Monterey Bay Aquarium
Research Institute

Hyperparameter Evolution For Deepsea Detection and Tracking

Gozzi, Manuel, Monterey Peninsula College

Mentors: Duane Edgington, Danelle Cline

Summer 2022

Keywords: Deep Sea, Computer Vision, Deep Learning, Tracking

ABSTRACT

Machine learning networks are capable, but computationally intensive programs. The first experiment performed was to utilize test time augmentation, which mutates data during inference which, in theory, helps the model to better understand the object in question. This proved to yield less than desirable results when benchmarking against the pre-trained YOLOv5 Deepsort system. The test time augmentation (TTA) results yielded a decrease in both F1 and MOT A scores. In order to increase the accuracy of these models while minimizing excess computations, new features such as hyperparameter optimizations were explored, while the TTA idea was discarded. The network was trained on 17 different benthic species with a total training data set of 1105 images. These hyperparameters were trained in a relatively short number of epochs so as to rapidly explore which parameters were most effective. The parameters were evolved using the integrated genetic algorithm in the YOLOv5 machine learning library. Initial experimentation showcased a significant improvement even in the early epochs of the test runs. These findings showcased that the train time augmentation that hyperparameters provide would be able to more finely tune for deep-sea imagery by adjusting for hue,

saturation, value, and various other parameters. The hyperparameter-trained version of the model showcased a significant increase in the mAP score in preliminary testing compared to the baseline model. The network configuration chosen was the YOLOv5s setup as it yields an acceptable mAP value range while being significantly quicker to compute than the larger configurations such as YOLOv5l.

INTRODUCTION

Analyzing deep-sea imagery requires copious amounts of manual review by specialized video annotators which can be both slow and costly. By implementing machine learning systems to offload the bulk of this work, both time and money can be saved when analyzing footage for statistical analysis and objects of interest. With numerous datasets collected from autonomous underwater vehicles (AUVs) and other platforms, the collection of video footage far exceeds the current capabilities of video analysis with manual review. For this reason, machine learning systems have been implemented in order to reduce the burden on annotators, and autonomously collect statistics as close to real-time as possible. Though these autonomous systems are decent, their performance is not perfect. Various parameters can be quickly calculated based on the dataset which opens new possibilities for recording location-based species densities. This information can be used to detect unusual population changes, overgrowth, threatened populations, and movement patterns.

The first goal of this project was to use an existing trained YOLOv5 detection model, and see if the test time augmentation system (TTA) would provide a performance increase that was worth the extra compute cost. The basic concept of TTA is to augment the image input data by running various transforms such as flipping and scaling, to see if the results are favorable for learning rate and accuracy (Dufour). YOLOv5 is a convolutional network (CNN) that is based on the "You only look once" principle (Ultralytics). A convolutional network is based upon the use of various filters to iterate over an image and detect characteristics of interest (Convolutional Neural Networks). These filters "convolve" over the entire image, and yield various perspectives on the image data in the scene. This allows more subtle details to be detected by the network which makes CNNs perfect for the deep sea imagery explored in this project.

In addition to image manipulation, multiple resolutions of the same image are passed to inference upon during TTA. A combination of below and above-average resolutions are analyzed, in comparison to the standard resolution that is utilized without data augmentation (Ultralytics). To be specific, enabling TTA increases "the image size by about 30%" while taking "about two to three times longer than normal inference as the images are being left-right flipped." The TTA method is particularly useful for smaller datasets such as those of underwater imagery, as collecting this data is expensive and time consuming. YOLOv5 contains an option to use test time augmentation, namely the –augment flag, allowing us to quickly utilize these transforms to, in theory, improve the recall and precision of the model by analyzing the object from various angles, viewpoints, etc.

