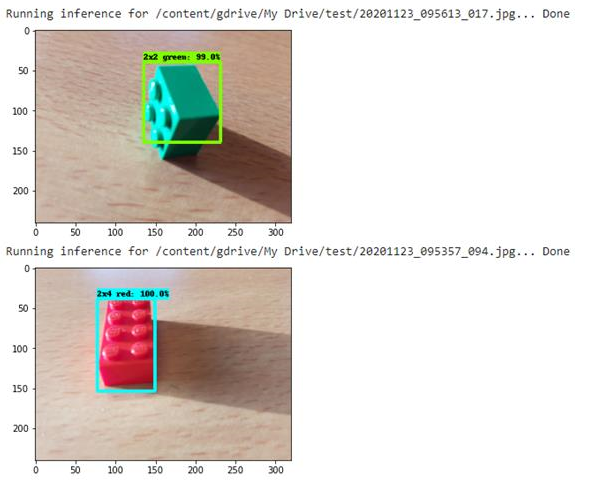
**Logboek Configureren**

datum



Docent:

Daniel Meinsma

Studenten:

Danique van Winden(19070136)

Olivia Boontjes(19053274)

Tom Dullaart(19079435)

Inhoud

[Inleiding 3](#_Toc61875684)

[Hoofdstuk 1: OpenCV 4](#_Toc61875685)

[Bevindingen OpenCV: 4](#_Toc61875686)

[Hoofdstuk 2: Classification 5](#_Toc61875687)

[Dataset van internet 5](#_Toc61875688)

[Testen met de dataset 6](#_Toc61875689)

[Een nieuwe dataset van 3D modellen 9](#_Toc61875690)

[Omzetten naar Zwart-Wit 10](#_Toc61875691)

[Template app gebruiken met ons model 10](#_Toc61875692)

[Eigen dataset maken 11](#_Toc61875693)

[Goed resultaat eigen dataset 11](#_Toc61875694)

[Gesprek met maker van LEGO sorteermachine – Daniel West 12](#_Toc61875695)

[Hoofdstuk 3: Object detection 13](#_Toc61875696)

[Van classification naar object detection 13](#_Toc61875697)

[Blokjes aangeven in de app 14](#_Toc61875698)

[Vooruitgang met object detection 15](#_Toc61875699)

[Poging object detection 16](#_Toc61875700)

[Werkende object detection 17](#_Toc61875701)

[Converteren naar TFLite 18](#_Toc61875702)

[Ander model converteren 19](#_Toc61875703)

[Meerdere blokjes herkennen 20](#_Toc61875704)

[Toevoegen meer classes aan dataset. 22](#_Toc61875705)

[Hoofdstuk 4: De toekomst 23](#_Toc61875706)

[Minimal viable product 23](#_Toc61875707)

[Toekomstige features 23](#_Toc61875708)

[Literatuurlijst 25](#_Toc61875709)

# Inleiding

De challenge van het LEGO-team is om een app te maken die gebruikt kan worden om een LEGO blokje te vinden in een berg van blokjes. Voor het bouwen van deze app hebben we verschillende routes overwogen. Als eerste hebben we verkend of we OpenCV konden gebruiken en wat daar de sterke kanten van zijn. We zijn echter vrij snel tot de conclusie gekomen dat we voor onze doeleinden gebruik moeten maken van machine learning. In dit logboek geven wij inzicht in de stappen die we gemaakt hebben en valkuilen die we tegen zijn gekomen.

Dit logboek bestaat uit vier hoofdstukken. In ons eerste hoofdstuk vatten we kort onze bevindingen van OpenCV samen. Daarna zijn we verdergegaan met tensorflow. Als tussenstap hebben we een classification model gemaakt en in een app verwerkt en dat proces uitgebreid beschreven in hoofdstuk 2. In hoofdstuk 3 zijn we verder gegaan met tensorflow, maar dan met een object detection model. In het vierde en laatste hoofdstuk hebben we beschreven wat er in de toekomst nog gedaan dient te worden aan dit project.

# Hoofdstuk 1: OpenCV

## Bevindingen OpenCV:

Voordat we aan de slag zijn gegaan met tensorflow en machine learning hebben geëxperimenteerd met OpenCV.

Eerste stap: Vind bepaalde kleurgroepen in een afbeelding.

OpenCV maakt gebruik van hsv kleurcodes, dus wij gaven een minimum en maximumkleurcode. Het grootste veld op het scherm waar pixels binnen dit minimum en maximum in voorkwamen werd gemarkeerd.

Tweede stap: herken vormen in het beeld.

In onze tests gebeurt er te veel in de achtergrond en dat zorgt ervoor dat er niet gericht naar een vorm gekeken kan worden. De lijnen worden immers vaak afgesneden en er zijn meestal meerdere vormen in beeld die te herkennen zijn. Een oplossing die wij hiervoor gevonden hadden was om een 'mask' te gebruiken. Hiermee zorgden we ervoor dat er alleen naar de pixels binnen bepaalde hsv waardes gekeken werd. Op deze manier lukte het ons wel om bepaalde vormen te herkennen, maar een groot nadeel is dat we hiervoor van vooraf bepaalde kleuren afhankelijk zijn en op een ander type camera of met andere belichting kunnen deze kleuren afwijken. Ook hebben we het niet goed voor elkaar gekregen om 3-dimensionale vormen te herkennen. Wellicht is hier door middel van thresholding wel een oplossing voor te vinden, maar zelfs dan zouden details als de punten boven op de blokjes heel lastig te zien zijn. Vandaar dat onze conclusie was dat het ons te veel tijd zal kosten om met OpenCV verder te gaan.

# Hoofdstuk 2: Classification

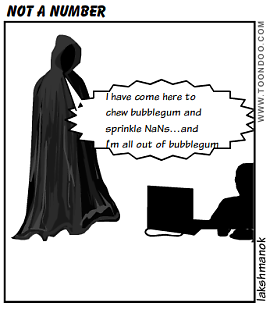
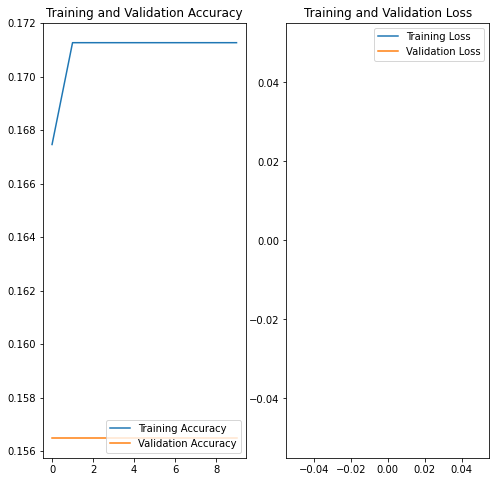
## Dataset van internet

16-11-2020

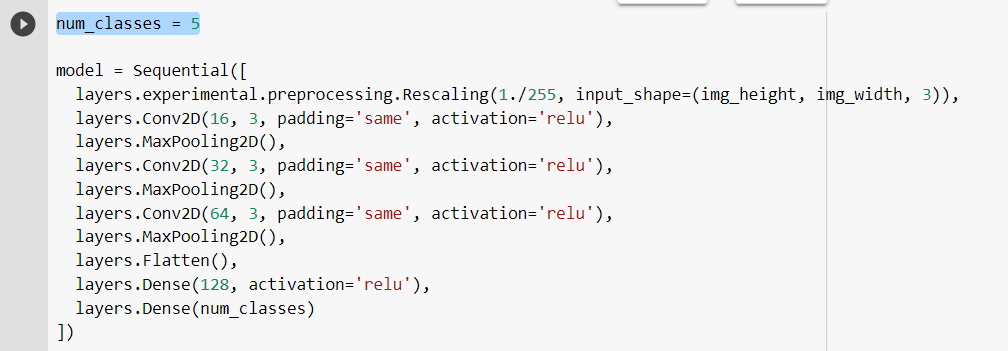
**Doel:** We wilden graag een dataset toevoegen aan een model die we al hebben gevonden.

