Object Detection through deep learning and live camera feed

Content

How it works

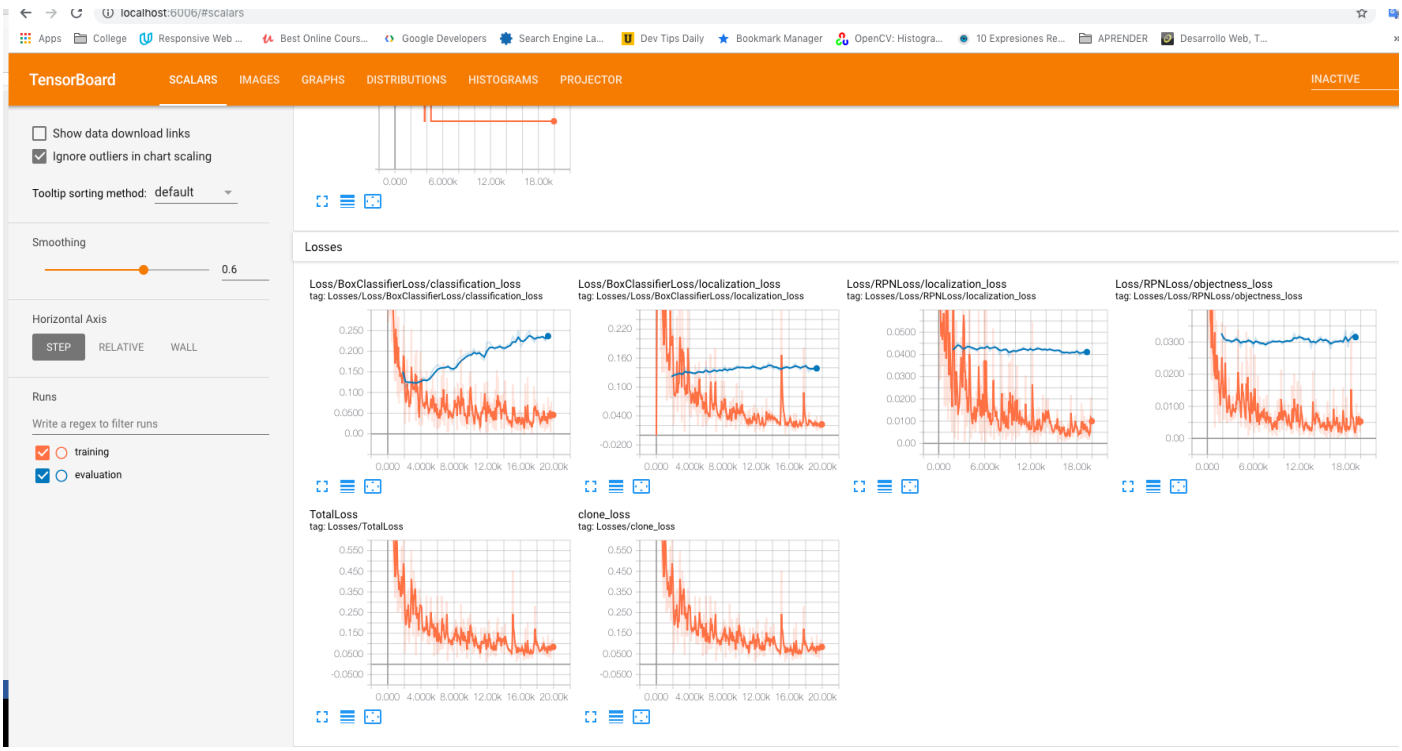
The Tensorflow program is the program that uses deep learning to identify objects in images, however some manual input is needed before it is able to identify the objects that you wish to target.
It uses a pre-trained model as a base and then retrains this base to the specific targeted items. The pre-trained model has many iterations already implemented and therefore retraining it makes the process faster. Creating your own model would take much time and it doesn't bring real benefits.

To train it the program is given a set of images. Each image is given a label. This is a square box with a given name made by the python program labelImg for the program to know where the to be identified object is located. For example the label "Can" needs 300 or more images of cans for accuracy in the results. This needs to be done for every label given, so 6 labels would need 1800 images at least with 300 images of each label. These images are then divided in a test group and a train group, 20% for testing and 80% for training (make sure that the images for each label are divided equally). It uses the training images to create a solution that it thinks can recognize all the objects accordingly and then tests this solution, called an inference graph, on the test images. It then learns from its mistakes and gives it another try to improve the inference graph.
This process then continues until stopped by the programmer. Its progress is visible in the loss rate. The more accurate the inference graph, the lower the loss rate is. Once the loss rate

drops consistently below 0.05 the graph is strong and you can stop the process, but lower is of course better. One thing to be careful with is that it is possible for the program to “overtrain”. It is then starting to learn the test images and starts overfitting on these images, causing it to give worse results once it works on a live feed.

During the training process information on this loss rate is given, but it is difficult to read and keep track of the last sets of iteration. Too ease the readability TensorBoard is used. Here all the data is gathered in a live graph. You can then see more easily if the graph has stabilised or is overfitting if the loss rate starts to increase again

After the training has been completed the inference graph is converted to a frozen inference graph. This graph can then be used in a live camera feed to identify the objects. In the code, that uses this graph, pieces of code have been implemented to save data gathered while identifying objects.



