HO
GENT

Faculteit Bedrijf en Organisatie

Het aanpakken van ecologische problemen met behulp van artificiële intelligentie

Tibo Geysen

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Giselle Vercauteren
Co-promotor:
Piet Pieters

Instelling: —

Academiejaar: 2019-2020

Derde examenperiode

Faculteit Bedrijf en Organisatie

Het aanpakken van ecologische problemen met behulp van artificiële intelligentie

Tibo Geysen

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Giselle Vercauteren
Co-promotor:
Piet Pieters

Instelling: —

Academiejaar: 2019-2020

Derde examenperiode

Woord vooraf

De bachelorproef 'het aanpakken van ecologische problemen met behulp van artificiële intelligentie' vormt de afsluiting van mijn academisch traject. Het laat de student toe op een zelfstandige manier een onderzoek te voeren naar een onderwerp naar keuze. Voor mij was het de uitgelezen kans om iets wat mij nauw aan het hart ligt, te combineren met mijn passie. Vandaag de dag is er veel te doen omtrent vervuiling, een oplossing (of op zijn minst een stap richting een oplossing) kunnen bieden voor dit probleem aan de hand van machine learning was voor mij de drijfveer voor het kiezen van dit onderwerp.

Graag zou ik van deze gelegenheid ook gebruik willen maken om een aantal mensen, die mij zowel op technisch als mentaal vlak, hebben ondersteund en mij geholpen hebben bij het voltooien van deze bachelorproef.

In de eerste plaats wil ik graag mijn promotor, Giselle Vercauteren, bedanken. Mede door haar flexibiliteit en het vertrouwen in mij kon ik deze bachelorproef tot een goed einde brengen.

Vervolgens wil ik ook graag mijn co-promotor, Naam, bedanken om mij op technologisch vlak bij te staan en te voorzien van feedback en bijsturing waar nodig.

Tenslotte wil ik mijn vriendin, familie en vrienden bedanken voor de steun. Op sommige momenten is het schrijven van een bachelorproef stresserend. Zonder de hulp die ik van hen kreeg tijdens deze periodes had ik deze bachelorproef nooit tot een goed einde kunnen brengen.

Tibo Geysen

Gent, 21 augustus 2020

Samenvatting

Vervuiling van de aarde door middel van wegwerpplastic, opwarming van de aarde, ... Het zijn allemaal problemen veroorzaakt door de mens. De impact hiervan is vaak enorm. Een weinig gekend voorbeeld is de schade die sigarettenfilters aan het milieu doen (Green e.a., 2019). Het oplossen van dit probleem kan dus verregaande gevolgen hebben.

Er zijn heel wat oplossingen, gaande van voorkomen tot oplossen. De ultieme oplossing is hoogstwaarschijnlijk een combinatie van de beide. Bij het voorkomen van het probleem, denken we bijvoorbeeld aan het maken van biologisch afbreekbare filters. Het oplossen van het probleem kan door het opruimen van dit soort afval. Het is echter zeer onpraktisch dit werk enkel uit te besteden aan mensen, daar er jaarlijks zo'n 4.5 triljoen filters weggeworpen worden (Green e.a., 2019). Deze bachelorproef probeert na te gaan of de mensen kunnen bijgestaan worden bij het oplossen van dit probleem met behulp van artificiële intelligentie.

De bedoeling van dit werk is om een praktisch overzicht te geven van hoe artificiële intelligentie dit probleem kan helpen oplossen met behulp van object detectie. Eerst en vooral aan de hand van een theoretische inleiding, waar een soort van tijdslijn wordt opgebouwd naar de huidige state-of-the-art methode. Om vervolgens na te gaan of deze state-of-the-art methode toepasbaar is op dit probleem.

1. **Resultaat**: wat was het resultaat?
2. **Conclusie**: wat is/zijn de belangrijkste conclusie(s)?

Naar de toekomst toe is het mogelijk om dit onderzoek verder uit te breiden door gebruik te maken van de reeds uitgewerkte methodes en deze te linken aan een robot arm, die effectief overgaat tot het oprapen van het afval. Een andere mogelijkheid is uiteraard om na te gaan of de besproken methodes ook toepasbaar zijn op andere soorten afval (plastieken afvalzakken, conservenblikjes, ...).

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

1.1 Probleemstelling

Object detectie en object tracking is een belangrijk gegeven in computer vision. Het heeft heel wat applicaties in verschillende domeinen. Het wordt gebruikt om statistieken bij te houden over het verkeer en om files te voorspellen of vermijden. Supermarkten kunnen het gebruiken om het koopgedrag van hun klanten te analyseren. Een laatste voorbeeld is zelfrijdende voertuigen, object detectie en tracking zorgt ervoor dat de voertuigen zich bewust worden van hun omgeving (Porikli, 2006).

Onderzoek toont aan dat plastiek, die 10 procent van de totale hoeveelheid afval inneemt, een grote impact geeft op de omgeving (Thompson e.a., 2009). De filters van sigaretten, die voor 98 procent uit plastiek bestaan, spelen hier een grote rol in daar ze het meest weggeworpen plastiek wereldwijd zijn. Jaarlijks worden 4,5 triljoen (4 500 000 000 000) sigaretten filters gedumpt (Green e.a., 2019; Novotny e.a., 2009). Deze filters zijn slecht zichtbaar en worden vaak opgegeten worden door dieren, daarnaast breken ze af in miniscule componenten en beïnvloeden zo ook het voedsel van dieren, en dus onrechtstreeks ook ons (Bonanomi e.a., 2015).

Het kunnen detecteren van deze sigaretten filters is dus van belang voor het in stand houden van de natuur. Om deze reden zijn het vooral overheidsinstanties die van dit onderzoek gebruik kunnen maken om op openbare plekken waar veel mensen samenkomen (treinstation, festival, ...), proper te houden. Dit kan in samenwerking met de momenteel bevoegde instanties hiervoor.

1.2 Onderzoeksvraag

Onderzoek die gebruik maakt van object detectie voor het verzamelen van afval is al meerdere malen uitgevoerd (Awe e.a., 2017; Zaeni e.a., 2018). Wat deze bachelorproef specifiek tracht te onderzoeken is nagaan of dergelijke methodes ook toepasbaar zijn op het detecteren van sigaretten filters. De onderzoeksvraag luidt dus als volgt:

> "Zijn de huidige object detectie mechanismen in staat om verschillende types sigarettenfilters te detecteren?"

1.3 Onderzoeksdoelstelling

De doelstelling van deze bachelorproef is dus om na te gaan welke technieken het meest geschikt zijn om de onderzoeksvraag op te lossen. Dit werk vormt een basis voor een verdere 'real-life' implemenatie gekoppeld aan bijvoorbeeld een robot arm. Uiteindelijk zal er ook een proof-of-concept gemaakt worden in de vorm van een applicatie verbonden met een application programming interface (hierna API).

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken. In dit hoofdstuk wordt nagegaan in welke andere doeleinden object detectie gebruikt kan worden en voor welke gelijkaardige doeleinden ze al gebruikt worden. Hier wordt stilgestaan bij de verschillende technieken. Deze worden echter niet theoretisch uitgelegd, maar praktisch.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de technieken besproken in onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 5 wordt besproken hoe we de voorgaand besproken methodes in de realiteit kunnen maken. Hier wordt stilgestaan bij de gebruikte modules, het platform om de modellen te maken en hoe we die in een applicatie kunnen verwerken.

In Hoofdstuk 6 zullen we vervolgens de resultaten meegeven die uit de praktische uitwerking komen van het voorgaande hoofdstuk. We zullen deze tevens ook bespreken.

In Hoofdstuk 7, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Literatuurstudie

Dit hoofdstuk bevat de literatuurstudie. Het doel van dit hoofdstuk is om de lezer een 'high-level' inzicht te geven van wat object detectie is. Het probleem, namelijk vervuiling door plastiek, wordt kort uitgewerkt om de nood tot een oplossing aantonen. Vervolgens volgt een overzicht van waar object detectie reeds gebruikt wordt, zowel voor het oplossen van vervuiling als voor andere zaken, en welke technieken er zijn om aan object detectie te doen. De technieken zullen besproken worden in de volgorde van hun ontdekking en zullen steeds ook de voordelen ten opzichte van de voorgaande methode bespreken. Tenslotte zullen we uitkomen bij de state-of-the-art op vlak van object detectie.

2.1 Probleemstelling

Het is algemeen geweten dat afval een negatieve impact kan hebben op onze omgeving. Dit hangt grotendeels af van wat we met dit afval doen, wegwerpen, vernietigen of recycleren. Deze sectie zal hier wat dieper op ingaan om het probleem duidelijker te maken en meer specifiek aan te tonen wat de effecten zijn van vervuiling op onze omgeving.

2.1.1 Afval en vervuiling

Eerst en vooral dient er een onderscheid gemaakt te worden tussen afval en vervuiling. Niet alles dat afval is, is vervuiling en omgekeerd. Ze zijn echter wel vaak gelinkt aan elkaar. Een onderscheid tussen deze werd gemaakt door on Environmental Pollution (Great Britain) (2003). Afval wordt gedefinieerd als ongewenste resten van een product die meestal als negatief gezien worden. Vervuiling daarin tegen wordt gedefinieerd als de introductie van

substanties, materialen of energie in de natuur door mensen, die een schadelijke impact hebben op mens en dier. Het is dus duidelijk dat niet alle afval vervuiling is, dit wanneer we het hergebruiken. Afval wordt pas vervuiling wanneer we het wegwerpen of vernietigen en dit proces een negatieve impact heeft op de omgeving.

2.1.2 Effecten van plastiek vervuiling

Onderzoek toont aan dat plastiek, die 10 procent van de totale hoeveelheid afval inneemt, een grote impact geeft op de omgeving (Thompson e.a., 2009). Een onderzoek van Panagos e.a. (2013), toonde aan dat er in Europa ongeveer 1,170,000 potentieel vervuilde grond plaatsen zijn, waarvan 10% bevestigd als vervuild. Van deze 10% zijn er reeds 45% opgeruimd.

Grondvervuiling creëert een significant risico voor de gezondheid van de mens. Door het wegwerpen van afval komen verschillende chemicaliën in de grond die via de vervuiling van de grond ook ons drink water en voedsel vervuilen. Wanneer plastiek bijvoorbeeld in contact komt met de grond, kunnen residuen ervan afbreken in de grond en zo de plantengroei tegenwerken (Green e.a., 2019). Een andere manier waarop weggeworpen plastiek zorgt voor vervuiling is dat het in de zee terechtkomt waar het afgebroken wordt en opgenomen wordt door plankton, het voedsel van vele zeedieren (Panagos e.a., 2013). Via deze cyclus komt het uiteindelijk ook terug bij ons. Verschillende studies naar de inhoud van vissen treffen bijvoorbeeld significante hoeveelheden micro plastiek aan (Güven e.a., 2017; Lusher e.a., 2013).

