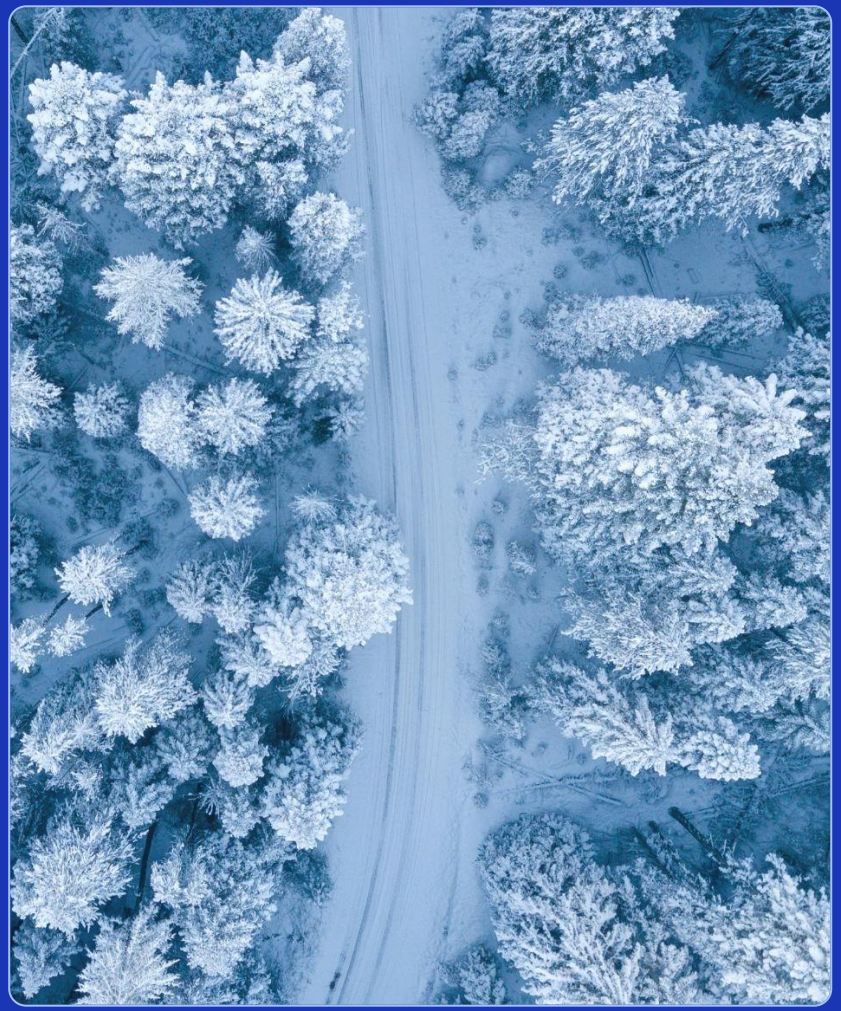
DATA 144

Final Project

Missing Person Identification in Harsh Conditions Using Drone POV Data

Akhil Venkatesh, Amber Le, Anshul Jambula, Arnav Mishra, Sarah Son



Introduction

Every year, thousands of individuals go missing in **low-visibility areas** such as dense forests and ice landscapes. Search-and-rescue operations give rise to challenges due to human error and the vastness of the grounds, which delays the response time of dispatchers and decreases the chances of survival.

Motivations

Objective

Aid search-and-rescue teams in finding missing persons by minimizing response time and optimize survival chances, having real-world impact in saving human lives and informing rescue operations.

Method

Use a computer vision model (YOLOv8) trained on drone-captured images to improve the detection of missing people in low-visibility environments. A mix of classification (human or not) + regression (box).

Benefits

Model may identify complex relationships between images otherwise undetectable to the human eye; also easy generalization over various climates, as well as weather conditions and terrain features.