We zijn begonnen met het zoeken van een dataset. Op kaggle vonden we een dataset die er veelbelovend uitzag. Het bestaat uit 20 verschillende vormen LEGO-blokjes. De verschillende vormen zijn opgedeeld in mappen. Er zijn twee grote mappen, een met de base images en een met de cropped images. We hebben besloten om de cropped images te gebruiken omdat we dan minder loze pixels hebben. De blokjes hebben wel allemaal verschillende kleuren. We hebben in overleg met onze opdrachtgever besloten om de prioriteit op vorm te leggen en pas later op de kleur te focussen.

We hebben de nieuwe data aan een bestaand image classification model toegevoegd, maar we kregen bij loss ‘nan’.[[1]](#endnote-2) Na we hebben er onderzoek naar gedaan en we kwamen twee dingen tegen: dat het voor not a number staat en dat het niet positief is. [[2]](#endnote-3)



We zijn dus gaan onderzoeken hoe het komt. We zijn begonnen met 2 classes met elk 5 foto’s, geen problemen. 2 classes met alle foto’s in die class, ook geen probleem. Mogelijk is het een corrupt file die ergens in de dataset zit. Dus we gaan telkens een paar mappen toevoegen in de hoop dat we de boosdoener kunnen vinden. 5 classes met alle foto’s, geen probleem. 7 classes met alle foto’s, nan is terug. Een map eruit, weer een nan. Een al eerder goed gekeurde class eruit en de nieuwe class bewaren en de nan is weg. Conclusie: als en meer dan 5 classes zijn gaat het fout en krijgen we een nan. Nu nog opzoek naar de oorzaak.

Update: Oorzaak is gevonden. De aantal classes stond hardcoded op 5 en dat gaf problemen. Dus dat hebben we veranderd naar 20, de hoeveelheid classes in onze dataset.

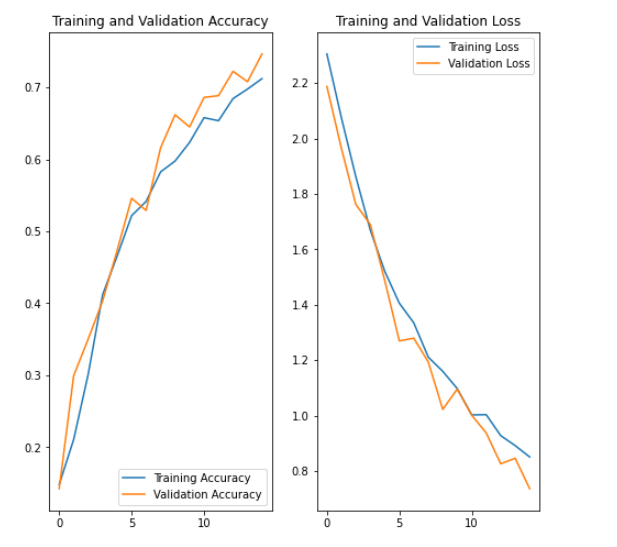
**Resultaat:**  We weten nu hoe we een dataset kunnen toevoegen aan het model en dat we de hoeveelheid classes moeten aanpassen.

## Testen met de dataset

19-11-2020

**Doel:** We willen dat het model grotendeels van de tijd (rond de 4 van de 5) blokjes correct kan herkennen.

Nadat dit probleem was opgelost hebben we de dataset aan het model gegeven. De accuarcy nam toe en de loss nam af, dat zag er veelbelovend uit. Helaas is de loss nog wel best hoog en accuracy best laag.

Loss was hierbij 0.8506 en de accuracy was 0.7120

De resultaten van ons model waren ook helaas niet zo accuraat als we gewenst hadden.

|  |  |  |
| --- | --- | --- |
| Gegeven afbeelding | Resultaat | Percentage |
|  | Brick\_2x2\_L | 97.46% |
|  | Brick\_1x4 | 74.30% |
|  | Brick\_1x4 | 85.33% |
|  | Brick\_1x4 | 76.67% |
|  | Brick\_2x2\_L | 48.31% |
|  | Brick\_2x2\_Slope | 58.30% |
|  | Brick\_2x4 | 96.08% |
| (uit de dataset) | Brick\_1x2 | 62.70% |
|  | Brick\_2x4 | 99.76% |
|  | Brick\_2x2\_L | 97.54% |

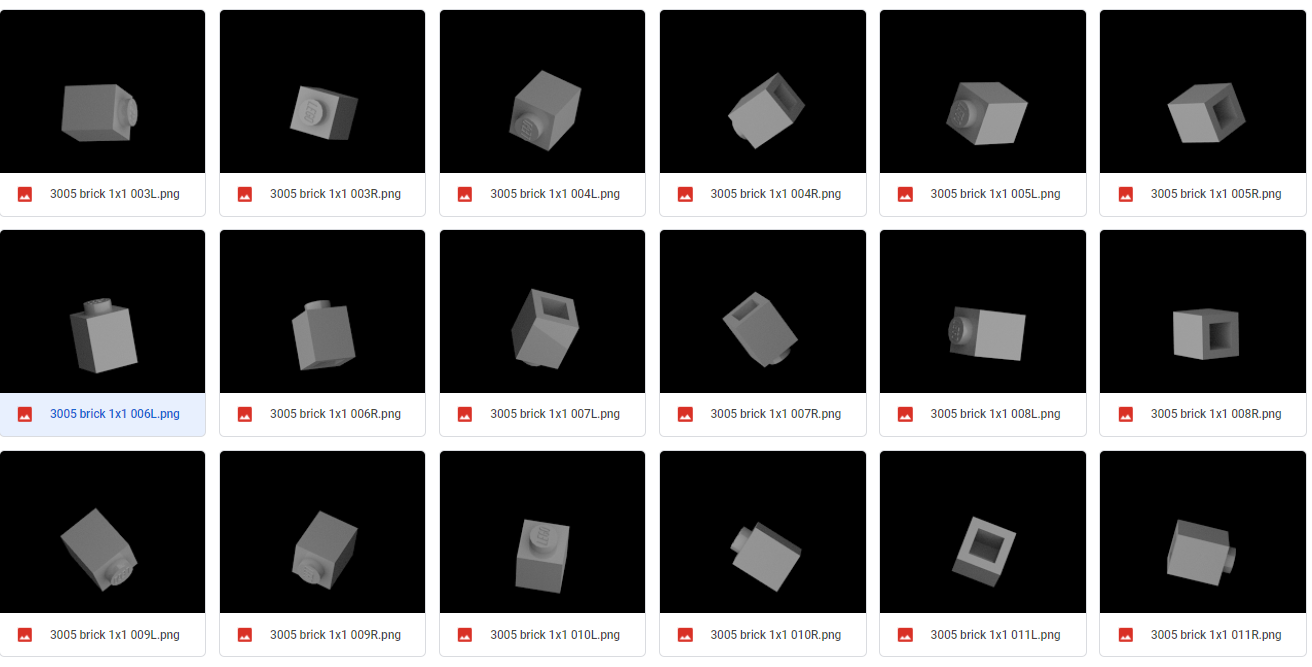
We hadden een aanname gemaakt dat dit te maken kon hebben met onze dataset te maken had dus gaan we zoeken naar een nieuwe dataset.