Vanaf zijn uitvinding tot 2015 werd 6,3 miljard ton plastiek afval gegenereerd, hiervan is 9% gerecycleerd, 12% verbrand en bevindt er zich 79% op stortplaatsen en in de natuur. Als de huidige productie zo verder gaat en er geen actie wordt ondernomen om de huidige afval verwerking te verbeteren zal er tegen 2050 12,5 miljard ton plastiek afval zich bevinden in stortplaatsen en de natuur. Het mag dus duidelijk zijn dat er een probleem is.

2.1.3 Sigaretten filters

Jaarlijks worden 4,5 triljoen (4 500 000 000000) sigaretten filters gedumpt (Green e.a., 2019; Novotny e.a., 2009). Deze filters bestaan voor 98% uit plastiek en spelen dus een grote rol in het weggeworpen plastiek. Ze zijn ook het vijfde meest voorkomende plastiek afval in de natuur (Novotny & Zhao, 1999). Deze filters zijn slecht zichtbaar en worden vaak opgegeten door dieren. Een andere manier waarop deze filters onze omgeving vervuilen is, zoals eerder vermeldt, dat plastiek zich afbreekt in minuscule componenten en zo in ons water terechtkomt, het water van vissen, het voedsel van dieren en dus ook weer onrechtstreeks bij ons (Bonanomi e.a., 2015; Novotny e.a., 2011). In uitzonderlijke gevallen komt het zelf voor dat kinderen deze filters op eten met symptomen van vergiftiging tot gevolg (Novotny e.a., 2011).

De huidige methodes van het voorkomen van giftige filters in onze omgeving zijn, biologisch afbreekbare filters, het bestraffen van mensen die betrapt worden op het wegwerpen

van filters en opruimen (Novotny & Zhao, 1999). Het is duidelijk dat we dus afhankelijk zijn van de 'goedheid' van mensen of ze hun filter al dan niet wegwerpen. Het is echter moeilijk om te 'targetten' wie er vervuilt en wie niet, vervuilers herkennen hun gedrag zelf vaak niet als vervuilend, wat het concept om op hun goedheid te rekenen niet efficiënt maakt (Smith & Novotny, 2011). Als we dus zoveel mogelijk willen filters uit onze omgeving houden moeten we er een en-en verhaal van maken. We moeten en sensibiliseren, en opruimen en onderzoek doen naar biologisch afbreekbare filters. Deze bachelorproef zal zich focussen op onderzoek naar het opruimen ervan.

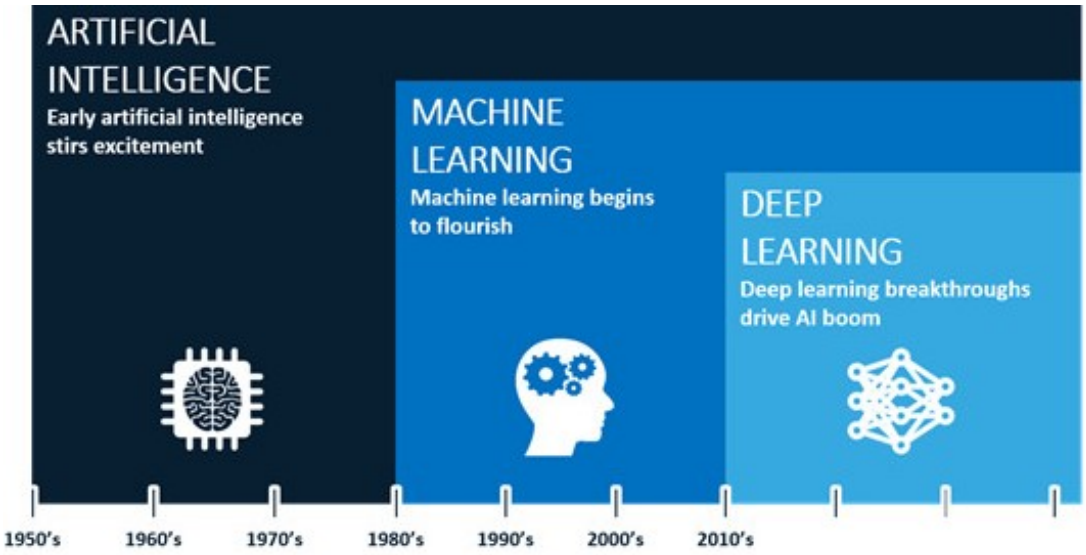
2.1.4 Oplossingen

What You Can Do About Plastic Pollution Use our plastic calculator to track how much plastic you use. Download our End Plastic Pollution Toolkits. Find ways to reduce your plastic waste. Make a pledge to reduce your use of plastic. Stay updated on news about progress in the fight to end plastic pollution. Send your ideas or propose a partnership to plastic@earthday.org.

2.2 Wat is artificiële intelligentie?

2.2.1 AI versus ML versus DL?

Zoals eerder aangehaald kan object detectie gedaan worden aan de hand van artificiële intelligentie (hierna AI). Deze benaming wordt vaak verward met 'machine learning' (hierna ML) en 'deep learning' (hierna DL) door de nauwe relatie tussen deze concepten.



Figuur 2.1: Artificiële intelligentie versus 'machine learning' versus 'deep learning' (USoft, 2020)

Zoals duidelijk wordt uit figuur 2.1, is AI een allesomvattende definitie. Het omvat alle

programma's die intelligentie demonstreren in machine en niet door mensen. Dit omvat zowel gewone programma's met if-else statements, maar ook ML en DL. De grote vraag hier is: 'wat is intelligentie?'

Een subset van AI is ML. De term werd het eerst gebruikt door Samuel (1959). ML algoritmes bieden machines de mogelijkheid om iets te leren op basis van wat ze voorgaand gezien hebben. Ze leren een mathematisch model aan de hand van voorbeelden, die we training data noemen, om voorspellingen te kunnen maken zonder expliciet geprogrammeerd te worden. ML algoritmes kunnen dus zelf leren welke zaken van belang zijn om een voorspellin te doen of beslissing te maken, op basis van wat ze voorgaand gezien hebben.

De huidige hype rond AI komt echter niet van ML maar van DL, die een subset is van ML. Het bestaat al langer dan de hype, maar door recente verbetering in computationele kracht door gebruik van 'graphical processing units' (hierna GPUs) zijn ze populair geworden. Het grote verschil tussen ML en DL is dat bij ML de kenmerken van je dataset gekend moeten zijn, terwijl DL algoritmes deze kenmerken door non-lineaire transformaties kunnen leren.

2.2.2 Verschillende types van leren

In het geval van ML en DL zijn er een aantal manieren waarop een machine kan leren. De drie gekendste vormen van leren in ML en DL zijn gesuperviseerd leren, ongesuperviseerd leren en 'reinforcement learning'. Deze bachelorproef focust zich op gesuperviseerd leren.

Gesuperviseerd leren

In supergeviseerd leren zullen we trachten een mathematische functie op te stellen die een input kan transformeren naar een output. Deze mathematische functie leert het op basis van input-output paren, die we de training data noemen. Elk paar in de training data bevat dus informatie, die we de variabelen noemen, en een uitkomst, die we de labels noemen. Wanneer deze mathematische functie opgesteld is kunnen we deze gebruiken om er nieuwe input aan te voeren, die dan op basis van deze functie een output zal geven, die we de voorspelling noemen (Mohri e.a., 2012; Russell & Norvig, 2010).

Hoe deze mathematische functie tot stand komt is door een zogenaamde 'loss function'. Het model heeft een aantal parameters, deze dient het vervolgens te optimaliseren zodanig dat het op basis van de training data zo weinig mogelijk fouten maakt. Om echter later goeie voorspellingen te kunnen maken moeten we ook het juiste model kiezen voor het probleem. Een model kan bijvoorbeeld zaken leren uit de training data, die niet voorkomen in het echt. Het is dus van belang om rekening te houden met het bias-variantie dillemma, die stelt dat we een model moeten kiezen die zowel goed presteert op de training data, maar ook op niet geziene data (Geman e.a., 1992). In Hoofdstuk 3 wordt dieper ingegaan op hoe deze mathematische functie tot stand komt.

Een voorbeeld van gesuperviseerd leren in computer vision is wanneer we tien foto's hebben. Deze foto's zijn de features van het input-output paar. De labels van deze foto's

kunnen verschillende zaken zijn, als voorbeeld nemen we vijf katten en vijf honden. Op basis van deze foto's, moet een model instaat zijn om te bepalen of de foto een hond is of een kat.

Ongesuperviseerd leren

In het voorgaande stuk bespraken we gesuperviseerd leren. Ongesuperviseerd leren is de tegenhanger hiervan. Ongesuperviseerd leren is een manier van leren die opzoek gaat naar patronen in de data. Het verschil met gesuperviseerd leren is dat er in ongesuperviseerd leren geen label beschikbaar is. Bij ongesuperviseerd leren gaan we opzoek naar de onderliggende distributie van de variabelen (Hinton e.a., 1999). Dit stelt ons instaat om bijvoorbeeld bepaalde datapunten in een dataset te groeperen.

Enkele voorbeelden van ongesuperviseerd leren in computer vision zijn het clusteren van foto's en het genereren van nieuwe foto's.

Reinforcement learning

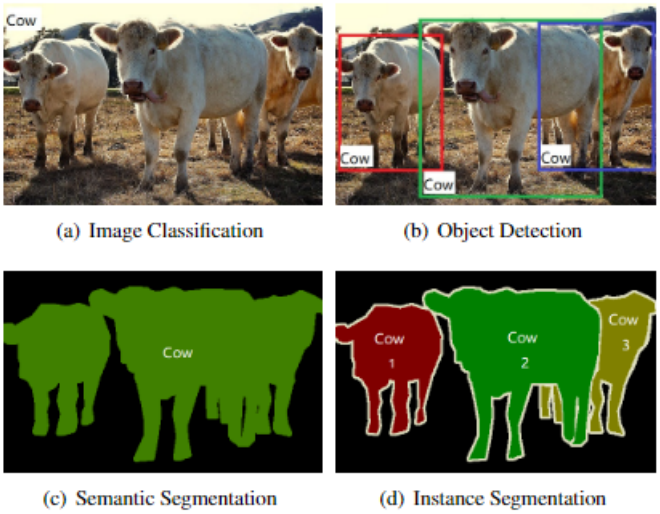
Tenslotte is er nog reinforcement learning. Deze vorm van leren is verschillend van de twee voorgaande daar het noch variabelen noch labels gebruikt om te leren. Het doel van deze vorm van leren is om een 'agent', bijvoorbeeld een robot, te leren welke actie hij moet uitvoeren in zijn omgeving. Het leert dit op basis van een trial-and-error manier (Bernstein & Burnaev, 2018), waarbij goeie beslissingen beloond worden. Het belangrijkste concept in reinforcement learning is om een balans te vinden tussen het exploreren van ongekend terrein en het gebruiken van reeds verworven kennis (Sutton & Barto, 2018).

