Results

Summer-project 2020

Ola Tranum Arnegaard

The topic of this project was to detect and classify floating objects at sea given optical and thermal aerial footage of Trondheimsfjorden. After considering different approaches, the object recognition algorithms YOLOv5 and Faster R-CNN were the chosen to solve this task – two machine learning algorithms based on a convolutional neural network structure.

Data

The footage in the project was taken by a FLIR Duo Pro R[1] mounted on a drone. The camera captured video in both color (EO) and thermal (IR) at around 30 frames per second (fps). However, EO and IR differed slighltly, hence the full-length videos varied in length, but both around 1h and 45 min. Thererfore, to be able to compare the EO and IR videos, the frames had to be synchronized. Once synchronized, the EO videos could be used to classify objects in the IR videos that were difficult to categorize, as this was often impossible. This was accomplshed with the Python library *Flirpy*[2]. One of the included scripts were applied to the data, which gave two synchronized sets of frames from the given IR and EO at around 8fps, resulting of a loss of frames from 30fps to 8fps.

Another obstacle in the annotation process was that without seeing the temporal changes of the objects, which is not possible when watching individual frames only, it was at times difficult to see what the object is. Therefore, converting the frames back to videos after the synchronization was needed. The conversion was accomplished using the *ffmpeg* tool[3]. To keep the original quality of the videos, a close to lossless convertion was applied. Also, certain restrictions were needed to have a format that was compatible with the annotation tool.

The floating objects were categorized into five groups: *buoy, vest, pallet, pole* and *boat*. The objects appeared at quite different frequencies, mostly because some of the categories included several distinct objects while some included only one - *multiple* types of buoys and a *single* vest for instance. Because the videos were captured by a drone, the objects were quite small compared to the total captured area of the images. As a result, the level of detail of the objects captured by the camera was quite low.

[1] www.flir.com/products/duo-pro-r/

[2] www.github.com/LJMUAstroecology/flirpy

```
Python3 split_seqs.py -i ./*.SEQ –rgb ./*.MOV --jpeg_quality 100 -sync_rgb
```

[3] trac.ffmpeg.org/wiki/Encode/H.264

```
Ffmpeg -framerate 8 -i frame_%*.jpg -c:v libx264 -crf 17 IR.mp4
```

VoTT[4] is an annotation tool created by Microsoft, and what was used in this project. However, it is not categorized as an official release by Microsoft and is available through Github. It works well with video annotation, but has some shortages and lack of documentation/error-handling which affected this project to a considerable extent.

The final annotation resulted in 826 annotated EO images and 833 IR images. Two additional sets were created from the annotaed images, which were labeled as object/not-object (binary). Consequently, EO and IR were divided into two sets each, one with multiclass labels and one as binary. This was to see if/how it would affect the the detection measures of the models. The data was divided into a training/validation ratio of 80/20. I chose to use both quantitative and qualitatively testing in this project, that is, watching the performance of the model by testing it on three video-snippets from the dataset, at a total of 5 minutes – one for EO and one for IR. In addition, from these snippets, around 50-60 images were used to test the finished models to achieve some sort of quantitative performance measurements. Naturally, no training/validation data were gathered during the period of those snippets.

The data was imported into the project using the online tool Roboflow[5]. This tool provides possibilities for augmentation and preprocessing of the data, as well as easy import of the data to Google Colab. An explanation to how Roboflow is used to import data in this project's Google Colab may be found here.

Algorithms

In short, *object recognition* is the process of detecting (a) object(s) in an image with a bounding box or bounding mask, and subsequently classifying the detected object(s). Most object recognition algorithms are either multi-stage or single-stage. The distinction is if the detection and classification steps of the object(s) are somewhat separate or combined. In this project I have chosen algorithms from the YOLO family, which are single-stage algorithms; and Faster R-CNN, which is a multi-stage algorithm. The field of object recognition is advancing rapidly, and the mentioned algorithms are as of June 2020 among the state-of-the-art algorithms in the field of computer vision. Often, multi-stage algorithms performs better than single-stage algorithms on the most recognized dataset tests[6]. However, the single-stage algorithms are much faster, thus more applicable to realt-time applications. Below is a couple of short summaries of the algorithms:

Faster R-CNN[7]:

In short, the original R-CNN algorithm[8] is based on using the *Selective Search*[9] algorithm to find object propositions (region proposals), i.e. areas that

[4] www.github.com/microsoft/VoTT
[5] https://roboflow.ai/
[6] https://cocodataset.org/#home
[7] https://arxiv.org/abs/1506.01497
[8] https://arxiv.org/pdf/1311.2524.pdf
[9] https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf

may contain obejects. These proposals are fed to a convolutional neural network used as a feature extractor. Finally, this is then fed to a support vector machine which outputs predictions of an object within that region (as well as some adjustment parameters). The successor to R-CNN is Fast R-CNN, which as the name states, cuts the high computation time (by changing the structure of the algorithm), mostly by using selective search on feature maps created by the CNN rather then the image itself. Later, Faster R-CNN was released, which instead of using *Selective Search* to find the region proposals, rather *learns* those regions in a separate network.

