Professionele Bachelor Toegepaste Informatica

Virtual Oswald

Niels Vaes

Promotoren:

Tim Dupont — Hogeschool PXL Hasselt
Georges Petrofski — Craftworkz

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Virtual Oswald

Niels Vaes

Promotoren:

Tom Dupont — Hogeschool PXL Hasselt
Georges Petrofski — Craftworkz

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Als eerste wil ik mijn stage bedrijf Craftworkz bedanken voor de mogelijkheid om mijn stage bij hun te doen. Michiel Vandendriessche, Sam Hendrickx en Hannah Patronoudis gaven mij en mijn stagepartner de nodige begeleiding en kritiek om de opdracht tot een goed einde te brengen. Ik bedank ook alle collega's die constant klaar stonden om mijn vragen te beantwoorden, raad te geven en enthousiasme te tonen voor ons resultaat.

Natuurlijk stond ik niet alleen voor de uitwerking van de stageopdracht. Vanaf dag één was er een uitstekende samenwerking tussen mij en mijn stagepartner Jari Crollet. Ik ben hem dankbaar voor onze aangename samenwerking en ben trots op de applicatie die wij samen ontwikkeld hebben.

Speciale dank gaat uit naar mijn hogeschoolpromotor Tim Dupont voor de uitgebreide feedback die hij mij regelmatig gaf. Hij was altijd bereid mijn vele vragen te beantwoorden en leerde me kritisch te zijn over mijn eigen eindwerk.

Als laatste wil ik iedereen die mij op een of andere manier gesteund heeft bij het tot stand brengen van dit eindwerk bedanken. Ik ben zeer blij met alle begeleiding en steun die ik gekregen heb.

Abstract

Oswald is een *chatbot* platform ontwikkeld door Craftworkz. Demo's zijn een onderdeel van het promoten van het platform. Momenteel gebeurt dit nog door live demo's op de website zonder een *hands-on experience* voor de klanten. Voor een eerste kennismaking met de *chatbot* is het echter interessanter om een interactieve demo te geven. Zo zou er een beter beeld gegeven kunnen worden van de mogelijke implementaties van de *chatbot*. Hiervoor werd er een *Augmented Reality* (AR) omgeving ontwikkelt die verkend kan worden met de hulp van een virtuele gids. De gebruiker kan via spraak informatie aan de *chatbot* vragen die dan op zijn beurt eveneens via spraak zal antwoorden. Zo wordt er een echte conversatie nagebootst. Om dit te realiseren wordt er gebruikgemaakt van *speech to text* en *text to speech*. Het ARKit-platform maakt het mogelijk om de AR-omgeving op te zetten. Dit alles werd samengebracht in een iOS-applicatie die de mogelijkheden van het Oswald platform demonstreren.

Image recognition wordt steeds vaker gebruikt en geïmplementeerd in uiteenlopende toepassingen, onder andere in mobiele applicaties. Er zijn meerdere manieren waarop deze integratie kan gebeuren. De *image recognition* kan binnen dit eindwerk op het toestel zelf uitgevoerd worden of er kan een *cloud* API gebruikt worden. Met dit onderzoek zal achterhaalt worden welke methode het efficiëntst is voor iOS-applicaties in verschillende toepassingen. Hiervoor worden een aantal aspecten van de integratiemethodes onderzocht en vervolgens vergeleken. Er zal bijvoorbeeld gekeken worden naar de snelheid van *Image recognition* na implementatie, de accuraatheid, etc. Op basis van de resultaten worden aanbevelingen per toepassing aangereikt.

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

Afkorting	Betekenis	Definitie
AI	*Artificiële intelligentie*	Artificiële intelligentie is een technologie waarmee computers, apparaten of software zelfstandig kunnen leren, problemen oplossen of beslissingen nemen.
ANN	*Artificial neural network*	*Artificial neural networks* zijn computersystemen die geïnspireerd zijn op biologische neurale netwerken.
API	*Application programming interface*	Een *application programming interface* is een verzameling definities op basis waarvan een computerprogramma kan communiceren met een ander programma of onderdeel.
AR	*Augmented reality*	*Augmented reality* is het toevoegen van virtuele 3D modellen in de werkelijke wereld.
CNN	*Convolutional neural network*	Een *convolutional neural network* is een type van artificieel neuraal netwerk dat gebruikt wordt bij *image recognition.* Een CNN is speciaal ontworpen om pixeldata te verwerken.
GPU	*Graphics processing unit*	Een *graphics processing unit* is een processor die gebruikt wordt voor alle videotaken.
SDK	*Software development kit*	Een software development kit is een verzameling hulpmiddelen die handig zijn bij het ontwikkelen van computerprogramma's voor een bepaald besturingssysteem, type hardware, desktopomgeving of voor het maken van software die een speciale techniek gebruikt.

Inleiding

In dit eindwerk zal enerzijds een gedetailleerd stageverslag gegeven worden over de ontwikkeling van een applicatie voor het stagebedrijf Craftworkz en een aanvullend onderzoek in verband met verschillende implementatiemethodes van *image recognition*.

In het eerste gedeelte wordt de stageopdracht van Craftworkz uitgewerkt. Het bedrijf was opzoek naar een nieuwe manier om hun *chatbot* platform Oswald voor te stellen aan hun klanten. Voordien gebeurde dit door informatieve demo's op hun website, maar het leek hun interessanter een demo te ontwikkelen met een meer *hands-on experience*. Het eindwerk gaat dieper in op de iOS-applicatie die ontwikkeld werd en de technieken die ervoor gebruikt werden.

Anderzijds zal in het tweede deel van het eindwerk het onderzoek rond *image recognition* besproken worden. Na een uitgebreide literatuurstudie rond *image recognition* via *convolutional neural networks* (CNN), wordt er gekeken welke implementatie methode het meest efficiënt is voor een iOS-applicatie. De vergelijking zal gemaakt worden tussen *image recognition on-device* en *image recognition* via een *application programming interface* (API).

I. Stageverslag

1 Bedrijfsvoorstelling

Figuur 1: Craftworkz logo

Craftworkz is een bedrijf met veel verschillende werkzaamheden: van het van het maken van geavanceerde *software prototyping* en *machine learning*, tot innovatieve consultancy. Het bedrijf is gelegen in Leuven en werd opgericht op 28 april 2015 als onderdeel van de Cronos groep. De Cronos groep zelf werd opgericht in 1991 en is geëvolueerd van een eenmansbedrijf naar een grotere cluster van bedrijven met meer dan 5000 medewerkers. Het is hiermee één van de grootste IT-bedrijven van België. [1]

Alhoewel craftsworks in eerste instantie al hun werkzaamheden onder éénzelfde naam aanbood, richtte zij uiteindelijk de Raccoons groep op die hun diensten in verschillende kleine ondernemingen indeelde. Elke deel-onderneming van de Raccoons groep heeft zich gespecialiseerd in een of meerdere van deze diensten om voor de klanten duidelijkheid te scheppen bij wie ze terecht kunnen. De Raccoons groep bestaat momenteel uit zes ondernemingen: Craftworkz, Oswald, Brainjar, Wheelhouse en de recenter opgerichte TheLedger en QNTM.

Figuur 2 Raccoons

Deze stage werd aangeboden binnen Craftworkz, maar een nauwe samenwerking met Oswald was noodzakelijk voor het opzetten van de opdracht en het verkrijgen van het eindresultaat. Craftworkz focust op het ontwikkelen van software prototypes voor hun klanten terwijl Oswald een chatbotplatform is voor het maken van intelligente chatbots dat ontwikkeld werd door Craftworkz. [2]

2 Stageopdracht

Steeds meer bedrijven tonen interesse in het gebruik van een *chatbot*, maar helaas blijven veel van dit type bots onpersoonlijk: een chatvenster waar een zogezegde virtuele assistent achter leeft die de gebruiker kan helpen met al zijn problemen. Wat als deze virtuele assistent nu eens tot leven kwam? Dat zou kunnen door de '*augmented chatbot*' waarbij er gebruik gemaakt wordt van een bestaande *chatbot* die in *augmented reality* geplaatst wordt. Dat kan in de vorm van een poppetje of avatar die mee rondloopt met de gebruiker of hem zelfs rondleidt. Door middel van *voice recognition* kunnen vragen omgezet worden naar tekst die te interpreteren is door de *chatbot*. Daarna wordt het antwoord van de *chatbot* uitgesproken via *voice synthesis*. Zo krijgt de gebruiker een virtuele assistent waarmee hij een conversatie kan voeren in een *augmented* wereld voor een persoonlijkere ervaring.

2.1 Probleemstelling

Oswald is een *chatbot* platform ontwikkeld door Craftworkz. Demo's zijn een onderdeel van het promoten van het platform. Momenteel gebeurt dit nog door live demo's op de website zonder een *hands-on experience* voor de klanten. Voor een eerste kennismaking met het platform is het dus interessanter om een interactieve demo te geven waarbij klanten het platform zelf kunnen testen. Zo zou er een beter beeld gegeven kunnen worden van de mogelijke implementaties van het Oswald platform.

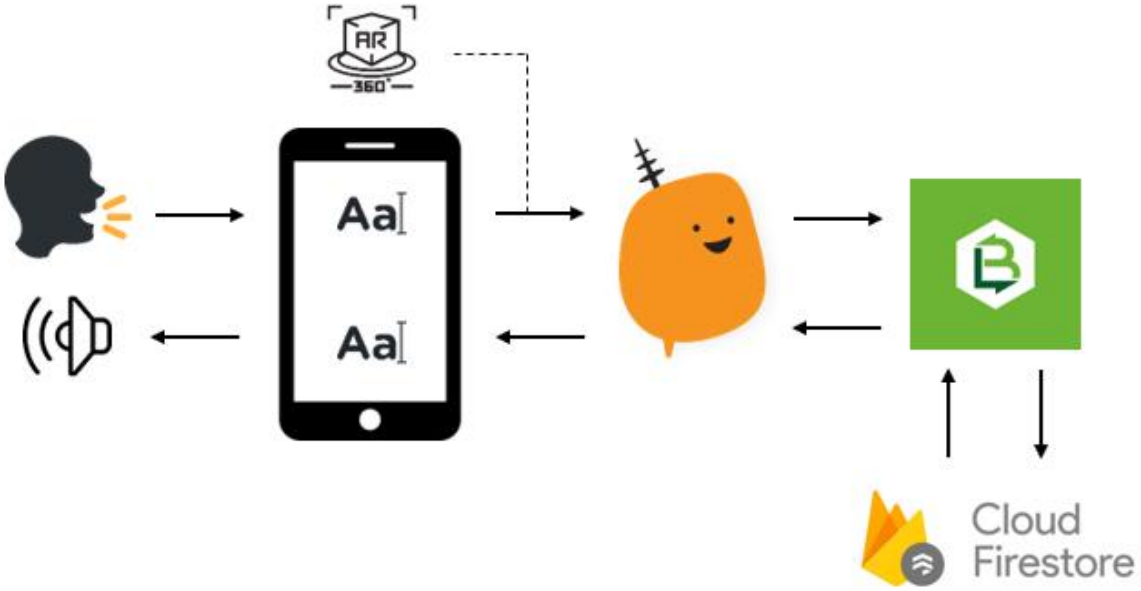
2.2 Doelstelling

Voor een interactievere benadering van demo's voor het *chatbot* platform Oswald is er een AR-omgeving in elkaar gezet. Die zou bijvoorbeeld de vorm van een dierentuin kunnen aannemen die verkent kan worden met de hulp van een *chatbot*. De gebruiker kan via spraak informatie over de dieren opvragen en de *chatbot* zal daarna eveneens via spraak antwoorden. Zo wordt er een echte conversatie nagebootst. Om dit te realiseren wordt er gebruikgemaakt van van *speech to text* en *text to speech*. Daarnaast maakt het ARKit-platform het mogelijk om de AR-omgeving op te zetten met verschillende diermodellen. Dit alles zal samengebracht worden in een iOS-applicatie die de mogelijkheden van Oswald ter plaatse kan demonstreren.

3 Uitwerking stageopdracht

3.1 Gebruikte Technologieën

Er werden verschillende technologieën gebruikt om de applicatie te realiseren. Een overzicht over hoe deze technologieën samenwerken wordt in Figuur 3 weergegeven. Wanneer een gebruiker een vraag stelt aan de applicatie zal de uitgesproken vraag omgezet worden naar tekst door *Speech to text*. Deze tekst wordt naar de *chatbot*, gemaakt in Oswald, gestuurd. De *chatbot* haalt via de API, gemaakt in LoopBack, de nodige data uit de database. De *chatbot* geeft een tekstueel antwoord terug dat door middel van *text to speech* uitgesproken wordt. Naast de vraag van de gebruiker kan de *chatbot* ook gebruik maken van parameters die uit de AR-omgeving van de applicatie komen. In de volgende paragrafen zal dieper ingegaan worden op de individuelen technologieën.



Figuur 3 Schema samenwerking technologieën

3.1.1 Speech to text

Om ervoor te zorgen dat gebruikers spraak kunnen gebruiken om vragen te stellen aan de *chatbot*, is er *speech to text* geïmplementeerd. *Speech to text* is het omzetten van spraak naar tekst. Dit werd gedaan met behulp van het *speech framework* van iOS. Om dit te implementeren zal de applicatie toegang moeten hebben tot de microfoon en de spraakherkenning van het toestel.

Voor de omzetting van spraak naar tekst is er eerst een 'SFSpeechRecognizer' object nodig dat verantwoordelijk is voor het omzetten van de verkregen audio naar tekst. Elk 'SFSpeechRecognizer' object kan één taal herkennen. In deze stage is dat Nederlands. Daarnaast maakt het 'AVFoundation' *framework* het mogelijk om de live audio van de microfoon te verkrijgen. Deze audio wordt dan doorgegeven aan het 'SFSpeechRecognizer' object die deze audio gaat analyseren en omzetten naar tekst [3].

