



Emerging technologies for the Early location of Entrapped victims under Collapsed Structures & Advanced Wearables for risk assessment and First Responders Safety in SAR operations

D3.5 Data-driven analytics applied on UAV imagery using deep learning

Workpackage: WP3 - Situation Awareness

Authors:	KT, AIDEAS
Status:	Draft
Due Date:	31/12/2021
Version:	1.00
Submission Date:	31/12/2021
Dissemination Level:	PU

Disclaimer:

This document is issued within the frame and for the purpose of the Search and Rescue project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 882897. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are the property of the Search and Rescue Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the Search and Rescue Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the Search and Rescue Partners. Each Search and Rescue Partner may use this document in conformity with the Search and Rescue Consortium Grant Agreement provisions.










(*) Dissemination level.-PU: Public, fully open, e.g. web; CO: Confidential, restricted under conditions set out in Model Grant Agreement; CI: Classified, Int = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.












Search and Rescue Project Profile

Grant Agreement No.: 882897

Acronym:	Search and Rescue
Title:	Emerging technologies for the Early location of Entrapped victims under Collapsed Structures & Advanced Wearables for risk assessment and First Responders Safety in SAR operations
URL:	https://search-and-rescue.eu/
Start Date:	01/07/2020
Duration:	36 months

Partners

	NATIONAL TECHNICAL UNIVERSITY OF ATHENS (NTUA) <u>Co-ordinator</u>	Greece
	AIDEAS OÜ (AIDEAS)	Estonia
	SOFTWARE IMAGINATION & VISION S.R.L (SIMAVI)	Romania
	MAGGIOLI SPA (MAG)	Italy
	KONNEKT-ABLE TECHNOLOGIES LIMITED (KT)	Ireland
	THALES ITAIA Italia SPA (THALIT)	Italy
	ATOS IT SOLUTIONS AND SERVICES IBERIA SL (ATOS)	Spain
	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH)	Greece
	UNIVERSITA DEGLI STUDI DI CAGLAIRI (UNICA)	Italy

	UKEMED GLOBAL LTD (UGL)	Cyprus
	PUBLIC SAFETY COMMUNICATION EUROPE FORUM AISBL (PSCE)	Belgium
	UNIVERSITA DEGLI STUDI DI FIRENZE (UNIFI)	Italy
	DEUTSCHES FORSCHUNGSZENTRUM FÜR KUNSTLICHE INTELLIGENZ (DFKI)	Germany
	UNIVERSITA CATTOLICA DEL SACRO CUORE (UCSC)	Italy
	VRIJE UNIVERSITEIT BRUSSEL	Belgium
	SYNYO GmbH (SYNYO)	Austria
	UNIVERSITEIT HASSELT (UHASSELT)	Belgium
	SPOLECZNA AKADEMIA NAUK (SAN)	Poland
	GIOUMPITEK MELETI SCHEDIASMOΣ YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH)	Greece
Search and Rescue End-Users		
	ELLINIKI OMADA DIASOSIS SOMATEIO (HRT)	Greece

	ENOSI PTYCHIOYCHON AXIOMATIKON YPAXIOOMATIKON PYROSVESTIR OY SOMATEIO (EPAYPS)	Greece
	JOHANNITER-UNFALL-HILFE EV (JOHAN)	Germany
	JOHANNITER OSTERREICH AUSBLIDUNG UND FORSCHUNG GEMEINNUTZIGE GMBH (JOAFG)	Austria
	CONSIGLIO NAZIONALE DELLE RICERCHE	Italy
	POMPIERS DE L'URGENCE INTERNATIONALE (PUI)	France
	ASOCIATA CLUSTERUL ROAMN RENTRU PROTECTIE SI ECOLOGIE IN DOMENIUL MATERIALELOR CHIMICE, BIOLOGICE, RADIOLOGICE/NUCLEARE SI EXPLOZIVE (PROECO)	Romania
	SERVICIO MADRILENO DE SALUD (SERMAS)	Spain
	FUNDACIÓN PARA LA INVESTIGACIÓN E INNOVACIÓN BIOSANITARIA DE ATENCIÓN PRIMARIA (FIIBAP)	Spain
	ESCUELA ESPAÑOLA DE SALVAMENTO Y DETECCIÓN CON PERROS (ESDP)	Spain

Document History

Version	Date	Author (Partner)	Remarks/Changes
0.10	01/03/2021	Christos Angelidis Iosif Sklavidis (KT)	ToC
0.11	06/04/2021	Athanasios Siouras, Serafeim Moustakidis (AIDEAS)	Draft with first batch of results
0.20	30/09/2021	Athanasios Siouras, Konstantinos Stergiou (AIDEAS)	Second draft including thermal images
0.30	30/11/2021	Christos Angelidis Iosif Sklavidis (KT) Athanasios Siouras, Konstantinos Stergiou, Serafeim Moustakidis (AIDEAS)	Final draft submitted for internal review
0.31	09/12/2021	AIDEAS, UBITECH, CERTH	Review comments and remarks
0.40	10/12/2021	Athanasios Siouras, Konstantinos Stergiou (AIDEAS)	Updated draft addressing the comments/remarks
0.50	16/12/2021	Christodoulos Santorinaios (NTUA)	Quality Control
1.00	31/12/2021	Christos Ntanos (NTUA)	FINAL VERSION TO BE SUBMITTED

Executive Summary

This deliverable presents the overall development status of the deep learning analytics applied on UAV imagery (both visual and thermal) on M18 of the project's lifetime. Within the duration of T3.3, several DL pipelines were designed, implemented, and tested on the task of object detection with humans as the main object of interest. Different variants of the pipelines were investigated including various powerful State-of-the-Art object detection network including Yolov4, scaled-Yolov4, Yolov5, Detectron2 and FasterRCNN. In addition, a novel hybrid inference mechanism was proposed, developed, and tested to cope with the identified challenges especially with respect to the effect of the UAV flight altitude. The proposed inference mechanism combines the output of altitude-dependent local deep learning increasing the generalisation capabilities of the OD system. An extensive experiment set-up was designed to identify the best performing deep learning networks and demonstrate the detection performance of the proposed object detection pipeline utilizing both spectral and thermal information.

Table of Contents

1	Introduction	11
1.1	Objectives	11
1.2	Structure of the deliverable	11
1.3	Relation with other documents	11
2	Advanced AI pipeline for object detection	12
2.1	Pipeline design overview	12
2.2	Data preprocessing	13
2.3	Transfer Learning.....	13
2.4	Models.....	15
2.4.1	You Only Look Once (YOLO)	15
2.4.2	YOLO version 4 with Darknet framework.	16
2.4.3	YOLO version 5 with Pytorch framework.....	17
2.4.4	Scaled YOLO version 4.	18
2.4.5	Efficient Det.....	19
2.4.6	Detectron2	21
2.5	Google Colab	22
2.6	Metrics	22
2.6.1	FPS	25
2.7	Validation Strategy	25
3	Results	26
3.1	Presentation of Datasets	26
3.1.1	High altitude dataset	26
3.1.2	Low altitude dataset.....	27
3.1.3	Thermal images dataset	28
3.2	Object detection results on the high-altitude dataset	29
3.3	Object detection results on the low altitude dataset	33
3.4	Performance with respect to different image resolutions and fps	41
3.5	Cross dataset testing using scaled-Yolov4	42
3.6	Results on the concatenated dataset	46
3.7	Results using a hybrid inference mechanism	48
3.8	Results on thermal images	54
4	Conclusions	56
	Annex I: References	57

List of Figures

Figure 2-1: Object detection pipeline.....	12
Figure 2-2: Transfer Learning diagram	14
Figure 2-3: Learning improvement [3].....	14
Figure 2-4: A simplified input of the YOLO object detector pipeline	15
Figure 2-5: Architecture of YOLOv4-large, including YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. The dashed arrow means replace the corresponding CSPUp block by CSPSPP block. [9]	19
Figure 2-6: Model Scaling [10]	19
Figure 2-7: EfficientDet architecture [10].....	20
Figure 2-8: Feature network design.....	20
Figure 2-9: Detectron 2 pipeline [12]	21
Figure 2-10: Precision metric formula	23
Figure 2-11: Recall metric formula	23
Figure 2-12: A sketch of a graphical depiction of the IoU metric	24
Figure 3-1: Examples from the Lacmus dataset.....	27
Figure 3-2: Example of Mixed dataset	28
Figure 3-3: Example of Thermal dataset	29
Figure 3-4: Output metrics for scaled-YOLOv4 with 32 batch size, CSP pre-trained weights	32
Figure 3-5: Output metrics for scaled YOLOv4 with 32 batch size, CSP pre-trained weights	32
Figure 3-6: Output metrics for YOLOv5 with 2500 images, small pre-trained weights	33
Figure 3-7: Output metrics for scaled YOLOv4 with 200x200 resolution, CSP pre-trained weights, 150 epochs, and 800 images.....	35
Figure 3-8: Output metrics for scaled YOLOv4 with 16 batch size, CSP pre-trained weights	35
Figure 3-9: Output metrics for scaled YOLOv4 with 16 batch size, P5 pre-trained weights, and 100x100 resolution.	36
Figure 3-10: Output metrics for scaled YOLOv4 with 500x500 resolution, CSP pre-trained weights, 150 epochs, and 800 images.....	36
Figure 3-11: Output metrics for scaled YOLOv4 with 749 images, CSP pre-trained weights and 200 epochs.	37
Figure 3-12: Output metrics for scaled YOLOv4 with 749 images, P5 pre-trained weights and 50 epochs.	37
Figure 3-13: Output metrics for scaled YOLOv4 with 8 batch size, CSP pre-trained weights, 50 epochs, and 800 images.	38
Figure 3-14: Output metrics for scaled YOLOv4 with 10 batch size, CSP pre-trained weights, 800x800 resolution, and 150 epochs.....	38
Figure 3-15: Output metrics for scaled YOLOv4 with 10 batch size, CSP pre-trained weights, 800x800 resolution and 300 epochs.....	39

Figure 3-16: Output metrics for scaled YOLOv5 large with 24 batch size, and 583 images.	39
Figure 3-17: Output metrics for large YOLOv5 with 32 batch size, and 300 epochs.	40
Figure 3-18: Output metrics for YOLOv5 with no weights, 64 batch size, and 583 images.....	40
Figure 3-19: Output metrics for YOLOv5 with 64 batch size, small pre-trained weights and 300 epochs.	41
Figure 3-20: Average precision of Scaled-Yolov4 per image resolution and GPU speed	42
Figure 3-21: Identification of humans in images taken at a low flight altitude	44
Figure 3-22: Identification of humans in images taken at a high flight altitude	44
Figure 3-23: Identification of humans in images taken at a high flight altitude.....	45
Figure 3-24: Identification of humans in images taken at a high flight altitude	45
Figure 3-25: Output metrics for the concatenated dataset.	46
Figure 3-26: Indicative successful detections of humans from the OD model trained on the concatenated dataset.....	48
Figure 3-27: Hybrid-DA methodology framework.....	49
Figure 3-28: Best models outputs	50
Figure 3-29: Successful detection of humans using the proposed hybrid inference	53
Figure 3-30: Indicative successful examples of human detection using thermal images	55

List of Tables

Table 2-1: YOLOv4 object detector backbones.....	17
Table 3-1: Training models and parameters for Lacmus dataset.....	30
Table 3-2: Results for Lacmus dataset.....	31
Table 3-3: Training models and parameters for Mixed dataset	34
Table 3-4: Results for mixed dataset	35
Table 3-5: Output metrics per input image resolution	42
Table 3-6: Cross dataset weights metric results	43
Table 3-7: Parameters of the model trained in the concatenated dataset.....	46
Table 3-8: Performance of the model trained in the concatenated dataset.....	46
Table 3-9: Best performances achieved by Scaled-Yolov4 on different testing scenarios	50
Table 3-10: Parameters of the model trained in the thermal dataset	54
Table 3-11: Performance of the model trained in the thermal dataset	54

1 Introduction

1.1 Objectives

The scope of this report is to demonstrate the performance of the developed Deep Learning (DL) algorithms in detecting objects of interest (humans) in search and rescue operations. Several DL pipelines have been designed, implemented, and tested integrating recent and powerful State-of-the-Art object detection (OD) networks. This report demonstrates the effectiveness of the proposed pipelines via thorough comparative experimentation that involves numerous data management, pre-processing and OD learning algorithms.

The ultimate objectives of this report are to:

- i) Identify human presence using aerial UAV images and advanced deep learning analytics
- ii) Employ and utilize both spectral and thermal information in the task of object detection
- iii) Explore the efficacy of several powerful DL networks and select the one providing the optimal detection performance
- iv) Design, implement the final DL-enhanced OD pipeline that will be capable of detecting human under varying conditions.
- v) Demonstrate the predictive performance of the proposed OD pipelines providing visual examples (objects detected at various sceneries).

1.2 Structure of the deliverable

The rest of this deliverable is structured as follows. Section 2 presents the main characteristics of the advanced AI analytics and the associated pipelines that were used for implementing the object detection task. Technical characteristics of all the experiments are analytically provided. Section 4 presents and discusses the results of the proposed methodologies and finally, Section 5 concludes this report by presenting a short summary and general conclusion deriving from this report.

1.3 Relation with other documents

This report is related to WP4 and more specifically to tasks 4.2 (Data aggregation) and 4.5 (Development of DSS Components). The proposed DL component will be fully aligned with the data aggregation mechanisms and tools decided in Task 4.2. The outputs of this component will be the detection findings that will be formulated using well-known standardized formats (e.g., JSON files). The standardized detection outputs will be then communicated to the SnR Decision Support System (Task 4.5) that will use them to take decisions or suggest appropriate actions. The SOT DSS by KT and the PHYSIO DSS by CNR will be affected by this input in operational level. The detections (only text output) of the algorithms will be involved in CONCORDE's notification service. As a result, the end-user will see these detections on the screen and then decide enhancing his/her situation awareness capacity.

2 Advanced AI pipeline for object detection

2.1 Pipeline design overview

Object detection is a computer vision technique, for object identification and location in an image or video. It is applied to count objects in a scene, evaluate, map, and mark their exact positions. There is a variety of object detection algorithms that can be applied in different classes. The classes are defined from the behaviour and representation of the object in space (images, videos). People in motion, animals, objects such as glass, bicycles, etc., are some of the main classes that can be recognized using CNN-based deep learning networks either using custom-made designs or pre-trained algorithms.

Convolutional Neural Networks (CNN) [1] is one of the most popular Deep Learning Neural Networks, and its name comes from the convolution mathematical linear operation between the matrixes. A CNN consists of multiple layers such as convolutional layer, non-linearity layer, pooling layer and fully connected layer. The convolutional and fully connected layers have parameters, while the non-linearity and pooling ones don't. CNN are able to provide high performance in Machine Learning problems, especially with applications where image data, computer vision, and natural language processing are involved.

