AI for European Wildlife Challenge

Generalizable Pipeline Team

Date: 30/06/2023

Final Report

Members:
Adrian Azoitei
Icxa Khandelwal
Arnold Kinabo
Christian Winkelmann
Aakash Gupta
Merve Gunak
Patryk Neubauer

Contents

Glossary

API - Application Programming Interface

HTTP - HyperText Transfer Protocol

CV - Computer Vision

FTP - File Transfer Protocol

IoU - Intersection over Union

mAP - Mean Average Precision

MViTv2 - Multiscale Vision Transformer version 2

SAHI - Slicing Aided Hyper Inference

YOLO - You Only Look Once

YOLO-NAS - YOLO-Neural Architecture Search

The Challenge

Introduction - Problem Statement and Goal

Introduction:

The challenge is to develop a machine learning pipeline that enables researchers to effectively classify European wildlife captured in camera trap images. The current approach relies on the MegaDetector, which has limitations in terms of the number of supported classes and accuracy in identifying certain animal types. The goal of the Pipeline Team is to build a generalizable pipeline that can be trained and retrained on various location-specific datasets, allowing for a larger number of classes and improved accuracy. This report aims to provide a reliable overview of the pipeline and its integration with the Trapper API.

The software architecture diagram in Figure 1 shows the operation and functionality of the pipeline.

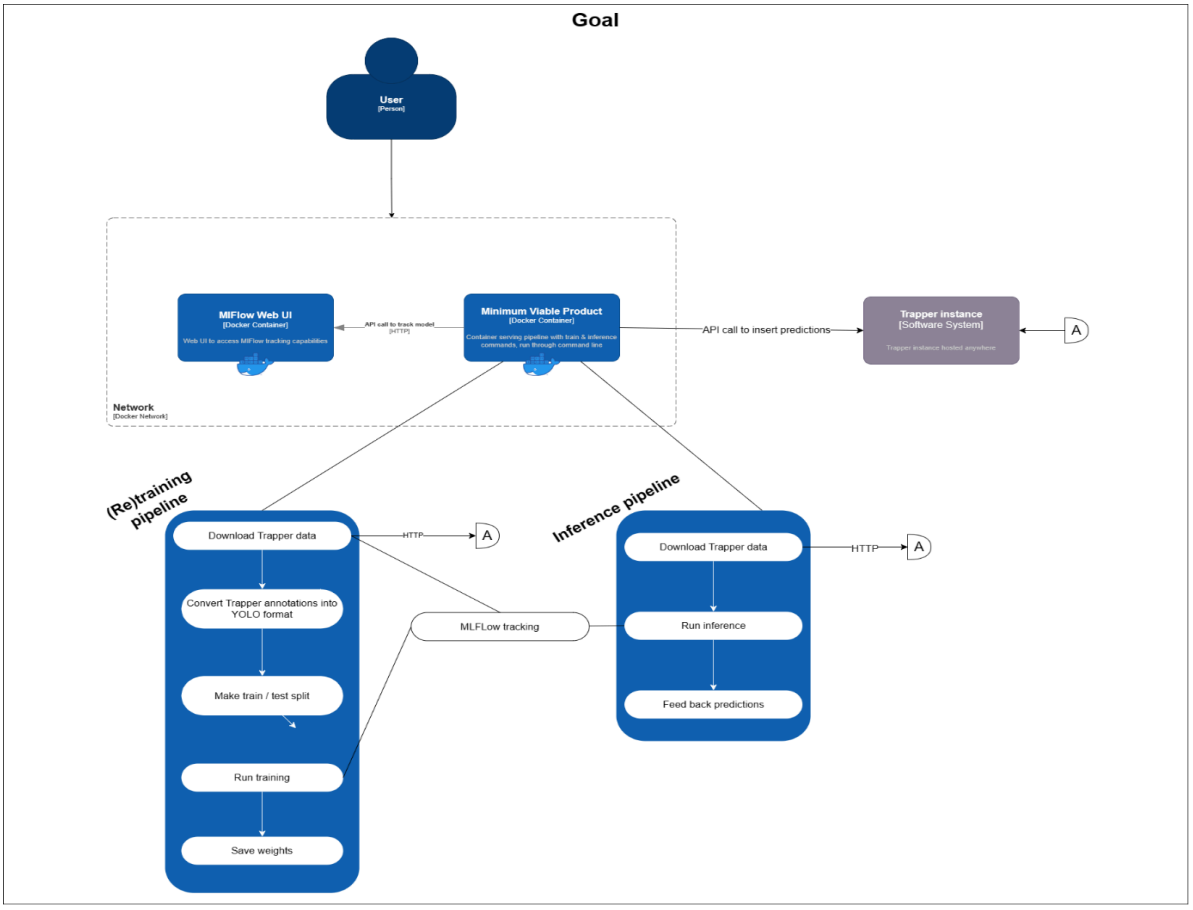

Figure 1: Current Generalisable Pipeline System Architecture

Pipeline Architecture:

The software architecture diagram (Figure 1) depicts the functionality and operation of the pipeline. The pipeline consists of several components, including data processing, model training, and inference. The user can conveniently upload data from multiple sources, after which it is then processed and made available for training. The pipeline supports choosing the model for training and enables inference on the trained models. Integration with the Trapper API allows the pipeline to interact with available datasets on the platform.