Reinforcement learning kan ook gebruik maken van computer vision. Naast de sensoren in een 'agent', kan het ook nuttig zijn voor een 'agent' om de omgeving te zien. Op basis van deze visuele input, de acties die een 'agent' onderneemt en de beloning die ermee eventueel mee gepaard gaat, kan de 'agent' leren beslissingen nemen. .

2.3 Computer vision

Wanneer mensen een foto waarnemen, zien ze direct welke objecten zich in de foto bevinden, waar deze zich bevinden en hoe deze met elkaar verbonden zijn. Dit laat ons toe om complexe zaken uit te voeren, zonder er erg bij stil te hoeven staan. Dit is het uiteindelijke doel die we voor robotica willen bereiken. Zodat deze bepaalde taken kunnen uitvoeren, die een mens niet wenst uit te voeren of niet kan uitvoeren.

'Computer vision' is een onderzoeksveld dat zich hier mee bezighoudt. Computer vision heeft twee grote doelen. Ten eerste wenst het te kunnen uitleggen hoe mensen zaken waarnemen, ten tweede houdt het zich bezig met het schenken van zicht aan machine. Met behulp van computer vision proberen we zaken te automatiseren die mensen kunnen door hun zicht. Dit probeert het te doen door informatie te halen uit foto- en videomateriaal. In het geval van fotomateriaal spreken we over 2D data zonder tijdsvariante, een foto is



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

Figuur 2.2: Voorbeelden van object herkenning (a), detectie (b) en segmentatie (c, d) (X. Wu e.a., 2020)

statisch. In het geval van videomateriaal is er een temporale correlatie tussen opeenvolgende beelden, wat er zich eerder afspeelde kan belangrijk zijn, hier spreken we over 3D data (Huang, 1996). Deze bachelorproef spitst zich toe op het tweede doel van computer vision, namelijk het automatiseren van een taak die mensen kunnen door hun zicht.

Computer vision heeft een aantal subdomeinen (Morris, 2004), waarvan we er 4, diegene van belang voor de bachelorproef, zullen bespreken. Drie hiervan (herkenning, detectie en segmentatie) zijn weergegeven in figuur 2.2.

- Object herkenning
- Object detectie
- Object segmentatie
- Video tracking

2.3.1 Object herkenning

In object herkenning of object classificatie, terug te vinden in figuur 2.2 (a), is het doel om op basis van een foto te bepalen wat er op de foto staat (X. Wu e.a., 2020).

2.3.2 Object detectie

Object detectie, terug te vinden in figuur 2.2 (b), gaat een stap verder dan object herkenning. Naast het correct kunnen bepalen wat er zich op de foto afspeelt, is het bijkomende doel van object detectie om ook te kunnen bepalen waar zich dit afspeelt (X. Wu e.a., 2020). Het doet dit door middel van een omkadering. Een meer formele definitie van object detectie is de volgende (Dasiopoulou e.a., 2005):

'Object detectie is een computer technologie gerelateerd aan computer visie dat

met behulp van artificiële intelligentie instanties van objecten met gelijkaardige kenmerken kan detecteren in foto's en video's.'

2.3.3 Object segmentatie

Object segmentatie is een verdere uitdieping van object detectie. Waar object detectie grofweg bepaalt waar een object zicht bevindt in een foto, zal object segmentatie zo exact mogelijk proberen een object te segmenteren van de rest van de foto. Het doet dit door aan elke pixel in een foto een bijhorende klasse toe te kennen.

In het geval van gewone segmentatie, terug te vinden in terug te vinden in figuur 2.2 (c), wordt geen onderscheid gemaakt tussen verschillende objecten van dezelfde categorie. Dit is waar instantie segmentatie, terug te vinden in figuur 2.2 (d), probeert een oplossing te bieden. Het is een relatief nieuw concept in computer vision en is gedefinieerd als een intersectie tussen object detectie en semantische segmentatie (X. Wu e.a., 2020).

2.3.4 Object tracking

Tenslotte is er ook object tracking. Voor de applicatie van deze bachelorproef is het minder belangrijk. Het is echter wel interessant om aan te halen om een eventuele uitbreiding naar een robotarm te vergemakkelijken. Object tracking tracht een object te volgen. Het doet dit door op basis van de huidige staat (positie en grootte) van een object in een frame van een video, te voorspellen wat de staat (positie en grootte) van het te volgen object in toekomstige frames zal zijn (Y. Wu e.a., 2013).

Volgens Rosebrock (2018) kan object tracking gezien worden als een proces bestaande uit 3 stappen.

1. Neem een initiële set van gedetecteerde objecten (coördinaten van de omkaderingen)
2. Creeër een unieke ID voor elke eerste detectie
3. Volg elk object wanneer ze zich bewegen tussen frames in, met behoud van hun uniek ID

Het algoritme die we hiervoor zullen gebruiken, namelijk 'centroid tracking', wordt uitvoeriger besproken in Hoofdstuk 3.

2.4 Wat is reeds gedaan?

De laatste sectie van dit hoofdstuk heeft een kort overzicht van recent, praktisch gebruik van bovenstaande methodes. Zowel in het domein van recyclage als andere domeinen, zoals de medische wereld.

2.4.1 Recyclage

2.4.2 Andere vakdomeinen

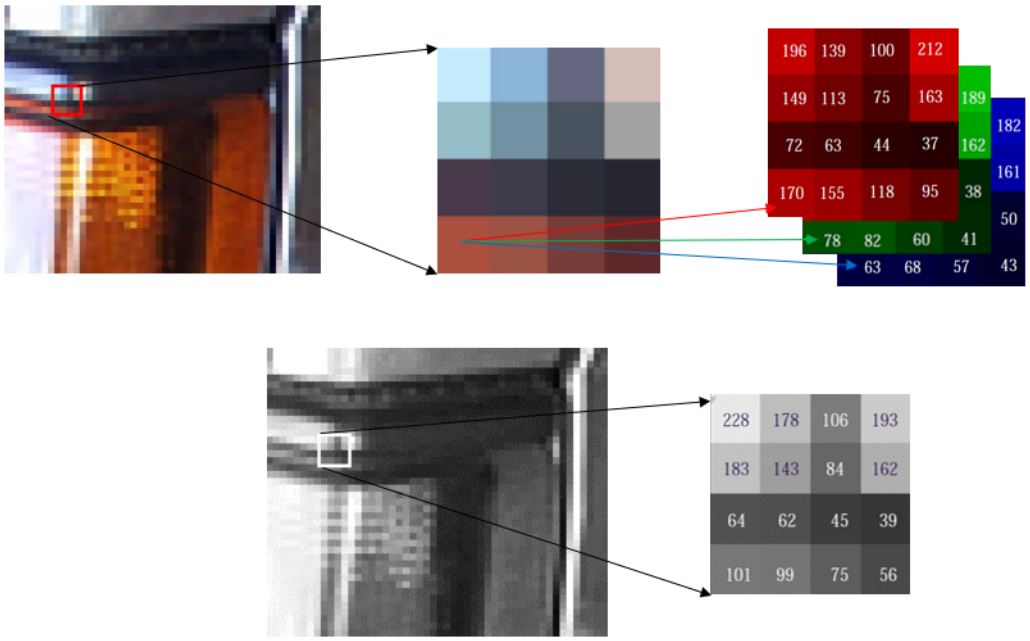
3. Methodologie

In deze sectie zullen we dieper in gaan op de methoden voor object herkenning, detectie, tracking, ... Eerst en vooral zullen we stilstaan bij wat een foto en video precies zijn. Dit is van uiterst belang om een beter beeld te kunnen scheppen over hoe machines kunnen leren wat er zich op een foto bevindt. Vervolgens zullen we dieper in gaan op wat de methoden bij machine learning en deep learning zijn. Bij machine learning staan we stil bij ..., bij deep learning gaan we na wat neurale netwerken precies zijn, wat hun voordelen zijn, wat hun nadelen zijn en hoe ze in staat zijn om complexe zaken te leren, we werken hierbij ook een specifiek voorbeeld uit. Vervolgens bespreken we convolutionele netwerken, een specifiek type neurale netwerken ontworpen om aan computer vision te doen. Daarna staan we stil bij transfer learning, die ons in staat stelt om vooraf getrainde modellen te gebruiken met een hoge accuraatheid. Tenslotte gaan we nog over de evaluatie metrieken, waarmee we kunnen nagaan hoe goed ons model is.

3.1 Wat is een foto?

Om te begrijpen hoe machinaal leren en deep learning foto's kunnen verwerken en ze gebruiken om aan object detectie en herkenning te doen, is het belangrijk om te weten wat een foto precies is. Gonzalez en Woods (2018) definiëren een digitale foto als een compositie van elementen, genaamd pixels. Elk van deze pixels wordt voorgesteld als een eindige discrete numerieke waarde. Deze waarde bepaalt de intensiteit en kleur van de pixel of in het geval van een zwart-wit foto de grijswaarde.

We kunnen in het geval van een computer, een foto dus zien als een verzameling van nummers in een twee-dimensionale array met een vast aantal rijen en vast aantal kolommen.



Figuur 3.1: Kleuren foto array representatie (bovenaan), zwart-wit foto array representatie (onderaan) (2020)

In het geval van zwart-wit foto's, is elke pixel gelijk aan een waarde tussen 0 en 255, die de intensiteit van de pixel bepaalt. In de meeste gevallen is 0 gelijk aan zwart en 255 gelijk aan wit. De waarden ertussen zijn verschillende schaduwen grijs. Om kleur foto's voor te stellen is elke pixel een vector van 3 nummers. Een kleurenfoto bestaat dus uit 3 lagen, een rode, groene en blauwe laag waarvan de combinatie van de 3 nummers een kleur voorstelt volgens de RGB kleurencode. Een voorbeeld van hoe een zwart-wit foto en kleurenfoto er uitzien in een array vorm is weergegeven in figuur 3.1.

3.2 Wat is een video?

Nu we een definitie vastgesteld hebben van wat een foto is, kunnen we dit verder uitdiepen naar wat een video precies is. Een video wordt gedefinieerd als een visueel medium voor het opnemen, kopiëren, terugspoelen en weergeven van bewegend beeld („An introduction to the Principles of Video 101", g.d.). Het is dus een verzameling van opeenvolgende foto's. Een video wekt dus enkel de perceptie op van beweging door het snel afspelen van foto's na elkaar. Dit concept wordt beschreven door de 'frame rate' of het aantal foto's per tijdseenheid. Mensen kunnen tussen 10 en 12 foto's per seconde verwerken, meer foto's per seconde wordt gepercipieerd als beweging (Read & Meyer, 2000). Dit concept van het omzetten van foto's naar video is van belang voor object tracking omdat ons model de informatie van voorgaande frames kan gebruiken om te voorspellen waar het object naar toe zal bewegen. Deze informatie tussenin opeenvolgende frames noemen we temporale correlatie. Hoe dichter 2 frames zich bij elkaar afspelen, hoe groter deze correlatie (Mahmood & Khan, 2007).