Key Stakeholders

Search and Rescue Teams

Improved prioritization and efficiency for search team operations & enforcement

Missing Individuals

Enhancing their chances of survival by informing leads on whereabouts

Families & Loved Ones

Providing relief through timely recovery of lost family members

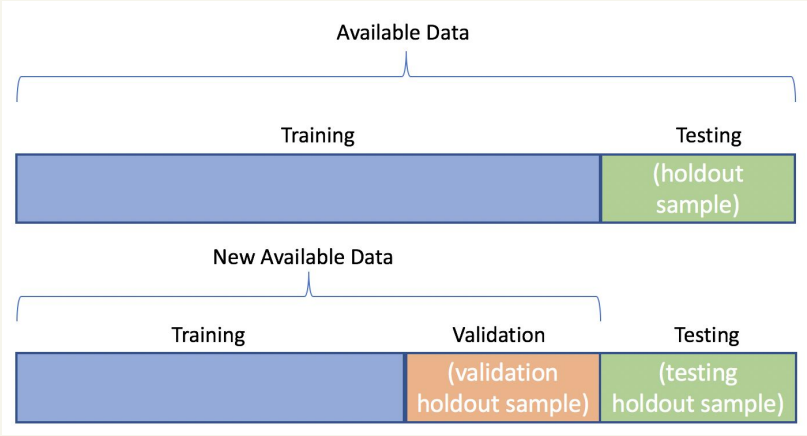
Our Dataset

LADD (Lacmus Drone Dataset) created by Mikhail Shuranov, Denis Shurenkov, Dmitry Ruzhitsky, Victoria Martynova, Ekaterina Bykova, and Georgy Perevozchikov

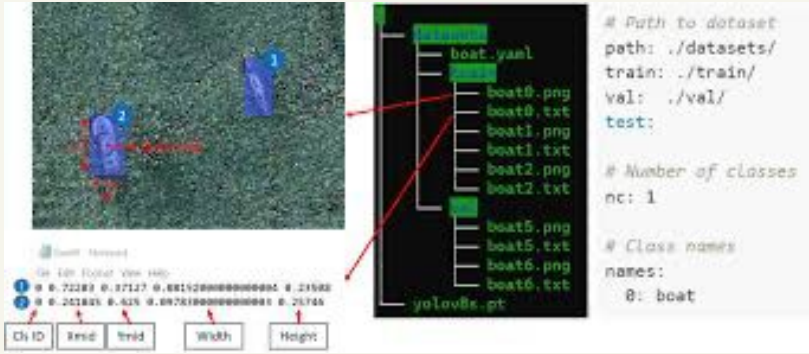
- Captured at 50-100m altitude using DJI Mavic Pro and Phantom drones, with images at 3000×4000 resolution and average human size of 50×100 pixels
- Contains **1365** images, each annotated with bounding boxes in VOC (Xmin, Ymin, Xmax, Ymax) and YOLO formats
- Focuses on 24 common poses of missing individuals, insights developed with input from search and rescue experts

Preprocessing

- Split data into train, val, and test (train_test_split) and then created a data frame of image paths of every image and annotation.
- YOLOv8 requires that the train, test, and val images and annotations are organized within **specific directories** outlined in the *config.yaml* file
  - Therefore, we wrote scripts in order to move all of the images and annotations from the root directory into the proper directories specified in the *config.yaml* file



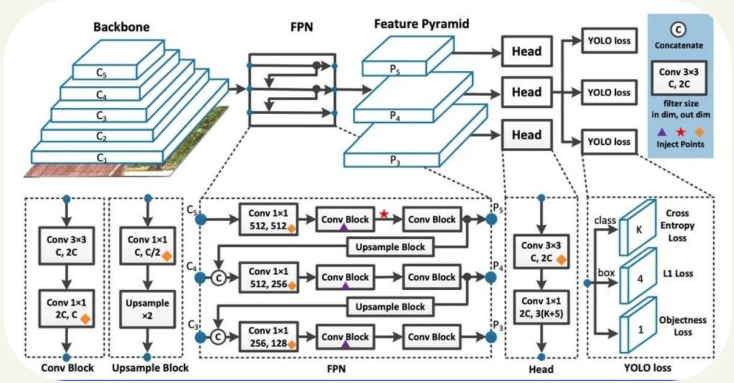

```
{'nc': 1,
 'names': ['human'],
 'path': '/work',
 'train': '/work/train/images',
 'val': '/work/val/images',
 'test': '/work/test/images'}
```



YOLOv8 Architecture

You-Only-Look-Once v8 (YOLOv8) - state-of-the-art object detection model with a modular and efficient architecture

- Input image is divided into a grid (typically 13×13 or 26×26), where each grid is responsible for its encompassing area object detection (segmentation)
- Employs **CNN backbone** (CSPDarknet53) for image feature identification, which captures details like edges, textures, shapes, and other visual patterns critical for accurate recognition and analysis
- **Path Aggregation Network** (PANet) boosts info-flow in from layer-to-layer, grid-to-grid to capture varying object sizes
- The output head is responsible for making final predictions, including bounding boxes, “object-ness” scores, and class probabilities for each grid cell in the feature map



Training Pipeline

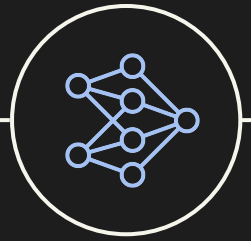
Image-Annotation Pairings

Retrieve matching image-annotation pairs from data directories.