Limitations and Targeted Improvements:

The pipeline addresses certain limitations of the existing MegaDetector. Firstly, it expands the classification capabilities to support a larger number of classes, providing more detailed categorization of animals. Secondly, it aims to improve the accuracy in identifying reptiles and aquatic mammals, which were challenging for the MegaDetector. These enhancements are crucial for supporting comprehensive wildlife research activities.

Platform and Deployment Approach:

Initially, the plan was to develop a web service to serve the model and pipeline, integrating it with the Trapper API via HTTP. However, due to time constraints, the team opted to integrate the pipeline with Trapper using a Docker container. Future work includes the implementation of the web service. Additionally, a local instance of Trapper was deployed to improve performance during testing, ensuring reliable network connectivity and low response times.

This is the general procedure to follow to upload data to Trapper:

1) **Create a research project**
   Firstly, set up a research project in Trapper
2) **Add location (optional)**
   If the location information is already available (locationID, longitude, and latitude) in the deployment.csv file, this step may be skipped. Trapper automatically adds location information when the deployment.csv file is uploaded through its interface.
3) **Set up the Trapper-Client**
4) **Add deployments to Trapper**
   a. Automatically add deployments using camera trap data: Instead of manually adding deployments, which can lead to errors, the camera trap data itself may be utilised. A `.csv` file was created with the deployment

information, following specific guidelines. The data was organised in a main directory, with sub-directories for each collection, for example, rounds of deployments, and further sub-directories for each deployment. Consistently labelling the sub-directories with relevant information, such as a unique code (for example, R1 for round 1) and the location label (for example, 32_01), will prove helpful. An example of a deployment label is **R1-32_01**.

b. Access the “Package” tab in Trapper Client: Here, the team selected the main directory as the “Media root” and specified a folder to store the .csv file with the deployment table as the “Output path”. The team chose the collection for which they wanted to create the deployment table and specify the image file type (for example, `.jpg` for Reconyx cameras). Clicking “Get DT'' will generate the deployment table.

5) Add resources to Trapper

a. Create a data package: A data package was created in a similar manner as with the Deployment table. This step results in two files: a `.zip` file containing all the data and a `.yaml` file serving as a package definition file, containing metadata.

b. Upload the data package via FTP: Once the `.zip` and `.yaml` files are generated, they can be uploaded to the Trapper server using Trapper-Client.

c. Load the data into Trapper: On the Trapper server, the `.yaml` and `.zip` files are found in Storage > Upload collection. The relevant ones which were just uploaded are selected, and clicking “Upload” loads the data into Trapper.

Description of the Data

There were several subsets of data, each belonging to one of two main owners of data: ReWilding Europe and SLU. Among the datasets at our disposal, we made use of the ReWilding dataset to train our models. It is a comprehensive dataset that is a repository to over 378, 000 camera trap images.

For the validation stage, the team used the Lila dataset.

The dataset had to be processed twice: firstly, before being uploaded to Trapper; and secondly, in the training pipeline before the models could be trained on it (Figure 1).

ReWilding Dataset Pre-processing

Due to the nature of camera traps a lot of false positives are captured due to background movements (*like grass moving in the wind*). The dataset has been annotated with the species class by expert biologists, however the bounding boxes are not available on the provided data.

Steps for annotation of the ReWilding Dataset:

1. The ReWilding dataset was downloaded locally. *Script for the download function is provided here*
2. A local instance of the MegaDetector v5 was set up for annotation of the images
3. The images were annotated with bounding boxes and then merged with the expert species-level annotations
4. A threshold of 0.8 was selected for filtering bboxes with high confidence. This is heuristics and some [Reference]
5. The bounding box annotation was then converted into COCO, YOLO and TF format
    1. COCO - x1,y1, w, h
    2. YOLO - xc, yc, w, h
    3. TF - x1, y1, x2, y2
6. The dataset was also split into train, validation and test splits (80%-10%-10%)
7. The label map was also created for the species class

