Marine Life Detector
Computer Systems Research Laboratory
Eshi Kohli
2021-2022

INTRODUCTION

The Marine Life Detector is a user interface for detecting marine life in the Oceanography and Geophysical Systems Lab; specifically, Clownfish, Orchid Dottyback, Zebrafish, Rainbowfish, and Yellow Tang. In the interface, users can upload images or videos for detection and view live detection from a computer webcam stream.

Fig. 1 Clownfish

Fig. 2 Rainbowfish

Fig. 3 Zebrafish

Fig. 4 Orchid Dottyback

Fig. 5 Yellow Tang

The inspiration for my project came from work I did last summer in MIT Lincoln Labs' Beaver Works Autonomous Underwater Vehicle or AUV challenge. In this challenge, I had to develop an algorithm where I could autonomously navigate an AUV through red and green buoy gates. To do this I used image processing and detection to identify the positions of the red and green buoy gates, and then created a sequence to adjust the angle of the AUV's rudder and speed accordingly. After this summer research, I wondered what the real life applications of AUVs could be, because I became interested in AUVs and wanted to expand from the MIT project.

I looked into how AUVs could be used to help with marine environmental issues, such as identifying plastic in oceans or detecting illegal fishing. During my time at MIT, I also got to learn about Autonomous drones, and while I was researching, I saw how they were being used to monitor endangered species. However, I did not see much about AUVs monitoring endangered marine animals, only stationary cameras monitoring marine life.

I had initially aimed to connect an Underwater Remotely Operated Vehicle which came with a camera built in with it to connect the video stream to the live stream portion of my user interface. This is why I chose to work with species that were readily available in the Oceanography Lab instead of endangered species. I was planning on using a CHASING Dory

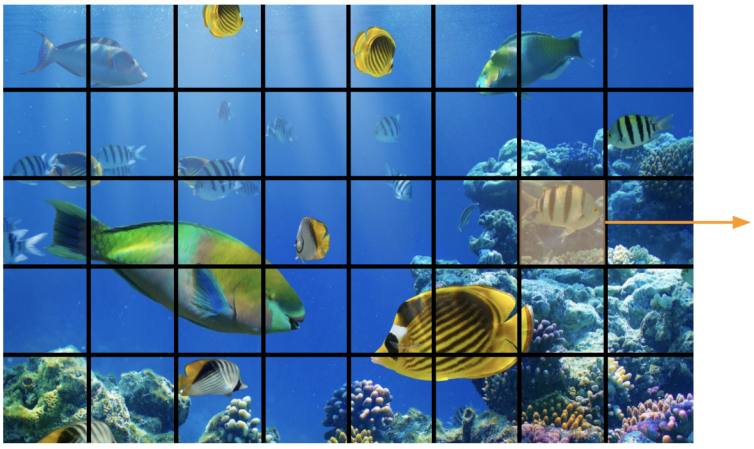
Fig. 6 CHASING Dory Underwater Drone

Underwater Drone (Fig. 6) which had an in-built camera. I would have taken the IP address of the camera from the CHASING Dory App, and input it into OpenCV's "VideoCapture" method to integrate the stream with Tkinter. I wanted to use the CHASING Dory vehicle because it was also small in size (7.4 x 3.6 x 9.7 inches), which was perfect to fit into the Oceanography Lab tanks.

METHODS

Logistics of Detection Model

Dr. Gabor introduced me to You Only Look Once (YOLO). I decided to use version 4 of YOLO (YOLOv4) instead of the most recent one, version 5 of YOLO (YOLOv5), because YOLOv4 used the same YOLO neural network framework previous versions had used before, Darknet, which made it easier to edit the framework of the neural network compared to the Ultralytics framework which YOLOv5 uses. Also, YOLOv4 and YOLOv5 were on par with each other in terms of accuracy after training and YOLOv4 allowed for more storage while training.



	b_x	b_y	b_h	b_w	c_1	c_n
box 1						
box 2						
box 3						
box 4						
box 5						

Fig. 7 Visual Representation of the vector assigned to every cell in the grid in YOLO

I read and understood the basic theory of how YOLOv4 functions. YOLOv4 first takes an image as an input and then divides the image into a grid. This allows for image classification to be applied to every image on the grid. Forward and backpropagation are used on a convolutional neural network and Max Pool layers on each cell in the grid, and assigns a vector to each cell which contains the predictions for five bounding boxes (Fig. 7). In the vector, every predicted bounding box is given at least five variables: two variables for the x and y coordinates of the center of the bounding box, two variables for the height and width, and N variables (where N is equal to the number of classes) for the confidence level that the Nth class is present. In the context of Figure 6, if the only two classes in this model were "Fish" and "Plants", then the vector would have 6 variables, and the variable "c1" would most likely be assigned a high

confidence level because the bounding box contains a fish, and the variable "c2" would most likely have an extremely low confidence percentage or even a 0 because there are no plants.

