Introduction

Our ocean habitats and fisheries are under tremendous stress, which worsens every year. As fisheries collapse, the incidence of illegal fishing continues to increase, only adding to the pressure on these ecosystems. Criminals disable their transponders, leaving the only methods for detection of these activities as costly patrols or satellite imagery.

There have been multiple challenges issued in the past few years for better algorithms to flag illegal fishing activity, using satellite imagery. Winners have invariably trained varieties of convolutional neural networks to the task.

The core challenges to training these models are differentiating between ships and other objects/artifacts, and between fishing vessels and other types of ships. Beyond training, the time to inference is also a critical factor, given the massive scale of global satellite imagery.

To those ends I have investigated potential enhancements to the YOLO v5 pipeline through noise reduction through gaussian filtering, and feature enhancement using Otsu thresholding and a Sobel threshold.

I chose those modifications for both the use case, and out of consideration of the purported origins of some improvements between YOLO v3 and YOLO v5. Time and resource constraints were such that my results are inconclusive to date. However combining noise suppression with feature enhancement appears to be superior to batch image augmentations beyond mosaics, for this use case.

Background

Object classification approaches in general share the need to balance speed and accuracy; accordingly the standard metrics are Mean Average Precision (mAP) at a given number of Frames Per Second (FPS).

mAP is assessed by comparing known object locations and classifications against the location and class determined by the algorithm. In the simplest terms: If I know for certain that there is a cruise ship at precise coordinates, I can separately evaluate whether or not you know it is a cruise ship, and how accurately you assess its location. The “Average” in mAP is calculated individually per type of object (class), and combined into a total score.

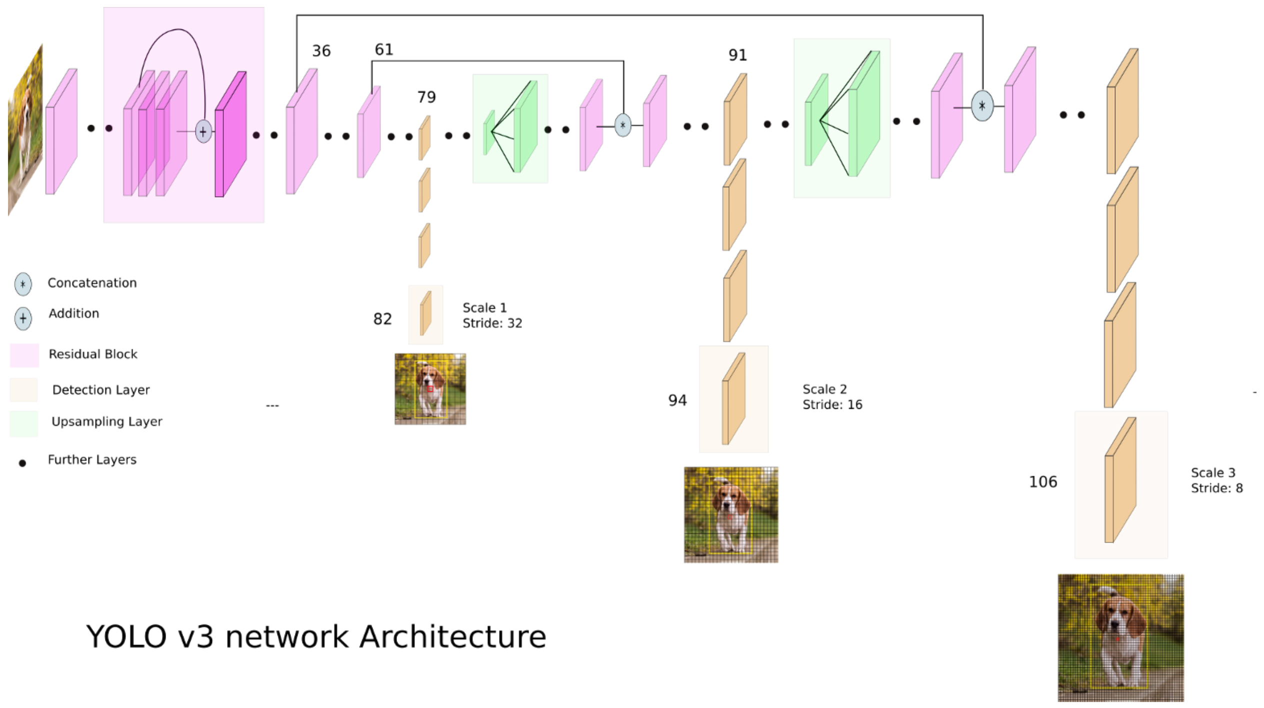
$$\text{mAP@}\alpha = \frac{1}{n}\sum_{i=1}^{n} \text{AP}_i \quad \text{for n classes.}$$

FPS is determined by how quickly an object is classified, and stated generally as “Model/Network X reached mAP of Y% at Z FPS” (Elgendy 2020).

