AI for object detection and tracking in subterranean environments

Exercises

[Introduction 3](#_Toc69300678)

[Goals 3](#_Toc69300679)

[Methodology 4](#_Toc69300680)

[Research 4](#_Toc69300681)

[Data collection 7](#_Toc69300682)

[Training and benchmarking 9](#_Toc69300683)

[Discussion 10](#_Toc69300684)

[Tools 10](#_Toc69300685)

[Bibliography 10](#_Toc69300686)

[Results 11](#_Toc69300687)

# Introduction

This project aims to explore the capabilities of different state-of-the-art object detectors in the task of detecting some of the DARPA Sub-t Challenge artifacts from image data.

The DARPA Subterranean Challenge aims for the localization of various artifacts in subterranean (subT) environments by unmanned vehicles. Artifacts are elements of interest that can reasonably be found in such places. Each of these artifacts has a specific interest in subT exploration, so that its finding would provide relevant information under high-risk situations in unpredictable and dangerous subT spaces. Figure 1 shows the physical artifacts defined in this challenge. Full list available at [1].



Figure 1: DARPA SubT Artifact Detection Challenge artifacts.

Source: [1]

Since this work aims to perform detection using CNN-based object detection pipelines, i.e. image data, the items cell phone, CO2 source and ventilation duct were removed, since their detection should be performed with additional sensory.

# Goals

The main goals of this work are defined as follows:

* Provide the author with knowledge on the Computer Vision techniques supported on convolutional neural networks that had led the development of the state-of-the-art until today.
* Build a representative dataset for the objects to be detected by means of gathering existing / generating new data.
* Compare several state-of-the-art object detection models and discuss their performance in this particular application considering a potential robot implementation in an on-board computing device.

# Methodology

This project work can be classified in three stages.

* **Research**: Gather knowledge on the field of neural networks and object detection. Identify state of the art object detection models and search for implementations backed-up by original research papers that are available for use.
* **Data gathering**: Produce different datasets to train a model able to identify a specific set of items from pictures.
* **Training and benchmarking**: Train different neural network models on the gathered data and evaluate and compare their performance on the object detection task.

Automation and overall scripting was mainly performed with scripts in python [1] and bash [2]. Some cloudcomputing was done by using google colaboratory [3]. A Github repository was used for storage at [4]. More technical tools will be later introduced through this work.

# Research

Prior to this work, the authors knowledge on neural networks reached only practical applications of small Densely-connected and CNN models, the most complex task performed being the transfer learning of Alexnet to a binary image classification.

Early stages of research were meant to build some foundation on the field of object detection. Opposite to what was known at the moment, object detection is not perform by only stacked layers of neurons, but by elaborate and well-designed information pipelines that use neural networks for feature extraction and classification, and have evolved over time to increase performance and optimize computational burden, which is usually very high.

Overall, there are two kinds of object detectors [2]:

* One stage pipelines: which perform region proposal and object detection in the same “stage”. Modern solutions typically use a set of premade anchor boxes that “sweep” the image looking for possible objects. Some examples in this category are SDD, RetinaNet or YOLO. These models are faster but tend to be less accurate.
* Two stage pipelines: which perform region proposal in a separate stage and then perform detection operations. Some examples in this category are R-CNN, Faster-RCNN and RFCN. These tend to be slower but more accurate.

Later stages of research focused on more practical issues, such as a review on the performance metrics that are meaningful towards comparing object detection pipelines or the very model selection to decide which models are to be trained towards building a benchmarking table.

The most common metric for evaluating detectors is the Average Precision or AP. The AP is computed by inferring object detection on a batch of test data as the area under the Precision-Recall curve. This requires introducing the concept of true and false positives and negatives, as well as IoU:

* False positives: A false positive is called when a detection is either made for a ground-truth bounding box that had already received one (detection duplicates) or when a detection does not meet an lower-bound IoU.
* True positives: Which are those predictions made for the correct class, without duplicates and meeting a high-enough IoU with the ground-truth annotation.
* False negatives: When an item is either predicted with the wrong class or on background scenery.
* True negatives: which are instances of background that are ignored, thus not classified as objects.
* The IoU: or intersection over union, is an scalar value of intersection for two bounding boxes. The closer to one, the most similar the bounding box coordinates are. Graphical representation is shown in Figure 2.