The metrics used for the TTA benchmarking consisted primarily of the F1 score and the Multiple Object Tracking Accuracy MOTA score. The F1 score is a combination of both the recall and precision metrics (Korstanje). This is important, as a model can find a precise bounding box around objects, but at the same time classify the object incorrectly and vice versa. For this reason, unifying the two metrics into one allows us to optimize around one metric. The second metric used during this section was the MOTA score. The MOTA score pertains primarily to the tracking system that this project was designed to improve on. The MOTA score stands for multiple object tracking which, in the current pipeline, is dependent on how well the YOLOv5 model classifies the objects in a scene (Multiple object tracking). This metric allowed us to see how significantly TTA would affect the tracking portion of the project if it proved to show promise. The last metric was the loss metric. Essentially, "if the deviation in the predicted value [compared to] the expected value by our model is large, then the loss function gives [a] higher number as [an] output, and if the deviation is small & much closer to the expected value, it outputs a smaller number" (Pedamkar, P).

The second focus of this project was to find if hyperparameter optimizations would provide an appreciable improvement to the performance of the machine learning network. Hyperparameters act as "higher-level parameters that control the learning process" of the network (Dilmegani). These hyperparameters are mutated using a genetic algorithm provided by the YOLOv5 model (Ultralytics). With the exception of a few

parameters, these values don't require alteration during the runtime of the network, thereby avoiding a significant increase in computation time. The values of most significance, in regards to the model speed, would be the initial and final learning rate values which do alter the runtime of the model; although, the learning rate should remain quite similar regardless of whether hyperparameters are used. This makes the data more palatable for the network to learn from, increasing learning rate. In essence, the primary use for these hyperparameters is to make a machine learning system perform better in fewer epochs than it would take without manipulating the data beforehand. This may also mean that as the system is trained for more epochs, the time at which the system plateaus may reach a higher overall mean average precision (mAP) value than previously possible.

The primary metrics used were precision, recall, and a combination of the two previous metrics, namely the mean average precision (mAP). The precision metric showcases "the ratio of correctly predicted positives and predicted positives" (Simic). In other words, is the system overestimating true predictions or not. The next metric, recall, is a measure of how well the network is at identifying objects when they are of the right class (Simic). In other words, a low recall showcases that the network is marking objects that are not of the specified class, as being a member of said class. This means the network doesn't entirely understand the unique qualities of the object class well enough. The last primary metric is the mean average precision (mAP). This value represents the averaged combination of precision and recall metrics of each class, making it a valuable metric to maximize (Gad). This is the exact metric that was maximized during the evolution of the hyperparameters, as the balance between recall and precision is more encompassing than just recall or precision.

One key feature of the YOLOv5 network is the auto-anchoring system. This system utilizes a genetic algorithm to find the best anchor values for object dimensions (Oni). The anchor values essentially assign certain shape dimensions that are most likely to fit the class types. This is important for datasets that contain objects with high shape variability. For example, a normal square bounding box would have a hard time differentiating a long eel in comparison to a clam since the majority of the data inside of the bounding box wouldn't be the eel, but rather everything other than an eel. Focusing the bounding box on the object itself allows quicker computations as the excess pixels are

discarded while reducing the confusion between non related pixels. YOLOv5's auto anchor system recalculates these anchor values upon every training session, attempting to best fit the data every time within a certain threshold. This system benefits the test results at times but also hinders reproducibility. The ability to keep these anchors the same across training sessions has proven to be difficult.

MATERIALS AND METHODS

Data

The data used in the TTA portion of the study consisted of 691 classes, with a total image count of 33,667 images, with 20% of those images reserved for validation. The data used throughout the duration of the hyperparameter evolution section of this study consisted of 1105 images, of which were categorized into 17 classes of benthic species. The AWS (Amazon) SageMaker resources used for this study can be found via the Github URL provided under the Gozzi, M. citation.

Test Time Augmentation

To test TTA, a pre-trained model for the YOLOv5 Deepsort machine learning system was utilized and the TTA flag provided by YOLOv5 was appended to the benchmarking command, namely, the –augment option. This method was chosen for its rapid results which don't require retraining the machine learning system. The benchmark was performed on an Amazon SageMaker ml.g4dn.xlarge instance and the labels were downloaded locally. The input video of choice was a 10-minute video benchmark that could be compared to the previous non-TTA model. The output labels containing the data were downloaded and input into the vaa-track-benchmark tool, and the results were processed locally. The benchmark outputs a .csv file that automatically calculates the metrics of the network against the validation data.

