Professionele Bachelor Toegepaste Informatica

Smart Pooltable: Deep learning en reinforcement learning vergelijken

Yente Martens

Promotoren:

Jeroen Benats
Arno Barzan

Bewire
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Smart Pooltable: Deep learning en reinforcement learning vergelijken

Yente Martens

Promotoren:

Jeroen Benats
Arno Barzan

Bewire
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Eerst en vooral wil ik graag PXL bedanken voor de kennis die ze mij bijgebracht hebben en de ondersteuning die ze mij de afgelopen jaren gegeven hebben. Ik heb veel bijgeleerd tijdens mijn opleiding maar tijdens deze stage heb ik nog veel meer en sneller kunnen leren.

Bewire verdient ook een woord van dank. Eerst en vooral een dankwoord voor mijn stagepromotor Jeroen Benats om de kansen en het vertrouwen die mij gegeven zijn. Jens Mortier die de ondersteuning en check ups doorheen de gehele stage op zich nam. Jonas Liekens van C4J en Anissa Schirock van Trase verdienen ook een woord van dank voor hun technische expertise en hulp tijdens deze stage. Jonas verdient nog een extra woord van dank voor het lenen van zijn persoonlijke computer.

Mijn Teamgenoten Dary Schaeken en Timo Biesmans verdienen ook een dank u wel voor de hulp en tips doorheen de gehele stage. Ook de andere stagiairs bij Bewire die tips gaven doorheen de stage verdienen een dank u.

Alle medewerkers van Bewire die me hulp gaven en hun eerlijke mening zeiden ook een dank u.

Ten slotte wil ik mijn familie, vriendin en vrienden bedanken voor hun steun tijdens mijn opleiding en stage.

Abstract

Vandaag de dag wordt AI overal op toegepast. In deze paper zullen zowel *deep learning*, in de vorm van objectdetectie als *reinforcement learning*, in de vorm van een *agent* die het spel pool leert spelen aan bod komen.

Bij het onderzoek zal een vergelijking gemaakt worden tussen *deep learning* en *reinforcement learning.* Er zal rekening gehouden worden met de voor- en nadelen alsook de toepassingsgebieden en het gebruiksgemak.

Uit het onderzoek blijkt dat vooral de manier van zowel aanleren als trainen verschilt. *Reinforcement learning* is moeilijker toe te passen en op te stellen dan *deep learning*. Uit het onderzoek is ook gebleken dat *deep reinforcement learning* ook een mogelijke oplossing is als *reinforcement learning* niet werkt.

Deze bachelorproef legt van de grond af aan de concepten van *reinforcement learning* en *deep learning* uit. Verschillende manieren van leren alsook de neurale netwerken zullen uitgelegd worden. Tot slot zullen enkele *policy gradient* algoritmes vergeleken worden.

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte vergelijkingen

Lijst van gebruikte afkortingen

ACKTR	Actor-Critic using Kronecker-Factored Trust Region
AI	Artificiële Intelligentie
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network Library
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DNN	Deconvolutional Neural Network
DRL	Deep Reinforcement Learning
FFNN	Feed Forward Neural Network
FPS	Frames Per Second
GAN	General Adverserial Network
GPU	Graphics Processing Unit
GTK	Gimp Toolkit
GUI	Graphical user interface
ML	Machine Learning
MNN	Modular Neural Network
OpenCV	Open Source Computer Vision Library
PPO	Proximal Policy Optimization
RAM	Random Access Memory
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SSL	Semi-Supervised Learning
TRPO	Trust Region Policy Optimization
VoTT	Visual object Tagging Tool
YOLO	You Only Look Once

Inleiding

Artificiële intelligentie is vandaag de dag een groot deel van ons leven zonder dat de meeste mensen erbij stilstaan. Het is een geweldige uitvinding en kan bijna overal toegepast worden. Maar vandaag de dag wordt het toegepast op alles wat mogelijk is, zelfs waar het niet nodig is. Dit komt doordat er een hype gecreëerd wordt rond AI.

Dit project dient als een manier om te achterhalen hoe gemakkelijk *deep learning* en *reinforcement learning* toepasbaar zijn en om onderzoek te doen naar de werking en de mogelijkheden van zowel *reinforcement learning* als *deep learning*. Na dit project heeft Bewire een idee van de toepassingen en mogelijkheden van AI, *computervision,* en RL.

In dit document worden ten eerste de probleemstelling en doelstelling besproken. Vervolgens zullen de gebruikte technologieën beschreven worden. Om de voorstelling van de stageopdracht af te ronden wordt de nodige hardware ook even aangehaald.

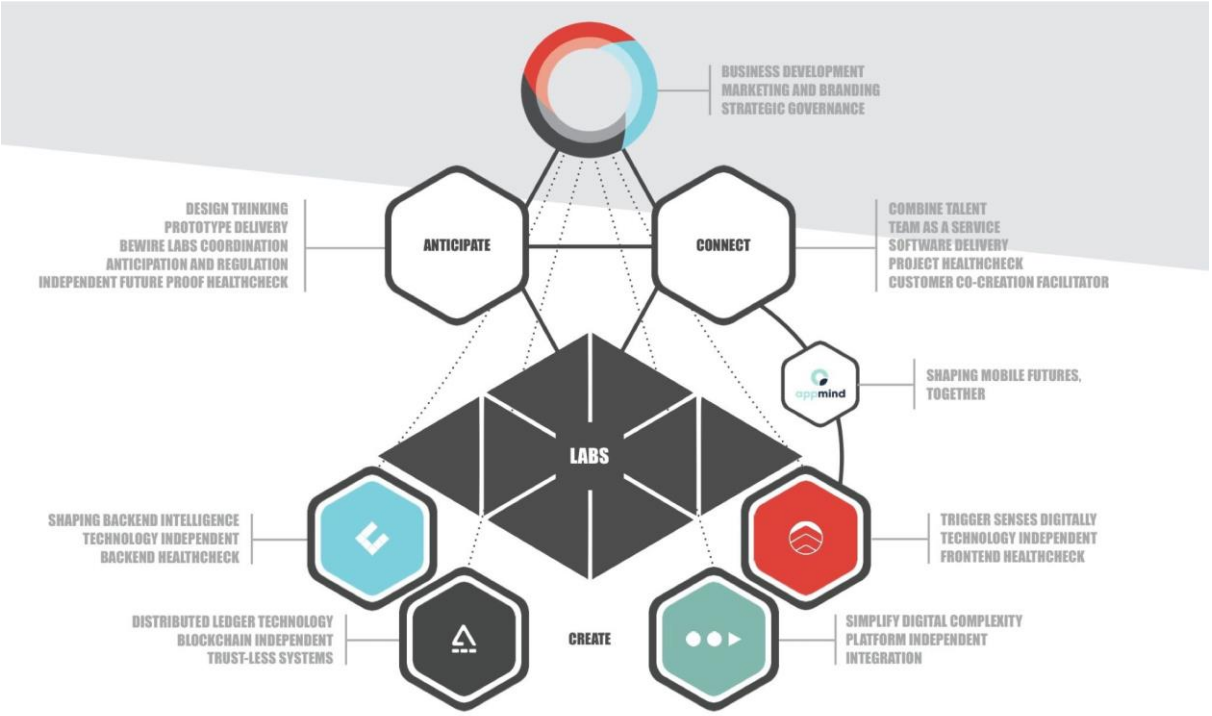
Ten tweede zal er een korte installatiegids staan en zal er een kijkje genomen worden naar de uitwerking en de resultaten van de stageopdracht.

Het derde topic omvat het onderzoek van deze stage, hierin zal DL vergeleken worden met RL om zo een beslissing te vormen wat wanneer toepasbaar is en hoe dit net gebeurt.

I. Stageverslag

Bedrijfsvoorstelling

Bewire is een bedrijf gevestigd in Herentals. Ontstaan vanuit C4J, een bedrijf dat vooral Javaproducten ontwikkelde. Dit was nodig doordat C4J uitgegroeid was tot een bedrijf dat ook frontend ontwikkeling en integratie kon aanbieden. Bij de komst van Bewire werden deze onderdelen opgesplitst en kregen ze hun eigen naam. C4J staat nog steeds in voor backend ontwikkeling. Evance houdt de frontend voor zijn rekening en Dots & Arrows staat in voor de integratie. Er zijn ondertussen al verscheidene sub-initiatieven ontstaan zoals Codrigo, Trase en Appmind. Codrigo is de software *factory* die bijstaat tijdens de volledige ontwikkeling van een applicatie. Trase specialiseert zich vooral in blockchain. Tot slot is er Appmind die zich toelegt op het design en de ontwikkeling van mobiele applicaties. Bewire en al zijn subdivisies zijn te zien in Figuur 1.



Figuur 1: Organigram Bewire

1.1 Evance

Evance is deel van Bewire NV en zorgt voor professionele frontend oplossingen op Enterprise niveau, zowel voor web als mobiel. Evance zorgt niet enkel voor een geslaagd project maar ook om een frontend dat gebruiksvriendelijk is zodat de klant een goed gevoel krijgt bij zijn/haar applicatie of website. Naast de ontwikkeling en de analyse wordt er ook nog aan consultancy en training gedaan. Om te ontwikkelen wordt gebruik gemaakt van frameworks zoals AngularJS, HTML5 en CSS3 [1].

1.2 C4J

Het Bewire dat iedereen kent heette voorheen C4J. Door meer en meer vraag naar frontend toepassingen is Bewire ontstaan met verschillende bedrijven die elk hun eigen specialiteit hebben. Door gebrek aan een deelbedrijf dat zich bezig houdt met innovatie sluit de pooltafel stage het meest aan bij C4J. C4J voorziet de backend in zowel web, mobiele als Cloud applicaties. Door de uitgebreide java-consulting en training zorgt C4J keer op keer opnieuw voor de perfecte backend voor alle applicaties. C4J doet net zoals Evance aan *coaching on the job* en consultancy [2].

1.3 Dots & Arrows

Dots & Arrows zorgt voor de integratie van verschillende applicaties. Beter uitgelegd wordt ervoor gezorgd dat verschillende applicaties met elkaar kunnen praten. De oplossingen zijn gebaseerd op Mulesoft, een integratieplatform om SaaS te koppelen met Enterprise applicaties [3].

1.4 Trase

Trase is Bewire's specialist in blockchain en Smart Contracts. Trase verricht verschillende activiteiten. De eerste is natuurlijk consultancy. De tweede is de Trase University. Hier wordt naast een opleiding te geven rond blockchain ook onderzoek gedaan rond gedecentraliseerde technologie [4].

1.5 Codrigo

Ten slotte hebben is er Codrigo. Codrigo zorg voor de *software delivery.* Er zijn verschillende doelen zoals *Agile Project Management, Enterprise Software Solutions, Flexible Sourcing Models, Global Cloud Infrastructure, Hybrid & Native Mobile Apps* en *Evolutionary designs.* Net zoals de andere deelbedrijven van Bewire doet Codrigo aan consultancy [5].

1.6 Appmind

Appmind zorgt voor de ontwikkeling van mobiele applicaties. Ze maken de applicatie van het begin tot het einde. Dit wil zeggen van het initiële idee van de klant, tot een *proof of concept,* tot de uiteindelijke oplevering van een applicatie. Dit gebeurt natuurlijk in nauwe samenspraak met de klant [6].

1.7 Anticipate

Anticipate was voorheen 1 van de principes waar Bewire rond draaide. Sinds kort is Anticipate een bedrijf op zich geworden. Anticipate bekijkt nieuwe technologieën en maakt hier *proof of concepts* van. Deze *proof of concepts* worden dan mogelijk geïntegreerd in bestaande projecten om zo de mogelijkheden te bekijken.

Voorstelling stageopdracht

1.8 Probleemstelling

Op het hoofdkantoor van Bewire staan een pooltafel waar alle werknemers tijdens de middagpauze en na de uren kunnen spelen. Niet iedereen is mee met alle regels van pool. Worden de caféregels of officiële regels gebruikt, was de vorige zet nu een fout of niet, is de zwarte bal wel juist gespeeld, wat als een persoon niet kan poolen maar toch eens wil spelen enzovoort. Allemaal mogelijke vragen die wel eens voorkomen tijdens een spelletje pool.

1.9 Doelstellingen

In de eerste fase van de stage wordt met behulp van *computer vision* geprobeerd om de locatie van alle ballen te vinden en deze te kunnen volgen als ze bewegen of binnen gespeeld worden. Boven de pooltafel is een camera aangebracht die aangesloten is op een computer waar YOLOv3 op draait voor de objectherkenning.

Tijdens de tweede fase van de stage wordt een spel pool gevolgd. De camera stuurt de beelden door naar de objectherkenning. Nadat de positie van de ballen bekend is volgt een applicatie het volledige spel van start tot einde. Punten worden toegekend voor het binnenspelen van een juiste bal. Ook wordt er rekening gehouden met de regels van het spel pool.

Tijdens de laatste fase van de stage wordt een model getraind, dat nieuwe spelers kan helpen door de beste zet te bepalen of manieren te verzinnen om de tegenstander te blokkeren.

1.10 Omgevingen en Technologieën

1.10.1 Ubuntu 18.04 lts

Allereerst moest er een besturingssysteem gekozen worden. De meest logische keuze hiervoor is een versie van Ubuntu. Bij het begin van de stage leek dit de 18.04-versie, omdat dit een *long term support*-versie is, wat wil zeggen dat er nog geregeld updates uitkomen om het besturingssysteem te verbeteren. Linux is een open source besturingssysteem. Er zijn verschillende varianten zoals Ubuntu, Linux mint, Fedora enzovoort. Deze worden distributies genoemd. Meestal zijn deze verschillende distributies gratis te verkrijgen.

1.10.2 Pop_OS!

Door de aankoop van het nieuwe *workstation* en één van de laatste nieuwe GPU's moest er gekozen worden voor een nieuwe versie van Linux, omdat de GPU niet herkent wordt in de 18.04-versie en dit een essentieel deel is tijdens de training van een *computer vision*-model. Een belangrijk punt van de 19.04-versie is de ondersteuning van de Nvidia RTX-kaarten. Er werd gekozen voor "Pop_OS!". "Pop_OS!" is een aangepaste versie van Linux. Het is gemaakt door System76. System76 is een bedrijf dat computers maakt met Linux op geïnstalleerd en ze maken ook besturingssystemen [7].

1.10.3 CUDA

CUDA of voluit *Compute Unified Device Architecture* is een technologie die gebruikt kan worden om programmeurs de kans te geven om hun applicaties te gebruiken met de kracht van een *graphics processing unit*. Bij deze stage is CUDA gebruikt om modellen te trainen met hoge snelheid [8].

1.10.4 CuDNN

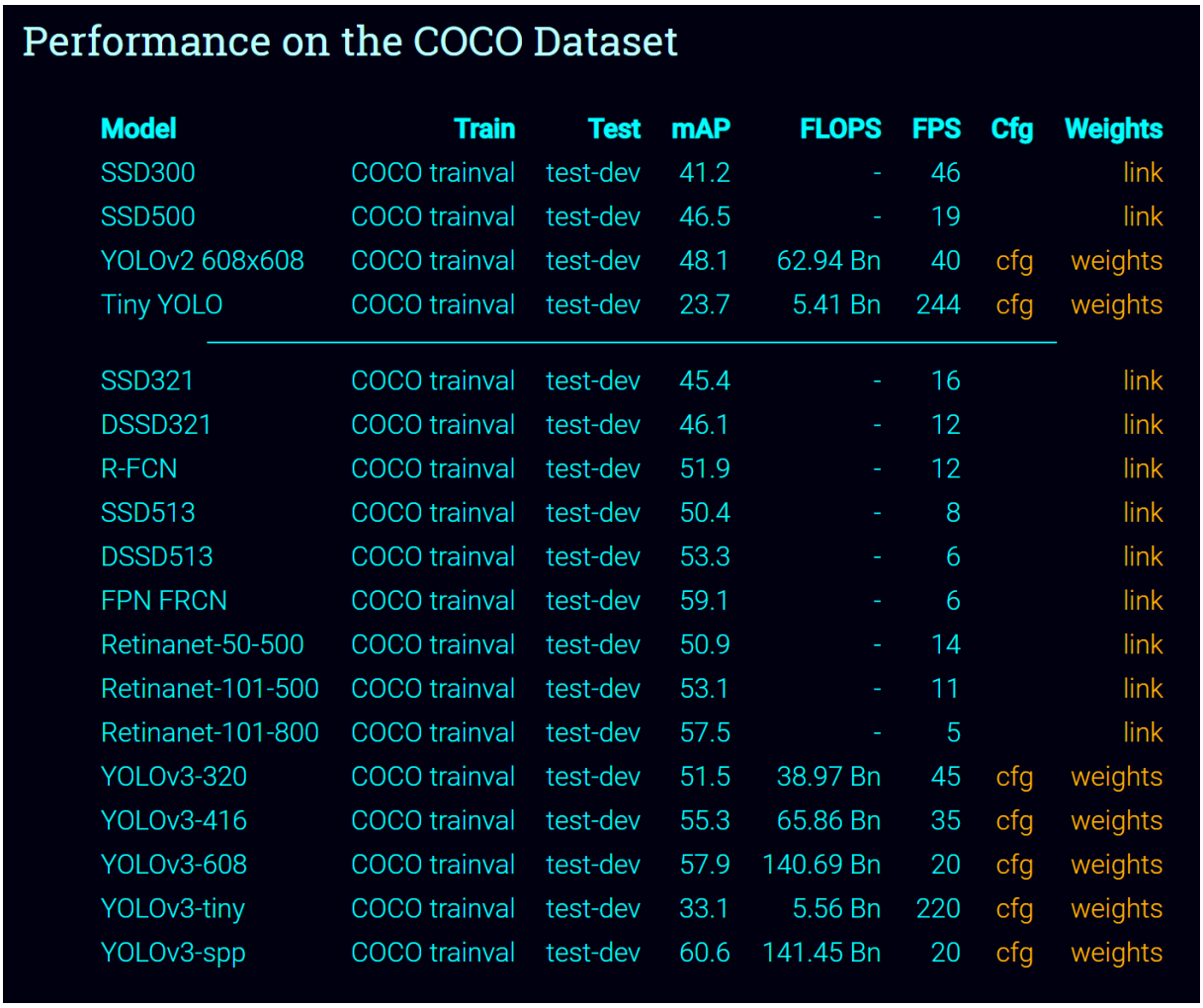
CuDNN of voluit geschreven *CUDA deep neural network library* is een GPU-geaccelereerde bibliotheek om bij *deep neural networks* te gebruiken die gemaakt zijn met gebruik van CUDA. CuDNN voorziet optimalisatie voor de meest voorkomende acties in *deep neural networks* [9].