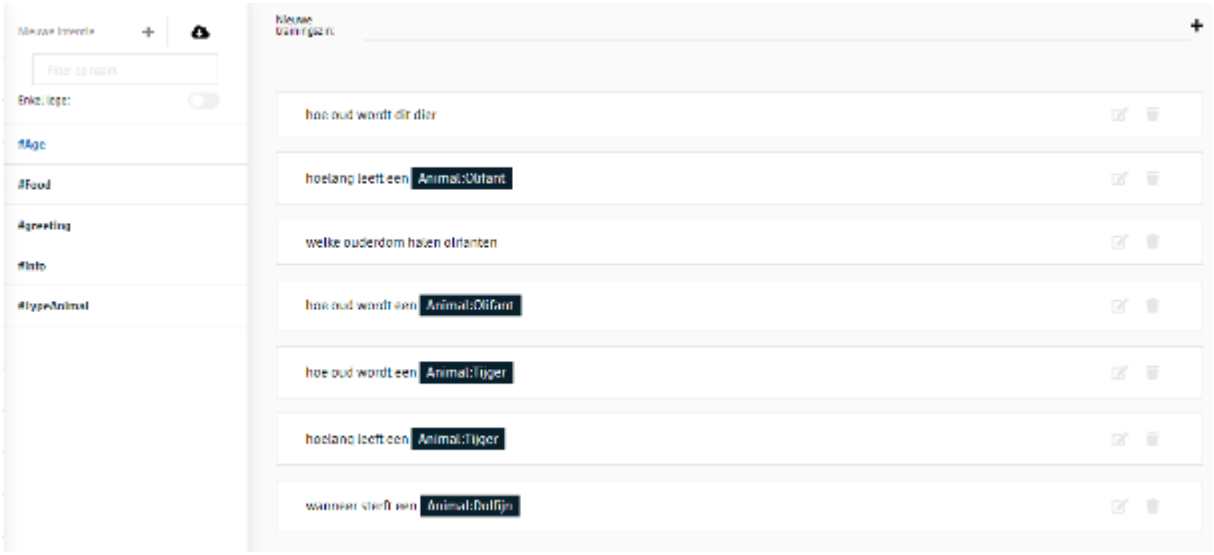
3.1.2 Oswald

De *chatbot* die gebruikt wordt in de applicatie, werd gemaakt met Oswald van Craftworkz. Dit is een platform dat ontwikkeld is in Python en het toelaat om op een gemakkelijke manier *chatbot*s te ontwikkelen. Om een *chatbot* binnen Oswald te maken en hem in staat te stellen tot het voeren van complexere conversaties zijn er een aantal onderdelen vereist: intenties, entiteiten en een conversatieboom die meerdere componenten kan bevatten.

3.1.2.1 Intenties

Het is mogelijk om de vraag 'Hoe oud wordt een olifant' te stellen, of: 'Hoelang leeft een olifant?'. Beiden vragen verwachten de levensduur van een olifant als antwoord ondanks de verschillende formulering. Omdat een gebruiker op verschillende manieren naar deze informatie kan vragen, werkt Oswald met intenties. Intenties representeren de algemene vraag waarop de gebruiker een antwoord wil en bevat een verzameling van voorbeeldzinnen.

De *chatbot* zal dan zichzelf trainen op basis van die voorbeeldzinnen en patronen of sleutelwoorden proberen te zoeken. Aan de hand van de voorbeeldzinnen kan de *chatbot* een algemeen model opstellen om nieuwe zinnen, die verschillen van de voorbeelden, aan de juiste intentie te koppelen. Dit is het *machine learning* gedeelte van Oswald dat volledig automatisch gebeurd [4]. Hoe intenties aangemaakt worden in Oswald wordt getoond in Figuur 4. In de linker kolom zijn de verschillende aangemaakte intenties zichtbaar met een *hashtag*. De intentie 'Age' is geselecteerd in de figuur. In de rechter kolom zijn de voorbeeldzinnen zichtbaar die aan de *chatbot* werden meegegeven voor die intentie. Deze zinnen zijn dus de trainingsdata voor de *chatbot*.



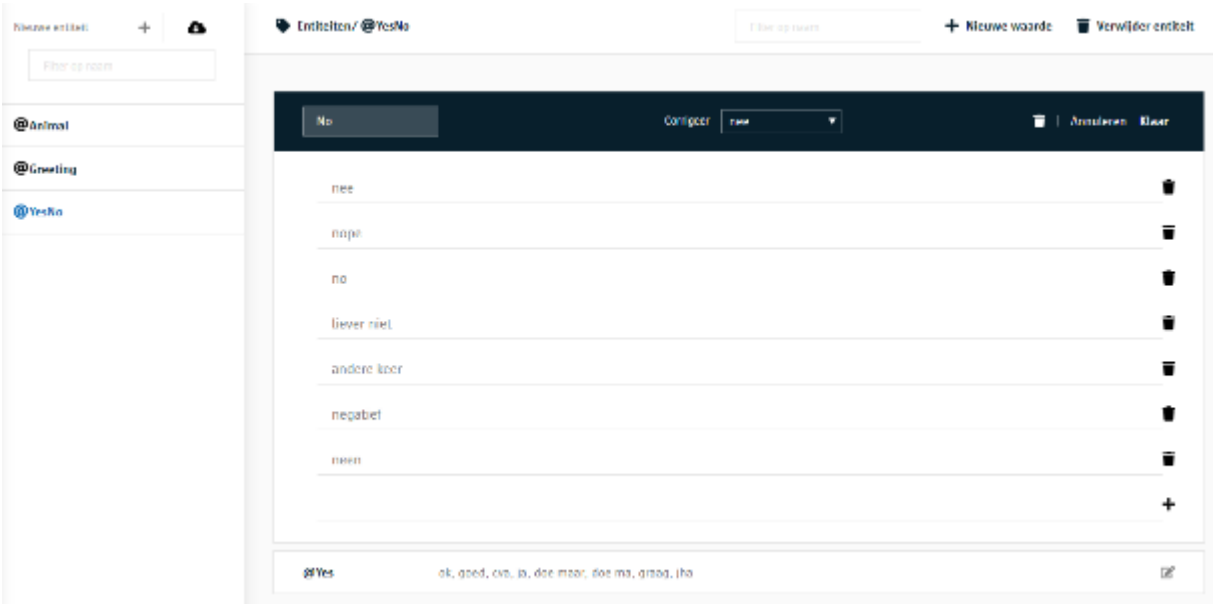
Figuur 4 Intenties aanmaken in Oswald.

3.1.2.2 Entiteiten

Intenties alleen zijn in de meeste gevallen niet genoeg om antwoorden te krijgen van de *chatbot*. Indien er enkel gebruikt gemaakt zou worden van intenties, zouden de vragen: 'Hoe oud wordt een olifant?' en 'Hoe oud wordt een leeuw?' ieder door een aparte intentie gerepresenteerd moeten worden. Wanneer dit voor een hele dierentuin gedaan moet worden, stapelen de intenties zich snel op. De onderscheiden factor in beiden vragen zit hem in het dier waarover het gaat, maar in beiden gevallen wordt er naar een leeftijd gevraagd. Om een antwoord te kunnen geven moet de *chatbot* dus

niet alleen de intentie ‘Age’ herkennen, maar ook het dier waarover het gaat. Het dier kan voorgesteld worden door een entiteit met type ‘Animal’, waarin heel wat verschillende dieren gedefinieerd worden. Het antwoord van de *chatbot* zal afhangen van de gedetecteerde intentie en van de entiteiten die gevonden worden in de zin. [5]

Soms is het handig dat verschillende synoniemen voor één entiteit op dezelfde manier herkend worden. Deze synoniemen kunnen per entiteit meegegeven worden. Zo kan een gebruiker op heel wat verschillende manieren ‘ja’ of ‘nee’ antwoorden op een vraag van de *chatbot*. Het toevoegen van synoniemen zorgt er bijvoorbeeld voor dat de antwoorden ‘negatief’, ‘no’ en ‘liever niet’ allemaal geïnterpreteerd worden als de entiteit ‘nee’. In Figuur 5 zijn in de linker kolom de verschillende entiteiten zichtbaar met een apenstaartje die aangemaakt werden in Oswald. De zwarte balk aan de rechter kant stelt een van de waarden ‘No’, binnen deze entiteit voor met zijn synoniemen.



Figuur 5 Entiteiten en hun synoniemen in Oswald

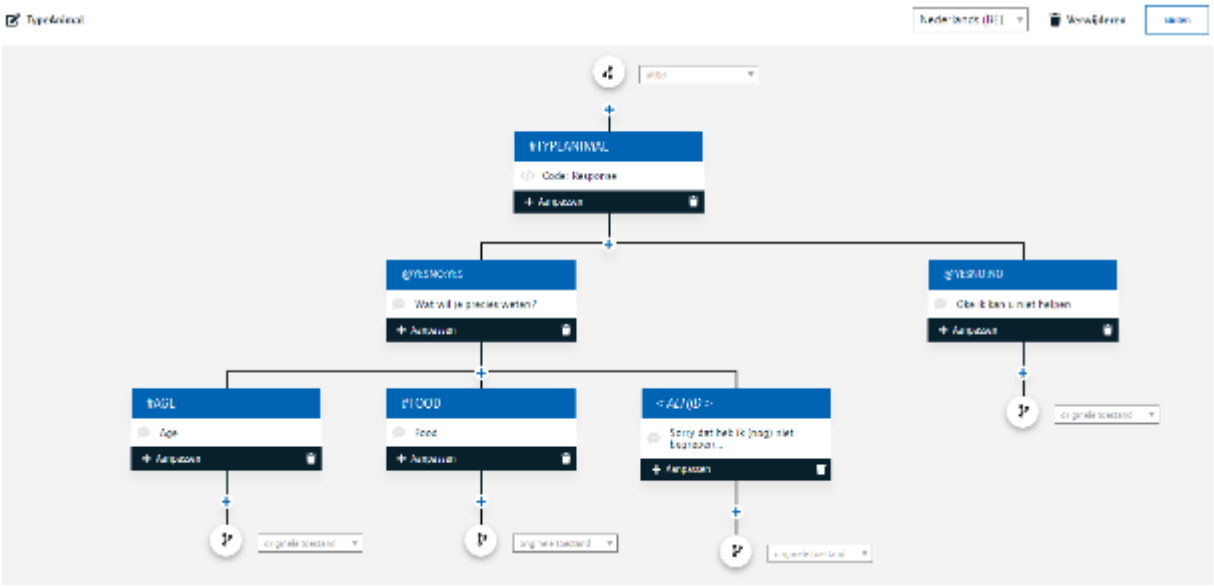
3.1.2.3 Metadata

Omdat er gewerkt wordt met een AR-applicatie waarin de dieren vertegenwoordigd worden door 3D-modellen die de gebruiker kan selecteren, wordt er niet altijd een dier meegegeven in de vraag. Zo kan de gebruiker over het geselecteerde dier de vraag stellen: ‘Hoe oud wordt dit dier?’. Er wordt hier geen entiteit ‘Animal’ meegegeven in de vraag en zou de *chatbot* geen antwoord kunnen geven. Er moet dus een manier gevonden worden om de *chatbot* te laten weten welke entiteit of dier de gebruiker bedoelt zonder het specifiek te vermelden. Extra variabele zoals het geselecteerde dier kunnen aan een vraag meegegeven worden door metadata. Die informatie kan gebruikt worden door de *chatbot* wanneer een entiteit ontbreekt.

3.1.2.4 Scenario’s

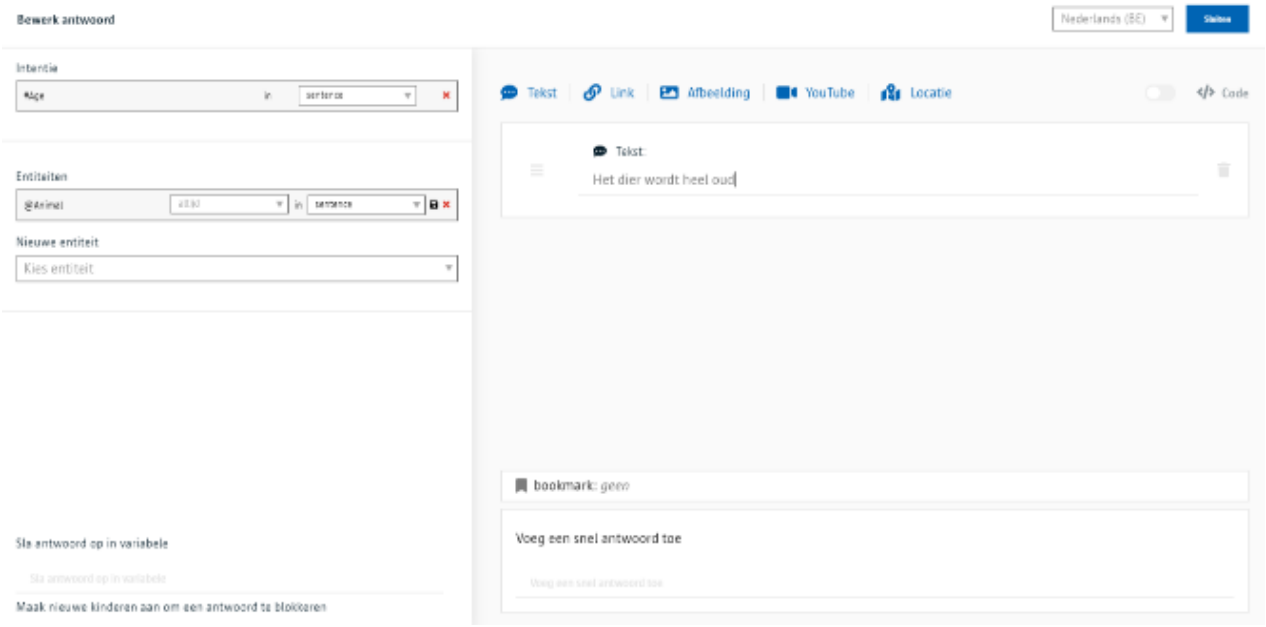
Enkel intenties en entiteiten zijn niet genoeg om een antwoordt te krijgen van de *chatbot*. Deze zullen de vraag wel interpreteren, maar geen antwoord produceren. Het antwoord van de *chatbot* wordt geformuleerd in de vorm van een scenario. Een scenario bestaat uit een conversatieboom die opgebouwd wordt uit een meerdere blokken van vraag en antwoord. Elke blok bestaat uit vooropgestelde combinaties van intenties en entiteiten die de *chatbot* kan herkennen. Bij het krijgen

van een input wordt de bijhorende blok in het scenario geselecteerd en geeft deze de gepaste output. Op basis van verdere vragen en antwoorden van de gebruiker kunnen verschillende takken in de conversatieboom gevolgd worden. Een visuele voorstelling van een scenario en de bijhorende conversatieboom wordt getoond in Figuur 6. [6]



