(Established under Karnataka Act No. 16 of 2013) 100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Dissertation on

"Detection and Tracking of Marine Life and Fish Abundance Using Computer Vision"

Submitted by

George J Malayil(01FB16EEC101)

Jayant L(01FB16EEC372)

JAN - MAY 2020

Under the Guidance of

Prof. Swetha R

Department of Electronics and Communication Engineering

PES University

Bengaluru

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRONICS AND COMMUNICATION
BTECH IN ECE

CERTIFICATE

This is to certify that the Dissertation entitled

"Detection and Tracking of Marine Life and Fish Abundance Using Computer Vision"

Is a Bonafide work carried out by

George J Malayil (01FB16EEC101)
Jayanth L (01FB16EEC372)

In partial fulfillment for the completion of 8 semester course work in the Program of Study B.Tech in Electronics and Communication Engineering under rules and regulations of PES University, Bengaluru during the period Jan – May. 2020. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The report has been approved as it satisfies the 8 th semester academic requirements in respect of project work.

Signature with date & Seal	*Signature with date & Seal*	*Signature with date & Seal*
Internal Guide	*Chairperson*	*Dean of Faculty*

Names of the students

George J Malayil (01FB16EEC101)

Jayanth L (01FB16EEC372)

Name of the examiners:

1.

2.

3.

DECLARATION

We, **George J Malayil** and **Jayanth L,** hereby declare that the report entitled, '**Detection and Tracking of Marine Life and Fish Abundance Using Computer Vision',** is an original work done by us under the guidance of **Prof. Swetha R**, Assistant Professor, PES University, and is being submitted in partial fulfillment of the requirements for completion of 8th Semester course work in the Program of Study B.Tech in Electronics and Communication Engineering

PLACE:

DATE:

George J Malayil
(01FB16EEC101)

Jayanth L
(01FB16EEC372)

ACKNOWLEDGMENT

We extend our gratitude to our chairman **Dr. M.R.Doreswamy** (Founder, Chancellor – PES University), **Prof. Jawahar Doreswamy,** Pro-Chancellor of PES University and **Dr. J Suryaprasad**, the Vice Chancellor of PES University for giving me an opportunity to be a student of this reputed institution.

We thank our Principal **Dr. K.S. Sridhar** for his valuable support and giving us students such an opportunity.

It is also our privilege to thank Chairperson of the Department **Dr. Anuradha M.**, Department of Electronics and Communication Engineering, for her support and guidance in doing our Project.

We express our sincere gratitude to our guide **Prof. Swetha R** for her valuable guidance and technical help.

We also express our gratitude to all my faculty members and parents who helped us carry out this work.

ABSTRACT

Video observatories used by fisheries and coast guards are used for sampling of species at frequencies and durations that are not possible with other methods. It is cost efficient and can be used in most environments. The aim of this project is to trace fish abundance in marine environments using image recognition with computer vision and Machine Learning. With the assistance of this software, the real world implementations can be realised using drones or submersible cameras attached to boats.

After referring to multiple papers and projects, we had come to a conclusion that it was feasible with the right object detection model. Further we were able to determine that YOLOv3 and Faster RCNN were the best performing algorithms for the specific applications we had in mind. We aim to obtain high precision and real time implementation of object detection.

CONTENTS

ABSTRACT

1 INTRODUCTION 1

1.1 BACKGROUND 1

1.2 MOTIVATION / PROBLEM STATEMENT 1

1.3 Hurdles to overcome

2 LITERATURE SURVEY 2

2.1 Paper[1] 2

2.1.1 Conclusions drawn 3

2.2 Paper[2] 4

2.2.1 Conclusions drawn 4

3 METHODOLOGY ADOPTED 5

3.1 Dataset 6

3.2 Basic Terminology 6

3.3 CNN 7

3.3.1 Convolutional Layer 7

3.3.2 Max Pooling Layer 8

3.3.3 ReIU(Rectified Linear Unit) Layer 9

3.3.4 Weights 10

3.3.5 Fully Connected Layers 10

3.4 Software Environment 11

4 RCNN 12

5 Faster RCNN 14

5.1 Code Used 15

6 YOLOv3 20

6.1 Method 20

6.2 Code used for video implementation 21

7 Results 25
7.1 RCNN 25
7.2 YOLOv3 27
8 Performance 28
9 Conclusion and Future Scope 30
10 References 31

TABLE OF FIGURES

1.	Diagrams explaining Harr cascade classifier(2)	2,3
2.	Graph From Paper 2-Comparisons of precision	5
3.	Flowchart of Methodology	6
4.	Diagram Depicting basic terms	7
5.	Diagrams showing convolution operation in CNN (3)	8,9,10
6.	Flowchart depicting RCNN	12
7.	Flowchart depicting Faster RCNN	14
8.	Code and outputs for faster RCNN	15-19
9.	Code and outputs for YOLOv3	21-24
10.	Results for RCNN	25
11.	Results for YOLOv3	27

1. Introduction

1.1 BACKGROUND

Underwater imagery is being used to provide fisheries data to support stock assessments for living marine resources. Automation of underwater video observatories is needed to reduce cost of analysis and increase accuracy for fisheries, coast guards as well as marine biologists.

1.2 MOTIVATION / PROBLEM STATEMENT

The motivation behind this project is to find a feasible way to detect and track abundance of fish underwater. With the help of this project, real world implementation could be realised using drones or submersible cameras attached to boats for fisheries and marine biologists. This would also be a non- invasive sampling method, one of the few of its kind.

1.3 Hurdles to overcome

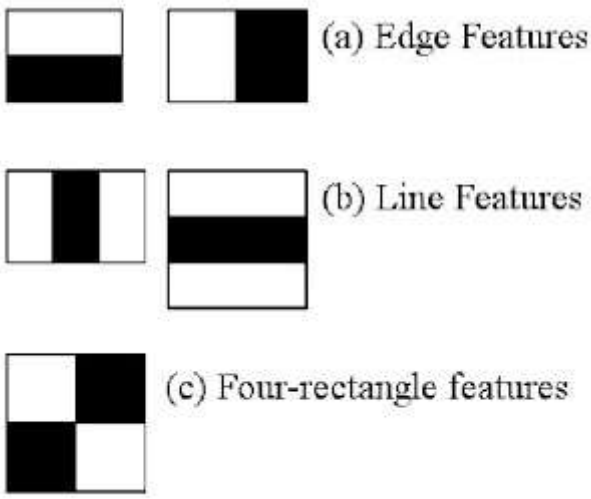
Underwater imagery presents challenges that include changes in light, low contrast, floating vegetation, and changes in visibility due to turbidity. However recent developments in neural networks research and computer vision has enabled us to overcome these challenges.