**Resultaat:** Het model is grotendeels van de tijd incorrect.

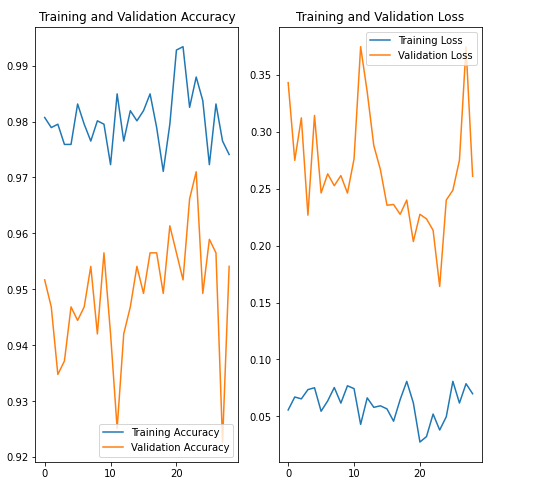
## Een nieuwe dataset van 3D modellen

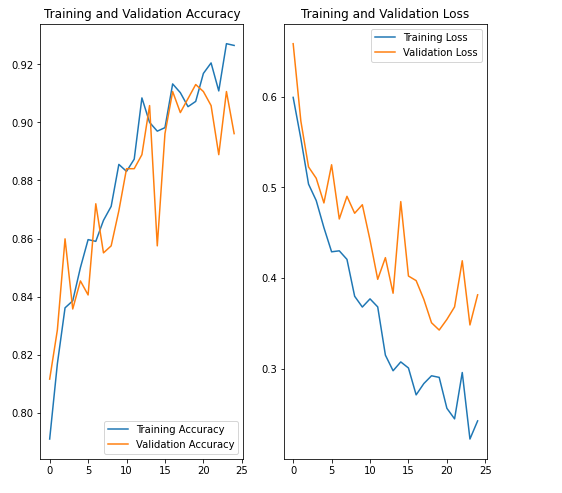
**Doel:** We willen dat het model met onze nieuwe dataset grotendeels van de tijd (rond de 4 van de 5) blokjes correct kan herkennen.

We zijn verdergegaan met een andere dataset. Deze bestond uit afbeeldingen die aan de hand van 3d modellen zijn gemaakt. De blokjes zijn allemaal grijs en hebben een zwarte achtergrond.



Met deze dataset zijn we ook gaan testen. Het gaf de hele tijd aan dat het een blokje van 2x4 was. Na het verwijderen van de 2x4 uit de dataset hebben we het model weer opnieuw getraind, maar nu gaf het aan dat het blokje 1x4 was. Deze dataset werkte dus eveneens niet zo goed als gehoopt.

We hebben geprobeerd om de nauwkeurigheid van het model omhoog te brengen door meer epochs toe te voegen. We merkte dat als we de set code vaker lieten draaien de loss en accuacy niet stegen. Het schommelt een beetje, met als resultaat dat de grafieken erg hard gingen schommelen.



### 

### Omzetten naar Zwart-Wit

Omdat we niet op kleur filteren willen we proberen om de accuracy te verhogen met door de testdata zwart wit te maken. Dat probeerde we met de volgende code

img = keras.preprocessing.image.load\_img(

    path, color\_mode = "grayscale", target\_size=(img\_height, img\_width)

)

Het probleem daarvan was dat de shape (180, 180, 1) in plaats van (180, 180, 3) werd. We hebben niet kunnen vinden hoe we dit kunnen oplossen. We wouden we niet te veel tijd aan besteden en te focussen op andere dingen.

**Resultaat: het model is nogsteeds niet nauwkeurig**

## Template app gebruiken met ons model

23-11-2020

**Doel:** Begrijpen hoe de template app werkt en het model met onze dataset erin verwerken.

We hebben al eerder wat geëxperimenteerd met het exporteren van een tflite model die we in een Android app konden plaatsen. We hebben de een tutorial van codelab als basis gebruikt.[[3]](#endnote-4) We hebben het oorspronkelijke model getest met de oorspronkelijke dataset. We zijn tot de conclusie gekomen dat het originele model correct werkte. Als we onze camera op een bloem richtte gaf hij correct de bloem aan.

We zijn daarna aan de slag gegaan met het vervangen van de dataset in bestaande modellen. Het lukte ons niet om de dataset van het model van de tutorial te vervangen, maar wel van een andere. Die hebben we dus met de laatste stukjes code van het oorspronkelijke tflite-model kunnen downloaden. Eens we hem in de app plaatste liepen we tegen het probleem aan dat de app crashte. We kwamen er met behulp van trial and error achter dat het eraan lag dat er 5 labels nodig waren, toen testen we nog maar met 3 classes. We vermoede dat het ergens in de app stond verwerkt hoeveel labels er nodig zijn.

Later kwamen we erachter dat het aan het model lag hoeveel labels je moet plaatsen. We hadden al eerder het probleem dat het model een NaN gaf als we de hoeveelheid classes niet goed aangaven. Na het aanpassen van de aantal classes konden hadden we niet meer het probleem dat de app crashte. We hadden wel dat het model niet klopte. Het hoogste percentage die we hadden was rond de tweeduizend procent. Het schommelde wel een beetje, maar er was weinig verandering ongeacht waar je de camera op richtte.

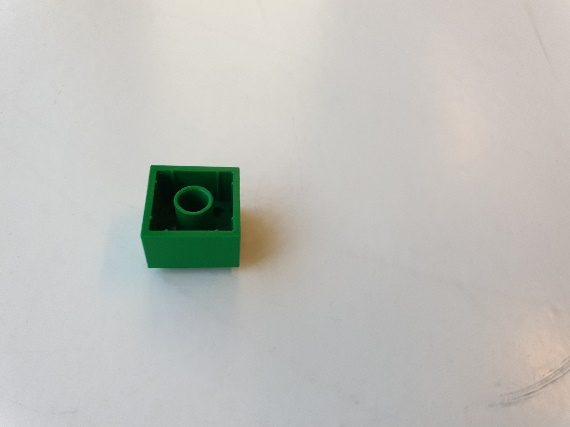
Omdat het nieuwe model niet werkte zijn we teruggegaan naar het oorspronkelijke model waar het nog niet gelukt was om de dataset te vervangen. Met de opgedane kennis kwamen we erachter dat we hetzelfde probleem hadden met de hoeveelheid classes. We hebben dus een dataset samengesteld met 5 classes en dat die konden we wel uploaden. Nadat we onze eigen dataset hadden gemaakt wilden we die ook testen in de app. We hebben gezocht op het getal 5 in de code, vonden die op een plek waar mogelijk de hoeveelheid classes werkt aangegeven en hebben het aangepast naar een 3. Dat werkte en we konden succesvol testen met het model.

**Resultaat:** We hebben een app die met ons model blokjes probeert te herkennen.

Eigen dataset maken

**Doel:** Accurate resultaten behalen met het image classification model.

Vanwege het lage succes met de eerste twee datasets hebben we in overleg met de labprof besloten om een eigen dataset te maken. Het idee hierachter is dat de foto’s waarmee we testen dichter in de buurt komen bij de foto’s waarmee we trainen, deze zijn immers erg vergelijkbaar. Door het maken van deze dataset willen we te weten komen of het probleem daadwerkelijk bij onze oude datasets ligt of toch bij het model dat we gebruiken.