Machine Learning Network of Choice

The machine learning network used in this study was the YOLOv5 model from Ultralytics. The YOLOv5l configuration was used in the preexisting model that TTA was tested upon. The configuration used for the experimental hyperparameter evolution tests was the YOLOv5s model because of its quick runtime. This allowed for more cost-

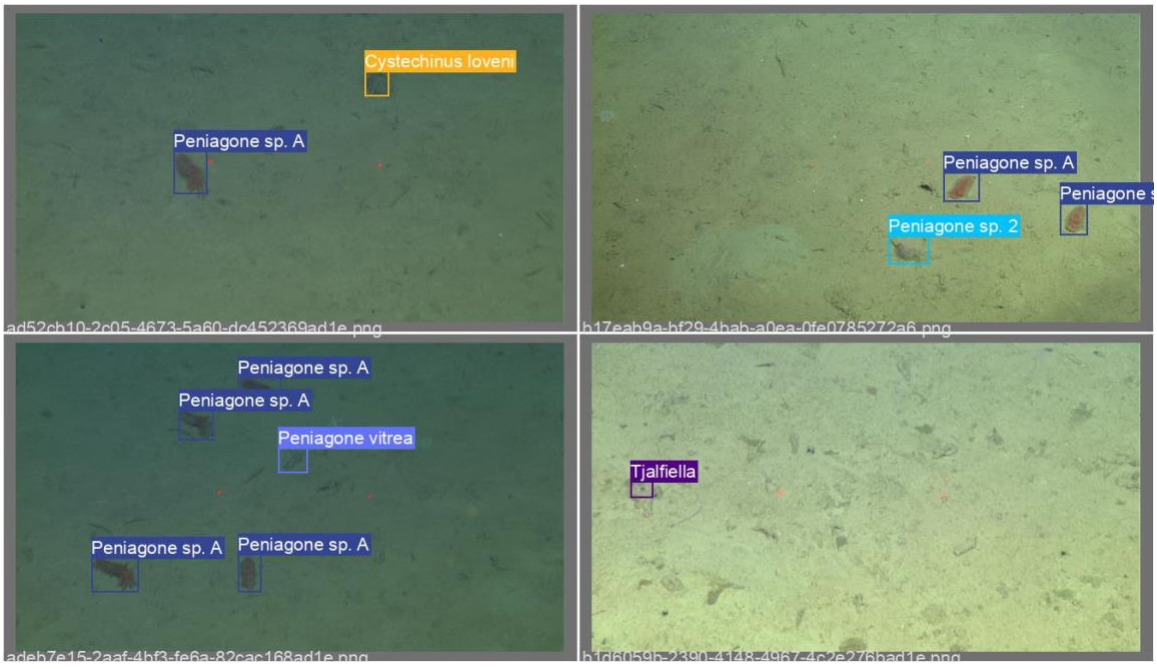
effective tests while utilizing Amazon's Sagemaker Studio. Hyperparameter evolution runs were performed for 30-50 evolutions, with each evolution consisting of 25 epochs. This process was performed on a ml.g4dn.xlarge GPU ML EC2 instance using the PyTorch 1.10 GPU optimized kernel for the notebook environment. The cost of on-demand computation from this instance at the time of writing is $0.7364 per hour (Mishra). Training epochs took an average of around 64 seconds. This means that a 50-epoch training session would cost about $0.65. Results were logged to a WandB (https://wandb.ai/mbari/902005-vaa/reports/Hyperparameter-Evolution-For-Deepsea-Tracking--VmlldzoyNDA5MTA4) graph in order to check the key metrics as previously discussed.