Figuur 6 Een scenario in Oswald

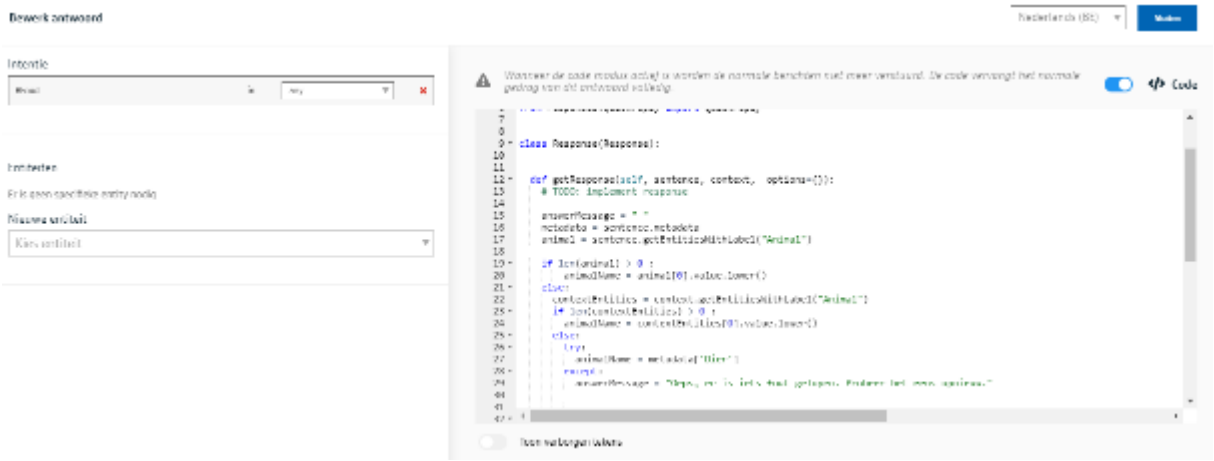
In Figuur 7 is te zien hoe er als intentie ‘Age’ werd gekozen voor een bepaald antwoord. Deze kan herkend worden in combinatie met de gekozen entiteit ‘Animal’ tijdens het analyseren van de vraag van de gebruiker. Indien de intentie en entiteit inderdaad overeenkomen, zal het voorop ingestelde antwoord gegeven worden: ‘Het dier wordt heel oud’.



Figuur 7 Een snel antwoord toevoegen aan scenario in Oswald

Deze manier van werken is een zogenaamde *quick reply* omdat de tekst die teruggegeven wordt altijd dezelfde is en snel ingesteld kan worden. Dit kan handig zijn, maar meestal is het gewenst om een gevarieerd antwoord te geven. Omdat er bij een *quick reply* enkel gekeken wordt naar de ingestelde intentie en entiteiten is het niet mogelijk om met metadata te werken die de applicatie naar de *chatbot* stuurt.

Het alternatief is het gebruik van *'advanced code responses'* zoals te zien is in Figuur 8. Deze *response* vervangt de *quick reply* en biedt veel meer mogelijkheden. Bij de *'advanced code response'* krijgt de *chatbot*-maker toegang tot de *response* klasse van de *chatbot*. Binnen deze klasse heeft de *chatbot*-maker een zekere maat van vrijheid omdat hij gebruik kan maken van de functies binnen Python. Zo kan hij onder andere *API-calls* maken om extern data op te halen of zelf data weg te schrijven. Er kunnen eveneens extra parameters toegevoegd worden waarmee rekening gehouden moet worden, zoals bijvoorbeeld metadata.



Figuur 8 Advanced code response aanmaken in Oswald

3.1.3 Text to speech

Een gebruiker kan tegen de *chatbot* praten en zal een antwoord terugkrijgen. Dit antwoord zal in eerste instantie tekstueel zijn, maar om een conversatie na te bootsen is een gesproken antwoord interessanter. Om de *chatbot* menselijker te maken, worden zijn antwoorden naar spraak omgezet doormiddel van *text to speech.* Om tekst om te zetten naar gesproken taal is er gebruik gemaakt van het 'AVFoundation' *framework* van iOS. Als eerste is er een 'AVSpeechUtterance' object nodig dat de uitgesproken zin representeert. Die bevat de tekstuele zin alsook extra parameters zoals stemintonatie en spraaksnelheid. Daarnaast is er een 'AVSpeechSynthesizer' object nodig die verantwoordelijk is voor het omzetten van de tekstuele zin naar een gesproken zin [7].

3.1.4 ARKit

De diermodellen die deel uitmaken van de virtuele dierentuin worden in de iOS-applicatie getoond via AR. Om deze AR-omgeving op te zetten, werd er gebruik gemaakt van het ARKit-*framework*. De 3D-modellen kunnen door de gebruiker zelf in de AR-omgeving geplaatst worden op een gewenste positie door op het scherm te drukken. In Figuur 9 is te zien hoe een gebruiker een 3D-model kan plaatsen.

Figuur 9 AR-omgeving

Om de 2D positie op het scherm van het apparaat te vertalen naar een locatie in de 3D AR-omgeving, werd er gewerkt met de *hittests* van het ARkit-framework. Deze *hittests* gebruiken de 2D coördinaten om herkenningspunten uit de 3D-omgeving op te halen van deze locatie. Deze herkenningspunten of *featurepoints* dienen dan voor het aanmaken van een ankerplaats voor het 3D-model [8].

3.1.5 LoopBack en Cloudstore

LoopBack is een *framework* dat het mogelijk maakt om op een zeer snelle en makkelijke manier web API's te ontwikkelen. De API wordt gebruik om een connectie te leggen tussen de *chatbot* en de Cloud Firestore database waarin alle antwoorden voor de *advanced code responses* worden bijgehouden. Een antwoord kan uit deze database gehaald worden door een *http request* te doen naar de API. De API leest hierbij de opgevraagde informatie uit de database uit en stuurt deze terug naar de *chatbot*.

3.2 Resultaat

De eigenlijke iOS-applicatie werd ontwikkeld in Swift. De voorgaande technieken werden verwerkt in de applicatie. De verschillende onderdelen van de applicatie worden in de volgende paragrafen besproken zodat er een idee geschept kan worden over hoe de applicatie opgebouwd is.

3.2.1 Startscherm

Het eerste wat een gebruiker te zien krijgt bij het openen van de applicatie is het startscherm, dit wordt visueel weergegeven in Figuur 10. Omdat de applicatie gebruikt wordt als een demo is ervoor gekozen om dit scherm elke keer te tonen als de applicatie gestart wordt. Op dit startscherm zal de gebruiker begroet worden en zal hij de keuze krijgen om meer uitleg over de applicatie te krijgen of om direct aan de demo te beginnen.

Figuur 10 Startscherm

3.2.2 Handleiding

Binnen de handleiding krijgt de gebruiker meer uitleg over hoe de applicatie werkt en welke functies beschikbaar zijn. Deze handleiding is opgebouwd uit meerdere stappen en zal bij elke stap uitleg geven over een specifieke functionaliteit. In Figuur 11 is weergegeven hoe zo een stap eruitziet voor de gebruiker. Om te navigeren tussen stappen kan de gebruiker over het scherm heen *swipen*. Een *swipe* naar links geeft de eerstvolgende stap weer terwijl het *swipen* naar rechts de voorgaande stap zal tonen. Een gebruiker zal ook altijd een visuele aanwijzing krijgen over hoeveel stappen hij nog te gaan heeft. Dit wordt helemaal onderaan op het scherm aangeduid doormiddel van bolletjes.

Figuur 11 Handleiding scherm

3.2.3 Hoofdscherm

In het hoofdscherm kan de gebruiker 3D modellen van dieren in AR zien en met de *chatbot* praten. Dit scherm is opgedeeld in twee onderdelen met als eerste een *AR-view* en een *overlay* menu met een aantal knoppen.

3.2.4 AR-view

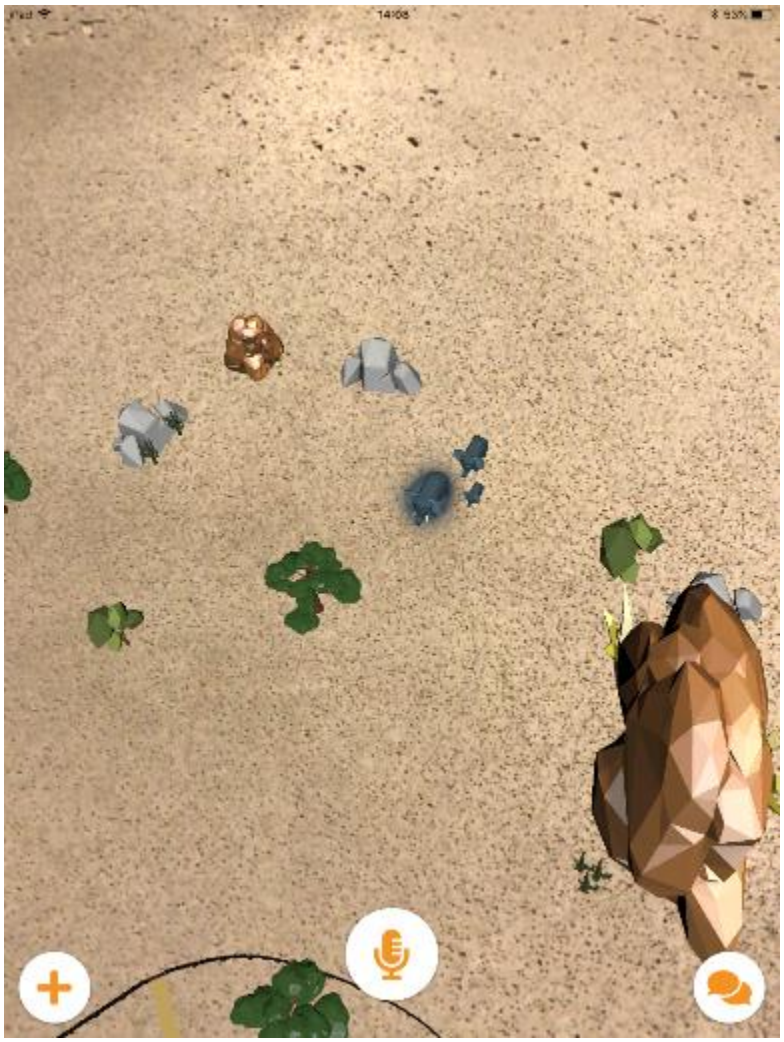
Binnen de AR-*view* kan de gebruiker de virtuele dieren en het landschap bekijken. Het eerste wat de gebruiker te zien krijgt, is enkel de live camerabeelden. De applicatie zal de omgeving scannen en *feature points* detecteren. Wanneer er genoeg *feature points* gedetecteerd zijn verschijnt er een 3D model op de gebruiker zijn scherm. Dit model is doorschijnend en staat altijd in het midden van het scherm, in Figuur 12 wordt dit visueel weergegeven. Dit model is een indicatie van waar het finale 3D model zal staan. Het plaatsen van het model in het midden van het scherm gebeurt door een *hittest* die het *feature point* het dichts bij het centrum van het scherm selecteert als anker voor het 3D-model. Om ervoor te zorgen dat het 3D-model ook bij bijvoorbeeld rotatie zijn centrale positie behoud wordt de *hittest* opnieuw uitgevoerd zodra er andere feature point gedetecteerd worden.

Figuur 12 Visuele indicatie van plaatsing 3D model

Door het toestel te bewegen kan de gebruiker het 3D-model plaatsen waar hij wil. Wanneer de gebruiker klaar is en het model wil vastzetten, kan hij dit doen door eenmaal op het scherm te drukken. Nu zal het model vastgezet worden op het *feature point* dat op dat moment het anker was voor het model. Omdat dit de finale positie van het model is, is er geen nood meer aan het uitvoeren van *hittesten*. Na het vastzetten van het model krijgt de gebruiker de mogelijkheid om het 3D-model nog aan te passen. De gebruiker kan het 3D-model veranderen door te drukken op het scherm. Om de grote van het model aan te passen kan de hij de *pinch* of *zoom* beweging gebruiken. Om het 3D-model te draaien kan hij een roterende beweging maken met twee vingers. Wanneer het 3D-model helemaal naar wens is kan hij door lang op het scherm te drukken het plaatsingsproces afronden. Hierdoor zal het model niet meer transparant zijn en kan het ook niet meer aangepast worden. In Figuur 13 is te zien hoe een model geplaatst wordt en zijn transparantie verliest. Indien de gebruiker het model toch wil aanpassen, kan hij door lang op het scherm te drukken dit wel weer doen.

Figuur 13 Aanpassen en vastzetten van het 3D model

Omdat er meerdere dieren op het scherm kunnen zijn, is het onmogelijk voor de *chatbot* om te weten over welk dier een gebruiker een vraag stelt. Daarom kan de gebruiker het dier waarover hij een vraag wil stellen selecteren door op het dier op het scherm te drukken. Dit selecteer process is opnieuw een hittest die uitgevoerd wordt. In dit geval wordt er enkel rekening gehouden met de aanwezige 3D-modellen. Indien de hittest een dier als resultaat geeft,wordt er een contour rond het dier getekend die een visuele aanduiding geeft over welk dier geselecteerd is. Dit wordt weergegeven in Figuur 14. Als er echter geen dier gevonden word in de hittest, maar bijvoorbeeld een boom of een steen, gebeurt er niets. Nu het dier geselecteerd is, wordt deze selectie als metadata meegegeven bij elke vraag die de gebruiker stelt.