1.10.5 OpenCV

OpenCV, of *Open Source Computer Vision Library* is een bibliotheek die gebruikt wordt voor toepassingen in real time objectdetectie. De bibliotheek bevat meer dan 2500 geoptimaliseerde functies die gebruikt worden bij *real time* objectdetectie of ML-algoritmes. Deze algoritmes kunnen gebruikt worden om gezichten, objecten of beweging te detecteren, herkende objecten kunnen ook gevolgd worden. In deze toepassing wordt OpenCV gebruikt voor de live objectdetectie van het poolspel [10].

1.10.6 YOLOv3

Voor het eerste deel van deze opdracht moesten de ballen herkend worden met behulp van *computer vision*. Als technologie hiervoor is gebruikgemaakt van YOLOv3. YOLOv3 is een *real-time object detection-system*. Er bestaan verschillende varianten van YOLOv3. De detectieresultaten van verschillende andere neurale netwerken zijn te vinden in Figuur 2.



Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Figuur 2: Testresultaten object detection systems [11]

Zoals op de bovenstaande afbeelding te zien is, zijn er verschillende varianten met elk hun eigen voor - en nadelen. Wat hieruit opvalt, is dat voor een hogere accuraatheid FPS ingeleverd moet worden. Bij deze stage lag de voorkeur bij YOLOv3-tiny, omdat deze variant een heel hoge FPS heeft en dus ook ballen in beweging beter kan herkennen, zodat de botsingen van ballen vastgelegd kunnen worden.

YOLOv3 werkt achterliggend met de *darknet neural network library*. De trainingsdata passeert door een aantal verschillende lagen. Zo kan het netwerk kenmerken bepalen en tijdens het gebruik zoeken in de inputdata.

Deze trainingsdata wordt verder beschreven in punt 3.2.1.

In neurale netwerken wordt met lagen gewerkt. De data die verwerkt wordt passeert door elke laag. Elke laag heeft een specifieke functie die de data bewerkt of analyseert:

- *Convolutional*: een *convolutional* laag passeert over de foto in een blok om de berekeningen te versimpelen, in plaats van de hele foto in een keer te berekenen kan er telkens een klein deel van de foto bekeken worden. De bedoeling van deze laag is om *high-level features* uit een foto te halen, bijvoorbeeld de randen van een object [12].
- *Maxpool:* een *maxpool*-laag heeft dezelfde bedoeling dan een *convolutional*-laag, de grootte van de data verkleinen. Er bestaan twee soorten *pooling*-lagen. Een *maxpool*-laag en een *avgpool*-laag. Hoe deze lagen werken, spreekt voor zichzelf. Max van maximum gaat naar de

hoogste waarde in bijvoorbeeld een drie op drie blok zoeken en deze opslaan. De *avg,* wat voor gemiddelde staat, gaat de gemiddelde waarde in een blok berekenen en opslaan [12].

- Yolo: de Yolo laag is de detectie laag. De Yolo laag heeft een anker attribuut. Dit attribuut zorgt voor het plaatsen van de herkenningsboxen. Het anker attribuut beschrijft negen ankers, maar enkel de ankers die geïndexeerd zijn door de *mask tag* worden gebruikt. De waarde van de *mask tag* is 0, 1, 2 wat betekent dat het eerste, het tweede en het derde anker gebruikt worden. Elke cel van de detectielaag voorspelt drie *boxes*. In totaal hebben we dectectielagen op drie schalen wat zorgt voor een totaal van negen ankers.
- *Route:* de route laag heeft een attribuut genaamd *layers.* Dit attribuut kan een of twee parameters bevatten.
    - Bij één parameter gaat de *feature map* van de aangegeven laag als output doorsturen. Als er bijvoorbeeld -3 staat gaat dit de output zijn van de derde laag voor de *route* laag zijn.
    - Bij twee parameters gaat de *feature map* van de twee desbetreffende lagen gelinkt worden en vervolgens doorgestuurd. Als er bijvoorbeeld "-2, 63" staat zal de tweede laag voor de *route* laag gelinkt worden met de 63^{e} laag. Dit wordt dan doorgestuurd als output [13].

De standaard lagen van YOLOv3-tiny zijn:

1) [convolutional]
2) [maxpool]
3) [convolutional]
4) [maxpool]
5) [convolutional]
6) [maxpool]
7) [convolutional]
8) [maxpool]
9) [convolutional]
10) [maxpool]
11) [convolutional]
12) [maxpool]
13) [convolutional]

Na deze lagen gebeurt de voorbereiding voor de *yolo*-laag.

14) [convolutional]
15) [convolutional]
16) [convolutional]
17) [yolo]
18) [route]
19) [convolutional]
20) [upsample]
21) [route]
22) [convolutional]
23) [convolutional]
24) [yolo]

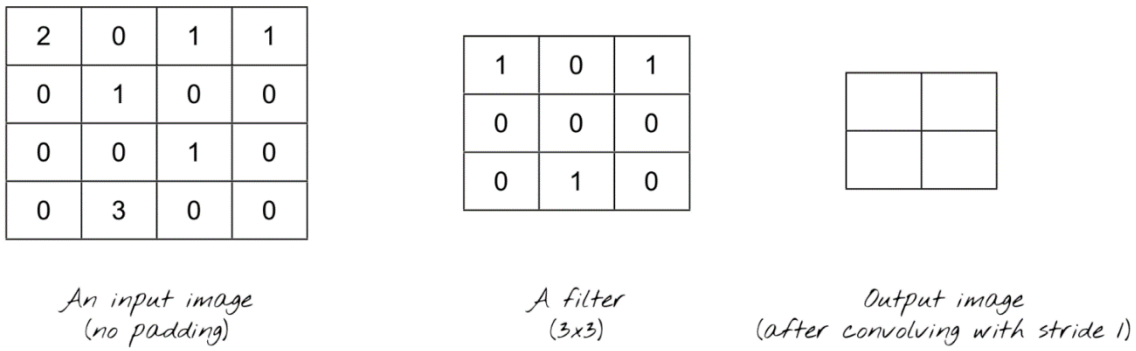
Voor de pooltafel stage was er een model nodig dat de ballen kon herkennen. Voor het herkennen van bewegende poolballen is een hoge FPS waarde nodig. De enige variant van YOLO die een hoge FPS waarde ondersteunt is de tiny-versie. Hierdoor is de beslissing genomen om deze variant te gebruiken.

Zoals hierboven te zien is, zijn er eerst een aantal *convolutional* lagen afwisselend met *maxpool*-lagen. Bij elke *convolutional* laag wordt het aantal filters in de laag telkens verdubbeld tot 1024 filters. Een groter aantal filters betekent een dieper zicht op kenmerken van de objecten.

Gevolgd door drie *convolutional* lagen met 256, 512 en 255 filters gevolgd door een *yolo*-laag die in dit voorbeeld tachtig klassen bevat. De laatste laag voor de *yolo*-laag moet telkens het aantal klassen plus vijf maal drie bevatten, wat in dit geval resulteert in (80+5) * 3. De klassen zijn de objecten die herkend moeten worden. Voor te pooltafel zijn dit 22 klassen namelijk de zestien ballen + zes gaten. Hierna komen er verschillende lagen die de data weer voorbereidt op de laatste *yolo*-laag [14].

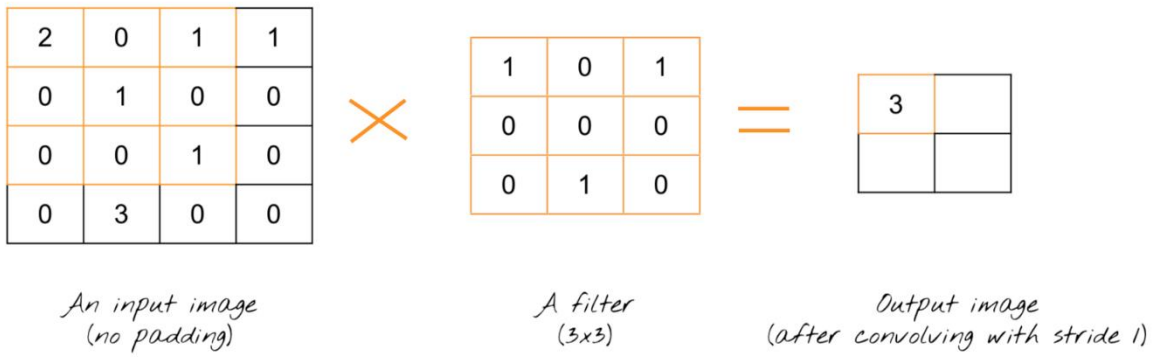
Elke *convolutional* laag gebruikt een aantal filters om de foto te analyseren.

De grootte van de filter bepaald hoe er naar de foto gekeken wordt. Er worden meerdere filters genomen om over de foto te navigeren van links naar rechts en boven naar onder. Neem bijvoorbeeld een vier maal vier foto en een drie maal drie filter. De visuele representatie wordt afgebeeld in Figuur 3.



Figuur 3: Een input image met de bijhorende filter en een lege output image [15]

De eerste stap is te zien in Figuur 4 en betreft het gele deel van de *input* vermenigvuldigen met de filter. Elk element wordt vermenigvuldigd met het overeenkomende element in de filter. Van deze vermenigvuldiging neemt de filter de som. Deze som is het resultaat dat in de *output* te staan komt. In Figuur 4 wordt dit afgebeeld.



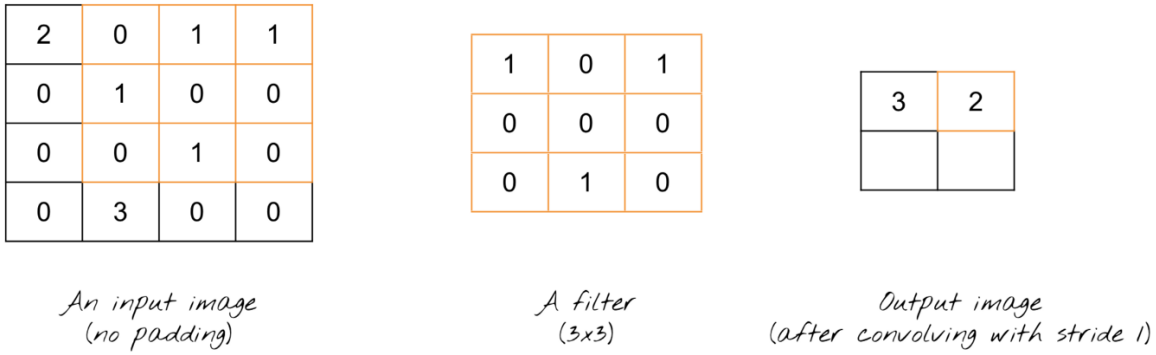
Figuur 4: De eerste positie en berekening van de filter [15]

De wiskundige formule staat afgebeeld in Vergelijking 1.

$$(2 * 1) + (0 * 0) + (1 * 1) + (0 * 0) + (1 * 0) + (0 * 0) + (0 * 0) + (0 * 1) + (1 * 0) = 3$$

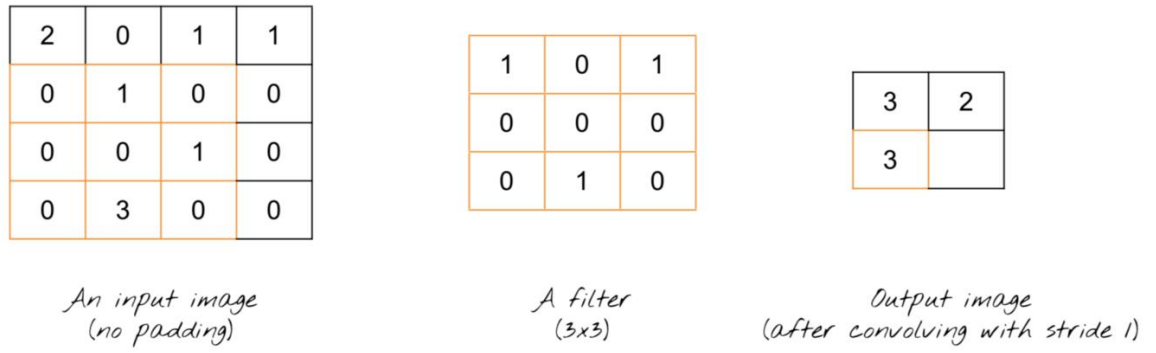
Vergelijking 1: De berekening van een filter [15]

Vervolgens wordt dezelfde stap herhaalt door de filter naar rechts te verplaatsen en de vermenigvuldiging te doen. Zoals te zien is in Figuur 5 berekent de filter nu de tweede output waarde.



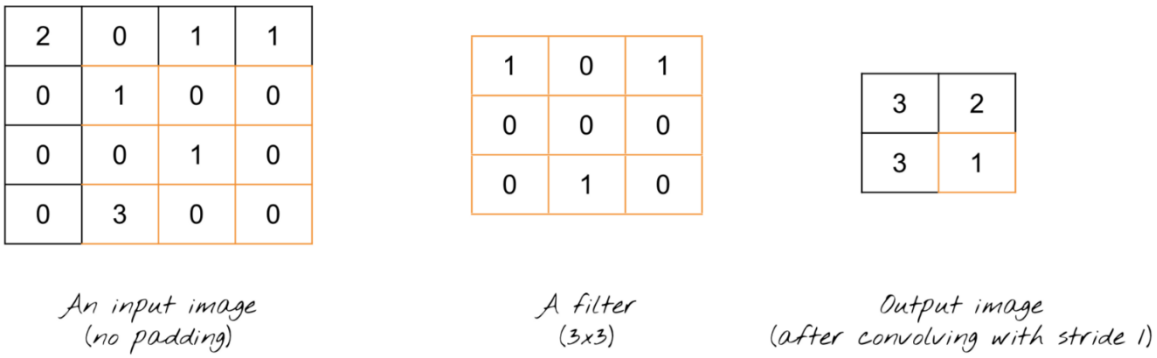
Figuur 5: De filter op de tweede positie [15]

Wat opvalt is dat de filter maar met een kolom verschoven is. De verschuifwaarde wordt de *stride* genoemd. Hier is “stride=1” gebruikt aangezien de filter met één plaats verschoven is. Hetzelfde wordt nog eens herhaald voor de derde output te krijgen. Dit wordt afgebeeld in Figuur 6.



Figuur 6: De filter op de derde positie [15]

Na de laatste keer verschuiven wordt de volledige output bekomen zoals te zien is in Figuur 7.



Figuur 7: De filter op de laatste positie en de volledige output [15]

Hier is te zien dat de output kleiner is dan de input, dit is in de meeste gevallen zo [15].

Voor deze stage zijn er aanpassingen gemaakt in de bestanden van darknet zodat elke honderd iteraties de gewichten opgeslagen worden in plaats van elke tienduizend iteraties. Dit is veranderd zodat er meer kans is dat de beste gewichten nog beschikbaar zijn omdat de duizendtallen nog steeds opgeslagen worden maar alle honderdtallen ertussen ook. Dit resulteert in 10 mogelijke gewichten die gebruikt kunnen worden in plaats van 1. Ook is ervoor gezorgd dat de volledige output die in de console geprint wordt, opgeslagen wordt zodat de variatie in het gemiddelde van de training gezien kan worden.

1.10.7 VoTT

VoTT oftewel *Visual Object Tagging Tool* is een programma om foto's of video's te annoteren. Dit is een programma dat gemaakt wordt door Microsoft. Het handige aan dit programma is dat een hele video kan ingeladen worden en er gekozen kan worden hoeveel *frames* per seconde geannoteerd worden. Nadat de video is ingeladen, kunnen *tags* geplaatst worden en met behulp van de e-toets naar het volgende frame gegaan worden en de vorige tags overnemen kan met de d-toets. Vervolgens kunnen deze tags verslepen worden.

1.10.8 Python 3

Python is gebruikt om fase 2 te realiseren. Deze keuze was het meest logisch, aangezien Linux gebruikt wordt als OS en Python heel goed samenwerkt met Linux. De focus van dit project ligt meer op de achterliggende technologie en niet op het frontend deel.

1.11 Hardware

1.11.1 GoPro HERO5

Om de data te verzamelen is gekozen voor een GoPro. Deze keuze is gemaakt met de prijs/video-kwaliteit in gedachten en het lage gewicht van een GoPro. Er wordt gefilmd op de hoogste kwaliteit die in *linear mode* kan filmen, wat in het geval van de GoPro 2.7k is. De *linear mode* werkt het *fisheye*-effect van de lens weg en zorgt dat de pooltafel rechthoekig is op de beelden. Op Figuur 8 is een GoPro HERO 5 te zien.

Figuur 8: GoPro HERO 5 [16]

1.11.2 GoPro Handlebar mount

Voor de bevestiging aan het achtergrondsysteem is een *handlebar mount* de perfecte oplossing. Door de ronde klem kan de GoPro hard vastgeklemd worden zodat er geen beweging mogelijk is. Dit is belangrijk voor wanneer er per ongeluk iemand tegen het achtergrondsysteem loopt. Als de GoPro goed vasthangt, zal er geen beweging zijn bij trillingen. De *mount* is te zien op Figuur 9.