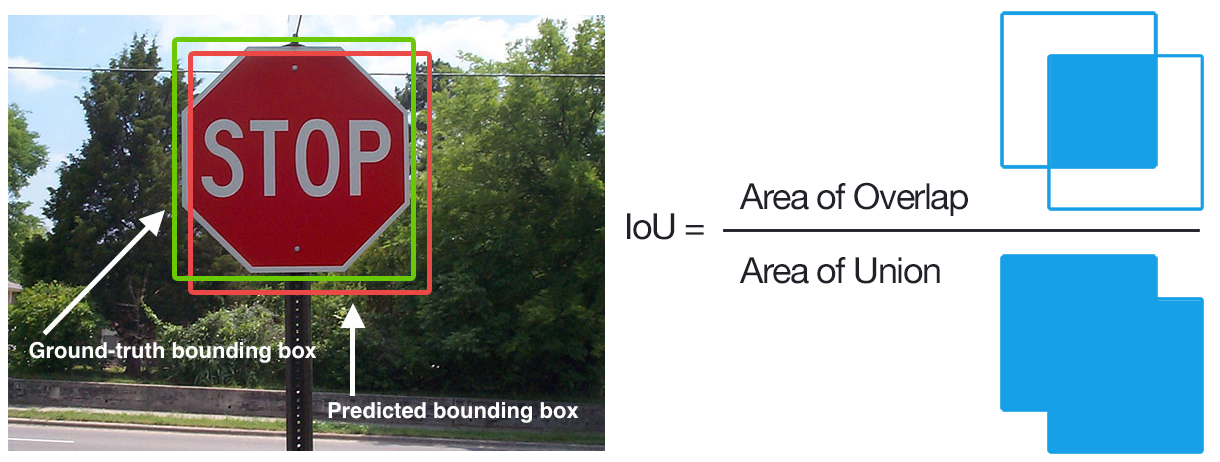


Figure 2: Jaccard Index or IoU

Source: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

The AP is dependent on the minimum IoU required for a detection to be considered spatially accurate. A more general way of this metric is the mean Average Precision or mAP, which consists on averaging different AP values within two IoU extremes. Common AP metrics are:

* AP@0.50 or AP@50: computed for a IoUthreshold of 0.50.
* AP@0.75 or AP@75: computed for a IoUthreshold of 0.75.
* AP@0.50:0.05:0.95 or simply mAP: Average from IoUthreshold 0.5 to 0.95 in steps of 0.05.

In this work, metrics are computed using the MS COCO tools at [5].

The plot depicted in Figure 3 shows a benchmarking of various state-of-the-art scalable CNN-based object detectors on the MS COCO dataset [6]. In FIGURE, the horizontal axis represents inference time (left is faster) and the vertical axis the mAP (higher is more accurate).