Als het blijkt dat de accuraatheid met onze eigen dataset ook heel laag is gaan we er van uit dat ons model niet geschikt is. De volgende stap wordt dan om de configuratie van layers aan te passen en hiervoor eventueel naar andere vergelijkbare projecten te zoeken. We zijn een tensorflow project tegengekomen van iemand die het gebruikt had om te zien of handen steen, papier of schaar deden. Mogelijk is zijn model geschikter omdat het meer gericht is op de vormen en niet zo zeer op de kleuren in de foto’s in tegenstelling tot het bloemenmodel.

**Resultaat:** We hebben onze eigen dataset klaar om te testen.

## Goed resultaat eigen dataset

24-11-2020

**Doel:** Accurate resultaten behalen met het image classification model.

Vandaag is er veel vooruitgang geboekt. De dataset die we gemaakt hebben levert boven verwachting goede resultaten. We hebben de dataset met 3 classes gebruikt voor het trainen van het model. De loss werd heel laag en we haalde een accuracy van 1.000, wat wenselijk is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We hebben het model van alle drie de blokjes foto's van het internet gegeven. Hij vond ze allemaal goed met “100% zekerheid”.

Les van vandaag: Don’t mess up your labels! Toen we ons net getrainde model in de android app wilde testen schrokken we. De app zat er de hele tijd met hoge zekerheid naast. Wat wel opviel was dat de app consistent was. Het gaf dat wel het verkeerde resultaat, maar als je een blokje twee keer liet zien gaf de app wel twee keer hetzelfde resultaat. Uiteindelijk kwamen we erachter dat het bestand waar we de labels in aangeven op de verkeerde volgorde stond. Eenmaal aangepast werden de blokjes wel goed gelabeld.

## Gesprek met maker van LEGO sorteermachine – Daniel West

26-11-2020

**Doel:** Extra informatie over machine learning leren.

Op dinsdag hebben we een gesprek gehad met Daniel West. Daniel komt uit Australië en heeft als hobby project een LEGO sorteermachine gebouwd die gebruik maakt van machine learning om de blokjes te onderscheiden. Op youtube heeft hij een vidoegemaakt waar hij een demo geeft van zijn machine.[[4]](#endnote-5)

Een van de belangrijkste punten die we uit dit gesprek opgemaakt hebben is dat foto’s van 3d modellen niet zullen werken als dataset. Dit heeft ermee te maken dat er altijd subtiele verschillen zijn met echte beelden wat betreft belichting etc. Voor ons zien die verschillen er minimaal uit, maar een tensorflow model kan er helemaal niks mee. Verder vertelde Daniel ons dat we op moeten passen dat er geen sprake is van over- of underfitting met onze dataset. Daniel vond ons huidige model eigenlijk te eenvoudig en vertelde ons over het model dat hij gebruikt had voor zijn project genaamd resnet v2. Hij adviseerde ons om ons hierin te verdiepen als onze resultaten niet accuraat genoeg zijn.

**Resultaat:** We hebben wat extra informatie gewonnen

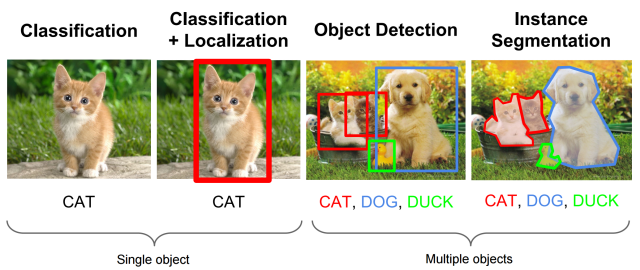
# Hoofdstuk 3: Object detection

## Van classification naar object detection

30-11-2020/1-12-2020

**Doel:** Overstap maken naar object detectie.

We hebben nu veelbelovende resultaten met een app die een afbeelding kan classificeren. Maar we gaan sowieso ons model moeten aanpassen zodat we meerdere objecten tegelijk kunnen herkennen.



Voor de volgende stap willen we gericht zoeken. Dat je de app een hoop met LEGO laat zien en de app het gevraagde LEGO blokje zoekt en dit dan in plaats van dat hij aangeeft wat hij denkt te zien. Om dit te realiseren hebben we twee methodes gevonden:

De eerste methode is dat we een model hebben die in staat is om alle blokjes te herkennen. Die laten we het de LEGO zien en output genereren. De app kijkt naar welk blokje er gevraagd wordt. Alleen de data die over het gevraagde blokje gaat wordt dan gebruikt. Aan de hand van die data kan de app aangeven waar het blokje ligt. Het voordeel van deze methode is dat we maar één model nodig hebben. Het nadeel is dat het model getraind dient te worden voor alle blokjes die in de app staan en opnieuw getraind dient te worden eens er een blokje toegevoegd wordt. Ook moet het model in staat zijn om veel blokjes te gelijk te herkennen. Als er bijvoorbeeld tien blokjes in beeld liggen willen we dat het minimaal tien blokjes kan herkennen zodat hij niet het te zoeken blokje over het hoofd ziet.

Een andere optie is om voor elk blokje een apart model te maken. De app gebruikt alleen het model van het gevraagde blokje. Het model hoeft dan alleen maar te weten of het blokje er ligt of niet, vergelijkbaar met het hotdog or not hotdog model.[[5]](#endnote-6),[[6]](#endnote-7) Aan de hand van de data die het model genereerd geeft de app aan waar het blokje ligt. Het voordeel van deze methode is dat we met kleinere modellen werken die sneller getraind zullen zijn. Voor het toevoegen van een nieuw blokje hoef je niet het grote model opnieuw te trainen, er hoeft alleen een toegevoegd te worden. Een nadeel is dat als we dezelfde methode gebruiken als bij de hotdog modellen, we ook data moeten hebben dat niet het blokje is. In plaats van alleen verzamelingen met foto’s van het blokje, hebben moeten we ook een map samenstellen met foto’s waar andere blokjes op staan.

Een tussenweg is dat we meerdere modellen maken die een kleine hoeveelheid blokjes kunnen herkennen. Het voordeel daarvan is dat we geen datasets hoeven samen te stellen met een ‘niet-het-blokje-class’. Ook hoeven we niet het hele model opnieuw te trainen voor het toevoegen van een dataset. Een nadeel is wel dat het model mogelijk veel false positives gaat geven door dat het blokjes ziet waar het model niet op getraind is maar wel denkt te herkennen. Mogelijk is dit met een toevoeging van een ‘niet-het-blokjes-class’ weer opgelost, maar dan moeten we alsnog weer de classes samenstellen.

**Resultaten:** We hebben ideeën welke kant we verder op kunnen gaan.

## Blokjes aangeven in de app

3-12-2020

**Doel:** App uitwerken voor metingen van ontwerpen dataset.

Omdat we vastlopen met het verbeteren van het model hebben we besloten om voor de afwisseling maar de app te verbeteren.

We hebben radio buttons toegevoegd waarmee het te zoeken blokje geselecteerd kan worden. Daarvoor hebben we documentation van andoid studio gebruikt.[[7]](#endnote-8) Daarna heb ik in de code het gedeelde waar de resultaten worden weergegeven opgezocht en aangepast. Eerst liet het de top 3 zien met percentages, nu alleen of het geselecteerde blokje nummer 1 is.