Total Size of the Dataset	**378,564**
Blank Images	247,675
Images with data*	130,889

Almost all images have multiple animals of the same species in the image

There are very few images with animals of different species

Pipeline Data Pre-processing

Data preprocessing is an important step in preparing the downloaded images from Trapper for training the YOLO family of models in the pipeline. The pre-processing stage consists of the following key steps:

1. **Downloading Image Files:** The pipeline retrieves a CSV file containing image URLs and corresponding file names using the Trapper Base query. These URLs are used to download the actual image files from Trapper's server, ensuring access to the necessary image data for further processing.
2. **Converting Bounding Boxes to YOLO Format:** Bounding box coordinates within the images are extracted. These bounding boxes define the location and extent of objects of interest in the images. To make them compatible with YOLO models, the pipeline converts the bounding boxes into the YOLO format. This format represents normalised coordinates, including centre coordinates and width/height, relative to the image dimensions. This conversion ensures standardised bounding boxes suitable for training YOLO models.

3. **Training and Weights Generation:** The pre-processed data, consisting of the images and YOLO-formatted bounding boxes, is used to train the YOLO models within the pipeline. During the training process, the model's parameters are optimised, enabling it to learn and accurately detect and classify objects in the images. The outcome of the training stage is the generation of a weights file. This file encapsulates the learned parameters and features essential for object detection.

4. **Saving Weights Files:** After training, the weights file is saved, allowing the user to select a specific weights file during the inference stage. Multiple weights files may be generated during different stages or iterations of the model's training. Saving these weights files provides flexibility for the user to choose the most appropriate model or compare the performance of different models during inference.

By performing these pre-processing steps, the pipeline ensures that the downloaded images are properly processed, the bounding boxes are converted to YOLO format, and the corresponding weights files are generated. This enables effective training and subsequent accurate detection and classification of objects in European wildlife camera trap images.

Augmentations

To address the limited occurrence of certain rare classes in the training data (with less than 10 occurrences), data augmentation techniques were considered essential for building effective models. Specifically, the team aimed to augment the data for species

of animals that had few occurrences. Research was conducted to identify suitable data augmentation methods for camera trapping.

Various approaches beyond standard computer vision methods were explored, including techniques based on animal pose, mix-up, simulating animals in empty images, and re-sampling rare classes. The team evaluated different libraries such as Torchvision, Albumentations, and Imgaug. Ultimately, **Albumentations** was chosen due to its extensive set of augmentations.

A script was developed to apply the selected augmentations to the data. However, implementing certain augmentations on camera trapping images posed challenges, particularly when segmentation and pose detection were required.

To facilitate the configuration of augmentations, a wrapper was implemented around Albumentations. It loads a `.yaml` configuration file that specifies the desired augmentations and their parameters. The wrapper allows for control over whether augmentations should be applied separately or all at once with a certain probability. By default, the pipeline creates copies of the data with different augmentations applied, but it can be easily modified to generate multiple copies with different combinations of augmentations.

The integration of Albumentations and the flexibility provided by the wrapper enable efficient data augmentation in the pipeline. This augmentation process enhances the training data by creating variations of the original images, leading to improved model performance in detecting and classifying rare classes in European wildlife camera trap images.

Modelling

The development of the project involved two main pipelines: a training and re-training pipeline, and an inference pipeline.

The team maintained the following models in their model zoo:

- YOLOv5
- YOLOv8
- YOLONAS
- Vision Transformer

Other models were researched, but weren't fully implemented:

- SAHI
- MViTv2

For object classification, the YOLOv5 model was trained using the subset of data that contains species information (10-20%). This was the equivalent of training for object detection, but with enriched classes instead of the generic “animal” label.

In the training pipeline, the data was divided into train, test, and validation sets using the sequence ID of the image files to ensure data integrity. The YOLO training data was generated accordingly. The size of the training data directly impacted the training time of the models. To manage the large dataset, it was possible to filter the data by selecting specific class names and limiting the number of images per class. The training process was performed on the FruitPunch servers, known for their high processing power.

Initially, the focus was on replicating the behaviour of Megadetector and evaluating the resulting model's performance on animal images. Therefore, the YOLOv5 model was trained on Mega Detector classes and a subset of the classes from Trapper. These models were included in the serving container, which constituted the entire service.

In the inference pipeline, the models generated predictions, and the results were saved in a CSV file named **"observations.csv**". These results were then published to Trapper within the Docker container using a POST REST API call. All HTTP calls in both pipelines were executed from scripts run via the command line within the container.