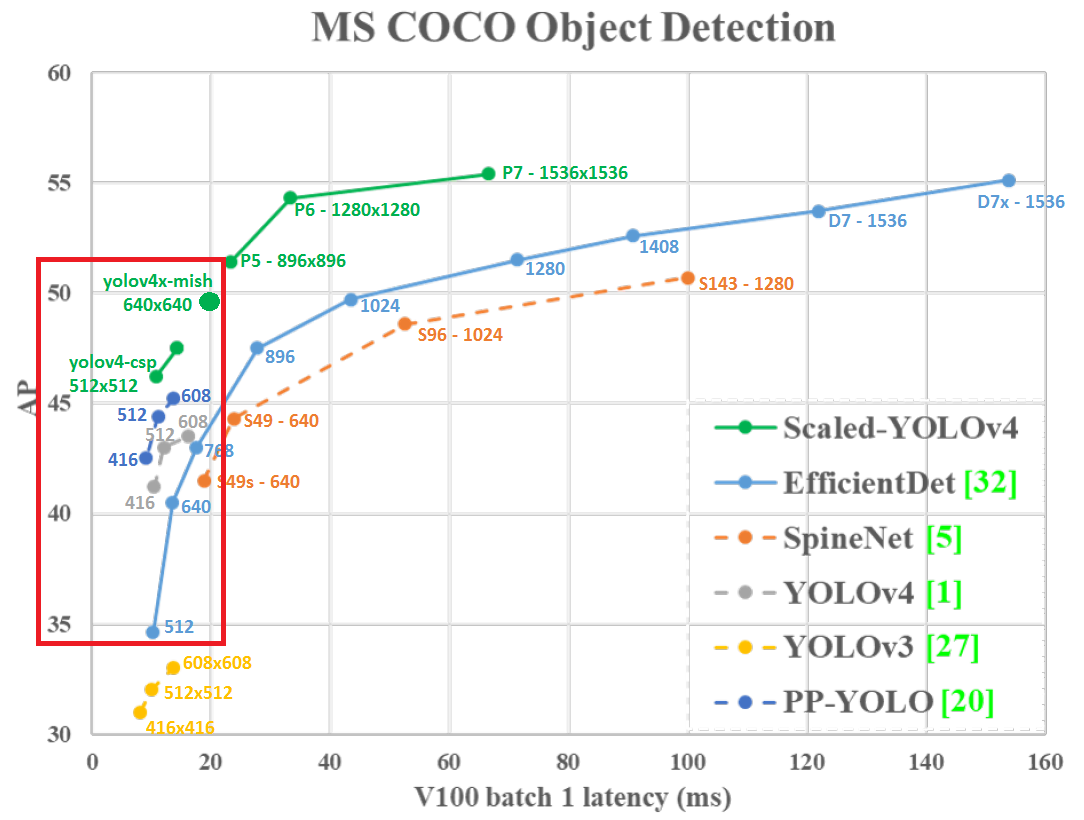


Figure 3: Benchmarking of state-of-the-art object detection models on the MS COCO dataset.

Source: [6]

The model families contained in this plot are scalable, meaning that there are various available baseline sizes for each of their architectures, defined in their respective research papers [3] [4] [5] [6]. The numbers shown for each model in FIGURE represent its design baseline input resolution. Big models, such as EfficientDet-D7 with an input resolution of 1536x1536 pixels, are meant for running object detection on high quality pictures on very powerful GPUs. This work focuses on on-board computations and thus aims for smaller models, mostly within the red bounding box in Figure 3.

There also exist the so-called portable models, which are much smaller networks meant precisely for computations of low end devices. These are expected to have low performance but high inference speed. Table 1 holds the models of interest that have been chosen towards performing a benchmarking.

Table 1: Models of interest to be benchmarked

|  |  |  |
| --- | --- | --- |
| **Model name** | **Platform** | **Network size** |
| yolov4-tiny | darknet | 416x416 |
| yolov4-tiny-3l | darknet | 416x416 |
| yolov4 | darknet | 416x416 |
| yolov4-csp | darknet | 512x512 |
| yolov4x-mish | darknet | 640x640 |
| efficientdet-d0 | google-automl | 512x512 |
| efficientdet-d1 | google-automl | 640x640 |

Official implementations for these models were fetched for the platforms darknet (*yolo* networks) and google-automl (*efficientdet* networks), available in the repositories [6] and [7]. During the model selection stage, quick training and inferring tests were made on both platforms for a set of arbitrary data to check that the computing skills of the author were enough for committing to train the selected models.

# Data collection

Towards this work, all networks will be pre-trained on MS COCO dataset, so that after some field work and from the conclusions of [7] it is estimated that a total of 400 instances of each DARPA artifact should suffice. In the beginning, there was no access to real pictures of all of the artifacts, so some experimentation on producing synthetic data was performed. Later, real DARPA artifacts became available to take and label pictures, being then available real images of all artifacts.

Figure 4 shows the roadmap for the data collection that was performed towards this work. The two left branches hold tasks for creating synthetic data, the discontinued branch following an approach as described in Cut Paste Learn [8] from the CAD models available at [8], the other for generating photorealistic imaging with the rendering tool SynthDet [9] and the same models following [9], used only as training data in some of the experiments. The other two branches include the tasks performed for importing pictures from the Image-Segmentation-annotated PST dataset [10] and for generating new pictures ourselves with a camera and a labeling tool.

