**FlySight project – Object detection**

# Introduction

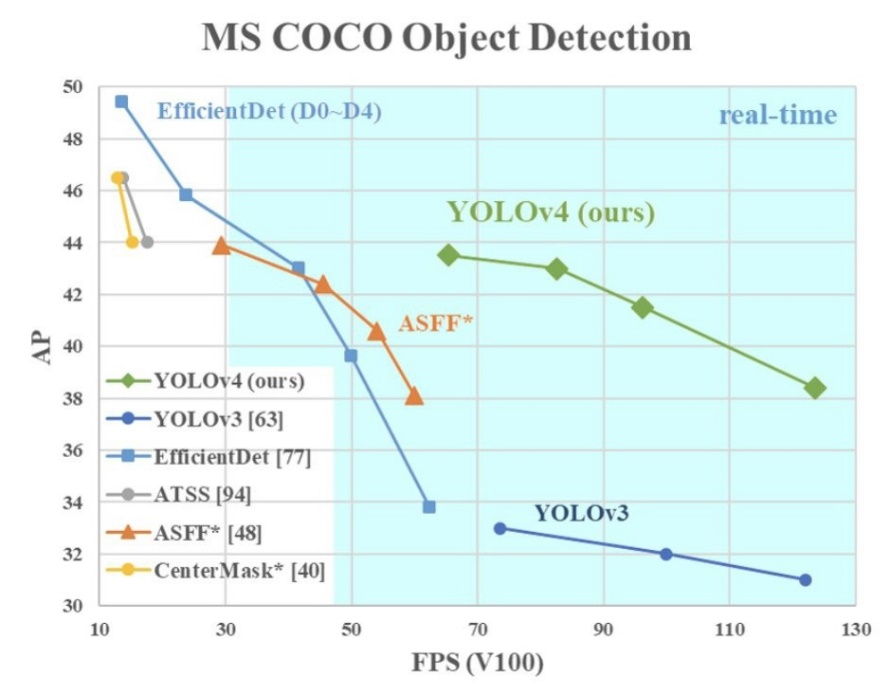
Search And Rescue (SAR) operations in non-urban areas (mountain, forest, canyon, water body…) usually requires a large amount of human personnel and when using drones, even an operator able to focus the entire time of the operation could potentially fail to identify a body in an unusual situation. Automated object detection could thus help save lives in such situations.

In this paper, we will be comparing the performance of 2 different object detectors: YOLOv5 and SSD. We will first briefly present the structure of the 2 nets used afterwards. Then we will introduce our database before showing the explorative data analysis done on this set. Next, we will explain the metrics used to compare the nets and the different trains. Eventually, we will showcase the performance of the nets and the influence of the train parameters for SAR operations.

# Presentation of the nets

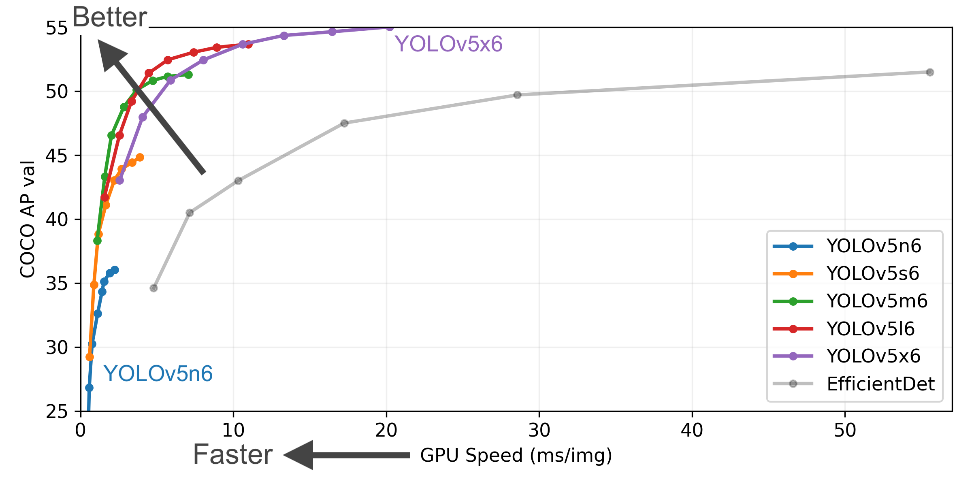
## YOLOv5

YOLOv5 is a state-of-the-art detector, it is based on the PyTorch framework. This detector is a You Only Look Once type of detector meaning the full images are processed by the neural network instead of applying the network multiple times to the image at different locations and scales. This network has one of the best speed on the market (*Figure 1*) with a relatively good precision.