Conclusions tests up until now

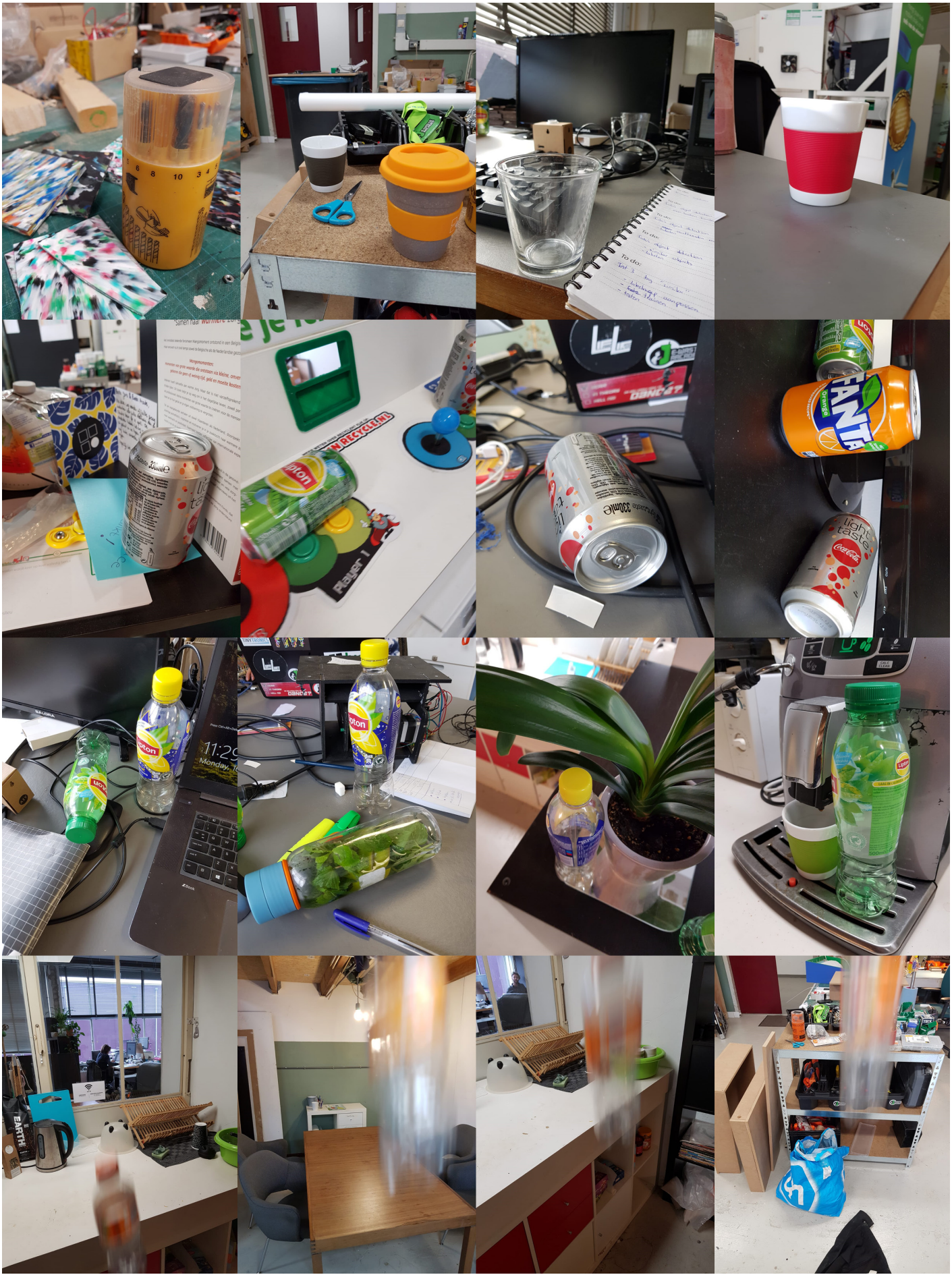
Tests have been done already to determine what can and can't be done with the camera and how to improve the results (Individual tests with notes are at the end of the document):

- A moving object can be detected (Test 12)
- A moving object can be identified (Test 13 - 16)
    - Implementing both a static and dynamic label for each object with the accompanying images. The static label consists of only static objects, while the dynamic label consists falling objects. Without the double labels the label becomes too broad and mistakes rise.
- Variables that improve the model whilst taking pictures
    - The photos need to be taken in a noisy back ground even though the model will be used in a calm background later (Test 8-10)
    - Pictures need to be taken with high and low brightness
    - Different angles need to be photographed
    - Variation within a label. For example multiple sorts of cans not just a fanta can (Test 4)
    - Put other but similar objects that should not identified in the images as well to not take them into the label (Test 3, 5)
    - Holding or covering the object whilst making pictures (Test 2)
    - Color should be in both the to be identified objects and other objects such as a red can and a red mug to cancel it out as a variable (Test 3 - 4)

- SSD Mobilenet model does not have the same output as faster_rcnn (Test 1, 19)
    - Its certainty is way lower
    - Dynamic objects are not identified
    - The targeted static objects are not identified, given the wrong label or double labels
    - Other objects are most of the time not identified, however more often than with faster_rcnn
    - Frame rate is however way higher

- The code and the type of model are the main links that slow down the FPS/speed, not for example the camera (Test 18)

All the images that have been made so far, along with the graphs, are saved on the micro SD card that belongs to this project.