3.3 Machinaal leren

De ML methoden die we bespreken zijn opgenomen om een volledig beeld te creëeren van de beschikbare methoden om aan object detectie te doen. We zullen ze echter slechts heel kort bespreken omdat ze afhankelijk zijn van zelf te creëeren variabelen, die soms jaren onderzoek vergen, waar de DL methoden dit zelf kunnen.

3.3.1 Scale-invariant feature transform

3.3.2 Viola–Jones object detection framework

3.4 Deep learning

Zoals eerder vermeldt is het grote voordeel van DL methoden dat ze in staat zijn om zelf variabelen te ontdekken die belangrijk zijn voor het herkennen of detecteren van een object in een foto. Door deze zelf-detectie van variabelen kunnen deze methoden op een heel aantal foto classificatie problemen los gelaten worden zonder elk probleem afzonderlijk eerst te analyseren.

3.4.1 Neurale netwerken

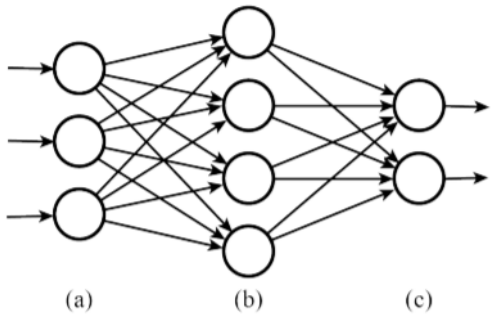
In tegenstelling tot wat vaak wordt gedacht zijn neurale netwerken en deep learning geen nieuwe concepten. Reeds in 1942 werd een eerste poging gedaan tot het creëeren van een neural netwerk (McCulloch & Pitts, 1943). Artificiële neurale netwerken vinden hun oorsprong terug in biologische neurale netwerken en zijn opgebouwd uit verschillende lagen die verbonden zijn door neuronen (figuur 3.2). Figuur 3.2 geeft een voorbeeld weer van wat we een 'fully feed-forward neural network' of FFFNN noemen. Elke neuron is verbonden met elke neuron in een voorwaartse graaf. Deze combinatie van verschillende eenvoudigere functies, die elke individuele neuron bevat, tot complexere non-lineaire functies zorgt ervoor dat we complexe problemen met neurale netwerken kunnen oplossen. Deze functie kan op een simpele manier weergegeven worden 3.1, waar 'x' de input is, 'f(x)' een afbeelding van x ten opzichte van y, en 'y' de output.

$$f(x) = y \tag{3.1}$$

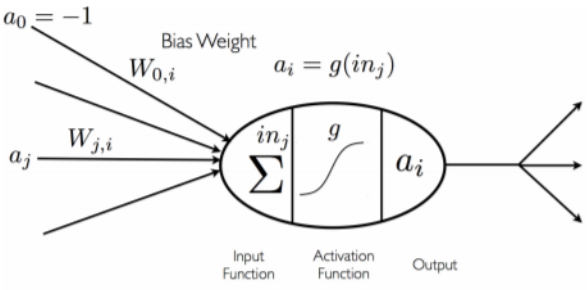
De neuron

In deze sectie wordt dieper ingegaan op wat een neuron precies is en doet. Neuronen kunnen ingedeeld worden in 3 groepen.

1. **Input neuronen**: deze neuronen ontvangen de input van je netwerk. Dit kan bijvoorbeeld een foto zijn, natuurlijk zoals eerder besproken in een array.



Figuur 3.2: Neuraal netwerk met 3 lagen. Een eerste inputlaag met 3 neuronen, een verborgen laag met 4 neuronen en een outputlaag met 2 neuronen. (Russell, 2016)



Figuur 3.3: Grafische voorstelling van 1 neuron. Een neuron combineert de gewogen (Russell, 2016)

2. **Verborgen neuronen**: van deze neuronen kennen we de output niet. Daarom worden ze verborgen neuronen genoemd. Hun input is de output van voorgaande neuronen.
3. **Output neuronen**: deze neuronen geven ons de output van het netwerk. In het geval van een classificatie probleem kunnen die meerdere outputs zijn. In het geval van een regressie probleem is dit 1 output.

We nemen er figuur 3.2 bij ter illustratie. De eerste laag in figuur bevat 3 input neuronen. In het geval van een foto krijgt elke neuron hier dezelfde foto binnen, het verwerkt deze foto in een neuron, waar we verderop dieper op in gaan, en produceert een output. Deze output wordt dan verder door het netwerk gestuurd naar een volgende laag. Merk op dat in dit geval, de output van 1 neuron doorgestuurd wordt naar elke neuron in de volgende laag. In deze volgende laag, die we een verborgen laag noemen, gebeurt weer hetzelfde. Echter komen hier in plaats van 1 input, 3 inputs in 1 neuron binnen. Deze voorwaartse pijlen bevatten leerbare gewichten, die de input wegen.

Elke neuron heeft een input functie, een activatie functie en produceert een output. De input functie kan geschreven worden als in_j (3.2 en is een lineaire combinatie van gewichten en de input van de neuron. Waar 'j' de input van de neuron specificeert, en 'i' de neuron zelf specificeert.

$$in_j = \sum_{i=1}^{n}(w_{i,j}a_i) + b_j \tag{3.2}$$

Om non-lineariteit in het network te introduceren wordt deze lineaire combinatie door een activatie functie gestuurd 3.3. Een aantal populaire, elk met hun voor- en nadelen en hun use cases, zijn de sigmoid activatiefunctie (3.4), de ReLU activatiefunctie (3.5) en de softmax activatie functie (3.6)

$$a_i = g(in_j) = g(\sum_{i=1}^{n}(w_{i,j}a_i) + b_j) \tag{3.3}$$

$$Sigmoid(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.4}$$

$$ReLU(x) = f(x) = \begin{cases} 0, & \text{if } x < 0. \\ 1, & \text{anders} \end{cases} \tag{3.5}$$

$$Softmax(x) = \frac{\exp(x)}{\sum_j \exp(x_j)} \tag{3.6}$$

Een intuïtieve uitleg van deze activatiefuncties volgt hieronder:

1. **Sigmoid**: na deze activatiefunctie is de output een waarde tussen 0 en 1. Dit maakt het mogelijk om bijvoorbeeld probabiliteiten te voorzien. Grotere waarden die uit de inputfunctie komen zullen, naar door een sigmoid functie gestuurd te zijn, dichter naar 1 gaan.
2. **ReLU**: de sigmoid functie brengt, in het geval van veel verborgen lagen, een vervelend probleem met zich mee, namelijk het 'vanishing gradient' probleem. Hier wordt niet dieper op ingegaan. De ReLU activatiefunctie biedt hiervoor een oplossing. Wanneer een output van een inputfunctie kleiner is dan 0, dan wordt de output na de activatiefunctie 0, anders blijft de waarde zoals ze is.
3. **Softmax**: wanneer ons netwerk verschillende objecten moet kunnen classificeren, wensen we een output die verschillende probabiliteiten, tussen 0 en 1, output. Een bijkomende vereiste echter is dat de som deze verschillende probabiliteiten niet groter mag zijn dan 1. Dit is waar de softmax activatiefunctie voor zorgt. Na een softmax activatiefunctie krijgen we bijvoorbeeld een output [0.01 (kat); 0.99 (hond)], deze stelt dat ons netwerk met 99% zekerheid zegt dat de input die het kreeg een hond is.

Hoe leert een neuraal netwerk?

In de voorgaande sectie besproken we de sleutelelementen waaruit een neuraal netwerk architectuur bestaat. De vraag die we in deze sectie zullen beantwoorden is hoe zo een neuraal netwerk de functie 3.1 leert om zo een accuraat mogelijke predicties te kunnen doen. De bedoeling is dat het neuraal netwerk de gewichten $w_{i,j}$ leert zodat het een input

x, kan transformeren aan de hand van de functie $f(x)$, naar y (zie figuur 3.2 en figuur 3.3). Echter wanneer we een neuraal netwerk opstellen, heeft het deze gewichten nog niet geleerd omdat het niet weet wat het moet leren, de gewichten zijn random of volgens een bepaalde distributie geïnitialiseerd.

Het proces van leren kan opgedeeld worden in 2 fases. Eerst is er de 'forward propagation'. In deze fase van het proces worden een aantal voorbeelden door het netwerk gestuurd. Elke neuron berekent een activatie en stuurt deze verder door het netwerk tot we bij de outputlaag komen die een output berekent. In de eerste iteraties van dit proces, zullen de outputs niet correct zijn omdat het netwerk de gewichten nog niet heeft kunnen leren. Dit is waar de 'backward propagation' in beeld komt, de tweede fase van het proces. De output van het netwerk wordt vergeleken met de reële output van de voorbeelden die door het netwerk gestuurd werden. Het verschil tussen de output van het netwerk en de reële output is wat we de foutscore noemen, hoe fout is ons netwerk? Op basis van deze fout zullen de gewichten aangepast worden, waar een grotere fout, een grotere aanpassing van deze gewichten betekent.

Om aan deze 'backward propagation' te kunnen doen hebben we dus 2 zaken nodig. Een verliesfunctie die voor de verschillende voorbeelden door het netwerk gestuurd een distributie toont van hoe fout ons netwerk is, en een algoritme die de gewichten kan aanpassen zodat wanneer de volgende voorbeelden door het netwerk gestuurd worden, het verlies kleiner is.

Verliesfunctie

Net zoals er verschillende activatie functies bestaan, bestaan er ook verschillende verlies functies, weer elke met hun specifieke use case. Ter illustratie bespreken we de loss functie gebruikt voor 'multi-class' classificatie problemen, de categorische cross entropy loss of L_{CCE} (3.7), omdat deze vaak gebruikt wordt voor het correct classificeren van objecten op een foto. Waar 'n' het aantal voorbeelden zijn waarop we de verliesfunctie baseren, 'v' het aantal mogelijke klasses zijn (bijvoorbeeld kat en hond zijn 2 klasses), 'y' de werkelijke waarde is en $\hat{y}$ de voorspelde waarde.

$$L_{CCE} = J(w) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{v} y_{i,j} log(\hat{y}_{i,j}) \tag{3.7}$$

Gradient descent algoritme

Op basis van deze verliesfunctie die we zonet gedefinieerd hebben, kunnen we een algoritme toepassen die de gewichten van het netwerk aanpast zodanig dat we bij volgende iteraties een lagere foutscore hebben. Dit algoritme noemt het 'gradient descent' algoritme. Het is een zoekalgoritme die probeert om de foutscore steeds te verbeteren, in het geval van L_{CCE} dus verkleinen. Intuïtief gezien is de verliesfunctie een gebergte en probeert het 'gradient descent' algoritme een stapje te nemen in de richting waar de verliesfunctie $J(\theta)$ het sterkst daalt (3.7). Dit doen we door de partieel afgeleide van alle gewichten w_i

te berekenen, we noemen dit de gradient, wat de richting van de sterkste stijging in een functie is. Aangezien we de sterkste daling willen berekenen van de verliesfunctie zullen we uiteraard het negatieve gradient gebruiken.