Due to the relatively high cost of training, a shorter epoch count was used in order to find the most effective hyperparameters in the earlier stages of training. These results are thought to translate even when extended to larger epoch counts making them a useful data point to improve network performance. The three primary files of concern in order to configure the YOLOv5 system are the following files: the yolov5/utils/metrics.py file used to define the fitness function, the /deepsea-yolov5/opt/ml/input/data/ hyp.scratch-low.yaml file for our hyperparameter defaults and optimal values, and the /deepsea-yolov5/opt/ml/custom_config.yaml for our dataset configuration. The hyp.scratch-low.yaml file is utilized as the hyperparameter configuration file for manipulating the training input data. This file contains various transforms such as gamma, hue/saturation/value, scale, and more which are all intended to be tuned towards our specific dataset. The contents of this file will need to be replaced upon completing a full evolution cycle. The most effective hyperparameters will be automatically saved under the name hyp_evolve.yaml. This file specifies which evolution was the fittest, along with the metrics of the best training run, and the total number of evolutions tested. The contents of this file must be copied into the hyp.scratch-low.yaml file when running the full 50 epoch sessions for optimal results. The last file of importance is the yolov5/utils/metrics.py module which is home to the fitness function. By default YOLOv5 defines the most fit evolution weighted as 90% of the mAP_0.5-0.95 and 10% of the mAP_0.5 (Ultralytics). For our application, this weighting must be switched as the mAP_0.5 is the metric that should be maximized due to the importance of approximate

object recall rather than the precision of bounding boxes.

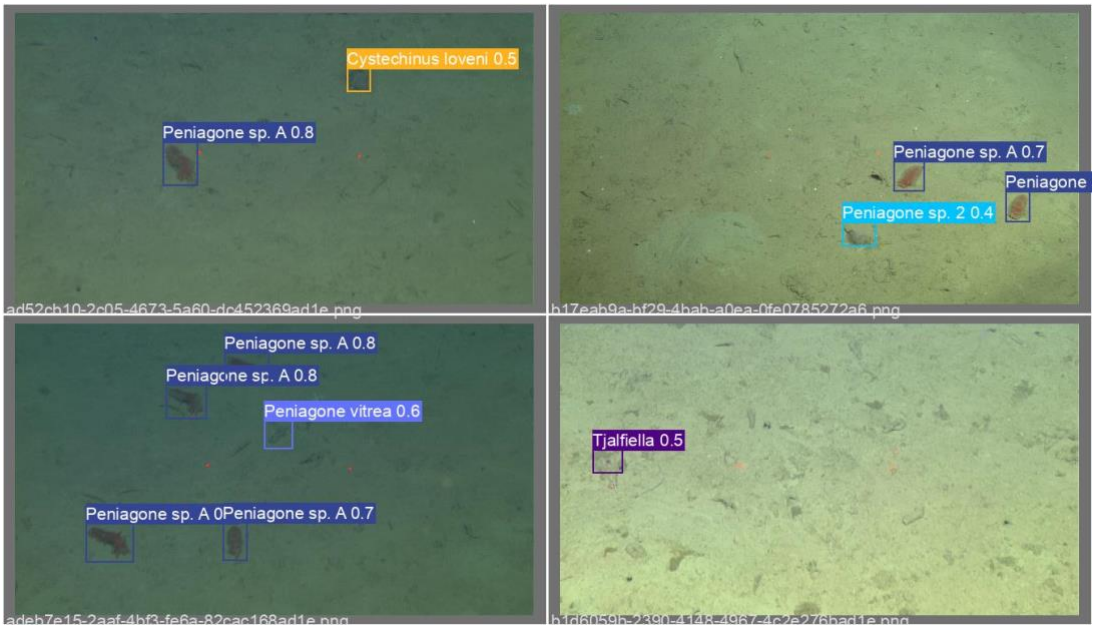
Dependencies are downloaded from the YOLOv5 repository directly. Version 6.1 is utilized in this study and is the latest at the time of writing. The MBARI m3-download tool was used to extract the annotated benthic training and validation images from the MBARI database. This was then compressed and uploaded to an Amazon S3 bucket. The dataset is loaded onto the instance from an Amazon Simple Storage Service (S3) using the setup Jupyter notebook. The data must then be decompressed from the .tar.gz format using another script located under the setup notebook. Before being ready to train the dataset, the downloaded data must first be split into two categories: training and validation data. This is done utilizing the split.py script located in the setup notebook file. In order to graph the results of both the normal training and evolution training runs, WandB must be imported utilizing pip. Simply install WandB and login in order to utilize the graphing feature. An image size of 640x640 is optimal for this dataset as larger resolutions haven't proven to yield much better results according to the prior work of my mentors. As this resolution increases, the network will take longer to process as there is more pixel data to process.