A sample of pictures that are in the dataset. The images have different objects, angles, backgrounds and lightsources

Why it didn't work

I started with figuring out if the object detection technology is a feasible achievable option for the products in use right now. Since the last test with the SSD Mobilenet modeltype went so horrible it seemed less likely that the Raspberry Pi, every version of it, would have the computation power to achieve the wanted results. The Pi needs an even weaker version of this modeltype according to the vision kit wiki and after browsing the web all the examples that used the Pi showed an FPS of 1. This is obviously way too low for falling objects to detect and this was on SSD Mobilenet which showed mistakes with identifying objects, static and dynamic.

Model	Prediction Time in seconds	Graph Load Time in seconds	mAP	Training Steps
ssd_mobilenet_v1	0.72	44.06	0.76	9200
ssd_inception_v2	1.76	50.18	0.78	12000
faster_rcnn_inception_v2	9.89	78.9	0.8	20000
faster_rcnn_resnet50_lowproposals	15.9	80.83	0.82	20000
mask_rcnn_inception_v2	19.4	85.96	None	None
faster_rcnn_resnet50	34.52	87.55	0.82	20000
faster_rcnn_resnet101_coco	None	None	0.83	15000

Even though the version of the raspberry pi was unclear, the prediction times show the problems that the Pi bring. The faster_rcnn model, which is most likely used, needs nearly 10 seconds to predict. This means that the FPS is roughly 0.1. Conclusion: the Pi is not capable of running a live feed through an object detection model.

Next up was to figure out if there is a work around for this problem by still using a Pi. An idea where the raspberry Pi would try to see movement and then take a picture of the moving object, was immediately shut down since computing the movement did still take too much time for it to be consistent enough plus you would probably need two raspberry Pis to first compute the movement and second to compute which object is in the picture.
Whilst confirming this it came to surface that the Pi uses Tensorflow, while in the tests Tensorflow-gpu was used on the laptop. What seemed the case is that Tensorflow runs on the CPU (Central Processing Unit) aka the processor and Tensorflow-GPU runs on the GPU (Graphic Processing Unit) aka the graphics card. Deep learning, especially using images, needs multiple cores which are present in the GPU while the CPU only has a few. Since the two are so fundamentally different the addition of the high amount of cores in the GPU highly enhances the results. The GPU is enabled by Tensorflow-GPU. The vision bonnet is the “graphical card” for the Pi developed by Google to enhance the results. Therefore the FPS on the vision kit was higher than the results found on the web. This is however the only graphical computation addition for the Pi and is still too weak. Normal graphic cards cannot be installed on the Pi and therefore its computation power cannot be enhanced, which explains the high difference between the Pi and previous tests.
With this explanation of how Tensorflow actually works and what elements slow it down, the possibilities are clearer.

Possible solutions

Any form of taking pictures and then analysing the picture of a falling object is not feasible, since the analysis would still take 10 seconds to give accurate results. So for example using the ultrasonic sensor to sense an object, then take a picture and then analyse it, would never work since only the analysis would take too long let alone the whole process. You are then also limited to the ultrasonic sensor, which have shown some inconsistency already with smaller objects. If it doesn't detect an object it would just fall past it without ever enabling Tensorflow. Also the delay of first detecting and then taking picture could lead to missing the object before the picture is taken.

Sending the photo or live feed to an external stronger pc for the analysis is too dependent on the internet connection, which in most cases is not present at all.
Using a live feed or photos from above to see the content of the bin instead of identifying a falling object leads to a complex and difficult analysis. Apart from just identifying the type of object it also has to figure out which object has been thrown into the bin last. Because of movement in the bin after something has been added and sometimes, even if it is for one frame, failure of identification it is very difficult to identify which object has been thrown in last.

An idea where photos are taken during the products usage, then uploading them to a faster pc to analyse them is a possibility if you could accurately take pictures. A complete video is of course not possible due to storage limitations. The problem that this brings forth is that the product is no longer able to identify the waste object by itself and therefore not able to give this feedback immediately to the user. The bin will always trigger

regardless of if the item should be thrown into the bin. The only addition that Tensorflow would bring is data after usage. Therefore the question rises if the data has a large enough value to outweigh the costs to implement the addition.

A more expensive option would be to build a PC or implement a complete PC/laptop in the product to get the same results as during the tests. As mentioned however this increases the costs tremendously, but this does bring live feed and feedback. If this would be the go to option the most important element of the PC would be the GPU so the graphics card as explained earlier. Tensorflow GPU uses the Nvidia cuDNN program for its GPU calculations, which needs Nvidia's CUDA. This is only possible on the Nvidia graphic cards so it then is important to look which type of card is the most accurate for this purpose.