$$\nabla J(w) = (\frac{\partial}{\partial_1}, \frac{\partial}{\partial_2}, ..., \frac{\partial}{\partial_n}) J(w) = 0 \tag{3.8}$$

Op basis van deze gradient, kunnen we onze gewichten aanpassen aan de hand van een learning rate α. Deze constante factor bepaalt met hoeveel we onze gewichten updaten, het betekent intuïtief hoe hard we geloven in de berekende gradient.

$$w_i \longleftarrow w_i - \alpha \frac{\partial}{\partial_i} J(w) \tag{3.9}$$

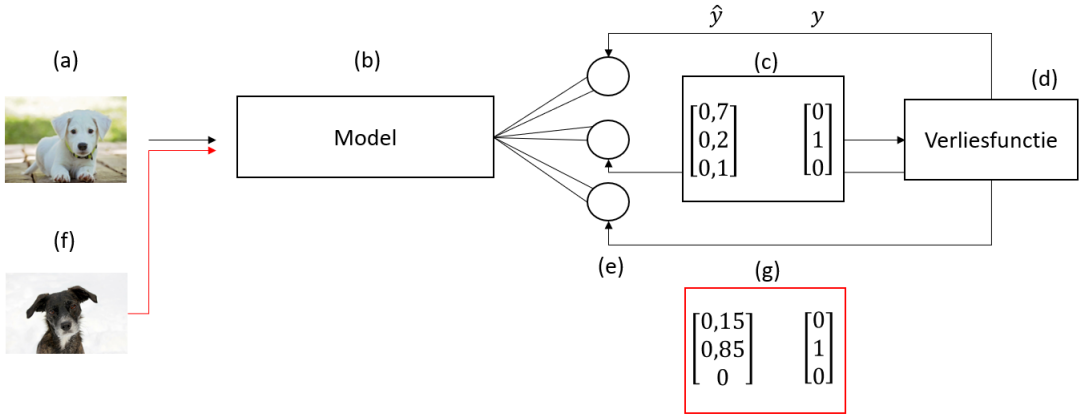
Samenvattend leert een neuraal netwerk dus door eerst een forward propagation te doen waar we een aantal voorbeelden door ons netwerk sturen. Deze worden vergeleken met de reële waarden, wat ons een verliesfunctie geeft. Met behulp van het 'gradient descent' algoritme kunnen we voor elk gewicht in het netwerk bepalen wat de bijdrage was van een bepaald gewicht tot de verliesfunctie. Deze gradient wordt gedurende een backward propagation berekend, de fouten worden dus omgekeerd door het netwerk teruggestuurd.

Voorbeeld

Laten we dit alles illustreren met een voorbeeld. In tabel 3.1 hebben we onze dataset. 2 foto's van een kat, 2 van een hond en 1 van een vis. Dit zijn onze variabelen. De labels worden voorgesteld aan de hand van een one-hot encoded vector [Kat, Hond, Vis]. Dit betekent dat een label bestaat uit 0'en en 1'en, waar een 0 betekent dat het niet tot die klasse behoort, en een 1 betekent dat het wel tot die klasse behoort. In het geval van Kat 1 is deze vector dus [1, 0, 0], wat betekent dat we 100% (1) zeker zijn dat dit een kat is, 0% (0) zeker zijn dat het een hond is (of dus 100% dat het geen hond is), en 0% (0) zeker dat het een vis is. Het doel is dus om een model te maken, die op basis van de waarden (pixels) op een foto, kan bepalen of het een kat, hond of vis is.

Foto	Kat	Hond	Vis
Kat 1	1	0	0
Kat 2	1	0	0
Hond 1	0	1	0
Vis 1	0	0	1
Hond 2	0	1	0

Tabel 3.1: Voorbeeld van een dataset met 2 katten, 2 honden en 1 vis



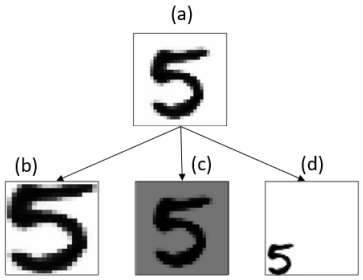
Figuur 3.4: Grafische voorstelling van hoe een neuraal netwerk leert honden, katten en vissen te classificeren

Figuur 3.4 toont visueel aan hoe een neuraal netwerk leert. In een eerste iteratie krijgt ons model een foto (de pixelwaarden) te zien van een hond (stap a). In het model die versimpeld weergegeven is, maar gelijkend is aan figuur 3.2, berekent het voor elke neuron in het model een activatie, die het doorstuurt naar een volgende laag, tot het in de laatste laag terechtkomt, de outputlaag. Deze outputlaag transformeert zijn output aan de hand van een softmax activatiefunctie (3.6) zodat de waarden tussen 0 en 1 liggen en de som van de waarden gelijk is aan 0 (stap b). In de eerste iterate, wanneer ons netwerk nog geen gewichten update ondergaan heeft, heeft het netwerk een output van $\begin{bmatrix}0.7\\0.2\\0.1\end{bmatrix}$ terwijl de reëele output $\begin{bmatrix}0\\1\\0\end{bmatrix}$ is (stap c). Ons netwerk beweert classificeert onze hond op dit moment met een probabiliteit van 70% als een kat, 20% als een hond en 10% als een vis. Met behulp van de 'categorical crossentropy' verlies functie (3.7), bekomen we een verlies van 1,609438 (stap d). Aan de hand van dit verlies en het 'gradient descent' algoritme, bepalen we nu in welke mate de gewichten die tot dit verlies geleidt hebben moeten aangepast worden (stap e). Het verlies is tamelijk groot, dus de aanpassing in de gewichten zal dit ook zijn. We doen dit voor elke laag in het netwerk, zodat alle gewichten van alle lagen aangepast worden. Met voldoende iteraties en voldoende foto's van honden komen we uiteindelijk tot een punt waar we een ongeziene hond aan het netwerk tonen (stap f) en deze met een hoge probabiliteit, namelijk 85%, kunnen classificeren als een hond. Merk ook op dat voor deze uitkomst (stap g), het verlies veel lager ligt, namelijk 0,17185032 [1].

3.4.2 Convolutionele neurale netwerken

Standaard neurale netwerken zoals we die zonet gezien hebben, hebben 1 groot nadeel. Ze zijn niet verschuiving invariant. Een verschuiving invariant systeem wordt gedefinieerd als een systeem waarbij een verschuiving in de onafhankelijke variabele van het inputsignaal, een overeenkomstige verschuiving in het uitgangssignaal veroorzaakt (Bull, 2014).

[1] Deze verliesscores werden berekend met het TensorFlow framework voor deep learning, de uitwerking vind je terug in bijlage B figuur B.1



Figuur 3.5: Een standaard neuraal netwerk die cijfers classificeert

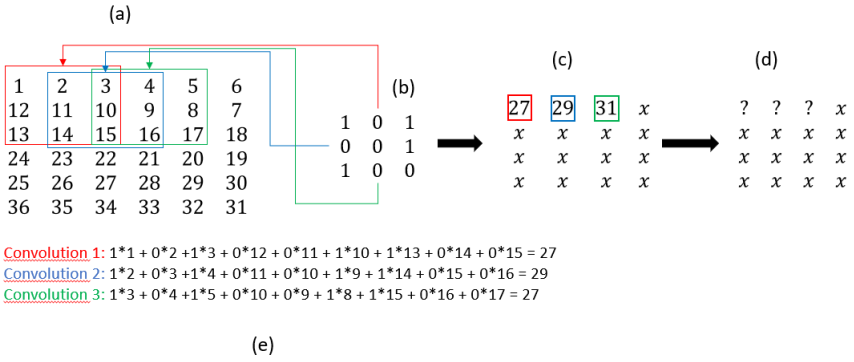
Een systeem kan zowel tijdsinvariant zijn als plaatsinvariant. In de context van computer vision met foto's is het van belang dat ons netwerk plaatsinvariant is, en de voorgaand besproken neurale netwerken zijn dat niet. We kunnen dit eenvoudig uitleggen aan de hand van figuur 3.5. Wanneer een model niet plaatsinvariant is, dan leert het de specifieke locatie van de pixels. Wanneer een foto dan varieert, in grootte, positie, achtergrond, ... is het onmogelijk voor een plaatsinvariant netwerk om een juiste voorspelling te maken. Een model getraind op figuur 3.5 (a), zal (b), (c) en (d) niet correct kunnen interpreteren door hun variantie op (a). Net zoals de temporale correlatie aangehaald voor video's in sectie 3.2, hebben foto's ruimtelijke correlatie. Pixels die dicht bij elkaar liggen, zijn sterker gecorreleerd met elkaar dan pixels die ver van elkaar liggen.

Dit is waar convolutionele neurale netwerken of 'space invariant' artificiële neurale netwerken (SIANN) ter sprake komen. Deze naam is gebaseerd op het feit dat dit soort neurale netwerken gewichten tussen neuronen kan delen en 'space invariant' is (Zhang e.a., 1990). De eerste vermelding van convolutionele neural netwerken (CNNs vanaf hier) werd gedaan door Fukushima en Miyake (1982), onder de naam van 'Neocognitron'. Het concept is gebaseerd op werk van Hubel en Wiesel (1962), die onderzoek deden naar de architectuur van een visuele cortex bij dieren. Zij vonden dat er 2 visuele cel typen waren in het brein. Simpele cellen die randen detecteren in een deel van het receptief veld en complexere cellen die een groter deel van het receptief veld zagen en deze informatie (randen) uit de simpelere cellen combineerden ongeacht hun plaats. Tenslotte werd het 'Neocognitron' model gelinkt, en hierdoor gepopulariseerd, met de backpropagation stap, die we in sectie 3.4.1 bespraken, door LeCun e.a. (1989).

We zullen een kort overzicht geven hoe deze CNNs te werk gaan op basis van een cursus van Stanford omtrent dit onderwerp, de volledige uitleg is beschikbaar in de bibliografie (Li, 2020). Om een convolutioneel netwerk te maken zijn er 3 verschillende lagen van belang.

1. Convolutionele laag
2. Pooling laag
3. Volledig geconnecteerde laag (zoals reeds gezien in figuur 3.2)

Convolutionele laag
1 convolutionele laag bestaat uit een set van filters. Deze filters bevatten de gewichten zoals eerder besproken. De filter is een matrix met een bepaalde hoogte, breedte en diepte.



