Monthly Report

Onboard Classification of Tree Species using RGB UAV Imagery

Mahendra Priolkar, Shantanu Prabhudessai, Mansel Martins

Onboard VS Offboard Processing

Onboard

1) Faster, done in one step on UAV itself.
2) Lower flight times, due to increased payload and power requirements.
3) Lesser accuracy due to limited compute capability.

Offboard

1) Slower, need to survey land, then transfer files to the system, then run inference.
2) Higher flight times, due to lower payload, and lower power requirements.
3) High accuracy as the compute capability isn’t limited (relatively), can run much larger models.

Why YOLO

YOLO is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy. Its streamlined design makes it suitable for various applications and easily adaptable to different hardware platforms, such as edge computing devices, like the raspberry pi.

It uses a single shot process to both detect objects and draw bounding boxes around them, and classify them, unlike existing methods that use two-shot algorithms, one to detect objects, and one to classify them, such as R-CNN.

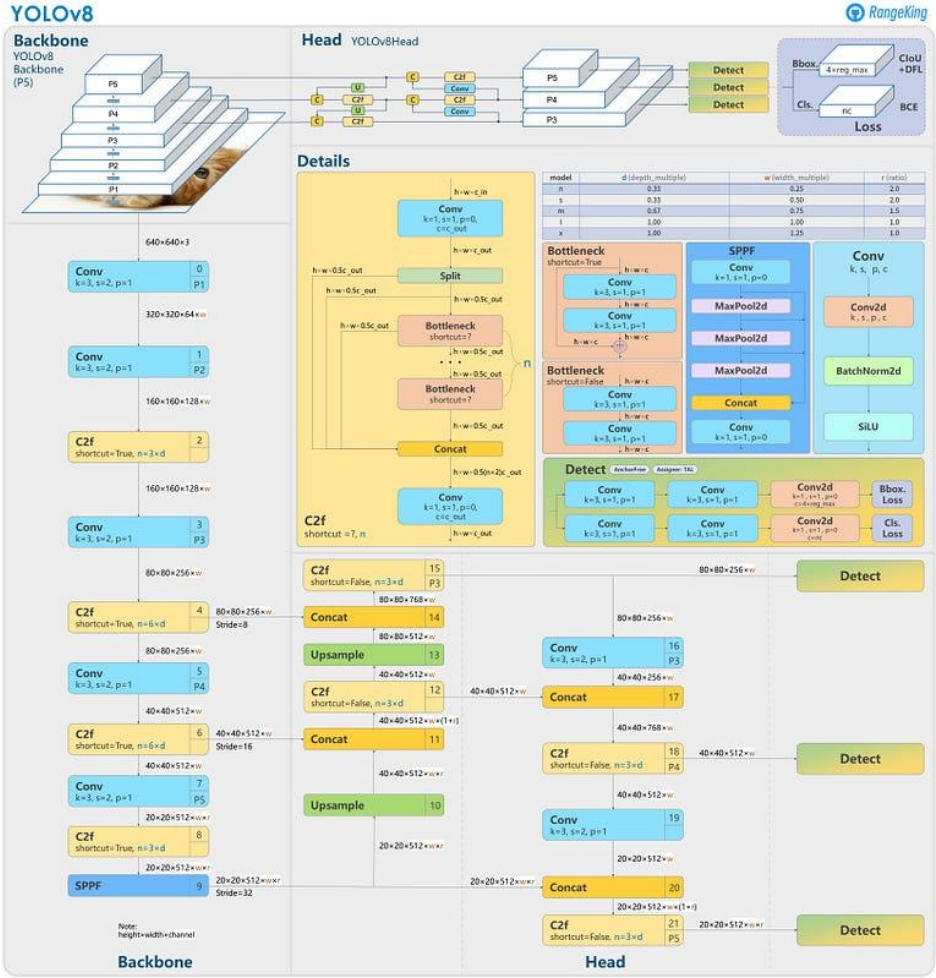
YOLO : How it Works

YOLO is a type of object detection algorithm that predict the bounding box and the class of the object in one single shot. This means that in a single forward pass of the network, the presence of an object and the bounding box are predicted simultaneously. This makes YOLO very fast and efficient, suitable for tasks that require real-time detection, but have lower accuracy.

YOLOv8 Architecture

The YOLOv8 architecture can be broadly divided into three main components:

- **Backbone:** This is the convolutional neural network (CNN) responsible for extracting features from the input image. YOLOv8 uses a custom CSPDarknet53 backbone, which employs cross-stage partial connections to improve information flow between layers and boost accuracy.
- **Neck:** The neck, also known as the feature extractor, merges feature maps from different stages of the backbone to capture information at various scales. YOLOv8 utilizes a novel C2f module instead of the traditional Feature Pyramid Network (FPN). This module combines high-level semantic features with low-level spatial information, leading to improved detection accuracy, especially for small objects.
- **Head:** The head is responsible for making predictions. YOLOv8 employs multiple detection modules that predict bounding boxes, objectness scores, and class probabilities for each grid cell in the feature map. These predictions are then aggregated to obtain the final detections.



https://www.researchgate.net/figure/The-model-architecture-of-YOLO-v818_fig3_378824507
[Accessed: 21-09-2024]

YOLOv10 Architecture

The architecture of YOLOv10 builds upon the strengths of previous YOLO models while introducing several key innovations. The model architecture consists of the following components:

1. **Backbone**: Responsible for feature extraction, the backbone in YOLOv10 uses an enhanced version of CSPNet (Cross Stage Partial Network) to improve gradient flow and reduce computational redundancy.
2. **Neck**: The neck is designed to aggregate features from different scales and passes them to the head. It includes PAN (Path Aggregation Network) layers for effective multiscale feature fusion.
3. **One-to-Many Head**: Generates multiple predictions per object during training to provide rich supervisory signals and improve learning accuracy.
4. **One-to-One Head**: Generates a single best prediction per object during inference to eliminate the need for NMS, thereby reducing latency and improving efficiency.