In the laptop used to do the first set of tests a Quadro card is used. These are more expansive than the more accessible Geforce cards, but has the same results as the Geforce cards with deep learning. It is better with 3D rendering and CAD programs but since these programs aren't used here it is a better option to go for the Geforce cards. The Titan cards are specifically designed for deep learning, however are at another level in comparison to the Quadro card used in terms of cores and costs. To match the results and keeping the costs low the geforce is probably the better option. (This was all compared by the CUDA computing power provided by Nvidia. The Quadro card had 5.0, the requirement for cuDNN is 3.0. This variable was taken for the main comparison, so different results could be acquired).

Currently there are two Geforce series on the market: Geforce RTX 2060-2080 and Geforce GTX 1050-1080. The previous models with a lower but sufficient computing power are less present and therefore more expensive. The costs for a PC or laptop with the graphics card is also given.

	Computing power	From €	RAM from	PC €	Laptop €
Geforce RTX 2060 - 2080	7.5	360	6 GB	1500	2000
Geforce GTX 1050 - 1080	6.1	147	4 GB	800	1000

The PC and laptop are still quite expensive and costs can probably be cut if a homemade PC is built. There is a large variety for every needed element. The CPU starts from an Intel i5 core up to i9, which is next to the GPU the main element for the PC. The motherboard and the CPU cooler need to be compatible with the CPU which is determined by its socket. Lastly the power supply unit (PSU) needs to be able to deliver the total needed Watt for the CPU, coolers and GPU.

	Min €	**Max €**
Motherboard	68,25	273
CPU	127,4	445,9
CPU Cooler	18,2	91
RAM 8 GB	27,3	45,5
NVME SSD (250 GB - 2000 GB)	27,3	433,16
PSU (watch needed watt)	27,3	91
Windows	113,75	113,75
Case	45,5	45,5
GPU	147	1180
Total	602	2718,81

A minimum cost for a custom build pc would be around 600 euros, however since I am no expert on the field and do not know the exact differences between the Quadro cards and the Geforce cards I do not know if the cards give the same results. Since there are many variables that could be of influence, like the RAM of the PC, the RAM of the GPU and the cores in the GPU and many more, so the outcome could differ greatly which is difficult to predict. Also the bin has to be wired again since the implemented PC needs to be connected to the power grid.

Nvidia Jetson

The Pi was not a feasible option for the detection, however there is a different microprocessor that is designed for deep learning by Nvidia: The Nvidia Jetson series. These products do contain the multiple cores like the normal graphic cards. The cheaper version is around 100 euros which makes this a more viable option than the custom PC, however after a quick investigation about its performance it became clear that the FPS is only up to 10 FPS and the model used is still the problematic SSD Mobilenet. Even though the performance is better than the Pi it seems still too low for it to function consistent enough. This is probably since deep learning on images is a difficult process that needs more power than other deep learning processes.

There are more advanced models in the Jetson series, but naturally the price goes up with these models next to the fact that the results are unclear with the faster rcnn model. This is however still a probably better option to go for than the custom PC since it is much smaller if the results are consistent enough. Right now the costs are still too high to implement in the products since the model that could run Tensorflow-GPU costs 800 euros, but the development of this line of microprocessors is something to watch and keep track off.

Setup Tensorflow

Requirements

- Windows 10
- Nvidia GPU (GTX 650 or newer, but better to start from GTX 1050 ti)

Install CUDA Toolkit

- https://developer.nvidia.com/cuda-downloads?target_os=Windows&target_arch=x86_64

Install cuDNN

- Create an account on the Nvidia developer membership: https://developer.nvidia.com/rdp/cudnn-download
- Download the cuDNN file that matches the CUDA version

Install Anaconda

- https://www.anaconda.com/distribution/

Every line in italic is a command that needs to be entered in an anaconda prompt

Install Git

- https://git-scm.com/download/win

Install Visual C++ Build Tools

- https://go.microsoft.com/fwlink/?LinkId=691126
- Run it and install with default options

Install Pycoco/Coco API tools

- https://github.com/philferriere/cocoapi
- *python --m pip install --upgrade pip*
- *pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI*

Download the Tensorflow package

- Create a folder named tensorflow in C:/Users/"youruser"/tensorflow1
- Download the full package and implement it in this folder https://github.com/tensorflow/models
- Rename the directory "models-master" to just "models"

Download the faster_rcnn_inception_v2_coco model from the model zoo

- https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
- Open the .tar.gz file with WinZip or 7-Zip
- Extract the folder to ~\tensorflow1\models\research\object_detection

Extract the object_detection_Ewaste folder in the /object_detection/

Alter the .config file located in ~/object_detection/training (A nice program to easily do this is notepad++: https://notepad-plus-plus.org/downloads/):

- Line 106: Check if the faster_rcnn model name in the path matches the model that was just downloaded and change it to match the model.
- Line 106, 123, 125, 135 and 137: change the paths to match your directory, probably by just changing the username (which is 20174016 now)

Set up a new Anaconda virtual environment

- *conda create --n tensorflow1 pip*

Activate the environment, update pip and install tensorflow-gpu

- *activate tensorflow1*
- *pip install --ignore installed --upgrade tensorflow-gpu*

Check numpy. Tensorflow atm does not run on the newest numpy but on 1.16.4. Check if this still is the case.

