Overhead object detection

Stian Jernæs[1], **Stian Aamli**[2], **and Alexander Olsen**[3]

[1]**stianje@uia.no**
[2]**stiana@uia.no**
[3]**alexao@uia.no**

ABSTRACT

The intention of this project is to create an object detection system with YOLOv7 that can identify pedestrians, cars, and other objects. We are using the Stanford campus drone data set to train our algorithm [1] which is then used to output the tracking data. We originally intended to count the people entering and exiting the frame, but as a result of jittery object tracking, we decided to rather visualize the data in aggregate. Resulting in a trace of the paths of different objects in the frame using Deepsort algorithm.

INTRODUCTION

The project idea is to be able to identify multiple moving objects in a scene. All the prerecorded scenes are from a UAV or unmanned aerial vehicle from a bird's view angle and contain different types of agents. Our goal is to be able to separate these agents and develop a tracking system to sort them into different categories. This gives us an insight into which type of traffic that is present at each place. After we have collected this data, we can then use this information to analyze the throughput of a geographic area. This could be used for stores or other traffic-dependent establishments to determine their location value. Furthermore, this could also be used to analyze traffic inside stores to optimize layouts.

DATA COLLECTION

We are using the "Stanford campus drone data set" (69 GB) which contains different scenes and objects. The collection contains multiple moving agents in a real-world outdoor setting, including colleges, roundabouts, and parks. There are six different agents that can vary from pedestrians, bikers, cars, carts, skaters, and buses. The data set also includes annotations for all the agents which makes it easier for us to train our algorithm. This data set was published as a supplement for a study to analyze social etiquette in crowded spaces [1]. In figure 1 is an overview by the authors of the study showing the breakdown of the objects it contains.

Scenes	Videos	Bicyclist	Pedestrian	Skateboarder	Cart	Car	Bus
gates	9	51.94	43.36	2.55	0.29	1.08	0.78
little	4	56.04	42.46	0.67	0	0.17	0.67
nexus	12	4.22	64.02	0.60	0.40	29.51	1.25
coupa	4	18.89	80.61	0.17	0.17	0.17	0
bookstore	7	32.89	63.94	1.63	0.34	0.83	0.37
deathCircle	5	56.30	33.13	2.33	3.10	4.71	0.42
quad	4	12.50	87.50	0	0	0	0
hyang	15	27.68	70.01	1.29	0.43	0.50	0.09

Figure 1. Overview of the Stanford campus data set. [1]

This video data had to be processed to achieve object recognition. As such we opted for the use of artificial intelligence, more specifically: YOLO or "you only look once" algorithm to do fast object recognition. It requires annotation and images to train. And luckily for us, the authors of the data set have done the hard work of annotating all the frames in the video. All that was required, was to convert the annotations included in the data set by converting them into YOLO format. The way they were formatting their data is as follows:

1. Track ID
2. X min - here is the lower bound of the x-axis for the box containing the object.
3. Y min - and equivalent for the y-axis.
4. X max - here is the upper bound of the x-axis for the box containing the object.
5. Y max - and equivalent for the y-axis, all put together it is enough to make a box.
6. Frame - the frame that contains the object.
7. Lost - whether or not the object is in or out of the field of view.
8. Occluded - whether or not the object is obscured by foliage or similar.
9. Generated - is when the object was generated using, interpolation or other means.
10. Label - human-readable string identifying the object.

Regarding, items 7 - 9 they indicate that some of the points might be machine-generated in and of themselves. We chose not to include these data points in the training of our model. As it might introduce unforeseen errors, or might even break the object detection by pointing to interpolated frames (like someone walking under a tree).

YOLOv7 training model is restricted to YOLO format only, requiring conversion of the annotation files. First and foremost, the position data is relative, meaning the input is unaffected by the scaling of the image/video. The following item list explains the YOLO format:

1. Object class - is the mutually exclusive type or class of objects in the boundary of the bounding box.
2. x center - is the center of the bounding box on the x-axis.
3. y center - is the center of the bounding box on the y-axis.
4. width - is the width of the bounding box.
5. height - is the height of the bounding box.