Figuur 9: GoPro Handlebar mount [17]

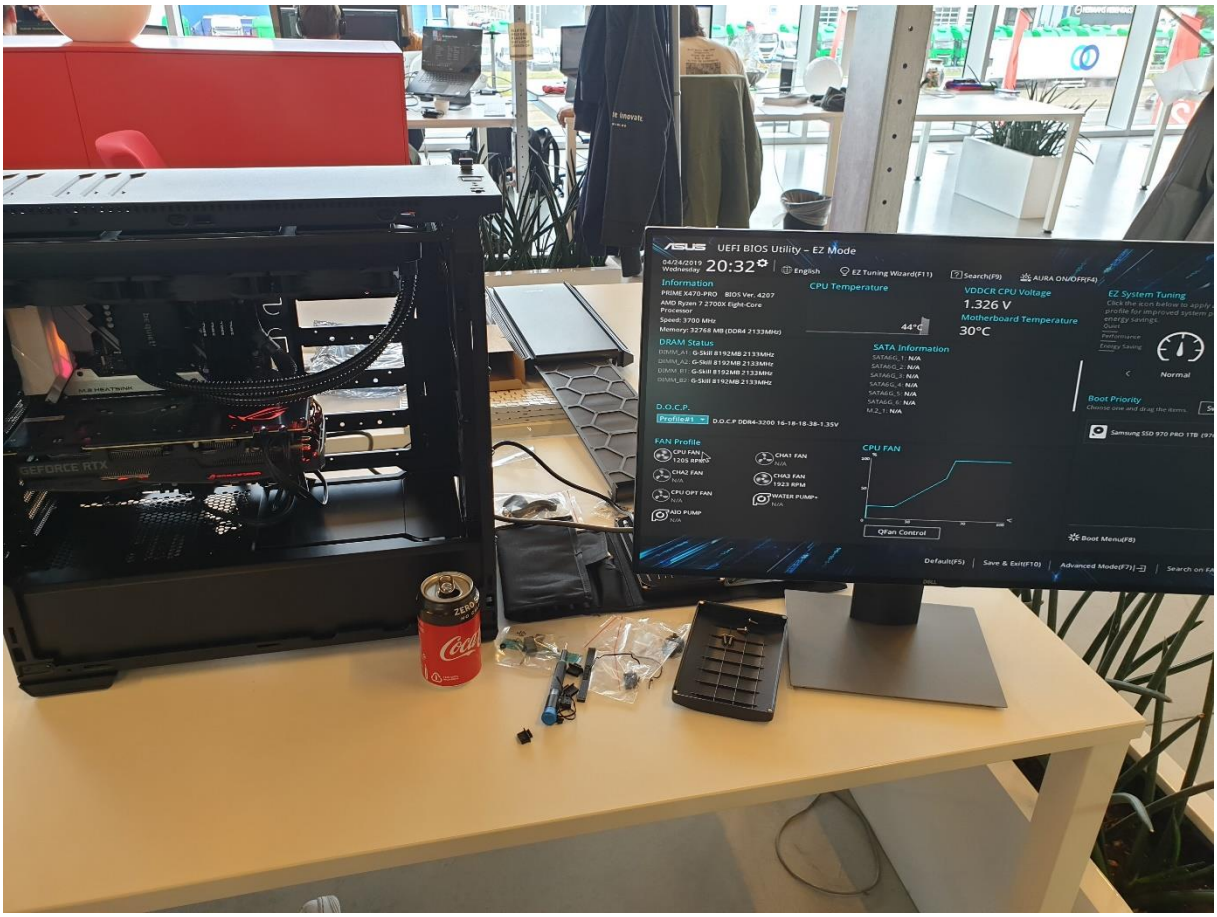
1.11.3 Achtergrondsysteem

Om de GoPro te bevestigen boven de tafel is gekozen voor een achtergrondsysteem dat in de fotografie gebruikt wordt om een rol papier aan te bevestigen om een gekleurde achtergrond te creëren op foto's. Dit leek de beste keuze aangezien het geheel verplaatsbaar en compact is wanneer het opgeplooid is. Het opgestelde systeem wordt afgebeeld op Figuur 10.

Figuur 10: Achtergrondsysteem

1.11.4 Workstation

Bij het begin van de stage was er een vaste computer voorzien door een medewerker van C4J. Al snel doken er enkele problemen op. Na overleg met de stage coördinator en enkele andere medewerkers van Bewire is er besloten dat er toch meer kracht nodig zou zijn. Er is een hardware lijst opgesteld waaruit de managing partners konden kiezen. De uiteindelijke beslissing was dat een volledige case gebouwd zou worden. De gekozen GPU is er echter niet gekomen door problemen bij de leverancier maar er is een andere, gelijkaardige GPU gekozen. Uiteindelijk is er voor een variant van de NVIDIA RTX2080Ti gekozen. Deze keuze is gemaakt door de 11gb virtuele RAM die deze GPU tot zijn beschikking heeft. Nog een belangrijk aspect van deze kaart is dat er veel *cuda cores* zijn. *Cuda cores* worden gebruikt voor de verwerking van data dus meer cores, geeft een snellere kaart [18]. Het workstation is te zien op Figuur 11.



Figuur 11: Eerste start workstation

Uitwerking stageopdracht

YOLO is een onderdeel van darknet. Darknet is een neuraal netwerk geschreven in C en CUDA [19]. Om darknet te installeren moet een “make” uitgevoerd worden in de darknet-map. In het begin van de stage zijn hier echter problemen mee geweest en zijn er files handmatig aangepast. Bij de nieuwe pc verliep alles wel vlot en zijn er geen problemen geweest tijdens de installatie. Alle bronnen zijn gemaakt voor Ubuntu 18.04, maar werkten ook op de 19.04-versie die op het workstation geïnstalleerd is.

1.12 Implementatie

Voordat er getraind kan worden, moeten alle benodigdheden geïnstalleerd worden. Allereerst moet een besturingssysteem geïnstalleerd worden. Dit doet men met behulp van een *bootable* usb. Dit is een usb met een versie van een besturingssysteem, dat door een programma zoals BalenaEtcher uitvoerbaar gemaakt werd.

Bij de eerste opstart van de computer of de *dual boot* moet via de BIOS ingesteld worden dat de start gebeurt vanaf een USB. De BIOS kan geopend worden door op F2 of DEL te drukken. Afhankelijk van de hardware kunnen de toetsen verschillen.

Na de installatie van het besturingssysteem kunnen alle nodige programma’s geïnstalleerd worden.

1.12.1 Installatie CUDA

Bij de installatie van CUDA op de computer die ter beschikking gesteld was in het begin van de stage is CUDA 9 geïnstalleerd met de bijhorende CuDNN versie. CUDA 9 installeren op Ubuntu 18.04 kan met behulp van de volgende commando’s [20].

```
1.  #!/bin/bash
2.
3.  ### eerst wordt er nagekeken of de GPU CUDA-compatibel is.
4.  lspci | grep -i nvidia
5.
6.  ### gcc compiler is nodig voor te ontwikkelen met de cuda toolkit. De installatie
    van gcc kan nagekeken worden met het volgend commando.
7.  gcc --version
8.
9.  # eerst moeten de PPA repository drivers binnen gehaald worden.
10. sudo add-apt-repository ppa:graphics-drivers/ppa
11.
12. # nvidia drivers installeren
13. sudo apt install nvidia-384 nvidia-384-dev
14.
15. # andere belangrijke packages installeren
16. sudo apt-get install g++ freeglut3-dev build-essential libx11-dev libxmu-dev libxi-
    dev libglu1-mesa libglu1-mesa-dev
17.
18. # CUDA 9 heeft gcc 6 nodig
19. sudo apt install gcc-6
20. sudo apt install g++-6
21.
22. # download een van de "runfile (local)" installatie paketten van het cuda toolkit
    archief
23. wget https://developer.nvidia.com/compute/cuda/9.0/Prod/local_installers/cuda_9.0.17
    6_384.81_linux-run
24.
25. # Maak de file uitvoerbaar
26. chmod +x cuda_9.0.176_384.81_linux.run
27. sudo ./cuda_9.0.176_384.81_linux.run --override
```

```
28.
29. # beantwoord de volgende vragen bij het begin van de installatie.
30. # You are attempting to install on an unsupported configuration. Do you wish to cont
    inue? y
31. # Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 384.81? n
32. # Install the CUDA 9.0 Toolkit? y
33.
34. # stel de symbolische links voor gcc/g++ in
35. sudo ln -s /usr/bin/gcc-6 /usr/local/cuda/bin/gcc
36. sudo ln -s /usr/bin/g++-6 /usr/local/cuda/bin/g++
37.
38. # stel de paden in
39. echo 'export PATH=/usr/local/cuda-9.0/bin:$PATH' >> ~/.bashrc
40. echo 'export LD_LIBRARY_PATH=/usr/local/cuda-
    9.0/lib64:$LD_LIBRARY_PATH' >> ~/.bashrc
41. source ~/.bashrc
42.
43. # installeer cuDNN v7.2
44. # enkel geregistreerde gebruikers kunnen cuDNN downloaden. De registratie kan met de
    volgende link. https://developer.nvidia.com/developer-program/signup
45. # hierna kan cuDNN gedownload worden met de volgende
    link. https://developer.nvidia.com/cudnn
46. CUDNN_TAR_FILE="cudnn-9.0-linux-x64-v7.2.1.38"
47. wget https://developer.nvidia.com/compute/machine-
    learning/cudnn/secure/v7.2.1/prod/9.0_20180806/${CUDNN_TAR_FILE}
48. tar -xzvf ${CUDNN_TAR_FILE}
49.
50. # kopieer de volgende bestanden naar de cuda toolkit folder
51. sudo cp -P cuda/include/cudnn.h /usr/local/cuda-9.0/include
52. sudo cp -P cuda/lib64/libcudnn* /usr/local/cuda-9.0/lib64/
53. sudo chmod a+r /usr/local/cuda-9.0/lib64/libcudnn*
54.
55. # ten slotte, om de installatie na te kijken kunnen de volgende commando's gebruikt
    worden.
56. nvidia-smi
57. nvcc -V
```

Bij de installatie op "Pop_OS!" konden er heel wat dingen fout gaan, aangezien de laatste nieuwe versie van CUDA gemaakt is voor een 18.10-versie en niet voor de laatste nieuwe Linux-versie. Bij de eerste poging is CUDA 9 geïnstalleerd met behulp van de bovenstaande commando's. Na een geslaagde installatie kon YOLOv3 opgestart worden. Bij de eerste start van YOLOv3 werd de GPU niet gebruikt en na wat research moest CUDA 10 geïnstalleerd worden omdat deze versie compatibel is met de GPU van de nieuwe computer en CUDA 9 niet. Als guide voor deze installatie worden de volgende commando's gebruikt [21].

Eerst worden de juiste *dependencies* geïnstalleerd.

```
1. sudo apt-get install build-essential dkms
2. sudo apt-get install freeglut3 freeglut3-dev libxi-dev libxmu-dev
```

Na de installatie van de *dependencies* kan op de officiële site van Nvidia de deb-versie gedownload worden. Hier staan de commando's voor de installatie van CUDA 10.

```
1. sudo dpkg -i cuda-repo-ubuntu1804_10.0.130-1_amd64.deb
2. sudo apt-key adv --fetch-
   keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/7fa2
   af80.pub
3. sudo apt-get update
4. sudo apt-get install cuda
```

Na de installatie van CUDA 10 moest er enkel nog een aanpassing gebeuren in de .bashrc file. Dit is een bestand waarin onder andere gespecifieerd wordt welke versie van CUDA gebruikt moet worden. Deze commando's moeten enkel gebruikt worden bij meerdere installaties van CUDA.

```
1. export PATH=$PATH:/usr/local/cuda-10.0/bin
2. export CUDADIR=/usr/local/cuda-10.0
3. export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-10.0/lib64
```

1.12.2 Installatie cuDNN

Voor elke CUDA-versie is er een bijhorende cuDNN-versie. Deze kan gevonden worden op de site van NVIDIA. De installatie van cuDNN kan gebeuren met de volgende commando's [22]. De URL en de CUDNN variabele zal aangepast worden naargelang de versie van CUDA en de laatste nieuwe CUDNN versie voor de desbetreffende CUDA versie.

Eerst wordt de CUDA *Toolkit* geïnstalleerd.

```
1. wget https://developer.nvidia.com/compute/cuda/9.0/Prod/local_installers/cuda_9.0.17
   6_384.81_linux-run
2. chmod +x cuda_9.0.176_384.81_linux.run
3. sudo ./cuda_9.0.176_384.81_linux.run --override
```

Tijdens de installatie zullen de volgende vragen beantwoord moeten worden.

- *You are attempting to install on an unsupported configuration. Do you wish to continue? => yes*
- *Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 384.81? => no*
- *Install the CUDA 9.0 Toolkit? => yes*
- *Create symbolic links? => yes*

Na de installatie van de *Toolkit* kan cuDDN geïnstalleerd worden. In plaats van de eerste twee lijnen kan de laatste versie van de site gedownload worden en uitgepakt met het derde commando.

```
1. CUDNN_FILE="cudnn-9.0-linux-x64-v7.2.1.38"
2. wget https://developer.nvidia.com/compute/machine-
   learning/cudnn/secure/v7.2.1/prod/9.0_20180806/${CUDNN_FILE}
3. tar -xzvf ${CUDNN_FILE}
```

Vervolgens worden enkele files naar de CUDA *toolkit* locatie gekopieerd.

```
1. sudo cp -P cuda/include/cudnn.h /usr/local/cuda-9.0/include
2. sudo cp -P cuda/lib64/libcudnn* /usr/local/cuda-9.0/lib64/
3. sudo chmod a+r /usr/local/cuda-9.0/lib64/libcudnn*
```

Om de installatie na te kijken kunnen de volgende commando's ingegeven worden. Het is mogelijk dat er een nieuwe terminal gestart moet worden of dat de computer opnieuw opgestart moet worden.

```
1. nvidia-smi
2. nvcc -V
```

1.12.3 Installatie OpenCV

Hier gaat OpenCV geïnstalleerd worden met de volgende commando's [23].

1.12.3.1 Stap 1 *Dependencies* installeren

Voor de installatie van OpenCV gestart wordt moet de *package manager* nog geüpdatet worden.

```
1. sudo apt-get update
2. sudo apt-get upgrade
```

Vervolgens moeten de *developer tools* geïnstalleerd worden:

```
1. sudo apt-get install build-essential cmake unzip pkg-config
```

Om foto's te kunnen verwerken met OpenCV moeten de volgende pakketten geïnstalleerd worden:

```
1. sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
```

Om video's of livestreams te kunnen verwerken zijn nog andere pakketten nodig:

```
1. sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
2. sudo apt-get install libxvidcore-dev libx264-dev
```

OpenCV heeft de GTK of Gimp *Toolkit* bibliotheek nodig voor GUI of voluit *Graphical user interface* operaties.

```
1. sudo apt-get install libgtk-3-dev
```

Er worden nog twee bibliotheken geïnstalleerd om een paar functies van OpenCV te optimaliseren.

```
1. sudo apt-get install libatlas-base-dev gfortran
```

Ten slotte worden de Python3 *headers* en *libraries* nog geïnstalleerd.

```
1. sudo apt-get install python3-dev
```

1.12.3.2 Stap 2 OpenCV Downloaden en bouwen.

Na al deze commando's kan OpenCV eindelijk geïnstalleerd worden.

```
1. cd ~
2. $ wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.4.zip
```

Gevolgd door de opencv_contrib module.

```
1. wget -
   O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/3.4.4.zip
```

Vervolgens worden de twee bestanden uitgepakt.

```
1. unzip opencv.zip
2. unzip opencv_contrib.zip
```

De namen van de uitgepakte folders worden veranderd met de volgende commando's.

```
1. mv opencv-3.4.4 opencv
2. mv opencv_contrib-3.4.4 opencv_contrib
```

OpenCV moet nog gebouwd worden.

```
cd ~/opencv
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D INSTALL_C_EXAMPLES=OFF \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D PYTHON_EXECUTABLE=~/.virtualenvs/cv/bin/python \
    -D BUILD_EXAMPLES=ON ..
```

Naargelang het aantal *cores* dat de CPU heeft kan deze *flag* aangepast worden. De workstation heeft er zestien dus zou de vier aangepast moeten worden naar zestien.

```
make -j4
```

1.12.3.3 Stap 3 OpenCV installeren

Wanneer het laatste commando succesvol verlopen is kan OpenCV geïnstalleerd worden.

```
sudo make install
sudo ldconfig
```

De installatie kan nagekeken worden met het volgend commando.

```
pkg-config --modversion opencv
```

1.12.4 Installatie YOLOv3

De installatie van darknet is gemakkelijk uit te voeren met de volgende commando's [11].

```
git clone https://github.com/pjreddie/darknet
cd darknet
make
```

Na darknet te installeren zijn er nog gewichten nodig voor de uitvoering van de detectie. Darknet voorziet een vooraf getrainde set van gewichten die gedownload wordt met het volgende commando.

```
wget https://pjreddie.com/media/files/yolov3.weights
```

Nadat de gewichten gedownload zijn kunnen deze gebruikt worden voor de herkenning van verschillende objecten in een foto. De uitvoering van de detectie gebeurt als volgt.

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

Voor de toepassing in het geval van de poolballen detecteren en volgen is er echter gebruik gemaakt van de Tiny-versie die minder lagen bevat en dus veel sneller werkt maar accuraatheid opoffert. Om deze versie te gebruiken moeten er andere gewichten gedownload worden.

```
wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

Er zijn ook nog aanpassingen gemaakt aan verschillende files om de output van een video op te slaan. De gewichten worden elke 100 iteraties opgeslagen in plaats van elke 1000. Er is ook voor gezorgd dat de output van de training opgeslagen wordt in een txt-bestand zodat dit gefilterd kon worden op beste gewichten met behulp van een python script dat terug te vinden is in bijlage C. Verder is “image_opencv.cpp“ aangepast om de output van een (live) video op te slaan. De aanpassing betreft de show_image_cv methode zijn terug te vinden op Figuur 12.