The model is trained on prelabeled underwater imagery such as the image below.



Deepsea annotated data from the MBARI database.

This image represents the labeled data, with each bounding box representing an object with 100% certainty. The next image showcases a prediction from the model. Note that the above image is a validation image and the model was not actually trained utilizing this image. The validation data is novel to the model and is therefore the best representation of the network's understanding of the classes that it has been trained on when benchmarked.



Prediction output from the trained model.

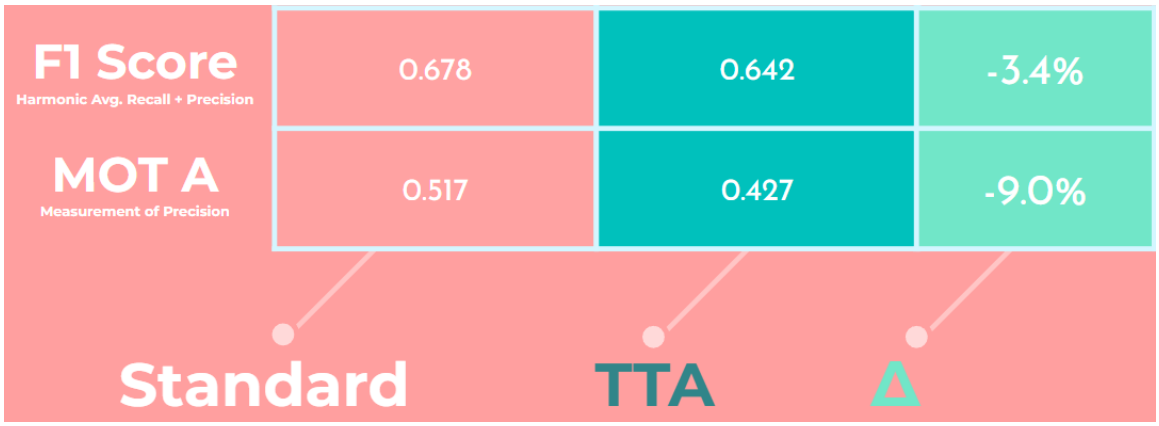
Here we can see that the model associates a number next to the bounding boxes of the object. This represents the certainty of the model's prediction. Values that are close to 1 represent high certainty, while values approaching 0 represent low confidence.

RESULTS

Test Time Augmentation

The test time augmentation results yielded a decrease of -3.4% in regards to the F1 score, and a decrease of -9.0% in the MOT A score. Overall this showcases that the pre-trained model doesn't obtain any benefit by viewing input data from different perspectives. It seems as though manipulating the imagery during inference may confuse the network since it is trained from a more standardized viewpoint. It would follow that the flip operations performed during the augmentation cycles simply confuse the model

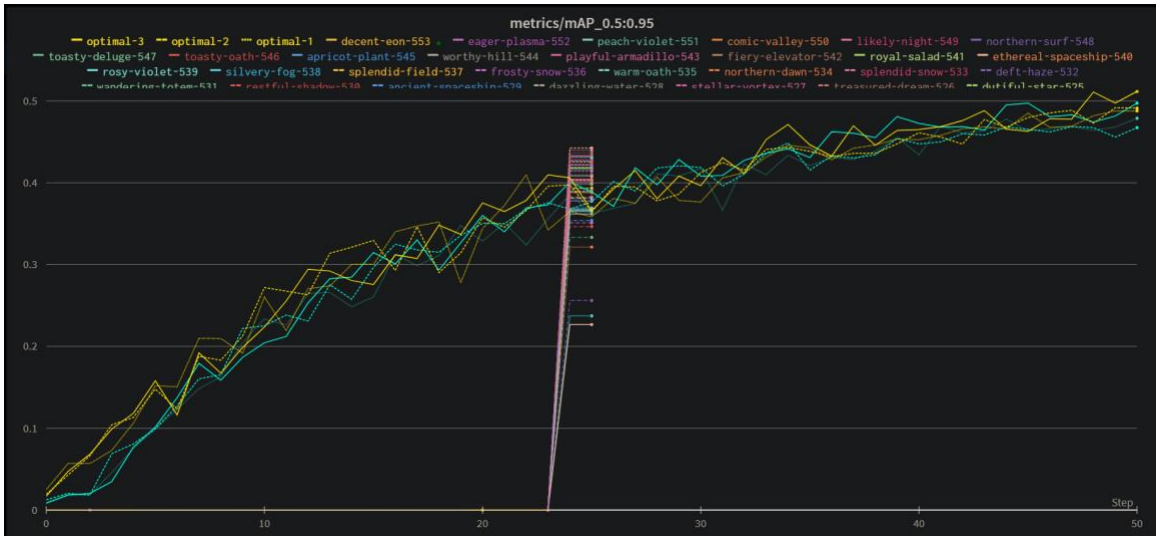
rather than providing a better understanding of the object in view. Fortunately, since test time augmentation is an expensive operation, this discovery means that excess computation time isn't the answer to better performance. This finding pivoted the research project towards the next section.