Post-processing was performed to log the evaluation metrics of the models. Two platforms, WeightsandBiases and MLFlow, were integrated for this purpose. Challenges were encountered while integrating MlFlow into the model training files; the source programs that our wrappers were written for, for the training were wrapped quite tightly, making it a challenge to directly integrate MlFlow into the trianing. This led to the decision to log the parameters passed into and out of the training function instead of logging within the function itself. The primary focus of logging was on the validation process, particularly in the inference and the validation scripts. Here the dataset, model, limits on training size, etc. were logged using MLFlow.

To execute MlFlow, it was run within its dedicated Docker container, as depicted in Figure 1 of the documentation.

Evaluation Metrics

In the evaluation of the pipeline models, the Mega Detector model serves as a reference for comparison. The Mega Detector model is specifically used for classification purposes, rather than detection, localization, or positioning.

The evaluation metrics used to compare the pipeline models are accuracy, precision, and recall. These metrics assess the classification capabilities of the pipeline models and provide insights into their performance when compared to the Megadetector model.

By using these common evaluation metrics (accuracy, precision, and recall), the pipeline models' classification capabilities are assessed and compared to the Megadetector model. This allows for a fair evaluation and comparison of their performance in classifying objects, without considering their localization or positioning.

Results

The Pipeline Team were able to deliver on their goal of producing a generalizable machine learning pipeline, and with it they were able to perform training and inferencing on different data fed into it. The team's goal was not to investigate training methodologies and compiling optimum datasets to produce intelligent models, as per the Model Exploration Team's directives. But this section shall go through some instances that demonstrate how the pipeline was used to train models, and how some of the resultant models compare.

Model Training:

Two models - YOLOv5X6 and YOLOv8x - were trained with trapper images from different projects, especially project 13. The images were filtered based on species to produce an adequate training dataset as Figure 2 illustrates.

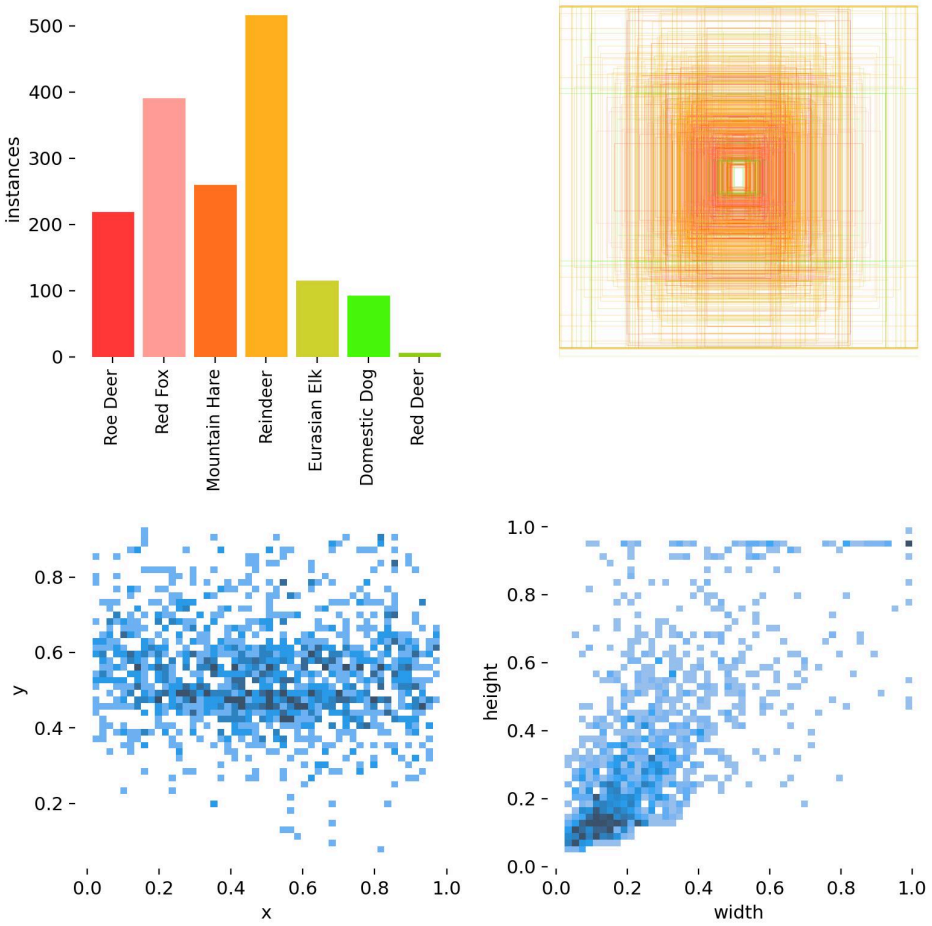

Figure 2: Training Set of Images for YOLOv5X6

Training results for YOLOv5X6, at Step = 100:

- mAP@0.5 = 0.8694 (Figure 3)

That is, it had a mean average precision of 0.8694, calculated at an IoU threshold of 0.5. By our metrics, this signifies a well performing model.