Het probleem waar ik tegen aanliep is dat de code van de radiobuttons niet bedoeld is om in een loop te laten werken, zoals het in de huidige code doet. Het geeft een error op de regel code waar er gecontroleerd wordt of de er iets gecheckt is. We hebben het probleem opgespoord en opgelost met een try and catch statement.[[8]](#endnote-9) Als de radiobutton nu veranderd wordt de waarde ook veranderd, maar als het niet lukt zorgt de try en catch ervoor dat de app niet crasht.

Update: we hebben geprobeerd om de checked variable eruit te halen, zodat we het hele try en catch systeem niet meer nodig hebben. Het systeem werkt correct zonder de check, dus dat is vooruitgang.

Dit is werkt nog niet optimaal. Behalve dat er nog meer 3 van de 9 blokjes in verwerkt zijn is het model niet getraind voor het verwerken van meerdere inputs. Als je alle blokjes laat zien zal hij de most likely als eindresultaat geven, met als gevolg de het niet goed werkt als je meerdere blokjes laat zien.

Er is een verbetering gemaakt: de percentages zijn terug.

Nog een verbetering: als je de app opent geeft hij een melding weer dat je een brick moet selecteren. Eerst hadden we hem in de code geplaats, maar blijkbaar niet op de juiste plek want de app crashde. Daarna heb ik heb in de xml geplaats, de veilige route, en dat werkte wel.

**Resultaat:** De app is aangepast voor het testen van de dataset.

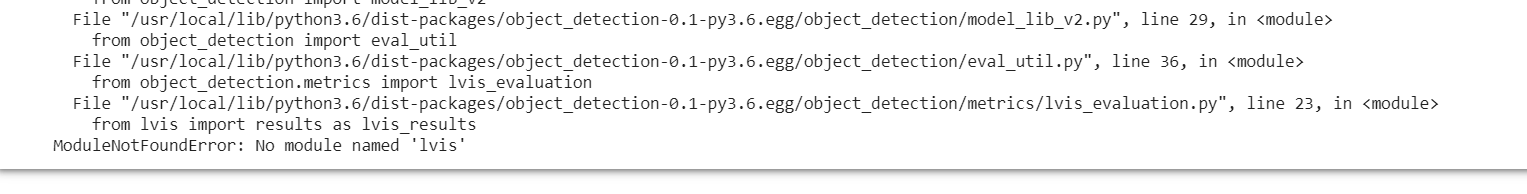
## Vooruitgang met object detection

8-12-2020

**Doel:** Onderzoek naar object detection.

Omdat we object detection toch een kans willen geven zijn we verder gaan zoeken naar een model. We hebben een tutorial gevonden op medium.[[9]](#endnote-10) Het werkt al met google drive en Colab, wat we al gewend zijn om mee te werken. In de eerste paar stappen moeten we data verzamelen die we op de drive kunnen zetten. Behalve de dataset moet er ook onder andere een pre-traint model bij. We hebben alle gesuggereerde data gebruikt, inclusief de dataset. De dataset bevat drie soorten fruit waarop we kunnen oefenen. We hebben ervoor gekozen om de dataset met fruit te gebruiken om te testen of het een bruikbaar model is. Voor object detection hebben een andere dataset nodig en die hebben we nog niet. Behalve een kleine typefout was het geen probleem om dit uit te voeren.

De andere stappen hebben we ook zonder problemen kunnen uitvoeren tot stap 17, het model testen. Dan geeft google colab een error dat hij het model niet kan vinden. Als we terug scrollen kwamen we wat verontrustends tegen bij stap 15:



Een module genaamd lvis kan niet gevonden worden.

Als we terugkijken naar de tutorial is de verwachte output ook anders dan die van ons.

Na even het internet afspeuren zijn we op een stackoverflow pagina gekomen die suggereert om de code “pip install lvis” de gebruiken.[[10]](#endnote-11) Dat hebben we geprobeerd en de error is weg. Op naar de volgende error.

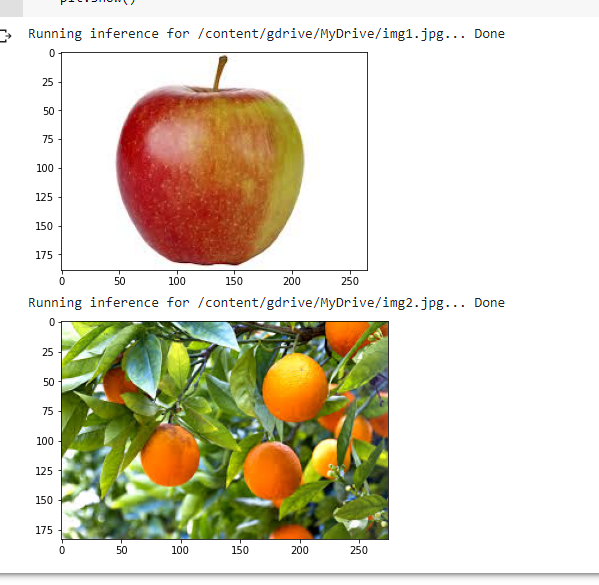
We krijgen nu meer regels code bij stap 15, maar het eindigt met:

ValueError: Unsuccessful TensorSliceReader constructor: Failed to get matching files on pre-trained-models/ssd\_resnet50\_v1\_fpn\_640x640\_coco17\_tpu-8/checkpoint/ckpt-0: Not found: pre-trained-models/ssd\_resnet50\_v1\_fpn\_640x640\_coco17\_tpu-8/checkpoint; No such file or directory

De eerste stap wat we gaan proberen is om de bestanden in te duiken en het bestand of map te zoeken.

Ik heb het model geopend, die in de vorm van een tar file staat. Daarin staat wel een bestand genaamd checkpoint. Mogelijk is het het probleem dat het bestand in een tar file staat. We gaan dus proberen de tar file uit te pakken en opnieuw te uploaden.

Het uitgepakte model werkt. Tijd om te testen!



Helaas lukt het ons hierbij nog niet om ook bounding boxes te krijgen.

**Resultaat**: Het model werkt niet.

## Poging object detection

10-12-2020

**Doel:** Onderzoek naar object detection.

Tijd om in de output data te gaan zoeken naar het probleem.

Bij stap 15 staat er in de output veel dingen die mogelijk problemen kunnen veroorzaken

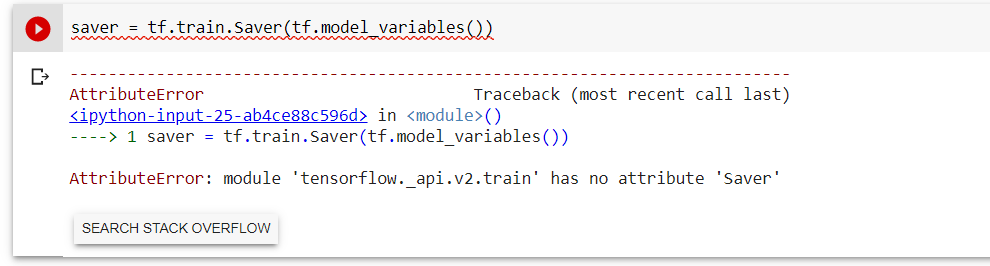




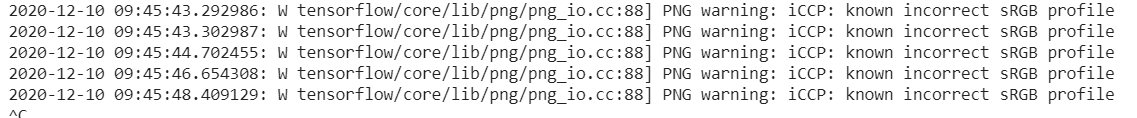
Deze melding kregen we voor meer regels dan nu weergegeven.