In the present study, we have set as main objective to recognize and locate humans from aerial photographs of drones. We consider three image datasets appropriate for this project including humans as the main object in different landscapes, hours, and seasons of a year.

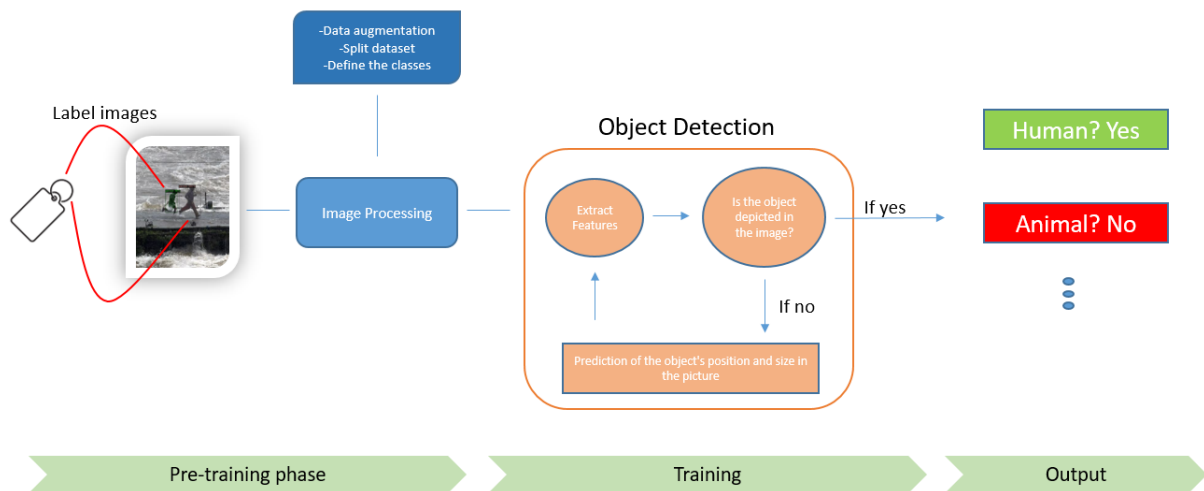


Figure 2-1: Object detection pipeline

Bellow we present the proposed AI pipeline which comprises the following processing steps:

- Build a dataset for object detection using selective search
- Annotate the dataset using appropriate tools
- Fine-tune classification models on dataset
- Run a selective search on the input image during inference
- Using a fine-tuned model, make predictions on each proposal.

- Return the final results of the object detection.

Each of the aforementioned processing steps is presented in the subsections below.

2.2 Data preprocessing

Organizing, preparing, and annotating the training data is a first crucial procedure, before training an object detection model. Selecting a labeling graphic user interface (GUI) is needed, and the main steps in the data annotation procedure are: (i) selection of images from the dataset, (ii) constructing boundary boxes and adapting the size of the boxes to the objects, and (iii) assigning the known object to one of the pre-determined classes (Figure 2-1). This step produces a new folder, with individual marking files for each image (most common, txt, xlm, json). Pre-processing and augmentation of images are measures that help with the smoother execution of the OD learning challenge. Moreover, different image preprocessing functions, such as image resizing and color transformations, can be added to the instruction, validation, and test sets. Picture resizing alters the size of the images and scales them to a specific set of dimensions. For example, to prevent the algorithm from identifying an object from its colors, color conversion can be used to convert a colored image (RGB channels) into a single grayscale channel. Image augmentation is also vital to increase the generalizability of the OD algorithm. Image enhancement involves, image shifting, random rotations, cropping, shearing, blurring, and random noise addition methods, which are used to magnify an actual dataset. Exporting the annotations in the correct format, which depends on the OD chosen model, is the final stage before the OD training.

2.3 Transfer Learning

In machine learning, transfer learning is a useful method for dealing with the issue of inadequate training data. In the transfer learning process a trained model in one task with limited data availability is repurposed on a second related task, allowing rapid progress or performance improvement. An advantage of Transfer Learning is that it allows Convolutional Neural Networks (CNNs) to learn from small data sets by importing information from models that have been pre-trained on massive datasets. A simple definition is as follows: "Transfer learning is the reuse of a pre-trained model on a new problem. It's currently very popular in deep learning because it can train deep neural networks with comparatively little data" [2]. More specifically, we use the knowledge from an already-trained network, with high amounts of data, and transfer the knowledge, through the weights, to our model. For transfer learning applications in problems that use image data, the most common models are the deep learning pre-trained ones. This technique is done between the Layers where the network architect decides to freeze. One of the main advantages of transfer learning are that the model has an improved initial skill along with an increased rate of improvement during the training phase. Finally, the trained model has a higher convergence capacity compared to a custom-made network.

In Figure 2-2 the structure of the aforementioned technique is presented, whereas in Figure 2-3 the expected learning improvement with transfer learning is indicatively given.

TRANSFER LEARNING

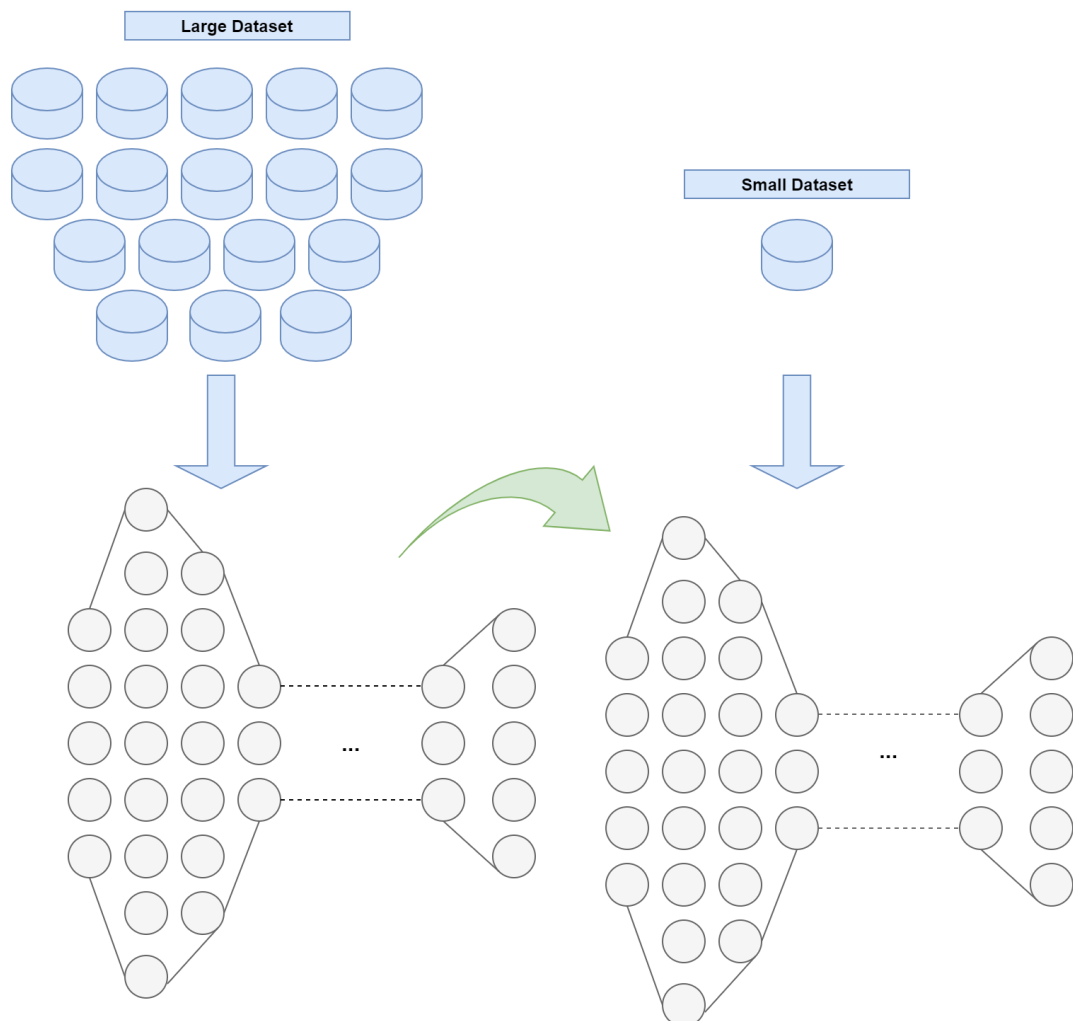


Figure 2-2: Transfer Learning diagram

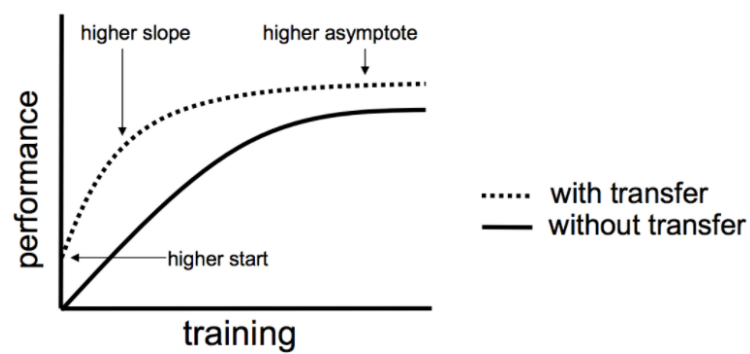


Figure 2-3: Learning improvement [3]

2.4 Models

In object detection models fast and accurate detection algorithms are needed. Detection systems repurpose classifiers for detection, evaluating them at various locations and scales in a test image. A number of state-of-the-art architectures have been evaluated in this deliverable to test their performance and thus select the best performing model. The necessary details for every network such as parameters, frameworks, etc., are presented in the subsections below. The most efficient and reliable one was finally selected to be integrated in our final OD pipeline.

2.4.1 You Only Look Once (YOLO)

YOLO is a unified system for detecting objects. The YOLO architecture is similar to a fully convolutional neural network (FCNN) [4]. Specifically, it utilizes a convolutional network for object detection, and uses the whole image to reframe the object detection as a regression problem. It feeds the image ($n \times n$) to the FCNN and the output is an ($m \times m$) prediction, where the $m \times m$ grid defines the bounding boxes. These models are optimized for detection performance by being trained in full pictures. For example, Figure 2-4 is divided into an $S \times S$ grid, and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. YOLO models have a lot of advantages compared to traditional object detection methods. First of all, YOLO is extremely fast with good real-time performance and high accuracy [5], it works with the entire image during training and test, it exports contextual information about classes and appearance, and learns generalizable objects' representations.



Figure 2-4: A simplified input of the YOLO object detector pipeline

YOLO consists of two main post-processing steps to quantify the bounding boxes: (i) intersection over union (IoU) and (ii) non-maximum suppression (NMS). Non-max suppression is an object detection approach that seeks to select the best bounding box from a series of overlapping boxes. The goal of non max suppression in the following image is to eliminate the yellow and blue boxes, leaving just the green box. The IoU measures how closely the predicted bounding box corresponds to the real object bounding box. NMS is applied extensively in important computer vision areas and is a key component of many proposed detection methods, whether we cope with edge, corner, or object detection. Moreover, the detection algorithms' inability to localize accurately the term of interest (clusters of multiple detections close to the actual size), defines YOLO systems as necessary for detection problems. YOLO creates incredibly fast predictions for different artifacts in an image using both IoU and NMS with its main limitation being the inability to detect multiple small objects that are close to each other.

2.4.2 YOLO version 4 with Darknet framework.

Cross Stage Partial Network (CSPNet) separates the feature map of the base layer into two parts: a) one part goes through a dense block and a transition layer, b) the other part is combined with transmitted feature map to the next stage. A feature map is a 2D matrix of neurons and it is an important part of CNN, because its convolutional layer receives a block of input feature maps and generates a block of output feature maps. Darknet framework is a custom framework written by Joseph Redmon [6]. It is incredibly versatile and produces state-of-the-art object detection results, although it is not user-friendly. Its main technical specifications are given below:

- Backbone - CSPDarknet53 with Mish activation
- Neck - PANet with Leaky activation
- Plugin Modules - SPP
- V100 FPS - 62@608x608, 83@512x512
- BFLOPs - 128.5@608x608

The original YOLO (You Only Look Once) was constructed by Joseph Redmon (now retired from CV) in a custom framework called Darknet. Darknet is a very flexible research framework written in low-level languages and has produced a series of the best real-time object detectors in computer vision: YOLO, YOLOv2, YOLOv3, and now, YOLOv4. Original YOLO – YOLO combines the problem of drawing bounding boxes and identifying class labels in one end-to-end differentiable network. *YOLOv2 - YOLOv2* proposes a number of iterative improvements on top of YOLO including BatchNorm, higher resolution, and anchor boxes. YOLOv3 - YOLOv3 is built upon previous models, adding an objectness score to bounding box prediction. It also, adds connections to the backbone network layers, and predicts at three separate levels of granularity to improve performance on smaller objects. YOLOv4 is a series of computer vision technique additions that work with a few small novel contributions. All YOLO models are object detection models trained to utilize an image and search for a subset of object classes, which are enclosed then, in a bounding box for class identification. Typically, object detection models are trained and evaluated on the COCO dataset [7], containing a broad range of 80 object classes. Object detection models can generalize to new object detection tasks when exposed to new training data.

All object detectors receive an image as input and compress features through a convolutional neural network backbone. In image classification, these backbones are the end of the network and prediction can be made from them. In object detection, multiple bounding boxes need to be drawn around images, along with classification, so that feature layers of the convolutional backbone can be mixed and held

together (neck). Object detectors can be split into two main categories: i) one-stage detectors and ii) two-stage detectors. A one-stage detector, on the other hand, just needs one run through the neural network to anticipate all of the bounding boxes. That's a lot speedier and better suited to mobile devices. In several disciplines, including autonomous driving, a two-stage detector seeks to generate multiple bounding boxes of objects in a given image. The detection happens in the head, where the two-stage detectors decouple the task of object localization and classification for each bounding box, while the one-stage detectors predict the object localization and classification at the same time. YOLO is defined as a one-stage detector, hence, You Only Look Once.

In our application, Darknet framework was used, because training YOLOv4 in TensorFlow, Keras, and PyTorch frameworks was still under construction. The following backbones exist for the YOLOv4 object detector (Table 2-1):

- CSPResNext50
- CSPDarknet53
- EfficientNet-B3

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	1058 K	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	27.6 M	950 K	52 (26.0 FMA)	66
EfficientNet-B3 (ours)	512x512	1311x1311	12.0 M	668 K	11 (5.5 FMA)	26

Table 2-1: YOLOv4 object detector backbones

2.4.3 YOLO version 5 with Pytorch framework.

This is the first release of models in the YOLO family to be written in PyTorch, rather than PJ Reddie's Darknet. YOLOv5 [8] was initially implemented in PyTorch, and it benefits from the established PyTorch ecosystem, in terms of support and deployment. Iterating on YOLOv5 may also be easier for the broader research community because it is a more widely known research framework. YOLOv5 is incredibly quick. We observe inference times up to 0.007 seconds per image, in a YOLOv5 Colab notebook running a Tesla P100, which translates to 140 frames per second (FPS). YOLOv4 on the other hand, after being converted to the same Ultralytics PyTorch library, achieved 50 FPS. Also, YOLOv5 includes five different model sizes: YOLOv5s (smallest), YOLOv5m, YOLOv5l, YOLOv5x (largest).