- Precision = 0.8697 (Figure 4)
- Recall = 0.820 (Figure 5)

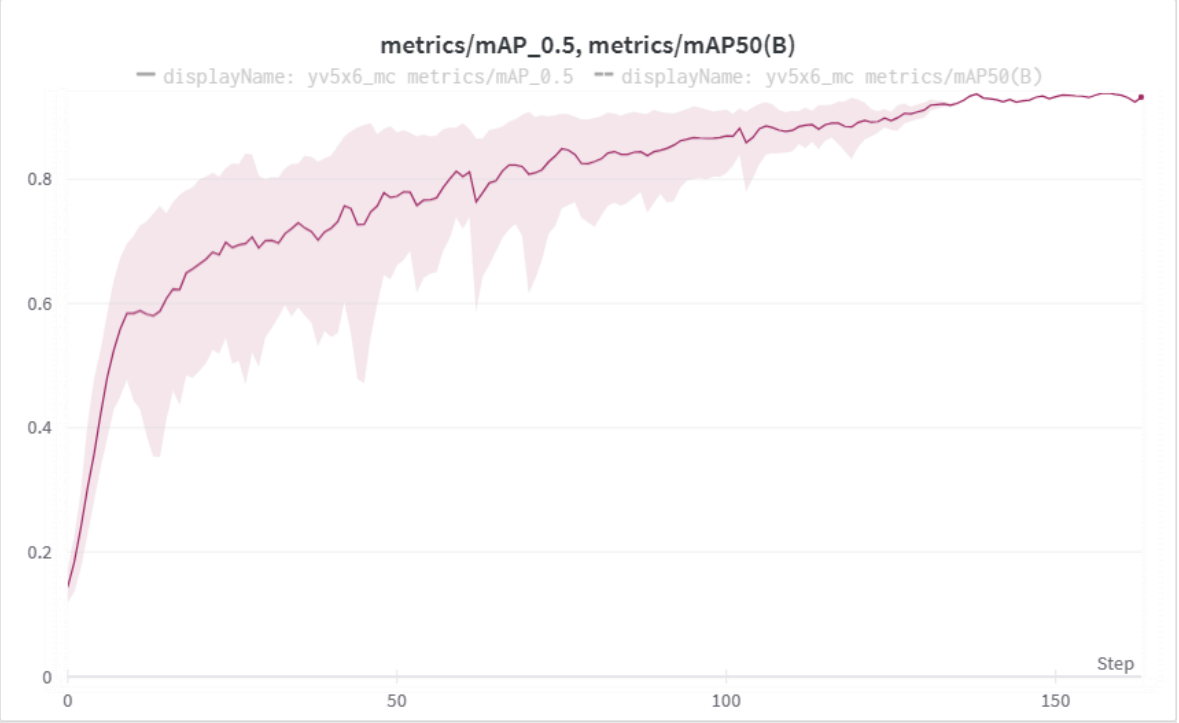

Figure 3: YOLOv5X6 mAP@0.5

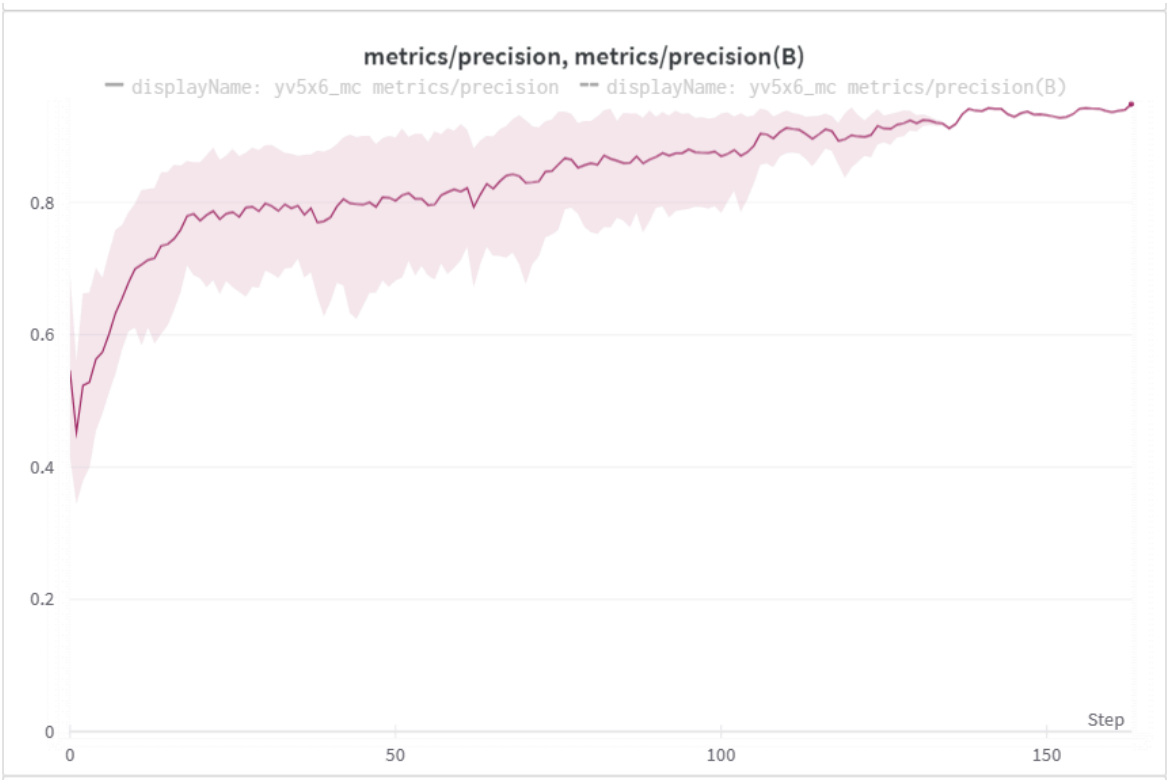

Figure 4: YOLOv5X6 Precision

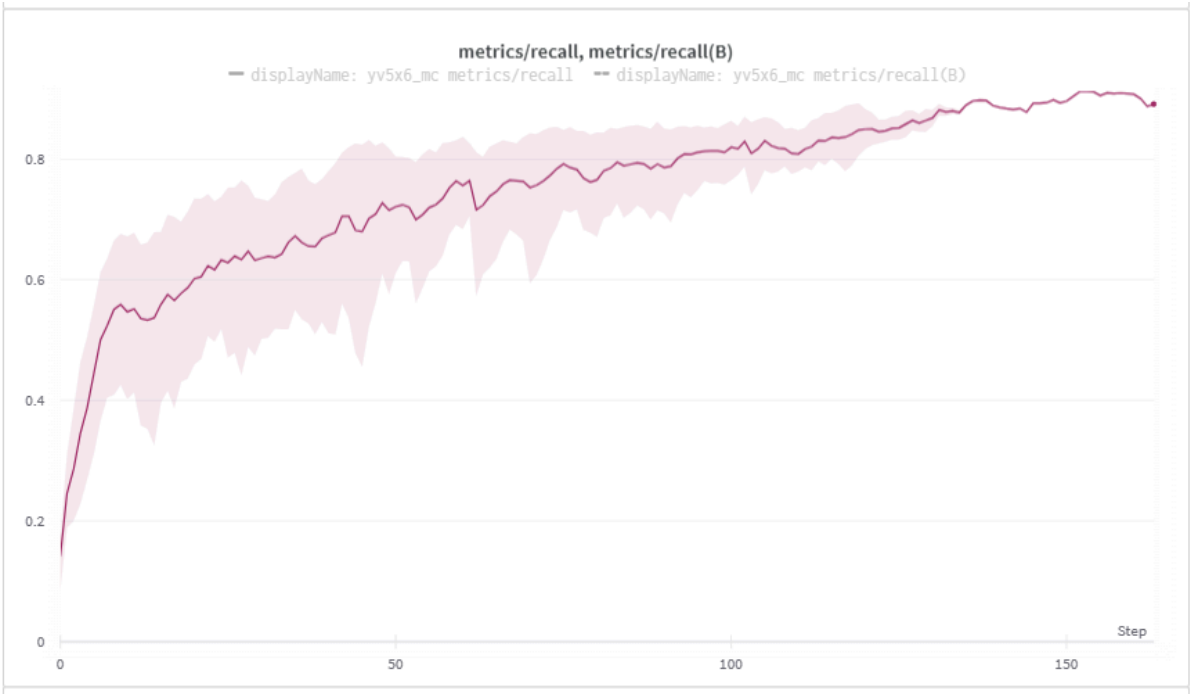

Figure 5: YOLOv5X6 Recall

Model Comparison:

The biggest models are not always the most accurate or best performing.

The team compared across three different YOLO models; from smallest to largest model respectively, they are: YOLONAS, YOLOv5n6, and YOLOv8n6. Figure 6 shows some experiments with the YOLONAS S running 640px, YOLOv5n6 running 1280px, and YOLOv8n6 running 640px (better results were obtained on the YOLOv5 using the 1280px size images as opposed to 640px).