Here the data is formatted to show center coordinates from 0 to 1. To determine the object center, for instance, we used this formula:

$$\text{x center} = \frac{\frac{\text{x max} - \text{x min}}{2} + \text{x min}}{\text{image width}}$$

using an equivalent operation for the y-axis it is possible to convert the annotation to a YOLO format. And convert the height and width of a box into the correct format can be done this formula: $\text{width} = \frac{\text{x max} - \text{x min}}{\text{image width}}$ And an equivalent operation for the height.

METHOD

In the aforementioned assignment, YOLOv7 is utilized for real-time object detection. The resulting data is then processed using Deepsort for tracking specific objects. Accordingly, YOLO is used for determining the class of an object, while Deepsort is used to store and track their frame position. The data set is not flawless; the drone footage in the data collection occasionally shakes and warps, diverse angles and changes in the sun's direction cause different shadow profiles. Hence, making our goal to generalize the algorithm challenges. It was our aspiration to be able to create an AI that could deal with these obstacles. In hindsight, the most varying attribute is the shadow profile of the object. And it is also the easiest one to detect, muddying the input data drastically, making generalization of the AI incredibly difficult.

Real-time object detection

The process of real-time object detection has made some headway in recent years. When picking what object detection to use, it must be considered what resources are available and determine the purpose of the task at hand. There are a variety of different object detection algorithms available, each with its own targeted application. There are different object detectors that are specially designed for divergent circumstances. Some might be

optimized for low processing power environments, say detection on a jetson nano in contrast to high processing power devices.

For training on the aforementioned assignment, we have access to robust GPU servers made available via the website ”fe.uia.no”.

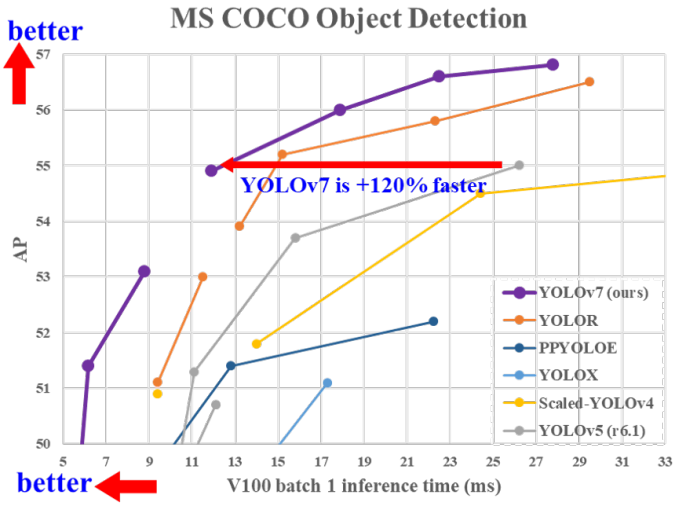

Figure 2. Benchmark comparison with other real-time object detectors [6]

In selecting object detection there are distinct versions of YOLO to be adopted and the best-fit algorithm for the large data set at hand. YOLOv7 outperforms others real-time object detectors, effectively utilizing parameters and computation. The methods proposed for making YOLOv7 perform effectively reduce about 40% parameters and 50% computation of state-of-the-art real-time object detector. As seen in figure 2. it surpasses other real-time object detectors in terms of accuracy related to batch interference. [3, 6].

Training

We started by gathering a collection of the most versatile videos regarding object differences. The goal is to even out the number of agents to achieve better results for the model. The video collection consists of a large amount of data that had to be analyzed, and the following videos were chosen:

Scene	Video ID
Bookstore	3, 4
Coupa	2, 3
Deathcircle	1, 4
Gates	1, 2
Hyang	0
Little	3
Nexus	11
Quad	3

Table 1. Videos from dataset.