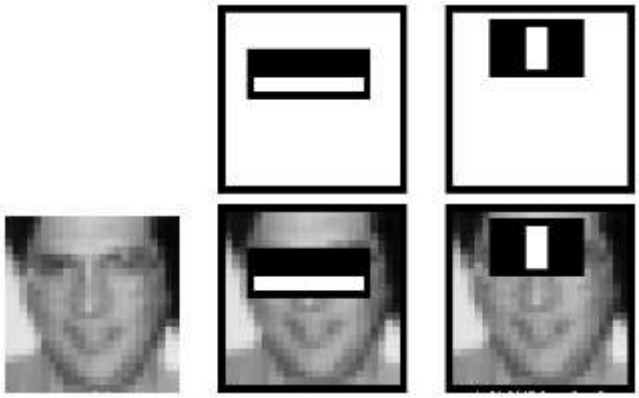
2. LITERATURE SURVEY

2.1 Paper [1] *Cutter, G.; Stierhoff, K.; Zeng, J. 2015 paper on "Automated detection of rockfish in unconstrained underwater videos using Haar cascades and a new image dataset: labeled fishes in the wild," IEEE Winter Conference on Applications of Computer Vision Workshops*

This paper uses Haar cascade classifiers on underwater images of fish to extract features and classify the fish objects.

It is a neural network training algorithm. The algorithm is trained using positive and negative images and then features are extracted. Haar features as shown below were used. A value is obtained using the formula X= (values of pixels under the white rectangle added) - (values of the pixels under the black rectangle added) for all features.





We apply every feature on all the training images and it extracts , for each feature, the best threshold value that can classify the faces. The process continues till accuracy required is achieved or the required number of features are found. An image 24x24 window is taken and 6000 features are applied to it.

Several Haar cascades were developed and applied to test for video images for evaluation. Detection rates of 63-89% were achieved for seven classifiers of fish.

2.1.1 Conclusions drawn

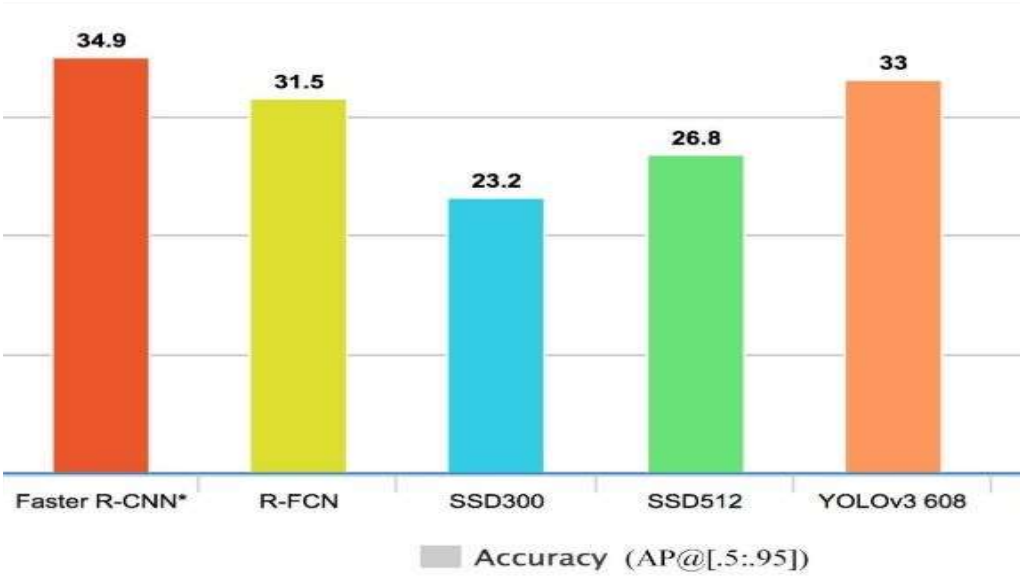
This paper was one of the first to implement image processing to underwater fish detection. The haar classifiers used were able to obtain 60-80% accuracy. The National Oceanic and Atmospheric Administration implemented this in some fisheries but were not effective.There was a need for a faster and more accurate training model and with such a model, this is a feasible endeavour.

2.2 Paper [2] *'Research on accuracy of various models' paper by Jonathan Hui on Medium.com*

In this paper, multiple training models were tested for speed and accuracy for different datasets as well as performance environments.

The performance of various training models were tested and compared for accuracy and speed. The graph in the following image shows the performance using a metric known as mean average precision (mAP). It tells us how much of the region of interest intersects with the ground truth.

Mean Average Precision
Precision= (TP/TP+FP)
TP=True Positive
FP=False Positive



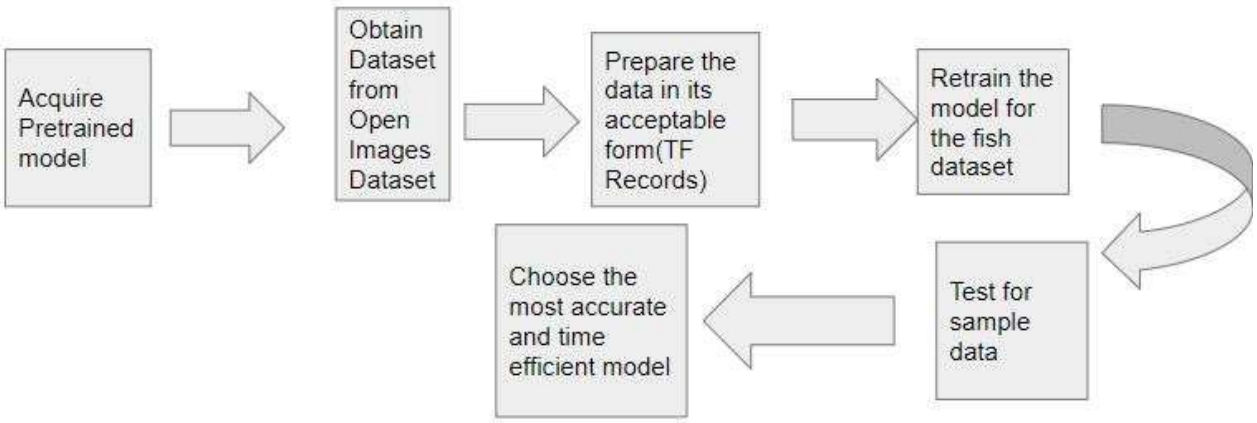
2.2.2 Conclusions drawn

The paper's results show that Faster RCNN and YOLO were the best performing training algorithms