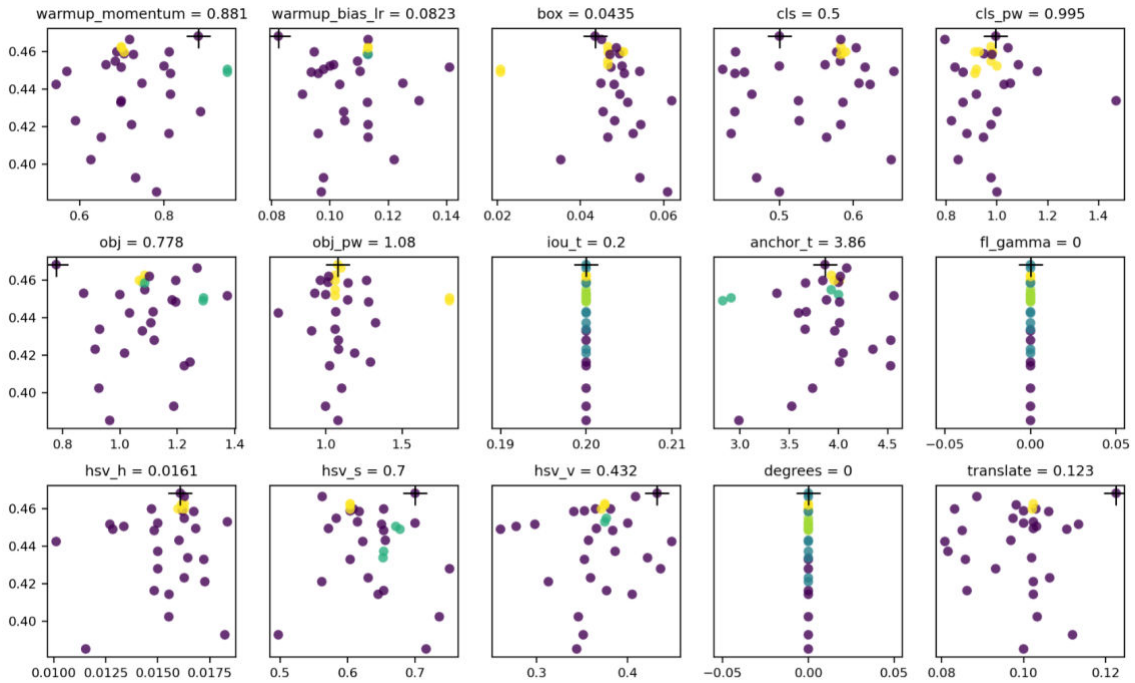
Comparison of TTA vs the standard non-TTA model. The TTA model shows a significant decrease in performance as shown in the delta column.