A recommended splitting percentage is, to apply 70% of data in the training set, 20% in the validation set, and 10% in the testing set. Then the training and validation folders are copied and pasted into the data directory, in the project folder. For the training process of the annotated dataset, a conversion in

YOLO format is needed. To convert the annotation format from XML to YOLOv5 txt format, we used an online tool that automatically generates two text files, containing the full path of all the images in the training and validation folders. These text files are referenced in the configuration yaml file. YAML (a recursive acronym for "YAML Ain't Markup Language") is a human-readable data-serialization language, commonly used for configuration files and applications where data is being stored or transmitted.

For this model we trained: yolov5-m, yolov5-l and yolov5-x. We downloaded the pre-trained weight, for transfer learning, from Ultralytics Google Drive, and we moved our preferred models in the appropriate file, that weights are contained. The YAML configuration file, of the corresponding version of YOLOv5 that used for training, modified, and the number of classes nc has been changed to 1. For the training process of YOLOv5 detector, the training command has been applied. The command options are presented below:

- `Img`: number of training images
- `batch`: determine batch size
- `epochs`: define the number of training epochs. (Note: often, 3000+ are common here!)
- `data`: set the path to our yaml file
- `cfg`: specify our model configuration
- `weights`: specify a custom path to weights
- `name`: result names
- `nosave`: only save the final checkpoint
- `cache`: cache images for faster training
- `device`: to select the training device, "0" for GPU, and "cpu" for CPU.

Once we have completed training, the trained models are saved and then we evaluate how well the training procedure performed by looking at validation metrics such as precision, recall, IOU, average precision (AP), and mean average precision (mAP), which are explained more precisely in the subsection 2.6.

2.4.4 Scaled YOLO version 4.

Wang et al., 2021 [9], present a scaling approach of the YOLOv4 for modifying the structure instead of the depth, the width, and the resolution of the network, while the proposed method is able to deal with the quantitative factors (inference time and average precision) that depend on the equipment of the database. They are focused on the model's construction (image size, number of layers, and number of channels) and optimization, in order to extract the performance and the inference speed. The authors considered, in their network, CSP-ized CNN backbones such as ResNet, ResNext, and the traditional Darknet backbone, and present the "CSP-ized" portion of the network. For big objects detection, into big images, the increase of depth and number of stages in the CNN backbone/neck is crucial, while increasing the width reportedly has much lower effect. This process allows the dynamically adjustment of the width and depth according to the scaled-up input size and the number of stages, as also to the real-time inference speed requirements.

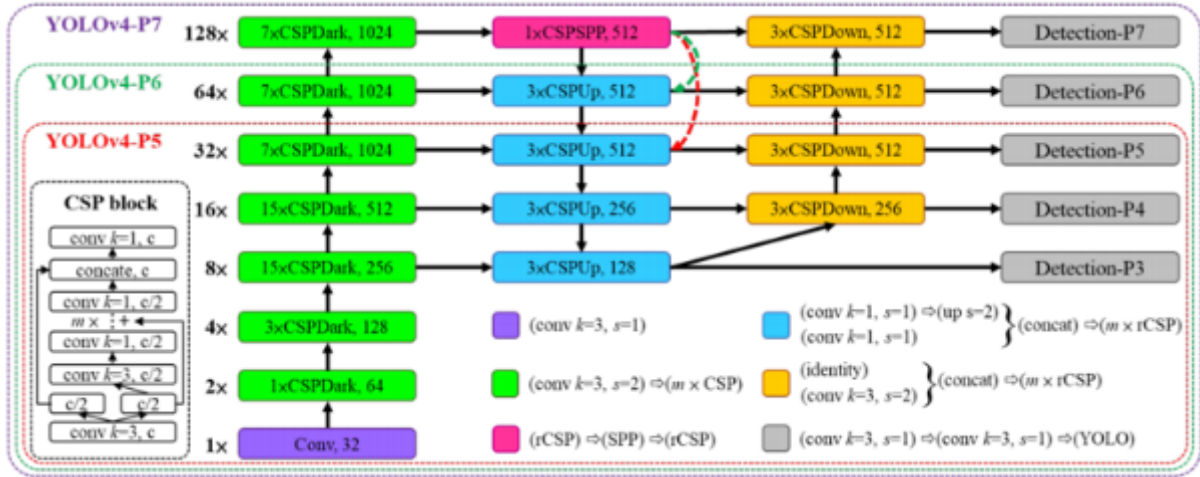


Figure 2-5: Architecture of YOLOv4-large, including YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. The dashed arrow means replace the corresponding CSPUp block by CSPSPP block. [9]

2.4.5 Efficient Det

TensorFlow2 Object Detection API is an extension of the TensorFlow Object Detection API, where a state-of-the-art object detection model collection can be trained under a unified framework (EfficientNet). EfficientNet [10] is a ConvNet, released by Google Brain Team that forms the backbone of the EfficientDet architecture, able to learn the scaling process of ConvNet architectures. There are many actions for adding more parameters to a ConvNet such as to: i) create wider layers, ii) create deeper layers, iii) insert high resolution images or iv) make a combination of these improvements.

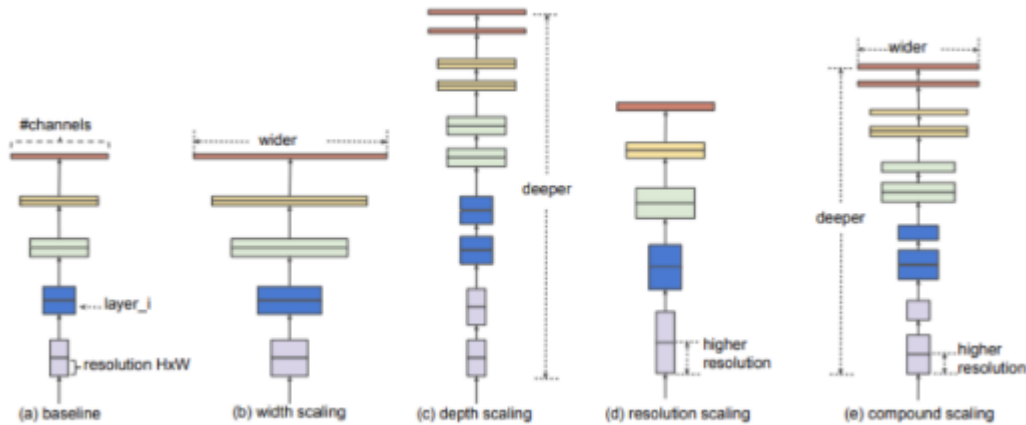


Figure 2-6: Model Scaling [10]

The exploration of all these possibilities can be quite tedious for machine learning researchers. The paper [10] seeks to optimize downstream performance given free range over depth, width, and resolution while staying within the constraints of target memory and target FLOPs. They found that their scaling methodology improves the optimization of previous ConvNets as well as their EfficientNet

architecture. The EfficientNet looks like a good backbone to build upon. It efficiently scales with model size and outperforms other ConvNet backbones.

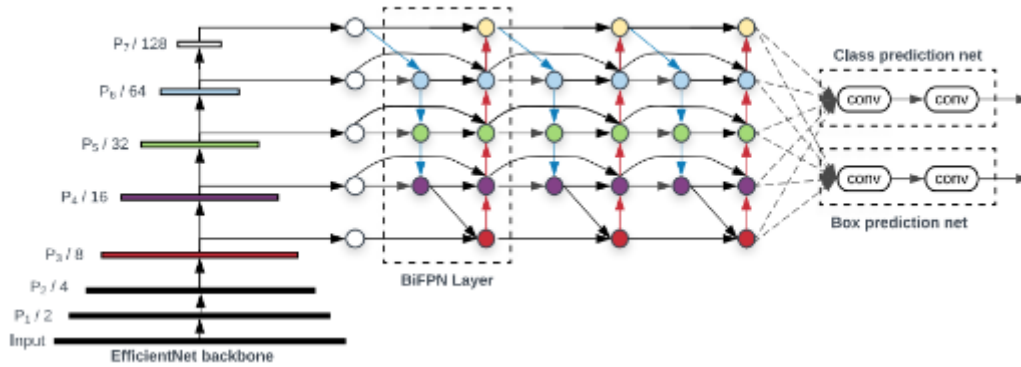


Figure 2-7: EfficientDet architecture [10]

EfficientDet Feature Fusion

Feature fusion combines representations of an image at different resolutions. Typically, the fusion uses the last few feature layers from the ConvNet, but the exact neural architecture may vary.

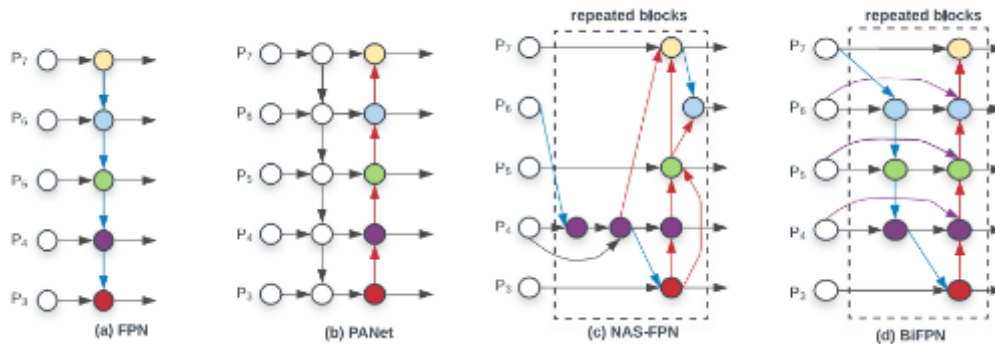


Figure 2-8: Feature network design

FPN (Figure 2-8) is able to fuse features with a top-down flow. Feature fusion can flow backwards and forwards from smaller to larger resolution, in the PA net. NAS-FPN is a feature fusion technique based on a neural architecture. EfficientDet uses “intuition” (and presumably many, many development sets) to edit the structure of NAS-FPN to settle on the BiFPN, a bidirectional feature pyramid network. The EfficientDet model stacks these BiFPN blocks on top of each other. In the model scaling procedure, the number of blocks varies. Additionally, there is a hypothesis by the authors of paper [10] that there might be a variety of certain features and feature channels in the amount that they contribute to the end prediction, so they add a set of weights at the beginning of the channel that are learnable. The EfficientNet checkpoints are further leveraged with feature fusion and all components of the architecture are efficiently scaled. Finally, these model weights are pre-trained on COCO, a generalized

image detection dataset. Each model consists of a model name, a base pipeline file, a pre-trained checkpoint, and a batch size. The base pipeline file is a shell of a training configuration, specific to each model type, provided by the authors of the TF2 OD repository. The pre-trained checkpoint is the location of the pre-trained weights file, saved from the pre-training on the COCO dataset. Starting from these weights we fine tune into our particular custom dataset task. From the pre-training process, our model does not need to start from square one in identifying what features might be useful for object detection. With all of those inputs defined, the base pipeline file is then edited to be in an appropriate form for our custom data and the pre-trained checkpoint. Training for a longer time need increasing the number of steps while training faster, increasing the batch size to a level that the GPU can handle. [11]

2.4.6 Detectron2

Detectron2 [12] is a PyTorch-based scalable computer vision model library, the second version of the Detectron project, which started as a Caffe2 project and provides state-of-the-art detection and segmentation algorithms. Using the Detectron2 platform, custom state-of-the-art computer vision technology can be incorporated into the workflow. All models from the original Detectron are included in Detectron2, including Faster R-CNN, Mask R-CNN, RetinaNet, and DensePose. A range of tasks related to object detection is supported from Detectron2, with boxes and instance segmentation masks, as well as human pose prediction, like the original Detectron. Detectron2 also includes support for semantic segmentation as well as panoptic segmentation, which incorporates semantic and instance segmentation. It is flexible and extensible, and also able to provide fast training on single or multiple GPU servers.

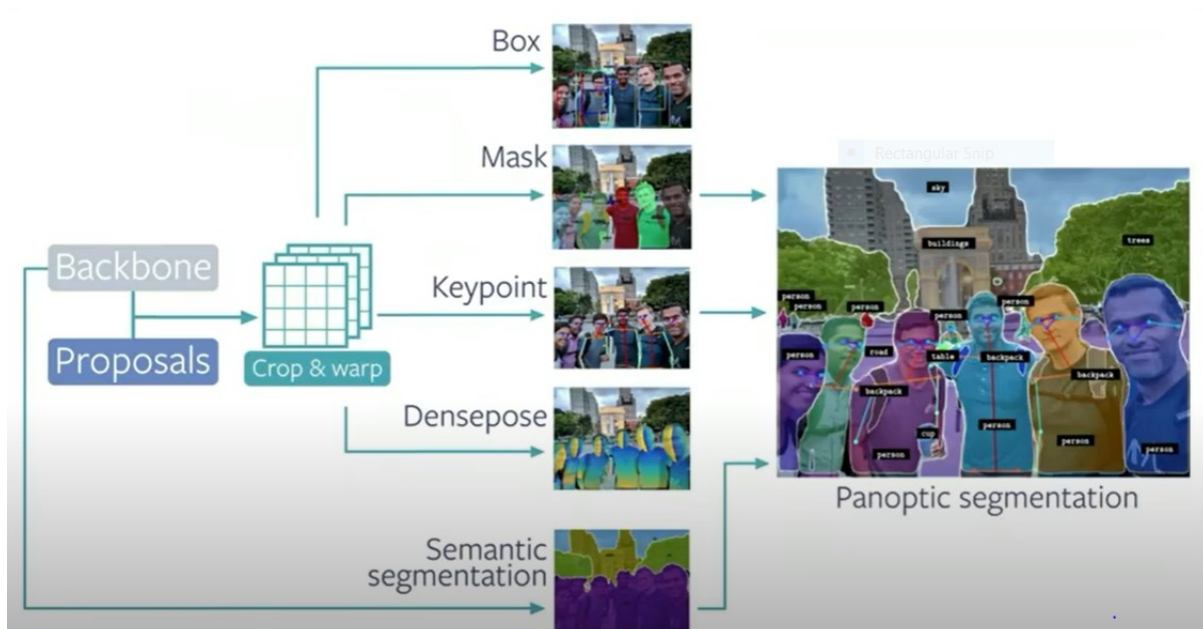


Figure 2-9: Detectron 2 pipeline [12]

Facebook AI Research (FAIR) constructed Detectron2 to help researchers quickly implement and evaluate new computer vision research. Applications for the following target detection algorithms are included:

- Mask R-CNN
- Fast R-CNN
- Retina-Net
- DensePose
- RPN
- TensorMask

2.5 Google Colab

One of the main problems in complex networks and large datasets is the computational cost. An average performance computer with an average graphics card may need huge amount of time for training. In order to minimize training time, we used Google Collaboration (Colab) a product from Google Research [13]. In Colab, it is able to write and execute arbitrary python code through the browser, and is also well suited to machine learning, data analysis and education. It is a free online cloud based Jupyter notebook environment, providing the ability to train machine learning and deep learning models on CPUs, GPUs, and TPUs. It is as safe, as a private Google Doc. No one can access private Colab notebooks, and Google has the incentive to make it as safe as possible for their reputation, and for selling reasons. As for the information provided by Google's Colab documentation, A GPU provides 1.8TFlops with 12GB RAM, while TPU delivers 180TFlops with 64GB RAM. In Colab, even with free GPU, someone should expect training to be a multi-hour process.