3. METHODOLOGY ADOPTED

From the above papers we inferred that the project was feasible and that Faster RCNN was the best performing model.

- We acquired a Faster RCNN Model that was pretrained
- We obtained Dataset from Open Images dataset
- In order to train a tensorflow model, we need to prepare the data in its acceptable form, which are TF Records from the .CSV format
- Retrained the Model for the given dataset
- For real time video implementation YOLOv3 was used



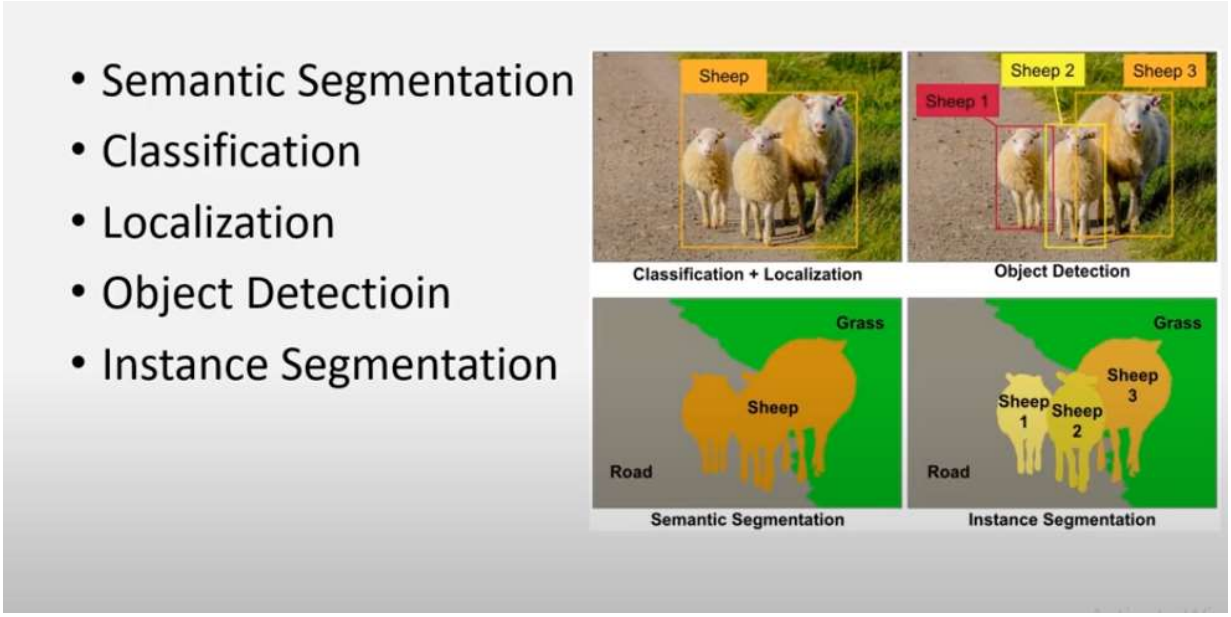
3.1 Dataset

- Acquired from Open Images Dataset by cvd foundation

  https://github.com/cvdfoundation/open-images-dataset#download-images-with-bounding-boxes-annotations

- Used to train the RCNN Model

3.2 Basic Terminology



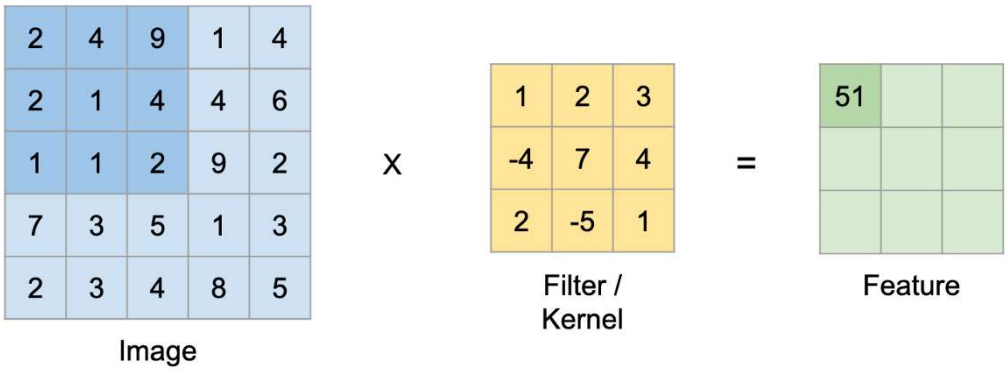
- **Semantic Segmentation : Trying to identify to which broad class each pixel belongs**
- **Classification: Grouping pixels into identifiable classes**
- **Localisation/Object detection: Forming a bounding box in the region around an object**
- **Instance segmentation: Combination of semantic and object detection**

3.3 Convolutional Neural Networks (CNN)

We use a Convolutional Neural Network in each of the mentioned training algorithms at some stage. Hence a brief introduction into CNN is required. The input to a CNN is a tensor. After the image goes through the convolutional layer, it then becomes a feature map.

A CNN is a class of neural networks that is mostly used for image detection and uses Deep Learning. A CNN consists of filters/kernels that pass through the image in square pixel amounts and computes a dot product and each filter is used to extract a feature.
It is split into multiple layers as follows.