Export pairs into dataframe containing rows of matched pairs.

Train-Val-Test Directory Split

Run scripts that update image & annotation file paths per split indices.



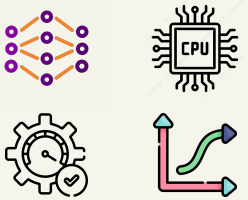
Run Epochs Across Entire Set

Run gradient descent iteration across entire image-annotation set in train split.

Capture Current Performance

Return active metrics per current training epoch, saving checkpoints per 5 epochs.

Hyperparameter Tuning



Epochs
Although it's often recommended to run 100, we initially attempted 20 given resource limitations and still hit a Colab runtime ceiling at ~7 epochs after 13 hours of runtime, so we proceeded to run **5 epochs** to completion, with active metric generation.

Batch Size
We set **batch = -1** to trigger autobatch. This calculates maximum batch size that can run on our device (Colab CPUs).

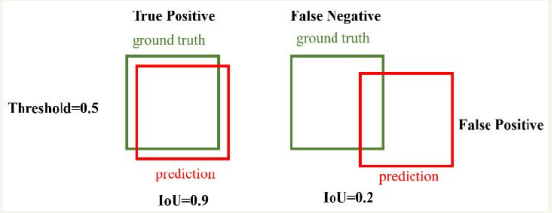
Momentum
We kept the value at the default of **0.937**, enabling the model to stray away from local minima and converge faster.

Patience
Number of epochs for early stopping if the validation loss does not change past a certain threshold to prevent overfitting. We chose **3** - if there were three successive epochs with no improvement, we would early stop.

Learning Rate
Set our learning rate (gradient descent step size) to **0.01,** this is a quite standard value and worked well.

Optimizer
Used **Stochastic Gradient Descent (SGD)** to find local minima while keeping compute manageable due to faster convergence.

Evaluation Metrics



Precision
Metric that measures the proportion of positive predictions made by the model that are correct

Recall
Metric that measures how well the model captures all true positive cases, or sensitivity

$$Precision = \frac{TP}{TP + FP}$$

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

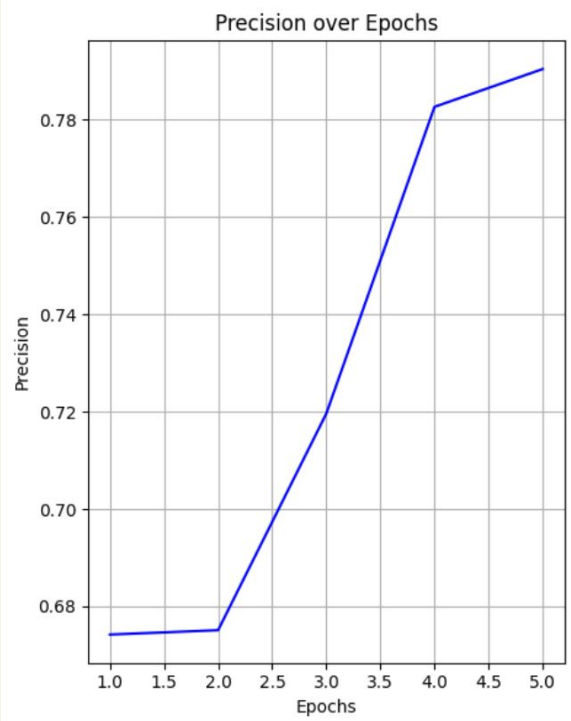
Intersection over Union (IoU)
Metric that determines overlap amount between YOLO prediction box and ground-truth box as a percentage, acts as a confidence score

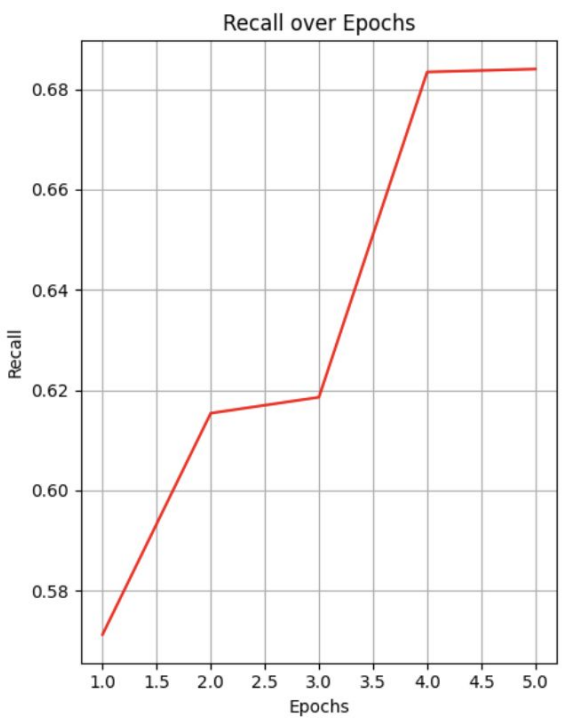
mAP50
Metric that measures the mean Average Precision (mAP) at an IoU threshold of 0.5 (50%)