Now there are multiple bounding boxes for one object, so then an algorithm called Non-Maximum Suppression comes into play to determine which bounding box is the most accurate and encompasses the object the best by comparison. For the first step, the bounding box with the highest probability is chosen. Then, the Intersection over Union of every remaining box with the first chosen box is calculated. Intersection over Union calculates the overlapping area between the bounding box with the highest probability and the chosen comparison bounding box, divided by the combined area of those two bounding boxes. If the Intersection over Union is high (above ~0.5), the box of comparison will be removed or better described as "suppressed."

Creating the Initial Detection Model

Initially, I wanted to train for fish in general instead of specific species in the Oceanography Lab, so that I could determine if using YOLO was the right path for my project. I used Google Collaboratory because YOLOv4 needs GPU to run. I utilized The AI Guy's tutorial "YOLOv4 in the CLOUD: Install and Run Object Detector (FREE GPU)" from May 20, 2020 totry to use YOLOv4's in-built detection model, which is trained for more than 80 classes. First, I had to clone the Github repository of AlexeyAB's YOLOv4 darknet code. I also made sure I was using GPU while running instead of CPU so that the image detection would run a lot faster. I checked a file in the repository which contained a list of all of the classes the model was trained for, and I saw "Fish" was one of them. AlexeyAB's repository also had a file named darknet.py, which had all the functions used to implement YOLOv4. This included loading the YOLOv4 network, image detection, and setting the bounding boxes.

To test if I correctly implemented YOLOv4, I ran my YOLOv4 code on the YOLO open-source images, which would result in high accuracy of classification. Then, I wanted to test different images of fish underwater. From the repository, there was a file that had a list of all of the classes, and for marine life, I saw there were options such as goldfish, anemone fish, starfish, etc. Therefore, I decided to pick images that fell into those classes. However, the images I tested for were not classified as marine life. For example, a picture of two goldfish was identified as birds, and multiple anemone fish were classified as kites. I wondered if the list I checked earlier with the classes were all the possible YOLOv4 classifications instead of the classes that were trained with the weights I was using. I found another file in the repository that separately listed which classes were being used in the trained model, and it did not include marine life.

I then built and trained my own object detector with YOLOv4. I used another one of The AI Guy's tutorials to figure out how to curate a dataset. In "YOLOv4 in the CLOUD: Build and Train Custom Object Detector (FREE GPU)" from June 29, 2020, the video creator suggested to use Google's Open Images Dataset and use the OIDv4_Toolkit from one of his repositories to extract all the images in the format required by YOLOv4 Darknet. Google's Open Images

Dataset has hundreds of classes available to choose from on this site, including the general class "Fish", and the images are already labeled. Since I was making this detection model to see if I should continue to use YOLO, I only took 50 images for my training dataset and 15 images for the validation dataset because the validation set is usually 20% to 30% of the training dataset. I changed the format of the images to fit the YOLOv4 vector format associated with each image by using the "cv2" library in Python, I retrieved the shape of the image, and using the "os" library in Python, I accessed and edited the classes text file, which stores the list of all the classes included in the training set.

Then, I uploaded the image datasets to my Google Drive to access them in my Google Colab notebook. Before running the command to train the data, I realized I needed to deal with the issue that Google Colab disconnects after 90 minutes of being idle, and I knew training the dataset would take at least a couple of hours. I thought of using a program to fake activity on the notebook, and I found a JavaScript program that accomplished that. In the "Console" from clicking "Inspect Element" on Google Colab, the program would fake a mouse click every ten minutes. This method worked, and I could leave my code running for an extended period of time. However, it didn't work for more than 3 hours at a time, and I also had the obstacle of only getting access to Google Colab's GPU for 12 hours a day. The Pro version of Google Colab allows for more time.