Het onderste probleem ziet er het problematische uit omdat het zo groot is. Daarom gaan we daar nu op focussen.

Het eerste antwoord op een stackoverflow pagina gaf geen succes:[[11]](#endnote-12)



Op een andere tutorial voor object detection word de zelfde regel code gebruikt om te trainen. Daar worden de warnings als verwachte output gegeven, dus die gaan we voor nu negeren focussen op andere problemen.[[12]](#endnote-13)



Volgende klant. Deze is aan het einde van de output, dus mogelijk wordt het hier afgekapt.

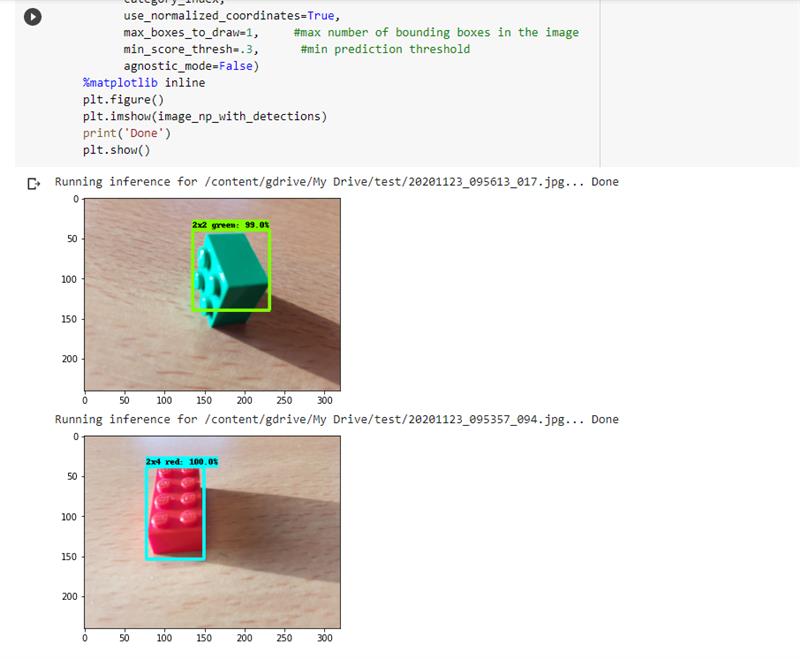
Het vreemde van deze waarschuwing is dat er gewaarschuwd wordt voor een PNG, terwijl wij met JPG werken…

Het probleem zit hem waarschijnlijk in de lvis die we eerder hebben geïnstalleerd.

## Werkende object detection

15-12-2020

Een doorbraak! Met een kleine gelabelde LEGO dataset is het model succesvol getraind. Uiteindelijk bleek het probleem bij de batch\_size te liggen. Standaard stond deze op 16 maar we zijn erachter gekomen dat er problemen kunnen ontstaan als de train dataset niet door de batch\_size te delen is. Althans er moet een rond getal uit komen. Onze train dataset bestaat op dit moment uit 56 foto’s en 56/16=3,5, dus dat werkt niet. We hebben de batch\_size naar 8 verandert want 56/8=7 en bij de test bleek dat het model zowaar getraind werd.



Het trainen van het model duurde wel erg lang, want het was ingesteld om hiervoor 25000 stappen te doorgaan. Dit zou ons te veel tijd kosten voor een test, dus hebben we de stappen teruggeschroefd naar 5000. Uiteraard gaat dit ten koste van accuraatheid, maar op dit moment maken we ons daar nog niet zo’n zorgen over. Als we op een later punt een uitgebreidere dataset hebben kunnen we de stappen altijd weer verhogen.

**Resultaat**: Het model werkt

## Converteren naar TFLite

17-12-2020

**Doel:** Object detection model converteren naar .tflite.

Als eerstvolgende stap willen we het model converteren van .pb naar .tflite om het in de app te kunnen testen. Daarbij lopen we echter tegen de volgende error aan:



Na dat we het halve internet hebben afgezocht zijn we erachter gekomen dat het aan de vorm van de tensor ligt. In de foutmelding staat: (tensor<1x?x?x3x!tf.quint8>) -> tensor<1x?x?x3xui8>. We hebben geprobeerd om de vorm aan te passen, maar tevergeefs.

Deze [githubpost](https://github.com/tensorflow/tensorflow/issues/30180#issuecomment-505959220) hebben we geprobeerd, maar dit gaf hetzelfde resultaat.[[13]](#endnote-14)

Op internet zijn we situaties tegengekomen waar mensen dezelfde problemen hadden. Een oplossing die we op [github](https://github.com/tensorflow/models/issues/9287) zijn tegengekomen is waar er geen problemen zijn met een ander pretrained model.[[14]](#endnote-15) Dus hebben we ervoor gekozen om een ssd\_mobilenet\_v2\_fpnlite\_320x320\_coco17\_tpu-8 model te gebruiken. We hebben de tutorial opnieuw gevolgd maar dan met een ander model.

Helaas heeft het niet gewerkt. Bij het converteren krijgen we dezelfde foutmelding.

We hebben ook een manieren om te converteren geprobeerd, volgens een blogpost van Gilbert Tanner, maar dat werkte ook niet. [[15]](#endnote-16)

**Resultaat**: Het model werkt wel, maar is niet geconverteerd naar tflite.

## Ander model converteren

4-1-2020

We hebben een ander notebook gevonden, Training\_MobileDet\_Custom\_Dataset.ipynb, die bedoeld is voor het maken van een TFLite model. [[16]](#endnote-17) Met de kennis die we bij de vorige notebook hebben opgedaan kunnen we hopelijk het model trainen en converteren.

Bij het model wordt alle benodigde data al zelf binnengehaald, we hoeven geen heel pakket samen te stellen zoals bij de vorige. Helaas liepen we wel weer tegen het probleem tegen het probleem van de “PNG warning: iCCP: known incorrect sRGB profile” aan. Vorige keer hadden we dit opgelost met de batch size, dus dat gaan we nu nog maar een keer proberen.

Na het aanpassen van de batch size is het model zonder problemen geconverteerd. Alleen is er een probleem met het testen van het model. Die geeft op het laatste stukje code de foutmelding: “TypeError: \_\_array\_\_() takes 1 positional argument but 2 were given”

De oorzaak van het probleem zit hem in de functie die in de cel erboven staat. Om het de oorzaak te achterhalen hebben we het de functie ontleed en er normale code van gemaakt. Daaruit blijkt dat de oorzaak in de volgende regel zit: results = detect\_objects(interpreter, preprocessed\_image,

threshold=threshold)

Na nog meer ontleden van fucties blijkt dat het probleem in de regel input\_tensor[:, :] = image

Zit. Die staat in de bovenste functie van de tweede cel van het Run infence hoofdstuk.

Na het verwijderen van de [:, :] komt de code wel voorbij dat punt, dus op naar de volgende error:

RuntimeError: There is at least 1 reference to internal data in the interpreter in the form of a numpy array or slice. Be sure to only hold the function returned from tensor() if you are using raw data access.

Hiervoor hebben we geen oplossing gevonden.

We hebben ook geprobeerd om ons model te vervangen met een model in een bestaande app van tensorflow die als voorbeeld diende[[17]](#endnote-18) Dat gaf zorgde ervoor dat de app meteen cashte na het openen.