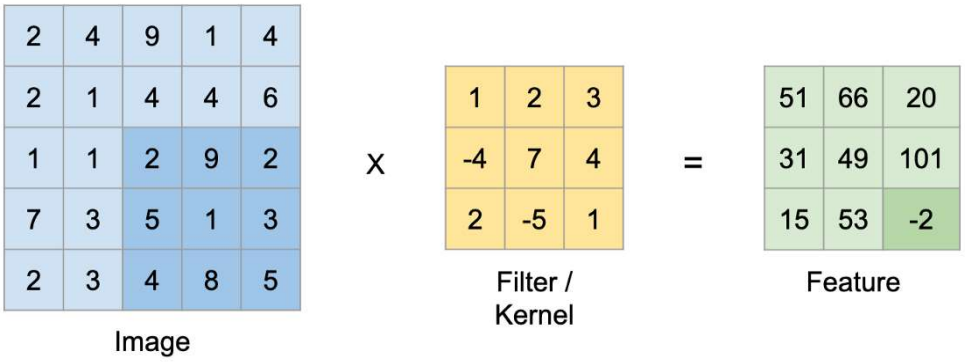
3.3.1 Convolutional Layer



The kernels pass through the image in parts which are a set amount of pixels and slide to the next part based on the Stride value; that is the number of pixels to move by. This process repeats for each part of the image till a map for the feature is obtained for the image.

The equation to calculate size of the feature is:

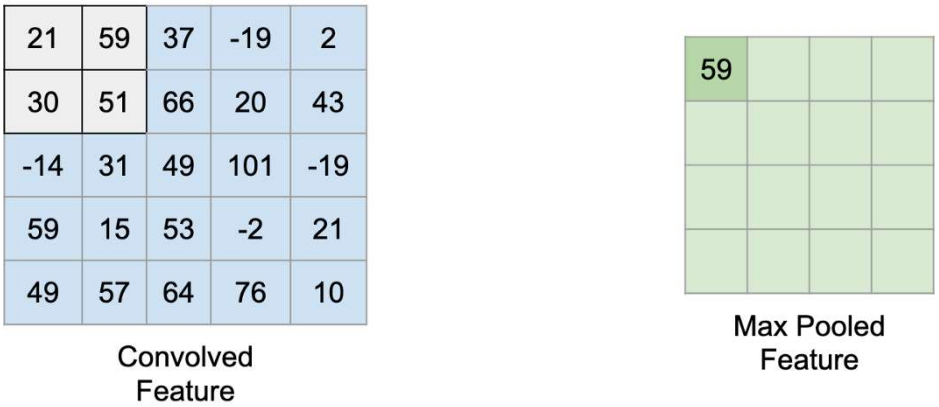
Feature Size= ((Image size − Kernel size) / Stride)+1

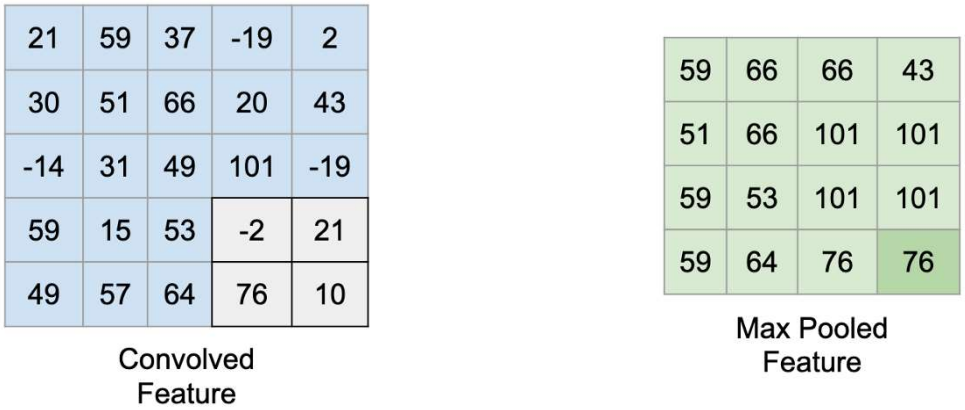


The above image shows a completed feature map. A neural network learns those kernel values through back propagation to extract different features of the image. Usually, a convolutional neural network would have more than 1 kernel at each layer. We can use the feature maps to perform tasks including classification, segmentation, object detection.

3.3.2 Max Pooling Layer

Max pooling layer helps reduce the spatial size of the convolved features obtained and also helps reduce overfitting by providing an abstracted representation of them. The process is best explained through diagrammatic representation. It basically takes the max value for a group of pixels.





Max pooling reduces noise by discarding noisy activations and is better than average pooling.

3.3.3 ReIU(Rectified Linear Unit) Layer

RelU function is a piecewise linear function that outputs the input directly if it is positive, otherwise, it will output zero. This is to remove all false outlier pixel values.

The image goes through these layers repeatedly until one final value is obtained for the feature which then passes to the fully connected layer. Every time it passes the convolutional layer it becomes more filtered and each time it passes the Max Pooling layer it becomes smaller in size.