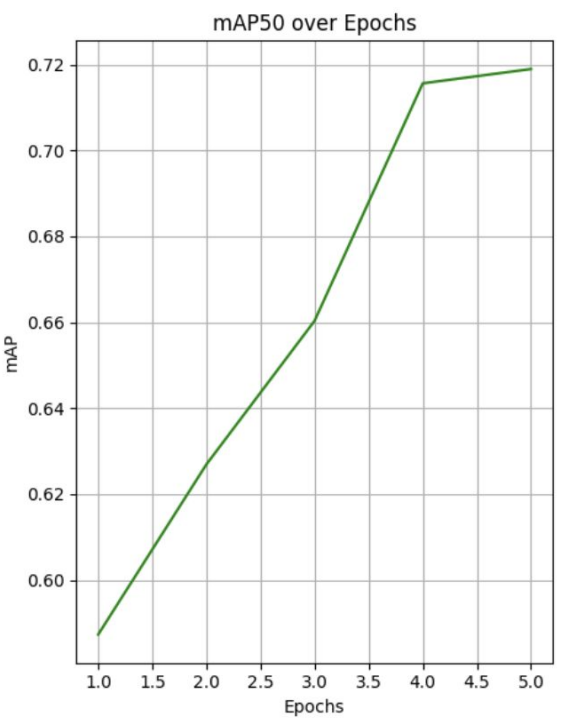
Box Loss
Metric that indicates how closely the model's predictions match the actual object positions

Probability	Case	Confidence Score
$\Pr(\text{Object}) = 0$	If the bounding box is included in the background area	$S_{conf} = 0$
$\Pr(\text{Object}) = 1$	If the bounding box is included in the area where the object exists	$S_{conf} = IOU_{pred}^{truth}$