Figuur 14 Selecteren van een dier

3.2.5 Overlay menu

Het *overlay* menu is een menu helemaal onderaan op het scherm dat bestaat uit drie knoppen, in Figuur 15 wordt dit visueel weergegeven. Als eerste is er de knop met een microfoonicoon. Wanneer een gebruiker hierop drukt zal de applicatie luisteren naar de uitgesproken vraag die de gebruiker stelt. Dit gebeurt doordat de *audio engine* start bij het indrukken van deze knop. Hierop volgend wordt eveneens de *speech recognizer* gestart die de spraak gaat herkennen. Elke keer als er een woord herkend wordt, zet de *speech recognizer* deze om naar tekst. De applicatie detecteert automatisch wanneer de gebruiker gestopt is met spreken indien er anderhalve seconde geen spraak gedetecteerd wordt. De output tekst wordt samen met eventuele metadata naar de *chatbot* gestuurd via een *http request*.

Figuur 15 Overlay menu

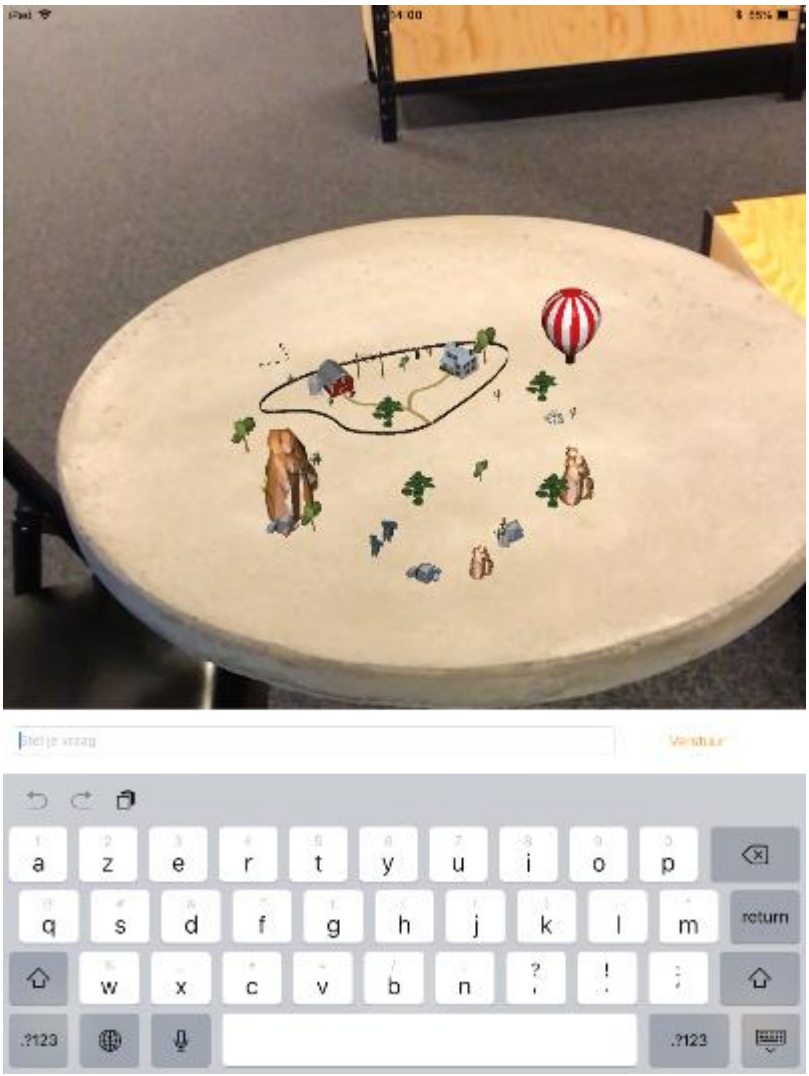
Omdat de Nederlandse spraakherkenning niet perfect werkt en sommige woorden moeilijk herkend worden, krijgt de gebruiker ook een visuele weergave van de zin die herkend werd in een label bovenaan op het scherm zoals te zien is in Figuur 16.

Figuur 16 Een vraag stellen aan de chatbot

De chatbot gaat de verkregen vraag matchen met een van de vooraf ingestelde intenties en houd hierbij rekening met de metadata. Via een API-*call* wordt het gewilde antwoord opgehaald uit de Cloud Firestore database. Wanneer de *chatbot* een antwoord teruggeeft, zal deze uitgesproken worden via *text to speech*. Het kan echter zijn dat de gebruiker het antwoord niet verstaan heeft. Om ervoor te zorgen dat de gebruiker dan niet nog eens dezelfde vraag moet stellen zal het antwoord van de *chatbot* op het scherm getoond worden, een voorbeeld hiervan is te zien in Figuur 17. Hierdoor zal de gebruiker altijd het antwoord kunnen lezen.

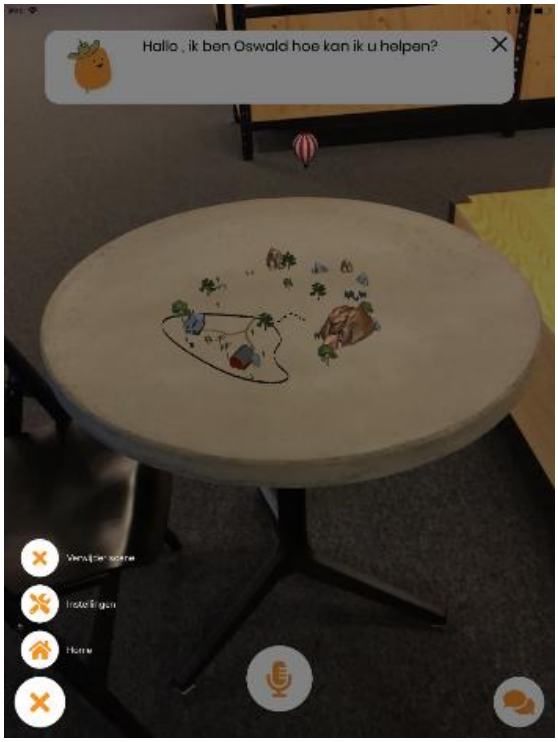
Figuur 17 Antwoord van de chatbot

Omdat de applicatie als demo gebruikt gaat worden en dus ook op drukkere plaatsen gebruikt zal worden, is het mogelijk dat de spraakherkenning niet goed werkt. Te veel achtergrondgeluid zou een oorzaak kunnen zijn of omdat de applicatie de gebruiker om een andere reden niet goed verstaat. Om dit op te lossen kan de gebruiker ook vragen stellen doormiddel van een getypt tekstbericht. Deze mogelijkheid krijgt hij wanneer hij op de knop met de tekstballonnen drukt. Hierdoor zal het virtueel toetsenbord van het toestel tevoorschijn komen en kan de gebruiker zijn vraag in typen, dit is te zien in Figuur 18. Wanneer de gebruiker klaar is, drukt hij op ‘verstuur’ en zal de *chatbot* deze tekst rechtstreeks gebruiken om te antwoorden.



Figuur 18 Overlay menu: tekstballonen knop geklikt

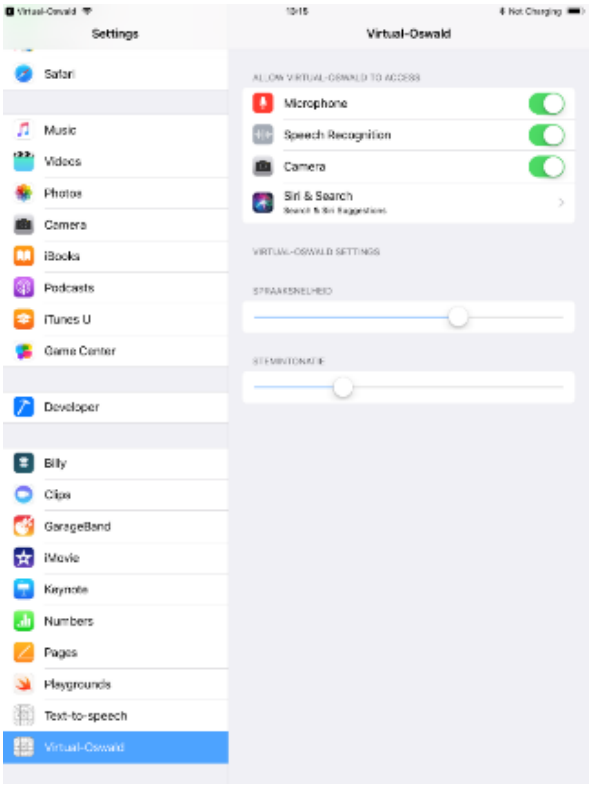
Als laatste is er de ‘+’ knop. Wanneer een gebruiker hierop drukt, zal er een menu openen met een aantal opties: ‘home’, ‘Instellingen’ en ‘verwijder scene’ dit wordt weergegeven in Figuur 19 Overlay menu: ‘+’ knop geselecteerd. Wanneer de gebruiker op ‘home’ drukt zal de applicatie terug naar het startscherm gaan en de demo sessie afsluiten zodat de applicatie klaar is voor de volgende gebruiker. De tweede optie is het openen van de instellingen van de applicatie. De laatste optie is om een 3D model van het scherm te verwijderen, hierdoor zal de gebruiker de kans krijgen om het 3D-model opnieuw te plaatsen.



Figuur 19 Overlay menu: '+' knop geselecteerd

3.2.6 Instellingen

Binnen de applicatie kan de gebruiker een aantal instellingen aanpassen. Dit gebeurt in een instellingen pagina van iOS zelf zie Figuur 20. De gebruiker heeft hier een overzicht van de toestemmingen die hij gegeven heeft aan de applicatie en enkele instellingen die hij kan veranderen.



Figuur 20 Instellingen scherm van de applicatie

II. Wat is de beste implementatie van image recognition in een iOS applicatie?

1 Inleiding

Image recognition wordt steeds vaker gebruikt en geïmplementeerd in uiteenlopende toepassingen, onder andere in mobiele applicaties. Er zijn meerdere manieren waarop deze integratie kan gebeuren. De *image recognition* kan binnen dit eindwerk op het toestel zelf uitgevoerd worden of er kan een *cloud* API gebruikt worden. Met dit onderzoek zal achterhaalt worden welke methode het efficiëntst is voor iOS-applicaties in verschillende toepassingen. Hiervoor worden een aantal aspecten van de integratiemethodes onderzocht en vervolgens vergeleken. Er zal bijvoorbeeld gekeken worden naar de snelheid van *Image recognition* na implementatie, de accuraatheid, etc. Op basis van de resultaten worden aanbevelingen per toepassing aangereikt.

In het eerste deel van het onderzoek wordt er gekeken naar de werking van *image recognition* via een literatuurstudie. In het tweede deel wordt er ingegaan op de praktische onderdelen van het onderzoek en zullen de gebruikte technieken overlopen worden. De uitvoering hiervan zal uiteindelijk leiden tot resultaten van de verschillende implementatiemethoden die vergeleken worden.

2 Image recognition

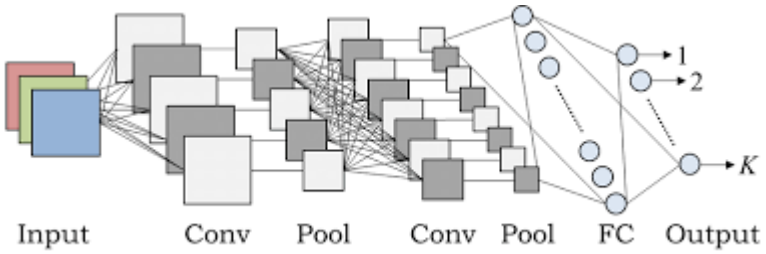
De manier waarop de mens een afbeelding interpreteert, is compleet verschillend dan die van een computer. Mensen hebben geen moeite met het identificeren van objecten in afbeeldingen doordat ze verschillende *features* ervan duidelijk en gemakkelijk kunnen herkennen en onderscheiden. Dat komt omdat de hersenen van kinds af aan onbewust getraind zijn door het herhaaldelijke zien van verschillende objecten, wat ertoe geleid heeft dat de mens moeiteloos dingen van elkaar kan onderscheiden en herkennen. In tegenstelling tot menselijke hersenen, bekijkt een computer visuele input als een matrix van getallen. Een computer gebruikt beeldverwerkingsalgoritmen om afbeeldingen te analyseren en te herkennen. [9]

Image recognition is het herkennen van voorwerpen in afbeeldingen en wordt gebruikt voor uiteenlopende toepassingen, van het classificeren van afbeeldingen tot het helpen bij het besturen van zelfrijdende auto's. De afbeeldingen die gepresenteerd worden aan het *image recognition* systeem worden eerst ontleed tot een *feature map* van hun *features* voordat er verder mee gewerkt kan worden. Deze *feature map* kan door *image recognition algorithms* verwerkt worden om de *features* te herkennen en te classificeren [10].

Een van de *image recognition* algoritmes is een *image classifier*. Een *image classifier* neemt een input zoals een afbeelding en classificeert die binnen een klasse of hangt hieraan een waarschijnlijkheid dat de input binnen een bepaalde klasse hoort [11]. Deze *image classifiers* is één voorbeeld van *image recognition* algoritmes, maar er bestaan ook andere soorten algoritmes die voor meer ingewikkelde taken gebruikt kunnen worden [10].

2.1 Convolutional neural network

Een CNN is een type van een neuraal netwerk dat gespecialiseerd is in het verwerken van pixeldata. Een CNN is opgebouwd uit meerdere *Layers*. Een *convolution layer, Pooling layer* en een *fully connected layer* vormen de basis van een CNN. De *convolution* en *pooling layer* kunnen meerdere malen herhaald worden terwijl een *fully connected layer* slechts eenmaal voorkomt. Sommige CNN's hebben naast deze basis *layers* nog andere die het systeem kunnen verbeteren. Een voorbeeld van de opbouw van een CNN is te zien in Figuur 21.