The results show that the recall of the smaller YOLONAS was better than that of the other two models, though its precision was unusable.
At Step = 99 (Figures 6 and 7):

- YOLONAS; recall = 0.9142, precision = 0.1464
- YOLOv5n6; recall = 0.902, precision = 0.9704
- YOLOv8n6; recall = 0.6556, precision = 0.9043

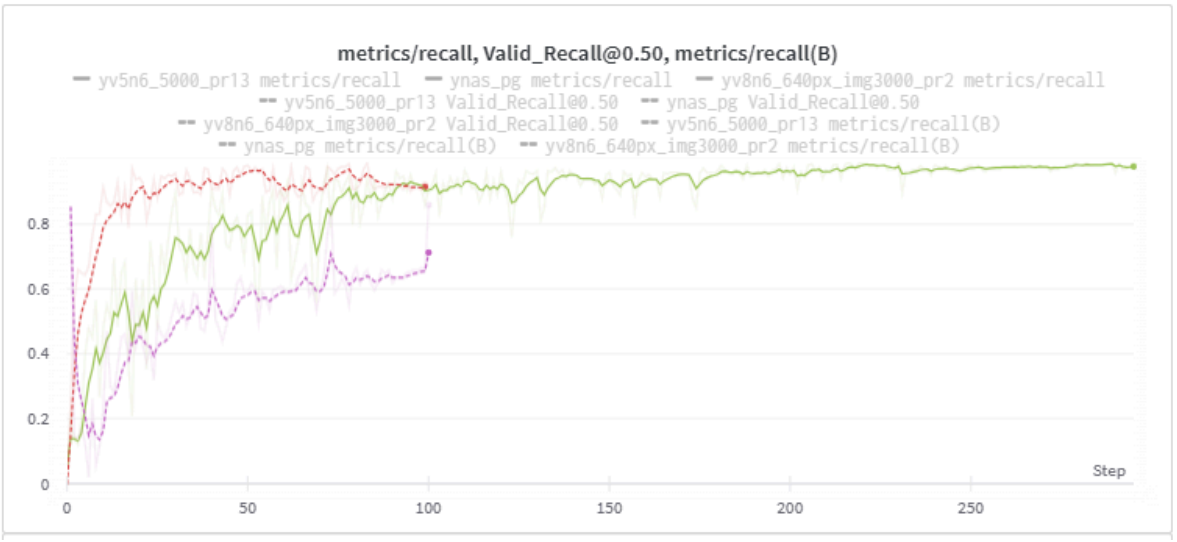

Figure 6: Recall@0.5 for YOLONAS, YOLOv5 and YOLOv8

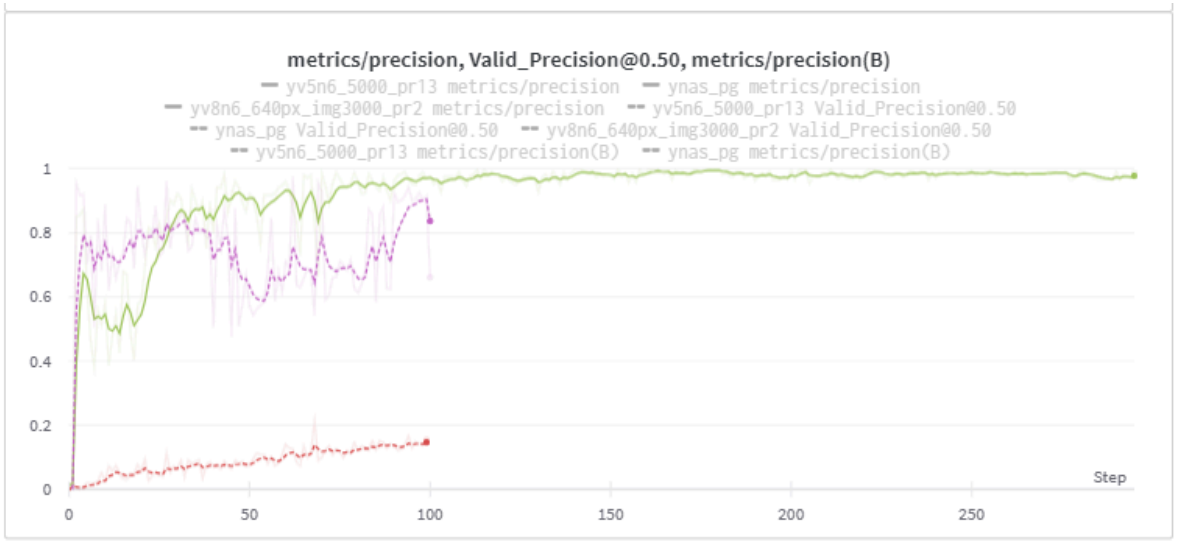

Figure 7: Precision@0.5 for YOLONAS, YOLOv5 and YOLOv8

Conclusion

It has been mentioned in the Augmentations section how it was challenging to implement some of the augmentations on some camera trapping images. In the Modelling section, hurdles faced on integrating MlFlow were also detailed.