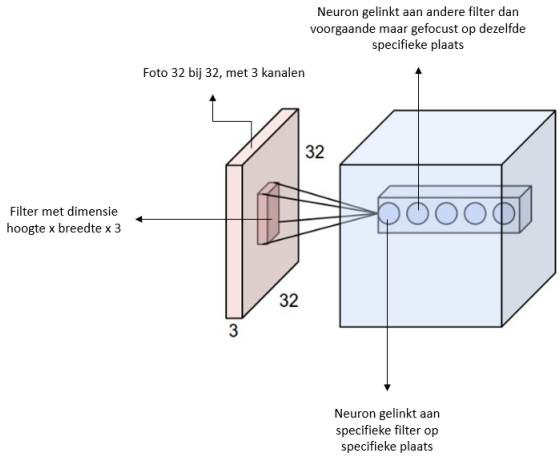
Figuur 3.6: Uitwerking van 3 convolutions op een foto bestaande uit 1 laag met een hoogte 6 en breedte 6, filter heeft een hoogte 3 bij 3 (en diepte 1 door slechts 1 laag in foto)

De breedte en hoogte zijn vastgesteld in het model, de diepte is afhankelijk van het input volume. Zoals we in figuur 3.1 besproken hebben kan een foto bestaan uit een aantal lagen, die we kanalen noemen. Dit aantal kanalen bepaalt de diepte van de filter. Wanneer we dus als input een foto hebben met 3 lagen, en de filter heeft een hoogte en diepte van 5, dan hebben we een filter met dimensie 5x5x3.

Wanneer een voorbeeld door het netwerk gestuurd wordt zal elke filter een operatie, die we convolven noemen, doen op de input. Op die manier creëert elke filter een nieuwe laag 2-dimensionele laag, die we in de diepte stapelen om nieuwe input te creëeren. Terugkerend naar ons voorbeeld met 3 lagen als input, wanneer onze eerste convolutionele laag 12 filters heeft, creëert dit 12 lagen. De filters in de volgende laag zullen dus een dimensie hebben van 5x5x12. De filter zal aan de hand van de gewichten die het bevat, leren om zaken in een foto te detecteren, dit kan een rand zijn, maar dieper in het netwerk kan dit een combinatie van randen zijn die bijvoorbeeld een hand moeten voorstellen. Een visueel voorbeeld van hoe convolven werkt is uitgewerkt in figuur 3.6. (a) is de input, (b) is de filter (waarvan de getallen de gewichten zijn die bij elke iteratie veranderen), (c) is de output na de convolution, (d) is de output nadat het door de neuron gegaan is en (e) is de uitwerking van de convolution.

Het concept van de neuron blijft mathematisch zoals ze is, maar ondergaat wel een structurele verandering. In het voorgaande neurale netwerk (figuur 3.2 en 3.3), zag elke neuron elke pixel. In geval van grote foto's zou dit leiden tot een gigantische hoeveelheid gewichten die we moeten trainen ten opzichte van de convolutionele neurale netwerken(zie bijlage B.2). Het verschil in neuronen bij FFNN en CNNs, en dus de reden waarom een CNN minder parameters moet leren, is dat elke neuron in een CNN slechts een deel van de pixels waarneemt. Het aantal neuronen blijft dus gelijkaardig, maar het aantal gewichten voor 1 neuron zijn in het geval van een CNN veel kleiner, en zijn gelijk aan de filter size. Wat 1 neuron kan zien noemen we zijn receptief veld.

Figuur 3.7 is hier een illustratie van. We hebben als input een foto met 3 kanalen, elk met een hoogte van 32 en breedte van 32. De figuur toont 5 neuronen in de diepte dus we ondergaan de assumptie dat er 5 filters zijn. Elke filter heeft een hoogte, breedte



Figuur 3.7: Grafische voorstelling van neuronen en filter in een convolutioneel netwerk (Li, 2020)

(parameters die zelf gekozen worden) en een diepte (gekozen door de input die het krijgt). Zoals eerder besproken convolven de filters over de input foto, de gewichten van de filter worden samen met de input van de foto doorgestuurd naar 1 specifieke neuron. Deze neuron is filter- en plaats afhankelijk en doet dan dezelfde computaties zoals we gezien hebben in figuur 3.3. Doordat elke filter overeenkomt met exact 1 geproduceerde laag, wordt er tussen de neuronen in die laag aan parameter delen gedaan. Dit reduceert het aantal te leren gewichten drastisch.

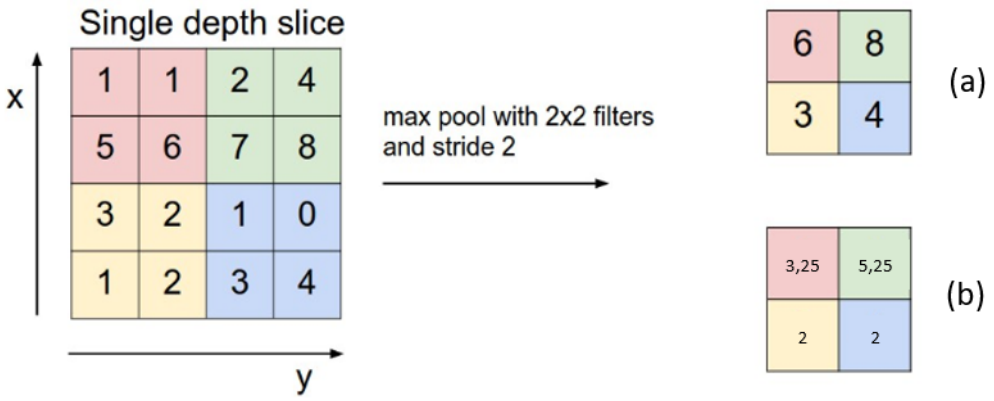
Pooling laag

Een vaak gebruikte truc om het aantal gewichten dat ons convolutioneel netwerk moet leren, nog meer te reduceren is het introduceren van pooling lagen. Deze lagen reduceren de dimensie van de input door een gemiddelde of maximale waarde te berekenen van een bepaalde filter grootte. Figuur 3.8 toont hoe deze laag werkt. De input wordt door deze laag gestuurd en afhankelijk van de grootte van de pooling filter en type maximum of gemiddeld, selecteert het respectievelijk de maximum en gemiddelde waarde in een regio van de input.

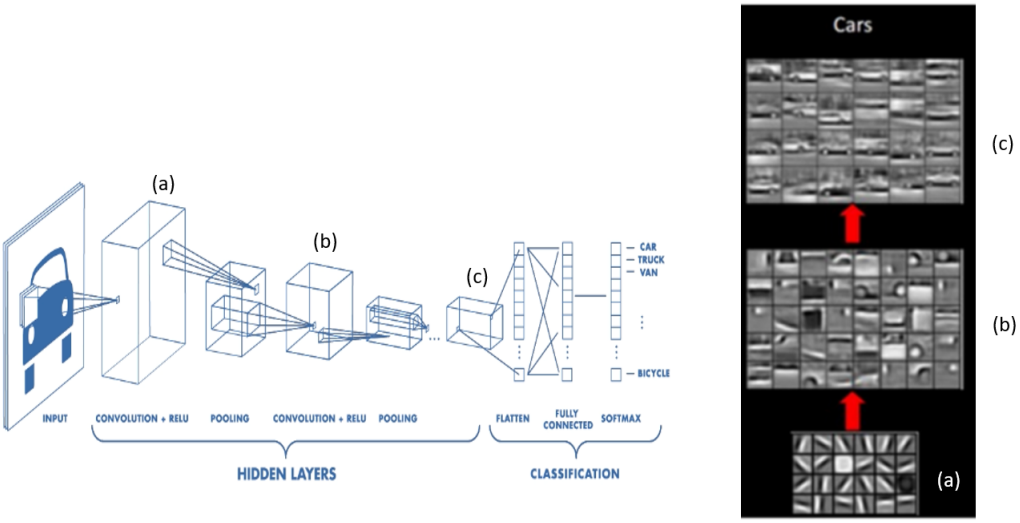
Samenbrengen

We kunnen dit nu alles samenbrengen in een geheel. De convolutionele lagen, de pooling lagen en de volledig geconnecteerde lagen zoals. Figuur 3.9 vat dit visueel samen. In een eerste laag sturen we de foto van een auto door het netwerk, wat het netwerk hier detecteert (nadat het uiteraard getraind is) zijn simpele zaken zoals strepen, rondingen, ... Verder in het netwerk worden deze strepen en randen gecombineerd tot abstractere zaken zoals bijvoorbeeld een autolicht, een wiel, ... Nog verder worden deze gecombineerd tot, in dit geval, een auto. Deze activaties, die in feite de geleerde features zijn zoals een autolicht, worden uiteindelijk geconverteerd naar een 1 dimensionale vector en doorgestuurd door een FFNNN zoals in figuur 3.2 om de uiteindelijke classificatie te doen.

Intuïtief kunnen we dus samenvatten dat de filters 'feature detectors' zijn, ze ontdekken zaken op een foto (waar deze zich ook afspelen) die belangrijk zijn. De combinatie van zo een features worden uiteindelijk gereduceerd tot een classificatie. Wanneer ons netwerk



Figuur 3.8: Pooling laag met behulp van een filter 2 x 2, en strides (opschuiven) per 2. (a) is maximum pooling, (b) is gemiddeld poolen (Li, 2020)



Figuur 3.9: Voorbeeld van een convolutioneel neuraal netwerk (links) en de tussentijdse output (rechts). (Chatterjee, 2019; Draelos, 2020)

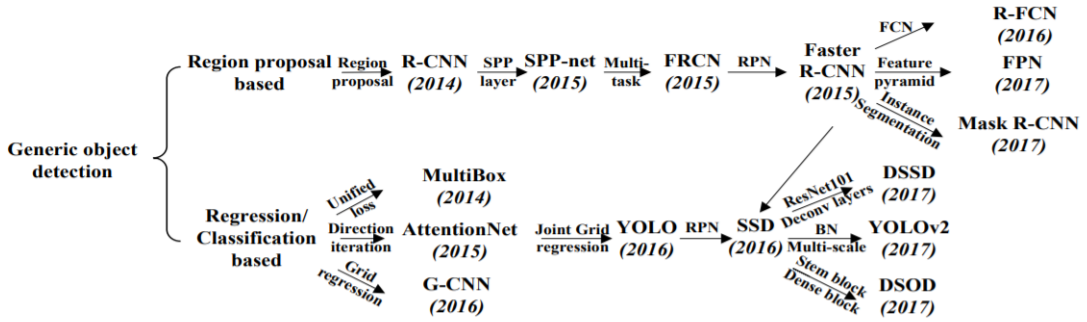
bijvoorbeeld op een foto, een autolicht, een wiel, een autodeur, ... zien, combineert het deze informatie om te bepalen dat de foto een auto bevat.

4. Specifieke architecturen

In dit hoofdstuk gaan we dieper in op de verschillende belangrijke use cases binnen computer vision voor ons probleem zoals besproken in sectie 2.3. We gebruiken hiervoor de eerder besproken concepten uit de wereld van deep learning (CNN, verliesfunctie, neurons, ...) in sectie 3.4. We staan eerst stil bij object detectie, het traceren en classificeren van een object op een foto, vervolgens staan we stil bij 2 methodes, object en instantie segmentatie, om een object pixel per pixel te onderscheiden van de rest op een foto. Voorbeelden van elk kunnen teruggevonden worden op figuur 2.2. Ter herinnering, we bespreken in dit hoofdstuk enkel deep learning gebaseerde methoden, daar deze de standaard zijn geworden de afgelopen jaren.