- *pip uninstall numpy*
- Until no version on numpy is present
- *pip install numpy==1.16.4* (or the current needed version)

Install the other necessary packages

- *conda install -c anaconda protobuf*
- *pip install pillow*
- *pip install lxml*
- *pip install Cython*
- *pip install contextlib2*
- *pip install jupyter*
- *pip install matplotlib*
- *pip install pandas*
- *pip install opencv-python*

Configure PYTHONPATH environment variable

- *set PYTHONPATH=C:\Users\"Name"\tensorflow1\models; C:\Users\20174016\tensorflow1\models\research;C:\Users\20174016\tensorflow1\models\research\slim*
- *set PATH=%PATH%;PYTHONPATH*
- To check: *echo %PYTHONPATH%*

Compile protobufs and run setup.py

- Change directories to tensorflow1: cd C:\Users\"usernaam"\tensorflow1\models\research
- *protoc --python_out=. .\object_detection\protos\anchor_generator.proto .\object_detection\protos\argmax_matcher.proto .\object_detection\protos\bipartite_matcher.proto .\object_detection\protos\box_coder.proto .\object_detection\protos\box_predictor.proto .\object_detection\protos\eval.proto .\object_detection\protos\faster_rcnn.proto .\object_detection\protos\faster_rcnn_box_coder.proto .\object_detection\protos\grid_anchor_generator.proto .\object_detection\protos\hyperparams.proto .\object_detection\protos\image_resizer.proto .\object_detection\protos\input_reader.proto .\object_detection\protos\losses.proto .\object_detection\protos\matcher.proto .\object_detection\protos\mean_stddev_box_coder.proto .\object_detection\protos\model.proto .\object_detection\protos\optimizer.proto .\object_detection\protos\pipeline.proto .\object_detection\protos\post_processing.proto .\object_detection\protos\preprocessor.proto .\object_detection\protos\region_similarity_calculator.proto .\object_detection\protos\square_box_coder.proto .\object_detection\protos\ssd.proto .\object_detection\protos\ssd_anchor_generator.proto .\object_detection\protos\string_int_label_map.proto .\object_detection\protos\train.proto .\object_detection\protos\keypoint_box_coder.proto .\object_detection\protos\multiscale_anchor_generator.proto .\object_detection\protos\graph_rewriter.proto .\object_detection\protos\calibration.proto .\object_detection\protos\flexible_grid_anchor_generator.proto*
- *python setup.py build*
- *python setup.py install*

Test TensorFlow setup to verify it works

- *jupyter notebook object_detection_tutorial.ipynb*
- Run the jupyter notebook. If it is working correctly at the end two labeled images should appear

Download LabelImg

- https://github.com/tzutalin/labelImg
- Open another Anaconda Prompt
- Navigate to where the folder is: *cd "path"/labelImg*
- *conda install pyqt=5*
- *pytcc5 –o libs/resources.py resources.qrc*
- This should start the program: *python labelImg.py*

Setup training

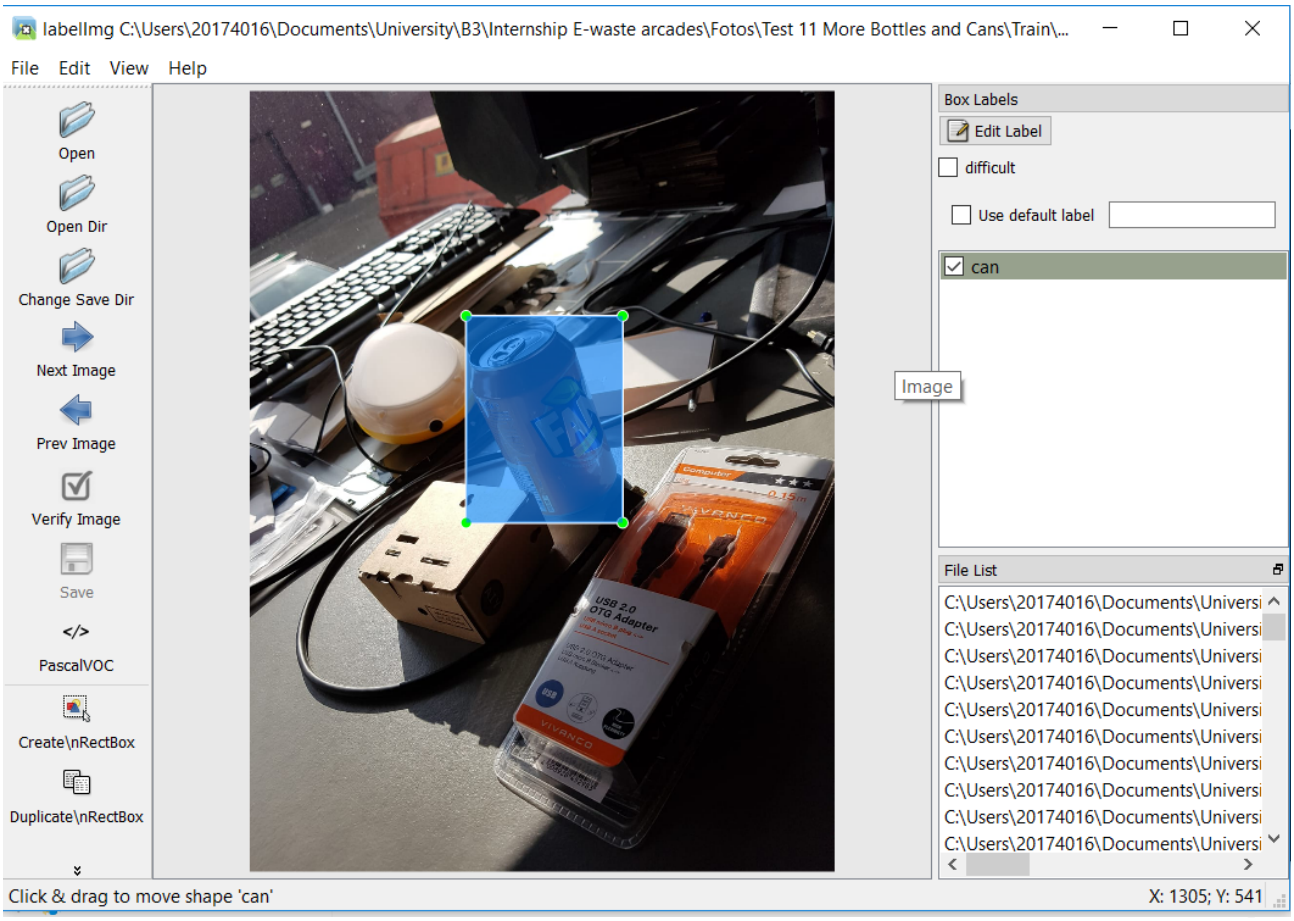
The lines put in italic are commands that need to be put in an anaconda prompt