YOLOv10 Architecture

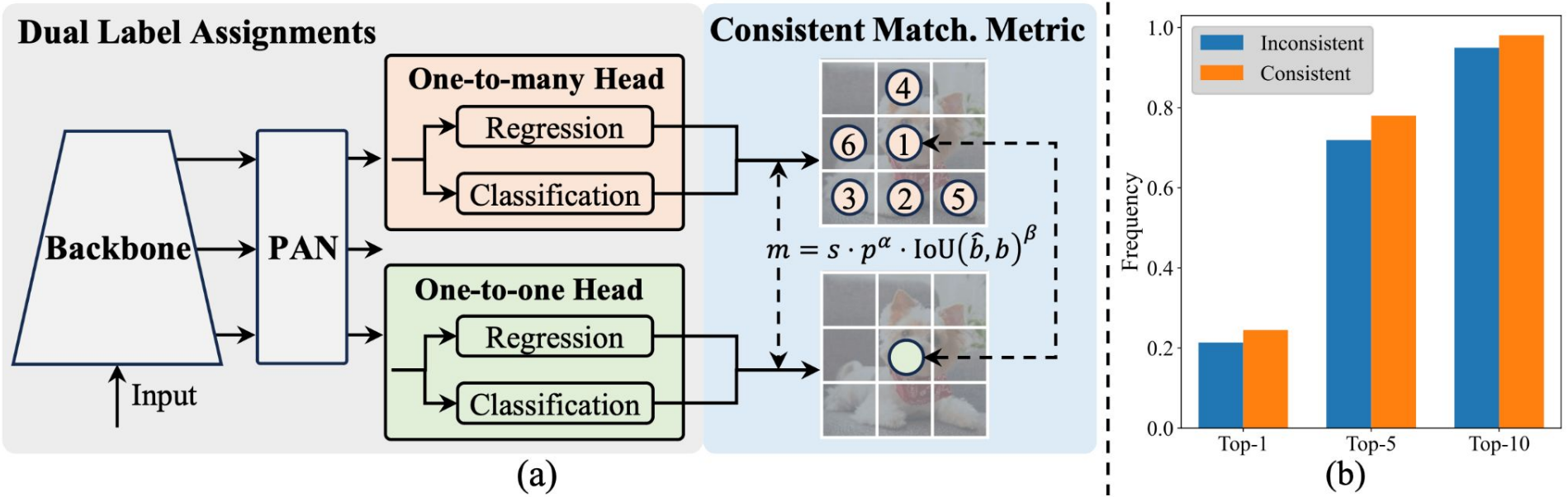


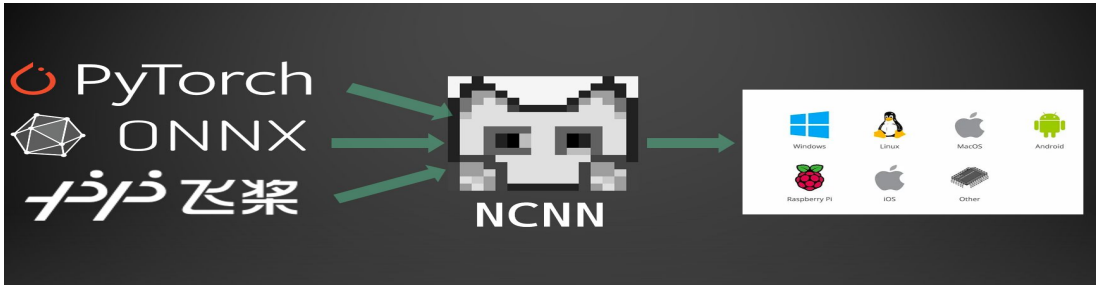
Figure 2: (a) Consistent dual assignments for NMS-free training. (b) Frequency of one-to-one assignments in Top-1/5/10 of one-to-many results for YOLOv8-S which employs α_{o2m}=0.5 and β_{o2m}=6 by default [20]. For consistency, α_{o2o}=0.5; β_{o2o}=6. For inconsistency, α_{o2o}=0.5; β_{o2o} =2.

https://www.researchgate.net/figure/a-Consistent-dual-assignments-for-NMS-free-training-b-Frequency-of-one-to-one_fig1_380821008 [accessed 21 Sept 2024]

NCNN Format

The NCNN framework, developed by Tencent, is a high-performance neural network inference computing framework optimized specifically for mobile platforms, including mobile phones, embedded devices, and IoT devices. NCNN is compatible with a wide range of platforms, including Linux, Android, iOS, and macOS.

NCNN is known for its fast processing speed on mobile CPUs and enables rapid deployment of deep learning models to mobile platforms. This makes it easier to build smart apps, with improved inference times at the cost of slight accuracy.

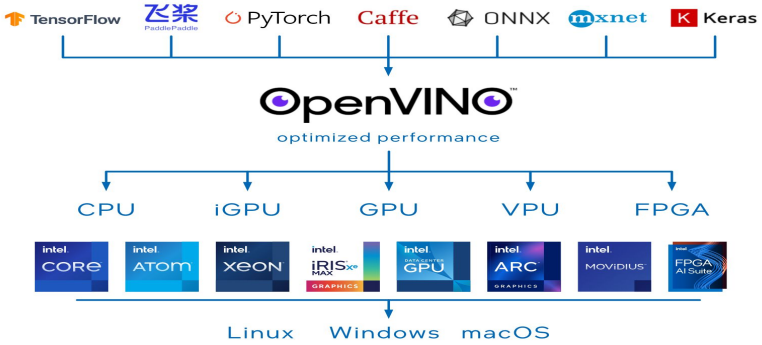


https://docs.ultralytics.com/integrations/ncnn/#why-should-you-export-to-ncnn [Accessed: 21-09-2024]

OpenVINO format

OpenVINO is an open-source toolkit for optimizing and deploying deep learning models. It provides boosted deep learning performance for vision, audio, and language models from popular frameworks like TensorFlow, PyTorch, and more.

OpenVINO boosts a model's speed even further with quantization and other state-of-the-art compression techniques available in OpenVINO's Post-Training Optimization Tool and Neural Network Compression Framework. These techniques also reduce the model size and memory requirements, allowing it to be deployed on resource-constrained edge hardware.



https://docs.ultralytics.com/integrations/openvino/ [Accessed: 21-09-2024]

Model format comparisons

Performance of YOLOv8n on RPi4, as reported by Ultralytics.