The most accurate algorithms such as Region Based CNNs (RCNN) employ multistage processes whereby regions of interest are essentially flagged as potentially containing an object by a search oriented network, followed by a classification process where potential features of each potential object are assigned probabilities for being an object or not. This 2-n staged process yields greater accuracy, at the cost of speed. (Huang et al 2017)

For the last few years there have been two leading competitors for striking a balance favoring speed over accuracy: You Only Look Once (YOLO) and Single Shot Detection (SSD). Both approaches combine the 2-n approaches of more accurate models into a single pass over an image. Both RCNN, and SSD employ approaches for handling the problem of fine-grained vs broad details, and integrating those into one object. How YOLO handles that particular issue significantly informed the experiments discussed in this paper, so it would be helpful to provide an overview of its peculiarities.

In YOLO the lower layers of the network learn what features look like at a larger scale, which becomes more fine-grained in higher layers. The properties of the image at each scale are abstracted so that the general features of what constitutes an “ear” or an “arm” can be learned separately from individual examples of either. At wider scales “ears” and “arms” can be combined into “a person”. To ensure that the overall picture isn’t lost as “arms” and “ears” are identified, so that the total likelihood that a person is in part of an image, residual or “skip” layers ensure that less abstracted data is available to later layers, along with the mathematical representation of “an arm”.



Source: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

Each of the layers considers both more and less abstracted information, and progressively at different scales to arrive at probabilities for whether or not an object is present, and to which class it might belong. The number of predictions grows throughout this process to the point where for a 416 x 416 pixel image YOLO produces 10,647 bounding boxes around possible objects! (Elgendy p. 324 2020). These predictions are then effectively grouped and the most centered box, calculated to have the highest probability of containing an object is retained while the rest are discarded.

YOLO v4 improved on v3 substantially in terms of time to inference, as well as fine scale object detection, however the changes to the network structure itself were primarily geared towards increasing the receptor fields, for finer grained discrimination. [2004.10934v1.pdf (arxiv.org)](2004.10934v1.pdf) Additionally data augmentations were introduced, most notably by incorporating mosaics of differently scaled and/or rotated images into the training pipeline. Each batch provides a new set of mosaics.

YOLO v5 retained those changes, but swapped out the input layer for what the authors call a "focus layer" wherein the first three layers of v3 are exchanged for one per the primary architect: "The main purpose of the Focus layer is to reduce layers, reduce parameters, reduce FLOPS, reduce CUDA memory, increase forward and backward speed while minimally impacting mAP." (Jocher, 05-2021)

YOLO v5 also provides built in methods to modify batches beyond mosaics. Scaling, hue modifications, translations, shear, and more are available. There are no apparent modifiers for feature enhancement specifically.

Data

The 2018 Airbus Ship Detection Challenge provided 104k satellite training images, 88k test images, 131k thousand of which contain ships, which are labeled (with masks) accordingly. The raw images are 768 x 768 full color.

The provided labels were Run Length Encoded: where a flattened matrix of pixels is assumed, and each pixel of the object is

The original contest was specifically aimed at object segmentation. To those ends the training labels were provided in 2D run length encoded format where effectively the object is traced in pixels to a 2D array.

While segmentation is a more difficult computational task than generating object classes with bounding boxes, we are here comparing methods for performing the latter. One of the first tasks then is to convert the given segmentation masks into boxes.

Additionally the training data is mostly comprised of images with no boats, or other classified objects. To accelerate training those empty images need to be dereferenced, thereby removed from the training set.

I borrowed the code in the rle_to_bbox.py file in the linked repository to generate a dictionary which accomplishes both mask to bounding box label transformation, as well as dereferencing. Attributions are provided in rle_to_bbox.py.

From that conversion of masks to bounding boxes, YOLO encoding requires annotations to be provided by:

<object-class> <x> <y> <width> <height>

Where class is given numerically, x, and y, represent the center of a bounding box, and width and height represent the distance from center.