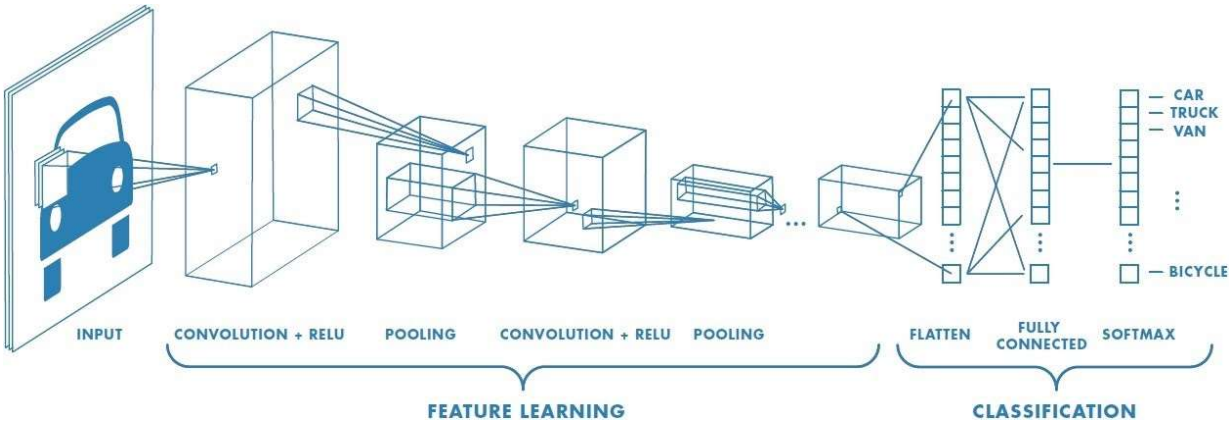
3.3.4 Weights

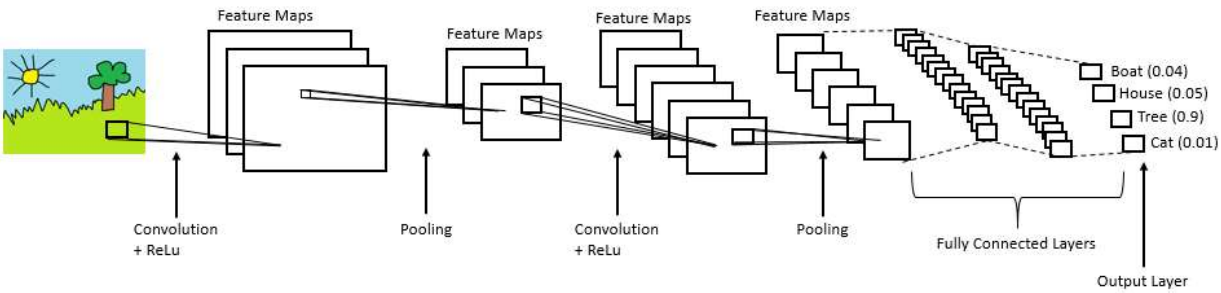
All neurons in a neural network have a value as output by applying a function to input values coming from the layer previous to the current one. The function is established by a vector of weights and also a bias is determined. Training is done by making changes to the bias values and weights.

These functions are also called filters and are used to extract particular features of the input .

3.3.5 Fully connected layers

The features with singular values then pass onto the fully connected layers. In the Fully connected layer, every neuron in one layer is connected to every neuron in another layer. The matrix of values goes through a fully connected layer to classify the images. We use one or more fully connected layers at the end of a CNN.

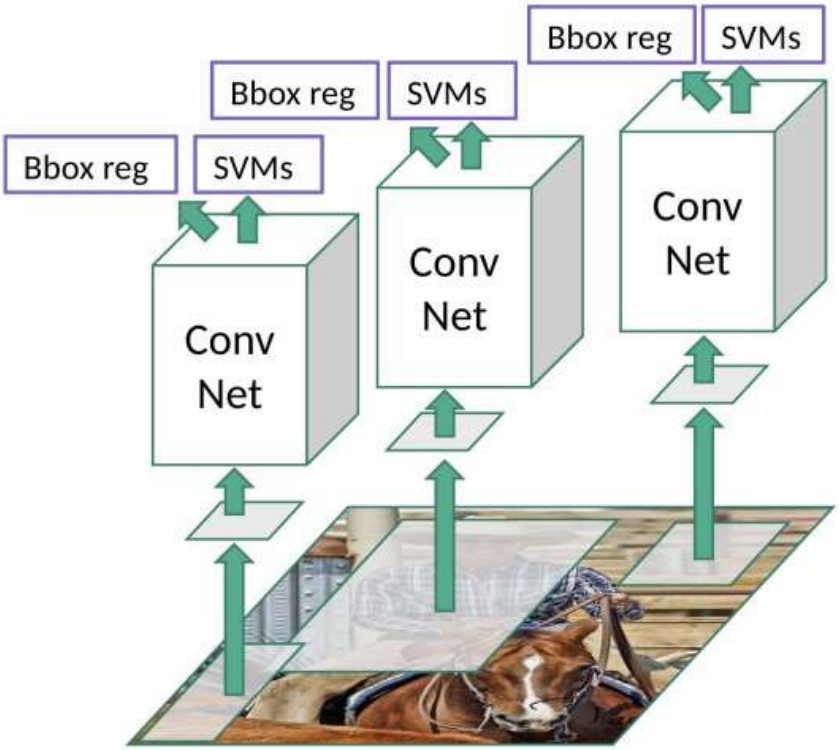




3.4 Software Environment

- Python
- Github
- TensorFlow API
- Open images Dataset
- Pretrained COCO Model (faster_rcnn_inception_resnet_v2_atrous_coco)
- Google Collab

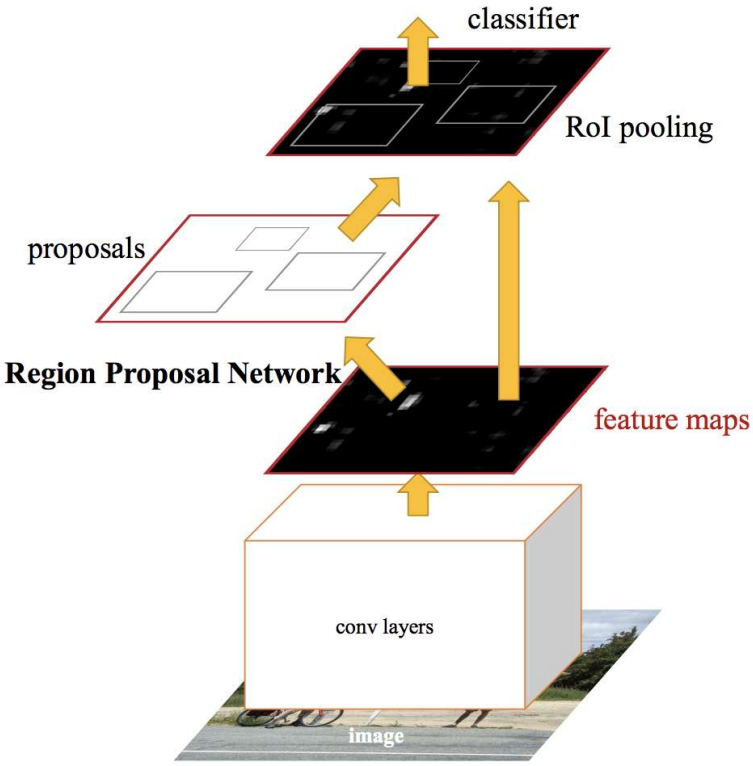
4. RCNN



- In Object Detection algorithms , bounding boxes are drawn around the regions to locate an object.
- The issue with this method is different spatial locations an object might be in and might have different aspect ratios.. Hence we need to take multiple regions of all sizes.
- A method was developed where selective search was used to extract a fixed number of regions from the image. This was called RCNN.