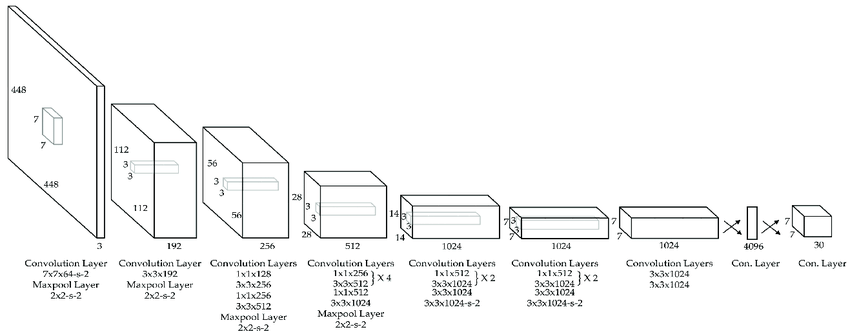
*Figure 1: Comparison of YOLOv4 and other state-of-the-art detectors*

Since detection speed is a key factor is SAR operations, we will be testing YOLOv5 as the best compromise between speed and accuracy amongst the YOLO versions. The main difference between YOLOv5 and previous YOLO versions is the translation of the Darknet framework to PyTorch framework. PyTorch framework speeds up the training and the inference using 16 bits floating point precision instead of the 32 bits used in the Darknet framework. To keep the training time reasonable, we will be using the YOLOv5n network which is the smallest and the fastest of all YOLOv5 nets (*Figure 2*).



*Figure 2: YOLOv5 nets comparison*

YOLOv5 uses *New CSP-Darknet53* as the backbone, *SPPF* and *New CSP-PAN* as the neck and *YOLOv3 Head* as the head. The architecture is a deep convolutional network as shown on *Figure 3*.



*Figure 3: High level diagram of YOLO architecture*

## SSD

TODO

# The dataset

The data we will be using for training and testing is the SAR dataset (SARD) built by FlySight. The images in this set present aerial view of persons in typical SAR situations: walking, running, sitting, standing or lying down. The database is made of 1920x1080 images and contains 1980 annotated frames (typical SAR situation on *Figure 4*).



*Figure 4: Image “gss.1” from the SARD database*

In the SARD database, the bounding boxes dimensions go from 7px for width and 8px for the heigh to 353px for the biggest width and 337px for the biggest heights. The area of bounding boxes range from 7x12px for the smallest to 322x231px for the biggest while the average area is 47x58px (according to FlySight article). In terms of data, there are 6532 person objects spread unevenly in terms of size.

In the original database that was given to us by FlySight, the labels were stored using XML files in PASCAL VOC format.

The SARD database is unique in its kind because it is specifically adapted for SAR operations. Other drone-made databases on the market (VisDrone, Okutama-action, UAV123, CARPK dataset, ERA dataset…) achieve good results in detection of person when used for training neural networks. However, all of them uses image from urban areas (*Figure 5*), thus reducing the efficiency of the detector when used in non-urban areas.



*Figure 5: Inference on a VisDrone dataset image*

# Explorative Data Analysis

## Format the labels

The original dataset we retrieved had 5 classes: stands, seated, walking, running, and laying down. For our work, we will only be using one object of type person. So, the first thing to do is to convert the labels from PASCAL VOC to YOLO format (and at the same time merge all objects into the person object). For that we used the website Roboflow which allows us to change the classes and convert the existing labels.

We then wrote a python script to check if the conversion went well: the script randomly selects an image, then uses the information in the YOLO .txt file to draw the bounding boxes over the image and display it on screen. This script doesn’t guarantee certainty that all the XML files were converted well but if randomly selected files are conformed to YOLO format, we can be almost sure that Roboflow generated the other in a same quality. *Figure 6* shows an example of an image with the bounding boxes.



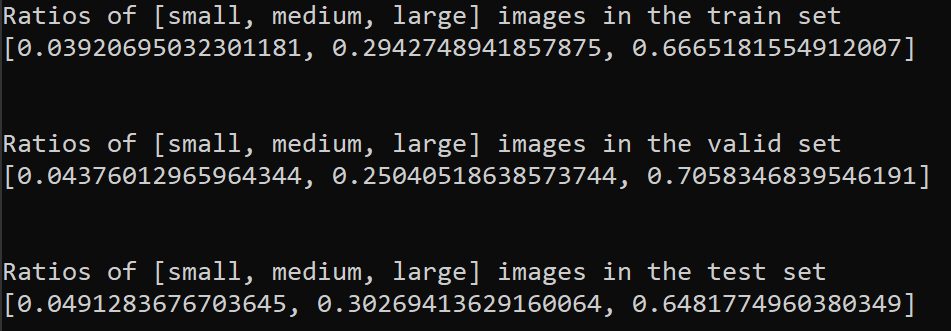
*Figure 6: Visualization script example*

We realized around 40 visualizations (only 2.5% of the number of images in the set) which were all showing correct bounding boxes. Anyways, if there were to be files with bad format, it would be visible on the results later.

