EyePark documentatie

Click here to add Subtitle  
Type a space if not applicable

Document owner: Robin De Bie/Wouter Vanloocke

Information security classification: Internal Use Only

Inhoudsopgave

[1 Voorwoord 3](#_Toc18677243)

[2 Camera’s 3](#_Toc18677244)

[3 Beeldherkenning 3](#_Toc18677245)

[3.1 Custom Vision AI 3](#_Toc18677246)

[3.1.1 Gebruik SDK 3](#_Toc18677247)

[3.1.1.1 Uploaden foto’s 4](#_Toc18677248)

[3.2 Onnx 4](#_Toc18677249)

[3.2.1 Toepassing code 5](#_Toc18677250)

[3.3 Handleiding 6](#_Toc18677251)

[4 Webtaken 8](#_Toc18677252)

[4.1 Analyse 8](#_Toc18677253)

[4.2 Herkenning 8](#_Toc18677254)

[5 Conclusie 9](#_Toc18677255)

# Voorwoord

Dit document is een uitbreiding op ons stagedossier. In dit document worden enkel de nieuwe zaken besproken. Alles over de websitecode, behalve de webjobs, is te vinden in het stagedossier. Al onze code in te vinden in onze GitHub repository: <https://github.com/WouterVanloocke/EyePark>.

# Camera’s

Tijdens de stage hadden we het probleem dat onze camera’s vaak foto’s verstuurden die corrupt waren. Dit kwam door een probleem in de samenwerking tussen de camera’s en de ftp server. Wanneer we minder frequent een foto doorsturen, d.w.z. niet elke seconde, hebben we dit probleem niet.

Daarnaast is de hoek en de hoogte zeer belangrijk bij de plaatsing van de camera. Wanneer de camera recht kan kijken op een bepaalde parking, zal de herkenning van de auto’ s veel groter zijn dan wanneer je de camera auto’s laat herkennen vanuit een bepaalde hoek. Ook de beeldkwaliteit is van belang. We raden beelden van 1080p aan.