Format	Status	Size on disk (MB)	mAP50-95(B)	Inference time (ms/im)
PyTorch	✅	6.2	0.6381	1068.42
TorchScript	✅	12.4	0.6092	1248.01
ONNX	✅	12.2	0.6092	560.04
OpenVINO	✅	12.3	0.6092	534.93
TF SavedModel	✅	30.6	0.6092	816.50
TF GraphDef	✅	12.3	0.6092	1007.57
TF Lite	✅	12.3	0.6092	950.29
PaddlePaddle	✅	24.4	0.6092	1507.75
NCNN	✅	12.2	0.6092	414.73

https://docs.ultralytics.com/guides/raspberry-pi/#detailed-comparison-table [Accessed: 21-09-2024]

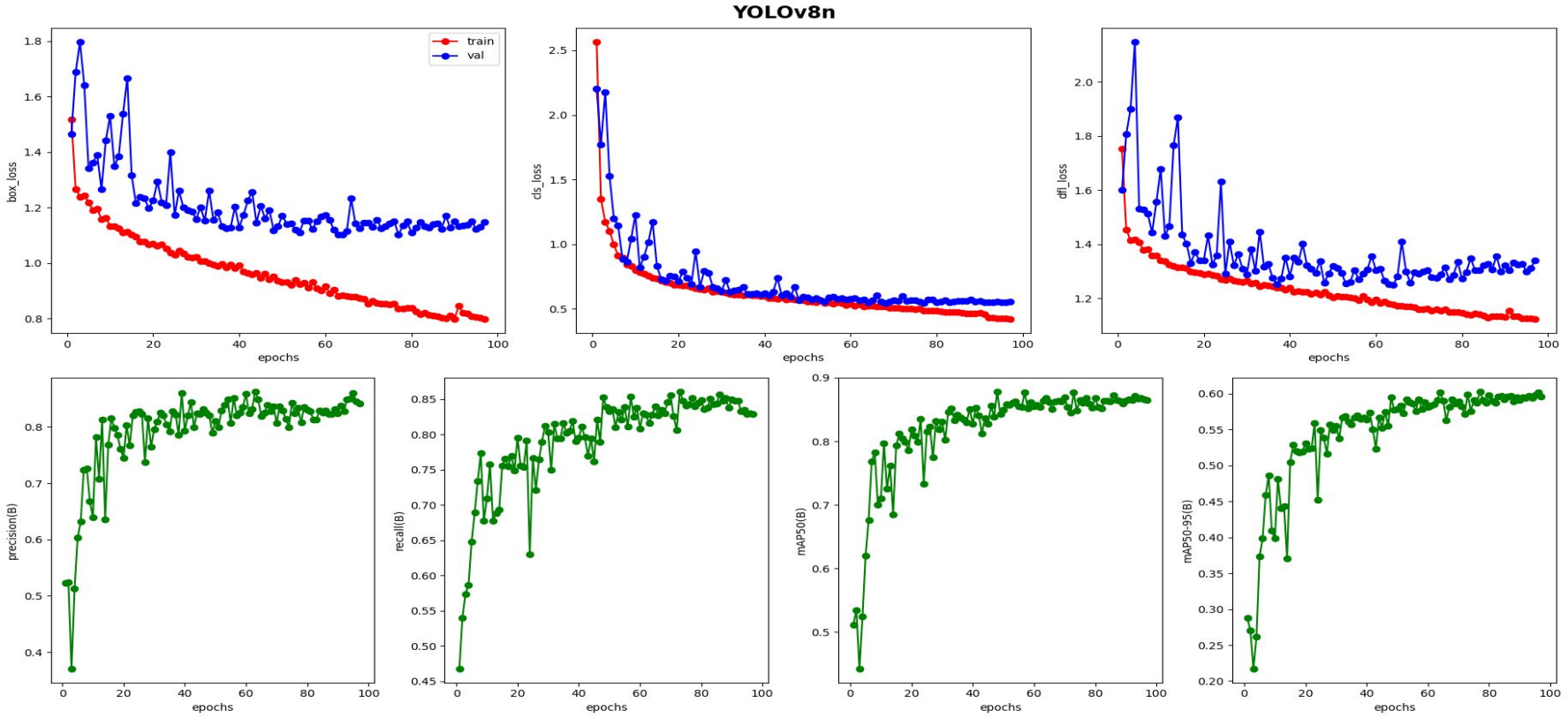
Finetuning Methodology

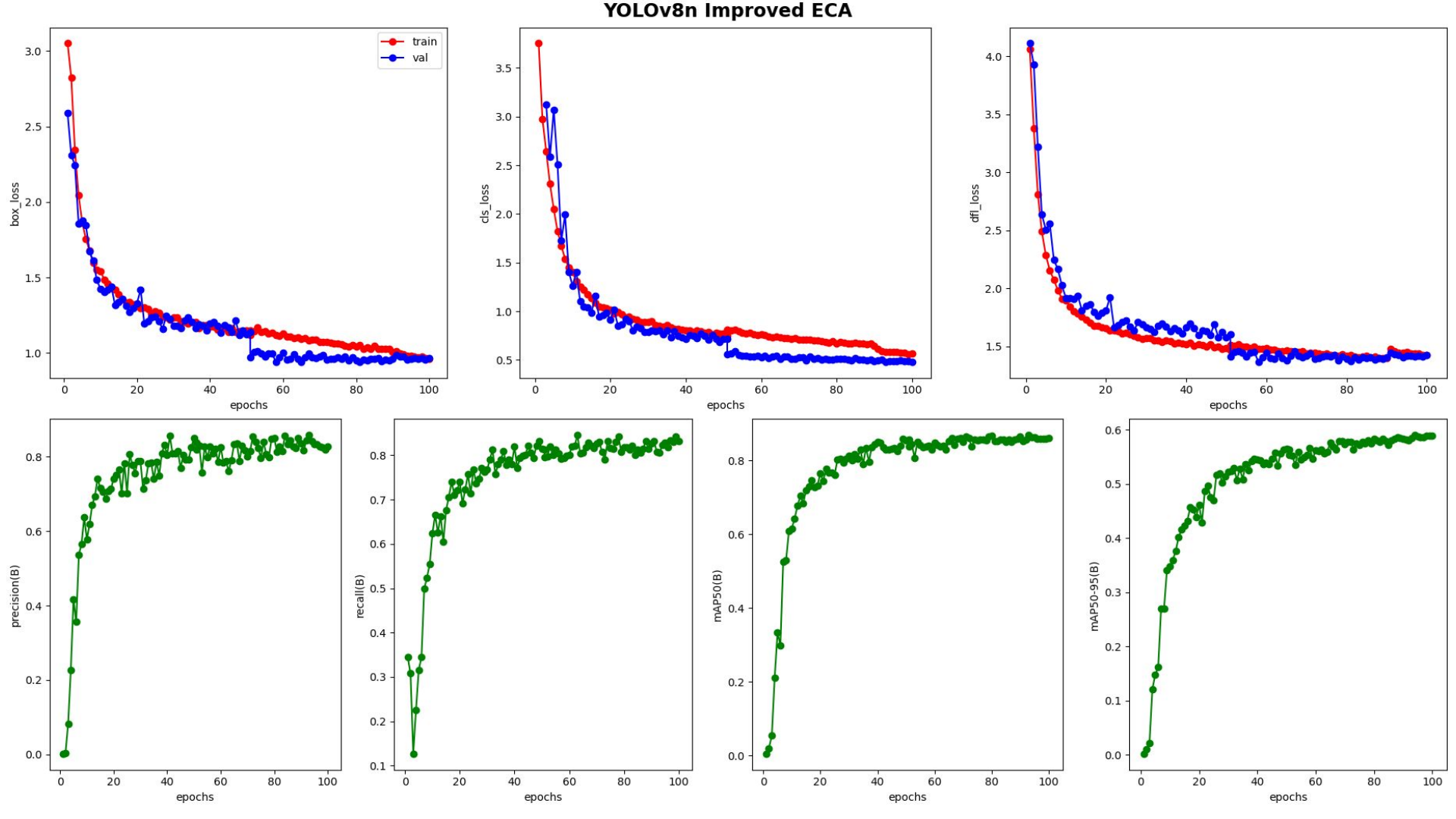
- We used the free plan on Google Colab for finetuning, which provides a Nvidia Tesla T4 GPU, for anywhere between 1 to 3.5 hrs for a single session.
- We used the augmented dataset, and fine tuned each model for 100 epochs, but with a patience value of 10.