The adhoc strings are parsed into objects, and then to the required YOLO format below.

I created an annotations subset of the data to enhance logging, sampling, and to facilitate coordination of images and labels.

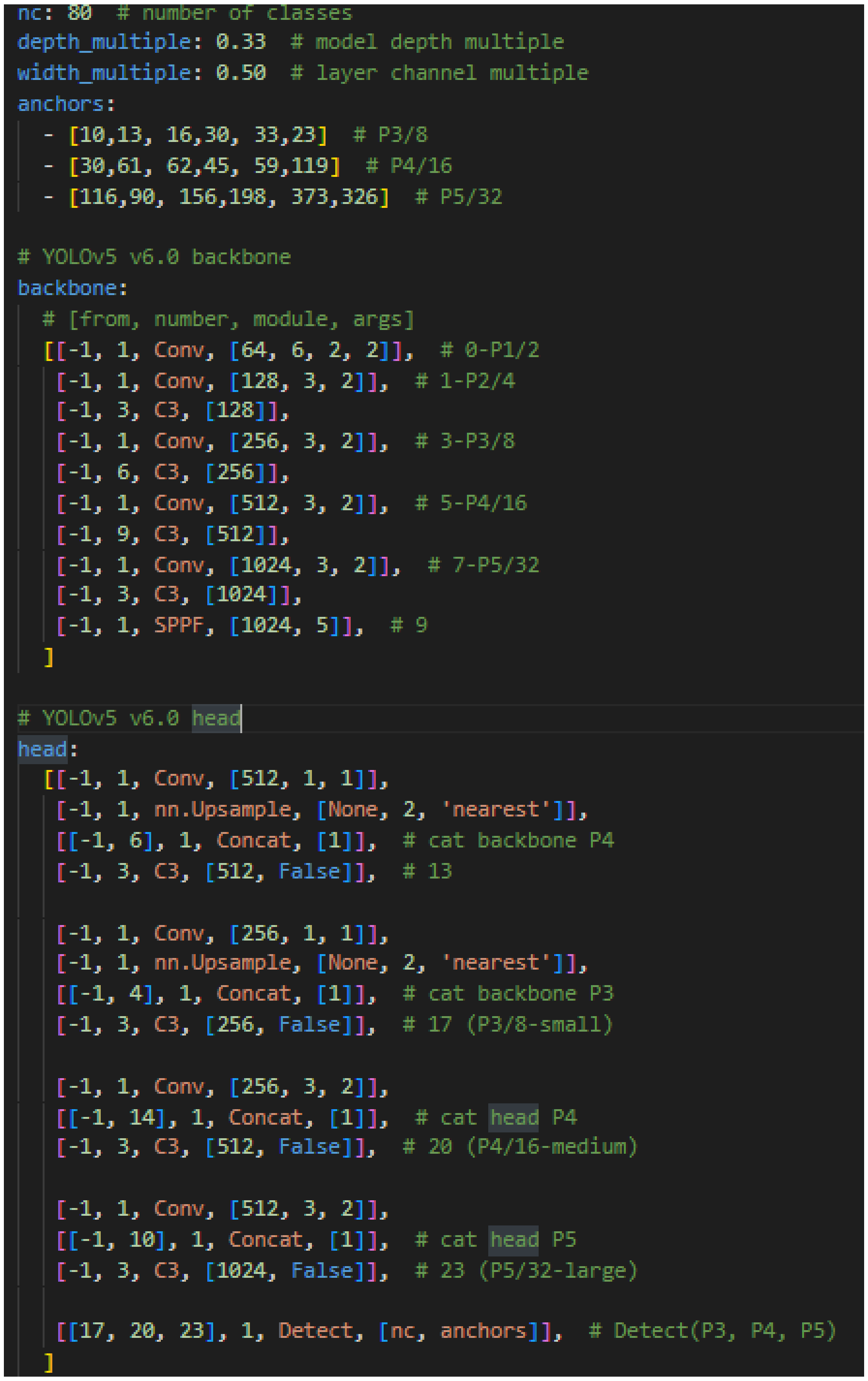
Those annotations are then used to generate train, test, and validation subsets at a 60-20-20 ratio from a subsample of 300.

