**SeaScout: An Underwater Organism Detector**

Student Contributors: Srihari Krishnaswamy & Vivian Wang

**Problem Description**

We built our model and the surrounding scripts to fulfill [MATE's 2023 Computer Coding Challenge](https://files.materovcompetition.org/2023/2023-OER-MATE-ROV-Computer-Coding-Challenge_FINAL.pdf).). This tasked us with identifying marine organisms in seafloor footage collected by Remotely Operated Vehicles.

The competition was divided into two levels:

* Level 1: Classify and place bounding boxes around organisms in the NOAA ship Okeanos Explorer remotely operated clips provided.
* Level 2: Identify the organisms in (a provided) broad morphological category and draw a bounding box for each instance.

**Data**

The classes that our project identifies are summarized below:

|  |  |
| --- | --- |
| **Class** | **Description** |
| Annelids | Segmented Worms |
| Arthropods | Crustaceans (shrimp, crabs, copepods, etc.), pycnogonids (sea spiders) |
| Cnidarians | Sea jellies, corals, anemones, siphonophores |
| Echinoderms | Sea stars, brittle stars, basket stars, urchins, sea cucumbers, sea lilies, sand dollars |
| Mollusca | Cephalopods (squid, octopi, cuttlefish), gastropods (sea snails and slugs), bivalves, aplacophorans (worm-like mollusks) |
| Porifera | Sponges, glass sponges |
| Vertebrates: Fishes | Cartilaginous, bony, and jawless fishes |
| Other Invertebrates | Includes tunicates (sea squirts and larvaceans), ctenophores, many worm phyla |
| Unidentified Biology | Unidentified Biology |

We based our dataset on last year's deepsea-detector project's dataset. We ended up adding a lot of images since at first, our model would label almost everything as fish. This was because of the large concentration of the species in the previous dataset. We added images from the World Register of Marine Species (WoRMS) and had to relabel some of their annotations using Roboflow. Our model was overfitting for a while during our training iterations and would still label almost everything as fish or arthropods, but we were able to solve this issue by tweaking some training parameters, as discussed in the next section.

**Object Detection Model**

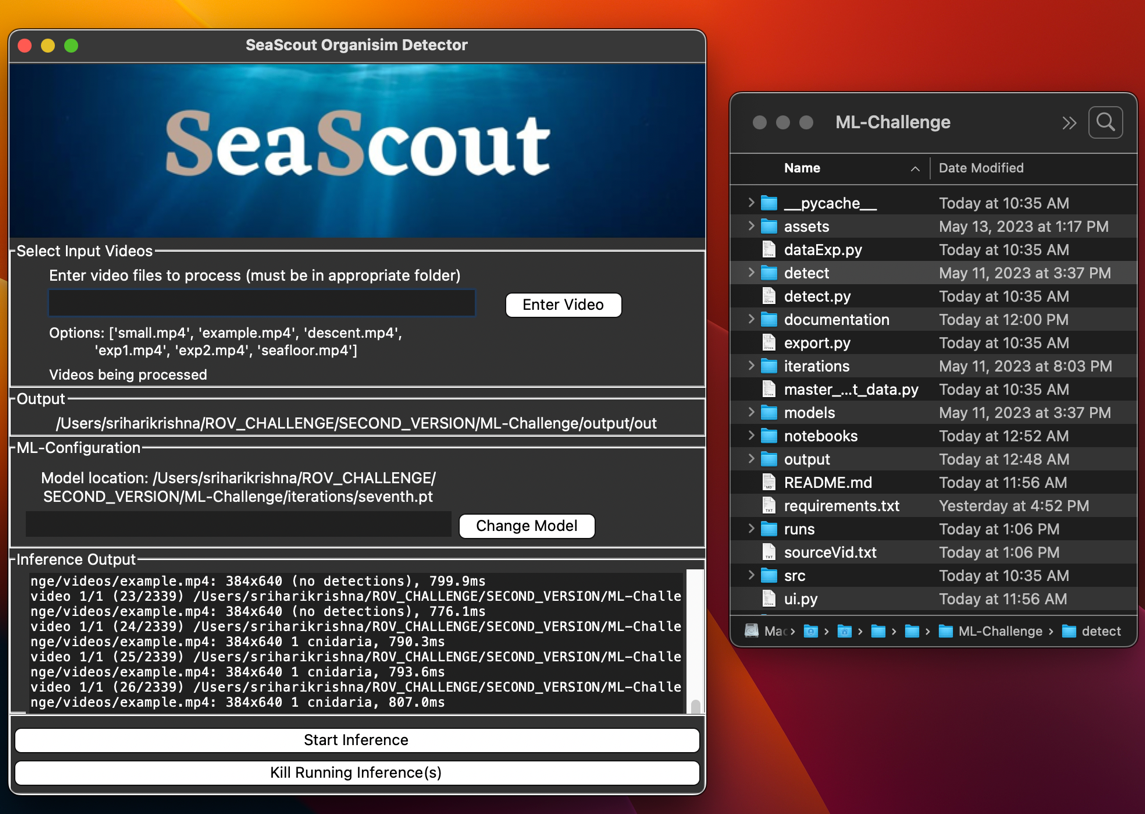
Our project uses a Yolov5 object detection model due to its popularity in the CV field and accuracy. We decided to train a pre-trained yolov5 model from FathomNet, specifically the MBARI Monterey Bay Benthic YOLOv5x model. This was because of the relatively small size of our dataset compared to those of other similar objectives, so we wanted to leverage the fact that the weights of the base model would already be tuned to detect underwater organisms.