2.6 Metrics

Precision is a measure of, "when a model guesses how often does it guess correctly?", that is the percentage of the predictions are correct.

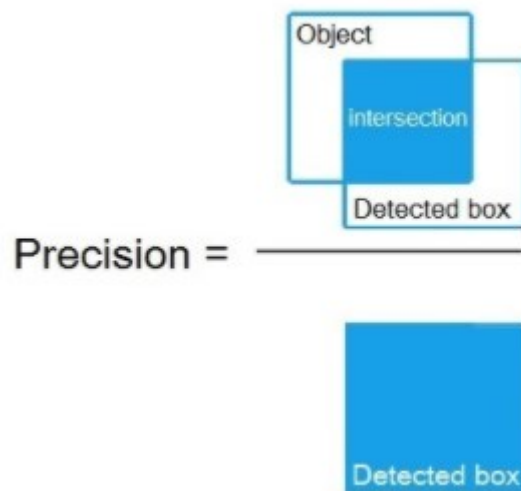


Figure 2-10: Precision metric formula

Recall is a measure of "has a model guessed every time that it should have guessed?", that is measures how good it finds all the positives.

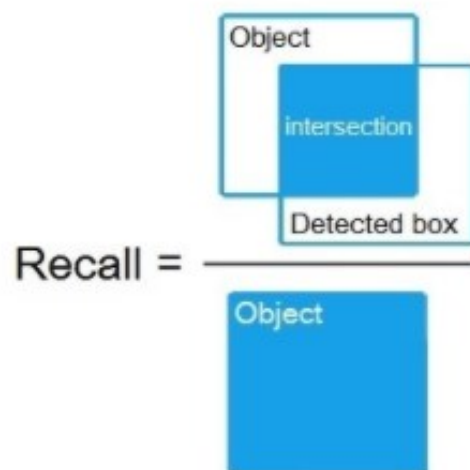


Figure 2-11: Recall metric formula

- *Intersection over Union (IOU)*

IOU is a metric that finds the difference between ground truth annotations and predicted bounding boxes. This metric is used in the most state-of-art object detection algorithms. In object detection, the model predicts multiple bounding boxes for each object, and based on the confidence scores of each bounding box it removes unnecessary boxes, based on its threshold value.

IOU = Area of union / area of intersection

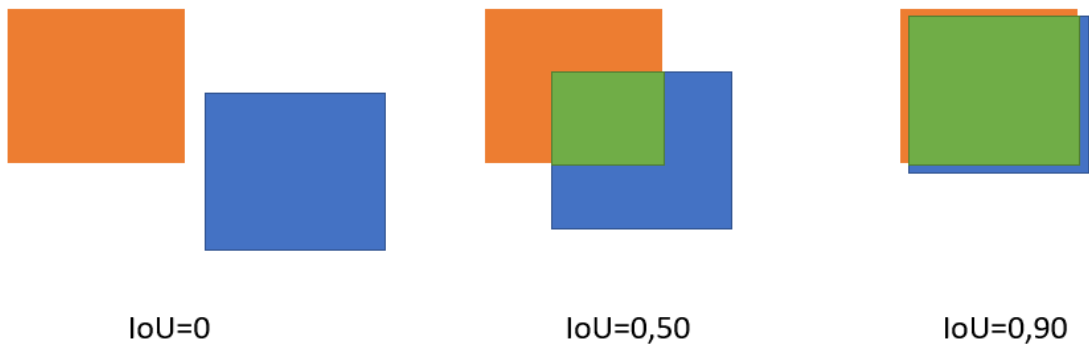


Figure 2-12: A sketch of a graphical depiction of the IoU metric

- *Average Precision (AP)*

To evaluate the detection commonly we use a precision-recall curve. Average precision produces numerical values, and therefore it is easy to compare the performance with other models. Based on the precision-recall curve, AP summarizes the weighted mean of precisions for each threshold with the increase in recall. Average precision is calculated for each object.

- *Mean Average Precision(mAP)*

Mean average precision is an extension of Average Precision. In Average Precision, we only calculate individual objects but in mAP, the precision is referred in the entire model. To find the percentage, of the correct predictions in the model, we are using mAP. Every object has its individual average precision values; we are adding all these values to find the Mean Average Precision.

- *Variations among mAP*

In most of the research papers, these metrics have extensions like mAP iou = 0.5, mAP iou = 0.75, mAP small, medium, large. Below, we present these extensions and the definition of them:

-mAP iou=0.5 model used 0.5 threshold value to remove unnecessary bounding boxes, it is the standard threshold value for most of the models.

-mAP iou=0.75 model used 0.75 threshold value, By using this we can get accurate results by removing bounding boxes with less than 25% of the intersection with ground truth image.

-mAP small model produced mAP score based on smaller objects in the data.

-mAP large model produced mAP score based on larger objects in the data.

2.6.1 FPS

FPS (Frame Per Second) defines how fast an object detection model processes a video and generates the desired output. At the end it only depends on the input data on which OD is performed (e.g., online/offline, video or photo). The calculation of frames per seconds processing speed (FPS) is a way to express how fast the method is. Often you can bump it to the term "real time" which is a way to express that the method can process a video sequence "online" with the assumption of the video having a frame rate of 25 (FPS). The term "real time" is often disliked, because in and of itself it does not state anything.

2.7 Validation Strategy

We separated the data into training, validation, and test set to prevent the model from overfitting and to accurately evaluate it. The training process of a computer vision model depends on the examples of images, where the learning process is applied. A loss function is used to direct the model toward convergence on how near or far away it is from making the right prediction. The model uses the loss function to create a prediction function that maps the pixels from the image to the output. The loss function on the training data will continue to display lower values as the model hyper-specifies to the training set, while the held-out validation set will gradually increase. The training set is the largest set of the dataset, the validation set is a subset used after testing to see how well the model does on photos that weren't used during training, and the test set gives the final results of how the model might do on the validation set after the training.

The recommended allocation of the dataset for this project was 70% of the dataset to train set, 20% of the dataset to validation set, and 10% of the dataset to test set.

3 Results

3.1 Presentation of Datasets

Three datasets with different characteristics were considered here. The first one (Lacmus) is a publicly available dataset consisting of images captured by drones in which humans are shown in a variety of poses and sceneries. All images have been captured from a horizontal shooting from approximately 40-50 meters height. The second and the third datasets were generated by us using a mixture of drone photos found online. The second is a collection of drone photos taken from a lower flight altitude ($<40\text{m}$), whereas the third is a collection of thermal images collected by drones in which humans are depicted. More information about the three datasets is given below.

3.1.1 High altitude dataset

The Lacmus drone dataset (LADD) is a collection of 1036 images captured by drones for pedestrian detection. The dataset was compiled by volunteers from Lisa Alert, Sova, and other organizations. This dataset was exported from a horizontal shooting of a 40- 50 meters height. The pictures depict people in a variety of poses. The VOC format is used for LADD annotations. Photos taken during different seasons of the year (winter, spring, and summer) are included in the dataset. This dataset was downloaded from <https://github.com/lacmus-foundation/lacmus>. Examples of this dataset are shown in Figure 3-1



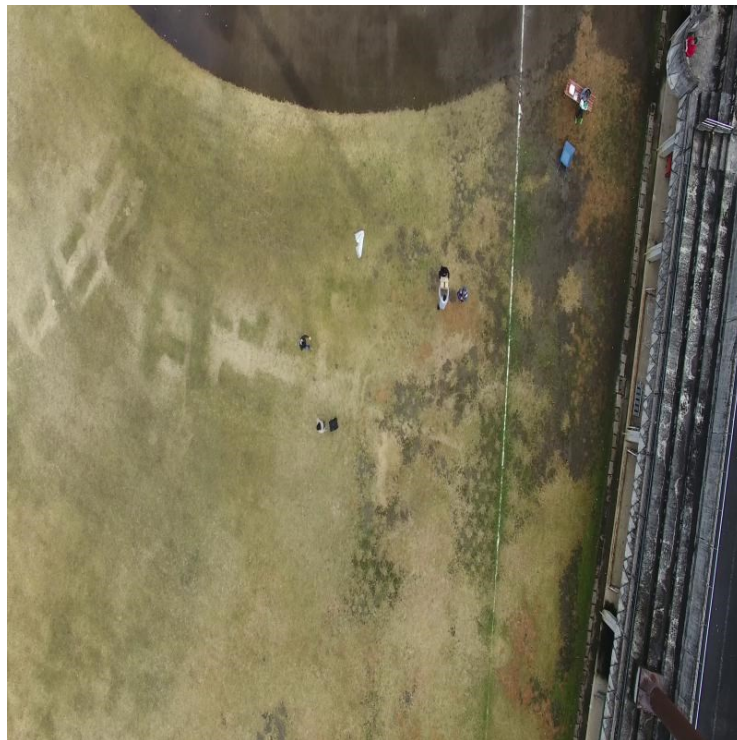
(a)



Figure 3-1: Examples from the Lacmus dataset

3.1.2 Low altitude dataset

A second dataset was created from different datasets with different backgrounds. The databases selected were from Kaggle, older databases or published articles with images of drones depicting humans. Some of these databases are: Okutama database [14], UAVDT database [15], UCF-ARG Data Set [16] and more. The inclusion criteria were the following: (i) to include humans, (ii) to have different backgrounds, and (iii) to be taken at a lower altitude compared to above database. This mixed low altitude dataset contains 800 images. Examples of this dataset are shown in Figure 3-2.



(a)

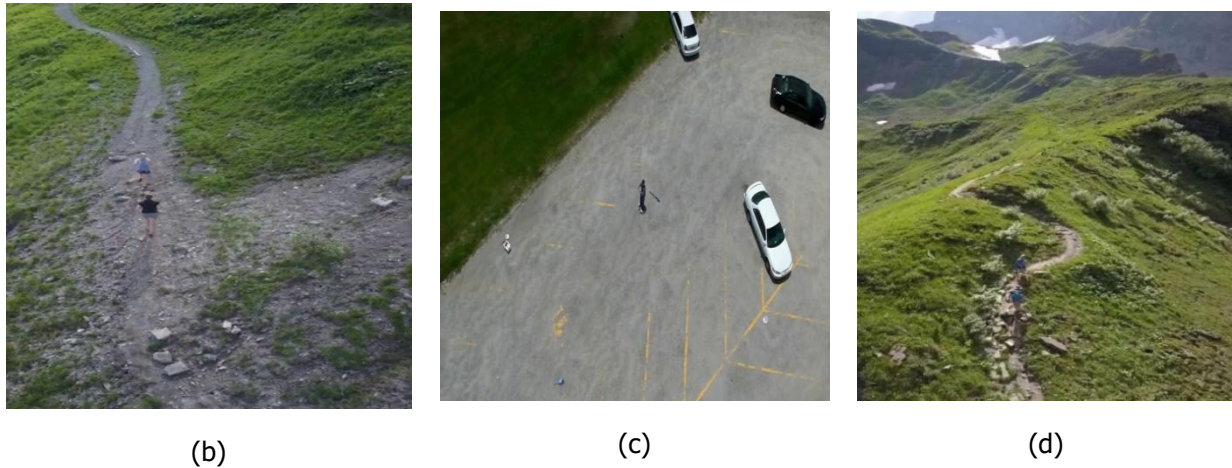


Figure 3-2: Example of Mixed dataset

3.1.3 Thermal images dataset

A dataset comprising of thermal photos from drones was finally produced for the project's needs. It was created out of thermal photos from three separate databases in which humans were the main object. A total of 600 photos were annotated from the following three source: (i) a dataset especially designed for SAR operations using a DJI Matrice 210 Drone outfitted with a FLIR thermal camera. (<https://zenodo.org/record/4327118>). (ii) The Synthetic Depth & Thermal (SDT) collection that includes 40k synthetic and 8k actual depth and thermal stereo pictures that describe human behavior in interior settings. It Includes samples with people laying, sitting, and standing in four distinct room types (living room, bedroom, bathroom, and kitchen), all filmed from a high vantage point (<https://zenodo.org/record/4124309#.YAVgZDmg-bg>). (iii) Four 30-second video sequences of eight persons playing soccer in an indoor arena (court size 4020 metres). Thermal cameras of the type of AXIS Q1922 with a resolution of 640480 pixels and a frame rate of 25 frames per second caught the footage (<https://www.kaggle.com/aalborguniversity/thermal-soccer-dataset>). Examples of these datasets are shown in Figure 3-3 (a).



(a)



(b)

(c)

(d)

Figure 3-3: Example of Thermal dataset

3.2 Object detection results on the high-altitude dataset

Table 3-1: Training models and parameters for Lacmus dataset

below gives information about the OD models that were tested here for their suitability in identifying humans in the high-altitude dataset. Specifically, the table cites models, frameworks, versions, and additional details such as the number of images used and the image resolution.

Models	No. of images	Augmentation	Resolution on Training	Framework	Model version
Yolov5	963	No	416x416	Pytorch	xlarge
Yolov5	2313	Yes	416x416	Pytorch	small
Yolov5	250	No	416x416	Pytorch	medium
Scaled-Yolov4	2313	Yes	416x416	Pytorch	CSP
Scaled-Yolov4	2313	Yes	416x416	Pytorch	CSP
Scaled-Yolov4	2313	Yes	416x416	Pytorch	CSP
Scaled-Yolov4	963	No	416x416	Pytorch	CSP
Scaled-Yolov4	963	No	416x416	Pytorch	CSP
Yolov4	200	No	416x416	Darknet	yolov4.conv.137
Yolov4	2313	Yes	416x416	Darknet	yolov4.conv.137
Detectron2	250	No	416x416	Pytorch	faster_rcnn_X_101_32x8d_FPN_3x
EfficientDet	2313	Yes	416x416	Tensorflow 2	D0
Faster R-CNN	2313	Yes	416x416	Tensorflow 1.5	rfcn_resnet101
Faster R-CNN	2313	Yes	416x416	Tensorflow 1.5	faster_rcnn_inception_v2
Faster R-CNN	2313	Yes	416x416	Tensorflow 1.5	ssd_mobilenet_v2
MobileNet	2313	Yes	416x416	Tensorflow 1.5	ssd_mobilenet_v2
MobileNet	963	No	416x416	Tensorflow 1.5	rfcn_resnet101_coco_2018_01_28

Table 3-1: Training models and parameters for Lacmus dataset

As it can be observed in the table above, the effectiveness of the models in different sample sizes (250, 963 and 2313) was also investigated. The optimal image resolution after testing was proved to be the 416x416. All images in dataset have been resized in this size, improving the performance of each algorithm. The frameworks used were Pytorch, Darknet, Tensorflow 2, and Tensorflow 1.5.