After the annotation files were converted to YOLOv7 format, the training environment was set up. Since the model is trained with custom data, some preparation of the data and configurations needed to be arranged. After collection of the data was labeled with the correlating annotations, all of the files were split into frames ready to be fed into the training model. The folder structure of the training model is as follows in fig. 3. The split ratio between training and testing is approximately 80-20%. 80% of the data were used for training and the rest for validation. The split ratio is a normalized standard for training in YOLOv7 and other neural networks. Therefore, our data was distributed accordingly, and creating a custom .yaml was necessary since we operate within the category of custom data. The file was edited in YOLOv7 where we point to the folder paths and include names for objects present in the training:

```yaml
# train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3) list: [
    path1/images/, path2/images/]
train: ../train_data/images/train/
val: ../train_data/images/val/

# number of classes
nc: 7

# class names
names: [ 'Bicyclist', 'Pedestrian', 'Skateboarder', 'Cart', 'Car', 'Bus', 'Biker' ]
```

Listing 1. custom_data.yaml - configuration of file structure and class names

Since the training of our model includes a large set of data, an average computer was not sufficient in terms of computing power. Consequently, this resulted in the utilization of UIA's cloud GPU servers. The cloud servers contain either GPU A100 or V100 with a large amount of video memory, which ensued in faster training and higher precision for the model. The training was done in the virtual environment JupyterLab. Eventually, the training data was uploaded and all of the prerequisites were installed. The data set was ready to be trained. The following command was used with a few parameters:

```
$python train.py --device 0 --batch-size 40
    --data custom_data.yaml --img 640 640 --name yolov1
```

This command was used to start the training of the model.

- Device 0 means that the GPU is utilized instead of the CPU to maximize training speeds.
- Batch size is the value of samples before the model is updated.
- Data custom_data.yaml points to the location of the config file, to locate the labels and images from the training.
- IMG tag is the resolution of the images processed for the training.
- Name is the name for the project folder.
- 300 epochs are the default value if not defined otherwise.

The training on the UIA server was running for 4 days in total. According to the output log, the best model was produced on day 3. The model ran through 300 iterations or epochs in 4 days which is the default length of a standard training length.

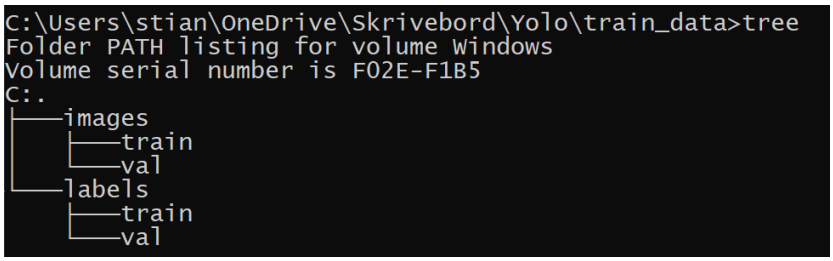

Figure 3. Folder structure for training and testing data.

Deepsort

Deepsort incorporates deep learning into the SORT(Simple Online Realtime Tracking) algorithm by using an appearance characterization to minimize identification swaps and therefore improve tracking efficiency. Sort is built upon four core components detection, estimation, association and destruction. SORT algorithms can successfully track the objects. The real-time object detector provides detection of objects, the Kalman filter provides tracking, and the Hungarian algorithm provides data association.

Deepsort tracks objects not just based on their velocity and motion, but also according to their appearance. Following the acquisition of the appearance descriptor, deepsort performs closest neighbor searches in the visual appearance to create the measurement-to-track correlation. [2, 4, 5]

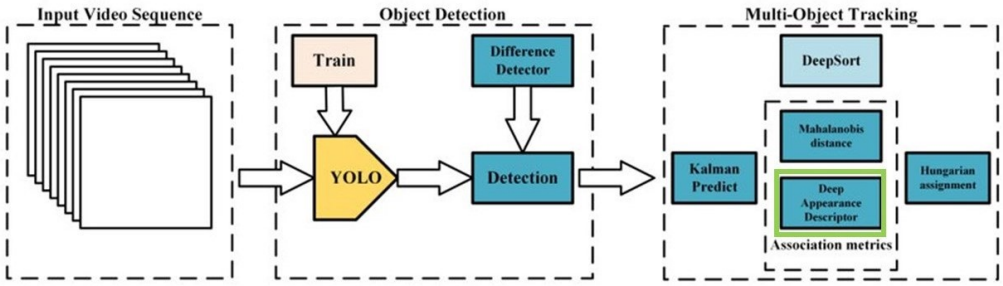

Figure 4. Diagram of the pipeline. [2]

RESULTS

Figure 5. Object tracking for Coupa2 scene.