YOLOv5[10][11]:

Contrary to R-CNN, the YOLO family of algorithms calculates both the regions and confidence of object in a single CNN. In short, the algorithm divides the algorithm into grid, each containing bounding boxes. Each bounding box includes a box confidence score, i.e. how likely the box contains an object; and each grid cell a conditional class probability for the box with the highest confidence score, i.e. the probability of the box containing each class.[12] As the procedure of detecting objects is quite less extensive in the YOLO algorithms than R-CNN, finding small objects has been seen as difficult.[13] However, techniques were implemented to solve this problem.[14] From the initial YOLO algorithm[15] to YOLOv4 (and recently YOLOv5) several changes were applied to the algorithm, making it faster as well as increasing performance. It should be noted that YOLOv5 was released in June 2020 and published by another developer than those that are unofficially maintaining the YOLOv3/4.[16] Unfortunately, not a lot of documentation and comparisons are available. However, the comparisons that are available show that the performance is roughly equal to YOLOv4. The reason why it was used in this project was because of the simplicity in the implementation. It should also be noted that most object recognition algorithms are available through open-source public repositories that are developed and maintained on a voulentary basis. In addition, multiple sizes of of the YOLOv5 algorithm are availiable. To make the process of training the models as fast as possible, the smallest of them was used, YOLOv5s.

Parameters of models

Little effort was put into designing the algorithms themselves. Therefore, as stated earlier, out of the box state of the art algorithms were used. Because of that, configurations or changes to the structure of the models were not applied.

[10] https://github.com/ultralytics/yolov5
[11] https://blog.roboflow.ai/yolov5-improvements-and-evaluation/
[12] https://medium.com/@jonathan˙hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088
[13] https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e «small»
[14] https://medium.com/@jonathan˙hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088 «small»
[15] https://pjreddie.com/darknet/yolo/
[16] https://github.com/AlexeyAB/darknet

To be able to do that, much time and effort would have been needed to understand each algorithm. However, some changes to the preprocessing and hyperparameters were applied. The parameters that were altered during the training are listed below. However, note that not all of the changes listed below have their corresponding models in the results section. This is because of loss of model weights, difficulties exporting models or lack of performance measurements.

Preprocessing:

- Resize: resizing images to a given resolution. May be by changing resolution, stretching or cropping.

Augmentation: randomly selected set of images added to the original set with an augmentation of either

- Rotation
- Brightness
- Exposure

Specifications in YOLOv5s:

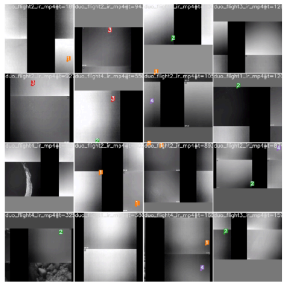
- Mosaic: Instead of simply using a single image, it combines four images as tiles in one image at a random ratio as input to the algorithm. To disable, use the keyword –rect.

  "the mosaic dataloader pulls up the selected image along with 3 random images, resizes them all to [selected resolution], joins all 4 at the seams, augments them, and then crops a center [selected resolution] area for placement as 1 image into the batch."

- Image Size: Input resolution. Similar to what may be applied in preprocessing.
- Epochs: Number of epochs to train for. 1 epochs is a runthrough of the training dataset once. Note, one mosaic augmentation is created for each image in the set.
- Evolve hyperparameters: Train while evolving hyperparameters for each new generation based on a normal distribution (at a 20% of a 1-sigma) of given initial values. The run with best fitness value is selected for the next mutation.[17] Fitness is calculated as a 50/50-value of the *F1* and *mAP* performance measurments, see *Types of Measure.*

[17] https://github.com/ultralytics/yolov3/issues/392

Mosaic data augmentation[18]



Example from IR dataset

Types of Measure

The most frequently used measurements of performance in object recognition is *mAP* and *F1*. In short, these variables are dependent on the recall value and precision of the model. Below is a quick introduction to how these values are calculated:

TP: true positive TN: true negative
FP: false positive FN: false negative

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precicion} = \frac{TP}{TP + FP}$$

$$\text{F1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

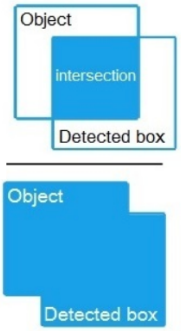
AP[19] $= \int_0^1 p(r)\, dr$, where p(r) is the precision by recall function

mAP: mean AP of each object

IoU: $\frac{area\ of\ overlap}{area\ of\ union}$

$mAP_{0.5}$: calculates precision and recall with minimum IoU of 50%

$mAP_{0.5:0.95}$: mean of mAP values at 0.05-increments from 50% to 95%



IoU

[18] https://blog.roboflow.ai/yolov4-data-augmentation/
[19] http://www.gabormelli.com/RKB/Bounding˙Box˙Intersection˙over˙Union˙(IoU)˙Measure

$mAP_{0.5}$ is the primary measure used in this project. They are applied to around 50-60 frames from the test video. In addition, a qualitive test is used to check the performance of the model, i.e. by visual inspection. This is done with an approx. five minute video of three clips. It includes most scenarios and all objects included in the training data, and naturally not included in the training. Also, frames per second (fps) is also an important measurement, as faster models are more desirable in real-time applications.

Computation

The computational power needed in this project exceeded what was availiable through my laptop. Therefore, the models were trained using Google Colab.[20] This resource offers powerful GPU's, which significantly decreased the computation time.

Results

The following tables show the results from the different models. There are four different datasets: optical camera with multiple classes (EO Multi-class), optical clases with one class (EO Binary), thermal camera with multiple classes (IR Multi-class) and thermal camera with one class (IR Binary). The differences among the models are either the base-model itself, input size of the image, number of epochs the model was trained for, or some applied configurations to the training. The performance measures are taken from the validation results at the final completed epoch of training, and the test results from testing the finished model on the test dataset. Note, P: precision and R: recall.

[20] https://colab.research.google.com/

EO – Multi-class

Model	Input Size	Epochs	Config	Valid: P	Valid: R	Valid: mAP 0.5	Test: P	Test: R	Test: mAP 0.5
YOLOv5s	416	600	--	0.495	0.800	0.795	0.452	0.803	0.778
YOLOv5s	416	650	--	0.500	0.810	0.800	0.456	0.825	0.799
YOLOv5s	416	1000	--	0.695	0.805	0.805	0.513	0.767	0.769
YOLOv5s	416	1000	Rectified	0.470	0.375	0.325	0.253	0.402	0.354
YOLOv5s	416	1000	Multiscaled	0.4 (0.6*)	0.880	0.805	0.287	0.773	0.732
YOLOv5s	416	1000	Hyp 25 runs	0.640	0.810	0.810	0.475	0.812	0.785
YOLOv5s	640	1500	--	0.3 (0.4*)	0.740	0.610	0.401	0.848	0.805
YOLOv5s	832	1750	--	0.3 (0.4*)	0.785	0.695	0.318	0.889	0.680
Faster R-CNN	600 (default)	363*	--	--	--	--	--	--	0.523

[Valid: P] **Actual maximum value, but reached at an earlier epoch*
[Epochs] **Calculated from the number of iterations:*

$$epochs = (iterations \cdot batch\text{-}size) \,/\, total\ images$$

EO – Binary

Model	Input Size	Epochs	Config	Valid: P	Valid: R	Valid: mAP 0.5	Test: P	Test: R	Test: mAP 0.5
YOLOv5s	416	1000	BINARY	0.575	0.810	0.770	0.550	0.879	0.807

IR – Multi-class

Model	Input Size	Epochs	Config	Valid: P	Valid: R	Valid: mAP 0.5	Test: P	Test: R	Test: mAP 0.5
YOLOv5s	416	1250	--	0.600	0.895	0.885	0.420	0.449	0.416
YOLOv5s	416	1500	--	0.54 (0.65*)	0.900	0.885	0.285	0.438	0.393
YOLOv5s	416	1750	Multi-scaled	0.490	0.820	0.750	0.300	0.498	0.445
YOLOv5s	416	1750	Hyp 15 runs	0.680	0.900	0.890	0.336	0.464	0.401
YOLOv5s	640	1500	--	0.48 (0.65*)	0.905	0.825	0.208	0.462	0.408