Na overleg met onze opdrachtgever hebben we besloten voorlopig niet op de app te focussen.

**Resultaat**: We hebben een model geconverteerd, maar die werkt niet. We gaan niet meer op de app focussen.

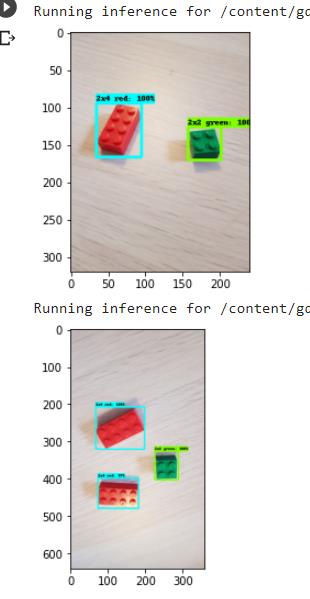
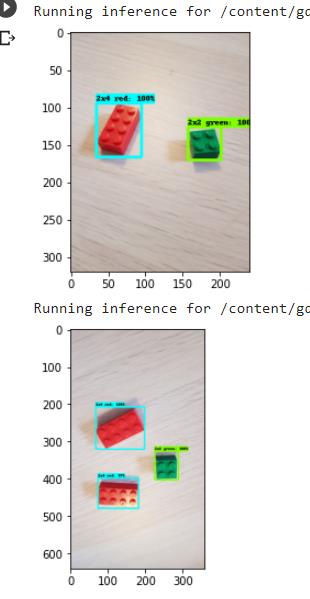
## Meerdere blokjes herkennen

4-1-2020

**Doel**: Testen of het model meerdere blokjes in een foto kan herkennen

Uit een gesprek met onze opdrachtgever is gebleken dat op dit moment de prioriteit ligt bij het onderzoeken hoe accuraat het model is. Dit houdt in dat de opdrachtgever wil weten of het model nog steeds werkt als de gegeven foto heel onrustig of onduidelijk is. Onze opdrachtgever dacht dat het een goed idee was om stappen in kaart te brengen waarmee we de foto’s testen, de zogenaamde stappen van chaos. Stap 1 zou een foto zijn van een LEGO blokje op een rustige achtergrond en de laatste stap zou bewegend beeld zijn van een grote stapel LEGO blokjes die overlappen.

Om te beginnen is er getest met twee verschillende blokjes in een foto. Na het aanpassen van de maximale hoeveelheid bounding boxes hebben we het volgende resultaat gekregen:



Op dit moment weten we dat het model in staat is om op een foto meerdere LEGO blokjes aan te wijzen die niet overlappen. Na het bespreken van onze stappen hebben we met de opdrachtgever besproken dat we ze nog wat verder uit moeten werken. Dat is dus waar we als volgende punt mee aan de slag gaan.

**Resultaat**: Het model kan meerdere blokjes in een foto herkennen.

## Toevoegen meer classes aan dataset.

18-1-2021

Natuurlijk is het toevoegen van extra classes aan de dataset een lastige klus omdat overal handmatig bounding boxes omheen getekend moeten worden, maar er zijn nog wat andere problemen die zich voorgedaan hebben. Zo is het opgevallen dat er voor elke class een ongeveer gelijke hoeveelheid train en test afbeeldingen moeten zijn anders is er een grote kans dat er false positives ontstaan voor classes met veel afbeeldingen. Het is ook nog belangrijk dat de foto's in de train en test folder deelbaar moeten zijn door de batch size, anders kan het model niet getrained worden. Op dit moment ligt de focus op het aanbrengen van tweaks en uitzoeken hoe de meeste accuraatheid behaald kan worden met onze dataset.

Het voornaamste doel van de laatste week van deze sprint is om te achterhalen hoe de accuraatheid van het model te manipuleren is. Het is heel interessant om te weten te komen hoe er invloed uitgeoefend kan worden op het gedrag van het model omdat er vaak dingen fout gaan bij het uitbreiden van de dataset. Wellicht is het mogelijk om een soort van stappenplan te maken met uitleg over wat er te proberen valt bij het behalen van bepaalde resultaten. Dit zou ideaal zijn om mee te geven aan een volgende groep.

# Hoofdstuk 4: De toekomst

Omdat het einde van het semester nadert komt het project voor nu tot een einde. Het product is nog niet af. De punten die nog gedaan kunnen worden zijn opgesplitst in twee categorieën, de minimal viable product en de toekomstige features. Bij de minimal viable product staan de essentiële zaken die nodig zijn om een werkend product te krijgen. Bij de toekomstige features staan punten die interessant zijn om het project uit te breiden, maar niet nodig zijn om het product werkend te krijgen.

## Minimal viable product

Omdat het einde van het semester nadert komt het project voor nu tot een einde. Het product is nog niet af. Om een minimal viable product te maken zijn er in ons opzicht de volgende stappen vereist:

Een van de stappen die er nog genomen moeten worden is het realiseren van de app. Mogelijk zijn er online geschikte apps te vinden die gebruikt kunnen worden als basis voor de app, maar omdat er geen uitgebreide verdieping heeft plaats gevonden in het combineren van een app en het model is het moeilijk om in te schatten welke werkzaamheden er nodig zijn voor het realiseren van de app.

Voor een compleet eindproduct moet de app behalve werken er ook goed uit zijn. Er dient een aantrekkelijke, gebruiksvriendelijke layout te zijn. Omdat er veel soorten LEGO blokjes zijn kan het een opgave zijn om het juiste blokje aan te geven. Daarom hebben we een human computer interaction experiment gehouden om het filtersysteem gebruikersvriendelijk te maken.

Een ander aspect dat zeker niet vergeten mag worden voor een werkend eindproduct is het toevoegen van meer blokjes. Er zijn heel veel verschillende soorten LEGO blokjes die de gebruiker niet kan vinden. Als de hoeveelheid blokjes maar beperkt is zal de gebruiker geneigd zijn om de app links te laten liggen omdat de kans dat het te zoeken blokje niet in de app zit te groot is.

Om tijd te besparen is het een optie om het maken van de dataset te automatiseren. Voor het maken van de dataset waren een handvol stappen nodig die relatief tijdrovend waren. We hebben gekeken naar het automatiseren van de dataset, maar zijn tot de conclusie gekomen dat het tijdrovend was om te realiseren en dat we het voordeel er niet uit zouden halen. Dat heeft er mee te maken dat er voor dit semester de focus niet lag op het uitbreiden van de hoeveelheid blokjes.

Bij het uitbreiden van het model is het belangrijk om de accuracy in de gaten te houden. Tijdens het testen van het model is er opgemerkt dat door het toevoegen van nieuwe blokjes het model sneller incorrect de blokjes aangeeft. Aan het einde van het semester is er veel gewerkt aan het verbeteren van de accuracy omdat het van de prioriteit is van de opdrachtgever.

## Toekomstige features

Hier staan wat punten waar het product in de toekomst mee uitgebreid kan worden. Ze zijn niet noodzakelijk, maar wel interessant om over na te denken.

**Het model trainen via de app**  
Een probleem waar gebruikers tegen aan kunnen lopen is dat de train omgeving van het model anders is dan omgeving van de gebruikers. Om dat probleem op te lossen kan er een feature gemaakt worden om het model verder te trainen voor eigen gebruikt. Het model kan dan bijvoorbeeld zich aan de achtergrond of belichting van de gebruiker aanpassen. Om deze feature toe te voegen dient er het model is staat te zijn om verder te trainen. De gebruiker kan dan foto’s maken van blokjes in zijn omgeving en aan de hand daarvan het model laten trainen. Omdat een mobiel hoogstwaarschijnlijk niet krachtig genoeg om een model te trainen zal de data verstuurt moeten naar een server.