Table 3-2 below presents the results of the aforementioned models in the high-altitude dataset. The order of the results is the same as in Table 3-1. In addition to precision, recall, and mAP, the iteration, and batch size were also reported. This was done to show how mAP varies when the batch size and the number of iterations change. EfficientDet and FasterR-CNN did not manage to converge thus providing inferior results (that are actually omitted from the table below).

Models	Iterations	Batch size	Precision	Recall	mAP
Yolov5	400	16	0,55	0,657	0,61
Yolov5	100	16	0,5613	0,7445	0,7177
Yolov5	250	16	0,498	0,73	0,6648
Scaled-Yolov4	300	16	0,5754	0,7383	0,6943
Scaled-Yolov4	200	32	0,5139	0,7886	0,7549
Scaled-Yolov4	350	40	0,6262	0,8024	0,7802
Scaled-Yolov4	500	32	0,641	0,801	0,794
Scaled-Yolov4	800	8	0,676	0,789	0,785
Yolov4	2000	16	0,5062	0,6226	0,5724
Yolov4	2000	64	0,67	0,65	0,6018
Detectron2	1500	16	n/a	n/a	0,5252
MobileNet	10000	12	0,018	0,08	0,02

Table 3-2: Results for Lacmus dataset

The best performing model for this dataset was scaled-YOLOv4 trained on 500 iterations and using the CSP architecture. Its performance was 0.794 mAP with an optimal combination of 32 batch sizes and 963 images. It should be mentioned that other versions of the scaled-YOLOv4 also achieved good performances (0,6943-0,794), whereas YOLOv5 accomplished the next best performances (0,61-0,7177). In Figure 3-4, Figure 3-5, and Figure 3-6, the performances of the best performing models are presented.

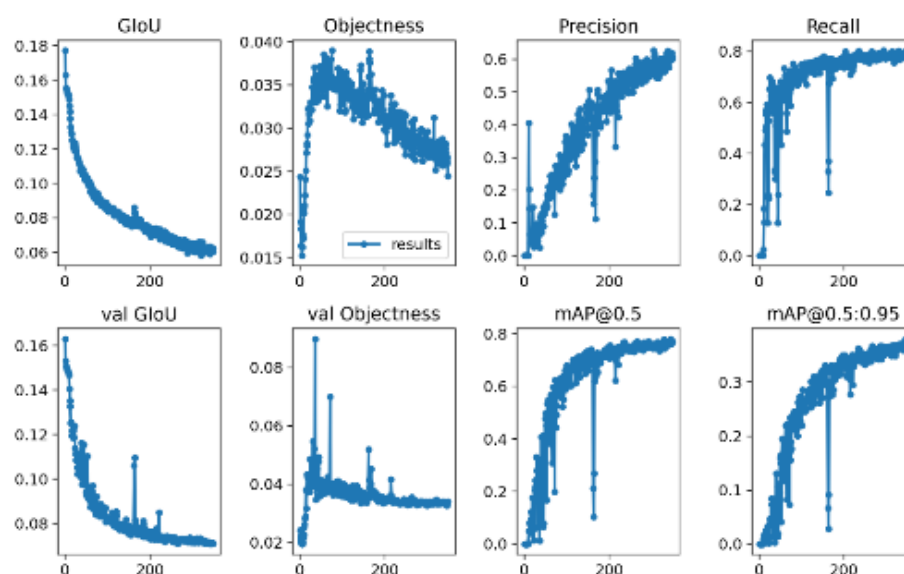


Figure 3-4: Output metrics for scaled-YOLOv4 with 32 batch size, CSP pre-trained weights

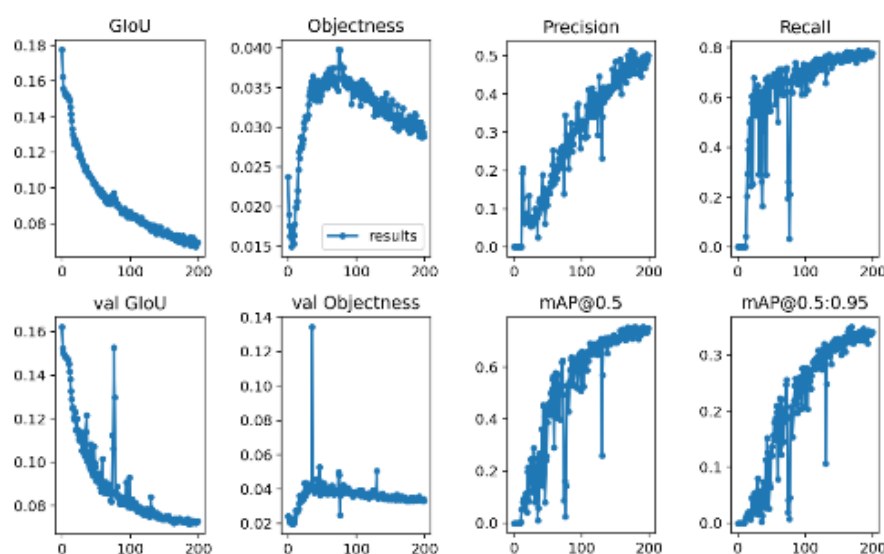


Figure 3-5: Output metrics for scaled YOLOv4 with 32 batch size, CSP pre-trained weights

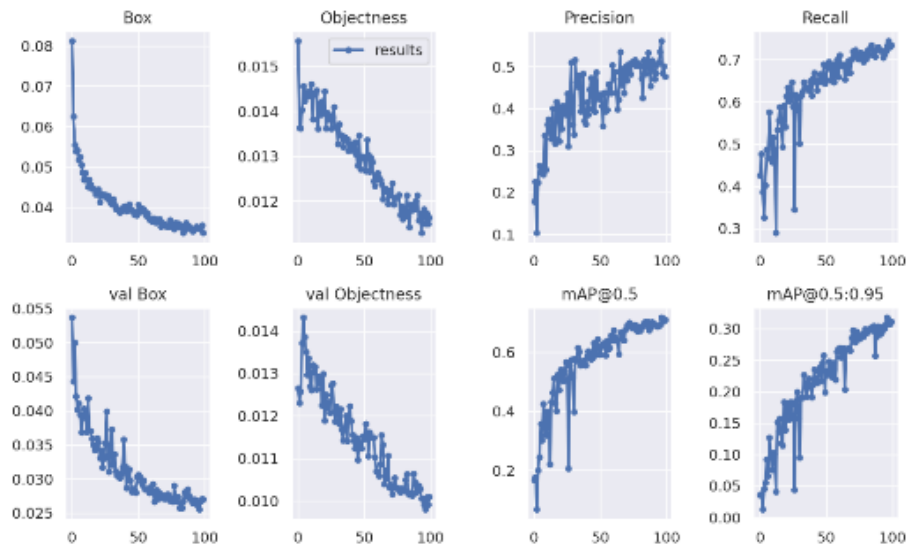


Figure 3-6: Output metrics for YOLOv5 with 2500 images, small pre-trained weights

3.3 Object detection results on the low altitude dataset

Table 3-3 cites the OD models that were trained using images from the low altitude dataset. The table summarizes all the main parameters of the models, including frameworks, versions, and additional details.

Models	No. of images	Augmentation	Resolution on Training	Framework	Model version
Yolov5	583	No	416x416	Pytorch	the small backbone but without its weights
Yolov5	583	No	416x416	Pytorch	large
Yolov5	719	No	416x416	Pytorch	small
Faster R-CNN	719	No	416x416	Tensorflow1.5	rfcn_resnet101
Faster R-CNN	749	No	416x416	Tensorflow1.5	rfcn_resnet101
Scaled-Yolov4	1926	Yes	416x416	Pytorch	CSP
EfficientDet	749	No	416x416	Tensorflow2	D0
Scaled-Yolov4	749	No	416x416	Pytorch	CSP

Yolov5	749	No	416x416	Pytorch	small
Detectron2	749	No	416x416	Pytorch	faster_rcnn_X_101_32x8d_FPN_3x
Scaled-Yolov4	749	No	416x416	Pytorch	P5
Scaled-Yolov4	1926	Yes	416x416	Pytorch	P5
Scaled-Yolov4	800	No	Raw resolution	Pytorch	CSP
Detectron2	800	No	Raw resolution	Pytorch	faster_rcnn_X_101_32x8d_FPN_3x
Yolov5	800	No	Raw resolution	Pytorch	large

Table 3-3: Training models and parameters for Mixed dataset

Similarly, as in the previous dataset, the effectiveness of the models in a smaller sample of images was also tested. Again, the optimal image resolution after testing is 416x416. The frameworks used were Pytorch, Tensorflow 2, and Tensorflow 1.5. Table 3-4 cites the performance of the aforementioned models in the low altitude dataset. Faster R-CNN did not converge in our dataset and therefore its performance is omitted from the table below. The same applies for EfficientDet.

Models	Iterations	Batch size	Precision	Recall	mAP
Yolov5	500	64	0,739	0,959	0,96
Yolov5	500	24	0,823	0,97	0,974
Yolov5	300	64	0,775	0,902	0,924
Scaled-Yolov4	300	32	0,443	0,881	0,839
Scaled-Yolov4	200	16	0,281	0,692	0,546
Yolov5	300	16	0,354	0,701	0,601
Detectron2	1500	64	-	-	0,4225
Scaled-Yolov4	50	16	0,296	0,701	0,518
Scaled-Yolov4	50	8	0,342	0,85	0,791
Scaled-Yolov4	50	6	0,538	0,929	0,91
Detectron2	1500	64	-	-	0,63936
Yolov5	300	32	0,913	0,888	0,925

Table 3-4: Results for mixed dataset

The best model for this dataset was YOLO v5-large version, trained on 500 iterations. The performance of YOLOv5 was 0.974 mAP with an optimal combination of 24 batch sizes and 583 images. Other versions of the YOLO v5 also achieved very good performances (0,924-0,974). Scaled-YOLOv4 also accomplished satisfactory performance (0,518-0,91). In the following figures some of the best results, are presented.

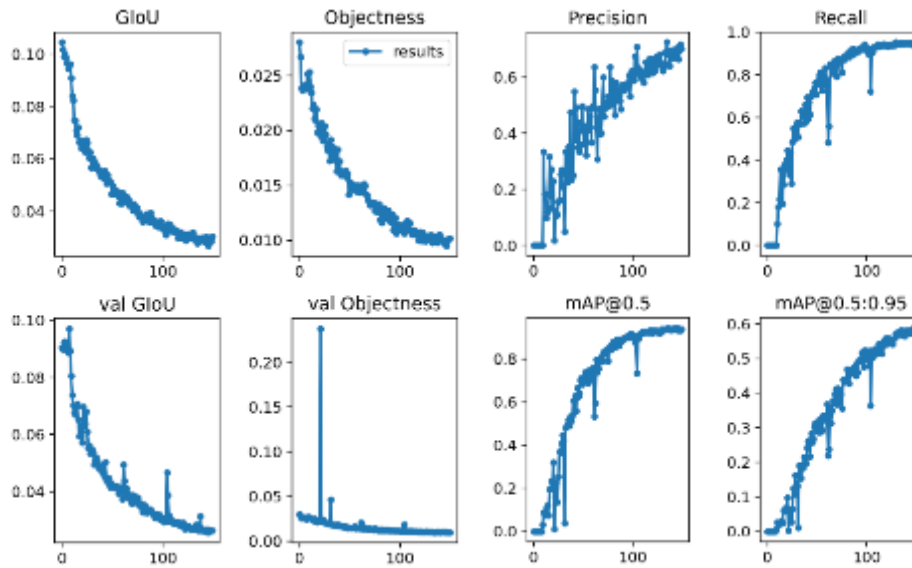


Figure 3-7: Output metrics for scaled YOLOv4 with 200x200 resolution, CSP pre-trained weights, 150 epochs, and 800 images.

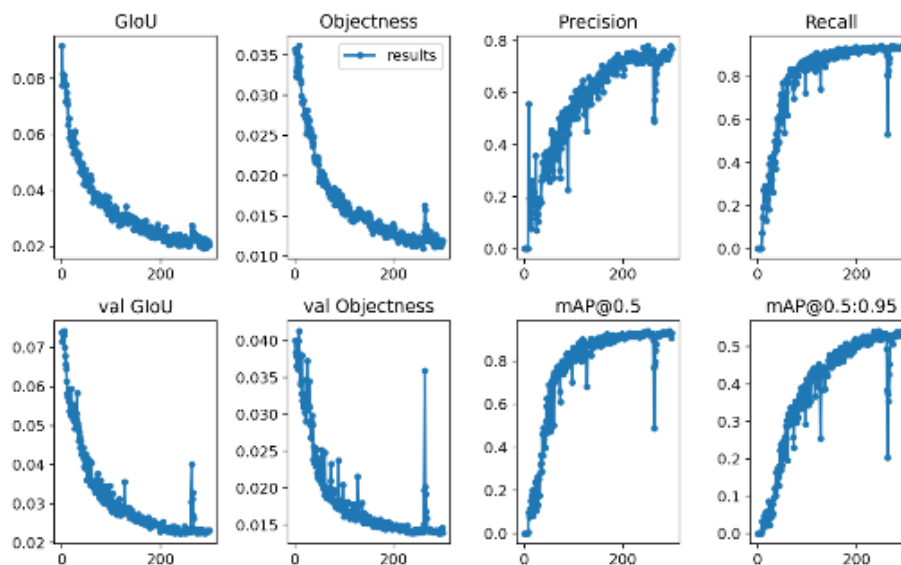


Figure 3-8: Output metrics for scaled YOLOv4 with 16 batch size, CSP pre-trained weights

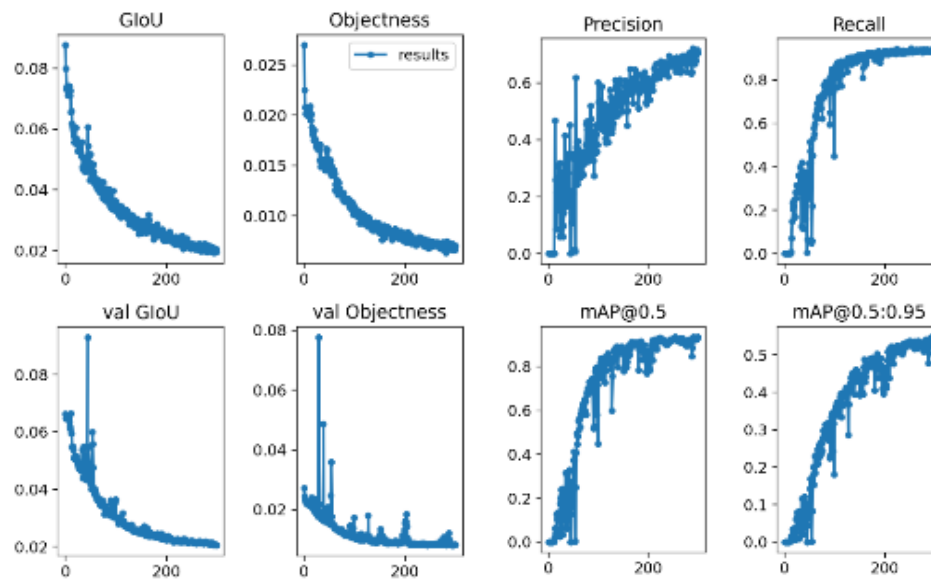


Figure 3-9: Output metrics for scaled YOLOv4 with 16 batch size, P5 pre-trained weights, and 100x100 resolution.