- 2000 region proposals are input into a CNN that outputs a 4096-dimensional feature vector.
- The features are then passed to an SVM(Support Vector Machine) to classify the objects in the image.
- It filters images using IoU(Intersection over Union) values which tells the amount of intersection a region has with the actual object, if the IoU value is larger than the trained value it rejects the region.
- The algorithm predicts four bounding box values[Bbox reg].

5. Faster RCNN



- We input images to the CNN only once instead of sending 2000 regions to output a convolutional feature map.
- From the feature map, the algorithm identifies the regions containing objects by using RoI (region of interest) pooling layer and reshapes them into a size that can be fed into a fully connected layer.

- The training time as well as execution time is reduced dramatically. It is one of the few algorithms that can work in real time at 0.2 seconds per frame

5.1 Code used for image implementation

The code was executed using Google colab. Below is the colab document

https://colab.research.google.com/drive/1zhFtafWL-kshLtkQHcXwSFPI4IwxNcVN

```
#importing pretrained model
!git clone https://github.com/Jayanth-L/fish_detection.git
!git clone https://github.com/tensorflow/models.git
!cd models/research
!protoc object_detection/protos/*.proto
!export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

```
Cloning into 'fish_detection'...
remote: Enumerating objects: 250, done.
remote: Total 250 (delta 0), reused 0 (delta 0), pack-reused 250
Receiving objects: 100% (250/250), 231.34 MiB | 36.07 MiB/s, done.
Resolving deltas: 100% (132/132), done.
Checking out files: 100% (46/46), done.
Cloning into 'models'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 34584 (delta 2), reused 10 (delta 0), pack-reused 345
Receiving objects: 100% (34584/34584), 512.62 MiB | 37.28 MiB/s, do
Resolving deltas: 100% (22359/22359), done.
Checking out files: 100% (2497/2497), done.
Missing output directives.
```

```
[2] %%bash
    cd models/research/
    protoc object_detection/protos/*.proto --python_out=.
```

```
[3] !protoc object_detection/protos/*.proto --python_out=.
```

```
object_detection/protos/*.proto: No such file or directory
```

```
[4] !export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

```
[5] !pip install cython pycocotools --user
    #installing prerequisites
```

```
Requirement already satisfied: cython in /usr/local/lib/python3.6/dist-packages (0.29.17)
Requirement already satisfied: pycocotools in /usr/local/lib/python3.6/dist-packages (2.0.0)
```

```
[6] import os
    import pathlib


    if "models" in pathlib.Path.cwd().parts:
      while "models" in pathlib.Path.cwd().parts:
        os.chdir('..')
    elif not pathlib.Path('models').exists():
      !git clone --depth 1 https://github.com/tensorflow/models
```

```
[7] cd models/research
```

```
/content/models/research
```

```
[8] import numpy as np
    import os
    import six.moves.urllib as urllib
    import sys
    import tarfile
    import tensorflow as tf
    import zipfile

    import matplotlib
    from collections import defaultdict
    from io import StringIO
    from matplotlib import pyplot as plt
    from PIL import Image
    from IPython.display import display
```

```
[9] from object_detection.utils import ops as utils_ops
    from object_detection.utils import label_map_util
    from object_detection.utils import visualization_utils as vis_util
```

```
[10] # patch tf1 into `utils.ops`
     utils_ops.tf = tf.compat.v1

     # Patch the location of gfile
     tf.gfile = tf.io.gfile
```

```
[11] def load_model(model_name):
       base_url = 'http://download.tensorflow.org/models/object_detection/'
       model_file = model_name + '.tar.gz'
       model_dir = tf.keras.utils.get_file(
         fname=model_name,
         origin=base_url + model_file,
         untar=True)

       model_dir = pathlib.Path("/content/fish_detection/fish_inception_v2_graph2/saved_model")
       #model_dir = pathlib.Path(model_dir)/"saved_model"

       model = tf.saved_model.load(str(model_dir))
       model = model.signatures['serving_default']

       return model
```

```
[12] !cp /content/fish_detection/data/fish_label_map.pbtxt /content/models/research/object_detection/data/
```

```
[13] # List of the strings that is used to add correct label for each box.
     PATH_TO_LABELS = '/content/models/research/object_detection/data/fish_label_map.pbtxt'
     category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

```
[14] !rm /content/models/research/object_detection/test_images/*.jpg
     !cp /content/fish_detection/test_images/*.jpg /content/models/research/object_detection/test_images/
```

```
[15] model_name = 'ssd_mobilenet_v1_coco_2017_11_17'
     detection_model = load_model(model_name)
```

```
Downloading data from http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_co
76537856/76534733 [==============================] - 1s 0us/step
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
```

```
[16] print(detection_model.inputs)
```

```
[<tf.Tensor 'image_tensor:0' shape=(None, None, None, 3) dtype=uint8>]
```

```
[17] detection_model.output_dtypes
```

```
{'detection_boxes': tf.float32,
 'detection_classes': tf.float32,
 'detection_scores': tf.float32,
 'num_detections': tf.float32,
 'raw_detection_boxes': tf.float32,
 'raw_detection_scores': tf.float32}
```

```
[18] detection_model.output_shapes
```

```
{'detection_boxes': TensorShape([None, 300, 4]),
 'detection_classes': TensorShape([None, 300]),
 'detection_scores': TensorShape([None, 300]),
 'num_detections': TensorShape([None]),
 'raw_detection_boxes': TensorShape([None, 300, 4]),
```

```
[19] def run_inference_for_single_image(model, image):
       image = np.asarray(image)
       # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
       input_tensor = tf.convert_to_tensor(image)
       # The model expects a batch of images, so add an axis with `tf.newaxis`.
       input_tensor = input_tensor[tf.newaxis,...]