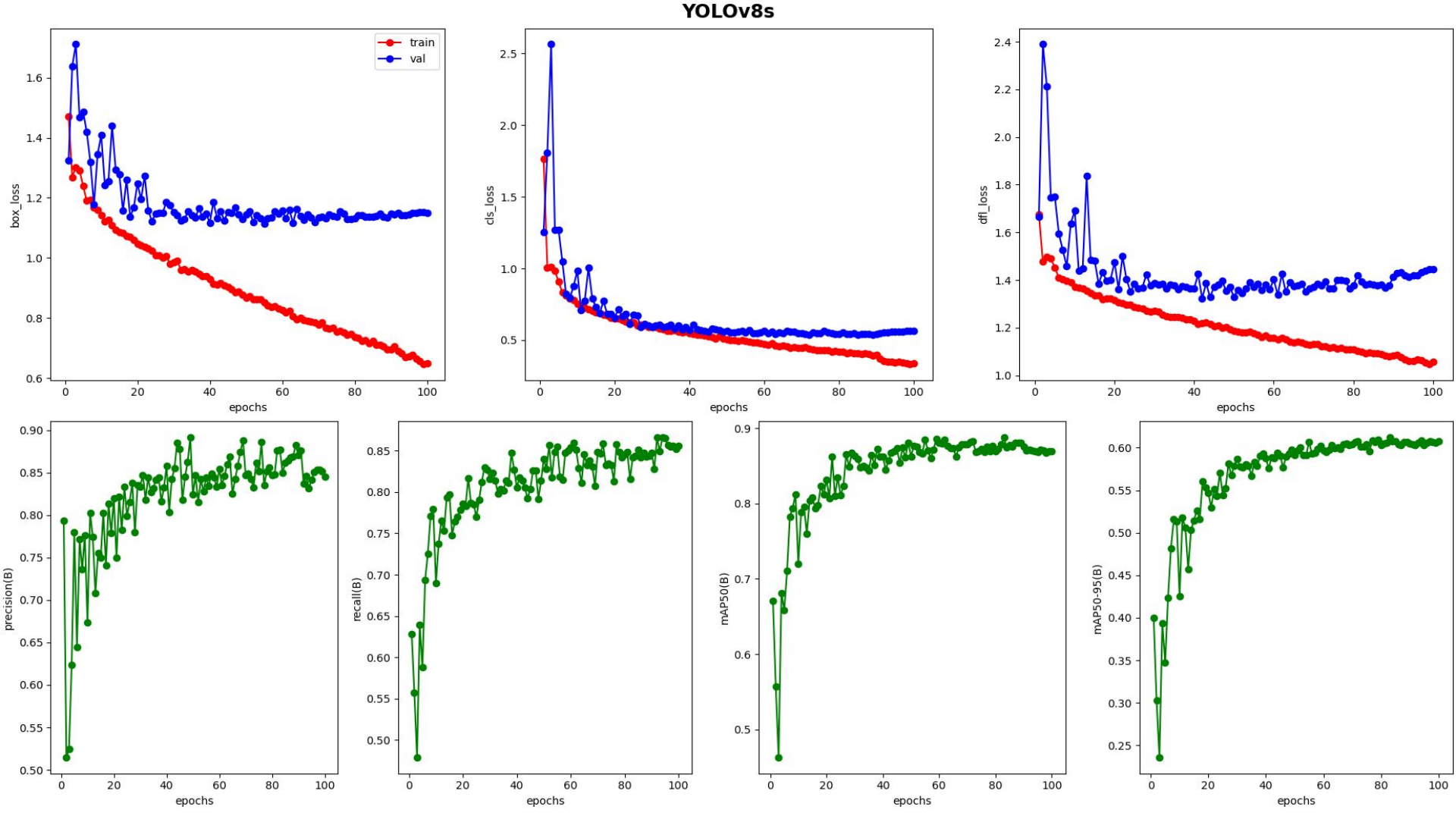
To finetune the models, we ran the following command:

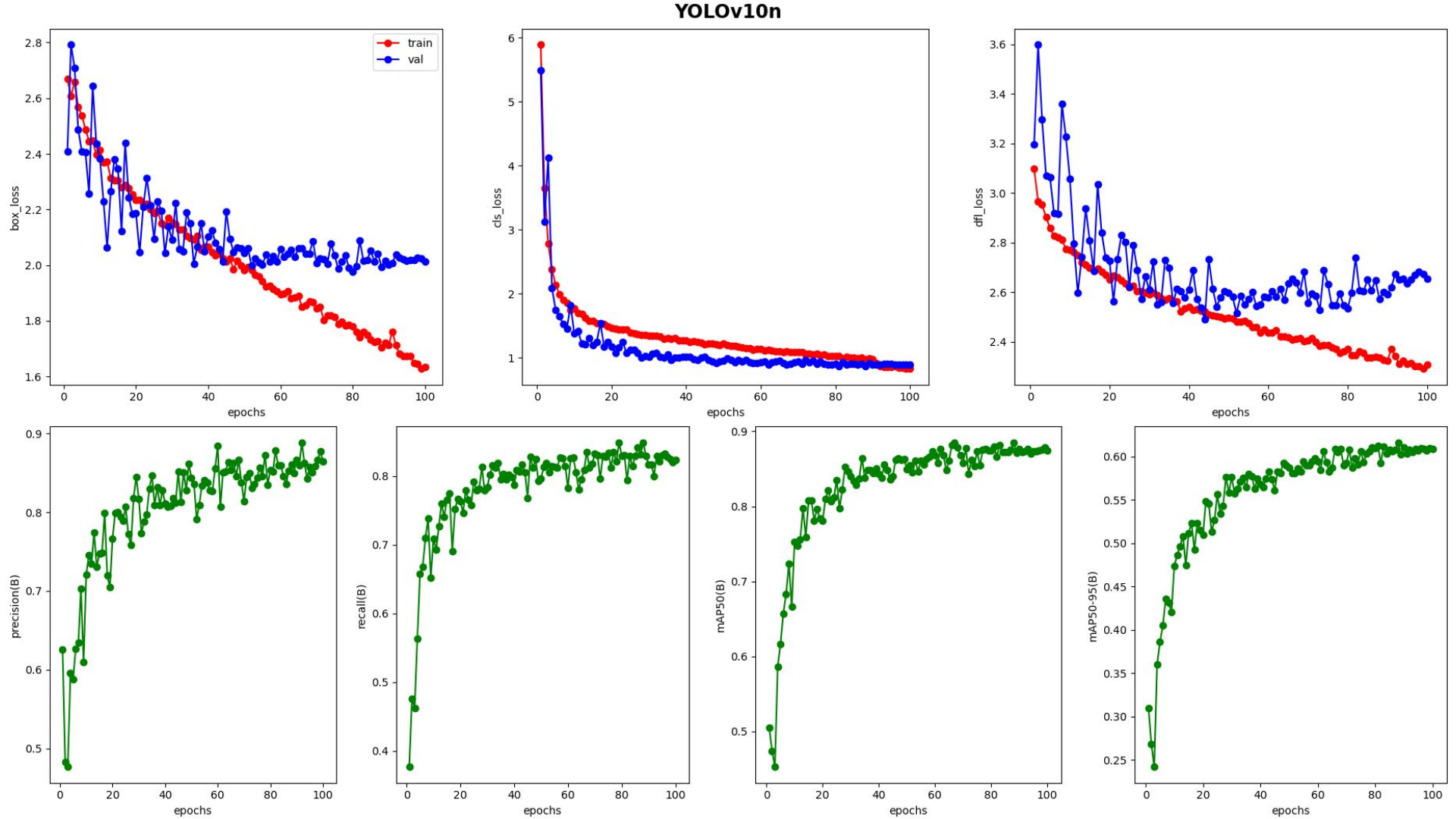
```
yolo detect train data=<path to dataset yaml file> model=<pretrained model name>
epochs=100 imgsz=640 batch=0.85 save_period=5 patience=10 plot=True
```

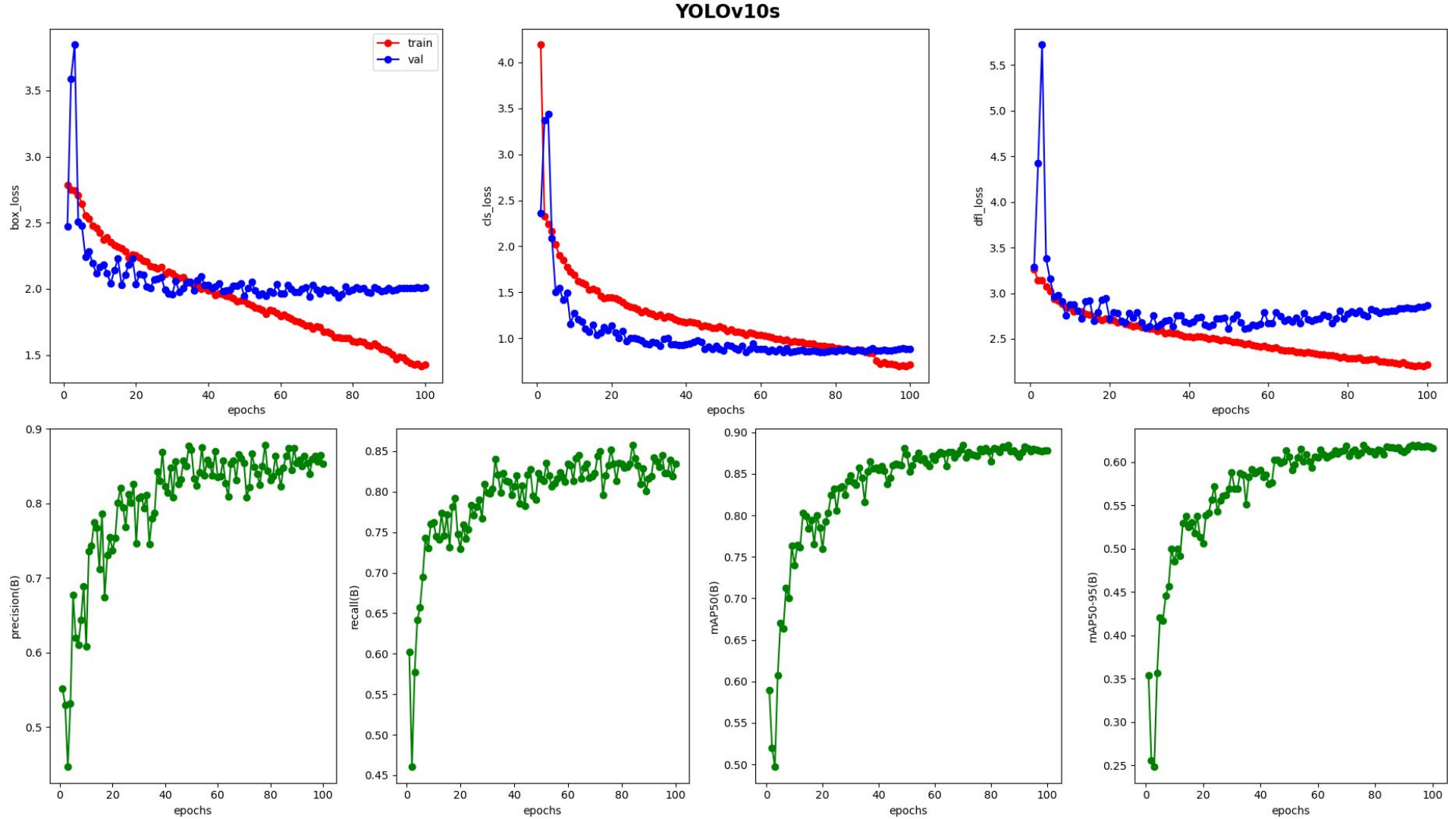
Finetuning Results: Per Epoch comparisons