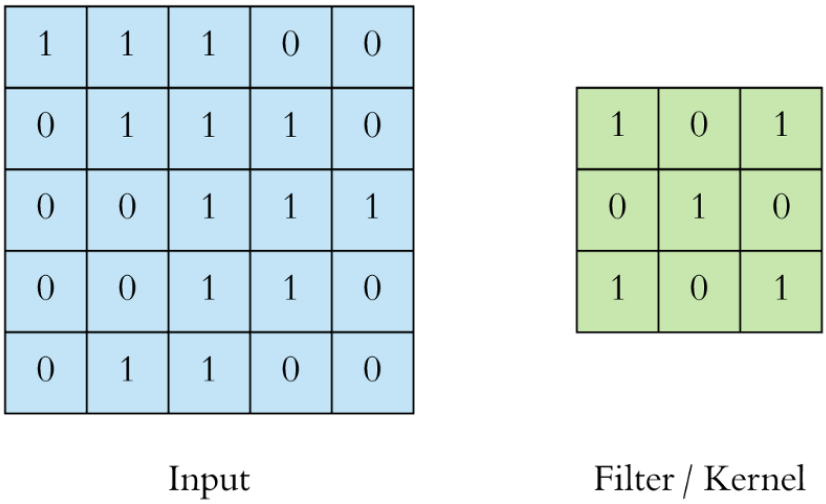


Figuur 21 Voorbeeld CNN

De input van een CNN kan bijvoorbeeld een afbeelding zijn. Deze afbeelding gaat eerst door de *convolution layer* die de *feateres* uit de afbeelding haalt en in een *feature map* plaatst. Na de *convolution layer* is er de *pooling layer.* Deze *layer* zorgt ervoor dat de *feature map* verkleind wordt zodat er minder parameters zijn en deze sneller verwerkt kan worden. Bij dit proces gaat er informatie verloren, maar de belangrijke informatie over de *features* blijft behouden. De laatste *layer* in een CNN is de *fully connected layer* die aan de hand van de verkregen *feature map* voorspelt tot welke klasse de afbeelding behoord.

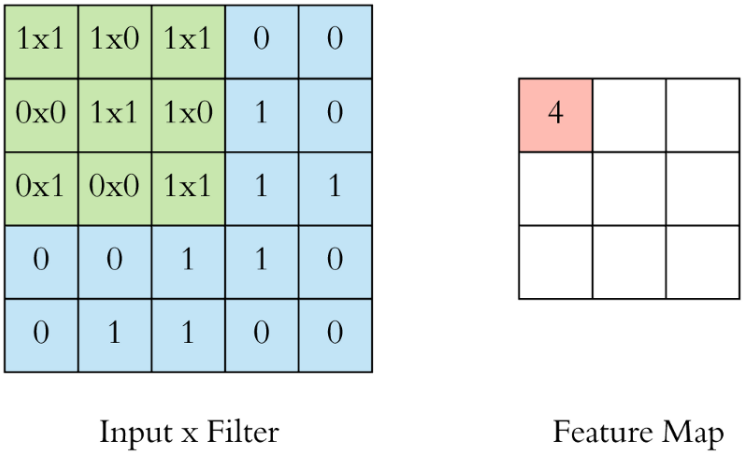
2.1.1 Convolution layer

De *convolution layer* is de eerste stap in een CNN en zorgt voor het extraheren van *features* uit de input afbeelding. De *features* worden herkend aan de hand van een filter, ook wel *kernel* genoemd en worden samengevat in een *feature map*. In Figuur 22 is aan de linkerkant een input van de *convolution layer* te zien, aan de rechterkant een filter.



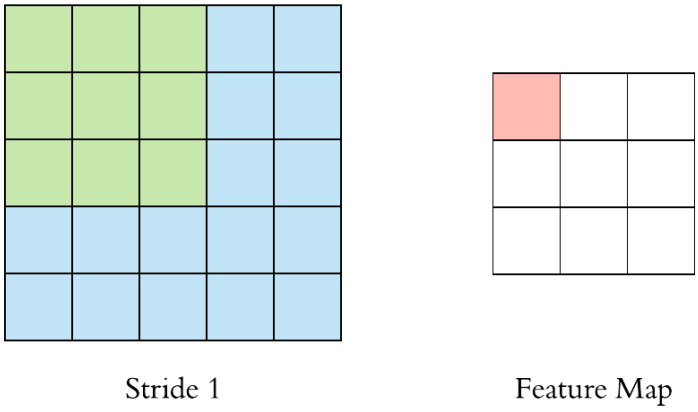
Figuur 22 Voorbeeld van een input en filter

In Figuur 23 is te zien hoe de *convolution layer* te werk gaat. De groene matrix, ook wel het *receptive field* genoemd, is de matrix waar de convolutie op toegepast wordt. Deze zal over de afbeelding heen schuiven zodat de filter elke keer op een andere groep van pixels uit de input toegepast wordt. In Figuur 23 Voorbeeld convolutionis een voorbeeld weergegeven. Hierbij wordt elke pixel, voorgesteld door een 0 of 1, eerst vermenigvuldigd met waarde van de filter waarna de som gemaakt wordt van het volledige *receptive field*. Dit resultaat wordt dan in een *feature map* geplaatst.



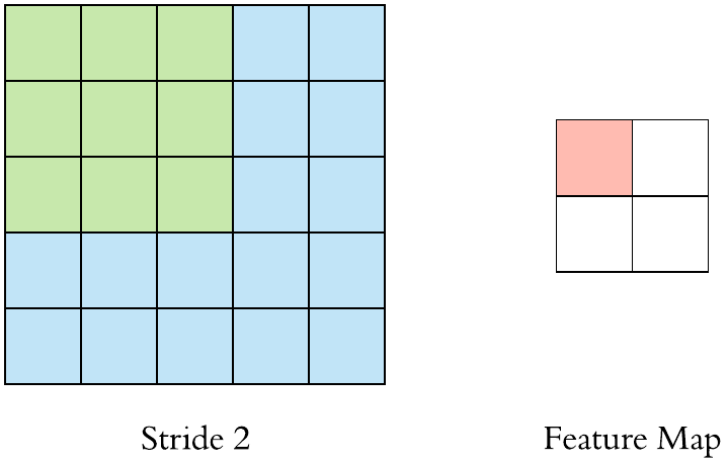
Figuur 23 Voorbeeld convolution

Hoeveel de filter verplaatst hangt af van de *stride*, standaard is dit één. Dit wil zeggen dat bij elke stap de filter één pixel zal verschuiven. Een voorbeeld hiervan is te zien in Figuur 24 voorbeeld stride 1.



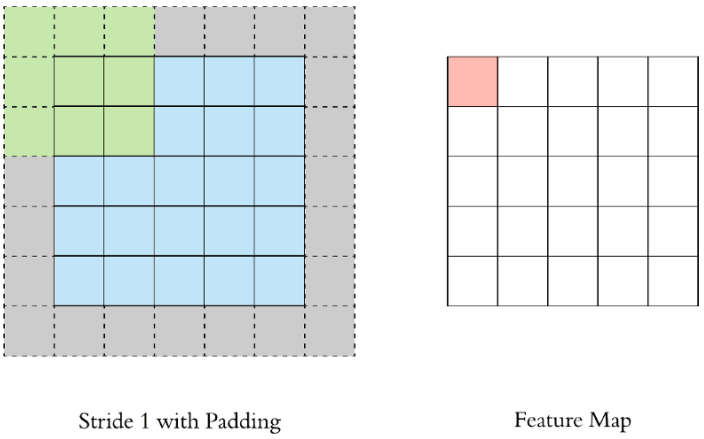
Figuur 24 voorbeeld stride 1

Het is natuurlijk ook mogelijk om een *stride* groter dan één te gebruiken. In Figuur 25 is voorbeeld te zien van een *stride* met waarde twee. Hierbij zal de filter na elke stap twee pixels verschuiven. Hierdoor zijn er minder stappen nodig om de filter over de hele input te laten gaan.



Figuur 25 voorbeeld stride 2

Zoals te zien is in Figuur 24 en Figuur 25 is de *feature map* kleiner dan de input. Om de originele grote van de input te behouden kan er gebruik gemaakt worden van *padding. Padding* is het toevoegen van waarden aan de randen van de input. Binnen een CNN wordt *padding* vaak gedaan om de grote van de input te behouden, anders zou deze na elke laag krimpen. Een voorbeeld hiervan is te zien in Figuur 26.



Figuur 26 voorbeeld padding

Hoe groot het *receptive field* is, hangt af van de dimensies van de filter. Deze dimensies bestaan uit een breedte, een lengte en een diepte. De breedte en lengte spreken voor zichzelf en zijn vrij te kiezen. De diepte, ook wel *channels* of kleurenkanalen genoemd, is echter geen werkelijke diepte, maar wordt bepaald door de kleurwaarden van de afbeelding. Voor afbeeldingen in grijswaarde zal de filter slechts een diepte van twee hebben. De pixels worden geïnterpreteerd als een 2D matrix waarvan de waarden tussen 0 en 255 kunnen liggen. 0 betekent hierbij volledig zwart, terwijl 255 overeenkomt met volledig wit. Alle waarde die hiertussen liggen stellen een bepaalde grijswaarden voor. Voor afbeeldingen met kleur zijn er drie dimensies nodig aangezien elke pixel drie kleurwaarden heeft, namelijk rood, blauw en groen. Ook die kleurwaarden hebben een waarde tussen 0 en 255. Om de effectieve kleur te reconstrueren moeten de drie waarden gecombineerd worden [11].

De diepte van de filter moet overeenkomen met het aantal kleurenkanalen van de afbeelding. Als er dus een filter aangemaakt moet worden voor een afbeelding met drie *channels* zal de diepte drie zijn. Zo kan de filter bijvoorbeeld bestaan uit een matrix met de dimensies 5x5x3. Op elke positie zal de filter de waarden in de filter vermenigvuldigen met de originele waarden van de pixels. Die vermenigvuldigen binnen de afmetingen van de filter worden dan opgeteld tot één waarde dat een deel van de afbeelding representeert. Dit proces wordt herhaald voor iedere pixelgroep over de hele afbeelding. De verzameling van al deze waardes vormen uiteindelijk een *features map*. Er kunnen meerdere filters gebruikt worden op dezelfde afbeelding om de spatiale relaties te behouden. De *feature map* is kleiner dan de originele afbeelding wat het makkelijker maakt om er mee te werken. Er gaat hierbij een deel originele informatie van de afbeelding verloren, maar de belangrijke herkenningspunten blijven behouden [11].

Er kunnen meerdere *feature maps* verkregen worden door het toepassen van verschillende filters. Zo worden er verschillende *features* geïdentificeerd die het CNN kan gebruiken om te leren. De matrices waaruit een filter bestaat, kunnen verschillende waarden bevatten om verschillende resultaten te verkrijgen. De verschillende patronen van waarden die gebruikt worden in de filter helpen bij het herkennen van de contouren, randen, hoeken, etc. van een afbeelding zoals te zien is in Figuur 27 en Figuur 28 . Het primaire doel van deze *convolution layer* is dus het detecteren van *features* in de afbeelding en ze te verzamelen in een *feature map*. Tegelijkertijd wordt de spatiale verhouding tussen de pixels bewaard [11] .

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

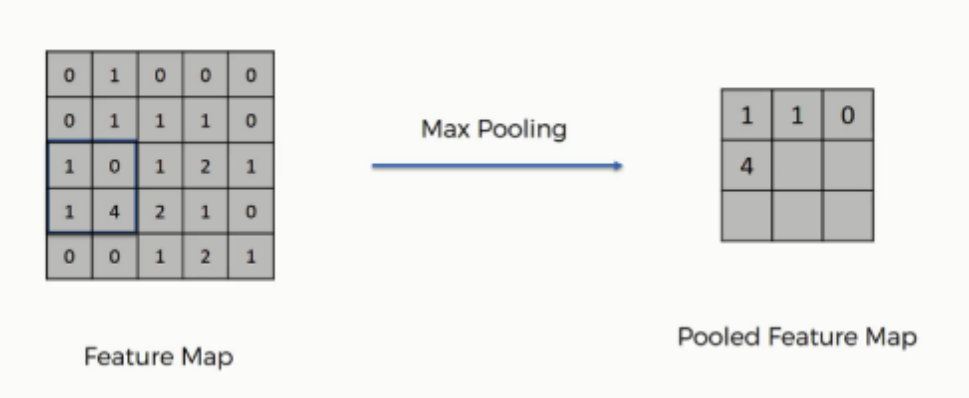
Figuur 27 Filter maakt afbeelding wazig

0	-1	0
-1	5	-1
0	-1	0

Figuur 28 Filter maakt afbeelding scherper

2.1.2 Pooling layers

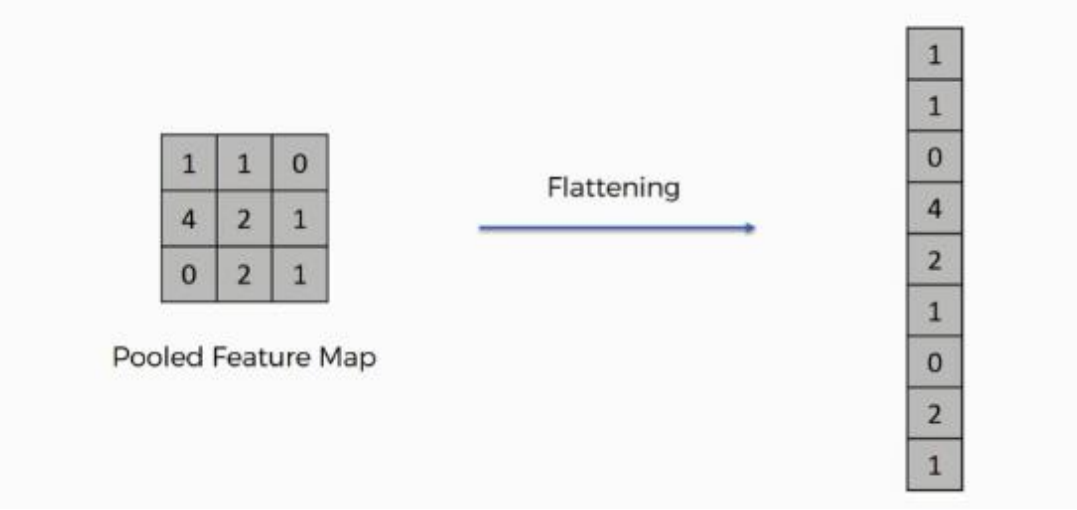
Er zijn verschillende types van *pooling* zoals: *min pooling, sum pooling, max pooling, etc.* Het meest gebruikte *pooling* type is *max pooling.* Dit proces bestaat uit het verschuiven van een matrix, meestal 2x2, over de *feature map*. Elke keer zullen er vier pixels genomen worden waarvan de maximumwaarde binnen de pixelgroep in de *pooled feature map* gezet worden. Na die vier pixels zal de matrix verschuiven naar de volgende groep van vier pixels totdat de hele *feature map* overlopen is. Tijdens dit proces zal elke pixel maar één keer door de matrix gaan waardoor er dus nooit een overlap is zoals te zien is in Figuur 29.