```
int show_image_cv(image im, const char* name, int ms)
{
   Mat m = image_to_mat(im);
   imshow(name, m);
   int c = waitKey(ms);
   if (c != -1) c = c%256;
   return c;
}
```

```
int show_image_cv(image im, const char* name, int ms)
{
   Mat m = image_to_mat(im);
   imshow(name, m);
   {

       if(video == NULL){
           const char* output_name = "predictions.avi";
           //output_video = cvCreateVideoWriter(output_name, CV_FOURCC('H', '2', '6', '4'), 25, size,
 1);
           video = new VideoWriter(output_name, VideoWriter::fourcc('M','J','P','G'),30, Size(im.w,im.
h));
           //output_video = cvCreateVideoWriter(output_name, CV_FOURCC('M', 'J', 'P', 'G'), 25, size,
 1);
           printf("\n DST output_video = %s  \n", output_name);
       }

       video->write(m);
        printf("\n cvWriteFrame \n");
   }
   int c = waitKey(ms);
   if (c != -1) c = c%256;
   return c;
}
```

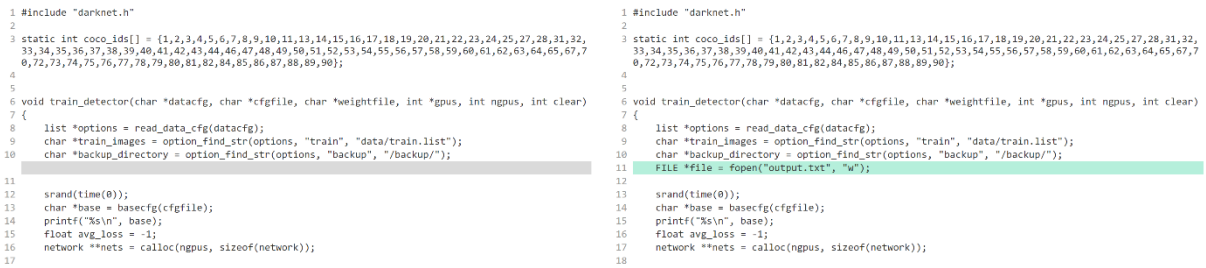
Figuur 12: Aangepaste image_opencv.cpp

De volgende code kan geplakt worden in plaats van de volledige methode.

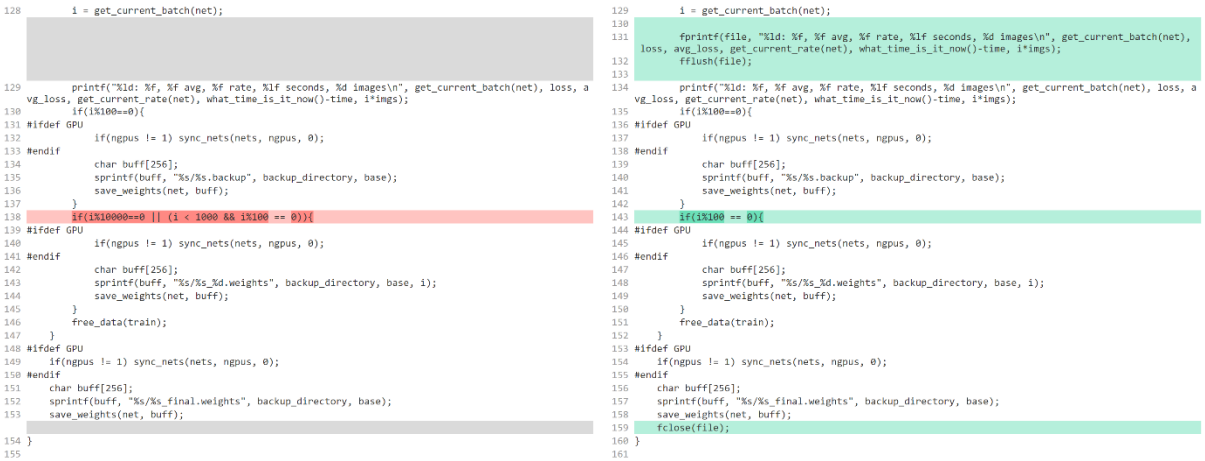
```
VideoWriter* video;

int show_image_cv(image im, const char* name, int ms)
{
   Mat m = image_to_mat(im);
   imshow(name, m);
   {

       if(video == NULL){
           const char* output_name = "predictions.avi";
```

```
            //output_video = cvCreateVideoWriter(output_name, CV_FOURCC('H', '2', '6', '4'), 25, size, 1);
            video = new VideoWriter(output_name, VideoWriter::fourcc('M','J','P','G'),30, Size(im.w,im.h));
            //output_video = cvCreateVideoWriter(output_name, CV_FOURCC('M', 'J', 'P', 'G'), 25, size, 1);
            printf("\n DST output_video = %s  \n", output_name);
        }

        video->write(m);
         printf("\n cvWriteFrame \n");
    }
    int c = waitKey(ms);
    if (c != -1) c = c%256;
    return c;
}
```

En ten slotte de aanpassingen voor het opslaan van de output zijn ook te vinden in het detector.c bestand. Voor deze aanpassingen is geen opzoekwerk gebeurd en was de voorgaande kennis voldoende om de nodige aanpassingen te maken. De aanpassingen zijn te zien in Figuur 13 en Figuur 14.



Figuur 13: Aanpassingen detector.c



Figuur 14: Aanpassingen detector.c

1.12.5 Installatie VoTT

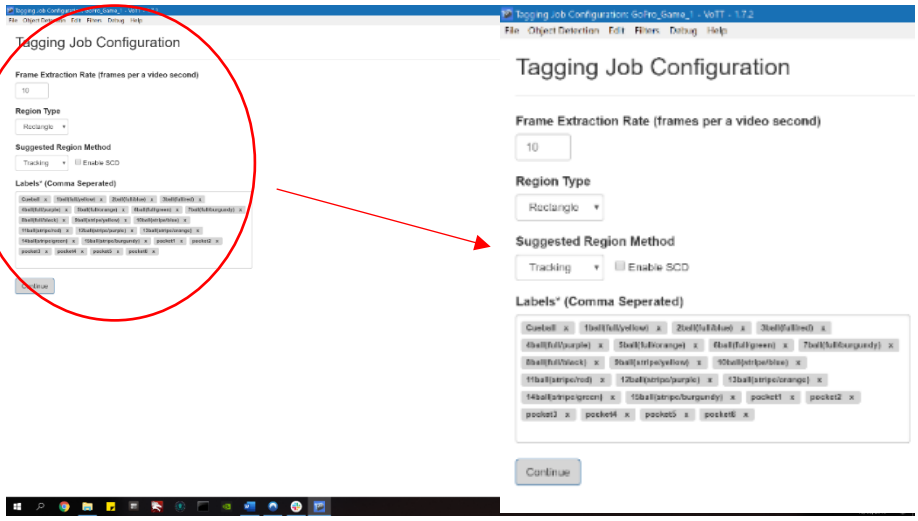
VoTT installeren is ook heel gemakkelijk. De git kan gekloond worden. De nodige commando's voor de installatie zijn hieronder te vinden. Verdere uitleg kan gevonden worden op de github pagina van VoTT [24].

```
wget https://github.com/Microsoft/VoTT/releases/download/v1.7.1/vott-linux.snap
snap install –dangerous vott-linux.snap
```

1.13 Gebruik

1.13.1 VoTT

Voor een model getraind kan worden, moet er geannoteerde data zijn. Deze data wordt binnen dit project gemaakt met VoTT. Een reeks foto's of een video wordt ingeladen en de nodige labels worden toegevoegd. Wanneer een video ingeladen wordt, kan aangegeven worden hoeveel frames per seconde geannoteerd worden. Dit is te zien op Figuur 15.

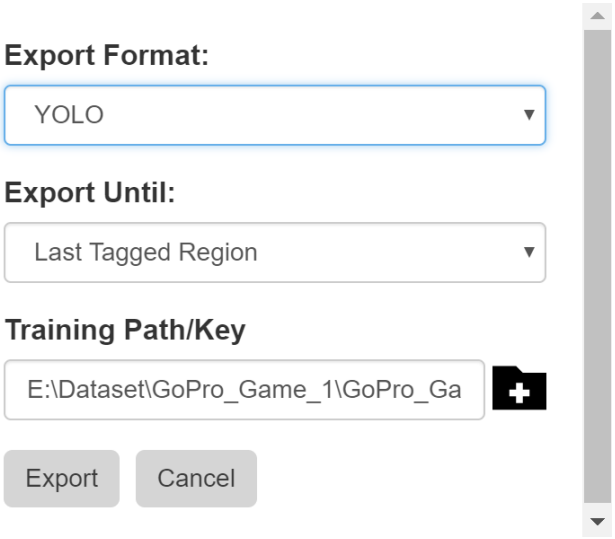


Figuur 15: VoTT configureren

Vervolgens worden alle frames geannoteerd zoals te zien is op Figuur 16.

Figuur 16: Gebruik VoTT

Als alle data geannoteerd is, exporteert men de foto's of frames naar een output folder, waar bij elke foto een tekstbestand staat met alle *tags* die op de bijhorende foto aanwezig zijn. Als er geëxporteerd wordt moet YOLO aangeduid worden zoals te zien is op Figuur 17.



Figuur 17: Het exporteren van de data

Niet enkel de frames met bijhorende ".txt" file worden geëxporteerd maar ook een ".cfg" die niet bruikbaar is aangezien deze niet voor YOLOv3 gemaakt is, een .data bestand dat te vinden is in het volgend punt, een ".names" bestand waarin alle namen van de *tags* vermeld staan en ten slotte een train- en een testbestand met de namen van de bestanden die gebruikt gaan worden om te testen en om te trainen. Dit bestand is ook niet gebruikt geweest aangezien er een andere verhouding van traindata/testdata aangenomen is [24].

1.13.2 YOLOv3

De geannoteerde data kan YOLO nu gebruiken om een model te trainen. Eerst moeten er twee bestanden voorzien zijn. Een bestand met trainingsfoto's in en een bestand met testfoto's in. Voor deze bestanden te maken is er een script geschreven dat de data opsplitst. Dit script is te vinden als bijlage A. Na de opsplitsing van de data kan de training beginnen. Eerst zijn de vooraf getrainde gewichten nodig. Deze kunnen op de volgende manier gemaakt worden [25].

```
1. wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

- Verkrijg de vooraf getrainde gewichten yolov3-tiny.conv.15 met het commando:

```
./darknet partial cfg/yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15
```

- Maak een eigen model yolov3-tiny-obj.cfg gebaseerd op cfg/yolov3-tiny_obj.cfg in plaats van yolov3.cfg

Nu wordt de training gestart met het volgende commando:

```
1. ./darknet detector train data/obj.data yolov3-tiny-obj.cfg yolov3-tiny.conv.15
```

Het is aanbevolen om minstens een aantal iteraties van de klassen maal tweeduizend te trainen. Gedurende het project is er gemerkt dat dit echter te weinig is en wordt het model nog beter naargelang er langer getraind wordt.

De Tiny-versie uitvoeren kan met hetzelfde commando als de gewone versie maar in plaats van de "yolov3.weights" zal er "yolov3-tiny.weights" gebruikt worden.

```
./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg
```

Al de voorgaande detectie werd uitgevoerd op een foto. Er kan ook detectie gedaan worden op een (live) video. Er kan een vooraf opgenomen video geanalyseerd worden met behulp van het volgende commando. In plaats van de *detect* in het vorige commando wordt hier *detector demo* gebruikt gevolgd door het .data bestand waarin het aantal klassen, de naam van de klassen, de testdata locatie en de traindata locatie staat. Hierna zien we het ".cfg" bestand en de gewichten.

```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights
```

Het '.data' bestand ziet eruit als volgt.

```
classes = 22
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/
```

De traindata en de testdata zijn 2 '.txt' bestanden met een relatief pad naar de foto's.

```
data/obj/GOPR1031_frame_2.jpg
data/obj/GOPR1031_frame_5.jpg
data/obj/GOPR1031_frame_13.jpg
data/obj/GOPR1031_frame_14.jpg
data/obj/GOPR1031_frame_15.jpg
```

Het cfg-bestand kan teruggevonden worden in bijlage B. Het bestand met de gewichten is niet leesbaar dus zal dit ook niet toegevoegd worden aan de bijlagen.

Dit commando kan ook toegepast worden op een videobestand. Na het vorige commando typt men de locatie van het videobestand [11].

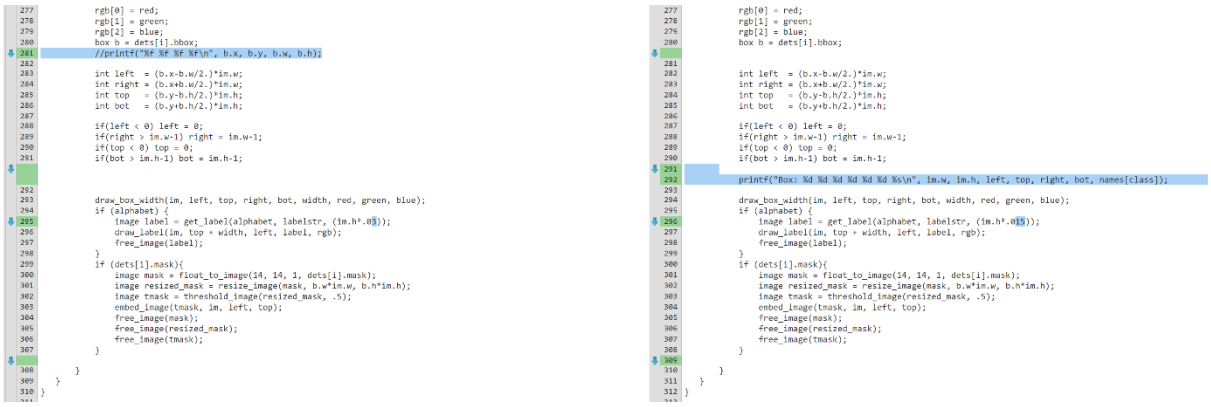
```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights <video file>
```

1.14 Uitwerking

1.14.1 Fase 2

Voor fase 2 moest er gebruik gemaakt worden van de objectherkenning uit de eerste fase om regels op toe te passen en de score kunnen bij te houden. Hier is gebruik gemaakt van een applicatie geschreven in Python. Jammer genoeg kunnen niet alle regels geïmplementeerd worden door de complexiteit van het implementeren en de begrenzingen van de balherkenning uit fase 1. De algemene en meest belangrijke regels zoals wie welke soort ballen heeft, welke ballen binnengespeeld zijn en waar de ballen binnen gespeeld zijn zitten er in.

Er zijn aanpassingen gemaakt aan bestanden binnen darknet zodat de output van de ballen ook doorgestuurd werd en zo verwerkt kon worden in de applicatie. Nog een kleine aanpassing zorgt ervoor dat de grootte van de herkenning aangepast is wat te zien is bij punt 3.4. Bij de laatste foto is de tekst kleiner en is het duidelijker om alles te zien. Deze aanpassingen zijn te zien op Figuur 18.



Figuur 18: Verschillen in het image.c bestand

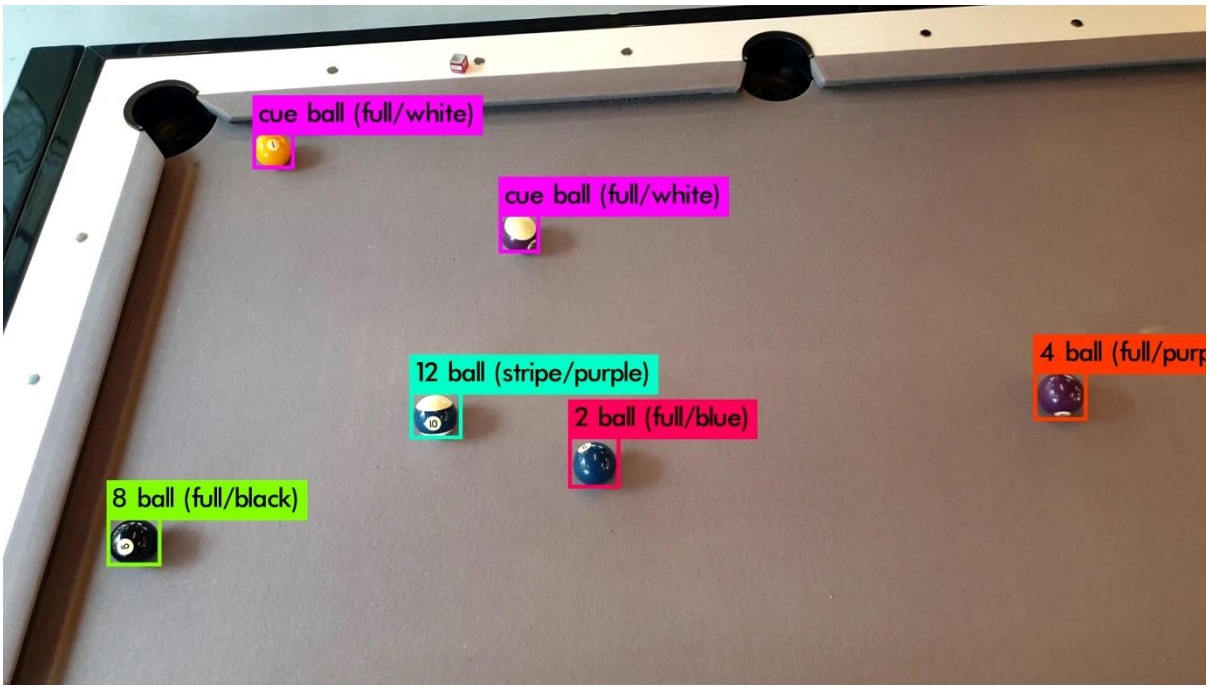
1.14.2 Fase 3

Fase 3 bestond uit de training van een AI die de beste zet kon bepalen uit de huidige opstelling van de ballen. Dit is geprobeerd met behulp van *Deep Reinforcement learning* maar de *agent* krijgt de waardes jammer genoeg niet geconvergeerd en uit fase 3 hebben we geen resultaat kunnen halen.