[Valid: P] **A higher value, reached in earlier epochs*

IR – Binary

Model	Input Size	Epochs	Config	Valid: P	Valid: R	Valid: mAP 0.5	Test: P	Test: R	Test: mAP 0.5
YOLOv5s	416	1200	BINARY	0.600	0.910	0.908	0.571	0.867	0.848

Most models were trained on the EO multi-class data. The reason for that was to simply to test if some configurations other than default would increase the performance significantly. For both EO and IR, this was not the case, i.e. the default configurations worked well compared to the others. By default, I mean not applying any configurations or having another input size than 416 for YOLOv5s and 600[21] for Faster R-CNN. Conclusively, it was likely to believe that small configurations would not improve the binary models significantly, and that a simple model would be adequate. For EO, separating the objects into classes or using a single common class did not make a significant difference. However, for IR, the differences were quite significant. It should be noted that because only one model was used for the binary sets, one may question the credibility of the results, but because of the significant performance improvement for IR, one have reason to believe that IR infact perform better at binary object recognition.

After watching the test video, it is apparent that a reason why the IR Multi-class performance is significantly lower than IR Binary is that, even though the former might detect certaing objects, it has substantial difficulties classifying them. This is noticeably not that problematic for EO multiclass, as distinguishing between the objects is a simpler task. This is reasonable, as it is even quite difficult for a human to classify objects in an IR video.

As a side note, in this project, the detection of the objects are more important than some miss-detections. Therefore, if compromised, a higher recall is more desirable than a high precision, even though some boxes are classified as objects when they are not, i.e. FP. In addition, limiting miss-classifications is not paramount as long as the object is detected. Therefore, if a EO Multi-class model had a slightly lower mAP than EO Binary, EO Multi-class could still be a more desireable model, as long as the miss-detected objects simply were miss-classifactions. However, the problem is how this may be measured.

Only a single Faster R-CNN model is included in the results section. Some were trained, but because of difficulties, the one displayed is the only one that I could produce test results with. As Faster R-CNN takes significantly longer time to train than YOLOv5s, the model could not be trained for a longer period of time due to limitations in computing power (12 hour computation cap). In addition, the reason why the model was not applied to different datasets was becauase of lack of time to do so, unfortunately. However, because of the weak

[21] https://groups.google.com/forum/#!topic/caffe-users/E5l9C˙QfcmI
https://stackoverflow.com/questions/58018623/input-image-size-for-tensorflow-faster-rcnn-in-prediction-mode#:˜:text=In%20Faster%20R%2DCNN%3A%20Towards,shorter%20is%20600%20pixels

results with EO Multi-class, it is likely to believe that similar results would be the case with the other datasets.

Discussion

In conclusion, after watching the performance of the models on the test-videos, it is apparent that the models may have a large recall/precision ratio at times, that is, having too many false positives (FP′s).

In this scenario, the model would be used the automatically detect floating objects at the sea surface. However, if FP′s are too frequently appearing, the model would simply perform similarly to manual inspection, making the concept of automatic object recognition redundant. It is also apparent that both IR and EO struggles with FP′s when the drone is in such an angle that the reflections in the water are intense. For EO, it becomes also quite difficult to see the object at all (TP). Both also struggles when the objects are small in size, or when the drone is far away from the object – both in detection and classification. At small to medium distance, the Binary IR/EO and Multi-class EO datasets perform well in both areas. The reason why Multi-class EO performs well at this distance is most likely that it may capture the separating features of the objects. On the contrary, finding features in IR images is difficult at most sizes, except for the pole. This is likely the reason why IR performs that much better when the classification is not valued. A model combining the IR and EO results were not conducted in this project. However, this may be a natural next step.

A few reasons why the performance of the models where not optimal are shown below:

- Objects might have been missed in annotation.
- Objects might have been wrongly annotated. Most probable for the IR data.
- Not enough added negatives, i.e. images without objects.
- No consistent strategy for annotating:
  - Strict bounding-box sizing rules
  - Waiting time between annotation of same object
- Test-snippets were chosen to represent most objects, which may be considered a biased choice of testing data.
- Train/Val split was a 80/20 **random** split, which may have resulted in frames close in time and therefore close to identical.
  - Evolve configuration is based on validation data independent from the training data.

- Test frames chosen by user, which may represent bias. An alternative strategy is necessary, as well as more test data (currently 50-60)

Link

Google Drive

This Google Drive includes summary of all models as well as tfrecords of the train/val figures which may be showed in Tensorboard. For most models, it also includes two different test runs on the five minute test video. The runs are created at a 0.4 and 0.75 confidence, i.e. the video will only show boxes around objects with a confidence score higher than the given threshold.