Figuur 29 Voorbeeld Max pooling

Net zoals bij het *Convolution* proces zal er bij dit proces ook informatie verloren gaan. Bij het gebruik van een matrix van 2x2 gaat tot wel 75% van de originele informatie van de *feature map* verloren omdat er uit de vier pixels enkel de pixel met de hoogste waarde overhouden wordt. De informatie die verloren gaat is echter niet belangrijk voor het CNN. De belangrijke informatie voor het detecteren van de afbeelding blijft behouden in de *pooled feature map*. Doordat door het *pooling* proces de *pooled feature map* kleiner is en minder parameters heeft, zal deze sneller verwerkt kunnen worden.

Na de *pooling layer* is er een *pooled feature map* bekomen. Deze kan echter nog niet aan het artificieel neuraal netwerk gegeven worden. Voordat de *pooled feature map* gebruikt kan worden zal hij van een matrix naar één kolom met getallen omgezet moeten worden ook wel *flattening* genoemd. Een voorbeeld hiervan is te zien in Figuur 30.

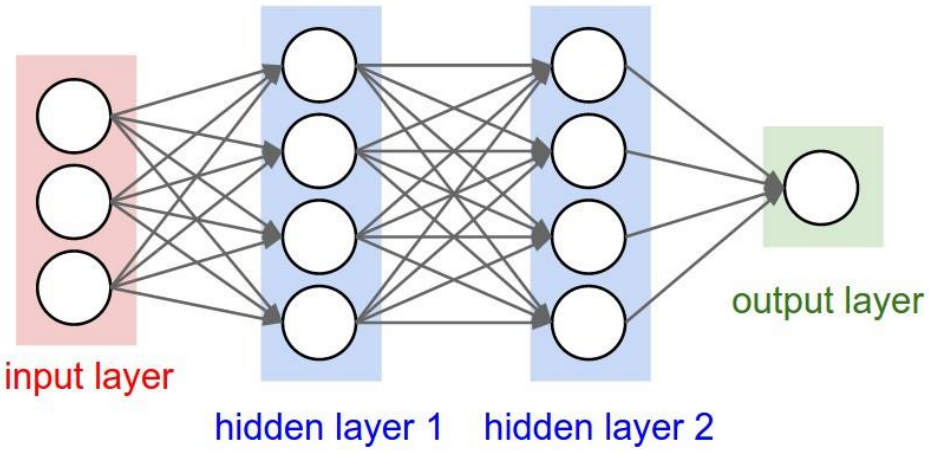


Figuur 30 Voorbeeld van Flattening

2.1.3 Fully-connected Layer

Aan het einde van een CNN wordt de output van alle voorgaande *convolution* en *pooling layers* als input gebruikt voor de laatste layer: de *fully-connected layer*. Deze kan bestaan uit een *input layer, output layer en* meerdere *hidden layers* die met elkaar samenwerken. *Fully-connected* duidt op het feit

dat elk *node* in een layer verbonden is met elke node van de vorige *layer*. Deze structuur wordt weergegeven in Figuur 31. *Fully- connected layers* voeren classificatie uit aan de hand van de features die ze verkrijgen uit de *input layer*. De *fully-connected layer* is niets meer dan een traditioneel *artificieel neuraal netwerk* (ANN) die de input verwerkt tot een waarschijnlijkheid of 'probability' voor iedere klasse die het model bevat. Meer informatie over ANN's is terug te vinden in het boek *Fundamentals of artificial neural networks* (Hassoun, Mohamad H.) [12].



Figuur 31 *Fully-Connected Layer*

2.2 Onderzoeksmethode

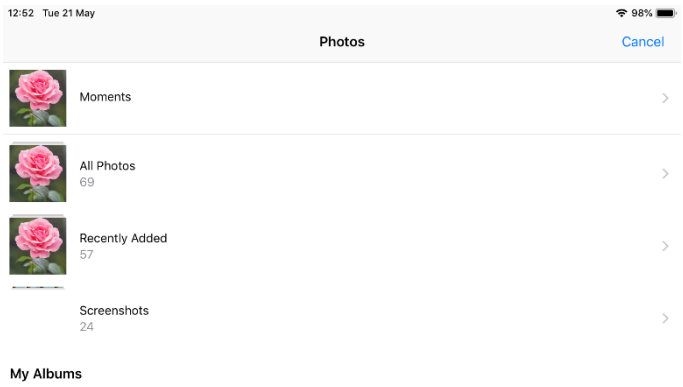
Image recognition op een mobiel toestel kan op meerdere manieren gedaan worden. Enerzijds is het een optie om de *image recognition* door het toestel zelf uit te laten voeren door bijvoorbeeld het Core ML framework. Anderzijds kan *image recognition* ook gebeuren via een API. Beide implementatiemethodes zullen verwerkt worden in een mobiele applicatie en gebruiken hetzelfde Tenserflow model om aan *image recognition* te doen. Na het opstellen van deze experimentele omgeving zullen dezelfde testen op deze twee applicaties uitgevoerd worden om met de resultaten een vergelijking te kunnen maken over hun performantie.

Om *image recognition* te doen is er een model nodig. Het is onder andere mogelijk om zelf een model te maken en te trainen, maar er kan ook een bestaand model gekozen worden. Voor het onderzoek werd het MobileNet_V1 model gekozen. Dit is een bestaand Tensorflow model dat al duizend klassen bevat waarin een afbeelding kan geclassificeerd worden. Als input verwacht het model een afbeelding van 224 x 224 pixels met drie kleuren kanalen.

Om de implementatie methodes te testen zijn er twee simpele applicaties gemaakt waarmee deze vergeleken kunnen worden. Beiden applicaties zijn gelijkaardig opgebouwd. Het hoofdscherm is zeer minimalistisch en bevat enkel een *label*, een knop en een *imageView*. Een voorbeeld hiervan is te zien in Figuur 32.

Figuur 32 Beginscherm applicatie

Wanneer er op ‘Selecteer afbeelding’ gedrukt wordt, opent een *imagePicker* en is het mogelijk om een afbeelding te selecteren zoals te zien is in Figuur 33.



Figuur 33 Selecteren van een foto

Wanneer een afbeelding geselecteerd is wordt deze doorgegeven aan de service die de *image recognition* doet. In beide applicaties is enkel deze service verschillend. Wanneer de afbeelding verwerkt is wordt het resultaat op het scherm getoond zoals te zien is in Figuur 34.

Figuur 34 Resultaat van de image recognition

2.3 On-Device

Één van de implementatie methodes van *image recognition* is de uitvoering op het toestel zelf. Eerder werd het gekozen *image recognition* model al besproken. In de volgende paragrafen wordt ingegaan op hoe de *image recognition* op het toestel werd opgezet via Core ML.

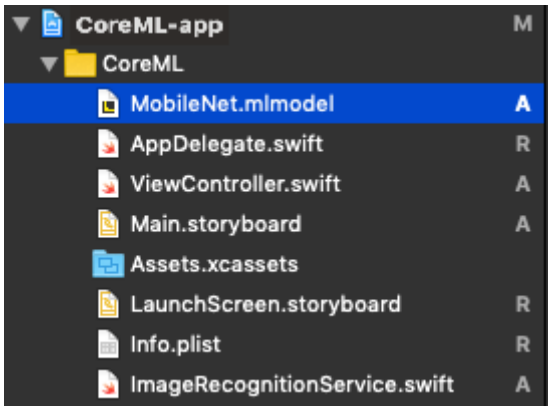
2.3.1 Core ML

Core ML is een *framework* ontwikkeld door Apple dat het toelaat om *machine learning* te integreren in een iOS applicatie. Core ML is geoptimaliseerd om op mobile toestellen te werken. Door deze optimalisatie zal het geheugengebruik en batterijverbruik minimaal zijn. Omdat Core ML volledig op het toestel zelf werkt, kan er verzekerd worden dat de gegevens van de gebruiker veilig zijn. Bovendien betekent dit dat een internetverbinding niet vereist is doordat Core ML ook zonder internet zijn volledige functionaliteit behoudt.

Omdat Core ML werkt met Core ML Models zal het gekozen tensorflow model eerst omgezet moeten worden. Apple heeft hiervoor een Pythonpackage uitgebracht: coremltool. Omdat het MobileNet_V1 model zeer populair is, heeft Apple dit model al omgezet naar een Core ML Model dat aangeboden wordt op de website.

2.3.1.1 Implementatie

Zoals eerder al aangehaald, verschilt de service die de *image recognition* doet in beide implementaties. Om *image recognition* op het toestel te doen wordt het gekozen MobileNet_V1 model naar het Swift project van de applicatie gekopieerd zoals weergegeven in Figuur 35.



Figuur 35 Gekozen model in Swift project

De code in Figuur 36 toont hoe een afbeelding daadwerkelijk herkend wordt. Als eerste wordt de oriëntatie van de afbeelding bepaald. Hierna wordt de afbeelding omgezet van een *UIImage* object naar een *CIImage* object zodat de data van de afbeelding gebruikt kan worden. Daarna wordt er een nieuwe VNImageRequestHandler gemaakt met als parameters het *CIImage* object en de oriëntatie ervan. Als laatste zal deze VNImageRequestHandler de afbeelding gaan classificeren door gebruik te maken van een VNCoreMLRequest.

```
func updateClassifications(for image: UIImage) {

    let orientation = CGImagePropertyOrientation(image.imageOrientation)
    guard let ciImage = CIImage(image: image) else {
        fatalError("Unable to create \(CIImage.self) from \(image).")

    }

    DispatchQueue.global(qos: .userInitiated).async {
        let handler = VNImageRequestHandler(ciImage: ciImage, orientation: orientation)
        do {
            try handler.perform([self.classificationRequest])
        } catch {
            print("Failed to perform classification.\n\(error.localizedDescription)")
        }
    }
}
```

Figuur 36 Herkennen van een afbeelding

Om de *image recognition* tot stand te brengen is er dus een VNCoreMLRequest nodig. In Figuur 37 is de opbouw hiervan weergegeven. Deze VNCoreMLRequest laadt het Core ML-model met de naam 'MobileNet' in. Na het voltooien van de *image recognition* via het ingeladen model, wordt de methode 'processClassification' opgeroepen. Als laatste wordt er ingesteld dat de afbeelding geschaald moet worden naar het 224 x 224 pixels formaat dat het model verwacht.

```
lazy var classificationRequest: VNCoreMLRequest = {
    do {
        let model = try VNCoreMLModel(for: MobileNet().model)

        let request = VNCoreMLRequest(model: model, completionHandler: { [weak self] request, error in
            self?.processClassifications(for: request, error: error)
        })
        request.imageCropAndScaleOption = .centerCrop
        return request
    } catch {
        fatalError("Failed to load Vision ML model: \(error)")
    }
}()
```

Figuur 37 Aanmaken VNCoreRequest

Wanneer de VNCoreMLRequest afgehandeld is, wordt de code in Figuur 38 uitgevoerd. Deze code zal het resultaat weergeven aan de gebruiker door het in het label van het startscherm te zetten.

```
func processClassifications(for request: VNRequest, error: Error?) {
    DispatchQueue.main.async {
        guard let results = request.results else {
            print( "Unable to classify image.\n\(error!.localizedDescription)")
            return
        }
        let classifications = results as! [VNClassificationObservation]

        if classifications.isEmpty {
            print("Nothing recognized.")
        } else {
            let topClassifications = classifications.prefix(2)
            let descriptions = topClassifications.map { classification in
                return String(format: "  (%.5f) %@", classification.confidence, classification.identifier)
            }
            self.labelText!("Classification:\n" + descriptions.joined(separator: "\n"))
        }
    }
}
```

Figuur 38 Weergeven van het resultaat

2.4 API

Naast *image recognition* op het toestel zelf is het ook mogelijk om een API te gebruiken. De volgende paragrafen geven een overzicht van de gebruikte technologieën en de opbouw van deze API.

2.4.1 Keras

Keras is een *open-source neural network library* geschreven in Python. Het is ontwikkeld om modulair, gebruiksvriendelijk en snel te zijn. Keras doet zelf geen berekeningen, maar gebruikt hiervoor een *backend*. Er kunnen verschillende *backends* gebruikt worden zoals Tensorflow Theano of CNTK. Keras zal standaard Tensorflow als backend gebruiken. Keras is dus een *high-level* API die ervoor zorgt dat er op een makkelijke manier met de *low-level* backend gewerkt kan worden. [12]

2.4.2 Flask

Flask is een *micro-web framework* gemaakt in Python dat het mogelijk maakt om op een snelle en makkelijke manier webservices te ontwikkelen.