Method

github repository: https://github.com/DanG-Mergin/yolov5_obj_detection

All training was based on modifications to the yolov5s model, with a focus layer, backbone, and head trained using COCO on 80 classes. GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite

In deference to time I limited model experimentation to transfer learning with fine tuning of the output layer, 3 layers + output, and 6 layers + output, and data augmentations discussed below.



```
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

See yolov5s.yaml in codebase 'models'

Weights were frozen for 18-24 layers, as augmentations were compared for different training times.

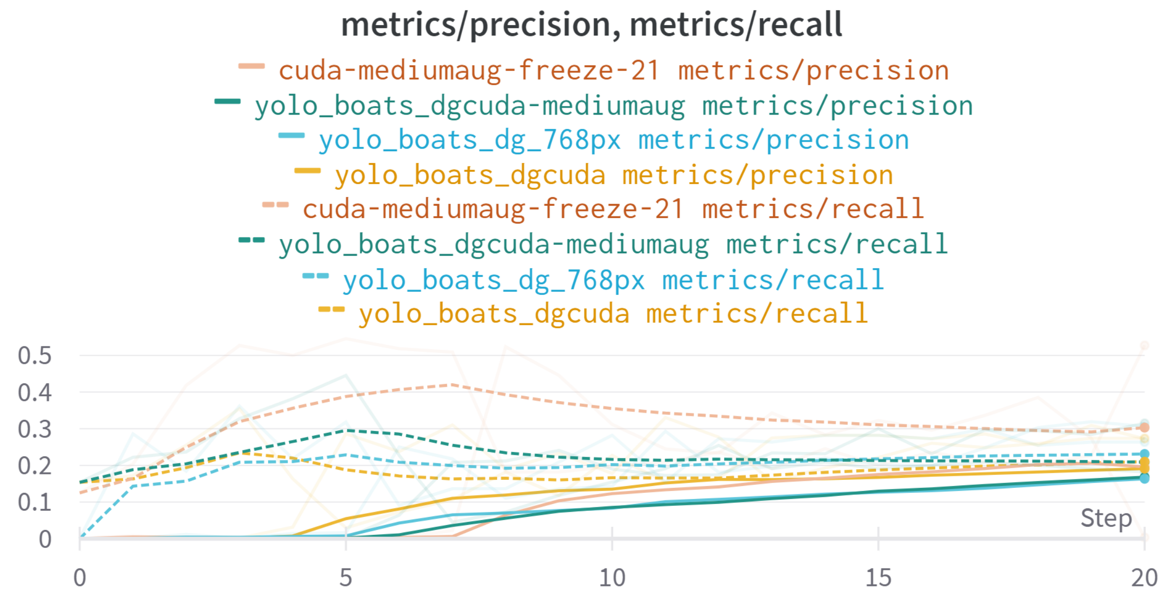
Augmentations were carried out in the following groups:

No Augmentations: Only the built in mosaic functionality

Low Augmentations: See hyp.scratch-low.yaml
Med Augmentations: See hyp.scratch-med.yaml
Gaussian + Otsu + Sobel + Low/Med/No
Gaussian + Low/Med/No
Hybrid: Half of images given Gaussian + Otsu + Sobel, Half unaugmented

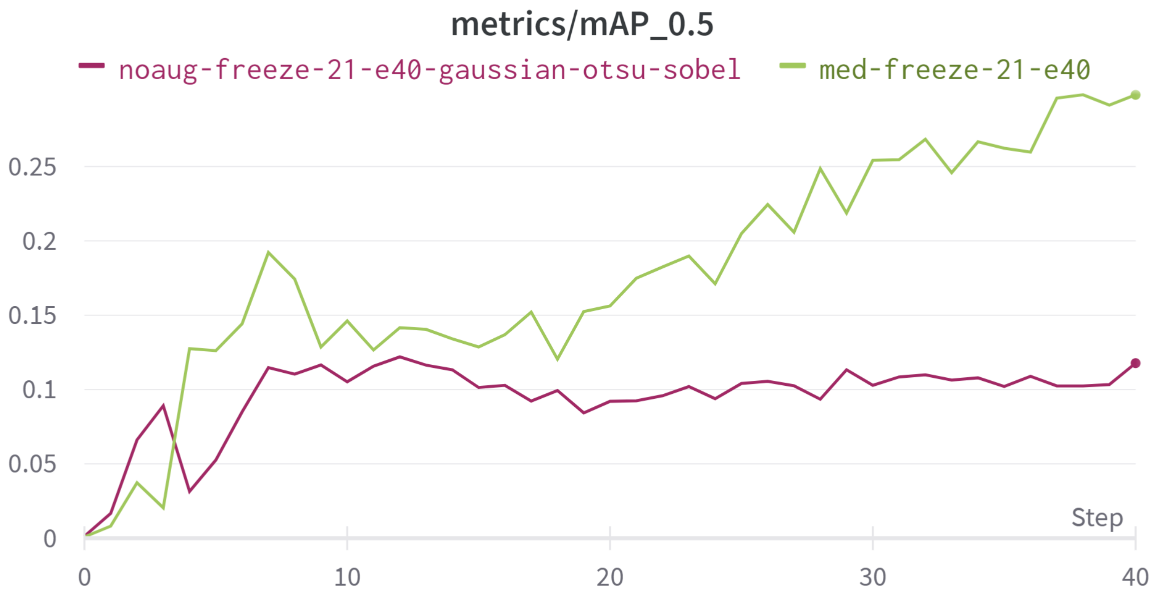
Results

Initial tests revealed little early stage differences in precision, accuracy, or the combined mAP scores. Below is a sampling of various attempts with 32 batch size x 20 epochs, no, low, medium augmentation, 500 initial image scaling vs the full 768px. cuda-mediumaug-freeze-21 even shows little difference with an additional 3 layers to train

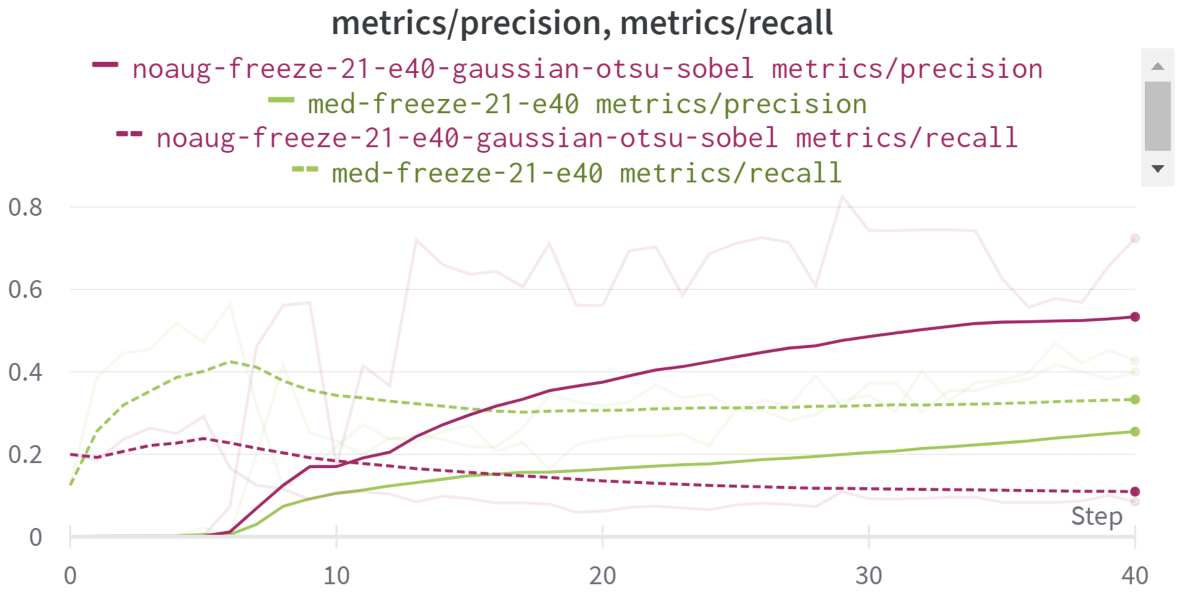