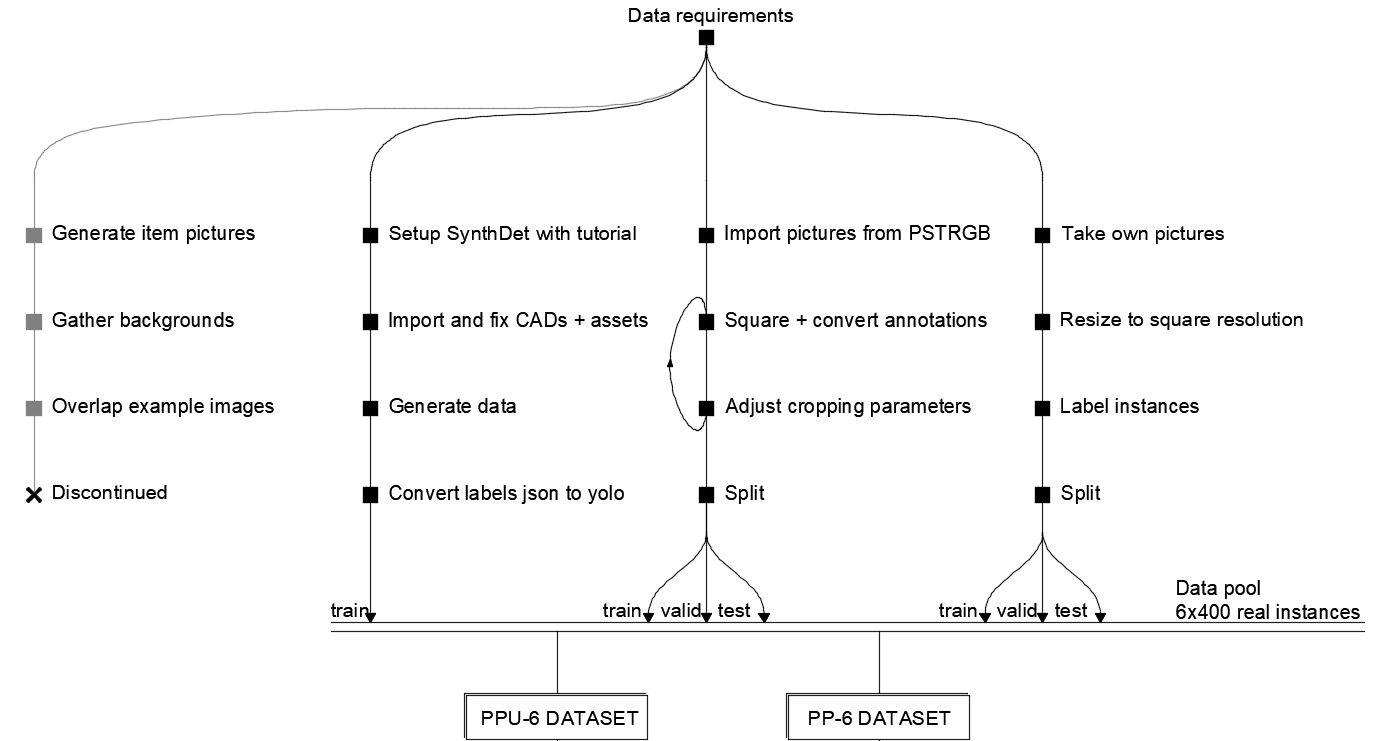


Figure 4: Roadmap fot the data collection performed in this work.

Source: Self-made

The main output dataset from this work is the YOLO-format-annotated PPU-6 dataset, a showcase of which is displayed on Figure 5. The PP-6 mostly consists on the same pictures, only removing all synthetic examples from the PPU-6 train set. The PPU-6 dataset is composed by:

* ~400 instances of the items backpack, drill, extinguisher and survivor borrowed from the PST dataset.
* ~400 instances of rope and helmet artifacts, in pictures taken by the author at the LTU facilities.
* A set of virtual pictures providing ~100 instances of each artifact produced with Unity SynthDet [9].

