Technical Report for Little Guy
A YOLO Model Trained on the DeepFish Dataset
Ben Blechman, Ben Annicelli, Dan Sanguino

Problem Definition:
This project seeks to address the problem of underwater object detection and identification. We approached this problem utilizing a You Only Look Once (YOLO) model. Primarily, we seek to accurately detect the presence of fish within a simulated marine environment and form bounding boxes around them. We then will have our autonomous vehicle track them utilizing the various pieces of hardware installed on it.

Data Processing:
Initially, we created a custom handler in order to handle the processing of our dataset. It began by collecting the images and labels within the dataset, and separating out the negative samples so they can be labeled as such. It then organized the labels so that the YOLO model could interpret them correctly. To mitigate overfitting, various transforms were applied to introduce random changes to the images. Finally, the dataset was split into training, testing and validation sets and loaders for our model to utilize.
After realizing that our model was insufficient for our task, we decided to use a pretrained YOLO model which does not require any processing outside of loading the data into the model using the yolo() function.
The dataset we trained on is called the DeepFish dataset, which contains 40000 RGB images of fish in various marine environments around the coasts of Australia.

Model Architecture and Training:
Our YOLO model employs a series of convolutional layers for feature extraction from the input images. The initial two layers increase channel size from sixteen to thirty two and have a kernel size of three with a padding of one. Each convolutional layer is then followed by a batch normalization layer to improve performance and stability. To introduce non-linearity, we use leaky-ReLU as we found the model makes more accurate predictions with it. Following activation, the results are sent through a pooling layer and then into the next convolutional layer. This is done twice before sending through a third layer that does not pool and a fourth that does not normalize before flattening to a fully connected layer and then finally through one more ReLU layer, applying dropout of thirty percent and then through one last fully connected layer.

Within the first iteration of the model, there were many false positives due to lack of training and lack of complexity (only 3 convolutional layers). Then the 2nd version, with 6 convolutional layers, was significantly better at determining if a fish was in frame but

was bad at placing bounding boxes. This meant it had high accuracy but low IOU. In the end, we decided to switch to a pretrained YOLO V11 trained on our deepfish dataset.

As for training, we use ten epochs and split our data further into batches within the trainloader for processing. Within the forward pass, the inputs are passed through the model, where the labels are processed to separate bounding box coordinates and confidence scores and to create the targets. The confidence scores are then added to the targets and the loss is calculated. Then the model backpropagates and uses the calculated gradients from that backpropagation to optimize the model's parameters. Finally, the running loss is calculated and the model moves on to the next batch.

Weights and Biases Tracking Insights:
In our first few tracked runs, we found that our model was reporting a validation loss of 80 million. This is obviously incorrect. We also tracked training loss, F1 score, precision, and recall. The values of these insights were also beyond the range of what is reasonable or expected. After switching and training, the metrics returned the following:
Validation Loss: 0.003
Training Loss: 0.003
Recall: 0.016
Precision: 0.6
F1: 0.032

Evaluation Metrics and Analysis:
Our evaluation metrics include the above from Weights and Biases as well as our observation of the vehicle's aptitude in finding and following any fish it observes. Our initial testing on the robot found that it struggled to follow any fish it found because it moved too quickly to keep anything in frame. We solved this problem by implementing a new system to make the bot hold in place until it finds something it deems a fish instead of rotating constantly.

User Interface Design and Usability Considerations:
We used a Raspberry PI with PI OS, where we uploaded our model and ran a script to make inferences with said model. It sends serial data over to the Elgoo Uno R3 (an Arduino Uno R3 clone) when a fish is found in frame which then directs the motors to turn in order to direct the vehicle towards the fish. When in "search" mode, the vehicle previously rotated clockwise continuously until a fish was found but not sits and waits until one is found so it does not leave the frame. Within the code, the following directions are encoded: F for forward, S for stop, L for left, R for right and B for backwards. When it finds a fish, it attempts to center the fish in frame, then moves towards it.

Future Improvements:
We would like the camera to scan back and forth using servo motors instead of rotating the entire vehicle continuously as that uses less power. We would also like to add an on-startup function on the pi script instead of having to manually start the model every startup. Finally, we would like to implement our own working model, having fixed the issues we ran into before switching to the pretrained version.