In het paper van Dary Schaeken [26] is een gedetailleerde uitleg te vinden over *Deep Reinforcement Learning.*

1.15 Resultaat

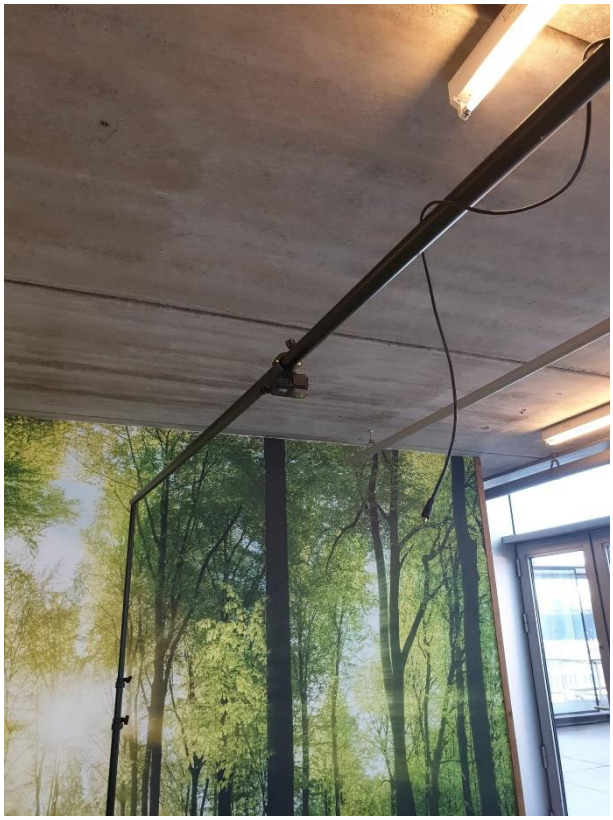
Hieronder volgen de resultaten die geboekt zijn doorheen de stage. Op de laatste foto is het uiteindelijke detectiemodel te zien. De progressie in de herkenning van objecten is duidelijk zichtbaar. Initieel kon het objectdetectie-systeem maar enkele ballen herkennen. Op Figuur 19 wordt de balherkenning getoond zonder de GoPro.



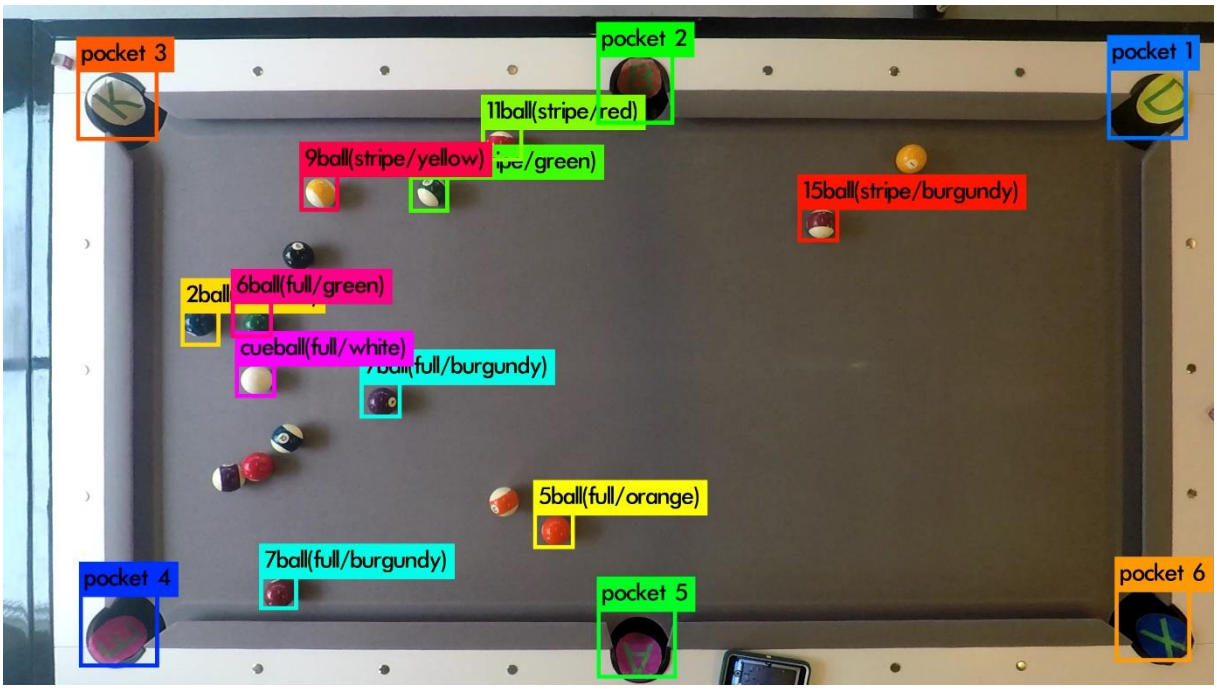
Figuur 19: Balherkenning zonder GoPro

Nadat de bovenstaande foto getoond werd is er beslist van in te zetten op beter materiaal en is er een GoPro bevestigd op een achtergrondsysteem. De opstelling is te zien op Figuur 20 en Figuur 21.

Figuur 20: Opstelling achtergrondsysteem



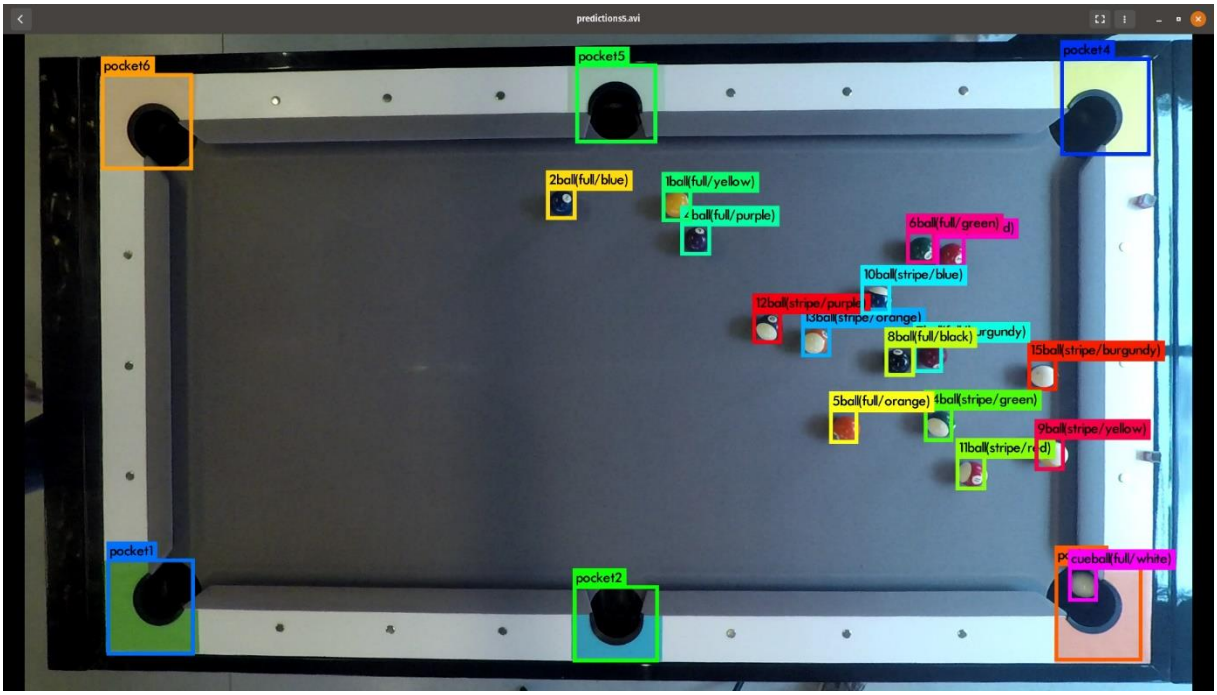
Figuur 21: Close-up bevestiging GoPro



Figuur 22: Balherkenning met GoPro

Zoals te zien is op Figuur 22 is de herkenning aanzienlijk verbeterd.

Toen deze verbetering er eindelijk kwam was de herkenning vrij goed zoals te zien is in Figuur 23. Na aanpassingen aan de herkenning van de gaten en een nieuwe dataset ziet de uiteindelijke herkenning er als volgt uit.



Figuur 23: Uiteindelijke balherkenning

Soms kan de herkenning nog wegvallen. Dit heeft vooral te maken met de gelimiteerde dataset en de verschillen in belichting die dagelijks voorkomen. Met meer tijd zouden de foute herkenningen en de wegvallende herkenning nog verminderd kunnen worden.

1.16 Reflectie

Ik heb voor deze stageopdracht gekozen omdat tijdens de opdracht gebruikt gemaakt wordt van verscheidene vernieuwende technologieën die tijdens de opleiding niet aan bod gekomen zijn. Natuurlijk was deze stage een grote uitdaging, hierdoor kreeg ik meer motivatie om toch een goed resultaat op te leveren.

In het begin van de stage heb ik me vooral gefocust op het maken van de balherkenning. Terwijl ik hier aan werkte heeft Dary onderzoek gedaan over *deep reinforcement learning*. later in de stage zijn deze rollen omgedraaid. Terwijl ik aan mijn onderzoek bezig was kon Dary werken aan het opstellen van de *reinforcement learning agent*.

Er zijn een aantal momenten geweest doorheen de stage waar ik niet meer wist wat ik eerst moest doen. Op deze momenten heb ik raad kunnen vragen aan Timo en Dary. Hierdoor wist ik weer beter wat ik moest doen.

Ik ben blij met het behaalde resultaat van fase 1 en 2 maar ik had gehoopt dat er ook nog resultaat zou komen uit fase 3. Ik vond het jammer dat het voorspellen van de stoten niet gelukt is maar dit is een limitatie van de technologie en/of de tijd die we hadden. Hier zou ik in mijn vrije tijd nog op willen terugkomen om toch een resultaat te behalen.

Ik heb geen aanbod gekregen bij het stagebedrijf doordat ik mijn enthousiasme niet toonde en niet zeker was welke richting ik uit wou gaan na de stage. Hier ga ik in de toekomst aan werken om een duidelijke keuze te hebben en mijn enthousiasme meer te uiten.

Conclusie

Computer vision is ongetwijfeld iets waar veel mogelijkheden in zijn. Ook is Computer vision vrij gemakkelijk toepasbaar en snel aangeleerd. Het annoteren van data neemt veel tijd in beslag maar in de eindresultaten is ook te zien waarom dit zo belangrijk is. Als de stage twee weken langer was kon er een betere dataset gemaakt worden waardoor de herkenning nog beter zou werken.

Deep reinforcement learning is niet toepasbaar omdat het aantal mogelijkheden waar ballen liggen veel te groot is om te kunnen berekenen voor de huidige technologie. Dit kan mede te maken hebben doordat de technologie die voor dit soort oplossingen gebruikt wordt nog maar een paar maanden oud is. In de toekomst zal deze technologie zeker en vast verbeteren en zal er wel een gelijkaardige toepassing gemaakt kunnen worden hierbij.

In toekomstige stages kan mogelijk gekozen worden om met *semi reinforcement learning* te werken. Hierdoor wordt er minder tijd verloren bij het annoteren van de data en zou het model ook goed moeten presteren.

II. Onderzoekstopic

In deze paper zal er verder gebouwd worden op een paper gemaakt door Anissa Schirock. Anissa heeft vorig jaar stage gelopen bij C4J en is dieper ingegaan op alles betreffende *computer vision* en één neuraal netwerk. In deze paper is het breder bekeken, de voordelen en nadelen van de verschillende neurale netwerken werden onderzocht en er werd bekeken hoe *deep learning* verschilt van *reinforcement learning*. Hier volgt een opsomming van de bekende informatie. Voor elk van de technologieën is er natuurlijk wel een kennisbasis [27] [28]. Tot op heden zijn, zover geweten zijn, nog geen research papers gemaakt die deze technieken vergelijken met elkaar.

Introductie

DL is een ML-techniek die computers leert doen wat natuurlijk komt met de mens: leren via voorbeeld. Het is een manier van *supervised learning* dat wil zeggen dat er nood is aan menselijke interactie. DL wordt bijvoorbeeld gebruikt in zelfrijdende auto's zoals Tesla. DL gebruikt bestaande, en geannoteerde, data die door verschillende lagen gestuurd worden in een neuraal netwerk. Hierdoor kan AI een heel hoge accuraatheid behalen die soms zelfs de mens overstijgt [27]. Alle data moet handmatig geannoteerd worden. Hier kruipt natuurlijk heel veel tijd in wanneer er van miljoenen foto's gesproken wordt. Bij RL moet er een omgeving opgesteld worden. Binnen deze omgeving gaan beloningen of strafpunten toegekend worden. De AI wil een zo hoog mogelijke score halen. De punten worden bepaald door een fitness functie.

DL, ML en RL mogen dan wel onderling verbonden zijn met elkaar maar er is geen van de drie dat een andere gaat overnemen [28]. Elk heeft zijn eigen toepassingsgebied en zal in de toekomst massaal groeien. In 2018 was het marktaandeel van "AI" in Amerika ongeveer $7,35 miljard waard, dit zou jaarlijks tot $3,5- $5,8 triljoen kunnen opbrengen [28].

Het is dus belangrijk voor Bewire dat als er potentieel in zit ook te investeren in de mogelijkheden die er zijn, om een speler op de markt te worden. De werknemers van Bewire hebben interesse in dit gebied. Vandaar dat dit onderzoek opgezet is om het bedrijf en de werknemers te kunnen helpen in het leren van deze technologie.

Probleemstelling of vraagstelling

Wat is de beste manier om een model te trainen voor de *Smart Pooltable*?

Met behulp van dit onderzoek zal er een antwoord gezocht worden op de volgende vragen:

- Wat zijn de verschillen tussen neurale netwerken?
- Wat zijn de toepassingsverschillen?
- Zijn er alternatieven?

Elke stap zal bijgehouden en uitgebreid uitgelegd worden zodat de werknemers of toekomstige stagiairs dit kunnen gebruiken om zelf een model te trainen.

Methode van onderzoek

Tijdens de hoofdopdracht van de stage wordt er gebruik gemaakt van DL. De kennis van DL zal vooral vanuit de hoofdopdracht voortkomen maar er zal ook extra research nodig zijn om de nodige vergelijkingen te kunnen maken. RL daartegen zal geleerd moeten worden door het gebruik van video's, papers en websites. Deze kennis is nodig om dieper in te gaan op beide technieken en niet enkel alles oppervlakkig te bekijken.

Hierna gebeurt er een op research gebaseerde afweging van de beste RL en DL samenstelling. Tenslotte volgt de vergelijking van de verschillende achterliggende netwerken. Door de voorgaande onderzoeken en vergelijkingen kan de beste opstelling van beide technieken samengesteld worden.

Nadat voldoende kennis opgedaan is zullen er verschillende papers over dit topic vergeleken worden en de resultaten van andere researchers afgewogen tegenover elkaar.

De voordelen en nadelen zullen nauwkeurig bijgehouden worden zodat voor ieder project de meest gepaste techniek gebruikt wordt. Er gaat een vergelijking gebeuren van de resultaten uit dit paper en de conclusie van de literatuurstudie. Nadat de vergelijking gebeurd is gaan de resultaten en problemen grondig beschreven worden.

Onderzoek

1.17 Algemeen

1.17.1 Supervised learning

SL maakt gebruik van een vooraf gemaakte dataset waarin alles geannoteerd wordt als een bepaald object. Dit heeft hetzelfde effect dan bijvoorbeeld gecorrigeerd worden door een lector. Zo is snel duidelijk wat juist is en wat niet. Er kruipt niet enkel veel tijd in een geschikte dataset maken maar ook geld en een menselijke annoteur. Er kunnen honderdduizenden tot miljoenen foto's in deze datasets staan [29].

1.17.2 Unsupervised learning

Om bijvoorbeeld met video's te kunnen leren zou er veel meer data nodig zijn dan bij foto's. Deze data verkrijgen zou veel tijd en geld kosten. Daarom wordt UL gebruikt zodat de data door een model kan passeren en het model vervolgens zelf gelijkenissen en patronen gaat moeten zoeken in deze data.

1.17.3 Semi-Supervised learning

Gelabelde data is enorm moeilijk verkrijgbaar. Bijvoorbeeld om een *speech-analysis*-model te trainen is voor elk uur spraak 400 uur annoteren nodig. Het doel van SSL is zowel gelabelde als niet-gelabelde data gebruiken. Het netwerk haalt voordeel uit het kleine deel gelabelde data tegenover een hele dataset van niet-gelabelde data. Een populair neuraal netwerk voor SSL is een *general adversarial network, of GAN.* Bij een GAN zijn er twee netwerken die tegen elkaar op trainen, Een van de twee probeert een foto na te maken. De andere probeert te achterhalen of deze foto deel is van de trainingsdata of een gegenereerde foto. Hoe beter de tweede wordt in het vinden van gegenereerde foto's, hoe beter de eerste wordt in de generatie van foto's [29].

1.18 Neurale netwerken

Hier zijn de belangrijkste neurale netwerken van vandaag de dag kort uitgelegd. Een neuraal netwerk is geïnspireerd door een biologisch zenuwstelsel. Het netwerk is een verzameling van neuronen die onderling verbonden zijn. De neuronen of cellen zullen vooraf beschreven worden.