2.4.3 Applicatie Implementatie

Wanneer een afbeelding geselecteerd is, wordt deze doorgestuurd naar de functie van de *image recognition* service die weergeven is in Figuur 39.

```
func imageRecognitionRequest(selectedImage: UIImage){
    let filename = "selectedImage.jpg"
    let boundary = UUID().uuidString
    let compressedImage = self.compressImage(selectedImage)

    let config = URLSessionConfiguration.default
    let session = URLSession(configuration: config)

    let endpoint: String = "https://10.1.15.50/IndentifyImage"
    guard let endpointURL = URL(string: endpoint) else {
        print("Error: cannot create URL")
        return
    }
    var request = URLRequest(url: endpointURL)
    request.httpMethod = "POST"
    request.setValue("multipart/form-data; boundary=\(boundary)", forHTTPHeaderField: "Content-Type")

    var data = Data()
    data.append("\r\n--\(boundary)\r\n".data(using: .utf8)!)
    data.append("Content-Disposition: form-data; name=\"fileToUpload\"; filename=\"\(filename)\"\r\n".data(using: .utf8)!)
    data.append("Content-Type: image/jpg\r\n\r\n".data(using: .utf8)!)
    data.append(UIImageJPEGRepresentation(compressedImage, 1.0)!)
    data.append("\r\n--\(boundary)--\r\n".data(using: .utf8)!)
    session.uploadTask(with: request, from: data, completionHandler: { responseData, response, error in
        if(error != nil){
            print("\(error!.localizedDescription)")
        }
        guard let responseData = responseData else {
            print("no response data")
            return
        }
        if let responseString = String(data: responseData, encoding: .utf8) {
            self.labelText!(responseData)

        }
    }).resume()
}
```

Figuur 39 Verzenden afbeelding naar API

Deze functie zet de afbeelding om naar een byte string en stuurt deze door naar de API via een *http-request*. Om dit op een zo optimale manier te doen, wordt de gekozen afbeelding binnen de applicatie geschaald. Dit zorgt ervoor dat er minder data verzonden wordt omdat de afbeelding kleiner is. Door de kleinere hoeveelheid data, wordt de afbeelding ook sneller verzonden. De 'compressImage' functie zorgt ervoor dat de afbeelding geschaald wordt naar het gewenste formaat, in dit geval 224 x 224 pixels. In Figuur 40 is te zien hoe dit gebeurt.

```
func compressImage (_ image: UIImage) -> UIImage {
    let rect:CGRect = CGRect(x: 0, y: 0, width: CGFloat(224.0), height: CGFloat(224.0))
    UIGraphicsBeginImageContext(rect.size)
    image.draw(in: rect)
    let img: UIImage = UIGraphicsGetImageFromCurrentImageContext()!
    let imageData:Data = UIImageJPEGRepresentation(img, CGFloat(1.0))!
    UIGraphicsEndImageContext()
    return UIImage(data: imageData)!

}
```

Figuur 40 Schalen afbeelding

2.4.4 API-implementatie

In deze paragraaf wordt uitgelegd hoe de API een afbeelding herkent. De functie in Figuur 41 wordt uitgevoerd wanneer de API opstart. Dit zorgt ervoor dat het model geladen wordt.

```
def load_model():
    app.model = mobilenet.MobileNet()
    app.graph = tf.get_default_graph()
```

Figuur 41 laden model

Omdat het model geladen wordt bij het opstarten van de API zal het geladen model gebruikt worden bij elke *request* die de API krijgt. Wanneer de API gestart is zal het *endpoint* '*IndentifyImage*' beschikbaar zijn. Wanneer er een POST *request* gebeurt op dit *endpoint* zal de API de meegestuurde afbeelding uit de body van de POST *request* halen. Daarna zal de ontvangen afbeelding herkend worden en het resultaat hiervan wordt teruggestuurd, hoe dit gebeurd is weergeven in Figuur 42.

```
@app.route('/IndentifyImage', methods = ['POST'])
def postIndentifyImage():
    imageBytes = request.files['fileToUpload'].read()
    recievedImage = Image.open(BytesIO(imageBytes))
    return jsonify(recognizeImage(recievedImage))
```

Figuur 42 API endpoint

In Figuur 43 is te zien hoe de *image recognition* wordt toegepast. Als eerste zal de afbeelding omgezet worden naar het gewenste formaat. Daarna zal de omgezette afbeelding worden doorgegeven aan het model en zal deze hier een voorspelling op doen. Hierna zal het resultaat vertaald worden naar duidelijke interpreteerbare gegevens.

```
def recognizeImage(recievedImage):
    with app.graph.as_default():
        pImg = preProcessImage(recievedImage)
        prediction = app.model.predict(pImg)
        results = imagenet_utils.decode_predictions(prediction)
        tmpResult = results[0]
        jsonData = [{"prediction": tmpResult[0][1], "predictionPrecentage": str(tmpResult[0][2])}
        ,{"prediction": tmpResult[1][1], "predictionPrecentage":  str(tmpResult[1][2])}]
        return jsonData
```

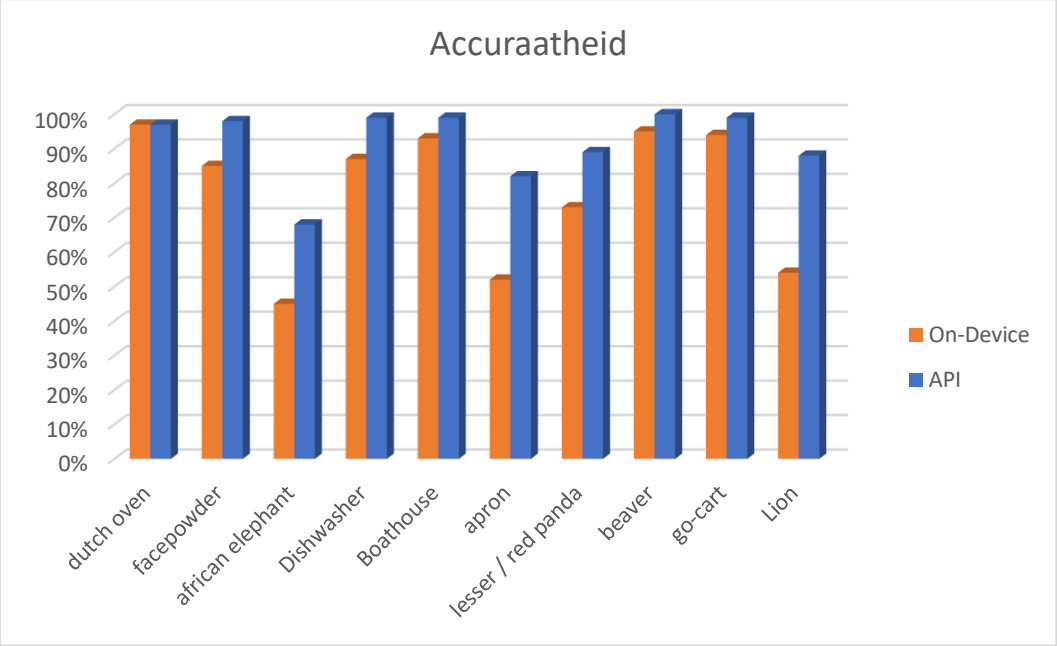
Figuur 43 herkennen afbeelding

2.5 Resultaten

In de volgende paragraaf zijn de resultaten terug te vinden. De applicaties zijn beiden getest op een iPad Pro 9.5 inch 2017 versie daarnaast draait de API op een HP ProBook 6570b.

2.5.1 Precisie

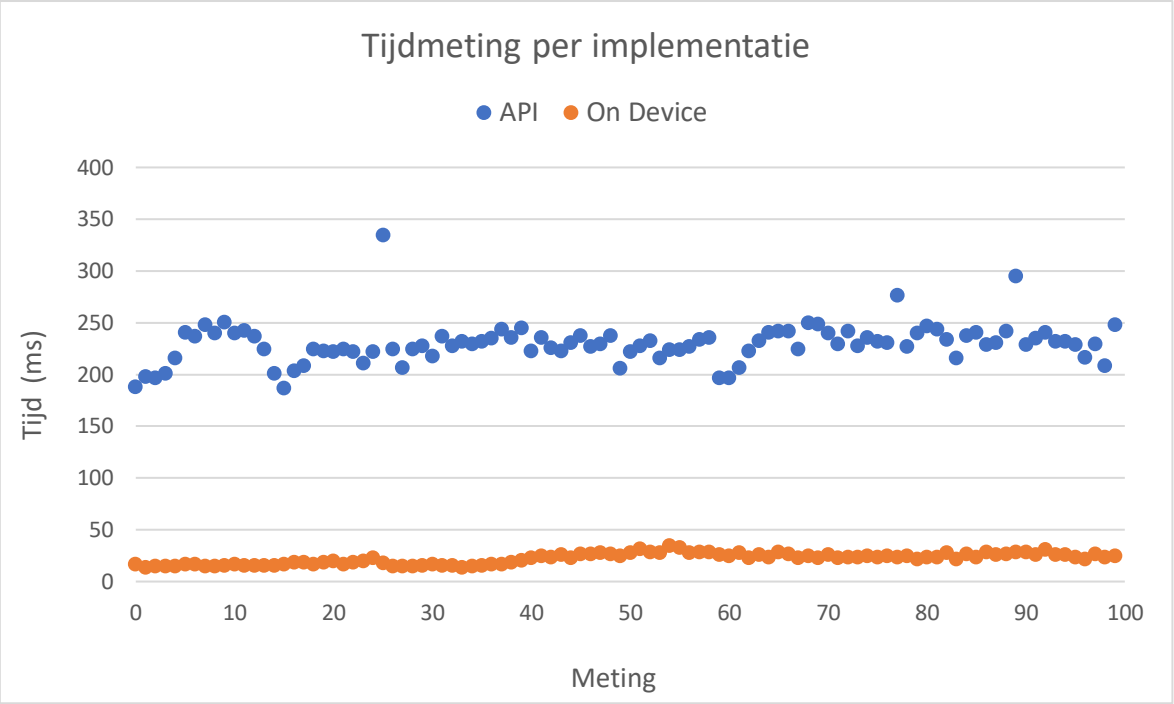
Om de precisie van de implementaties te testen werden tien verschillende afbeeldingen herkend in beide implementaties. In Figuur 44 wordt het percentage zekerheid weergegeven waarmee het model een bepaalde klasse kon herkennen in de afbeeldingen. De klasse die het model herkende waren altijd correct, maar de zekerheid verschilde per afbeelding. De *on-device* versie van het model bleek minder accuraat te kunnen achterhalen over welke klasse het ging. Dit wordt weergegeven door het lagere percentage van de blauwe staafjes. Dus ondanks dat beiden implementatiemethodes de afbeeldingen konden herkennen, presteerde de API-implementatie beter.



Figuur 44 Grafiek Accuraatheid *image recognition*

2.5.2 Tijd

Om de snelheid van de *image recognition* voor beiden implementaties te testen werd één afbeelding honderd keer herkend in beide implementaties en werd de tijd die nodig was voor de herkenning gemeten. In onderstaande Figuur 45 wordt de tijd per meting weergegeven voor de twee implementaties. Deze ligt voor de API-implementatie een stuk hoger dan die van de *on-device* implementatie. Dit betekent dus dat de *on-device* implementatie sneller aan *image recognition* doet. De verklaring hiervoor is simpel: Core ML is geoptimaliseerd om op het toestel zelf te werken en afbeeldingen hoeven dus niet via het internet verzonden te worden waardoor er geen afhankelijkheid is van de internetsnelheid zoals bij de API. De pieken in tijd voor de API zijn daarom te wijten aan een toevallige vertraging in internetsnelheid.

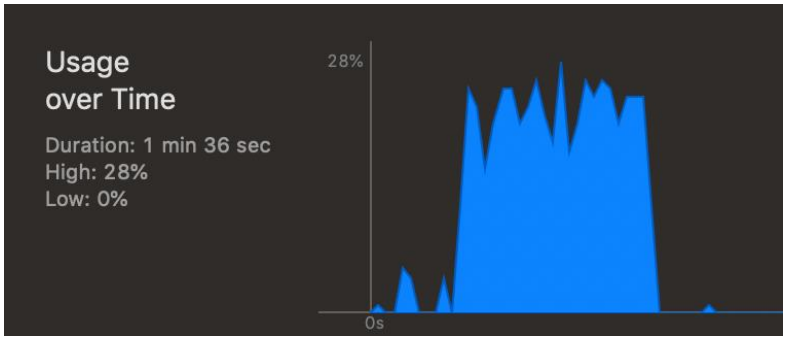


Figuur 45 snelheidsmeting

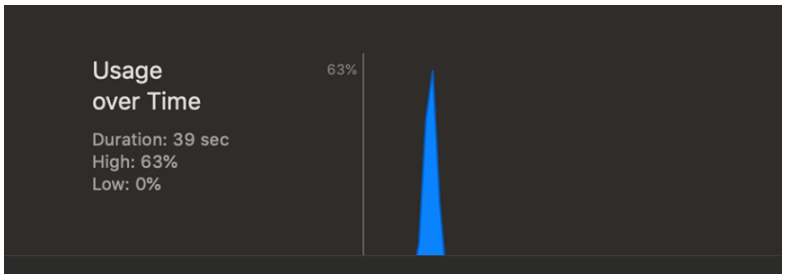
2.5.3 CPU-, geheugen-, energie- en internetgebruik

De volgende vier parameters werden gemeten door gebruik te maken van Xcode Instuments. Dit is een preformance analyse en testing tool. Metingen van CPU-, geheugen-, energie en internetgebruik werden uitgevoerd tijdens het herhaalddelijk uitvoeren van image recognition op één zelfde afbeelding.