Hyperparameter Evolution Training

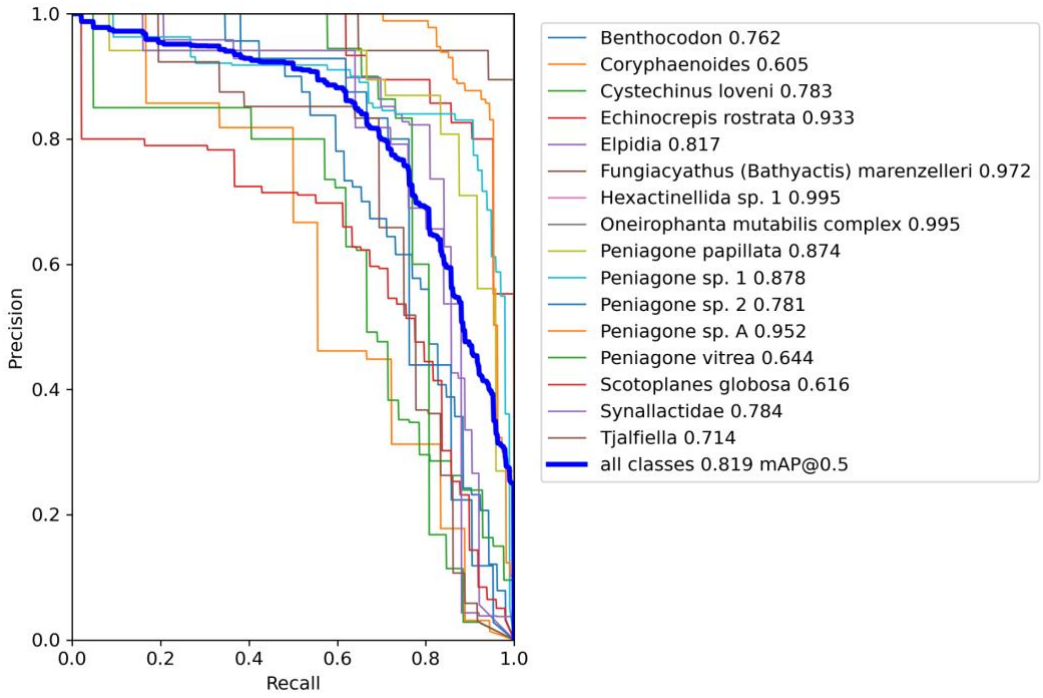
The baseline model using the provided hyperparameters trained on the COCO dataset, the default dataset used by YOLOv5, resulted in a mAP_0.5:0.95 of nearly 48.1% with a mAP_0.5 of 78.6% (Common objects in context). The optimal hyperparameters found by using the hyperparameter evolution option offered a respective improvement of about 1.6% for a mAP_0.5:0.95 of 49.7% and an improvement of 1.4% for a mAP_0.5 of 80.0% when averaged across 3 training sessions.



A WandB graph showcasing the evolution runs (at 25 epochs) and full length (50 epoch) runs utilizing the best hyperparameters during training, and the baseline with default parameters. The y-axis showcases the mAP value while the x-axis shows the epoch count. Access via: https://wandb.ai/mbari/902005-vaa/reports/Hyperparameter-Evolution-For-Deepsea-Tracking--VmlldzoyNDA5MTA4?accessToken=vk7hysjojmgbgml9igri06sg2lwzqeusyxq6lk70j8xt6hfeadhz4tt5ht1mdnbh.



A subsection of the evolve.png outputted when running evolutions. Optimal values are yellow in color while poor performers are in purple. Green/blue values perform half decent. Graphs that only show only a vertical component are constrained to be immutable as they don't provide useful augmentations.



A recall vs. precision graph showcasing the optimal balance between the two metrics for each class explored in this study.

DISCUSSION

As seen in the TTA results, the model doesn't seem to have any benefit from the extra viewpoints of the same image. This points to the fact that the network doesn't fundamentally understand the objects well enough to account for these changes in perspective. This makes sense as convolutional networks are rotationally reliant on the source imagery. In other words, if the input data is not in different orientations, the model will have a difficult time creating a correct prediction. For this reason, training with data augmentation was a natural choice. The reasoning for the counterintuitive TTA results is thought to be due to the relatively small dataset that was used to train the network on. Although there were 33k images, there was also a larger class number leaving only around 50 training images for each class if the training data was uniformly distributed. This leaves little time for the network to bake in orientation changes into its understanding of the classes. For this reason, the refeeding of augmented data didn't prove useful as the network doesn't have an orientation-neutral understanding of the class yet.