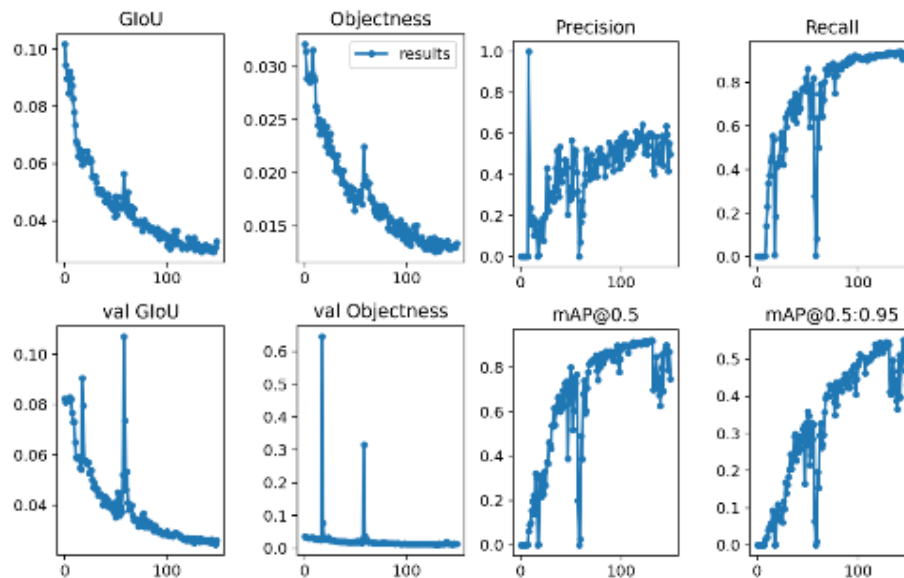


Figure 3-10: Output metrics for scaled YOLOv4 with 500x500 resolution, CSP pre-trained weights, 150 epochs, and 800 images.

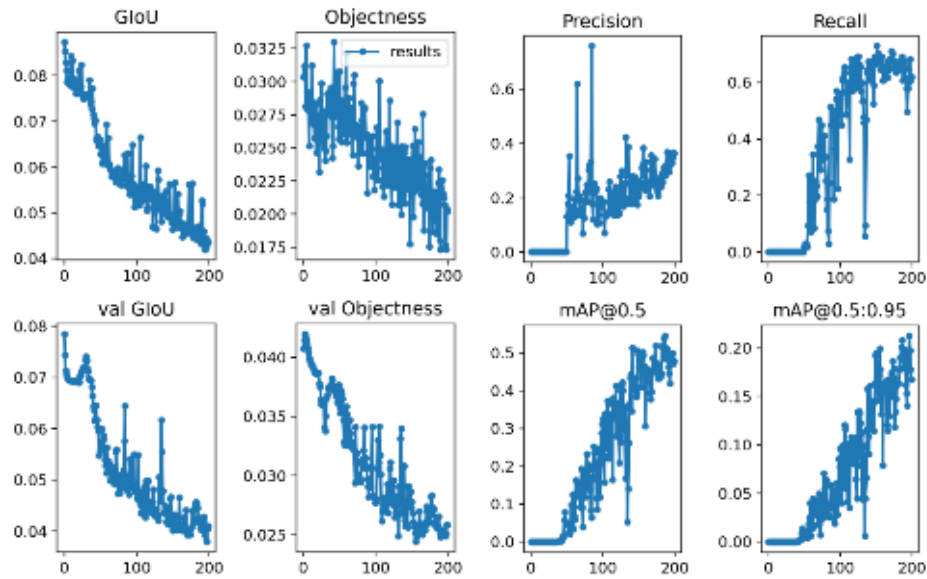


Figure 3-11: Output metrics for scaled YOLOv4 with 749 images, CSP pre-trained weights and 200 epochs.

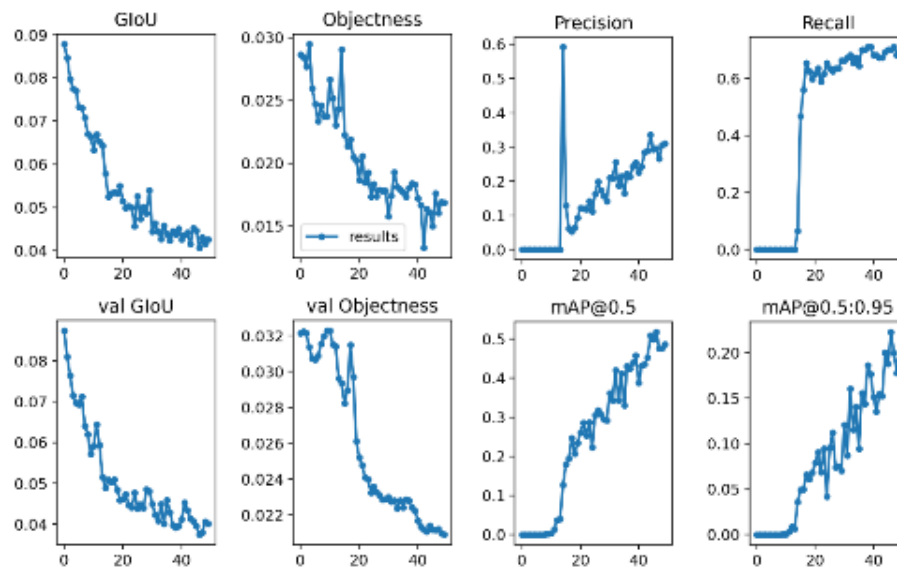


Figure 3-12: Output metrics for scaled YOLOv4 with 749 images, P5 pre-trained weights and 50 epochs.

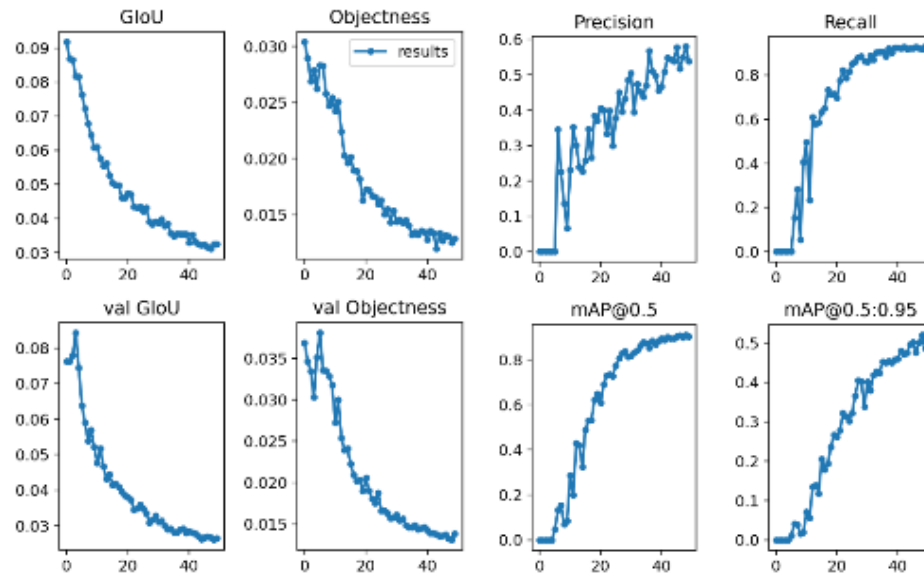


Figure 3-13: Output metrics for scaled YOLOv4 with 8 batch size, CSP pre-trained weights, 50 epochs, and 800 images.

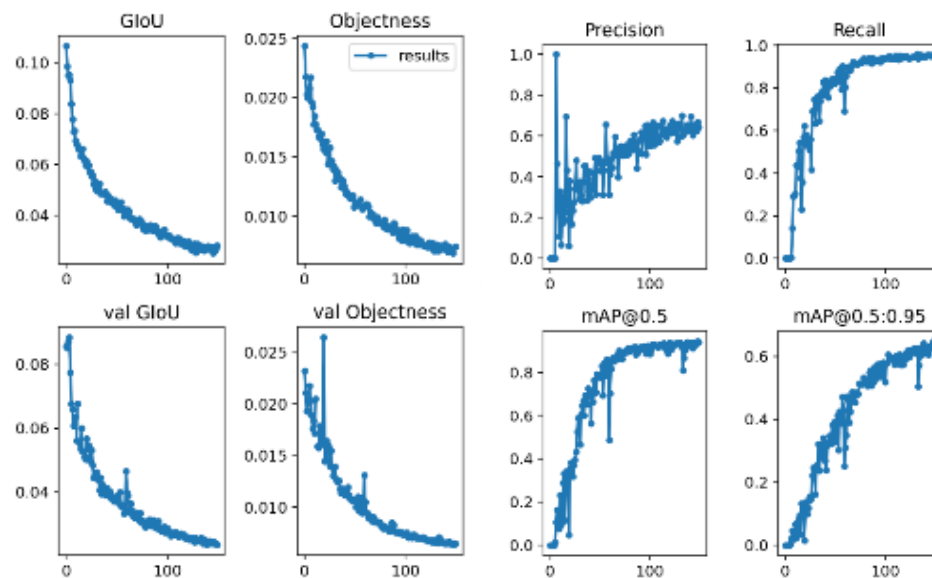


Figure 3-14: Output metrics for scaled YOLOv4 with 10 batch size, CSP pre-trained weights, 800x800 resolution, and 150 epochs.

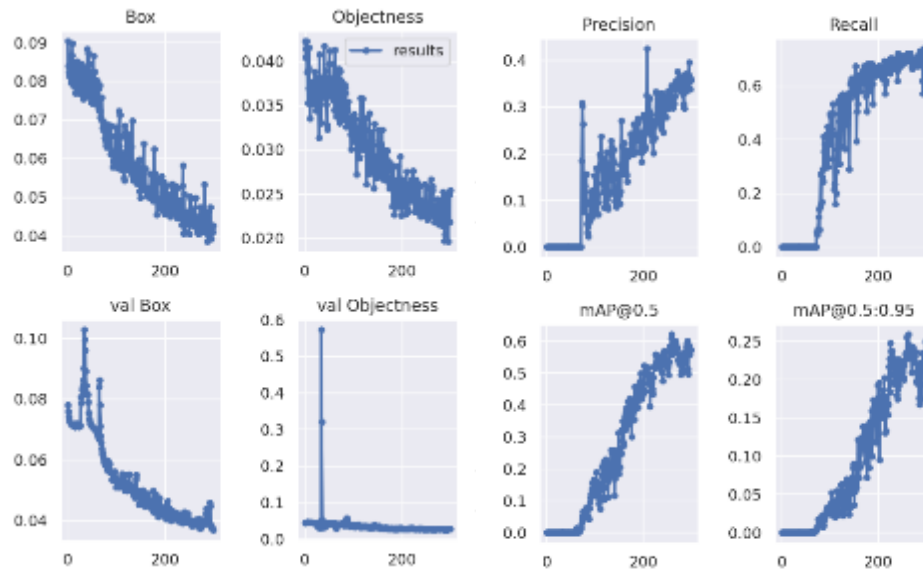


Figure 3-15: Output metrics for scaled YOLOv4 with 10 batch size, CSP pre-trained weights, 800x800 resolution and 300 epochs.

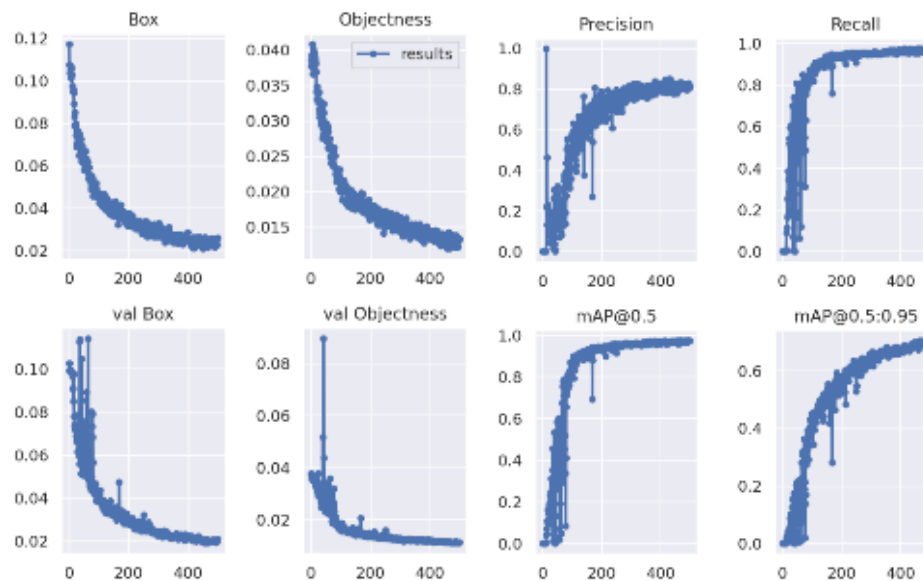


Figure 3-16: Output metrics for scaled YOLOv5 large with 24 batch size, and 583 images.