The most accurate model we produced (located in the iterations folder of our project) was the result of us freezing (keeping the weights of) 18 layers of the MBARI model and training the rest of the layers with our data. It was important to find a good balance of layers to freeze and unfreeze, since unfreezing all the layers could detract from accuracy since we would have abandoned the weights from the MBARI model. On the flip side, unfreezing less layers would allow our dataset to create less of an impact on the model's weights.

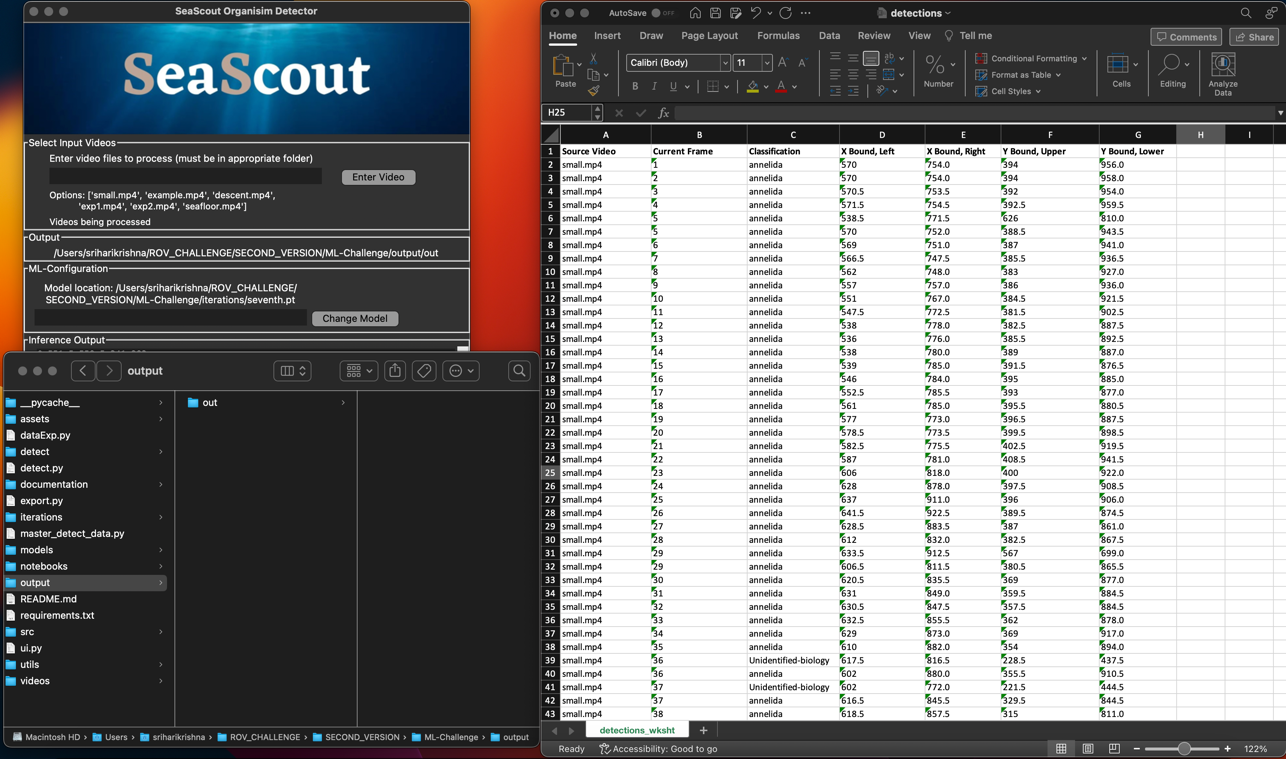
Our model was trained for 14 epochs, and all our training specs are available in our Model Training Colab Notebook. As mentioned, we faced some issues with overfitting while trying to find the optimal model with our dataset, but this was mitigated mainly by changing our number of epochs in training. We strived to create the most accurate model we could with the data we could find, but there are still some inconsistencies with the model's accuracy. Regardless, we still hold that it provides value with its detections and labels.