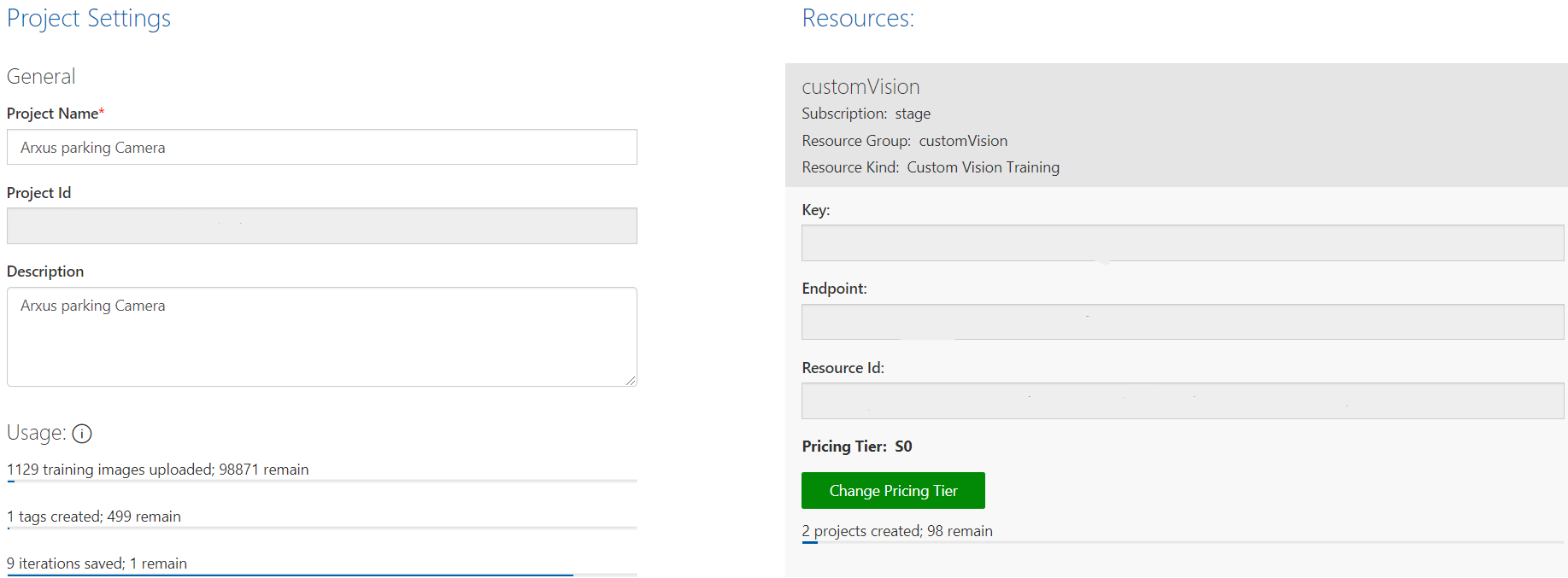
# Beeldherkenning

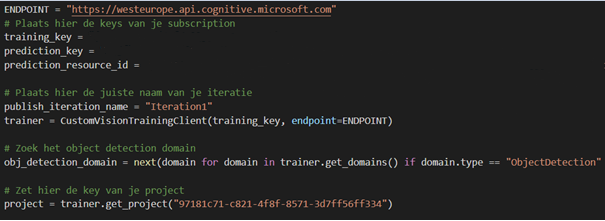
## Custom Vision AI

Als beeldherkenningsmodel gebruiken we de Custom Vision AI die Azure aanbiedt. We hebben dit model zelf getraind omdat de modellen die op het internet te vinden zijn niet volledig voldoen aan de eisen die we stellen voor ons project. Er zitten reeds meer dan 1000 foto’s in ons model. Dit werkt redelijk goed, maar dit zou nog veel beter werken mochten er nog meer foto’s in het model zitten.

Om te weten hoe je een foto manueel toevoegt en tagt, verwijzen we graag naar ons stagedossier waar dit allemaal duidelijk uitgelegd staat.

### Gebruik SDK

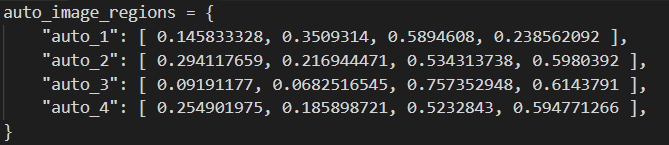
Ons project is gemaakt in Python vandaar dat we gebruik maken van de informatie die reeds ter beschikking is op de Azure website omtrent object detection. Eerst en vooral moet je je project kunnen aanspreken vandaar dat we eerst de credentials van ons project definiëren in het begin van het script. De credentials zijn te vinden bij de instellingen van je project in het online portaal van Custom Vision. 

De documentatie toont hoe je je eigen project kan maken en hoe je tags aanmaakt vanuit een python script maar aangezien we dit al manueel hebben gedaan slaan we dit stuk in documentatie over en gaan we rechtstreeks naar het uploaden van de foto’s.

Om foto’s te kunnen uploaden moet je de coördinaten weten van de rechthoeken die getrokken zijn rond de herkende objecten, in ons geval de auto’s. Aangezien we dit buiten Custom Vision doen, moeten we dus beroep doen op een ander herkenningsalgoritme om de foto’s te analyseren. Hierna moeten de foto’s nog geüpload worden naar Custom Vision om ze daar te trainen tot een beter model.

#### Uploaden foto’s

Custom Vision verwacht dat de coördinaten van de foto’s volgens een bepaald formaat binnen komen:

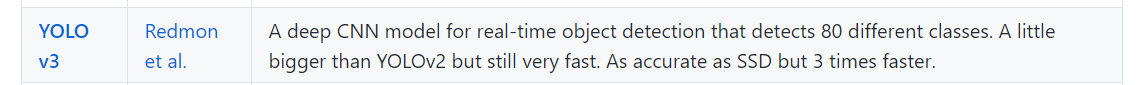


We zullen onze herkenningscode moeten omvormen zodat deze doorgestuurd kan worden in dit formaat.

## Onnx

We maken gebruik van het yolov3 model in samenwerking met onnx. Waar dat yolov3 het effectieve model is, is onnx het zogezegde framework waar het model op draait. Onnx zorgt ervoor dat verschillende modellen gemakkelijk tussen bepaalde tools geswitcht kunnen worden.

Op de Onnx website kan je op Model Zoo klikken, hierna krijg je een pagina vol verschillende modellen die je kan gebruiken voor verschillende doeleinden. Wij maken gebruik van het yolov3 model omdat dit het beste is voor object detection. Er zijn namelijk 80 verschillende objecten getraind binnen het model waar auto’s er één van is.



We volgen een GitHub tutorial die Python combineert met het yolov3 model om objecten te herkennen. Deze tutorial zal als basis dienen voor ons project. De tutorial is te vinden op: <https://github.com/ultralytics/yolov3>

Als je de link volgt zal er uitgelegd worden wat je moet doen en installeren om een werkend herkenningsscript te bekomen. In principe hoef je requirements.txt niet meer te installeren als je packages.sh hebt geïnstalleerd vanuit ons project. Meeste van deze dependencies komen overeen.