1.18.1 Cellen

Een netwerk kan verschillende cellen bevatten, elke cel verwerkt de data anders. Hier volgt een opsomming van de cellen in de belangrijkste netwerken die in het volgende punt aangehaald worden:

- Input cel: Deze cel zorgt voor een punt waar data ingevoerd kan worden aan het netwerk. Vaak wordt hier ook een activatie functie toegepast [30].
- Output cel: Zorgt voor de output van het netwerk [30].
- Hidden cel: Een cel die de gewichten toepast op de output van de vorige laag waar een activatiefunctie op uitgevoerd is [30].
- Recurrent cel: Wordt gebruikt in *Recurrent Neural Networks* als een methode voor de toepassing van historische signaalstatus om convergentie te verbeteren waarbij het tijdsdomein wordt aangeduid als een belangrijke dimensie van het leerproces. Deze cellen hebben ook een connectie naar hunzelf wat bij de gewone hidden cel niet het geval is [30].
- Kernel cel: Dit zijn cellen waarin een filter toegepast wordt. [31]
- Convolution cel: Geeft het resultaat door van een *convolution kernel* vermenigvuldigd met een relatief bereik van de cellen van de onmiddellijk vorige laag naar de volgende laag [30].
- Pool cel: Voegt de resultaten van de cellen van een convolutielaag samen [30].
- Backfed input cel: Zorgt voor een punt waar bewust feedback gegeven wordt [30].
- Match input- output cel: Wordt gebruikt in auto-encoders om te convergeren op de identiteitsfunctie van netwerkinvoer naar netwerkuitvoer, meestal via een versmald gegevens pad waardoor functies worden geëxtraheerd na convergentie [30].

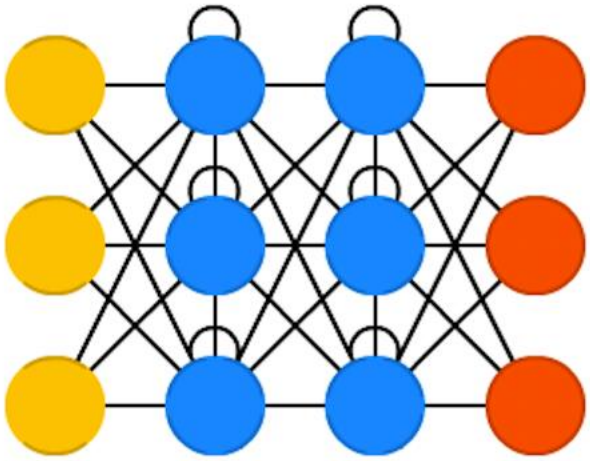
1.18.2 Feedforward neural network of perceptrons

Figuur 24: Feedforward neural network en Perceptron [32]

Een FFNN zoals afgebeeld is op Figuur 24 is het meest eenvoudige netwerk. De input gaat doorheen het model zonder de output van het netwerk terug te gebruiken in volgende iteraties. Een FFNN heeft één of meerdere lagen. Elke laag herkent een ander deel van de input. Bijvoorbeeld bij de herkenning van een gezicht kijkt de eerste laag naar de randen van het gezicht. Bij de tweede laag naar de contouren en hoeken en bij de derde laag naar delen van de foto bijvoorbeeld oren, ogen, neus, … om zo tot de outputlaag te komen waarin de meest waarschijnlijke uitkomst bepaald wordt [33].

Perceptrons hebben meerdere inputs waaruit een output resulteert. Elke input heeft ook een gewicht. Neem als voorbeeld dat een man naar een festival wil, maar zijn vrouw niet. Maar de man wil enkel gaan als het niet regent. De input “regent het?” zal zwaarder doorwegen. Er kan een grens ingesteld worden voor de weergave van een resultaat. Als de grens bijvoorbeeld op vijf gezet wordt en het gewicht van zes op het weer en op zowel “wil de man?” als “wil de vrouw?” een waarde van twee gaan ze enkel als het droog weer is. Als de grens op drie gezet wordt gaan ze ook naar het festival ookal is het slecht weer. Door het veranderen van de gewichten en de grens kunnen er verschillende outputs zijn. [34]

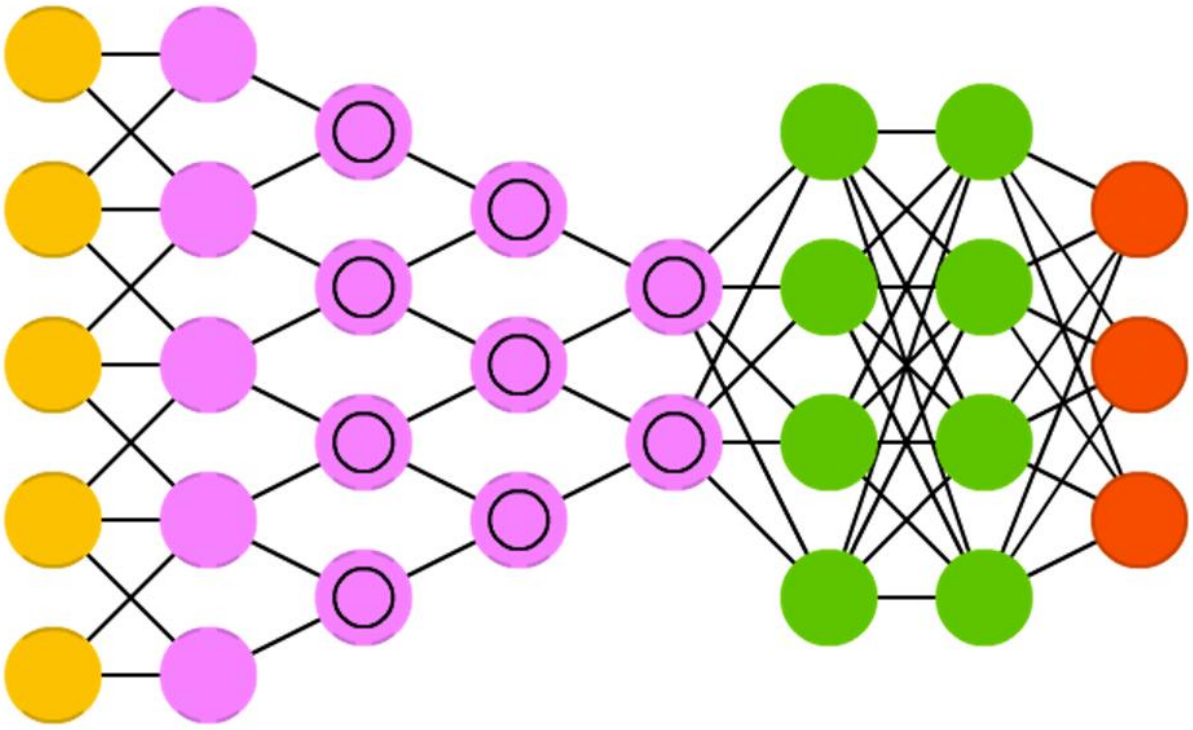
1.18.3 Recurrent neural network



Figuur 25: Recurrent neural network [32]

Een RNN is niet staatloos zoals andere neurale netwerken. Op Figuur 25 wordt een RNN afgebeeld. Het heeft connecties tussen iteraties en tijd. De binnenkomende informatie is niet enkel van de vorige neuronen, maar ook terugkomende informatie van zichzelf uit vorige iteraties. Bij dit soort netwerk is de volgorde van de input belangrijk. Als de volgorde veranderd wordt, is er een grote kans dat de output ook verschilt. RNNs kan enkel gebruikt worden waar er geen tijdlijn bij te pas komt bijvoorbeeld een video of geluidsfragment bevat een tijdslijn en kan dus niet gebruikt worden. De aparte frames van een video kunnen wel gebruikt worden deze worden dan opgeslagen als foto’s. Voor foto’s of tekst wordt pixel per pixel of letter per letter door het netwerk gestuurd. In het algemeen zijn RNNs het beste om als *auto complete* netwerk te gebruiken [32].

1.18.4 Convolutional neural network



Figuur 26: Convolutional neural network [32]

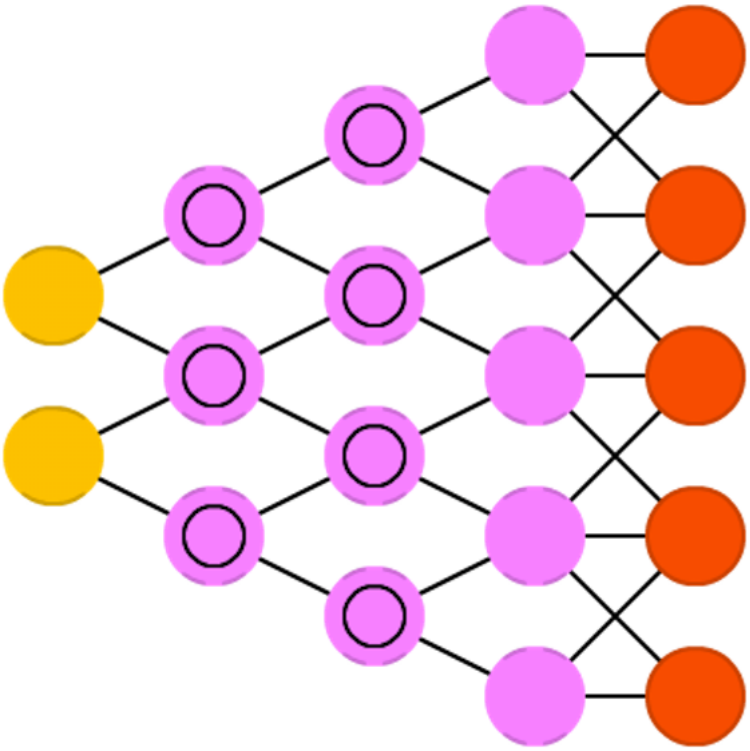
Een CNN wordt vooral gebruikt om afbeeldingen te verwerken. Een CNN is een soort van FFNN. Het netwerk krijgt bijvoorbeeld een foto van een *cue bal* en het model retourneert “cue bal”. De ingevoerde afbeelding is bijvoorbeeld tweehonderd maal tweehonderd pixels. Een laag van vierhonderdduizend nodes is te groot om in een keer te verwerken dus gaat de filter over de afbeelding in delen van twintig maal twintig pixels.

De filter begint meestal in de linkerbovenhoek en stuurt deze twintig maal twintig pixels door het netwerk. Bij gebruik van meerdere filters is dit de output van alle filters die in één keer door het netwerk gestuurd wordt. De filter schuift één pixel op naar rechts en herhaalt het proces tot het einde van de foto bereikt is. De invoer wordt verwerkt door *convolutional layers* in plaats van de normale lagen. Het verschil hier is dat de nodes zich enkel bekommeren om de nabijgelegen cellen in plaats van alle cellen. Deze lagen worden normaal gezien ook kleiner naarmate het netwerk dieper wordt. De grootte van deze lagen is meestal een macht van twee, omdat deze gemakkelijk deelbaar zijn.

Naast deze lagen zijn er ook nog *pooling layers.* Deze lagen worden gebruikt om details uit de foto te halen. Veel voorkomende waardes zijn bijvoorbeeld max pooling: twee maal twee. Uit deze twee maal twee wordt de meest voorkomende kleur genomen en deze wordt gebruikt om de rest te kleuren [32].

Meer informatie over lagen en filter kan gevonden worden in de omgeving en technologieën bij de uitleg van YOLOv3 oftewel in punt 1.10.6. Een visuele representatie van een CNN is te zien op Figuur 26.

1.18.5 Deconvolutional neural network



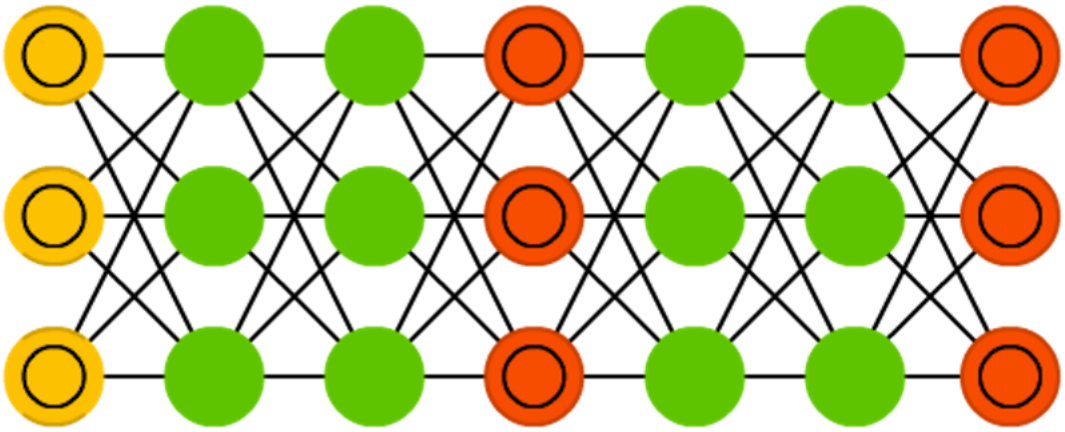
Figuur 27: Deconvolutional neural network [32]

Een DNN is een achterwaartse versie van een CNN. Bijvoorbeeld het netwerk wordt getraind op het woord “kat”. Het netwerk gaat een foto genereren van hoe het denkt dat een kat eruitziet en deze dan vergelijken met echte foto’s van katten om te leren wat er fout was en beter kan [32]. Een DNN is te zien op Figuur 27

Dit netwerk kan ook foto’s vergroten in resolutie en verscherpen. Zoals te zien is op Figuur 28 zijn deze netwerken hier heel goed in. De eerste foto is de originele, de tweede foto is verkleind en wazig gemaakt en de derde foto is de tweede die door een DNN verwerkt is [35].

Figuur 28: Een voorbeeld van DNN [35]

1.18.6 General adversarial neural network



Figuur 29: General adverserial network [32]

Een GAN is in feite een tweeling van twee netwerken die tegen elkaar werken waar er een maker van (neppe) data is en een beoordelaar. Dit type netwerk is te zien op Figuur 29. De maker gaat kiezen of er neppe data gemaakt en doorgestuurd wordt of dat er echte data doorgestuurd wordt naar de beoordelaar. De beoordelaar moet dan beslissen welke data het is. Zo leert de maker manieren om de beoordelaar slimmer af te zijn en de beoordelaar wordt beter in gegenereerde data herkennen [36]. Zo leren ze van elkaar en kunnen ze bijzonder slim worden.

De bovenstaande netwerken zijn vandaag de dag de meest gebruikte netwerken en kunnen zowel in RL als in DL gebruikt worden om een model te trainen. Welk netwerk je gebruikt is naargelang de toepassing waar het model getraind voor wordt.

1.19 Deep learning

1.19.1 Algemeen

DL is een toepassing van *machine learning*. DL maakt gebruik van neurale netwerken die het menselijk brein willen nabootsen. In een menselijk brein kunnen alle neuronen met elkaar verbinden. Een neuraal netwerk kan dit niet en heeft voorgedefinieerde lagen, connecties en data-propagatie.

Om deep learning te begrijpen kan volgend voorbeeld gebruikt worden. Een kleuter leert zijn eerst woord, 'hond'. De kleuter leert wat een hond is (en niet is) door naar objecten te wijzen en het woord hond te zeggen. De ouders gaan dan reageren met "Ja dat is een hond" of "nee dat is geen hond". Hoe meer dingen de kleuter aanwijst hoe meer hij zich bewust wordt van verschillende kenmerken die alle honden bezitten. Wat de kleuter eigenlijk aan het doen is zonder het te weten Is, zonder het te weten, is een complexe abstractie maken door een hiërarchie op te bouwen. Elk niveau van abstractie wordt gemaakt door de verkregen kennis van de voorgaande laag in de hiërarchie te verwerken.

Voor computerprogramma's werkt dit vrijwel op dezelfde manier. Elk algoritme in de hiërarchie past een non lineaire transformatie toe op zijn input en gebruikt wat hij leert om een statistisch model te creëren. Het aantal iteraties blijft oplopen tot het model een aanvaardbaar procent accuraatheid behaald heeft. Het aantal lagen waar de data door moet heeft voor de naam "*deep" learning* gezorgd [37].

Er kan bijvoorbeeld een foto verdeeld worden en deel voor deel door de neuronen gestuurd worden. Elke neuron stuurt de data door naar de volgende tot de laatste laag. Deze laag is de output van het neuraal netwerk. In elke laag gaat iets anders geleerd worden. In de ene laag kunnen dit vormen zijn van een object, in de andere laag kleuren, in de volgende laag de details enzovoort.

Elke neuron kent een gewicht toe aan de input. De output is een optelling van alle gewichten in het netwerk. Als output wordt er een *probability factor* gegenereerd die een gok naar het object weergeeft.

Om een juiste gok te verkrijgen uit een neuraal netwerk moet het netwerk getraind worden. Dit wordt gedaan met honderdduizenden of zelfs miljoenen foto's [38].

1.19.2 Toepassingen

1.19.2.1 Zelfrijdende auto's

Merken zoals Tesla zijn al enige tijd bezig met zelfrijdende auto's. Hier moet een computer leren hoe het bepaalde (of alle) delen van het rijden overneemt. Hier is natuurlijk enorm veel data voor nodig. De meeste bouwers gebruiken de data van rijdende auto's dus ook voor het trainen van hun zelfrijdende auto's [39].

1.19.2.2 Spraakherkenning

Assistenten zoals Siri, Bixby, Google Now, enzovoort zijn allemaal getraind met een diep neuraal netwerk. Standaard wordt hier ook alle data verstuurd naar de makers om verdere training bij te faciliteren [40].