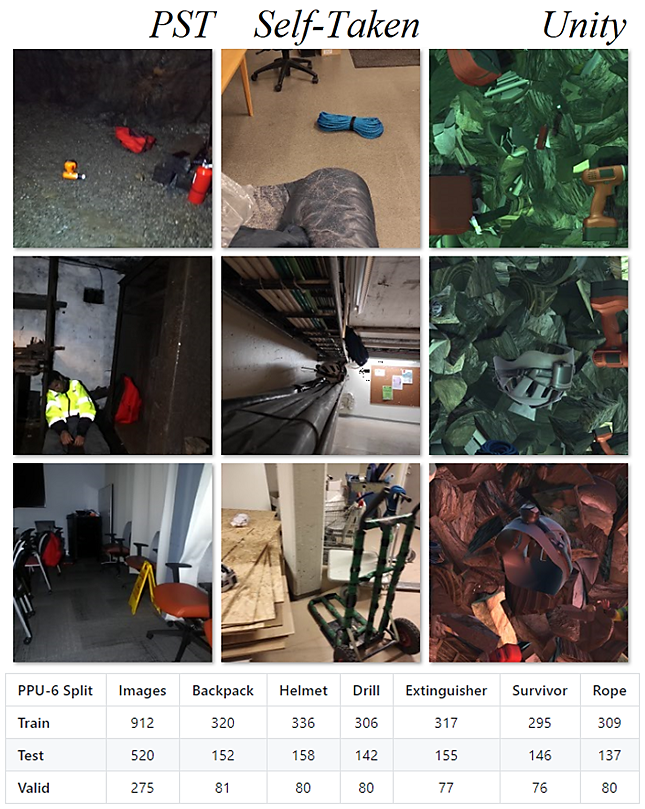


Figure 5: Showcase of the pictures in the PPU-6 dataset, column-ordered by source

Source: Self-made

# Training and benchmarking

The training and benchmarking pipeline is highly platform-dependent and the overall roadmap for it is shown in Figure 6. While this is an example for the PPU-6 dataset, the steps are representative for any other YOLO-annotated dataset. These steps are repeated to train and benchmark the models in each of the performed experiments.

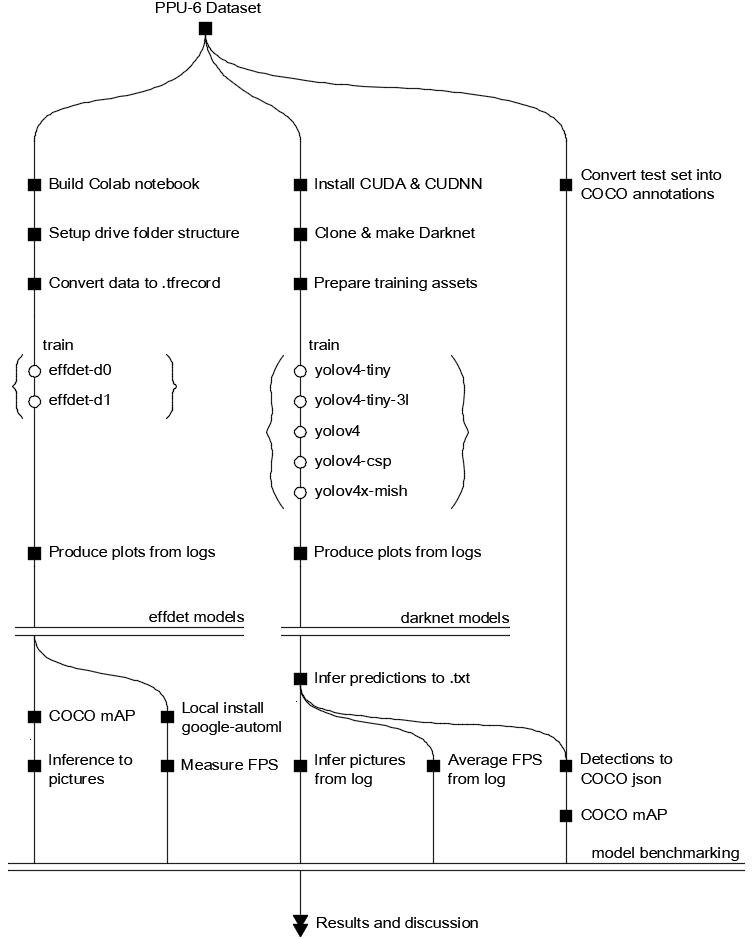


Figure 6: Training and benchmarking pipeline for the PPU-6 dataset.