Next I modified all of the training images by converting to greyscale, used gaussian filtering to reduce high frequency noise, applied Otsu thresholding to enhance separation from the background, and applied a Sobel filter to that result in order to enhance the resulting edges.

To “even the playing field” I used the supplied “normal medium” augmentations provided by the model, vs no additional augmentations. At first glance this approach didn’t seem to help with our mAP score.

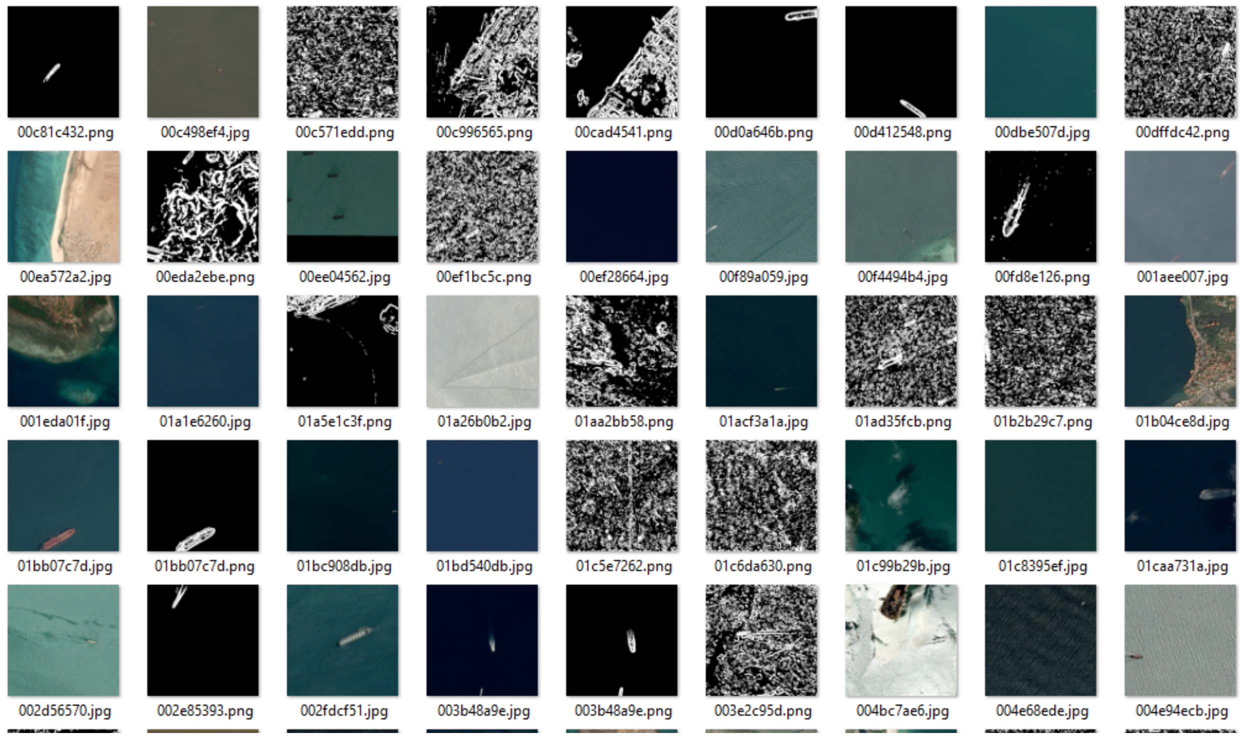


Examining precision and recall more closely revealed that the pre-augmented images were enhancing feature detection at the expense of decreasing overall detection.



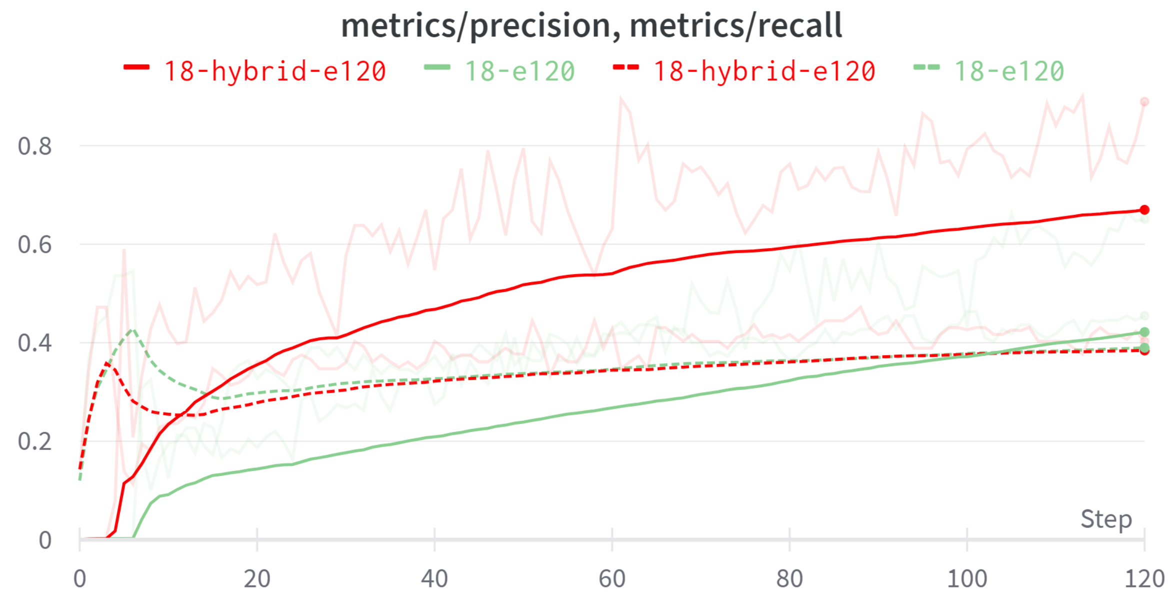
As the difference between precisions was greater than that between recall, I first compared just a gaussian filter, an otsu threshold, and various other combinations which evidenced worse performance compared to the standard medium with 21/24 frozen layers.