1.19.2.3 Objectherkenning

Misschien wel de belangrijkste toepassing, gezien het project gelinkt aan deze paper, is objectherkenning. Dit is een veel gebruikte toepassing. Het wordt ook toegepast bij zelfrijdende auto's om mensen en dieren te herkennen. In beveiligingssystemen geldt hetzelfde, hier herkent het model

ook de mensen en dieren die voor een camera passeren. In dit project wordt er gebruik gemaakt van DL om poolballen te herkennen [41].

1.19.2.4 Kankerherkenning

Een revolutionaire toepassing is de herkenning van kankercellen op een foto. Hier wordt een combinatie gemaakt van het model en de arts. Uit metingen is gebleken dat een arts accurater was maar minder snel tegenover het model. Bij het model werden er *false positives* herkent. Dit wil zeggen dat er meer cellen aangeduid werden ook de gezonde cellen. De *output* van dit model werd nagekeken door de dokter en hier konden dan de juiste cellen uit aangeduid worden [42].

1.19.3 Nadelen

Een van de nadelen van DL is dat het model bijleert door te observeren. Het probleem ligt hier in de data waar het model op traint. Als de data van een enkele bron komt er niet genoeg variatie is in de data om een goed model te kunnen trainen. Dit geldt ook voor te weinig data of verkeerd gelabelde data.

Nog een voorkomend probleem is vooroordelen in de dataset. Als er bijvoorbeeld een voetbal en een basketbal herkend moeten worden en er zijn 10.000 foto's van de voetbal en 100 van de basketbal zou er een mogelijkheid zijn dat het model de basketbal ook als voetbal herkent. De dingen waar een model op let worden vaak niet duidelijk gemaakt aan de programmeur. Dit wil zeggen dat bijvoorbeeld een gezichtsherkenning model beslissingen gaat maken over mensen hun karakter op basis van hun huidskleur of geslacht zonder dat de programmeur het door heeft [37]. Het belangrijkste aspect is dat een AI geen gezond verstand heeft om na te denken. Er kunnen wel verbanden gelegd worden maar de AI weet niet wat deze verbanden betekenen. Deze nadelen zorgen voor onrust onder de "gewone" mens als het bijvoorbeeld over zelfrijdende auto's gaat. Mensen vertrouwen deze auto's niet. Ook al zijn de doden en gewonden bij ongevallen veel minder dan bij de mens zelf. Er is geen twijfel dat DL op veel manieren toepasbaar is maar het vertrouwen dat vandaag de dag gestoken wordt in DL en AI's over het algemeen is zorgwekkend. Hoeveel voordelen er ook mogen zijn moet er ook stilgestaan worden met mogelijke nadelen van AI [43].

1.20 Reinforcement learning

1.20.1 Algemeen

Bij RL wordt rekening gehouden met de score die verkregen is na het uitvoeren van een actie net zoals de mens zelf doet. Dit wil zeggen dat tijdens de training het model gaat achterhalen wat nodig is om een zo hoog mogelijke score te behalen in zo weinig mogelijk stappen [29].

RL bevat enkele begrippen zoals *agents, actions, environments, states* en *rewards*. Deze begrippen zullen hieronder uitgelegd worden [44].

- *Agent*: Een *agent* onderneemt acties. Bijvoorbeeld een spel uitspelen. In het echte leven ben jijzelf de *agent.*
- *Action(A)*: A is de set van alle acties die de *agent* kan uitvoeren. Een actie legt zichzelf uit, een *agent* kiest uit een lijst van deze acties. In een videogame kunnen dit bijvoorbeeld lopen naar rechts of links, hoog of laag springen, bukken enzovoort zijn. In toepassingen met aandelen kan dit kopen, verkopen of houden zijn.
- *Environment*: Dit is de wereld waar de *agent* zich in begeeft. Een *environment* neemt de huidige *state* en *action* van de *agent* en geeft de gepaste *reward* terug samen met de volgende *state*. Als een mens de *agent* is kunnen dit de wetten van de natuurkunde zijn en de regels van de samenleving die de *actions* verwerken en de gevolgen ervan berekenen.
- *State(S)*: Een *state* is de concrete en huidige situatie waarin een *agent* zich begeeft.
- *Reward(R)*: Een *reward* is de feedback waarbij we het succes of falen van een actie weergeven. Bijvoorbeeld als Mario een munt raakt krijgt hij een punt bij. Voor elke *state* stuurt de *agent* output als acties, het *environment* stuurt dan de nieuwe *state* van de *agent* en de *reward* als deze er is.

1.20.2 Toepassingen

1.20.2.1 Helikopter controle

Een toepassing waar een helikopter getraind werd om ondersteboven te balanceren [45].

1.20.2.2 Dialogue management

Nog een toepassing is het volgen van een dialoog om de persoon in kwestie te kunnen helpen met bijvoorbeeld een aankoop doen of zich ergens naartoe navigeren enzovoort [46].

1.20.2.3 Games

Er zijn verscheidene games waar RL op is toegepast [47].

1.20.3 Nadelen

1.20.3.1 Exploration vs exploitation

Een van de grootste problemen bij RL is *exploration vs exploitation*. Bij DL wordt aan generalisatie gedaan. Als de agent goed kan generaliseren in nieuwe situaties hoeft hij niet zoveel te exploreren. *Exploration vs exploitation* wil zeggen dat er ofwel vertrouwt wordt op de manieren die gekend zijn en beloningen opleveren of dat de kans genomen wordt om nieuwe manieren te zoeken die mogelijk nog een grotere beloning opleveren. Hier de juiste balans vinden is moeilijk omdat er geen juist antwoord is.

Zonder twijfel is de beste manier om te verkennen dan willekeurig. De agent weet niet wat een bepaalde actie kan opleveren zonder het te proberen. Zelfs na het krijgen van een beloning bij een bepaalde *state* wil dit niet zeggen dat deze beloning elke keer even groot zal zijn als de agent zich in die *state* bevindt. Het meest logische om te doen is niet te veel vertrouwen in een beloning maar om telkens de actie met kleine beetjes aanpassen om mogelijk een grotere beloning te krijgen.

Nog een groot probleem voor RL zijn *continuous action* en *state spaces.* Hoe kan de agent een oneindig aantal keer een oneindig aantal *states* bezoeken en een oneindig aantal acties uitvoeren vanuit elke state? Hierop kan een generalisatie toegepast worden. Dit kan gebeuren met behulp van *deep learning.*

Generalisatie in RL noemt ook wel *function approximation. Function approximation* gebruikt het idee dat een *state* en actie door een functie gevoerd kunnen worden en hieruit een waarde berekend wordt in plaats van elke waarde voor elke *state* en actie in een grote tabel op te slaan.

Om het nog eens kort samen te vatten is het grootste probleem van RL of eigenlijk AI: *exploration*.

Neem als voorbeeld Rainbow DQN. Rainbow is een soort van agent en DQN is de manier van leren uitgevonden door Deepmind. De meest ideale acties in de videogame leren duurt 83u. Omdat er vooraf niet geweten is wat een videogame is, dat vijanden kogels schieten, dat kogels slecht zijn, dat een aantal pixels die samen lijken te blijven een kogel zijn, dat kogels bestaan, dat objecten bestaan. Al deze dingen helpen de mens om het aantal opties drastisch te verlagen naar een kleine set van goede opties. AI moet deze opties leren door willekeurige exploratie [48].

1.20.3.2 Long term credit assignment

Sommige personen krassen hun loterij ticket met een geluksmunt omdat ze ooit eens gewonnen hebben met die munt. RL *agents* spelen eigenlijk de loterij met elke stap om te weten te komen wat ze ooit gedaan hebben om te winnen. Een serie van acties vinden voor de hoogst mogelijke winst te hebben is het probleem van *credit assignment.*

Een beloning geven moet gemakkelijk zijn. Men vertelt een robot wanneer iets juist uitgevoerd is en wanneer niet zodat over tijd geleerd kan worden hoe die actie betrouwbaar uitgevoerd kan worden.

Het probleem begint zich hier al voor te doen. De schaal waarin beloningen gegeven kunnen worden is groter dan wat de algoritmes vandaag de dag aan kunnen. Er is de mogelijkheid dat de tijdspanne tussen de actie en de beloning te groot is en dat de robot zo niet weet wat goed was en wat niet.

Hier zijn twee oplossingen voor. Een van de oplossingen is voor de schaal van beloningen te verkleinen. Met andere woorden meer opbouwende beloningen geven. Natuurlijk ligt hier de kans dat de robot dit gaat misbruiken om meer beloningen te krijgen als de beloningen niet goed gekozen zijn.

Een andere manier is hiërarchische RL om de te late rewards op te splitsen in *goals* en *subgoals.* Door het probleem op te splitsen kan de tijd tussen acties en beloningen verkleind worden [48].

1.21 Vergelijking Deep learning en Reinforcement learning

AI is een zeer breed veld, ML is een deel hiervan. DL en RL-algoritmes zijn beiden technieken binnen ML die gebruikt worden om data te verwerken. Het verschil zit in de techniek hoe het model leert. DL, zoals al gezegd is, heeft een dataset nodig met bestaande data die geannoteerd is, om wat hieruit geleerd wordt toe te passen op nieuwe data. RL daartegen leert dynamisch, door vallen en opstaan om informatieve keuzes te maken. Dit gebeurt door acties aan te passen naargelang de verkregen feedback van vorige acties [49].

Een groot verschil dat opviel tijdens de stage was dat de concepten van DL leren gemakkelijker en gebruiksvriendelijker is. RL is een “gevaarlijkere” piste om te belopen. De toekenning van juiste beloningen is zeer moeilijk en ervoor zorgen dat een agent niet te veel verkent is ook een moeilijke opdracht.

DL is ook combineerbaar met verschillende takken van ML. In tegenstelling tot RL waar de combinaties beperkt zijn.

1.22 Deep reinforcement learning

Deep Reinforcement Learning is een combinatie van Deep learning en Reinforcement learning. “Deep learning with no labels and Reinforcement learning with no tables” [50]. Dit onderzoeksveld heeft ervoor gezorgd dat complexe beslissingen die voorheen onmogelijk waren wel gemaakt kunnen worden. Hierdoor zijn er veel nieuwe mogelijkheden ontstaan in domeinen zoals in de gezondheidzorg, robots, financiën en veel meer.

DRL gebruikt een functie om een actie te bepalen in tegenstelling tot gewoon RL dat voor elke state de best mogelijke actie moet berekenen en opslaan in een tabel.

In DRL kan gebruik gemaakt worden van *Deep Q learning*. De Q in *(Deep) Q learning* staat voor de “*Quality”* van een *action* in een bepaalde *state*. Een actie is *high quality* als er een hoge *long term value* is. Neem als voorbeeld een kind die zijn mama een knuffel geeft net voor slaaptijd. Hij gaat waarschijnlijk niet direct een *reward* krijgen maar over een lange tijd gaat het hem veel liefde opleveren.

Q learning leert de *action-value* functie Q(s,a): hoe goed een actie is bij een bepaalde state. Bij *Q learning* wordt er een geheugentabel bijgehouden om de Q-waarde van alle mogelijke combinaties van s en a op te slaan. Dit is in feite een *cheat sheet* voor de beste actie te vinden.

In plaats van elke *state-action* combinatie te onthouden worden vaak *ConvNets* gebruikt voor de generalisatie van gelijkaardige *Q values. ConvNets* gebruiken *convolutional layers* om gelijkaardige dingen te vinden in de foto. Zo ja, waar?

ConvNets vertellen de videogame *agent* “deze positie is bijna hetzelfde als deze andere, doe dit”. Dit maakt de job van de *agent* gemakkelijker omdat er gekozen moet worden uit een paar miljoen opties in plaats van een paar miljard [51].

1.22.1 Policy gradient algorithms

Een *policy gradient algorithm* is een berekening om de snelst groeiende *reward* te bepalen. Er zijn vier *policy gradient algorithms* geselecteerd om te vergelijken: TRPO, PPO, DDPG en ACKTR.

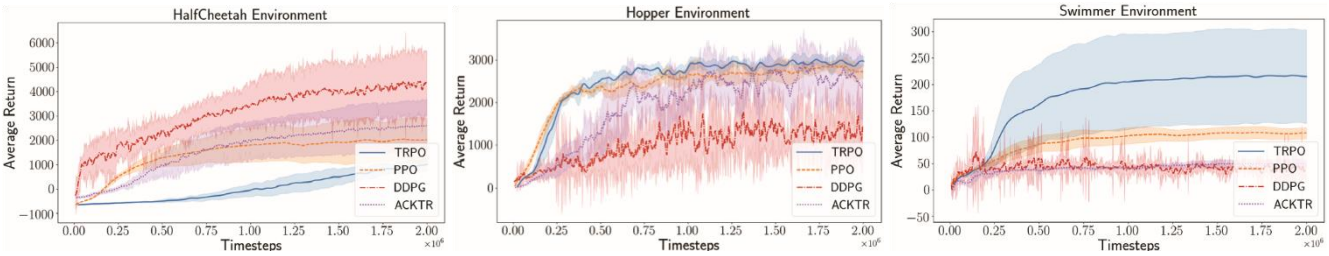
Trust region policy optimization: Zonder in de wiskunde te duiken is TRPO een *policy gradient* algoritme. Er wordt gebruik gemaakt van een *gradient ascent* om de snelst groeiende *reward* te bepalen. De *gradient ascent* berekent de snelst groeiende beloning om zo het beleid aan te passen in die richting [52].

Proximal policy optimization: PPO gebruikt een iets andere benadering. In plaats van een harde beperking op te leggen, formaliseert het de beperking als een sanctie in de doelfunctie. De doelfunctie zorgt ervoor dat de parameters van een model de hoogst of laagst mogelijke waarde bekomen. Door de beperking niet ten koste van alles te vermijden, kan een eerste-orde *optimizer* zoals de *Gradient Descent*-methode gebruikt worden om het doel te optimaliseren. De *Gradient-Descent*-methode zoekt naar de laagste error waarde om zo de hoogste accuraatheid te vinden [53]. Een *optimizer* heeft hetzelfde doel dan de doelfunctie, parameters aanpassen zodat de waarde van een functie zo hoog of laag mogelijk wordt [54]. Zelfs als de beperking eens in de zoveel tijd geschonden wordt, is de schade veel minder en is de berekening veel eenvoudiger [55].

Deep deterministic policy gradient: DDPG leert een deterministisch beleid dat de geschatte actiewaardefunctie maximaliseert. DDPG maakt gebruik van de actor-critic architectuur [56].

Actor Critic using Kronecker-Factored Trust Region: Dit is ook een *policy gradient* methode maar hier wordt een *actor-critic* architectuur gebruikt. De *Kronecker-factored approximation* wordt gebruikt om zowel da actor als de critic te optimaliseren door de complexiteit van berekeningen te verkleinen [57].

Uit eerdere studies kan afgeleid worden dat DDPG een van de betere algoritmes is dat in veel gevallen andere algoritmes overtreft behalve als er veel dynamiek aan te pas komt. Zoals te zien is in Figuur 30.



Figuur 30: Tests met verschillende algoritmes op verschillende environments [56]

Hier is te zien dat bij een dynamisch stabiel *environment* DDPG veel beter presteert dan de andere algoritmes. Maar bij minder stabiele omgevingen gaan de prestaties van DDPG drastisch verlagen en de beloningen verlagen natuurlijk naargelang de prestaties zoals te zien is in Tabel 1. [56]

Environment	DDPG	ACKTR	TRPO	PPO
HalfCheetah-v1	5037 (3664, 6574)	3888 (2288, 5131)	1254.5 (999, 1464)	3043 (1920, 4165)
Hopper-v1	1632 (607, 2370)	2546 (1875, 3217)	2965 (2854, 3076)	2715 (2589, 2847)
Walker2d-v1	1582 (901, 2174)	2285 (1246, 3235)	3072 (2957, 3183)	2926 (2514, 3361)
Swimmer-v1	31 (21, 46)	50 (42, 55)	214 (141, 287)	107 (101, 118)

Tabel 1: Vergelijking van belongingen tussen verschillende algoritmes met verschillende environments [56]

1.23 Reflectie

Bewire wil inzetten op vernieuwende technologieën. Het doel van deze stage was dus ook het achterhalen van hoe deze technologieën werken en hoe gemakkelijk ze aangeleerd kunnen worden. De toekomstige medewerkers kunnen nu een doordachte keuze maken wanneer ze een neuraal netwerk willen trainen.

Bij het begin van dit onderzoek wist ik niet goed wat ik moest behandelen en wat niet. Ik heb hulp gevraagd aan medewerkers van Bewire en mijn teamgenoot Dary Schaeken die mij op weg hielpen. Ik had ook veel moeite bij het combineren van de stageopdrachten waardoor ik vrij veel tijd heb moeten besteden aan mijn onderzoek.

Bij het begin van de stage heb ik veel tijd besteed aan de stageopdracht zelf. Naargelang de stage vorderde lag de focus meer en meer op het onderzoekstopic.

Ook had ik veel moeite met de juiste papers te vinden met de informatie in die ik wou verwerken in het onderzoek. Uiteindelijk heb ik een combinatie kunnen maken van papers en websites waaruit ik de nodige informatie heb kunnen halen.

Achteraf gezien had ik tijdens de paasvakantie mijn tijd beter gespendeerd aan het verder onderzoeken in plaats van data te annoteren. Maar aan het einde van het onderzoek ben ik toch tevreden over het resultaat.

Conclusie

AI speelt vandaag de dag al een grote rol in onze samenleving. Van mails filteren op spam tot beveiligingssystemen die personen kunnen herkennen, AI is overal. De toekomst van AI ziet er momenteel belovend uit maar er zou weer een AI-winter op komst kunnen zijn waarin alles AI gerelateerd opnieuw stilvalt. Een van de nadelen van projecten met betrekking tot neurale netwerken is de kracht die nodig is voor de training van deze netwerken.