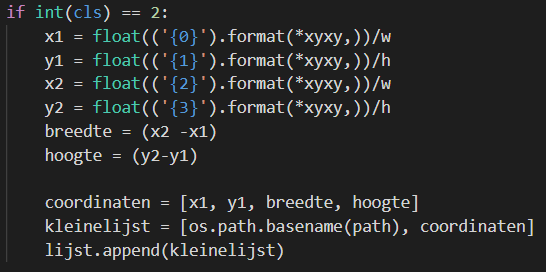
### Toepassing code

We tonen enkel de code die wij hebben toegevoegd of die wij hebben aangepast ten opzichte van de tutorial uit het vorige puntje.

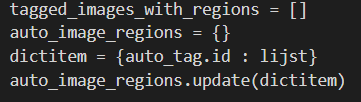
Aangezien we manueel al een tag hebben aangemaakt moeten we de id te weten komen van deze tag:

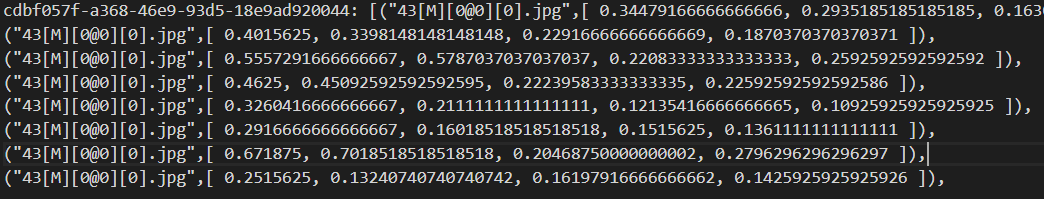


Als de herkenningsklasse van een bepaald herkend object overeenkomt met 2 (de klasse auto, te vinden in yolov3-master/data/coco.names) ga dan pas verder met de code. Slaag de x en y coördinaat op van de linkerbovenhoek en bereken daarna de hoogte en de breedte van de rechthoek die rond het object is getekend door yolov3. Dit zijn de waarden die Custom Vision wil krijgen voor het tekenen van de rechthoeken op de foto’s. Deze waarden slagen we op in een lijst samen met de naam van de foto.



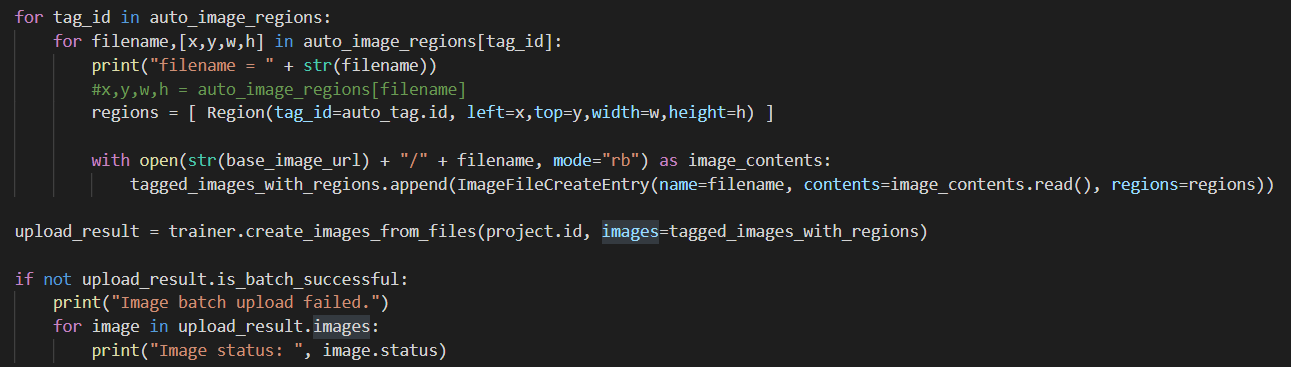
Hierna combineren we de opgehaalde tag id met de coördinaten en de naam van de foto om aan de gevraagde syntax van Custom Vision te voldoen.