**Gebruikers de dataset uitbreiden**Om tijd te besparen is het mogelijk om de dataset te laten uitbreiden door de gebruikers. Die maken dan foto’s van hun LEGO blokjes, labelen ze en sturen ze door. De foto’s hoeven dan alleen nog maar gecontroleerd te worden. Een ander voordeel is dat de dataset gevarieerder wordt doordat de blokjes in meer omgevingen gefotografeerd worden.

**Sets toevoegen**   
LEGO wordt vaak in sets verkocht. Daarom is het handig voor de gebruiker om aan te geven welke sets ze in bezit hebben. Zo kunnen ze bijvoorbeeld aangeven dat ze een stukje op een bepaalde bladzijde niet kunnen vinden. Aan de hand daarvan kunnen ze dan kiezen welke van de blokjes ze missen en hoeven ze niet te filteren tussen alle blokjes.

**Filteren met stem**Om de app moderner en gebruiksvriendelijker te maken kan er een optie toegevoegd worden om te filteren met stem. De gebruiker kan dan inspreken welk blokje er gezocht wordt, dan kan dat met behulp van software omgezet worden een zoekopdracht voor het te zoeken blokje.

**Favorieten**Om gebruikers de frustratie van het herhaaldelijk moet opzoeken van een blokje kan de feature om favorieten aan te geven toegevoegd worden. Er komt dan een lijst met alle blokjes die de gebruiker als favoriete heeft toegevoegd. Verder kan een optie komen om favoriete blokjes als eerste te laten zien.

**Samenvoegen van sommige blokjes in het model**Tijdens het proces is het opgevallen dat het model soms moeite heeft om vergelijkbare blokjes te herkennen. De app zal mogelijk niet een blokje herkennen omdat hij het verwart met een vergelijkbaar blokje. Een oplossing daarvoor kan zijn om sommige blokjes samen te voegen. De gebruiker kan dan vaker hebben dat de app incorrect aangeeft dat hij het blokje heeft gevonden, maar het zal minder vaak wat over het hoofd zien. Verder zal het schelen in maken van de dataset.

# Literatuurlijst

<https://www.youtube.com/watch?v=cvyDYdI2nEI>

<https://www.youtube.com/watch?v=pXLLNa4IrmM&t=729s>

<https://www.tensorflow.org/lite/models/object_detection/overview>

https://github.com/transcranial/inception-resnet-v2/blob/master/inception-resnet-v2.ipynb https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html https://colab.research.google.com/drive/1J5em0W8el38YJPCz6wxNRwTJ6spyYHCX?usp=sharing

1. Image classification | TensorFlow Core. (z.d.). Geraadpleegd op 16 november 2020, van https://www.tensorflow.org/tutorials/images/classification [↑](#endnote-ref-2)
2. Lakshmanan, L. (2019, 24 februari). Debugging a Machine Learning model written in TensorFlow and Keras. Geraadpleegd op 16 november 2020, van https://towardsdatascience.com/debugging-a-machine-learning-model-written-in-tensorflow-and-keras-f514008ce736 [↑](#endnote-ref-3)
3. Wei Wei (revamping author), Yash Katariya (original author). (z.d.). Recognize Flowers with TensorFlow Lite on Android |. Geraadpleegd op 23 november 2020, van https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android#0 [↑](#endnote-ref-4)
4. The WORLD’S FIRST Universal LEGO Sorting Machine. (2019, 3 december). [Videobestand]. Geraadpleegd van https://www.youtube.com/watch?v=04JkdHEX3Yk [↑](#endnote-ref-5)
5. Seth, V. A. P. B. Y. (2018, 14 september). Hotdog or Not Hotdog – Image Classification in Python using fastai. Geraadpleegd op 1 december 2020, van https://yashuseth.blog/2018/03/05/hotdog-or-not-hotdog-image-classification-in-python-using-fastai/ [↑](#endnote-ref-6)
6. Jain, Y. (2018, 30 augustus). Building The Hotdog/Not-Hotdog Classifier From HBO’s Silicon Valley. Geraadpleegd op 1 december 2020, van https://towardsdatascience.com/building-the-hotdog-not-hotdog-classifier-from-hbos-silicon-valley-c0cb2317711f [↑](#endnote-ref-7)
7. Android Studio. (2020, 18 november). Radio Buttons |. Geraadpleegd op 3 november 2020, van https://developer.android.com/guide/topics/ui/controls/radiobutton [↑](#endnote-ref-8)
8. w3schools.com. (z.d.). Java Exceptions (Try...Catch). Geraadpleegd op 3 december 2020, van https://www.w3schools.com/java/java\_try\_catch.asp [↑](#endnote-ref-9)
9. Kapkar, N. (2020, 27 oktober). TensorFlow 2 Object Detection API With Google Colab. Geraadpleegd op 8 december 2020, van https://medium.com/swlh/tensorflow-2-object-detection-api-with-google-colab-b2af171e81cc [↑](#endnote-ref-10)
10. Tensorflow: Object detection api error on google colab-no module named lvis. (2020, 29 oktober). Geraadpleegd op 8 december 2020, van https://stackoverflow.com/questions/64590903/tensorflow-object-detection-api-error-on-google-colab-no-module-named-lvis [↑](#endnote-ref-11)
11. TF2.0: Translation model: Error when restoring the saved model: Unresolved object in checkpoint (root).optimizer.iter: attributes. (2019, 8 oktober). Geraadpleegd op 10 december 2020, van https://stackoverflow.com/questions/58289342/tf2-0-translation-model-error-when-restoring-the-saved-model-unresolved-objec [↑](#endnote-ref-12)
12. Lyudmil Vladimirov Revision 725f2221. (2020). Training Custom Object Detector — TensorFlow 2 Object Detection API tutorial documentation. Geraadpleegd op 10 december 2020, van https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html [↑](#endnote-ref-13)
13. Oktai Tatanov. (2019, 26 juni). How to specify shape of input for TFLite model after receiving SavedModel format? · Issue #30180 · tensorflow/tensorflow. Geraadpleegd op 15 december 2020, van https://github.com/tensorflow/tensorflow/issues/30180 [↑](#endnote-ref-14)
14. Lech Wołowski. (2020, 23 september). SSD ResNet from model zoo not working after conversion to TFLite · Issue #9287 · tensorflow/models. Geraadpleegd op 15 december 2020, van https://github.com/tensorflow/models/issues/9287 [↑](#endnote-ref-15)
15. Tanner, G. (2020, 27 januari). Convert your Tensorflow Object Detection model to Tensorflow Lite. Geraadpleegd op 15 december 2020, van https://gilberttanner.com/blog/convert-your-tensorflow-object-detection-model-to-tensorflow-lite [↑](#endnote-ref-16)
16. Sayakpaul/E2E-Object-Detection-in-TFLite. (2020, 8 december). [GitHub]. Geraadpleegd van https://github.com/sayakpaul/E2E-Object-Detection-in-TFLite [↑](#endnote-ref-17)
17. TensorFlow Lite Object Detection Android Demo. (z.d.). Geraadpleegd op 12 november 2020, van https://github.com/tensorflow/examples/tree/master/lite/examples/object\_detection/android [↑](#endnote-ref-18)