## Splitting the data

Since our dataset is not huge, we choose the following split for the dataset: 70% of images for training, 20% of images for validation and 10% for testing. Same as before the splitting is done by Roboflow, the only thing we need to check is if images have been balanced meaning if the same ratio of small/medium/large bounding boxes is in every set.

We used the same delimitations as in the FlySight paper: bounding box area for small below 322px, medium between 322 and 962 and large above 962. Again, we used a python script to compute the ratio of small/medium/large bounding boxes in the 3 different sets.

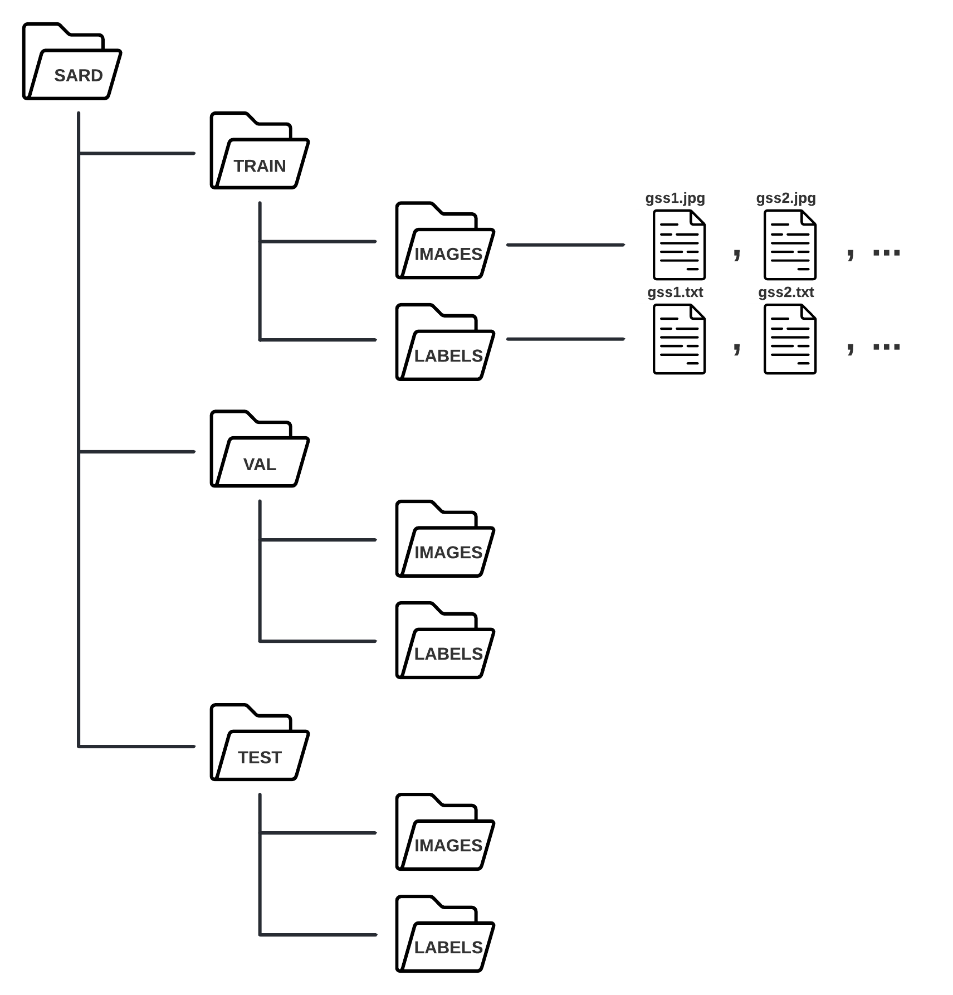


*Figure 7: Ratios computed by our python script*

With *Figure 7* we can conclude that balancing is good especially since we are counting bounding boxes so it is possible that a same image contains a small bounding box and a large bounding box which are impossible to split.

## Stretching

Before any augmentation of the dataset is done, images are stretched to 416x416. Stretched dataset is lighter and since images will be stretched to this resolution by the net afterwards, we might as well do it before. *Figure 8* shows the filesystem structure after the split.



*Figure 8: Filesystem of the splitted SARD database*

## Augmentation

We used different geometric and colour transformation to generate a new augmented dataset, once again Roboflow allows us to do the expansion easily. All the modified images will go into the train folder, putting images in the test or validation set is useless since it won’t improve our detector in any way whereas putting more image into the train dataset allows the detector to train on a larger variety of images.

