten maken MMDetection -¿ ONNX -¿ TensorFlow -¿ CoreML.

3.5.3 Van Detectron2 naar mobile implementatie

3.5.4 Van ImageAI naar mobile implementatie

3.5.5 Van DarkFlow naar mobile implementatie

3.6 Van een object herkening model naar een mobiele implementatie

3.7 Implementatie op mobiele platformen

Uit paragraaf 3.1 kan er afgeleidt worden dat sommige frameworks ondersteuning bieden voor het optimaliseren van het machine learning model naar een lichtere versie. Dus het bestaande CNN model zou naar één van deze frameworks geconverteerd moeten worden zodat er gebruik gemaakt kan worden van de mobiele optimalisatie die dat framework ondersteund. Niet elk framework zal even compatibel zijn met het bestaande model, er zal dus gekeken moeten worden tussen welke frameworks een conversie mogelijk is. Ook zal niet elk framework de optimalisatie voor mobiele platformen op dezelfde manier toepassen. Dus het converteren van het standaard model naar het mobiele model zal voor elk framework een ander resultaat geven. In deze paragraaf zal er besproken worden welke frameworks er compatibel zijn met elkaar en welke frameworks het beste optimaliseren voor een mobile implementatie.

We hebben al gezien dat voor het converteren van een model niet veel lijnen code nodig zijn. Maar dit is enkel het geval als het model in het zelfde framework is ontworpen. Als het model in een ander framework is ontworpen en getraind is de eerste stap van het process om het CNN model naar het gewilde framework te converteren.

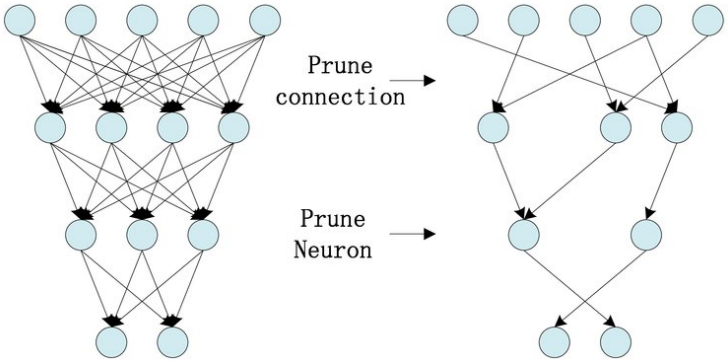
3.8 Optimalisaties van neurale netwerken voor snelheid en bestandgrootte

In deze paragraaf wordt er onderzocht welke optimalisaties er kunnen worden toegepast om de accuraatheid, snelheid en gebruikt geheugen te verbeteren. Maar het optimaliseren van een bepaalde factor zal vaak negatieve gevolgen hebben voor een andere factor, dit zal meestal de accuraatheid zijn. Dus er zal een goede belans gevonden moeten worden tussen de optimalisatie en de negatieve gevolgen op de andere factoren.

3.8.1 Pruning

Pruning is de eerste stap van de Deep comression methode voorgesteld door Han et al. (2016). Bij het trainen van een CNN hebben bepaalde gewichten een grotere invloed op het resultaat. Andere gewichten hebben weinig tot geen invloed op het resultaat. Maar alle gewichten worden steeds berekend ongeacht hun invloed op het resultaat. Bij pruning worden de gewichten met een kleine invloed op het resultaat verwijderd dit is weergegeven op figuur 3.3. Waardoor er geen

berekeningen meer moeten uitgevoerd worden voor de verwijderde gewichten. Deze parameters moeten dan ook niet meer worden bijgehouden waardoor het CNN model minder geheugen in beslag neemt. Eerst wordt het CNN op een normale manier getraind zonder pruning. Vervolgens worden al de kleine gewichten onder een bepaalde treshold verwijderd. volgens Han et al. (2016) wordt voor VGG-16 het aantal parameters met factor 13 verminderd, voor AlexNet met een factor 9. Deze methode heeft zeer weinig tot geen effect op de accuraatheid.