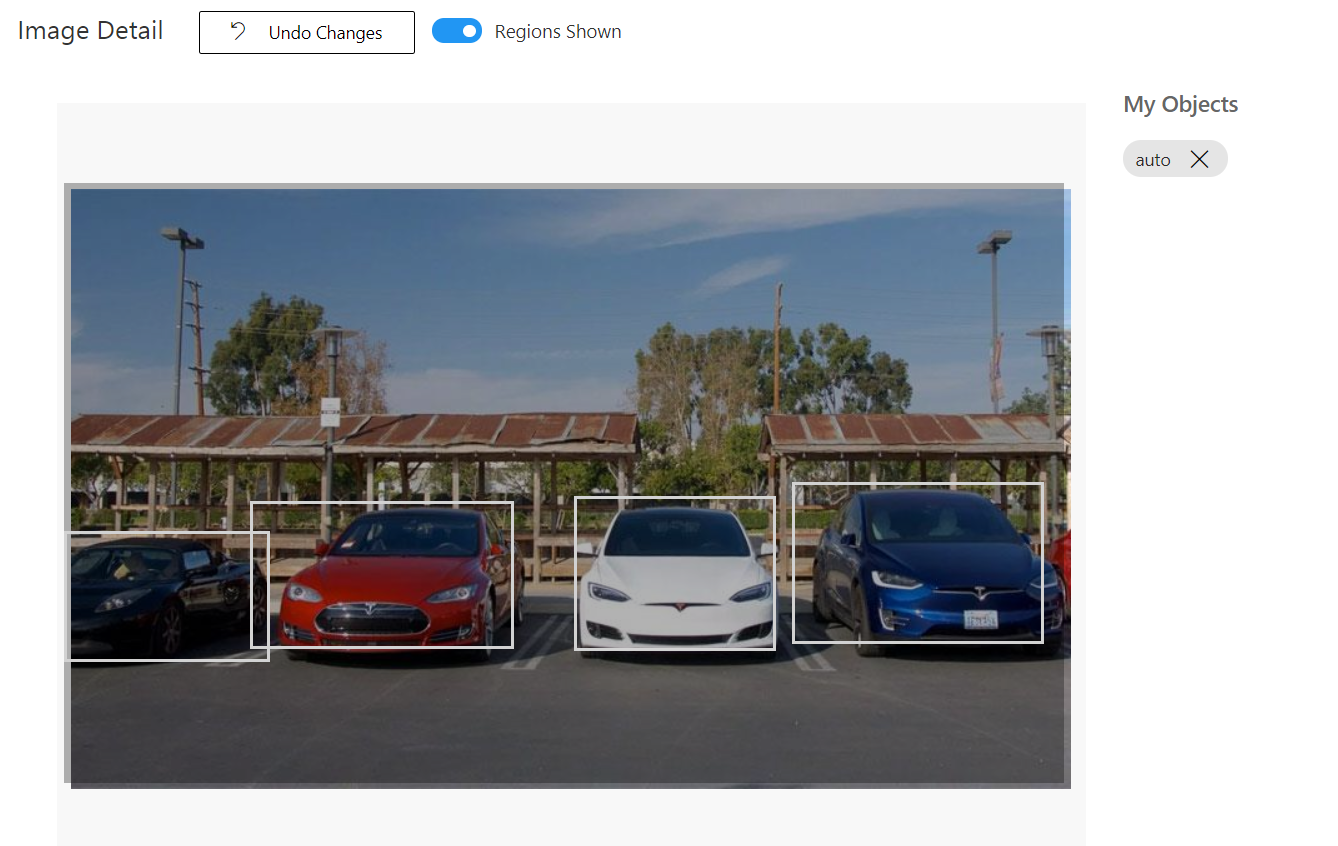
Als je de code dan runt krijg je dit te zien als output voor het uploaden van images: 

Hierna moeten we nog een stukje code toevoegen van de quickstart tutorial van de Azure website.

Deze code gaat de foto’s samen met de coördinaten uploaden naar het Custom Vision project:



## Handleiding

* Je cloned de repo van dit project van github.com: https://github.com/WouterVanloocke/EyePark.git
* Je plaatst alle foto’s die je wilt laten herkennen in de map: yolov3-master\data\samples. De foto’s mogen niet groter zijn dan 4MB
* Je roept het script op vanuit yolov3-master\ met het commando :
  + Linux: python3 detect.py --cfg cfg/yolov3.cfg --weights weights/yolov3.weights
  + Windows: python detect.py --cfg cfg/yolov3.cfg --weights weights/yolov3.weights
* Hierna kan je een kijkje nemen in yolov3-master\output. Hierin staan de foto’s met de rechthoeken aan de hand van het yolov3 model:
* Wanneer je gaat kijken in het portaal van Custom Vision, dan zie je dat deze foto er ook staat met de rechthoeken rond de auto’s: 
* Je kan maximaal 64 foto’s per keer laten herkennen dus als je meer foto’s hebt dan zal je het script meerdere malen moeten uitvoeren.

# Webtaken

In ons stagedossier vind je meer informatie over de werking van onze webtaken en hoe je een webtaak moet aanmaken.