1. Start with activating the environment
activate tensorflow1

2. Set the environment variables
set PYTHONPATH=C:\Users\20174016\tensorflow1\models; C:\Users\20174016\tensorflow1\models\research;C:\Users\"Name"\tensorflow1\models\research\slim
set PATH=%PATH%;PYTHONPATH
To check: *echo %PYTHONPATH%*

3. Label the images in storage, not in the tensorflow1 folder. In the picture the program is visible where a can is labeled by the box around the can and the label on the right
cd C:\Users\"username"\labelimg
(might need: *conda install pyqt=5*)
python labelImg.py
Open directory

4. Label all images for train and test



5. Remove the following files
The images and xml files from ~\object_detection\images\train and test
The csv files from ~\object_detection\images
The .record files from ~\object_detection
All the files from the inference_graph folder

6. Copy the needed images and xml files in ~\object_detection\images\train and test

7. In the prompt window navigate to object_detection
cd C:\Users\"username"\tensorflow1\models\research\object_detection

8. Convert the xml files to csv in prompt
python xml_to_csv.py

9. Alter and save the labelmap.pbtxt for every label used such as:
item {
    id: 1
    name: 'can'
}
item {
    id: 2
    name: 'similar'
}
10. Alter and save the generate_tfrecords.py with notepad++ to match the labelmap such as:
if row_label == 'can':
    return 1
elif row_label == 'similar':
    return 2
else:
    None

11. Generate tfrecords in prompt for both test and train
python generate_tfrecord.py --csv_input=images\train_labels.csv --image_dir=images\train --output_path=train.record
python generate_tfrecord.py --csv_input=images\test_labels.csv --image_dir=images\test --output_path=test.record

12. Alter the faster_rcnn_inception_v2_pets.config or ssd_mobilenet_v2_coco.config in notepad++
Num_classes should match the amount of different labels
Num_examples should match the amount of images in ~\object_images\images\test

13. Run the training in the prompt
python model_main.py --pipeline_config_path=training\faster_rcnn_inception_v2_pets.config --model_dir=images --num_train_steps=50000 --sample_1_of_n_eval_examples=1 –alsologtostderr

14. Open tensorboard in another prompt with:
activate tensorflow1
tensorboard --logdir=C:\Users\"Youruser"\tensorflow1\models\research\object_detection\images --host localhost --port 8088

15. Produce the inference_graph in prompt where XXXX is the last saved checkpoint in ~\object_detection\image
python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/faster_rcnn_inception_v2_pets.config --trained_checkpoint_prefix images/model.ckpt-XXXX --output_directory inference_graph

16. Copy the inference graph folder to storage

Use object detector

Alter and run the Object_detection_webcam1.py by:
- *activate tensorflow1*
- *cd C:/Users/"Youruser"/tensorflow1/models/research/object_detection*
- *idle*
- Open Object_detection_webcam1.py
- Change num_classes to match the number of labels
- Run Object_detection_webcam1.py
- Press q to exit the webcam

It will use the frozen inference graph that is located in the object_detection/inference_graph. If you want to use an older model simply replace the folders content by the inference_graph folder from storage that you want to use.