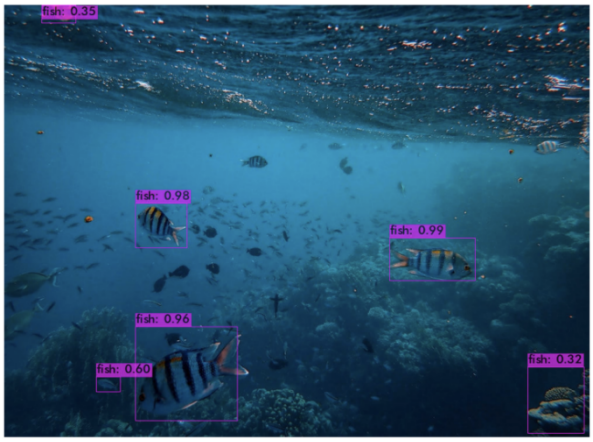


Fig. 8 Results from training YOLOv4 for class "fish"

To make the convolutional neural network accustomed to one class, I had to edit the configuration variables. I created a "config" text file in Google Drive, which is called during the training method. In this file, I changed the variables in the three layers of the network. For the activation function, I used Mish over ReLu, because I read from multiple sources that Mish is more reliable and accurate for training.Then, I changed the "max_batches" value to be accustomed to the number of classes I was training for, which is recommended to be 6000 if training for 3 classes or lower. The steps decide when to adjust the learning rate, so I set it to 80% and 90% of the max_batches value, 4800 and 5400, respectively. This meant that after every 4800 and 5400 steps, the learning rate would be modified. Finally, for each layer, I adjusted the "filters" value, also known as the convolution kernel, to 16. I would have a higher filter value if I had more than one class. It took about 2 hours to train and went through about 1000 epochs. After that, I inputted images and classified fish with a mean average precision of 0.72.

Training for Model Accustomed to Oceanography Lab

I decided to train my new YOLOv4 model for five species from the Oceanography Lab - Clownfish, Orchid Dottyback, Yellow Tang, Rainbowfish, and Zebrafish. Google's Open Images Dataset does not have those classes, so I had to acquire my pictures from Google Images. First, I tried a method to mass download the images using a Python module called fast.ai. This gathered the URLs of all of the images I wanted using a JavaScript command, and input those URLs into the "dowload_images" method in fast.ai. I downloaded 150 images per species, making a total of 750 images. I used around 75% to train the data, almost 20% to validate, and assigned less than 10% of the images for testing. After training with those images, testing gave no detection output. I realized that all the images were not matching the set resolution standard which is required by the object detection algorithm I was using, which is 416 by 416 pixels. I learned that if the images were smaller than that resolution, then training would be unsuccessful. Then, I manually downloaded all the pictures from Google Images. I used Roboflow to annotate the images because Roboflow is really simple to use and it has a feature which lets me export the data into YOLO Darknet format. This meant that it resized every image to 416 by 416, and generated a text file for each image which contains the class number and coordinates of the bounding boxes. This made it easier for me to go directly to training my data.

Before training I had to change my configuration file which I had used for one class to fit for three classes. The standard value for "max_batches" is the (number of classes) * 2,000 for 4 classes or more. Therefore, my "max_batches" value was now 10,000. Since the steps are usually 80% and 90% of the "max_batches", I changed the "steps" numbers to 8000 and 9000. The 'filters' equation is (the number of classes + 5) * 3, so I adjusted the filter variable to 30.

Roboflow provides a Google Colab notebook for train data with YOLOv4, however, I advise against using it because even though it supposedly saves the most recently calculated weights after every 100th epoch into the path "build\darknet\x64\backup\", it does not work. After Google Colab would timeout and interrupt training, if I tried picking up training with the last saved weights file, then it would give an error saying that it was empty. Using the same tutorial I mentioned earlier from the Youtube creator The AI Guy, I saved the weights calculated after every 100th epoch in a file in my Google Drive. This time when I tried picking up after the training got interrupted, it worked. Buying Google Colab Pro might have saved me some trouble, but I am not sure if that is completely true since Google Colab would timeout even though I had only used it for three out of the limit of twelve hours because of GPU usage.

Creating the User Interface

I used the Tkinter in Python to create my user interface, and I organized it so that videos and images can be uploaded and displayed for detection on the left side and live video stream is on the right.