YOLOv8n Improved ECA
train
val
box_loss
cls_loss
dfl_loss
precision(B)
recall(B)
mAP50(B)
mAP50-95(B)
epochs


YOLOv8s
train
val
box_loss
cls_loss
dfl_loss
precision(B)
recall(B)
mAP50(B)
mAP50-95(B)
epochs


YOLOv10n
train
val
box_loss
cls_loss
dfl_loss
precision(B)
recall(B)
mAP50(B)
mAP50-95(B)
epochs


YOLOv10s
train
val
box_loss
cls_loss
dfl_loss
precision(B)
recall(B)
mAP50(B)
mAP50-95(B)
epochs

Benchmarking Methodology

The benchmarks were run on various YOLO models on

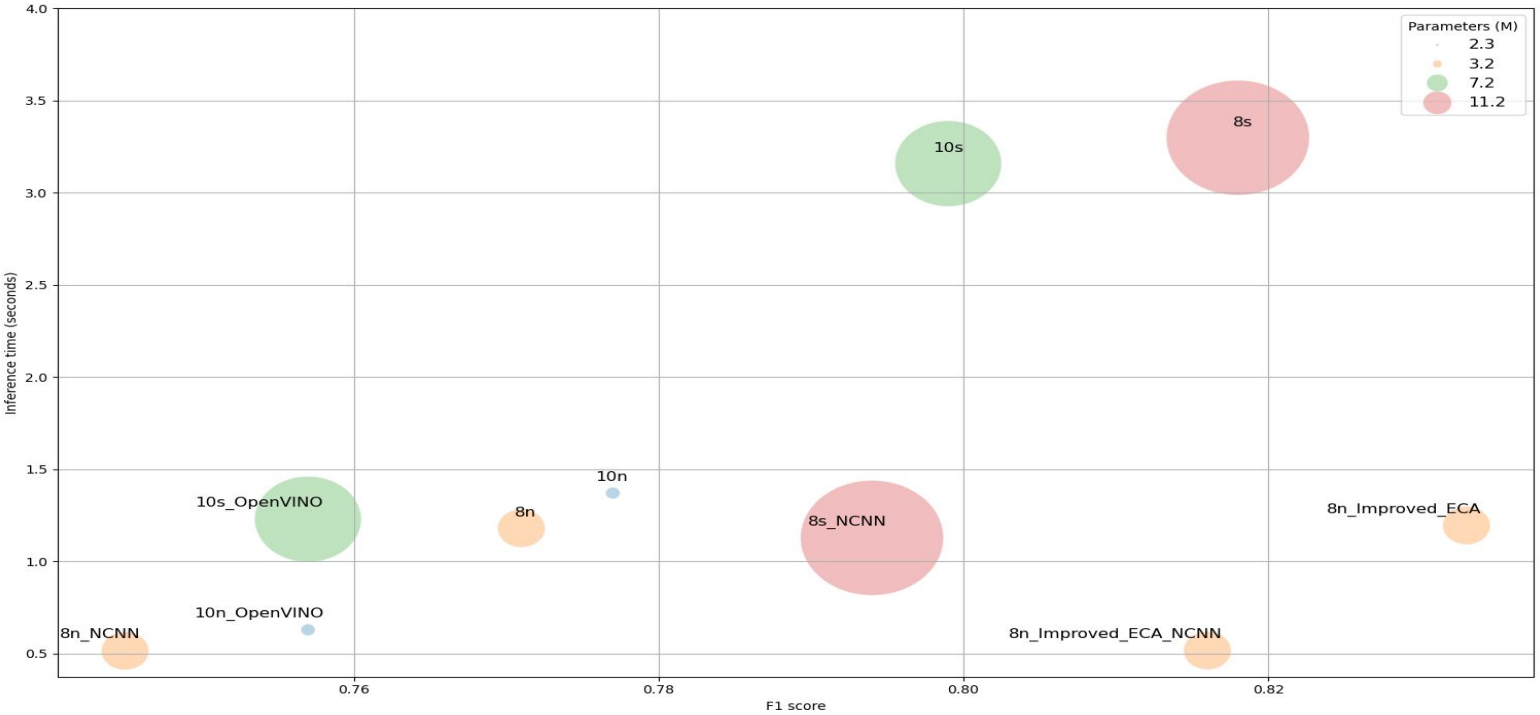
1. Raspberry Pi 4B 4GB SBC running Raspberry Pi OS 12, overclocked to 2GHz.
2. A Windows 10 PC, with a Ryzen 5 5500 CPU and a Geforce RTX 3060 12GB GPU, 16GB dual channel 3200MHZ DDR4 RAM.

We ran 5 different video clips, with 120 frames each, through each tested YOLO model one frame at a time, and the total inference time was averaged over the total frame count (120 * 5 = 600 frames).