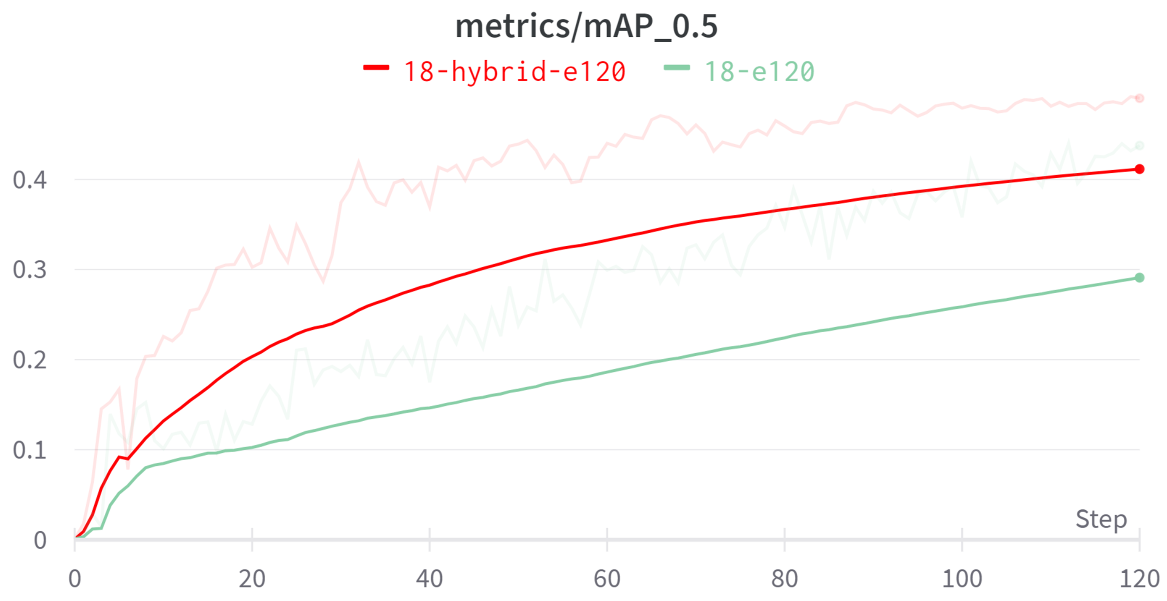
In an attempt to combine the best features of the supplied medium augmentations with our noise suppression and feature enhancement, I then created a hybrid training set which was a hybrid of unmodified images and pre-augmented, and ran the same comparison with more unfrozen layers, and 120 epochs for both. The hybrid pre-augmented training set was not further modified by the yolov5 batch augmentation process.   .



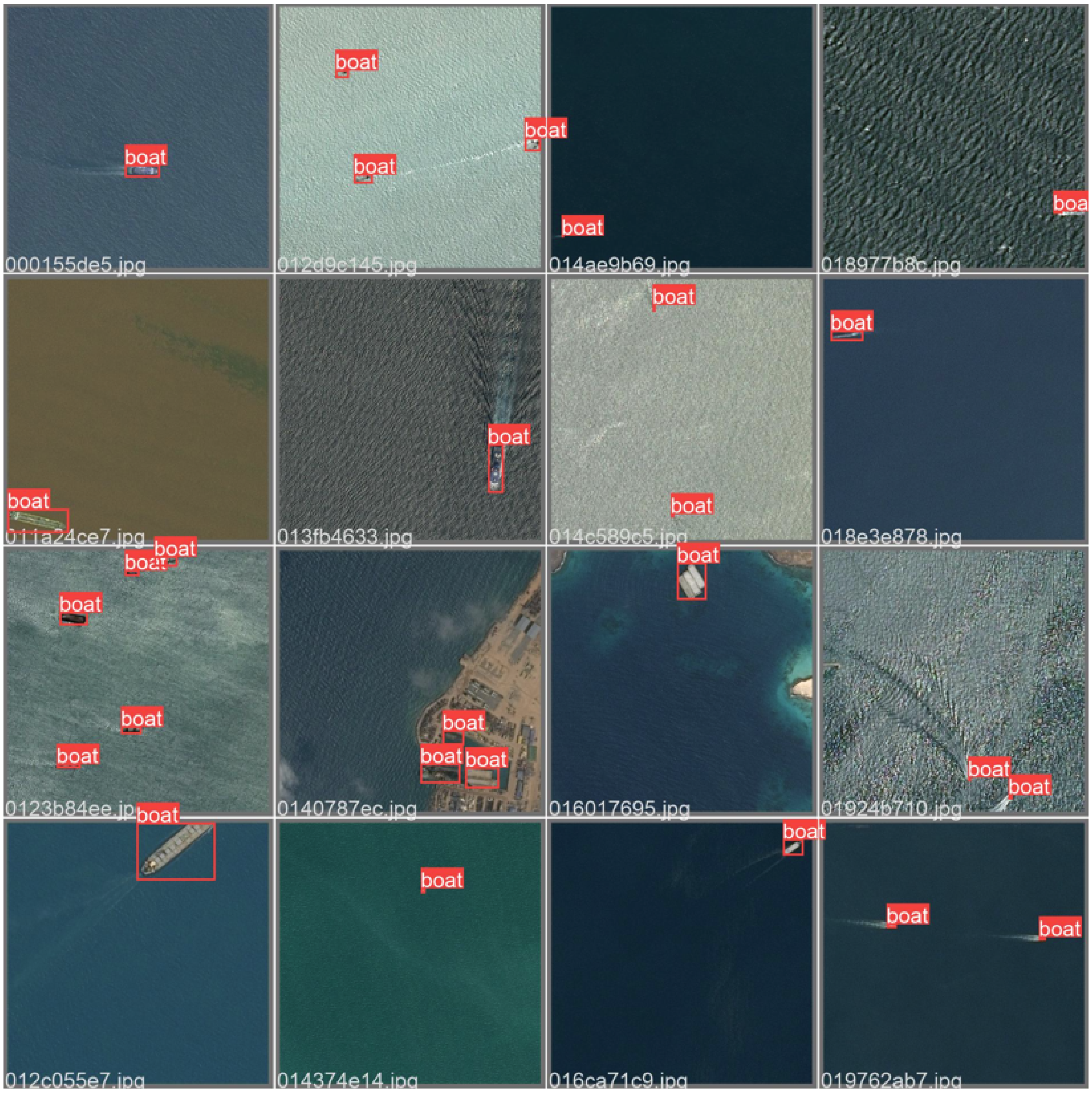
Sample of "hybrid" mixed with unmodified images

While more training would benefit both, the hybrid approach seems to be an improvement with better precision and very similar recall, for a better overall mAP score.





Sample Validation images for the 18-frozen layer hybrid pre-augmentations after 120 epochs:



boat
000155de5.jpg
012d9c145.jpg
014ae9b69.jpg
018977b8c.jpg
011a24ce7.jpg
013fb4633.jpg
014c589c5.jpg
018e3e878.jpg
0123b84ee.jpg
0140787ec.jpg
016017695.jpg
01924b710.jpg
012c055e7.jpg
014374e14.jpg
016ca71c9.jpg
019762ab7.jpg

Citations

Huang, Jonathan et al. “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors.” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017): 3296-3297. https://arxiv.org/pdf/1611.10012.pdf

Jocher Glen 2021 discussion, focus layer yolov5
https://github.com/ultralytics/yolov5/discussions/3181

Charts generated using https://wandb.ai/home

Github repository for this project: https://github.com/DanG-Mergin/yolov5_obj_detection