Due to a few bugs and usage issues, YOLOv5 has given some difficult to reproduce results. The primary issue being the reproducibility of the best case hyperparameter training sessions. Due to the auto anchor feature, the network is unable to be trained with exactly the same parameters from run to run which makes the results less reliable than desired. According to the experimental evolution training sessions, the performance gains of the optimal hyperparameters were sometimes in excess of 10% in the mAP_0.5 score when tested at 10 epochs which indicates a significant increase in model performance. After some back and forth with the maintainers of the YOLOv5 *repository, the reasoning behind this was not clear as the answers provided were vague. The problem with lower epoch counts is due to the change in learning rate when translating to higher epoch counts as a result of the learning rate scheduler (Ng, R.). This means that the model will make stronger changes to the weights when lower epoch counts are used, and more finely tune the model when a higher count is used. This is why results are difficult to translate when testing on low epoch counts.

An interesting finding during this project was that when rotation was not constrained, the hyperparameters that were output performed worse than with the standard rotation of 0 degrees. This was perplexing as the imagery is primarily a top-down view, which should benefit from rotation augmentations. In this regard, our suspicion is that the dataset used in the evolution portion of this project was far too small to have well-translated results. Solidifying a standard view of the classes was most likely possible with the dataset that we have, however learning an orientation neutral understanding of the data may have been too coarse of an adjustment for this small dataset.

CONCLUSIONS/RECOMMENDATIONS

Test time augmentation has shown to be both detrimental to model performance, while also costing about 3 times as much in terms of computation time. This concludes that TTA is not optimal for underwater imagery, at least for the benthic species and datasets that were explored in this study.

Optimizing hyperparameters did not show massive improvements as the training epoch count increased; rather only a couple of percentage points of improvement were

observed. This may still be useful in the long term, however, so more experimentation and longer evolution counts should be explored.

ACKNOWLEDGEMENTS

Thank you to my mentors, Duane Edgington and Danelle Cline for guiding me throughout this project. Their passion and depth of knowledge made this project much easier to transition into. Thank you to George Matsumoto for walking us through this summer internship, Megan Bassett for organizing meetings, and Lyndsey Claassen for her advice. And a final thank you to MBARI and the David and Lucile Packard Foundation for providing this opportunity.

References:

Common objects in context. (n.d.). https://cocodataset.org/#home

Convolutional Neural Networks (CNNs) explained. (2017). https://www.youtube.com/watch?v=YRhxdVk_sIs

Dilmegani, C, (2022). Hyperparameter optimization: Methods & Top Tools in 2022. https://research.aimultiple.com/hyperparameter-optimization/

Dufour, P. (2020). How to correctly use test-time data augmentation to improve predictions. https://stepup.ai/test_time_data_augmentation/

Gad, A. (2021). Mean average precision (MAP) explained. https://blog.paperspace.com/mean-average-precision/

Gozzi, M. (2022). Mbari-org/mgozzi-augment. GitHub. https://github.com/mbari-org/mgozzi-augment

Korstanje, J. (2021, August 31). The F1 score. https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

Mishra, A. (2019). Machine learning in the AWS cloud. Retrieved July 26, 2022, from https://aws.amazon.com/sagemaker/pricing/

Multiple object tracking. (2021). https://araintelligence.com/blogs/deep-learning/multiple-object-tracking/overview-multiple-object-tracking

Ng, R. (2019). Learning rate scheduling. https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/

Oni, S. (2020). Understanding anchors (backbone of object detection) using Yolo. https://becominghuman.ai/understanding-anchors-backbone-of-object-detection-using-yolo-54962f00fbbb

Pedamkar, P. (2021). Loss functions in machine learning. EDUCBA. https://www.educba.com/loss-functions-in-machine-learning/

Simic, M. (2022). Precision vs. average precision. https://www.baeldung.com/cs/precision-vs-average-precision

Ultralytics. (2022). Ultralytics/yolov5. https://github.com/ultralytics/yolov5