Our initial intentions were to produce, a heat map that would show traffic from an overhead view. However, as the project matured we realized that it would neither be particularly useful, specifically in regards to different objects we can detect and their paths. Hence, we opted to rather visualize only the movement paths of different objects we were able to track. Making what we coined as a move map. It utilizes transparency in order to make the result cumulative so that the top layer does not overwrite the ones underneath.

In figure 6, the cumulative object movement of a video can be observed. The blue shows the pedestrians while the red is bicyclists, and the yellow are carts. Here we can see that the algorithm even spotted a couple of pedestrians who were sitting/ standing beside a tree at the bottom left corner of the frame. It can be observed that on the stairs in the top right corner, there are predominantly, pedestrians as should be expected. Nevertheless, there are spots of red and given that we checked the video, it is a false positive.

`Video collection of different tracked scenes: https://youtu.be/xPWCL5W_zEI`

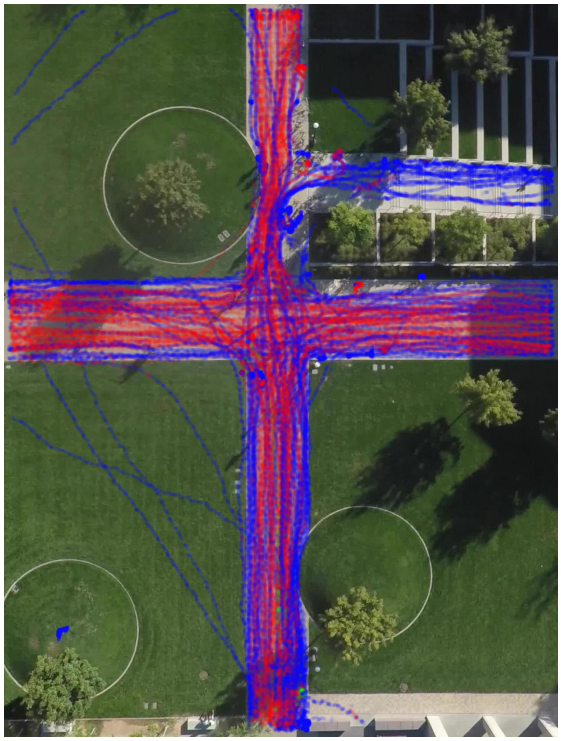

Figure 6. Result of the visualization of our object tracking.

Metrics

YOLOv7 provides convenient metrics and statistics when the neural network has finished training. These can be used to gauge the accuracy and performance of the said network. As could be observed in figure 7 we have issues with identifying certain objects, like misidentifying pedestrians for background noise, bikers for bicyclists, carts for cars, and busses for cars. The largest miss rate was among objects that are generally quite rare. Like carts, and busses. From the statistics, it appears like the buses were missed completely. They are quite rare in our data set, and our training data might not have included any of them.

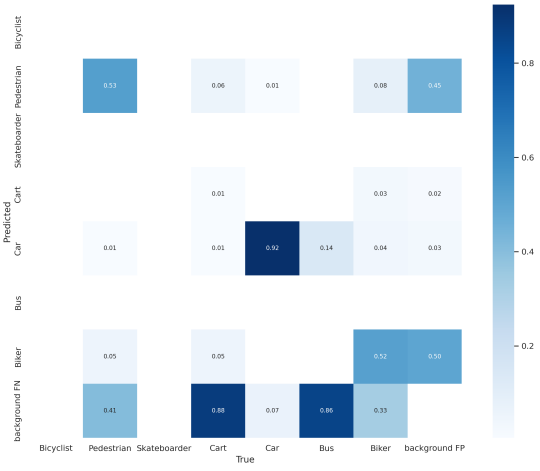

Figure 7. A matrix showing hit and miss rate.