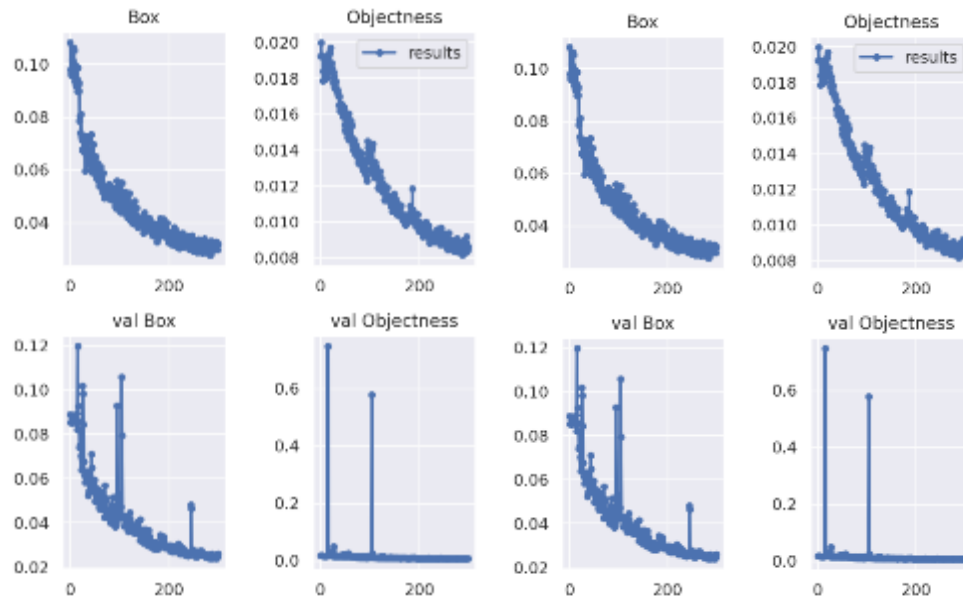


Figure 3-17: Output metrics for large YOLOv5 with 32 batch size, and 300 epochs.

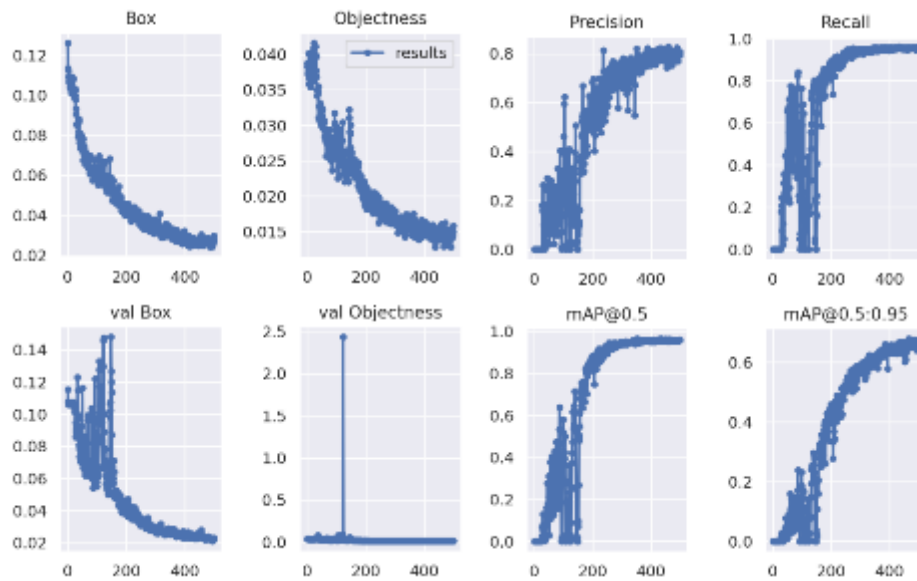


Figure 3-18: Output metrics for YOLOv5 with no weights, 64 batch size, and 583 images.

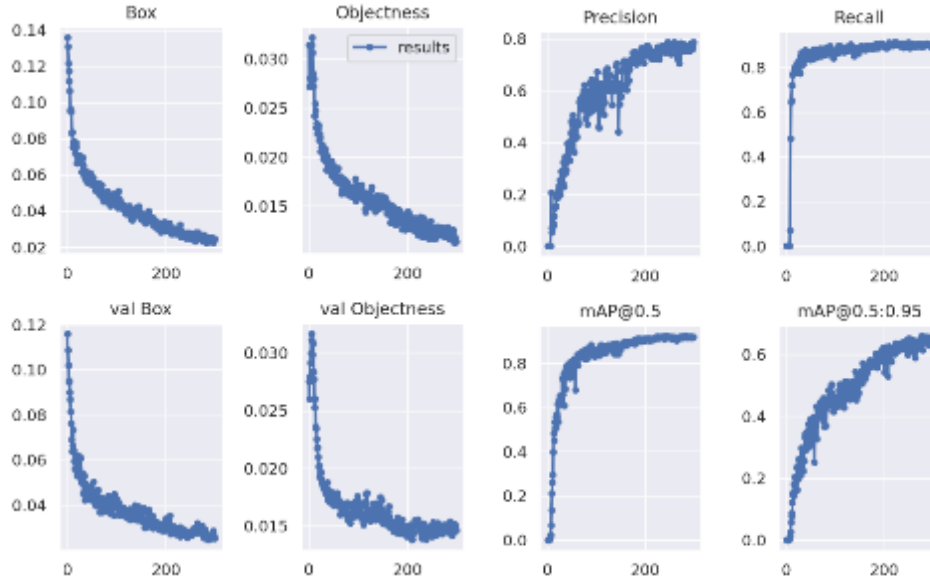


Figure 3-19: Output metrics for YOLOv5 with 64 batch size, small pre-trained weights and 300 epochs.

Based on the results so far in both the low- and high-altitude datasets, Scaled-Yolov4 was the best performer in the high-altitude dataset, whereas it was the second-best performing model in the low altitude dataset (with Yolov5-large being the best). Finally, Scaled-Yolov4 (Pytorch-CSP) was selected as the overall best model due its increased computational efficiency (compared to Yolov5-large) and due to its high performances in the task of human detection using drones' images taken at both high and low flight altitudes. Thus, scaled-Yolov4 was used in the subsequent analysis of the deliverable.

3.4 Performance with respect to different image resolutions and fps

Table 3-5 cites the performance achieved by the best performer (scaled-Yolov4-Pytorch-CSP) for different image resolutions. The GPU speed, defined as the average processing time (in ms) per image, is also provided.

Resolution	GPU speed (average ms / img)	Batch size	Precision	Recall	mAP
100x100	0.07	16	0,748	0,942	0,939
224x224	0.08	16	0,697	0,951	0,942
500x500	0.1	16	0,598	0,938	0,923
800x800	0.13	16	0,669	0,949	0,944
1000x1000	0.18	16	0,772	0,946	0,947

Table 3-5: Output metrics per input image resolution

As expected, the performance of the model increases when the resolution increases. This is mainly due to the better image quality, that leads to a more efficient algorithm. Figure 3-20 presents the AP performance of the best model with respect to the processing time that is needed on average per image.

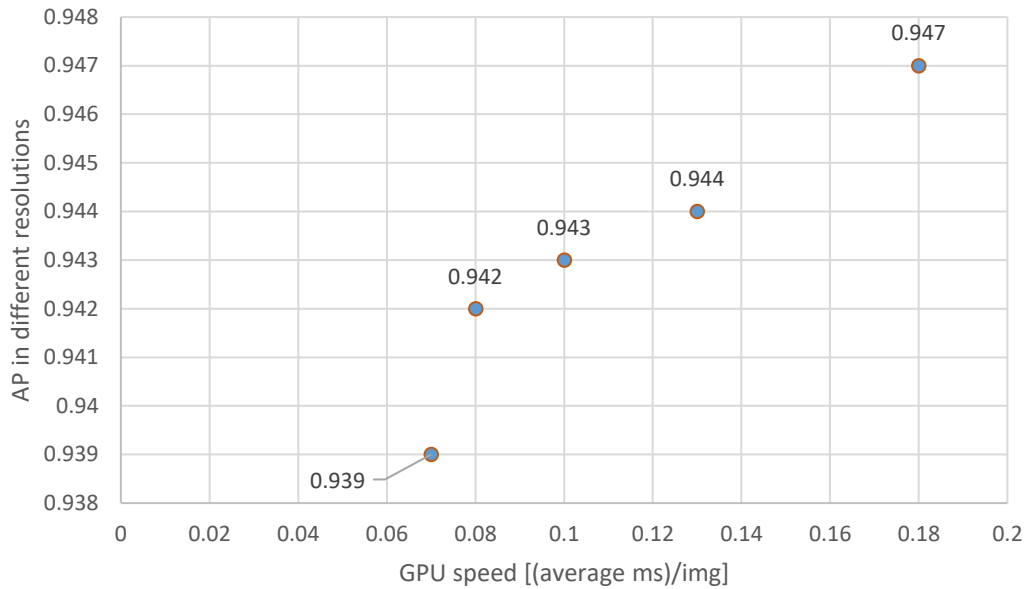


Figure 3-20: Average precision of Scaled-Yolov4 per image resolution and GPU speed

It was observed that for lower image resolutions, the model recognizes objects faster while at the same time the model's accuracy reduces. However, this drop in algorithm accuracy is relatively small compared to the observed time difference. So, the conclusion is that when we need to detect objects in real time, we can reduce the image size inside the model. When the detection is done off-line, then there is no need to reduce the image quality. In each of the two cases the detection time depends on the graphics card capacity and it's a decision of the user to find the best trade-off between accuracy and complexity that matches his/her needs.

3.5 Cross dataset testing using scaled-Yolov4

The generalization capacity of the best model is investigated in this section. To achieve the following experimentation was conducted:

- Scaled-Yolov4 was trained on the high-altitude dataset and then it was applied to recognize humans in the low-altitude dataset
- Scaled-Yolov4 was trained on the low-altitude dataset and then it was applied to recognize humans in the high-altitude dataset

- Finally, different image resolutions were considered to investigate the effect of image resolution in the generalization capacity of the models.

Table 3-6 presents the performance of the best model on the aforementioned experiments.

Training	Testing	Version	Resolution	Precision	Recall	mAP
High altitude dataset	Low altitude dataset	CSP	416x416	0.0498	0.0312	0.00246
Low altitude dataset	High altitude dataset	CSP	100x100	0.167	0.0556	0.055
Low altitude dataset	High altitude dataset	CSP	500x500	0	0	0
Low altitude dataset	High altitude dataset	CSP	1000x1000	0	0	0

Table 3-6: Cross dataset weights metric results

It is a fact that the models did not work properly. The model that was trained to recognize small objects, failed to recognize large objects and vice versa. These results were expected, and the conclusion is that certain conditions (such as altitude of the drone, size of the objects, etc.) must be defined appropriately, for a proper working algorithm.

Four indicative examples are given in Figure 3-21, Figure 3-22, **Error! Reference source not found.**, and Figure 3-24 in which one of the models fails to recognize the human objects. Specifically in images taken from higher altitudes (where the size of the object is small), the OD model that was trained on the low-altitude dataset fails to recognize humans and vice versa.



(a) Output of the OD trained on the low-altitude dataset



(b) Output of the OD trained on the high-altitude dataset

Figure 3-21: Identification of humans in images taken at a low flight altitude



(a) Output of the OD trained on the low-altitude dataset



(b) Output of the OD trained on the high-altitude dataset

Figure 3-22: Identification of humans in images taken at a high flight altitude



(a) Output of the OD trained on the low-altitude dataset



(b) Output of the OD trained on the high-altitude dataset

Figure 3-23: Identification of humans in images taken at a high flight altitude



(a) Output of the OD trained on the low-altitude dataset



(b) Output of the OD trained on the high-altitude dataset

Figure 3-24: Identification of humans in images taken at a high flight altitude

These findings necessitate the development of new OD models that can generalize well in both images taken at different flight altitudes. To cope with the challenge, two new approaches were further tested:

- In the first one, a new mixed dataset was generating concatenating all the available images into a single database.
- A new hybrid inference mechanism was designed that combines the outcomes of altitude-dependent local OD models.

The results of both new approaches are presented in the following subsections. The best performing model (scaled-Yolov4) was employed in both approaches.

3.6 Results on the concatenated dataset

As discussed above, the low and the high-altitude datasets were concatenated into one new dataset. Scaled-Yolov4 was trained on the training subset of the concatenated dataset and the performance achieved on the testing subset is shown in Table 3-8.

Dataset	No. of images	Resolution	Augmentation	Framework	Model version	iteration
Concatenated	2132	raw	no	Pytorch	Scaled-Yolov4-CSP	300

Table 3-7: Parameters of the model trained in the concatenated dataset

Dataset	Batch size	Precision	Recall	mAP	Average ms/ image
Concatenated	8	0.64	0.793	0.77	0.03

Table 3-8: Performance of the model trained in the concatenated dataset

The results of the OD model on the concatenated dataset were satisfactory despite the known difficulty of the problem (where multiple objects of varying sizes are to be detected). The achieved mAP was 0.77 and in Figure 2-1 the evolution of all metrics is presented in detail.

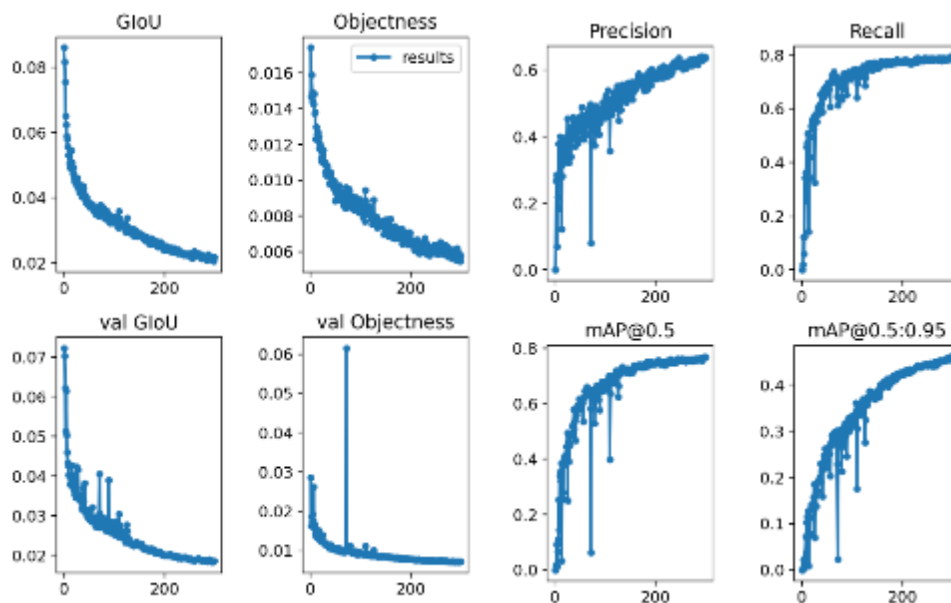


Figure 3-25: Output metrics for the concatenated dataset.

The following pictures show some of the results obtained by the model in the test set.