       # Run inference
       output_dict = model(input_tensor)

       # All outputs are batches tensors.
       # Convert to numpy arrays, and take index [0] to remove the batch dimension.
       # We're only interested in the first num_detections.
       num_detections = int(output_dict.pop('num_detections'))
       output_dict = {key:value[0, :num_detections].numpy()
                      for key,value in output_dict.items()}
       output_dict['num_detections'] = num_detections

       # detection_classes should be ints.
       output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

       # Handle models with masks:
       if 'detection_masks' in output_dict:
         # Reframe the the bbox mask to the image size.
```

```
         # Reframe the the bbox mask to the image size.
         detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
                   output_dict['detection_masks'], output_dict['detection_boxes'],
                    image.shape[0], image.shape[1])
         detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                            tf.uint8)
         output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

       return output_dict
```

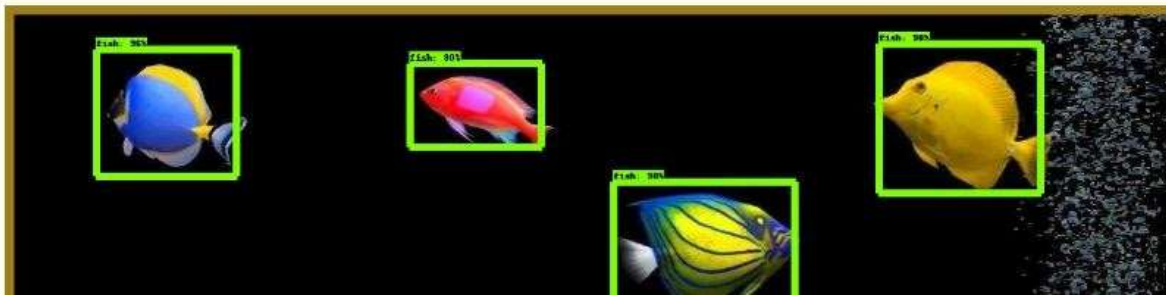
```
[20] def show_inference(model, image_path):
       # the array based representation of the image will be used later in order to prepare the
       # result image with boxes and labels on it.
       image_np = np.array(Image.open(image_path))
       # Actual detection.
       output_dict = run_inference_for_single_image(model, image_np)
       # Visualization of the results of a detection.
       vis_util.visualize_boxes_and_labels_on_image_array(
           image_np,
           output_dict['detection_boxes'],
           output_dict['detection_classes'],
           output_dict['detection_scores'],
           category_index,
           instance_masks=output_dict.get('detection_masks_reframed', None),
```

```
[20]        instance_masks=output_dict.get('detection_masks_reframed', None),
            use_normalized_coordinates=True,
            line_thickness=8)

      display(Image.fromarray(image_np))
```

```
# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = pathlib.Path('/content/models/research/object_detection/test_images')
TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))
TEST_IMAGE_PATHS
```

```
[PosixPath('/content/models/research/object_detection/test_images/fish1.jpg'),
 PosixPath('/content/models/research/object_detection/test_images/fish2.jpg'),
 PosixPath('/content/models/research/object_detection/test_images/fish3.jpg')]
```

```
[22] for image_path in TEST_IMAGE_PATHS:
       show_inference(detection_model, image_path)
```



6.YOLOv3

You only look once (YOLO) is a new age, real-time object detection system. We implemented this algorithm to detect fishes real-time for video recordings obtained from fisheries. We had obtained video samples from NOAA (National Oceanic and Atmospheric Administration).

This algorithm uses a different approach to others, the network divides the entire image into multiple regions and then predicts bounding boxes and probabilities for all regions. These bounding boxes are "weighted" by the predicted probabilities. YOLOv3 is the third version released of its type with improvements made to the network. It is similar to faster RCNN but YOLO does both detection and classification at once. Instead of selecting parts of an image, the algorithm predicts classes and bounding boxes for the whole image; while RCNN and Faster RCNN are based on classification and selects regions of interest instead of the whole image.
Because RCNN selects multiple ROIs it might go through the same part multiple times and is hence slower than YOLO, however it is more accurate and precise than YOLO.
This also makes YOLO viable for real time object detection.

6.1 Method

- We Installed all the prerequisites and darknet onto google drive.
- For every execution on google collab we transferred it to virtual storage for faster usage.
- We trained the network using Open images dataset by downloading it using OIDtookit that pre-labels the dataset for the YOLO format. Instead of downloading the entire dataset we using the "FISH" keyword
- We also added our own images and labelled it using open labelling toolkit as the existing dataset was not adequate.

- Trained the network for the dataset using two classes for around 2000 images.
- We had to update config files for the network for the given classes and adjust the number of iterations and also resolution settings.
- The training was done on a virtual environment, google colab, and took around 3 hours to complete.
- The code was then executed for sample images as well as a video

6.2 Code used for video implementation

The code was executed on Google colab. Below is the executable code after the algorithm was trained.

https://colab.research.google.com/drive/1MzPteh6uEgLIb2Fl5UYQIHzF4cDRIq6r

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
    #importing google drive
```

```
[2] !cp -r '/content/drive/My Drive/project_folder/darknet' '/content/'
    %cd '/content/darknet'
    !chmod +x ./darknet
    #copying drive storage to virtual storage
    #darknet already installed with dataset from open images dataset
```

```
/content/darknet
```