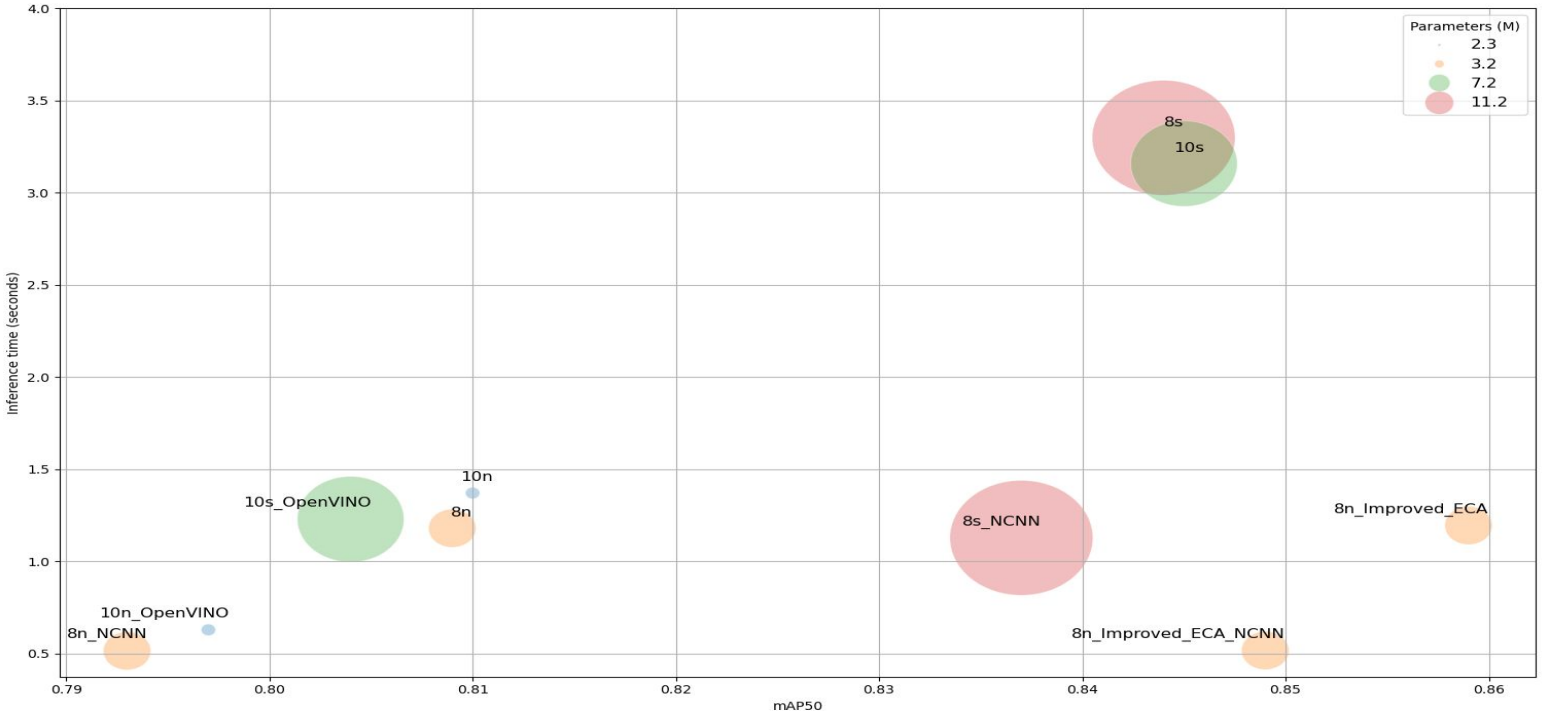
Benchmark Results: Summarised

Model	mAP 50	mAP50-95	Inference Time (s)		F1 Score
			Raspberry Pi	PC	
Yolov8s	0.844	**0.582**	3.297	0.031	0.818
Yolov8n - Improved ECA	**0.859**	0.575	1.194	**0.027**	**0.833**
Yolov10s	0.845	0.574	3.157	0.033	0.799
Yolov8n - Improved ECA NCNN	0.849	0.564	0.516	0.069	0.816
Yolov8s NCNN	0.837	0.560	1.127	0.150	0.794
Yolov10s OpenVINO	0.804	0.545	1.228	0.072	0.757
Yolov10n	0.810	0.544	1.370	0.031	0.777
Yolov8n	0.809	0.541	1.179	**0.027**	0.771
Yolov10n OpenVINO	0.797	0.523	0.628	0.041	0.757
Yolov8n NCNN	0.793	0.512	**0.514**	0.071	0.745