(a)



(b)



(c)



(d)

Figure 3-26: Indicative successful detections of humans from the OD model trained on the concatenated dataset

3.7 Results using a hybrid inference mechanism

A hybrid inference mechanism was also proposed, designed, and implemented by AIDEAS taking into account the generated inference-ready local models. Specifically, learning is performed separately in each one of the two datasets (A and B comprising photos taken from high and low altitudes,

respectively). All seven OD networks are tested at each dataset and the best performer is chosen per case. To detect objects on a testing photo the following inference approach is then applied:

- (i) the photo is first classified into one of the classes (low or high altitude) based on a predetermined threshold ($h=40\text{m}$).
- (ii) the inference-ready local model associated with the specific altitude class is then applied to produce the final detections.

The proposed methodology can be easily scaled considering more altitude classes (e.g. low, medium and high) depending on the characteristics and dynamics of the object of interest. In our case, the use of two altitude classes were proved to be adequate in terms of the achieved detection performance. Figure 3-27 depicts the proposed Hybrid-DA methodology framework.

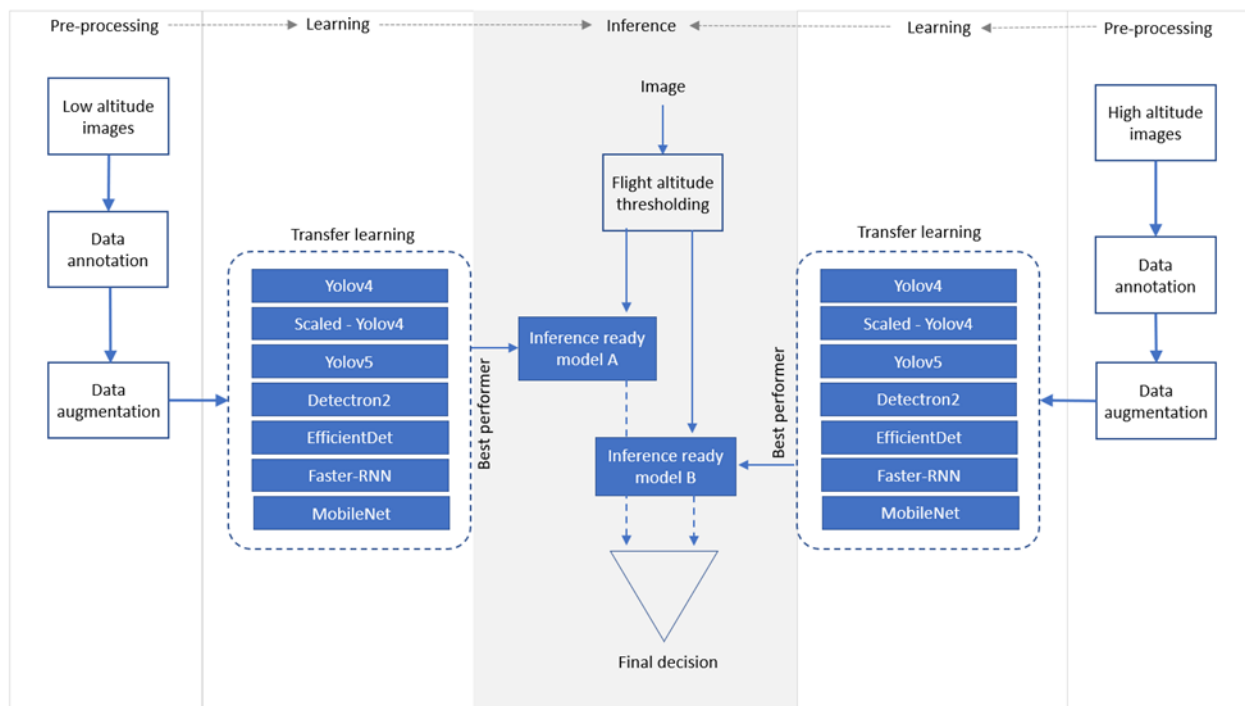


Figure 3-27: Hybrid-DA methodology framework

The proposed Hybrid-DA inference methodology was also applied on the images of the concatenated dataset and its results are given in Table 3-9 in comparison with the best results achieved by the previous testing scenarios. The accuracy for the best local model in the high-altitude database was 79.4%, however these models failed to recognize humans in the low-altitude dataset (mAP: 0.2%). Similarly, the accuracy of the best OD model in the low altitude database it was 97.4%, whereas its generalization capacity on the high-altitude dataset was very limited (mAP: 5.5%). Concatenating the two databases led to a moderate detection performance (77%). The overall best performance was achieved by the proposed hybrid inference technique with a mAP of 86.2% for images taken at both high and low flying altitudes (Figure 3 30).

Training	Testing	mAP (%)
Low altitude data	Low altitude data	97.4
High altitude data	High altitude data	79.4
Low altitude data	High altitude data	0.2
High altitude data	Low altitude data	5.5
Concatenated dataset	Concatenated dataset	77
Hybrid inference on the concatenated dataset	Hybrid inference on the concatenated dataset	86.2

Table 3-9: Best performances achieved by Scaled-Yolov4 on different testing scenarios

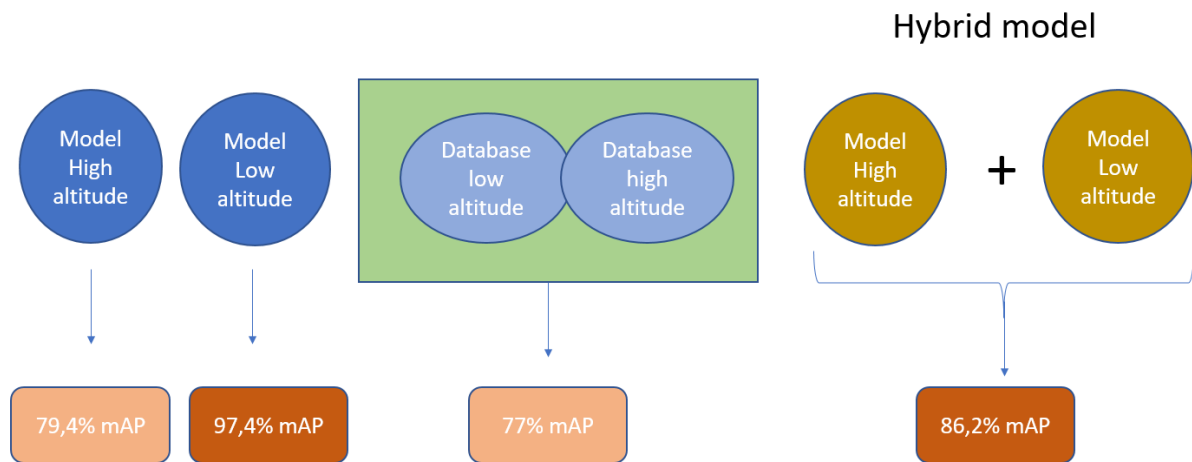
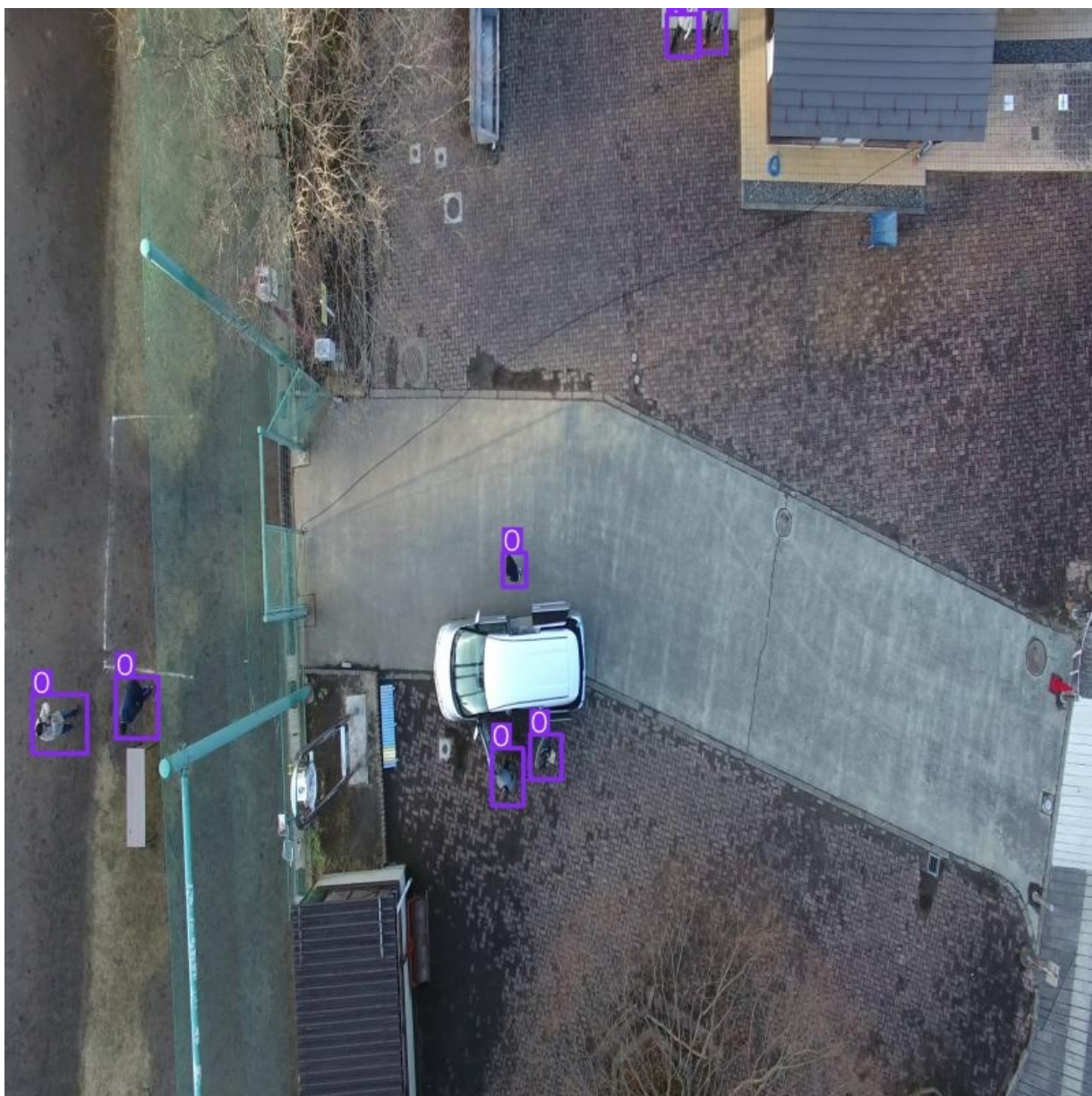


Figure 3-28: Best models outputs



(a)



(b)



(c)

Figure 3-29: Successful detection of humans using the proposed hybrid inference

3.8 Results on thermal images

The capacity of the proposed methodology in detecting humans using a different image source was also tested. Thermal images were finally employed to train the best OD model. Details on the characteristics of the thermal dataset can be found in the subsection 3.1.3. The Scaled YOLOv4 was again chosen to detect humans on the thermal images. Table 3-10 and Table 3-11 cite the parameters of the deployed model and the performance achieved.

Dataset	No. of images	Resolution	Augmentation	Framework	Model version	iteration
Thermal	600	416 x 416	no	Pytorch	Scaled-Yolov4-CSP	300

Table 3-10: Parameters of the model trained in the thermal dataset

Dataset	Batch size	Precision	Recall	mAP	Average ms/ image
Thermal	8	0.94	0.98	0.98	0.03

Table 3-11: Performance of the model trained in the thermal dataset

The mAP achieved was 0.983%, highlighting the excellent detection capacity of the model to detect humans in thermal images as well as the increased discrimination ability of thermography to quantify differences in temperature between humans and the background.

The following pictures show some of the results obtained by the model in the test set.





Figure 3-30: Indicative successful examples of human detection using thermal images

Overall, it was concluded that the proposed OD pipeline is capable of detecting humans under different conditions and using either spectral or thermal information. Thermal images were proved to be more informative leading to higher detection performances.

4 Conclusions

This deliverable presented the overall development status of Task 3.3 (Data-driven analytics applied on UAV imagery using deep learning). Within the development phase of this task, the following accomplishments have been achieved:

Several powerful DL networks were investigated in terms of their ability to detect human in three different datasets (comprising of either high and/or low altitude photos).

Each of the deployed OD network was tested on each of the three datasets and the best performer was identified per case.

A novel hybrid OD inference pipeline was designed and implemented. This pipeline combines the outputs of altitude-dependent local models to increase the generalisation capacity of the OD system.

The predictive performance of all the explored OD pipelines was demonstrated in a thorough experimental set-up. Visual examples of the identified objects were also provided on either spectral or thermal photos under various conditions (backgrounds, seasons, and sceneries).

The AIDEAS is currently preparing a manuscript with all the OD advancements and the proposed hybrid inference mechanism (to be submitted as a journal publication). Next steps include the integration of the DL-based UAV analytics into the SnR CONCORDE platform. These efforts will be carried out in Task 4.5.

Annex I: References

- [1] S. Albawi, T. A. M. Mohammed, and S. Alzawi, "Understanding of a convolutional neural network," *Ieee*, p. 16, 2017.
- [2] K. Weiss, T. M. Khoshgoftaar, and D. D. Wang, *A survey of transfer learning*, vol. 3, no. 1. Springer International Publishing, 2016.
- [3] L. (Eds.). Olivas, E. S., Guerrero, J. D. M., Martinez-Sober, M., Magdalena-Benedito, J. R., & Serrano, *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques*. IGI Global, 2009.
- [4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.
- [5] W. Fang, L. Wang, and P. Ren, "Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments," *IEEE Access*, vol. 8, pp. 1935–1944, 2020, doi: 10.1109/ACCESS.2019.2961959.
- [6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020, [Online]. Available: <http://arxiv.org/abs/2004.10934>.
- [7] "COCO Dataset - Common Objects in Context." <https://cocodataset.org/#home>.
- [8] G. Jocher, "Yolov5." <https://github.com/ultralytics/yolov5>.
- [9] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-YOLOv4: Scaling Cross Stage Partial Network," 2020, doi: 10.1109/cvpr46437.2021.01283.
- [10] 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.
- [11] "TensorFlow 2 Detection Model Zoo." https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.
- [12] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron 2," [Online]. Available: <https://github.com/facebookresearch/Detectron>.
- [13] "Google Colaboratory." <https://colab.research.google.com/>.
- [14] "Okutama-Action: An Aerial View Video Dataset for Concurrent Human Action Detection." <http://okutama-action.org/>.
- [15] H. Yu *et al.*, "The Unmanned Aerial Vehicle Benchmark: Object Detection, Tracking and Baseline," *Int. J. Comput. Vis.*, vol. 128, no. 5, pp. 1141–1159, 2020, doi: 10.1007/s11263-019-01266-1.
- [16] "UCF-ARG Data Set." <https://www.crcv.ucf.edu/data/UCF-ARG.php>.