Apart from these, the training process faced some other significant challenges during the project. One major issue was the presence of a bug, possibly in the Trapper codebase or the downloader, which caused incorrect placement of bounding boxes when downloading more than 250 images. This issue made it difficult to train the models effectively.

Additionally, there were problems with downloading from Trapper when using the existing observations table from a local Trapper instance. This resulted in HTTP 400 - Bad Request errors, indicating either an empty response or insufficient permissions for the requested object.

The team persisted through these hurdles and found a way to realise the proposed solution. Any additions and modifications to the project underwent thorough bug and workflow tests before being committed to a central codebase accessible on this Gitlab where it is maintained.

Future Work

We envision an operation as that captured in Figure 8 for a future challenge. We'd originally set out to realise this system architecture, as explained in the Platform and Deployment Approach section in the Introduction. This singular major contribution would see the addition of a model serving web service.

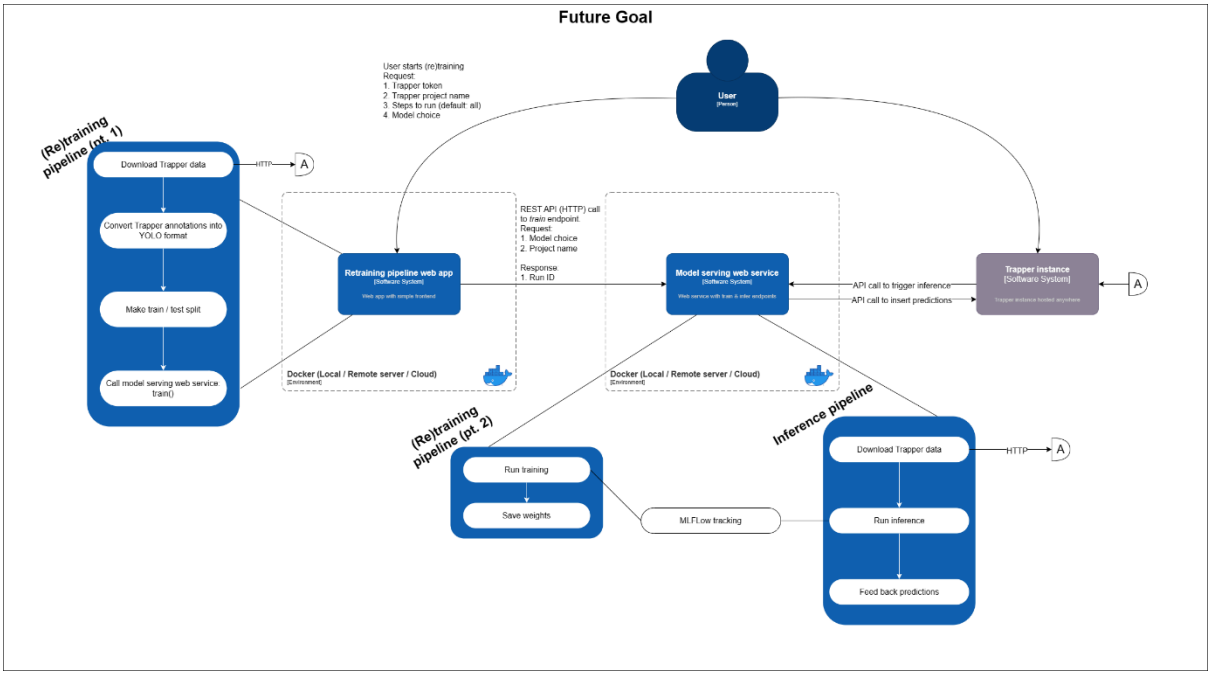

Figure 8: Future Generalisable Pipeline System Architecture

There are a few other contributions that would be beneficial to the user. The integration with Trapper to automatically trigger the training and inference pipelines to work, for one. Another is to define some pre-defined use cases that a user can select to quickly get a pipeline instance up and running, for example:
*"fetch **X** number of images classified as animal **Y** from project **Z**, then run object detection and upload back into Trapper"*,
where X, Y and Z are values specified by the user.