Uit het onderzoek blijkt dat vooral de manier van leren en aanleren van *deep learning* en *reinforcement learning* sterk verschilt. DL is gemakkelijker toe te passen dan RL. DL maakt gebruik van een dataset die gelabeld of niet gelabeld kan zijn. Dit kan ook verwoord worden als *supervised* of *unsupervised learning*. RL daartegen leert compleet dynamisch. Het enigste dat RL nodig heeft is een *environment* om in te trainen.

Een beter alternatief wanneer een probleem niet opgelost kan worden met *reinforcement learning* is *deep reinforcement learning* hier probeert het *deep* deel gelijkaardige *states* samen te voegen zodat hiervoor dezelfde actie gebruikt kan worden en zo de mogelijke acties verkleind worden. Bij DRL is er wel een risico dat het model niet kan convergeren. Wanneer een probleem opgelost kan worden met RL geeft dit over het algemeen betere resultaten dan DRL.

Bibliografie

[1] Evance, „Evance | trigger senses digitally,” Evance, [Online]. Available: https://evance.be/. [Geopend 29 Mei 2019].

[2] C4J, „C4J -- Adaptive to your IT | Backend Development,” C4J, [Online]. Available: http://www.c4j.be/. [Geopend 29 Mei 2019].

[3] Dots&Arrows, „Dots&Arrows -- Integration Professionals,” Dots&Arrows, [Online]. Available: https://dotsandarrows.be/. [Geopend 29 Mei 2019].

[4] Trase, „Trase -- Blockchain en gedecentraliseerde technologie,” Trase, [Online]. Available: https://www.trase.be/. [Geopend 29 Mei 2019].

[5] Codrigo, „Codrigo Belgian Software Delivery Center,” Bewire, [Online]. Available: https://www.codrigo.be/. [Geopend 29 Mei 2019].

[6] Appmind, „Appmind | Shaping your mobile future together | We create mobile apps,” Appmind, [Online]. Available: http://www.appmind.be/. [Geopend 29 Mei 2019].

[7] „Pop!_OS by System76,” sytem76, [Online]. Available: https://system76.com/pop. [Geopend 21 Mei 2019].

[8] NVIDIA, „CUDA Toolkit | NVIDIA Developer,” NVIDIA, [Online]. Available: https://developer.nvidia.com/cuda-toolkit. [Geopend 29 Mei 2019].

[9] „NVIDIA CuDNN | NVIDIA Developer,” NVIDIA, [Online]. Available: https://developer.nvidia.com/cudnn. [Geopend 21 Mei 2019].

[10] OpenCV, OpenCV, [Online]. Available: https://opencv.org/about/. [Geopend 27 April 2019].

[11] pjreddie, „YOLO: Real-Time Object Detection,” [Online]. Available: https://pjreddie.com/darknet/yolo/. [Geopend 6 Mei 2019].

[12] S. Saha, Towards Data Science, 15 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Geopend 27 April 2019].

[13] K. Majek, „How to implement a YOLO (v3) object detector from scratch in PyTorch: Part 2,” PaperspageBlog, 16 April 2018. [Online]. Available: https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/. [Geopend 12 Mei 2019].

[14] Darknet, Darknet, 15 Augustus 2018. [Online]. Available: https://github.com/pjreddie/darknet/blob/master/cfg/yolov3-tiny.cfg. [Geopend 26 April 2019].

[15] S. Kim, „A Beginner's Guide to Convolutional neural networks (CNNs),” Towards Data Science, [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-convolutional-neural-networks-cnns-14649dbddce8. [Geopend 12 Mei 2019].

[16] dansdiveshop, [Online]. Available: https://dansdiveshop.ca/product/gopro-hero5-black/. [Geopend 23 Mei 2019].

[17] „GoPro Handlebar / Seatpost / Pole Mount,” GoPro, [Online]. Available: https://shop.gopro.com/EMEA/mounts/handlebar-seatpost-pole-mount/AGTSM-001.html. [Geopend 23 Mei 2019].

[18] C. McKee, „What are Nvidia cuda cores?,” lifewire, 3 November 2018. [Online]. Available: https://www.lifewire.com/what-is-nvidia-cuda-834095. [Geopend 27 April 2019].

[19] Pjreddie, „Darknet: Open Source Neural Networks in C,” pjreddie, [Online]. Available: https://pjreddie.com/darknet/. [Geopend 27 Mei 2019].

[20] Mahedi-61, „cuda 9.0 complete installation procedure for ubuntu 18.04 LTS - GitHub,” [Online]. Available: https://gist.github.com/Mahedi-61/2a2f1579d4271717d421065168ce6a73. [Geopend 6 Mei 2019].

[21] D. D. Kinghorn, „How to install CUDA 10 (together with 9.2) on Ubuntu 18.04 with support for NVIDIA 20XX Turing GPUs,” Puget systems, 27 September 2018. [Online]. Available: https://www.pugetsystems.com/labs/hpc/How-To-Install-CUDA-10-together-with-9-2-on-Ubuntu-18-04-with-support-for-NVIDIA-20XX-Turing-GPUs-1236/. [Geopend 6 Mei 2019].

[22] Unkown, „Ask a swiss: How to install cuda,” 7 Januari 2019. [Online]. Available: http://www.askaswiss.com/2019/01/how-to-install-cuda-9-cudnn-7-ubuntu-18-04.html. [Geopend 6 Mei 2019].

[23] A. Rosebrock, „Ubuntu 18.04: How to install OpenCV | PyImageSearch,” PyImageSearch, 28 Mei 2018. [Online]. Available: https://www.pyimagesearch.com/2018/05/28/ubuntu-18-04-how-to-install-opencv/. [Geopend 23 Mei 2019].

[24] wbreza, Microsoft, [Online]. Available: https://github.com/Microsoft/VoTT. [Geopend 6 Mei 2019].

[25] AlexeyAB, 23 April 2019. [Online]. Available: https://github.com/AlexeyAB/darknet#how-to-train-tiny-yolo-to-detect-your-custom-objects. [Geopend 7 Mei 2019].

[26] D. Schaeken, PXL. [Online]. [Geopend 2019 Mei 23].

[27] Mathworks, „What is deep learning?,” Mathworks, [Online]. Available: https://www.mathworks.com/discovery/deep-learning.html. [Geopend 8 Maart 2019].

[28] B. Osiński en K. Budek, „What is reinforcement learning?,” deepsense, 5 Juni 2018. [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/. [Geopend 2019 Maart 8].

[29] I. Salian, „Difference between supervised, unsupervised & reinforcement learning," NVIDIA, 2 Augustus 2018. [Online]. Available: https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/. [Geopend 3 April 2019].

[30] D. Daseeco, 19 September 2018. [Online]. Available: https://ai.stackexchange.com/questions/5898/neural-network-cell-node-types. [Geopend 13 Mei 2019].

[31] Prabhu, „understanding of convolutional neural networks (CNNS)," medium, 4 Maart 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Geopend 13 Mei 2019].

[32] F. V. Veen, The Asimov Institute, 31 Maart 2017. [Online]. Available: https://www.asimovinstitute.org/author/fjodorvanveen/. [Geopend 4 April 2019].

[33] T. Gupta, „Deep Learning: Feedforward Neural Network," Towards Data Science, 4 Januari 2018. [Online]. Available: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7. [Geopend 17 April 2019].

[34] M. A. Nielsen, „Neural Networks and Deep Learning," Determination Press 2015, October 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons. [Geopend 17 April 2018].

[35] A. V., 28 Juni 2017. [Online]. Available: https://software.intel.com/en-us/articles/an-example-of-a-convolutional-neural-network-for-image-super-resolution. [Geopend 7 Mei 2019].

[36] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville en Y. Bengio, *Generative Adversarial Nets,* Montréal: Universite de Montr ´ eal, 2014.

[37] M. Rouse, „What is deep learning (deep neural network)?," WhatIs, Januari 2018. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/deep-learning-deep-neural-network. [Geopend 7 Mei 2019].

[38] M. Copeland, „What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?," NVIDIA, 29 Juli 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/. [Geopend 3 april 2019].

[39] M. Pal, „Deep Learning for Self-Driving Cars," Towards data science, [Online]. Available: https://towardsdatascience.com/deep-learning-for-self-driving-cars-7f198ef4cfa2. [Geopend 9 Mei 2019].

[40] R. Sharma, „Speech to text app in your browser using deep learning," medium, 2 December 2019. [Online]. Available: https://medium.com/datadriveninvestor/speech-to-text-app-in-your-browser-using-deep-learning-35889fbd50ed. [Geopend 9 Mei 2019].

[41] J. Redmon, S. Divvala, R. Girshick en A. Farhadi, University of Washington, [Online]. Available: https://www.cv-

foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf. [Geopend 9 Mei 2019].

[42] A.-R. Ali, „Deep Learning in Oncology – Applications in Fighting Cancer | emerj," emerj, 19 Feruari 2019. [Online]. Available: https://emerj.com/ai-sector-overviews/deep-learning-in-oncology/. [Geopend 23 Mei 2019].

[43] R. Iriondo, towardsdatascience, 12 Februari 2019. [Online]. Available: https://towardsdatascience.com/limitations-of-deep-learning-in-ai-research-5eed166a4205. [Geopend 7 Mei 2019].

[44] Skymind, „A Beginner's Guide to Deep Reinforcement Learning," Skymind, z.d. [Online]. Available: https://skymind.ai/wiki/deep-reinforcement-learning. [Geopend 8 Mei 2019].

[45] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger en E. Liang, Stanford University, [Online]. Available: http://www.robotics.stanford.edu/~ang/papers/iser04-invertedflight.pdf. [Geopend 9 Mei 2019].

[46] S. sigh, D. Litman, M. Kearns en M. Walker, AT&T Labs - Research, [Online]. Available: http://web.eecs.umich.edu/~baveja/Papers/RLDSjair.pdf. [Geopend 9 Mei 2019].

[47] P. kordik, „Reinforcement Learning: Artificial Intelligence in Game Playing," medium, 17 November 2016. [Online]. Available: https://medium.com/@pavelkordik/reinforcement-learning-the-hardest-part-of-machine-learning-b667a22995ca. [Geopend 2019 Mei 9].

[48] H. Sahni, „Reinforcement Learning never worked, and 'deep' only helped a bit.," Himanshu Sahni's Blog, 23 Februari 2018. [Online]. Available: https://himanshusahni.github.io/2018/02/23/reinforcement-learning-never-worked.html. [Geopend 23 Mei 2019].

[49] L. M. Cappel, „What is the difference between Deep Learning and Reinforcement Learning?," bigdata-madesimpel, 1 Maart 2019. [Online]. Available: https://bigdata-madesimple.com/what-is-the-difference-between-deep-learning-and-reinforcement-learning/. [Geopend 8 Mei 2019].

[50] M. Sanjeevi, „Ch:13: Deep Reinforcement learning — Deep Q-learning and Policy Gradients (towards AGI)," Medium, 10 September 2018. [Online]. Available: https://medium.com/deep-math-machine-learning-ai/ch-13-deep-reinforcement-learning-deep-q-learning-and-policy-gradients-towards-agi-a2a0b611617e. [Geopend 27 Mei 2019].

[51] L. Hinzman, „Beating video games with deep Q networks - towardsdatascience," towardsdatascience, [Online]. Available: https://towardsdatascience.com/beating-video-games-with-deep-q-networks-7f73320b9592. [Geopend 9 Mei 2019].

[52] J. Hui, „RL — Trust Region Policy Optimization (TRPO) Explained," Medium, 21 Oktober 2019. [Online]. Available: https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9. [Geopend 23 Mei 2019].

[53] S. S., „Gradient Descent: All You Need to Know - Hacker Noon," Hacker Noon, 12 Maart 2018. [Online]. Available: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da. [Geopend 31 Mei 2019].

[54] P. Skalski, „How to train Neural Network faster with optimizers?," TowardsDataScience, 10 November 2018. [Online]. Available: https://towardsdatascience.com/how-to-train-neural-network-faster-with-optimizers-d297730b3713. [Geopend 31 Mei 2019].

[55] J. Hui, „RL — Proximal Policy Optimization (PPO) Explained," Medium, 17 September 2018. [Online]. Available: https://medium.com/@jonathan_hui/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12. [Geopend 23 Mei 2019].

[56] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup en D. Meger, „Deep Reinforcement Learning That Matters," AAAI Conference on Artificial Intelligence, April 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16669/16677. [Geopend 23 Mei 2019].

[57] J. Hui, „RL — Actor-Critic using Kronecker-Factored Trust Region (ACKTR) Explained," Medium, 11 September 2018. [Online]. Available: https://medium.com/@jonathan_hui/rl-actor-critic-using-kronecker-factored-trust-region-acktr-explained-670777ec65ce. [Geopend 23 Mei 2019].

[58] K. Budek, „Learning to run an example of reinforcement learning," deepsense, 22 Juni 2018. [Online]. Available: https://deepsense.ai/learning-to-run-an-example-of-reinforcement-learning/. [Geopend 3 Maart 2019].

Bijlagen

A. Omschrijving Bijlage A

Bijlage A is de datasplitter.py file. er kan een input locatie meegegeven worden en het pad vanaf de darknet folder. Als dit script uitgevoerd wordt genereert er een test.txt bestand en een train.txt bestand gemaakt met een verhouding van 90% trainingsdata en 10% testdata.

B. Omschrijving Bijlage B

Bijlage B bevat de configuratiefile van YOLOv3-tiny. Deze is enkel aangepast met de *batch, subdivision, width* en *heigth* variabelen. Afhankelijk van de kracht die een pc bezit kunnen deze variabelen aangepast worden.

C. Omschrijving Bijlage C

In bijlage C is het ReadAndWrite script te zien. Dit script wordt gebruikt om de output.txt file in te lezen en te filteren op honderdtallen. Dit zijn de gewichten die opgeslagen worden. Ook wordt enkel het iteratienummer en het gemiddelde weggeschreven en van laag naar hoog gefilterd in het filtered.txt bestand.

A. Omschrijving Bijlage A

```
import glob, os, sys, getopt

def main(argv):
    output_location = ''
    input_dir = ''

    try:
        opts, args = getopt.getopt(argv, "hi:o:", ["oloc=", "iloc="])
    except getopt.GetoptError:
        print('datasplitter -i <input_location> -o <path_from_darknet>')
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print('datasplitter -i <input_location> -o <path_from_darknet>')
            sys.exit()
        elif opt in ("-i", "--iloc"):
            input_dir = arg
        elif opt in ("-o", "--oloc"):
            output_location = arg

    percentage_test = 10

    file_train = open('train.txt', 'w')
    file_test = open('test.txt', 'w')

    counter = 1
    index_test = round(100 / percentage_test)

    for pathAndFilename in glob.iglob(os.path.join(input_dir, "*.jpg")):

        title, ext = os.path.splitext(os.path.basename(pathAndFilename))

        if counter == index_test:
            print('test: ' + pathAndFilename)
            counter = 1
            file_test.write(output_location + title + '.jpg' + '\n')
        else:
            print('train: ' + pathAndFilename)
            file_train.write(output_location + title + '.jpg' + '\n')
            counter = counter + 1

if __name__ == "__main__":
   main(sys.argv[1:])
```

B. Omschrijving Bijlage B

```
[net]
# Testing
# batch=1
# subdivisions=1
# Training
batch=64
subdivisions=32
width=1280
height=1280s
channels=3
momentum=0.9
decay=0.0005
angle=0s
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=128
size=3
```

```
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=1

[convolutional]
batch_normalize=1
filters=1024
size=3
        stride=1
        pad=1
        activation=leaky

        ###########

        [convolutional]
        batch_normalize=1
        filters=256
        size=1
        stride=1
        pad=1
        activation=leaky

        [convolutional]
        batch_normalize=1
        filters=512
        size=3
        stride=1
        pad=1
        activation=leaky

        [convolutional]
        size=1
        stride=1
        pad=1
        filters=81
        activation=linear

```

```
130.
131.       [yolo]
132.       mask = 3,4,5
133.       anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
134.       classes=22
135.       num=6
136.       jitter=.3
137.       ignore_thresh = .7
138.       truth_thresh = 1
139.       random=1
140.
141.       [route]
142.       layers = -4
143.
144.       [convolutional]
145.       batch_normalize=1
146.       filters=128
147.       size=1
148.       stride=1
149.       pad=1
150.       activation=leaky
151.
152.       [upsample]
153.       stride=2
154.
155.       [route]
156.       layers = -1, 8
157.
158.       [convolutional]
159.       batch_normalize=1
160.       filters=256
161.       size=3
162.       stride=1
163.       pad=1
164.       activation=leaky
165.
166.       [convolutional]
167.       size=1
168.       stride=1
169.       pad=1
170.       filters=81
171.       activation=linear
172.
173.       [yolo]
174.       mask = 0,1,2
175.       anchors = 10,14,  23,27,  37,58,  81,82,  135,169,  344,319
176.       classes=22
177.       num=6
178.       jitter=.3
179.       ignore_thresh = .7
180.       truth_thresh = 1
181.       random=1
```

C. Omschrijving Bijlage C

```
1.  import sched, time, sys
2.
3.  s = sched.scheduler(time.time, time.sleep)
4.
5.
6.  def main(sc):
7.
8.      with open("output.txt") as f:
9.          content = f.readlines()
10.
11.     print('Lines read.')
12.     content = [x.strip() for x in content]
13.     filtered = []
14.
15.     for line in content:
16.         if '00:' in line and 'avg' in line:
17.             number = line.split(" ")
18.             avg = line.split(",")
19.
20.             filtered.append("" + number[0] + avg[1])
21.     print('Lines filtered.')
22.     savefile = open("filtered.txt", "w")
23.
24.     filtered.sort(key=lambda x: float(x.split(' ')[-2]))
25.     print('Lines sorted.')
26.     for line in filtered:
27.         savefile.write("%s\n" % line)
28.     print('Lines saved to filtered.txt')
29.     s.enter(20, 1, main, (sc,))
30.     print('Waiting twenty seconds for refresh')
31.
32. if __name__ == "__main__":
33.    main(sys.argv[1:])
34.    s.run()
```