Het *central processing unit* (CPU) gebruik wordt door Xcode uitgedrukt in %, maar deze zal niet stoppen bij 100% doordat er rekening gehouden wordt met het aantal processorkernen. In dit onderzoek is een iPad Pro 9.5 inch 2017 versie gebruikt. Deze heeft zes processorkernen en zal dus tot 600% CPU gebruik kunnen gaan. In Figuur 46 is te zien dat de applicatie met API-implementatie constant rond de 28% CPU verbruik zat tijdens *image recognition*. In Figuur 47 is het CPU verbruik van de *on-device* implementatie te zien. Wat hierbij opvalt, is dat bij het opstarten de applicatie tijdelijk veel CPU gebruikt maar het gebruik al snel naar 0% zakt. Dit komt doordat het model enkel ingeladen moet worden waarna de *image recognition* op de GPU (*Grafics processing unit*) van het toestel gebeurt waardoor de CPU minder belast wordt.



Figuur 46 CPU-gebruik API-applicatie



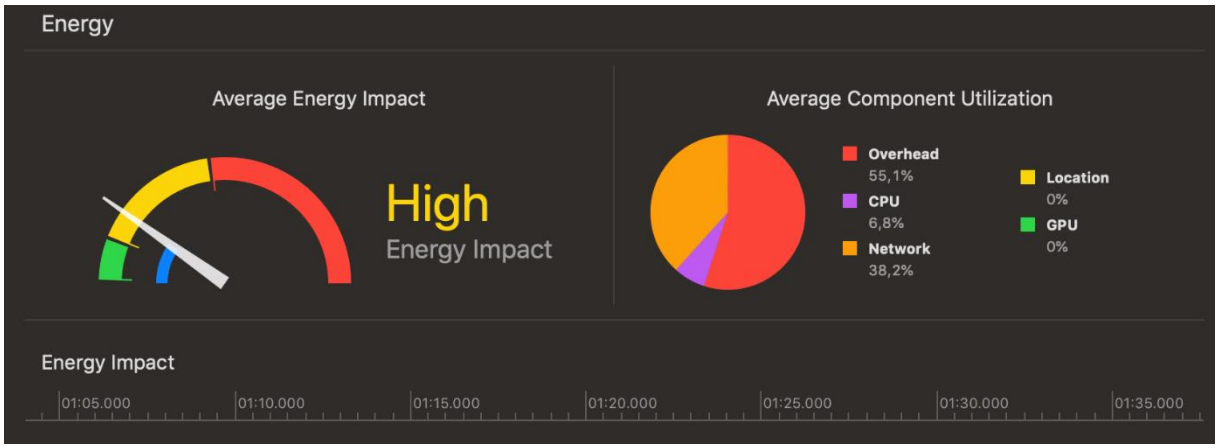
Figuur 47 CPU gebruik On-Device applicatie

Als tweede werd het geheugen gebruik van beiden inplementaties gemeten tijdens *image recognition*. In Figuur 48 is het geheugenverbruik van de API-applicatie te zien en in Figuur 49 die van de on device. Hierin is te zien dat het geheugengebruik heel stabiel is voor beiden implementaties. Bij de on device applicatie is er enkel een piek bij het instantiëren van de *image recognition service*. Het geheugengebruik in beide applicaties is laag, namelijk 35.7 MB voor de API-applicatie en 22.6 MB voor de on device applicatie. Het geheugengebruik van de *on-device* applicatie ligt dus lager dan die van de API.

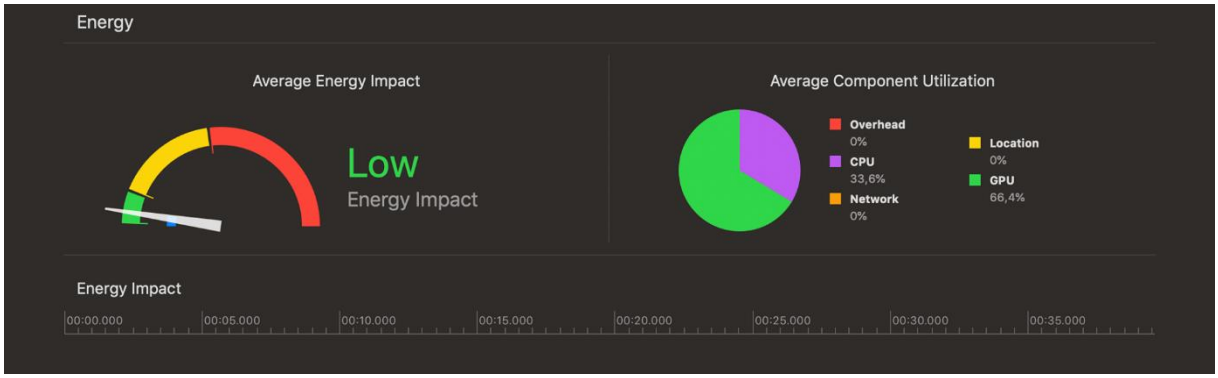
Figuur 48 Geheugen gebruik API-applicatie

Figuur 49 Geheugen gebruik On-Device applicatie

Het energieverbruik van applicaties werd als derde gemeten en wordt voor de API weergegeven in Figuur 50. Hieruit blijkt dat de CPU voor enkel 6.6% van het energie verbruik verantwoordelijk is. Het netwerk gebruik daarentegen is verantwoordelijk voor 38.2% en de *overhead* van het netwerkgebruik voor maar liefst 55.1%. De API-applicatie heeft een grote impact op het energie verbruik. In Figuur 51 wordt het energie verbruik van de on device applicatie weergegeven. De applicatie spendeert 66.4% van zijn energie aan de GPU. Dit komt doordat de berekeningen die Core ML uitvoert, worden berekend op de GPU. Dit wordt dus benadruk door het energieverbruik. Over het algemeen heeft de on device applicatie een lager energieverbruik dan de API-applicatie.

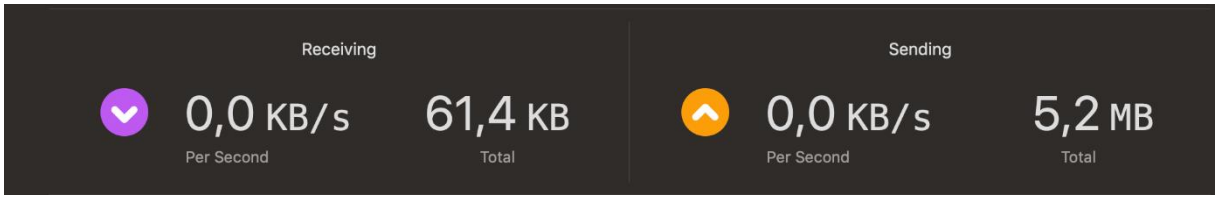


Figuur 50 Energieverbruik API applicattie



Figuur 51 Energieverbruik On-device applicatie

Als laatste wordt in Figuur 52 het netwerkverbruik weergegeven van de API-applicatie. De figuur toont aan dat er 5.2 MB aan gegevens verstuurd zijn. Omdat er honderd afbeeldingen achter elkaar werden verstuurd komt dit neer op 52 KB per afbeelding. Ook werd er 61.4 KB aan data ontvangen. Dit is de informatie over de voorspellingen van de API. Zoals te zien is in Figuur 53 heeft de on-device applicatie geen data verzonden of ontvangen. Dit bevestigt dus dat Core ML geen internettoegang nodig heeft om te functioneren.



Figuur 52 Internetverbruik API-applicatie

Figuur 53 Internetverbruik On-device applicatie

Conclusie

Uit de testen is gebleken dat Core ML inderdaad zeer performant is en de impact op de systeem *resources* heel miniem is. Ook hebben de testen bevestigd dat Core ML geen internettoegang nodig heeft om te functioneren. Na het bekijken van de accuraatheid van de twee applicaties, bleek Core ML toch minder goed te score. De voornaamste reden hiervoor is dat het Tensorflow model omgezet is naar een Core ML-model. Dit model is geoptimaliseerd om snel en performant te werken maar er is een verlies in de accuraatheid van het model ten opzichte van het origineel.

Zoals de resultaten hebben aangetoond, hebben beide implementaties hun voor- en nadelen. Een overzicht van de resultaten worden nog eens weergegeven in Tabel 1. De API heeft als voordeel dat het meer precisie heeft, maar iets meer systeem *resources* gebruikt en een hoger energieverbruik heeft. Core ML daarentegen heeft het voordeel dat het sneller is, minder systeem *resources* gebruikt, minder energieverbruik heeft en geen internetverbinding nodig heeft. Daartegenover staat dan wel dat deze implementaties minder precies is.

Uiteindelijk moet er ook rekening gehouden worden met de gebruiksvriendelijkheid van de implementatie methode. API's kunnen bijvoorbeeld voor alle besturingssystemen gebruikt worden terwijl *on-device* implementaties voor ieder besturingssysteem individueel ontwikkeld moeten worden. Bovendien kan het *image recognition* model van een API gemakkelijk veranderd worden terwijl voor de *on-device* versie de hele applicatie bijgewerkt zal moeten worden. Er kan dus geen éénzijdige conclusie gevormd worden. De beste implementatie zal dus afhangen van de noden van de applicatie.

Tabel 1 Overzicht onderzoekresultaten

	On-Device	API
Precisie	Goed	Beste
Snelheid	25 ms	200 ms
Performantie	Zeer goed	Goed
Energieverbruik	Laag	Hoog
Functioneert zonder internet	✓	✗

Zelfreflectie

Tijdens de stage ben ik in aanraking gekomen met veel nieuwe technologieën zoals Swift, AR en *image recognition*. Deze heb ik leren verwerken in een mobiele applicatie waardoor ik een voorsprong heb in de toekomst. Door de uitstekende communicatie met Craftworkz kregen we tijdig nuttige feedback over onze applicatie die we meteen konden toepassen. Zo leverde we elke keer weer een beter presterend product op met het oog op de gebruiksvriendelijk. Tijdens het ontwikkelen van de applicatie zijn we geen grote obstakels tegen gekomen. Er was enkel een gebrek aan gratis bestaande 3D modellen van dierentuinen voor in de AR-omgeving waardoor we besloten deze zelf in elkaar te zetten. Achteraf gezien was onze keuze voor ARkit misschien niet de beste omdat het niet genoeg mogelijkheden biedt omtrent animaties. Met een ander *framework* hadden we wellicht meer uit de AR-omgeving kunnen halen. Persoonlijk ben ik trots op de manier hoe ik deze stageopdracht heb aangepakt. Ik heb veel initiatief getoond op het gebied van communicatie met mijn stagepartner en heb mij aangepast aan zijn stageschema. Zo konden we samen al een eerste concept bedenken voor onze applicatie en een planning opstellen. Ondanks dat mijn stage twee weken eerder van start ging, ben ik direct zelfstandig aan het werk gegaan en heb ik mezelf leren werken met een aantal van de eerder vernoemde technologieën. Hierbij heb ik mijn stage partner dagelijks geüpdatet met de stand van zaken. Op het gebied van de opdracht zou het in de toekomst beter zijn om direct meer rekening te houden met de gebruikerservaring van de applicatie. Door de grote tevredenheid van het stagebedrijf lijken er dus weinig werkpunten te zijn voor mij en mijn stagepartner. Deze tevredenheid was natuurlijk grotendeels te danken aan de grote vrijheid die we kregen in het ontwikkelen van de applicatie en het gebrek aan specifieke eisen. Craftworkz wilde verrast worden door een creatieve demo en dat is ons zeker gelukt.

Bibliografie

[1] „Cronos-groep,” 2019. [Online]. Available: https://cronos-groep.be. [Geopend 2019 05 24].

[2] „ventures,” 2019. [Online]. Available: https://www.raccoons.be/ventures. [Geopend 15 4 2019].

[3] „SFSpeechRecognizer,” Apple, 2019. [Online]. Available: https://developer.apple.com/documentation/speech/sfspeechrecognizer. [Geopend 13 4 2019].

[4] M. Vandendriesche, „Intenties,” Oswald, 2 2019. [Online]. Available: https://docs.oswald.ai/bouwen/trainen-van-de-nlp. [Geopend 5 4 2019].

[5] M. Vandendriesche, „entiteiten,” Oswald, 2 2019. [Online]. Available: https://docs.oswald.ai/bouwen/entiteiten. [Geopend 4 4 2019].

[6] M. Vandendriesche, „Inleiding,” Oswald, 2 2019. [Online]. Available: https://docs.oswald.ai/scenarios/inleiding. [Geopend 4 4 2019].

[7] „AVSpeechsynthesizer,” Apple, 2019. [Online]. Available: https://developer.apple.com/documentation/avfoundation/avspeechsynthesizer. [Geopend 1 5 2019].

[8] „HitTest,” Apple, 2019. [Online]. Available: https://developer.apple.com/documentation/uikit/uiview/1622469-hittest. [Geopend 23 4 2019].

[9] S. GUPTA, „Understanding Image Recognition and Its Uses,” 28 9 2018. [Online]. Available: https://www.einfochips.com/blog/understanding-image-recognition-and-its-uses/. [Geopend 17 3 2019].

[10] missinglink, „neural networks image recognition methods best practices applications,” missinglink, [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/neural-networks-image-recognition-methods-best-practices-applications/. [Geopend 25 04 2019].

[11] A. Bonner, „wtf-is-image-classification,” towardsdatascience, 2 feb 2019. [Online]. Available: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb. [Geopend 25 4 2019].

[12] M. H. Hassoun, Fundamentals of artificial neural networks, Massachusetts Institute of Technology: Mit Press, 1995.

[13] „Home,” Keras, 2019. [Online]. Available: https://keras.io/. [Geopend 1 5 2019].

[14] Maruti techlabs, „https://www.marutitech.com," Maruti techlabs, 2018. [Online]. Available: https://www.marutitech.com/working-image-recognition/. [Geopend 03 03 2019].

[15] ujjwalkarn, „An Intuitive Explanation of Convolutional Neural Networks," 11 08 2019. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Geopend 10 03 2019].

[16] S. Ravindra, „How Convolutional Neural Networks Accomplish Image Recognition?," 2017. [Online]. Available: https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html. [Geopend 10 03 2019].

[17] „tensorflow," 2019. [Online]. Available: https://www.tensorflow.org/. [Geopend 15 05 2019].