## Analyse

Net zoals tijdens de stage hebben we nog steeds een webtaak die ieder uur het totaal aantal vrije plaatsen gaat opslaan in de databank. Deze data wordt weergegeven op de landing page van de website.

## Herkenning

We gebruiken ook nog ons herkenningsscript dat we tijdens de stage hebben gemaakt, maar deze periode hebben aangepast. We hebben de tijd die nodig is om een foto te analyseren om nieuwe parkeerplaatsen te herkennen teruggebracht van 5 minuten naar 20 seconden. Hierdoor hebben we ook een functie kunnen maken waardoor de kans kleiner is geworden dat er foute of dubbele waarden worden toegevoegd aan de databank.

De werkwijze van het toevoegen van parkeerplaatsen is nog steeds dezelfde. Wanneer een auto 2 foto’s na elkaar op dezelfde plaats staat zullen de coördinaten van deze locatie worden opgeslagen in de databank. Het maximum aantal plaatsen op een parking wordt nu dynamisch bepaald. De administrator kan in de databank een waarde aanpassen waardoor het script stopt met het proberen toevoegen van parkeerplaatsen voor een bepaalde parking.

De laatst genomen foto van elke camera wordt op de website geplaatst en met rode en groene bolletjes worden de bezette en vrije plaatsen weergegeven.

Wanneer een camera meer dan één uur geen foto verstuurd heeft zal het script de statuskolom van de bijbehorende parking aanpassen in de databank om te laten weten dat er iets mis is met deze camera.

# Conclusie

De hoek waarmee de camera naar de parking kijkt is zeer belangrijk! Wanneer je zo kostenefficiënt mogelijk te werk wil gaan wil je zoveel mogelijk met grote hoeken werken, maar dit vormt vaak een probleem voor de herkenning. We hebben verschillende herkenningsmodellen gebruikt en hebben hieruit geconcludeerd dat zelfs de beste herkenningsmodellen auto’s niet kunnen detecteren in bepaalde hoeken. Je wil dus een zo recht mogelijke hoek hebben ten opzichte van de auto’s die je wil detecteren. Dit zal er voor zorgen je meer camera’s nodig hebt, wat resulteert in een hogere kost.

Wat ook nog nodig zal zijn is dat ons model verder getraind moet worden met foto’s van veel verschillende parkings vanuit verschillende hoeken om er zo voor te zorgen dat het model uiteindelijk slim genoeg is om alle auto’s te herkennen ongeacht van de hoek.

Zolang ons model nog niet volledig op punt staat kan je ook gebruik maken van andere modellen die op het internet te vinden zijn, zoals Mask-RCNN of yolov3. Deze modellen werken ook niet goed in bepaalde hoeken, maar als beginfase van een nieuwe parking kan je beter een van deze modellen gebruiken omdat deze universeler getraind zijn op auto’s op verschillende parkings dan ons eigen Custom Vision model. Een andere mogelijkheid is het toevoegen van foto’s aan een reeds bestaand model en daarna het model opnieuw trainen. Maar dit werkt niet zo intuïtief als bij Custom Vision.

De initiële kost van het project zal vrij hoog zijn omdat je camera’s zal moeten aankopen(zie excel document met prijzen van camera’s en aanbevelingen). In het begin, wanneer je je project opstart, zal je je model redelijk vaak moeten trainen. Dit kost ongeveer €16 per uur en wanneer je ongeveer 1000 foto’s hebt, duurt het trainen ongeveer 2 uur(advanced training optie binnen Custom Vision). Wanneer je meer foto’s hebt zal het trainen ook langer duren, al kan je dit zelf instellen hoelang je je model wil laten trainen, maar hoe langer je traint, hoe beter je model auto’s zal kunnen herkennen.

De operationele kost zal niet erg hoog liggen, wanneer je niet meer moet trainen valt deze kost volledig weg. Er is wel een kost per duizend “transacties”. Hiermee bedoelen ze het analyseren van een foto. Deze kost bedraagt ongeveer anderhalve euro per duizend transacties. Wanneer je met 30 camera’s werkt en elke 30 seconden een foto laat analyseren per camera dan komt dit neer op een kost van 65 euro per dag. En er is ook een maandelijkse kost voor de opslag van de afbeeldingen. Per 1000 foto’s bedraagt dit ongeveer 50 cent per maand.

We hebben nog niet voldoende kunnen testen met de plaatsing in de hoogte van de camera’s. Dus we kunnen niet met zekerheid zeggen hoe de camera zal omgaan met de herkenning als we de camera hoger zouden hangen. Maar we veronderstellen dat de hoogte zeker een factor is waar rekening mee gehouden moet worden.