**UI**

The UI for SeaScout was written with Tkinter. The UI for our project allows the user to enter multiple videos to process and logs the results for the detections for each video on one spreadsheet. After a sequence of videos are processed, the processed videos with bounding boxes are available in the latest folder in the output folder, along with the spreadsheet. The user can cancel video processing at any time, but if this is done, no spreadsheet or videos with bounding boxes will be generated. It is worth noting that the same detection and logging process can be run via the Terminal/CLI, and that the UI is simply a wrapper for this.

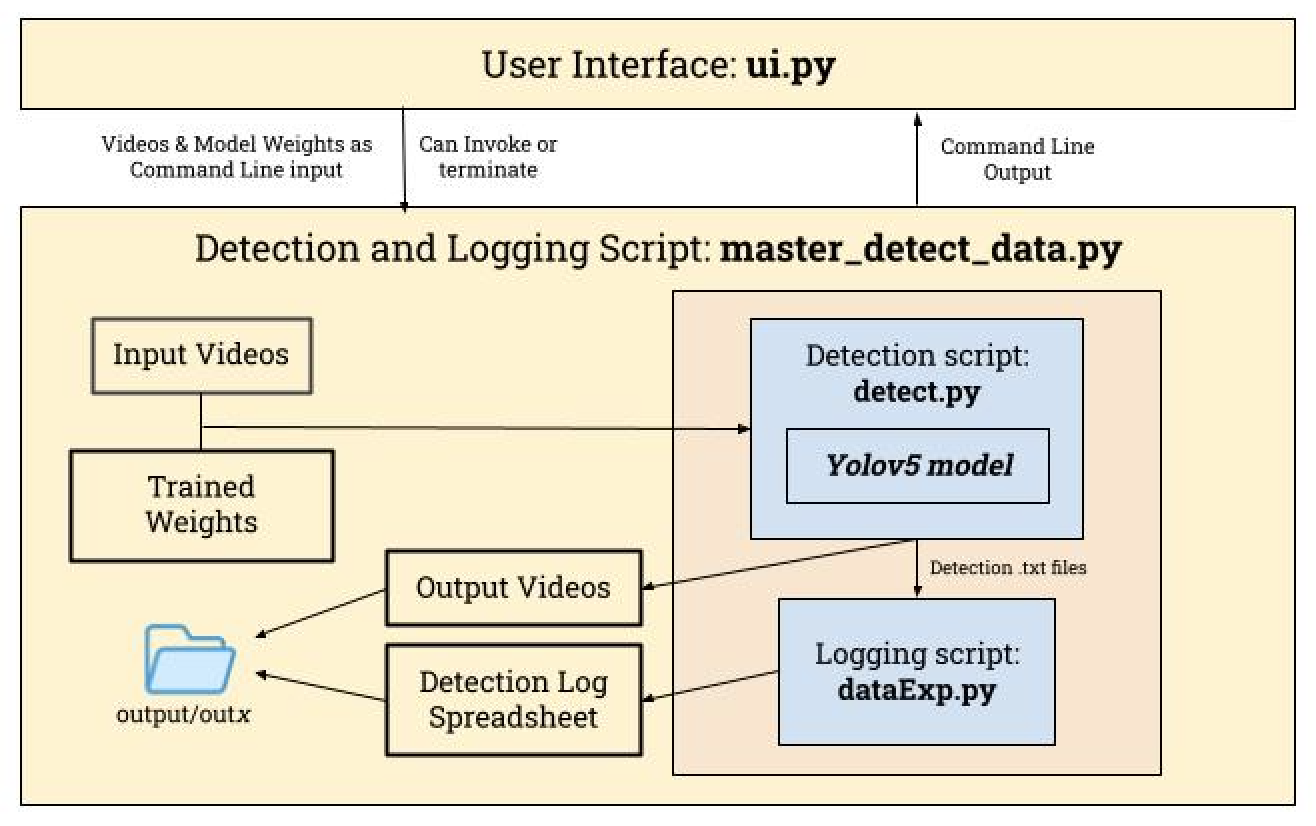


After the inference finishes, the detections are outputted to an excel spreadsheet located in the latest folder inside the output folder. The processed video files will also be available in the same folder:



**Architecture and Code Descriptions**

Below is a figure depicting the base architecture of our program:



**User Interface Script: ui.py**

This script runs the UI for SeaScout. It consists of 5 Tkinter frames (banner image, video selection, output path display, model selection, stdout display and buttons to start and kill inferences) nested within a parent frame.

There are 4 main functional components to the UI

1. **Input Video Selection:** The users can select however many videos they want to process as long as the video names they enter are valid .mp4 file names inside the /videos folder inside the project folder. Users cannot enter the same video for processing twice in the same cycle under the assumption that if they attempt to do so, it is accidental. When users enter videos, they are added to a list which will be used as standard input to the script that runs detection and spreadsheet logging. If the user enters redundant or invalid videos, they will not be added to the list, so the only videos remaining in the list at any point will be valid video entries.
2. **ML Model Selection:** Similar to input video selection, users can change the current model doing the video processing provided that they enter a valid yolov5 pytorch model in .pt format. The file name they enter must be in the iterations folder inside the project folder. By default, the selected model is our sea\_scout.pt model, so the user does not have to change the model if they do not wish to.