4.1 Object detectie

Zoals eerder besproken is object detectie een verdere stap van object classificatie. Naast het classificeren van een object, lokaliseert het het object ook en trekt het een omkadering rond het object die zijn label en betrouwbaarheid aanduiden (Zhao e.a., 2019). De huidige state-of-the art methoden (SOTA vanaf hier), kunnen grofweg opgedeeld worden in one-stage methoden (regressie/classificatie gebaseerde methoden) en two-stage methoden (region proposal gebaseerde methoden). Een overzicht van de evolutie en opsplitsing van deze methoden is te zien in figuur 4.1.



Figuur 4.1: Opdeling object detectie methoden in 2-stage methoden (boven) en 1-stage methoden (onder) (Zhao e.a., 2019)

4.1.1 Two stage methode

Two stage methoden of region proposal gebaseerde methoden, lossen het probleem van object detectie op door het op te splitsen in 2 taken. Ten eerste delen ze de foto op in region proposals op basis van een zoekalgoritme zoals 'selective search' (Girshick e.a., 2014). Ten tweede zullen ze deze region proposals door een convolutioneel netwerk sturen zoals we eerder gezien hebben om het te classificeren.

R-CNN

Het eerste model, met behulp van DL technieken, die deze techniek verder exploiteerde was het R-CNN of 'Regions with CNN'. Het verbeterde de toenmalige SOTA technieken met 30% (Girshick e.a., 2014). Een eerste iteratie van dit model bestond uit de volgende zaken. Ten eerste een 'selective search' algoritme die regio's met objecten voorstelt. Het stelt zo een bepaald aantal objecten per foto voor. Een tweede stap bestond er uit om deze objecten te classificeren met behulp van een convolutioneel netwerk. Op deze manier kregen we dus een foto met een aantal bounding boxes, en voor elke bounding box een voorspelling en betrouwbaarheid. Uiteindelijk worden de bounding boxes met een bepaalde betrouwbaarheid (bijvoorbeeld > 70%) weergegeven op de foto.

Het selective search algoritme is een ongesuperviseerd (sectie 2.2.2) algoritme ontdekt door de Nederlanders Uijlings e.a. (2013) en is een verbetering op het exhaustieve, voorgaand gangbare 'sliding window' algoritme. Het probleem met dit algoritme was dat ze de gehele foto in achting nam en deze opdeelde in een groot aantal regio's. Dit groot aantal regio's, waarvan een groot aantal niets bevatten, werd dan doorgestuurd naar het CNN, wat computationeel niet efficiënt was. Om een goed algoritme te zijn voor object detectie, moet ons zoekalgoritme een hoge 'recall' hebben. Recall wordt gedefinieerd als het aantal relevante objecten die gedetecteerd worden tegenover het aantal relevante objecten op de foto (Ting, 2010). Wanneer er bijvoorbeeld 1 relevant object in de foto aanwezig is, en we vinden met behulp van het zoekalgoritme dit object, hebben we een recall van 100%. Wanneer er 2 relevante objecten zijn en we vinden er slechts 1, dan hebben we een recall van 50%. Wanneer we dus bijvoorbeeld 2 relevante objecten hebben in een foto, er 2 vinden maar ook 50 niet relevante objecten, hebben we nog steeds een recall van 100%

(formule 4.1).

$$recall = \frac{TruePositive}{(TruePositive + FalsePositive)} \tag{4.1}$$

Het selective search algoritme kan als volgt beschreven worden. Het start met het oversegmenteren van de foto op basis van de pixelintensiteit (Felzenszwalb & Huttenlocher, 2004). Deze overgesegmenteerde foto kan niet gebruikt worden omdat een object die behoort tot 1 klasse in dit geval vaak meerdere segmenten bevat. Dit is waar het selective search algoritme start. Het neemt de overgesegmenteerde foto als input en zal aangrenzende segmenten samenvoegen op basis van hun gelijkenis. Het algoritme in pseudo-code is terug te vinden in bijlage B.3. We combineren kleinere segmenten dus in grotere segmenten op een bottom-up manier.

De gelijkenis tussen 2 segmenten wordt bepaald door 4 metingen (4.6). Kleurengelijkenis (4.2) berekent voor elk segment een kleurenhistogram $C_i = c_i^1, ..., c_i^n$, en zal voor 2 regio's de som nemen van elke minimumwaarde voor elke positie in het kleurenhistogram. Op deze manier zullen segmenten met gelijkaardige kleuren hier een hogere waarde halen dan segmenten met verschillende kleuren. Textuurgelijkenis (4.2) berekent voor elk segment een textuurhistogram $T_i = t_i^1, ..., t_i^n$, en zal opnieuw voor 2 regio's de som nemen van elke minimumwaarde voor elke positie in de textuurhistogram. Opnieuw zullen 2 segmenten die gelijkend in textuur zijn hier een hogere waarde behalen dan segmenten met verschillende kleuren. Om te voorkomen dat 1 regio alle kleinere regio's opslokt, worden deze kleinere regio's aangespoord om snel samen te komen door middel van de grootte similariteit (4.4). Vervolgens willen we regio's die goed in elkaar passen samenvoegen en voorkomen dat regio's die elkaar niet aanraken, niet samengevoegd worden. Hiervoor dient de compatibiliteit in vorm (4.5). $Size(BB_{ij})$ is een bounding box rond 2 regio's i en j, wanneer deze groot is en de grootte van de regio's klein, dan is er een grootte kans dat ze elkaar niet raken. De waarde voor 4.5 zal in zo een geval ook klein zijn waardoor dergelijke regio's niet vlug zullen samengesmolten worden. Om tenslotte de gelijkenis tussen 2 regio's te berekenen, berekenen we een lineaire combinatie van de voorgaande gelijkenissen (4.6). Een grotere waarde betekent dat 2 regio's gelijkend zijn aan elkaar (tot hetzelfde object behoren), deze worden ook als eerste samengevoegd.

$$s_{kleur}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k) \tag{4.2}$$

$$s_{textuur}(r_i, r_j) = \sum_{k=1}^{n} \min(t_i^k, t_j^k) \tag{4.3}$$

$$s_{grootte}(r_i, r_j) = 1 - \frac{size(r_i) + size(r_j)}{size(im)} \tag{4.4}$$

$$s_{vorm}(r_i, r_j) = 1 - \frac{size(BB_{ij}) - size(r_i) - size(r_j)}{size(im)} \tag{4.5}$$

$$s(r_i, r_j) = a_1 s_{kleur}(r_i, r_j) + a_2 s_{textuur}(r_i, r_j) + a_3 s_{grootte}(r_i, r_j) + a_4 s_{vorm}(r_i, r_j) \tag{4.6}$$

Fast R-CNN

Faster R-CNN

4.1.2 One stage methode

Yolo

4.1.3 Evaluatie

4.1.4 SOTA object detectie

4.2 Object segmentatie

4.3 Instantie segmentatie

5. Praktische uitwerking

5.1 Verschillende programmeermogelijkheden

Er zijn een honderden, misschien wel duizenden, verschillende programmeertalen. Hoe beslissen we dan hoe we de oplossing zullen implementeren? Om deze vraag te beantwoorden moeten 2 andere subvragen beantwoord worden.

1. Welke programmeertalen zijn populairst?
2. Welke programmeertalen zijn geschikt voor deep learning?

De eerste vraag proberen we te beantwoorden aan de hand van 2 indexen. De PYPL (PopularitY of Programming Language) en de TIOBE index. Op basis van deze 2 indexen besluiten we dat Python dezer dagen de populairste programmeertaal is.

De volgende vraag is of deze taal geschikt is om aan deep learning te doen? Het antwoord is ja. Python heeft een aantal 'frameworks' waarmee aan deep learning kan gedaan worden, waarvan de 2 populairste PyTorch (ontwikkeld door Facebook) en TensorFlow (ontwikkeld door Google) zijn. https://www.kdnuggets.com/2019/05/which-deep-learning-framework-growing-fastest.html

We zullen ons uiteindelijke model ontwikkelen voor zowel PyTorch als TensorFlow en bespreken wat de voordelen- en nadelen zijn van elk framework in hoofdstuk 7.

5.2 Google Colab

Zoals we in hoofdstuk 3 uitvoerig besproken hebben, zijn neurale netwerken niets anders dan matrix vermenigvuldigingen. We nemen de gewichten, de input, doen de matrix vermenigvuldiging en sturen dit door een activatiefunctie op naar de volgende laag. In deze volgende laag herhalen we deze stap tot aan de output. Wanneer we de backpropagation fase ingaan, doen we ditzelfde nogmaals in de andere richting.

Een doodnormale computer, met een gewone central processing unit (CPU), is hier niet voor gemaakt. Het zou dus voor grote netwerken (met miljoenen en zelf miljarden (ref) parameters) een onmogelijke taak worden. Dit is waar grafische processing units (GPU) in beeld komen, deze wijzen veel meer transistors (de manier waarop de 0'en en 1'en weergegeven kunnen worden met behulp van elektrische pulsen) toe aan het uitvoeren van rekenkundige operaties (zoals matrix vermenigvuldigingen). Dit laat ons toe om in een parallelle en op een veel snellere manier deze grote netwerken te trainen.

Het probleem is dat niet elke computer zo een GPU bezit, en dat het gebruiken van cloud GPU kostelijk kan uitdraaien voor het uitvoeren van een proof-of-concept zoals in deze bachelorproef. Gelukkig voorziet Google ons met een alternatief, Google Colab(oratory). Het is een cloud programmeer omgeving, die ontstaan is om mensen op een eenvoudige manier te laten samenwerken aan programmeer projecten. Sinds kort is er ook de optie om een GPU, gratis, vast te maken aan de omgeving. We zullen deze omgeving dus ook gebruiken om ons model te creëeren en uit te laden. De inferentie (voorspellingen) van het model zullen voor deze proof-of-concept lokaal gebeuren en dus via een CPU verlopen, wat uiteraard iets trager zal verlopen.

5.3 OpenCV

5.4 Transfer learning

5.5 Applicatie

5.5.1 Backend API

5.5.2 Frontend

6. Resultaten

7. Conclusie

Startpunt (Review of Deep Learning Methods in Robotic Grasp ... - MDPI) https://paperswithcode.com/task/sr
object-detection

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

Hier introduceer je werk. Je hoeft hier nog niet te technisch te gaan.

Je beschrijft zeker:

- de probleemstelling en context
- de motivatie en relevantie voor het onderzoek
- de doelstelling en onderzoeksvraag/-vragen

A.2 State-of-the-art

Hier beschrijf je de *state-of-the-art* rondom je gekozen onderzoeksdomein. Dit kan bijvoorbeeld een literatuurstudie zijn. Je mag de titel van deze sectie ook aanpassen (literatuurstudie, stand van zaken, enz.). Zijn er al gelijkaardige onderzoeken gevoerd? Wat concluderen ze? Wat is het verschil met jouw onderzoek? Wat is de relevantie met jouw onderzoek?

Verwijs bij elke introductie van een term of bewering over het domein naar de vakliteratuur, bijvoorbeeld (Doll & Hill, 1954)! Denk zeker goed na welke werken je refereert en

waarom.