Figuur 3.3: CNN voor en na pruning

3.8.2 Parameter quantisatie

Het quantiseren en delen van gewichten is een tweede methode voorgesteld door Han et al. (2016). Een CNN bestaat uit miljoenen gewichten, en de waarde van elke van deze gewichten moeten op het systeem worden opgeslagen. De default representatie van een waarde wordt opgeslagen als een floating point nummer wat 4 bytes in beslag neemt. Dus voor miljoenen parameters hebben de gewichten veel schijfruimte nodig. Een mogelijke oplossing hiervoor is quantiseren van gewichten, waarbij de getal representatie van de gewichten wordt verandert naar fixed point. Hierbij worden de waarden van gewichten beperkt tot een set van beschikbare waardes. Waarbij de waardes ëënmalig worden opgeslagen en al de gewichten refereren naar een waarde van de vaste set met waardes. Hoe kleiner de set met waardes is hoe minder geheugen er in beslag wordt genomen, maar een kleinere set van waardes zorgt ook voor een mindere accuratie. Dus de grootte van de set moet goed worden gekozen zodat er niet te veel geheugen wordt gebruikt met een accepteerbare daling in accuratie. Han et al. (2016) past vervolgens Huffman encoding toe die een compressie uitvoert op de gekwantiseerde parameters.

3.8.3 Convolutionele filter compressie

een andere methode voorgesteld is Compressed Convolutional Filters. Hierbij wordt de grootte van de kernel verkleind om het aantal parameters en rekenwerk te verminderen. Maar door de kernels te verkleinen daalt de accuraatheid van het CNN.

3.8.4 Matrix factorisatie

Hierbij worden grootte en complexe matrices opgesplitst in verschillende kleinere en simpelere matrices.

Hoofdstuk 4

Richtlijnen voor formules

Hoofdstuk 5

Richtlijnen voor referenties

Bibliografie

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *arXiv:1605.08695 [cs]*. arXiv: 1605.08695.

Febvay, M. (2020). Low-level Optimizations for Faster Mobile Deep Learning Inference Frameworks. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 4738–4742, Seattle WA USA. ACM.

Girshick, R. (2015). Fast R-CNN. *arXiv:1504.08083 [cs]*. arXiv: 1504.08083.

Han, S., Mao, H., and Dally, W. J. (2016). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*. arXiv: 1510.00149.

Jiang, X., Hadid, A., Pang, Y., Granger, E., and Feng, X. (2019). *Deep Learning in Object Detection and Recognition, edited by Xiaoyue Jiang, Abdenour Hadid, Yanwei Pang, Eric Granger, Xiaoyi Feng.* Springer Singapore : Imprint: Springer, Singapore, 1st ed. 2019. edition.

Koehrsen, W. (2018). Neural Network Embeddings Explained.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., and Chintala, S. (2020). PyTorch distributed: experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13(12):3005–3018.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. (2016). SSD: Single Shot MultiBox Detector. volume 9905, pages 21–37.

Luo, C., He, X., Zhan, J., Wang, L., Gao, W., and Dai, J. (2020). Comparison and Benchmarking of AI Models and Frameworks on Mobile Devices. *arXiv:2005.05085 [cs, eess]*. arXiv: 2005.05085.

ONNX (2017). ONNX Tutorials. original-date: 2017-11-15T18:59:18Z.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. ISSN: 1063-6919.

Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*. arXiv: 1506.01497.

Uijlings, J. R., R, van de Sande, K. E., A, Gevers, T., Smeulders, A. W., and M (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171. Num Pages: 154-171 Place: New York, Netherlands Publisher: Springer Nature B.V.

Bijlage A

Uitleg over de appendices

Bijlagen worden bij voorkeur enkel elektronisch ter beschikking gesteld. Indien essentieel kunnen in overleg met de promotor bijlagen in de scriptie opgenomen worden of als apart boekdeel voorzien worden.

Er wordt wel steeds een lijst met vermelding van alle bijlagen opgenomen in de scriptie. Bijlagen worden genummerd het een drukletter A, B, C,...

Voorbeelden van bijlagen:

Bijlage A: Detailtekeningen van de proefopstelling

Bijlage B: Meetgegevens (op USB)

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS DE NAYER SINT-KATELIJNE-WAVER
J. De Nayerlaan 5
2860 SINT-KATELIJNE-WAVER, België
tel. + 32 15 31 69 44
iiw.denayer@kuleuven.be
www.iiw.kuleuven.be