First transformation is a flip. Flip allows us to substantially improve the number of raw images we have. We only did a horizontal flip since we are trying to detect persons and we don’t want them to be upside down. *Figure 9* shows an image and the flipped corresponding image.



*Figure 9: Original image on the left and flipped image on the right*

We then applied various colour transformation: hue, saturation, brightness and exposure. All of these transformations contribute to simulate different hours of the day, different seasons and different weather conditions. These transformations will also make our model less sensitive to camera differences. *Figures 10* to *13* show examples of these transformations.



*Figure 10: Image with hue transformation*



*Figure 11: Image with saturation transformation*



*Figure 12: Image with brightness transformation*



*Figure 13: Image with saturation transformation*

Last transformation we apply is blurring. This transformation simulates real conditions photos or videos taken by a drone which sometimes can be fuzzy if the drone is moving fast. *Figure 14* shows an example of a blurred image.



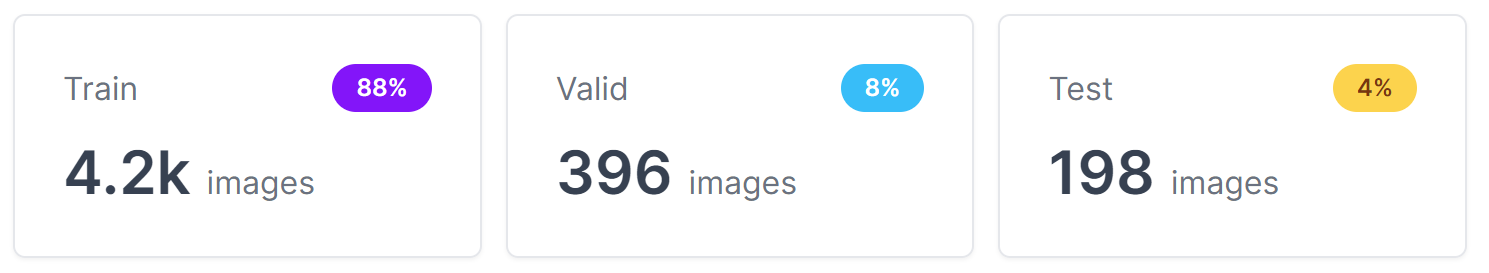
*Figure 14: Image with a blur transformation*

Finally we added a specific transformation: specific blurring of bounding boxes as shown on *Figure 15*. This allows us to take in account that people on the shots could be falling, running or doing some kind of rapid movement which would result in a blurred area around them.



*Figure 15: Image with blurred bboxes transformation*

At the end of the process, the dataset contains 4752 images, which is 2.4 times more than the original SARD database. Since all the augmented images are put in the training folder, the final ratios of splitting are the one shown on *Figure 16*.



*Figure 16: Repartition of images after data augmentation*

# Metrics and parameters

## Metrics

To analyse the performance of our detectors, we will use 4 main metrics: precision (*P*), recall (*R*), F1-Score (*F1*) and average precision (*AP*). We will now display the mathematical formulas behind these metrics and explain their meaning. In the following, *TP* will represent True Positive detection, *FN* False Negative detections, and so on…

* Precision is basically the percentage of correct detections amongst all the detections the net made. It is calculated using (1).
* Recall is a measure of the accurate detections amongst all the object that were to be detected. It is calculated using (2).
* F1-Score is basically the harmonic mean of precision and recall, computed following (3).
* Finally, the average precision is the area surrounded by the curve P(R) calculated by (4).

To these metrics we will add analyses using graphs combining these metrics and data provided by our detector.

## Parameters

Also, to improve the performance of our classifier we can modify the default parameters of yolov5 into another ones that fits better with our dataset. To get better parameters we use the evolve option that have yolov5 that use a Genetic algorithm to improve the performance modifying the parameter. The fitness formula used on this Genetic algorithm is calculated by:

ADD FORMULA (x \* % + y \* % + z \* % \* k \*%) = fitness.

To modify the default parameters that yolov5 we have run our detector with “–evolve” option for 134 times with 10 epochs (We don’t have time to deploy with higher number of epoch). Since our date is already augmented, we didn’t have in much account the parameters that rely on creating copies of the image by flip, mosaic, etc.

So

# Results

# Conclusion

# References

<https://chr043416.medium.com/how-hyperparameters-of-yolov5-works-ec4d25f311a2>

<https://docs.ultralytics.com/tutorials/hyperparameter-evolution/>

<https://github.com/ultralytics/yolov5>