DISCUSSION

The workflow in the group project was well-balanced in terms of divided work. We started by assigning each person to a planned task determined by their skill set. With a set of agreed-upon goals, the stages of productivity were increased and resulted in an optimized workflow. Throughout the stages, discussions, and decision-making were present at all times.

Our data shows some misclassifications. It should be said however that some of the objects are hard to spot, particularly when not looking at them in motion. If there is a misclassification there is an error cascade that, affects the next classification in the next frame, and so on. This issue could be mitigated by having a system that also went backward to adjust detections. For instance, if a biker appears for a frame or two, and then suddenly disappears without leaving the frame, he either was never there or the tracking stopped working. Nonetheless, we presume such a system would no longer be real-time. The system as it works currently does indeed track objects with a surprising amount of accuracy. Furthermore, the resulting system that we made is quite robust. And it works in real-time given you possess a modest PC.

Indeed, the different angles, shadows, and conditions made the detection of objects more difficult. For instance, the camera, of the UAV is not positioned at the same height compared to ground level in every video. Resulting in the objects being bigger or smaller in the different videos. The training from one clip to another is less applicable. Additionally, the objects are still quite small. This leads to higher noise sensitivity and increases the probability of mistakes. Because the scenes have different backgrounds and attributes, something that could have been done differently for more accuracy is to require training on each scene separately rather than combining them as done in the aforementioned assignment. Although, that could put us in danger of overfitting the model.

Counting objects never quite worked as intended. A solution we never tried was a more traditional method of counting objects. This method involves drawing virtual lines for common crossings and counting the objects that cross them. This method is fairly accurate and is widely used in object counters. So in this project, we could have made an algorithm that counts the objects entering and exiting these squares/ lines and enumerate how many have passed by. This method is commonly used in combination with YOLOv7 and Deepsort.

Another thing that could have been done was to balance the input data in training. Since there was an uneven distribution of objects that stems from some being less frequent than others and resulting in misidentifications. The training took a long time to conclude and resulted in difficulties trying to test with other input data balances.

CONCLUSION

As stated earlier we intended to create a program that would detect and track moving objects. We have managed to train an AI that can separate and track different objects. However, it has issues with accuracy and object retention. That is it loses sight of objects and thus artificially inflates the number of objects present on the screen. Likewise, the ability to count the objects we have identified is so error-prone that it rendered the feature pointless. We do nevertheless maintain the ability to track the class with reasonable accuracy and that is why we opted for projecting it as a movement path (map) instead. By utilizing transparency, we can display the movement data in aggregate, reducing the weight of errors and resulting in clear and concise data that tells a story. While there are ways to improve the accuracy/performance of our project. As mentioned in the discussions a cross-line counter is an optimal way to detect the number of agents entering/exiting a scene. To improve accuracy we never tried training a neural network for each of the scenes this might have increased the precision of the detector. At the risk at over fitting the data.

REFERENCES

[1] A. S. A.Robicquet, A.Sadeghian. Stanford drone dataset, 2016. Available: https://cvgl.stanford.edu/projects/uav_data/.

[2] R. Kanjee. Deepsort — deep learning applied to object tracking, 2020. Available: https://medium.com/augmented-startups/deepsort-deep-learning-applied-to-object-tracking-924f59f99104.

[3] S. R. Kukil. Yolov7 object detection paper explanation and inference, 2022. Available: https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/.

[4] D. P. Nicolai Wojke, Alex Bewley. Simple online and realtime tracking with a deep association metric, 2017. Available: https://arxiv.org/pdf/1703.07402.pdf.

[5] Sanyam. Understanding multiple object tracking using deepsort, 2022. Available: https://learnopencv.com/understanding-multiple-object-tracking-using-deepsort/.

[6] C.-Y. Wang. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. Available: https://arxiv.org/abs/2207.02696.