For the left side, I made a text label to let the user know that the left side was for inputting images for fish detection. Beneath the title, I placed a button that would open the computer's files so that the user can select an image they have. I did this by using the filedialog module in Tkinter. Then, I took the path of the file selected and used the Python Imaging Library (PIL) to open the image. To display video, I used OpenCV's VideoCapture method, and entered the path to the video from the file selection as an input to the method. Then OpenCV's "readNetFromDarknet" and "DetectionModel" methods are used to run my YOLOv4 method onto the image or frame captured from the video. The readNetFromDarknet method allowed me to load my detection model's configuration and weights files. The configuration file contains all of the settings of the neural network, like the number of classes I was training for, after how many steps the network would recalculate its weights, the learning rate of the model, and more. The DetectionModel method ran the network on whichever image was uploaded from the filedialog module. It also returns the confidence levels and the labels of the bounding boxes. I took that information and added it to the original image/video, and I also used that information to print the number of fish visible for uploaded images and videos. Using Python's File I/O, I added the option for the user to download the data returned from the DetectinoModel method.

```
img = cv2.imread(filename)

classIds, scores, boxes = model.detect(img, confThreshold=0.6, nmsThreshold=0.4)

for (classId, score, box) in zip(classIds, scores, boxes):
    cv2.rectangle(img, (box[0], box[1]), (box[0] + box[2], box[1] + box[3]),
                  color=(0, 255, 0), thickness=2)

    text = '%s: %.2f' % (classes[classId], score)
    cv2.putText(img, text, (box[0], box[1] - 5), cv2.FONT_HERSHEY_SIMPLEX, 1,
                color=(0, 255, 0), thickness=2)
```

Fig. 9 Running the DetectModel method and adding bounding boxes, class IDs, and scores to the inputted image/frame

For the right side, I needed to incorporate live video streaming. Similar to the left side of the interface, I created a text label to let the user know that this section was to show live video for fish detection. Using OpenCV's VideoCapture method again, I entered in "1" instead of a path to a video to access my computer's webcam stream. YOLOv4 on the livestream worked the same as it did on pre-recorded videos, by capturing a frame and then running the model on it.

To make the interface more user friendly, I made the buttons into access keys. For example, the button which allows the user to choose one of their files for object detection, I bound the keys 'control' and 'f' to my object detection method. This is the same command which I connected to the button.

RESULTS

After almost 40 hours of training, my YOLOv4 model had a mean average precision of 0.83. Fish that are in the foreground of images are detected with high accuracy, but if fish overlap then the model has some difficulty with accurately identifying the fish. This is a well known problem with any YOLO model.