Change model type

Download the desired model from:
- https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Extract the folder and move it to the ~/object_detection folder

Find the config file that suits the model in
- ~object_detection/samples/config

Copy the file to ~object_detection/training

Alter the file
- Num_classes to the amount of labels

- Fine_tune_checkpoint to path to the model.ckpt which is in the model folder that was just downloaded

- Train_input_reader input_path to path to the train.record file
    - The stated name can be altered but it has to end in .record

- Label_map_path to path to the labelmap.pbtxt which is in the training folder

- Eval_config num_examples to match the amount of images in ~/object_detection/images/test

- Eval_input_reader input_path to path to the test.record file
    - The stated name can be altered but it has to end in .record

- Label_map_path to path to the labelmap.pbtxt which is in the training folder

Test log

Test 1 Fanta blikje
- 100 foto's Fanta blikje
- Mobilenet model
- Te weinig foto's / te traag model waardoor alles wordt herkend als blikje

Test 2 Meer foto's Fanta en ander model
- 360 foto's Fanta blikje
- Faster_rcnn model
- Fanta blikje wordt ruwweg altijd herkend
- Andere blikjes worden al redelijk herkend, maar bepaalde hoeken minder
- Andere cilinder vormige objecten worden ook als blikje gezien
    - --> Test 3
- Bepaalde hoeken zorgen voor moeite model
- Fel tegenlicht zorgt voor problemen
- Vasthouden met hand zorgt voor problemen

Test 3 extra label similar
- 360 foto's Fanta blikje
- 188 foto's andere objecten (drinkfles, kopjes, glazen)
- Andere objecten en blikjes al beter gescheiden
- Nieuwe objecten krijgen soms beide labels maar vergelijkbare objecten krijgen allen label similar (drinkfles)
- Nieuwe blikjes (bijv cola) krijgen beide labels vanwege waarschijnlijk kleur andere object
    - -->Test 4
- Niet ingeprogrammeerde andere objecten zorgt voor problemen, mss op te lossen door meer andere objecten te implementeren
- Meer foto's van bovenaanzicht blikjes nodig, similar kreeg nu de bovenhand
- Mss is label similar nu te breed door de variatie in objecten en daardoor niet stabiel
- Mss dat label similar niet nodig is als de andere objecten al wel in de foto's staan maar zonder label
    - --> Test 5

Test 4 Andere blikjes
- 596 foto's blikjes cola, fanta, ict, cola l
- 188 foto's andere objecten
- Kleur lijkt nu niet als herkenningsvariabele gebruikt te worden, cola beter herkent
- Gevarieerdere dataset voor blikjes helpt

Test 5 Label similar removed

- 596 foto's blikjes cola, fanta, ict ,cola l
- 188 foto's andere objecten zonder label
- Andere objecten buiten drinkfles geen label gekregen
- Nieuwe objecten geven nog af en toe problemen, wss op te lossen door deze objecten mee te nemen in de foto's
- Similar label is niet nodig als de andere objecten maar wel in de dataset staan

Test 6 PET

- 302 foto's van ict + ictgr
- Basicly dezelfde resultaten als test 2 maar dan met flesjes
    - --> Test 7

Test 7 PET + blik

- 302 foto's van ict + ictgr
- 596 foto's van blikjes
- 188 foto's van andere objecten
- Blikjes beter herkend dan PET, mss door meer foto's mss door het transparante plastic
- Andere objecten door dubbel label minder vaak benoemd
- Af en toe overlap tussen PET en blikjes aangezien de labels op de verpakking ruwweg hetzelfde zijn, op te lossen via code ipv fotos wss

Test 8 Prullenbak setting zonder beweging

- 490 foto's van fanta, cola, cola l en andere objecten in
- Constante achtergrond en andere lichtinval maar wel stilstaand
- Ondanks het grote aantal foto's herkende het model alles buiten de rustige achtergrond van de bak als blikje dus veel slechtere resultaten ondanks dat zowel de andere objecten als blikjes gefotografeerd zijn
- In de fotocamera was alles duidelijk te zien, maar voor de webcam was de omgeving te donker om het blikje of een ander voorwerp te herkennen

Test 9 Prullenbak setting zonder beweging oud model

- Model van test 5
- Setting van test 8
- De webcam vind de omgeving nog steeds te donker om iets te herkennen, met behulp van een zaklamp of een andere lichtbron herkent ie de blikjes wel
- In tegenstelling tot test 8 herkent het model de blikjes nu wel en andere objecten niet, dus bij het trainen van het model moeten de foto's genomen worden in een rumoerige wisselende achtergrond ook al wordt de detector maar in een bepaalde setting gebruikt

Test 10 Prullenbak setting zonder beweging PET + blik

- Model van test 7
- Setting van test 8
- De grootte van de flesjes kan een probleem zijn aangezien de flesjes meestal helemaal in het beeld van de camera moet komen om goed herkent te worden
- De flesjes worden nog vaak herkent als blikje, aantal foto's van flesjes moet omhoog

Test 11 Meer foto's PET, PET + blik

- 610 foto's van ict, ictgr, cola z pet
- 596 foto's van blikjes
- 188 foto's van andere objecten
- Zowel blikjes als flesjes worden best goed herkend
- Met de achterkant van het ice tea pet label heeft het model moeite
- Gestapelde blikjes worden af en toe ook foutief herkent als flesje
- Andere blikjes en flesjes worden ook al goed herkend
- Ook in de setting worden beide goed herkend maar blijft oppassen met grootte flesjes en ice tea struggles

Test 12 Movement 1ste test

- 350 foto's van vallend fanta flesje
- Elke beweging wordt gezien als flesje
- Elke beweging wordt gezien :D
- Enkele keren valt het flesje sneller dan de frame rate, later testen met hit rate om te kijken of dit een probleem is
- In een statische achtergrond herkent het programma niet tot zelden een flesje, af en toe wel andere objecten die statisch toch herkend worden
- Toevoegen andere vallende objecten om te testen of het model vallende objecten uit elkaar kan halen
- Mix van statische en dynamische objecten, zowel foutief als goede, beter?
- Toevoeging van vallend en statisch label nuttig? Of niet haalbaar door alle herkenning van beweging?