3. **Standard Output Display:** A new thread is created for capturing stdout that would be on the terminal, and each individual statement is added to a queue. Element by element, the contents of the queue are rendered onto a tkinter text element, and the user can see stdout on the frame. Most of the code for the stdout display was adapted from last year’s deepsea-detect project.
4. **Starting Video Inference:** Once they have entered any nonzero number of valid videos, the user can run inference. Once the button to do so is pressed, the stdout display is cleared, and the entered videos and model are passed as command line arguments to the process running detection and logging. Slightly different code is implemented depending on the user’s OS due to limitations within the python os module. A code snippet of this implementation is below:

A screenshot of a computer program

Description automatically generated with medium confidence

1. **Killing Video Inference:** Users can also terminate their video processing early – however if they chose to do so, no excel file or processed videos will be generated. In order to do this, all child processes (detection and logging) of the SeaScout CLI process are terminated, along with that process itself. Once again, different code is implemented depending on the user’s OS. A code snippet is below:

**A screen shot of a computer program

Description automatically generated with low confidence**

**SeaScout CLI Script: master\_detect\_data.py**

Given Command Line Arguments, this script invokes the detection and logging processes to sequentially process videos and write detections to the excel file for the current sequence of videos. This script is also in change of determining the folder in which the output videos and spreadsheet will be available at for the user. After processing and logging is complete, it will create a new folder for the output and move the excel and video files into it (done by the determine\_output\_folder() method). Below is a code snippet of the loop that runs inference and excel logging:

A picture containing text, screenshot, font

Description automatically generated

This script also deletes all .txt files generated by yolov5 after they are no longer needed, as well as any yolov5 output other than the processed videos.