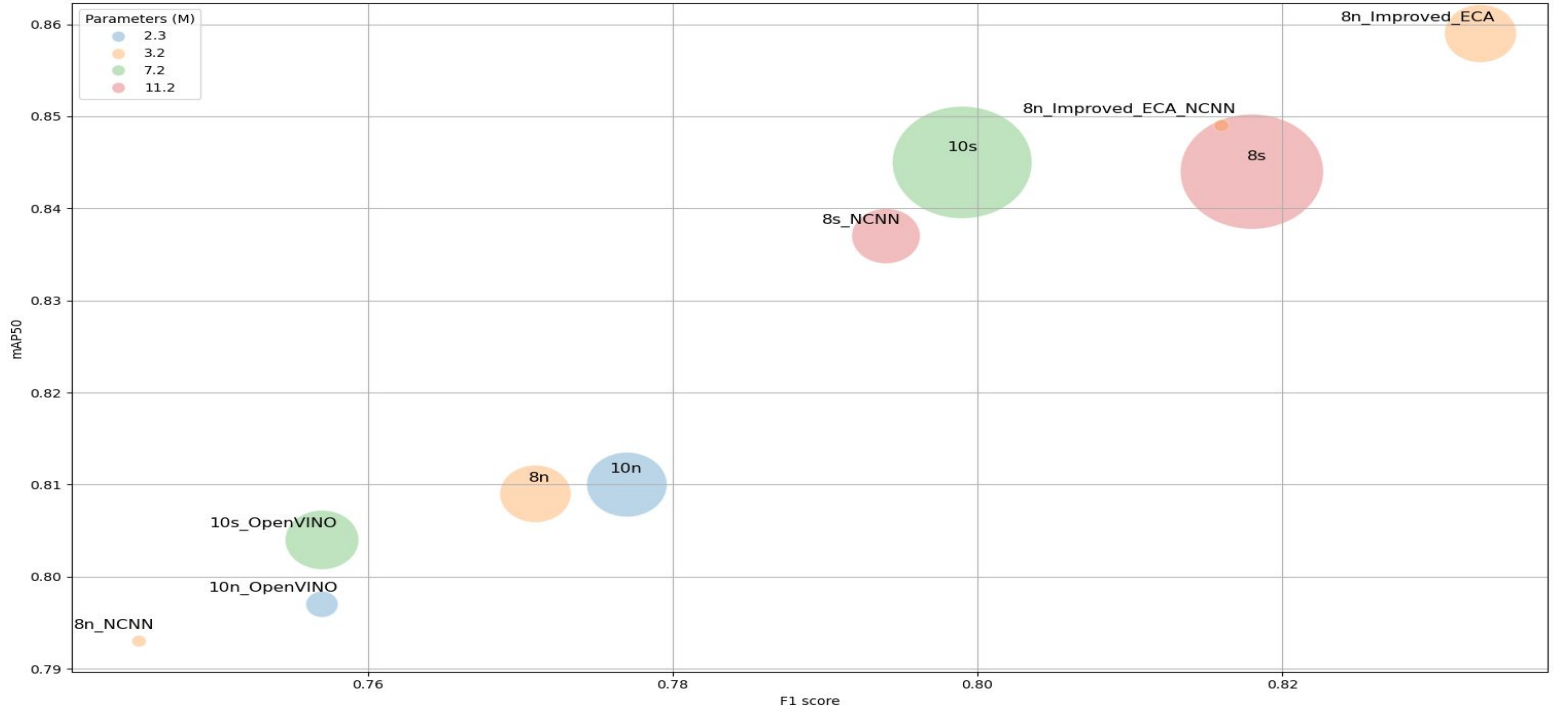
Benchmark Results: F1 vs Inf time vs Param count



Benchmark Results: mAP50 vs Inf time vs Param count

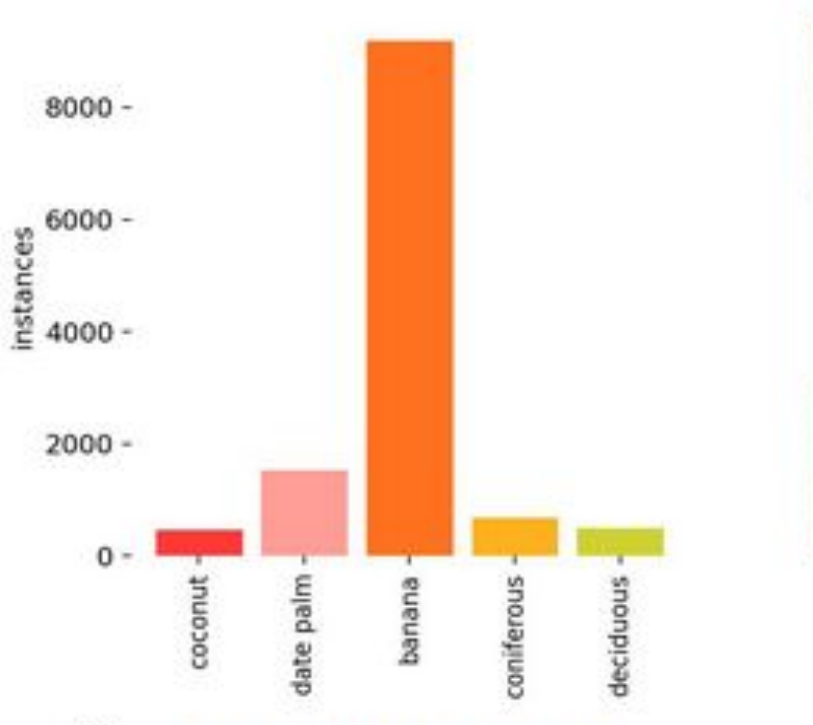


Benchmark Results: F1 vs mAP50 vs Inf time vs Param count



Pain Points

1) The uneven distribution of classes in the dataset
2) The small size of the dataset (>10,000 instances per class recommended by Ultralytics)
3) Consider running inference on the ground, rather than the UAV, to increase flight time and compute capabilities, increasing accuracy by enabling the use of a larger model, or another architecture altogether (Faster RCNN).



Future Course of Action

1) Improve the dataset by :

   i) Finding more data to merge into the dataset

   ii) Adding synthetic data using algorithms such as SMOTE

2) Try using a larger, slower model, but with higher accuracy, such as faster RCNN.

Papers Researched

https://www.mdpi.com/2072-4292/15/5/1463 : Individual Tree-Crown Detection and Species Identification in Heterogeneous Forests Using Aerial RGB Imagery and Deep Learning
https://link.springer.com/article/10.1007/s10586-022-03627-x : A survey on deep learning-based identification of plant and crop diseases from UAV-based aerial images
https://www.sciencedirect.com/science/article/pii/S1574954123003345 : Adoption of Unmanned Aerial Vehicle (UAV) imagery in agricultural management: A systematic literature review
https://www.sciencedirect.com/science/article/pii/S0168169922008663 : On-farm evaluation of UAV-based aerial imagery for season-long weed monitoring under contrasting management and pedoclimatic conditions in wheat
https://www.tandfonline.com/doi/pdf/10.1080/15481603.2023.2177448 : Comparison of high-resolution NAIP and unmanned aerial vehicle (UAV) imagery for natural vegetation communities classification using machine learning approaches
https://www.mdpi.com/2072-4292/13/23/4851 : ECAP-YOLO: Efficient Channel Attention Pyramid YOLO for Small Object Detection in Aerial Image
http://ecmlpkdd2017.ijs.si/papers/paperID11.pdf : FCNN: Fourier Convolutional Neural Networks
https://arxiv.org/abs/1709.01507 : SENet Paper
https://arxiv.org/abs/1910.03151 : ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks
https://arxiv.org/abs/1807.06521 : CBAM Convolutional Block Attention Module
https://arxiv.org/abs/2304.00501v1 : A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond
https://arxiv.org/abs/2402.13616 : YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information
https://arxiv.org/abs/2405.14458 : YOLOv10: Real-Time End-to-End Object Detection