Test 13 Movement + static data set

- 350 foto's van vallend fanta flesje
- 610 foto's van ict, ictgr, cola z pet
- 188 foto's van andere objecten
- In de statische achtergrond wordt minder foutief herkend dan bij test 12
- Vallende objecten worden minder snel gespot

Test 14 Static Dynamic labels

- 350 foto's van vallend fanta flesje met label "dBottle"
- 610 foto's van ict, ictgr, cola z pet met label "Bottle"
- 188 foto's van andere objecten
- Andere objecten op statische achtergrond minder herkent
- Vallende flesjes ziet het programma weer beter
- Andere vallende objecten/ bredere, sterkere data voor flesjes nodig

Test 15 Plastic bekers

- 300 foto's van plastic bekers
- Bekers worden prima herkend in stilstand
- Uitvoerigere dataset aan foto's kan daarentegen nooit kwaad
- Door het ontbreken van andere objecten worden deze af en toe wel eens herkent als beker

Test 16 Cup, Can, Bottle and dBottle

- 350 foto's van vallend fanta flesje met label "dBottle"
- 610 foto's van ict, ictgr, cola z pet met label "Bottle"
- 188 foto's van andere objecten
- 300 foto's van plastic bekers
- 596 foto's blikjes
- Bekers worden zwak herkend vanaf de zijkant, andere hoeken juist goed
- Flesjes stilstaand worden goed herkend in staande positie, zijkant minder maar goed genoeg. Onder en bovenkant worden daarentegen herkend als beker, wss vanwege het plastic
    - Is dit een probleem? Of is de aanname dat het flesje er altijd staand in gaat voldoende?
- Propjes plastic (verpakkings plastic) niet herkent
- Drinkfles die eerst niet werd herkent, wordt nu herkend als dBottle
- Andere bewegende objecten zoals een blikje worden nu ook herkend als dBottle
    - Implementatie vallende foto's van cups, cans en andere objecten

Test 17 dCup

- 270 foto's van vallende bekers met label 'dCup'
- 350 foto's van vallend fanta flesje met label "dBottle"
- 610 foto's van ict, ictgr, cola z pet met label "Bottle"
- 188 foto's van andere objecten
- 300 foto's van plastic bekers
- 596 foto's van blikjes
- Als de vallende bekers in het beeld komen dan worden ze wel opgepakt, maar alleen staand dus onderkant onder. Het vallende bekertje zal dus ook van andere aanzichten gefotografeerd moeten worden voor betere resultaten.
- Een ander probleem dat hierbinnen optreed is het feit dat de camera af en toe geen enkel frame van het vallende object laat zien, oftewel de frame rate is te laag. Hetzelfde probleem bij het flesje: Als het flesje in het scherm komt dan wordt hij geïdentificeerd, maar kan er ook zomaar voorbij vliegen zonder enig moment in de camera.
- Verder zijn er nog steeds af en toe problemen met blikjes en flesjes vanwege dezelfde soort labels

Test 18 Webcam

- Model test 17
- Heeft een andere webcam een hogere frame rate en ligt het dus aan de camera?
- Ligt niet aan de webcam maar wss aan de combinatie van type model en cv2

Test 19 SSD Mobilenet

- 270 foto's van vallende bekers met label 'dCup'
- 350 foto's van vallend fanta flesje met label "dBottle"
- 610 foto's van ict, ictgr, cola z pet met label "Bottle"
- 188 foto's van andere objecten
- 300 foto's van plastic bekers
- 596 foto's van blikjes
- ssd_mobilenet_v2_coco.config
- Laptop ergens in de nacht gecrasht en daarom geen idee over de trainingstijd en loss rate. Wel zeker 12 uur getraind
- Frame rate duidelijk veel hoger
- In vergelijking met de eerste test waar alles een fanta blikje was, werd er al een stuk minder herkend
- Ondanks dat werden nog steeds andere objecten foutief gezien als sommige labels en ook objecten die herkend moesten worden kregen soms meerder labels of geen label
- Ook in een rustige constante achtergrond kwamen dezelfde fouten voor
- Vallende objecten werden simpelweg niet herkend
- Frame rate van de webcam of cv2 zijn dus niet de vertragende factors bij de voorgaande tests, maar de code en het type model, faster_rcnn
- Het is maar de vraag of een ssd_mobilenet model de vereiste resultaten neer kan zetten. Op dit moment betwijfel ik dat vanwege de lange trainingstijd, onzekerheid van benoemen en geen vallende objecten herkennen