**Yolov5 Detection Script: detect.py**

This script is provided from the Yolov5 repository and runs the detection process. This script was slightly modified to log non-normalized coordinates (pixel numbers) of bounding boxes as opposed to normalized coordinates (expressed as a coordinate between 0 – 1 relative to the screen size). The script works by loading the model, running the inference algorithm on it, processing results, and writing and saving results. Code snippets are provided below.

Code snippet for model loading:

A screen shot of a computer code

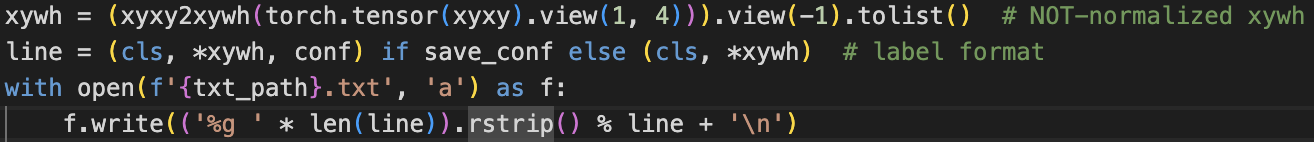
Description automatically generated with low confidence

Code snippet for running the inference algorithm:

A screen shot of a computer program

Description automatically generated with low confidence

Code snippet for writing results to .txt files (xywh stands for top left x & y position, width & height):



**Excel Spreadsheet Logging Script: dataExp.py**

After the current video has finished processing from the detect.py script, the .txt files generated (one per detection) are iterated through, and the contents of them are transcribed to a single excel file. This script works in a few different stages:

1. Each .txt file is read, and the raw contents are written onto a singular .txt file. The frame number is parsed from the title of the .txt file being processed. At this point, almost all of the data needed for the spreadsheet is ready, but the bounding coordinates need to be calculated and the class of the detection is represented by a number (index of the list of classes).
2. The data from the first output .txt file is read into a large dictionary of all the detections and sorted by frame. The class is converted to a string by accessing the list of classes, organized by the same indices. The rightwards x-bound and downwards y-bound are calculated by adding the width and height to the top left x and y coordinate respectively.

Code snippet for transforming read data into the values that will be later written to the spreadsheet:

A picture containing text, screenshot, font

Description automatically generated

Code snippet for sorting the dictionary with all the data by frame:



1. The dictionary is then written to the excel spreadsheet. This is done using the xlsxwriter module. If the video whose data is being processed is the first video in the cycle of videos to process, a new spreadsheet is created, with headers and data written to it. If not, then the data is simply written from the dictionary to the first vacant row of the spreadsheet. This allows the program to write the detection logs of multiple processed videos to the same spreadsheet.
2. Because of formatting issues that arose in continuously writing to the same spreadsheet, any formatting inconsistencies are solved by the fixFormat() method. It iterates through the spreadsheet and eliminates empty rows at the top of the spreadsheet that are created by writing to the same spreadsheet. This is done with the xlsxwriter and openpyxl modules.