Je mag gerust gebruik maken van subsecties in dit onderdeel.

A.3 Methodologie

Hier beschrijf je hoe je van plan bent het onderzoek te voeren. Welke onderzoekstechniek ga je toepassen om elk van je onderzoeksvragen te beantwoorden? Gebruik je hiervoor experimenten, vragenlijsten, simulaties? Je beschrijft ook al welke tools je denkt hiervoor te gebruiken of te ontwikkelen.

A.4 Verwachte resultaten

Hier beschrijf je welke resultaten je verwacht. Als je metingen en simulaties uitvoert, kan je hier al mock-ups maken van de grafieken samen met de verwachte conclusies. Benoem zeker al je assen en de stukken van de grafiek die je gaat gebruiken. Dit zorgt ervoor dat je concreet weet hoe je je data gaat moeten structureren.

A.5 Verwachte conclusies

Hier beschrijf je wat je verwacht uit je onderzoek, met de motivatie waarom. Het is **niet** erg indien uit je onderzoek andere resultaten en conclusies vloeien dan dat je hier beschrijft: het is dan juist interessant om te onderzoeken waarom jouw hypothesen niet overeenkomen met de resultaten.

B. Screenshot code

```
import tensorflow as tf
```

```
cat = tf.keras.losses.CategoricalCrossentropy()
```

```
CCELOS_img1 = cat(y_true=[0,1,0], y_pred=[0.7,0.2,0.1]).numpy()
CCELOS_img2 = cat(y_true=[0,1,0], y_pred=[0.15,0.8,0]).numpy()

print('De loss voor de eerste foto (VOOR de gewichten update) is: ' + str(CCELOS_img1))
print('De loss voor de tweede foto (NA de gewichten update) is: ' + str(CCELOS_img2))
print('------------')
print('De loss is dus al een pak kleiner voor de 2e foto')
```

```
De loss voor de eerste foto (VOOR de gewichten update) is: 1.609438
De loss voor de tweede foto (NA de gewichten update) is: 0.17185032
------------
De loss is dus al een pak kleiner voor de 2e foto
```

Figuur B.1: Code voor het berekenen van de ’categorical crossentropy’ verlies

(a) 6 mio parameters

```
width = 256
height = 256
depth = 3 # Color

inp = Input(shape=(width*height*depth,))
X = Dense(32)(inp)
X = Dense(64)(X)
X = Dense(128)(X)
X = Dense(256, activation = 'relu')(X)

model = Model(inputs = inp, outputs = X)

model.compile(loss='categorical_crossentropy', optimizer = 'Adam')
model.summary()
```

```
Model: "model_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_15 (InputLayer)        [(None, 196608)]          0
_________________________________________________________________
dense_67 (Dense)             (None, 32)                6291488
_________________________________________________________________
dense_68 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_69 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_70 (Dense)             (None, 256)               33024
=================================================================
Total params: 6,334,944
Trainable params: 6,334,944
Non-trainable params: 0
_________________________________________________________________
```

(b) 0,4 mio parameters

```
width = 256
height = 256
depth = 3 # Color

inp = Input(shape=(width,height,depth,))
X = Conv2D(32, 3)(inp)
X = Conv2D(64, 3)(X)
X = Conv2D(128, 3)(X)
X = Conv2D(256, 3)(X)
X = Dense(256, activation = 'relu')(X)

model = Model(inputs = inp, outputs = X)

model.compile(loss='categorical_crossentropy', optimizer = 'Adam')
model.summary()
```

```
Model: "model_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_18 (InputLayer)        [(None, 256, 256, 3)]     0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 254, 254, 32)      896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 252, 252, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 250, 250, 128)     73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 248, 248, 256)     295168
_________________________________________________________________
dense_71 (Dense)             (None, 248, 248, 256)     65792
=================================================================
Total params: 454,208
Trainable params: 454,208
Non-trainable params: 0
_________________________________________________________________
```

Figuur B.2: Code voor het berekenen van aantal parameters bij een kleurenfoto van 256 bij 256 pixels (a = FFNN) en (b = CNN)

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
  Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

Extract object location boxes L from all regions in R

Figuur B.3: Pseudo code voor selective zoekalgoritme (Uijlings e.a., 2013)

Bibliografie

(2020). http://www.cis.rit.edu/people/faculty/pelz/courses/SIMG203/res.pdf

Awe, O., Mengistu, R. & Sreedhar, V. (2017). Smart trash net: Waste localization and classification. *arXiv preprint*.

Bernstein, A. & Burnaev, E. (2018). Reinforcement learning in computer vision, In *Tenth International Conference on Machine Vision (ICMV 2017)*. International Society for Optics en Photonics.

Bonanomi, G., Incerti, G., Cesarano, G., Gaglione, S. A. & Lanzotti, V. (2015). Cigarette butt decomposition and associated chemical changes assessed by 13 C CPMAS NMR. *PLoS One*, *10*(1), e0117393.

Bull, D. R. (2014). *Communicating pictures: A course in Image and Video Coding*. Academic Press.

Chatterjee. (2019). Basics of the Classic CNN. How a classic CNN [(Accessed on 08/04/2020)].

Dasiopoulou, S., Mezaris, V., Kompatsiaris, I., Papastathis, V.-K. & Strintzis, M. G. (2005). Knowledge-assisted semantic video object detection. *IEEE Transactions on Circuits and Systems for Video Technology*, *15*(10), 1210–1224.

Doll, R. & Hill, A. B. (1954). The mortality of doctors in relation to their smoking habits: a preliminary report. *British Medical Journal*, *328*(7455), 1529–1533.

Draelos. (2020). Convolutional Neural Networks (CNNs) in 5 minutes [(Accessed on 08/04/2020)].

Felzenszwalb, P. F. & Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International journal of computer vision*, *59*(2), 167–181.

Fukushima, K. & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, In *Competition and cooperation in neural nets*. Springer.

Geman, S., Bienenstock, E. & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural computation*, *4*(1), 1–58.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation, In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Gonzalez, R. C. & Woods, R. E. (2018). *Digital image processing*. Pearson.

Green, D. S., Boots, B., Carvalho, J. D. S. & Starkey, T. (2019). Cigarette butts have adverse effects on initial growth of perennial ryegrass (gramineae: Lolium perenne L.) and white clover (leguminosae: Trifolium repens L.) *Ecotoxicology and environmental safety*, *182*, 109418.

Güven, O., Gökdağ, K., Jovanović, B. & Kıdeyş, A. E. (2017). Microplastic litter composition of the Turkish territorial waters of the Mediterranean Sea, and its occurrence in the gastrointestinal tract of fish. *Environmental Pollution*, *223*, 286–294.

Hinton, G. E., Sejnowski, T. J., Poggio, T. A. En andere. (1999). *Unsupervised learning: foundations of neural computation*. MIT press.

Huang, T. (1996). Computer vision: Evolution and promise.

Hubel, D. H. & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, *160*(1), 106.

An introduction to the Principles of Video 101 [(Accessed on 08/02/2020)]. (g.d.).

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

Li, X., Krishna. (2020). CS231n Convolutional Neural Networks for Visual Recognition [(Accessed on 08/02/2020)].

Lusher, A. L., Mchugh, M. & Thompson, R. C. (2013). Occurrence of microplastics in the gastrointestinal tract of pelagic and demersal fish from the English Channel. *Marine pollution bulletin*, *67*(1-2), 94–99.

Mahmood, A. & Khan, S. (2007). Exploiting inter-frame correlation for fast video to reference image alignment, In *Asian Conference on Computer Vision*. Springer.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Mohri, M., Rostamizadeh, A. & Talwalkar, A. (2012). Foundations of Machine Learning. Adaptive computation and machine learning. *MIT Press*, *31*, 32.

Morris, T. (2004). *Computer vision and image processing*. Palgrave Macmillan.

Novotny, T. E., Hardin, S. N., Hovda, L. R., Novotny, D. J., McLean, M. K. & Khan, S. (2011). Tobacco and cigarette butt consumption in humans and animals. *Tobacco Control*, *20*(Suppl 1), i17–i20.

Novotny, T. E., Lum, K., Smith, E., Wang, V. & Barnes, R. (2009). Cigarettes butts and the case for an environmental policy on hazardous cigarette waste. *International journal of environmental research and public health*, *6*(5), 1691–1705.

Novotny, T. E. & Zhao, F. (1999). Consumption and production waste: another externality of tobacco use. *Tobacco control*, *8*(1), 75–80.

on Environmental Pollution (Great Britain), R. C. (2003). *Tenth Report: Tackling Pollution-experience and Prospects*. HM Stationery Office.

Panagos, P., Van Liedekerke, M., Yigini, Y. & Montanarella, L. (2013). Contaminated sites in Europe: review of the current situation based on data collected through a European network. *Journal of Environmental and Public Health*, *2013*.

Porikli, F. (2006). Achieving real-time object detection and tracking under extreme conditions. *Journal of Real-Time Image Processing*, *1*(1), 33–40.

Read, P. & Meyer, M.-P. (2000). *Restoration of motion picture film*. Elsevier.

Rosebrock, A. (2018). *Simple objecct tracking with OpenCV*. Verkregen 29 juli 2020, van https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

Russell, S. J. (2016). *Artificial intelligence: a modern approach*. Pearson.

Russell, S. J. & Norvig, P. (2010). Artificial Intelligence-A Modern Approach, Third International Edition. Pearson Education London.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, *3*(3), 210–229.

Smith, E. A. & Novotny, T. E. (2011). Whose butt is it? tobacco industry research about smokers and cigarette butt waste. *Tobacco control*, *20*(Suppl 1), i2–i9.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Thompson, R. C., Swan, S. H., Moore, C. J. & Vom Saal, F. S. (2009). Our plastic age. The Royal Society Publishing.

Ting, K. M. (2010). Precision and Recall. *Encyclopedia of machine learning*, *781*.

Uijlings, J. R., Van De Sande, K. E., Gevers, T. & Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, *104*(2), 154–171.

USoft. (2020). *Introductie tot Artificial Intelligence, Machine Learning en Deep Learning*. Verkregen 29 juli 2020, van https://www.usoft.com/nl/blog-nl/introductie-tot-kunstmatige-intelligentie-machine-learning-en-deep-learning

Wu, X., Sahoo, D. & Hoi, S. C. (2020). Recent advances in deep learning for object detection. *Neurocomputing*.

Wu, Y., Lim, J. & Yang, M.-H. (2013). Online object tracking: A benchmark, In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Zaeni, I. A. E., Sendari, S., Lestari, D., Mahandi, Y. D., Jiono, M. & Syaifuddin, M. (2018). An implementation of multi-object tracking using omnidirectional camera for trash picking robot, In *2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)*. IEEE.

Zhang, W., Itoh, K., Tanida, J. & Ichioka, Y. (1990). Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, *29*(32), 4790–4797.

Zhao, Z.-Q., Zheng, P., Xu, S.-t. & Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), 3212–3232.