Source: Self-made

# Results

* **Stage 1: Training all models on default PPU-6**:
  + Default out-of-the-box training on official models. (Figure 7)
  + Observations:
    - Tiny models outperform regular sizes (overfitting suspicion)

⟶ *retrain on only real data*

* + - YOLOv4-tiny outperforms YOLOv4-tiny-3l

⟶ *deeper experiments with anchor sizes / layer attribution*

* + - YOLOs blow efficientdets on few-class detections

⟶ *discard efficientdets from further experiments*

* **Stage 2: Training some darknet models on PP-6**
  + Removed synthetic training data to check if overfitting was taking place in the bigger models.
  + Experimented with changing resolutions

[Train size, Network size, Test image size] =

* + - [train\_size, train\_size, train\_size] (Figure 8)
    - [train\_size, train\_size, 640x640] (Figure 9)
    - [train\_size, 640x640, 640x640] (Figure 10)
  + Observations:
    - Variying results, tiny models still predominant

⟶ *no overfitting. they aresimply best, architecture-wise, for this 6-class task*

* + - YOLOv4-tiny-3l now outperforms YOLOv4-tiny

⟶ *deeper experiment with anchor sizes / layer attribution*

* **Stage 3: Adjusting anchor sizes on best performing models**. Anchors not only impact expected aspect ratio but overall size. All yolo layers have to carry same anchors value, the masks= index is what decides what anchors are linked to each yolo layer. (Figure 11)
  + Modified anchors:
    - yolov4-tiny (2 yolo layers + 6 anchor pairs).
    - yolov4-tiny-3l (3 yolo layers + 9 anchor pairs).
  + Observations:
    - Detected a mistake in the original yolov4-tiny.cfg because mask=0 was not attributed to any layer. Should not have impact since PPU-6 has no items of that size.
    - The performance of both is now similar and makes sense.
    - **yolov4-tiny is simply the best architecture for this task**,according to the performed experiments.

|  |  |
| --- | --- |
| 113900765-74965b00-97ce-11eb-9e17-be0ff010c8b4.png (1300×500)  Figure 7: Stage 1 AP results for the PPU-6 dataset  Source: Self-made | |
| 113900759-73652e00-97ce-11eb-978c-cb6c536b9172.png (900×500)  Figure 8: Stage 2.1 AP results for the PP-6 dataset  Source: Self-made | 113900764-73fdc480-97ce-11eb-9629-cebc75e1ad7b.png (900×500)  Figure 9: Stage 2.2 AP results for the PP-6 dataset  Source: Self-made | |
| 114569558-62f0ff80-9c75-11eb-8189-e35090543ea3.png (900×500)  Figure 10: Stage 2.3 AP results for the PP-6 dataset  Source: Self-made | **114568366-59b36300-9c74-11eb-9c11-a05b4e7334c8.png (800×500)**  Figure 11: Stage 3 AP results for the PPU-6 dataset  Source: Self-made |

# Discussion

Results were, overall, unexpected. The first stage of training on the complete PPU-6 dataset, meant to test all of the models out-of-the-box, showed that the best pipelines were the smallest *yolov4-tiny* and *yolov4-tiny-3l.* At first, it was though that there may have been some overfitting on the bigger networks, which could have come from the synthetic data graphics, since smaller models are less prone to overfitting.

This suggested repeating the training after removing these data (PP-6 dataset), which was done only to observe varying results that didn’t lead to solid proof of overfitting. In this second experiment, the chance to play around with network and image resolution was taken, expecting performance to improve [6], which in this case, didn’t.

A third stage of training led to modifying the anchor sizes of the *yolov4-tiny* and *yolov4-tiny-3l,* since their training is fast and they proved to be the best models up to then. Anchor sizes were recomputed using darknet tools and the architectures modified, to find both models perform reasonably equally and confirming suspicions from the first stage that the performance gap between these two had been too big.