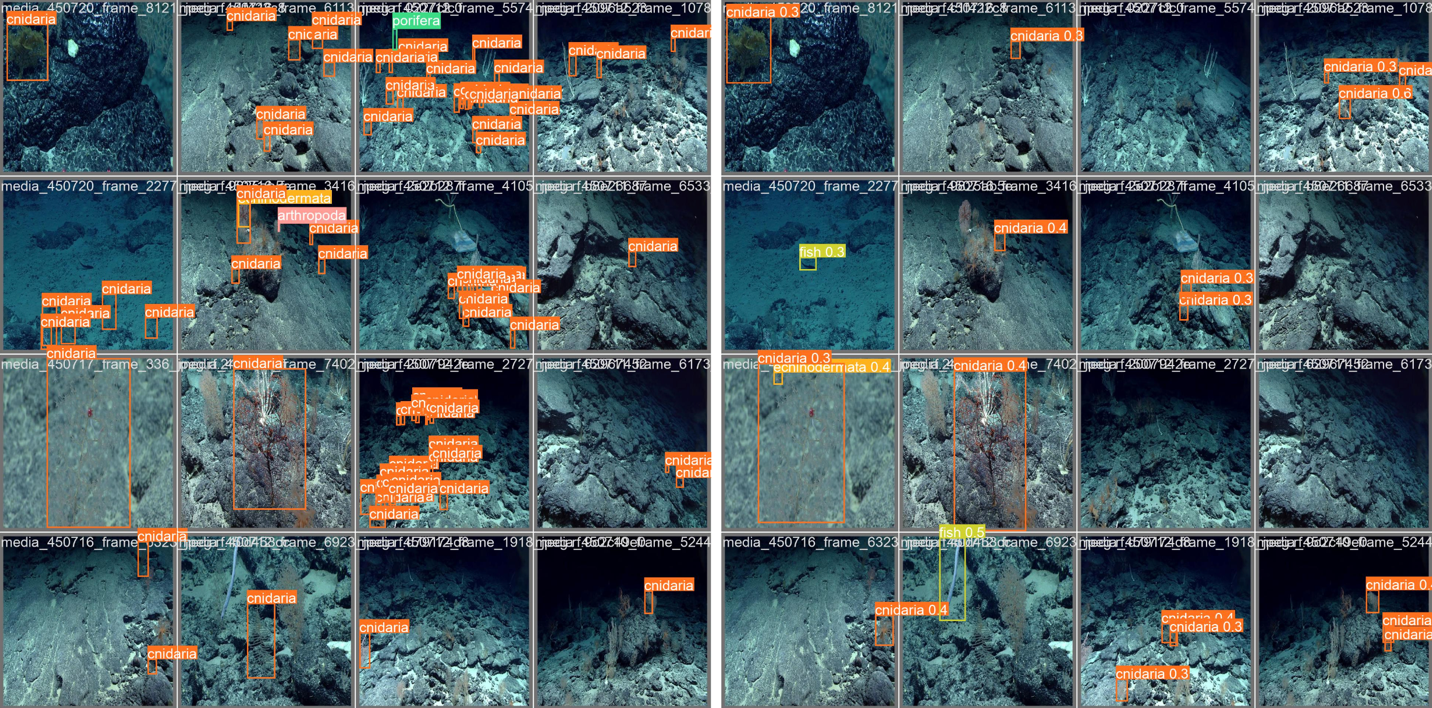
**Results**

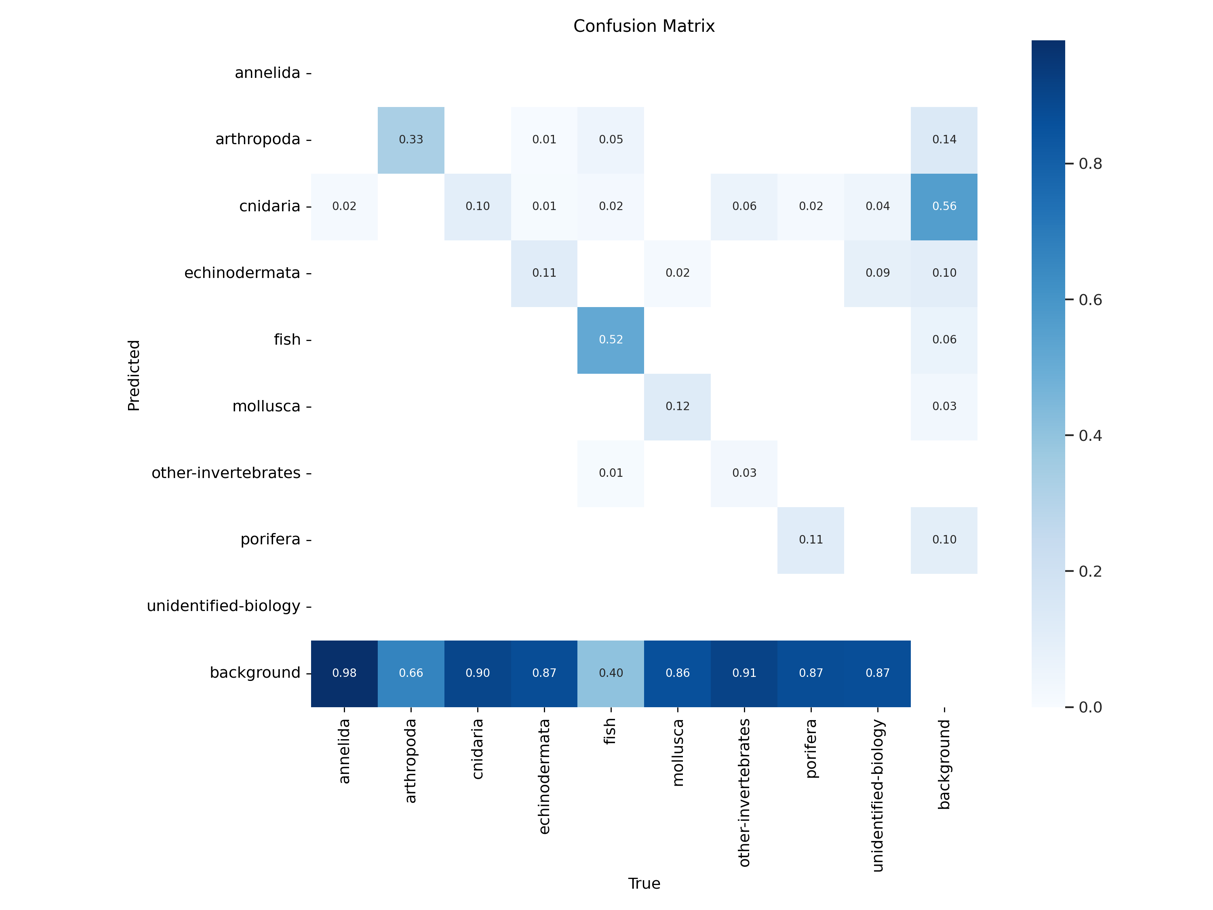
Our model performs well with detecting organisms, but generally struggles in classifying them correctly. In the future, we would try to collect even more data to improve the dataset even further to mitigate this issue. Below is a video of the model's annotations of the seafloor video:

Here are the training loss graphics for our model:



Here are the ground truth (left) v. model predictions image for one batch of images:



Finally, here is the confusion matrix showing accuracy between classes:  


**Limitations and Extensions**

One reason for the model's inaccuracy was the sheer amount of diversity present in each class. Multiple organisms that look very different were part of the same categories, something that would directly detract from model accuracy.

Of course, with more time, we would expand our dataset even further, however it is worth noting that annotating images by and finding correctly annotated images is still difficult.

**Acknowledgements and Resources**

We would like to thank MATE and NOAA Ocean Exploration for hosting the ML Challenge. We would also like to thank FathomNet and NOAA Ocean Exploration for providing data. Additionally, we would also like to thank Peyton Lee and the team from last year's UWROV deepsea-detector project for advice, providing their dataset for us to build on. In this year's project, we used a similar structure to them for documentation and the notebook to train our model, as well as the code for showing standard output to the UI.

Additionally, we want to acknowledge our use of Ultralytics Yolov5. The detection.py script is sourced from the Yolov5 repository and the other python files in the project's subfolders are from the repository as well (these are dependencies of detect.py).

**Video Demo and Explanation**

**Citations**

Jocher, Glenn. "yolov5." GitHub, 2023, https://github.com/ultralytics/yolov5

ShrimpCryptid. "deepsea-detector." GitHub, https://github.com/ShrimpCryptid/deepsea-detector

"The Main Differences Between Arthropods and Cnidarians." Biobubble Pets, https://biobubblepets.com/the-main-differences-between-arthropods-and-cnidarians