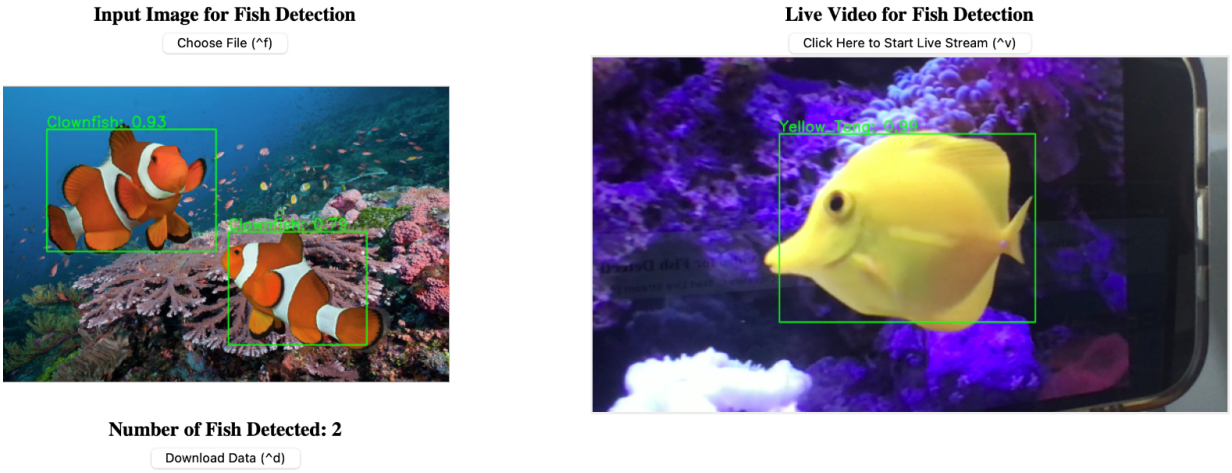


Fig. 10 Example of user interface

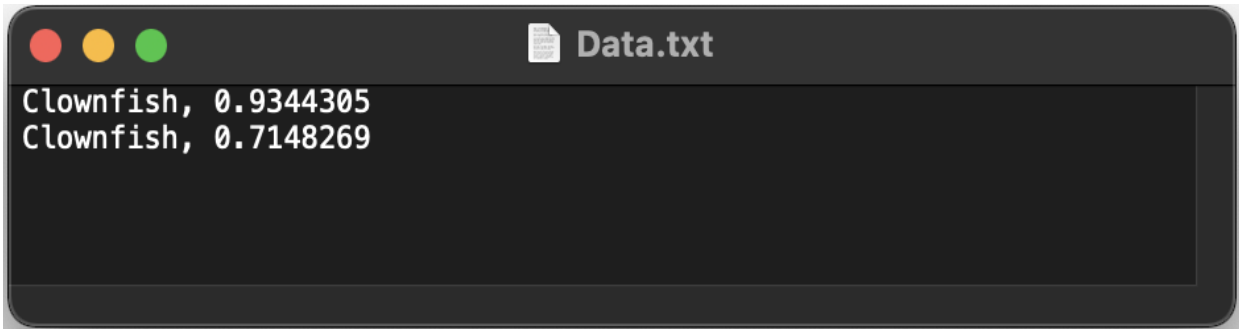


Fig. 11 Example of the text file generated in the downloads folder after clicking the "Download Data" button in the user interface with the image of Clownfish in Fig. 10

Fig. 12 Detected Rainbowfish

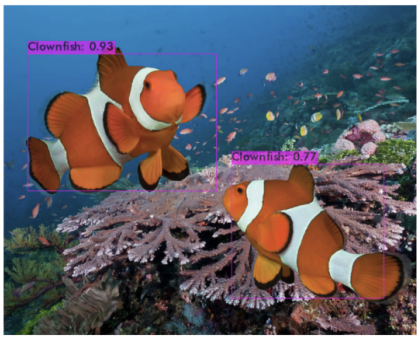


Fig. 13 Detected Clownfish

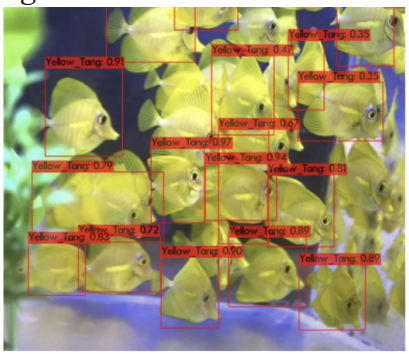


Fig. 14 Detected Yellow Tang

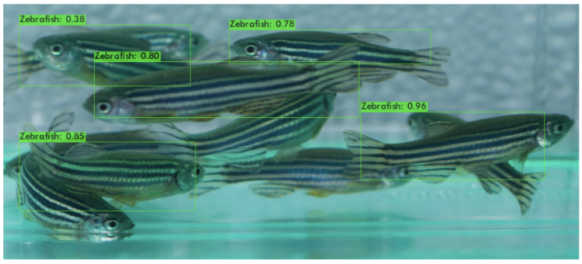


Fig. 15 Detected Zebrafish

Fig. 16 Detected Orchid Dottyback

References

AlexeyAB. (n.d.). Wikipedia, the free encyclopedia. Retrieved May 30, 2022, from https://github.com/AlexeyAB/darknet

Joseph, L. (2020, May 1). *A Gentle Introduction to YOLO v4 for Object detection in Ubuntu 20.04*. Robocademy. Retrieved May 30, 2022, from https://robocademy.com/2020/05/01/a-gentle-introduction-to-yolo-v4-for-object-detection-in-ubuntu-20-04/

Presser, S. (2020, June 12). *Responding to the Controversy about YOLOv5*. Roboflow Blog. Retrieved May 30, 2022, from https://blog.roboflow.com/yolov4-versus-yolov5/

Script to Stop Google Colab From Disconnecting. (2021, September 30). rockyourcode. Retrieved May 30, 2022, from https://www.rockyourcode.com/script-to-stop-google-colab-from-disconnecting/

Solawetz, J., Nelson, J., & Sahoo, S. (2020, May 21). *How to Train YOLOv4 on a Custom Dataset*. Roboflow Blog. Retrieved May 30, 2022, from https://blog.roboflow.com/training-yolov4-on-a-custom-dataset/

The AI Guy. (2020, May 20). *YOLOv4 in the CLOUD: Install and Run Object Detector (FREE GPU)*. YouTube. Retrieved May 30, 2022, from https://youtu.be/mKAEGSxwOAY

The AI Guy. (2020, June 29). *YOLOv4 in the CLOUD: Build and Train Custom Object Detector (FREE GPU)*. YouTube. Retrieved May 30, 2022, from https://youtu.be/mmj3nxGT2YQ

Wright, L. (2019, Aug 27). *Meet Mish — New State of the Art AI Activation Function. The successor to ReLU?* Less Wright. Retrieved May 30, 2022, from https://lessw.medium.com/meet-mish-new-state-of-the-art-ai-activation-function-the-successor-to-relu-846a6d93471f