Overall all of the YOLOs perform quite good, not the efficientdets, since it appears that YOLOs have an easier time with smaller classification tasks, while the efficientdets show the same order of mAP than for the MS COCO dataset. Within YOLOs, AP@50 are usually very high, meaning that the detection capabilities are very good, but not so much when it comes to accurate bounding box regression, since mAP and AP@75 are usually significantly smaller.

**From these experiments, it has been proven that the best model for this detection task is the *yolov4-tiny****,* despite being a portable, small-size model. This is thought to simply be the best model architecture-wise for this 6-class problem.

# Tools & external resources

[1] “Python 3.0 Release | Python.org.” https://www.python.org/download/releases/3.0/ (accessed Apr. 14, 2021).

[2] “Bash - GNU Project - Free Software Foundation.” https://www.gnu.org/software/bash/ (accessed Apr. 14, 2021).

[3] “Welcome To Colaboratory - Colaboratory.” https://colab.research.google.com/notebooks/intro.ipynb (accessed Apr. 14, 2021).

[4] “solder-fumes-asthma/sub-t: Object detection in Sub-T environments.” https://github.com/solder-fumes-asthma/sub-t (accessed Apr. 14, 2021). **[request access at** [***pabsan-0@student.ltu.se***](mailto:pabsan-0@student.ltu.se)**]**

[5] “cocodataset/cocoapi: COCO API - Dataset @ http://cocodataset.org/.” https://github.com/cocodataset/cocoapi (accessed Apr. 14, 2021).

[6] “AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet ).” https://github.com/AlexeyAB/darknet (accessed Apr. 14, 2021).

[7] “google/automl: Google Brain AutoML.” https://github.com/google/automl (accessed Apr. 14, 2021).

[8] “Ignition Robotics - Rescue Randy Sitting.” https://app.ignitionrobotics.org/OpenRobotics/fuel/models/Rescue Randy Sitting (accessed Apr. 14, 2021).

[9] “Unity-Technologies/SynthDet: SynthDet - An end-to-end object detection pipeline using synthetic data.” https://github.com/Unity-Technologies/SynthDet (accessed Apr. 14, 2021).

[10] “ShreyasSkandanS/pst900\_thermal\_rgb: ICRA 2020 | Repository for ‘PST900 RGB-Thermal Calibration, Dataset and Segmentation Network’ | C++, Python, PyTorch.” https://github.com/ShreyasSkandanS/pst900\_thermal\_rgb (accessed Apr. 14, 2021).

# Bibliography

[1] “Artifacts Specification Cave Circuit,” https://www.subtchallenge.com/resources/SubT\_Cave\_Artifacts\_Specification.pdf (accessed Apr. 14, 2021).

[2] K. Nguyen, N. T. Huynh, P. C. Nguyen, K. D. Nguyen, N. D. Vo, and T. V. Nguyen, “Detecting objects from space: an evaluation of deep-learning modern approaches,” *Electron.*, vol. 9, no. 4, pp. 1–18, 2020, doi: 10.3390/electronics9040583.

[3] J. Redmon and A. Farhadi, “YOLO v3,” *Tech Rep.*, pp. 1–6, 2018, [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[4] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv*, 2020.

[5] C. Y. Wang, A. Bochkovskiy, and H. Y. M. Liao, “Scaled-YOLOv4: Scaling cross stage partial network,” *arXiv*, 2020.

[6] M. Tan, R. Pang, and Q. V. Le, “EfficientDet: Scalable and efficient object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 10778–10787, 2020, doi: 10.1109/CVPR42600.2020.01079.

[7] C. Borngrund, “Machine vision for automation of earth-moving machines : Transfer learning experiments with YOLOv3,” 2019.

[8] D. Dwibedi, I. Misra, and M. Hebert, “[CutPaste&Learn] Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 1310–1319, 2017, doi: 10.1109/ICCV.2017.146.

[9] S. Hinterstoisser, O. Pauly, H. Heibel, M. Martina, and M. Bokeloh, “An annotation saved is an annotation earned: Using fully synthetic training for object detection,” *Proc. - 2019 Int. Conf. Comput. Vis. Work. ICCVW 2019*, pp. 2787–2796, 2019, doi: 10.1109/ICCVW.2019.00340.