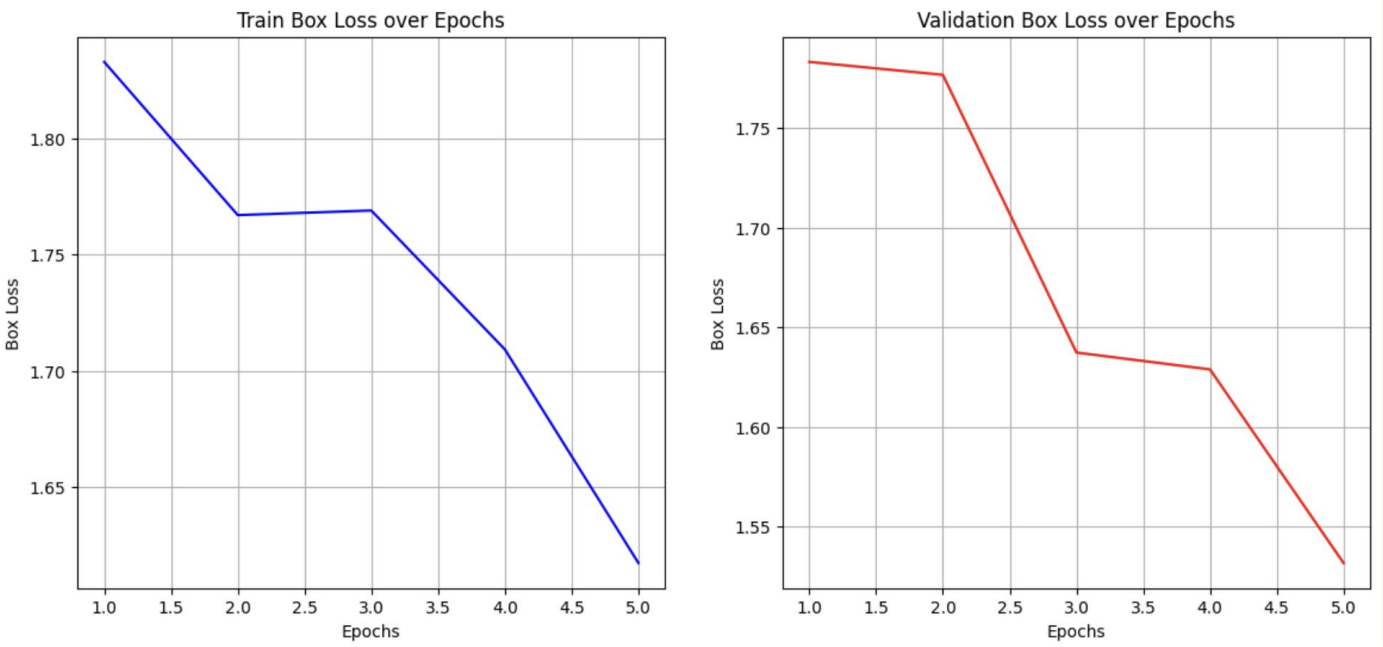
Precision, Recall, mAP50







Box Loss



epoch	time	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B)	metrics/mAP50-95(B	val/box_loss	val/cls_loss	val/dfl_loss	lr/pg0	lr/pg1	lr/pg2
1	1107	1.83301	3.37319	1.01915	0.67421	0.57126	0.5873	0.27475	1.78326	1.63546	1.00063	0.0701796	0.00331337	0.00331337
2	2213.69	1.76701	1.77409	1.00517	0.67511	0.61539	0.6269	0.3087	1.77676	1.39945	1.00264	0.0388636	0.00533066	0.00533066
3	3338.89	1.76899	1.68561	0.99039	0.71963	0.61857	0.66033	0.34126	1.63731	1.45749	0.95989	0.00622754	0.00602794	0.00602794
4	4440.97	1.70915	1.5292	0.9801	0.78267	0.68341	0.71563	0.38231	1.62885	1.20135	0.96516	0.00406	0.00406	0.00406
5	5550.81	1.61714	1.4258	0.96838	0.79044	0.684	0.71896	0.40549	1.53164	1.08924	0.95987	0.00208	0.00208	0.00208

Validation Batch Results

Validation Batch Results

Validation Batch Results

Original Image:

YOLOv8 Detection Result:

Model Results - Test Set

Model Results - Test Set

Model Results - Test Set

Model Results - Test Set


```
0 0.5606725146198830 0.2774528914879792 0.011513157894736841 0.01949317738791423
0 0.5718201754385965 0.2829759584145549 0.008771929824561403 0.014944769330734242
0 0.5740131578947368 0.3193632228719948 0.010782163742690058 0.03898635477582846
0 0.5363669590643275 0.5503573749187785 0.011513157894736841 0.022092267706302793
0 0.5736476608187134 0.5272904483430799 0.01663011695906433 0.021442495126705652
0 0.5977704678362573 0.5782975958414555 0.014071637426900584 0.02794022092267706
0 0.5778508771929824 0.6075373619233269 0.01827485380116959 0.0207927225471085l
```

Project Hurdles

Directory Mapping for YOLO

It was challenging to track image-annotation pairs, while formatting for YOLO training. We also had many issues with creating the config.yaml.

RAM Ceilings

Given Deepnote and Colab having 5GB and 12.7GB ceilings respectively, we ran out of RAM several times, especially when running on Deepnote.

Libgl1-mesa-glx

Ultralytics relies on OpenGL libraries (libGL.so.1), which are not Python libraries and must be explicitly installed. So, we ran into many installation issues.

Runtime

Since we are dealing with image data and are using large models, decent training amount (20 epochs) comes at the cost of large runtimes (up to 15 hours).

Future Improvements

Access to GPUs

With access to GPUs (such as A100) + more RAM with plans such as Colab Pro/Pro+, we could better optimize our model with faster runtimes and train more epochs.

Even just switching from CPU to the Tesla 4 GPU (free access via Colab), we saw training time drop by half.

Alternate Frameworks

We could also try using **ResNet** in order to extract detailed spatial features in the images, helping with detection. **DETR** is another potential model that looks promising too, but has high GPU requirements to run.

Real-World Implications

Real-World Readiness

Due to limited computational resources, our current model does not generalize too well and is **not ready** for real-world use. With more powerful GPUs and more RAM, we would increase training epochs; some additional hyperparameter fine-tuning may also improve our model performance enough for it to be used practically.

Implications

High-performing object detection models go beyond finding missing persons in low-visibility areas. These models may make broader societal impacts as they are applicable to other domains such as disaster management and public safety.

Our Team

Anshul Jambula

Computer Science & Data Science
‘2026

Amber Le

Data Science & Cognitive Science
‘2025

Arnav Mishra

Data Science (emp. Economics)
‘2026

Sarah Son

Data Science & Statistics
‘2025

Akhil Venkatesh

Data Science & Economics
‘2025