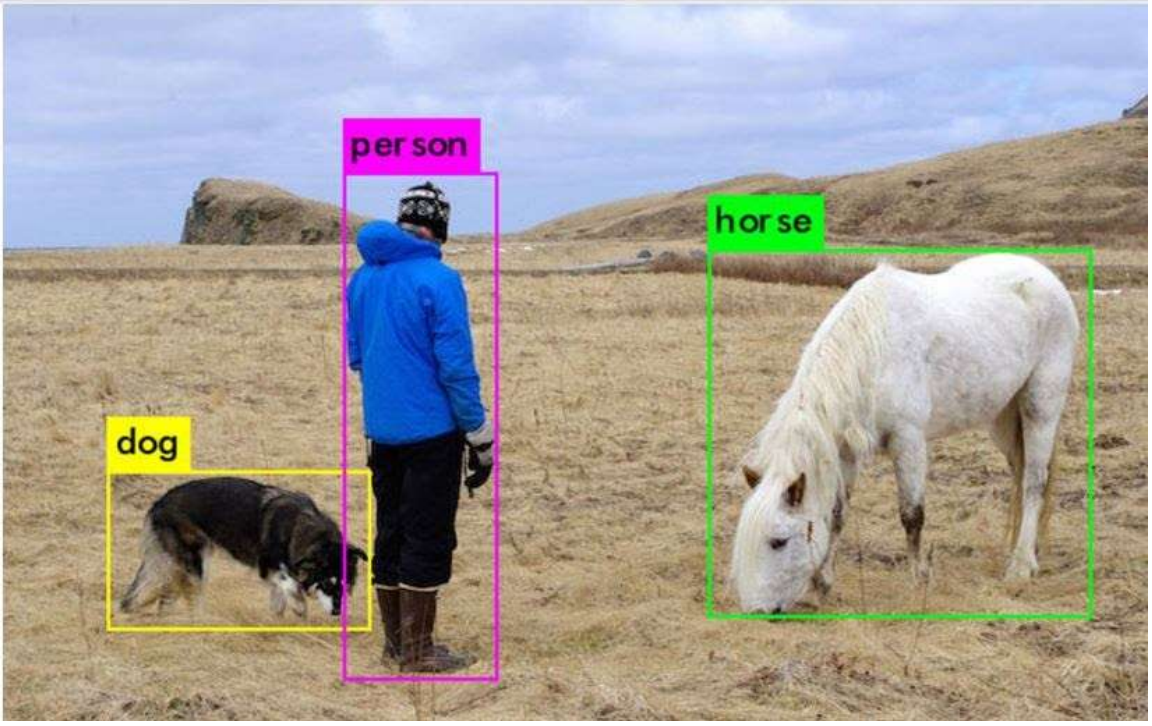
```
[3] #download files
    def imShow(path):
      import cv2
      import matplotlib.pyplot as plt
      %matplotlib inline

      image = cv2.imread(path)
      height, width = image.shape[:2]
      resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)

      fig = plt.gcf()
      fig.set_size_inches(18, 10)
      plt.axis("off")
```

```
  #plt.rcParams['figure.figsize'] = [10, 5]
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()


def upload():
  from google.colab import files
  uploaded = files.upload()
  for name, data in uploaded.items():
    with open(name, 'wb') as f:
      f.write(data)
      print ('saved file', name)
def download(path):
  from google.colab import files
  files.download(path)
```

[4]

```
# Execute darknet using YOLOv3 model with pre-trained weights to detect objects on 'person.jpg'
!./darknet detect cfg/yolov3.cfg yolov3.weights data/person.jpg -dont-show

# Show the result using the helper imgShow()
imShow('predictions.jpg')
```

[4]



```
[5] !./darknet detect cfg/yolov3.cfg yolov3.weights data/eagle.jpg
    #executing YOLOv3 implementation for and image of and eagle
```

```
[6] imShow('predictions.jpg')
    #showing outputs
```

```
%cd OIDv4_ToolKit/
```

```
/content/darknet/OIDv4_ToolKit
```

```
[8] pip install -r requirements.txt
```

```
[9] !python3 main.py
```

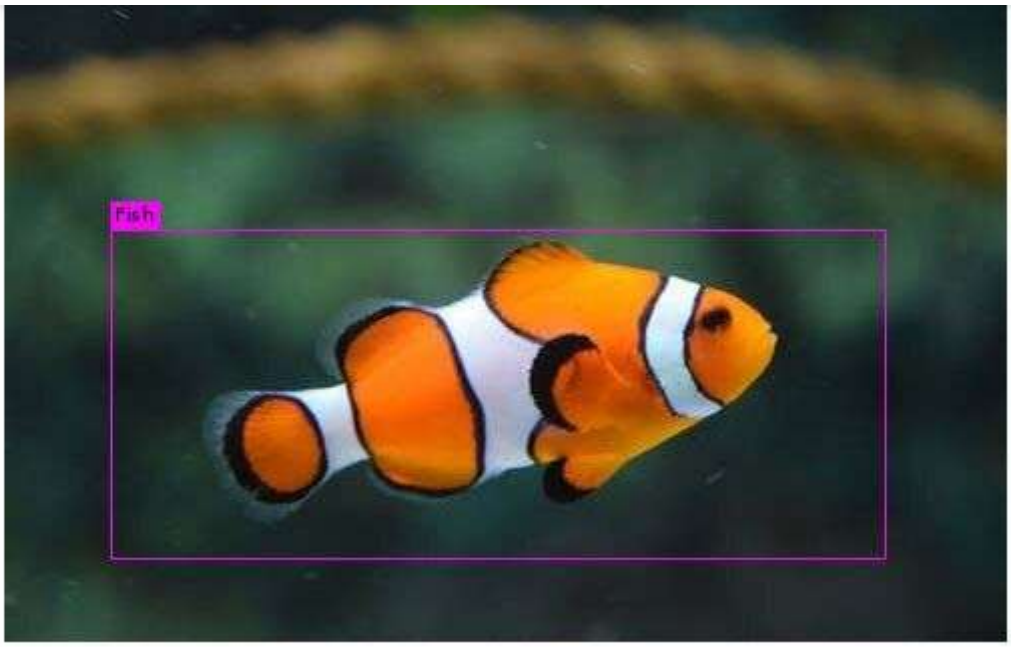
```
[10] %cd ..
```

```
/content/darknet
```

```
[11] #train
     #!./darknet detector train data/obj.data cfg/yolov3-custom.cfg backup/yolov3-custom_1000.weights -dont_show
```

```
[12] !./darknet detector test data/obj.data cfg/yolov3-custom-test.cfg backup/yolov3-custom_1000.weights fish.jpg
     from IPython.display import Image
     Image('predictions.jpg')
```

[12]




```
#Sample2
!./darknet detector test data/obj.data cfg/yolov3-custom-test.cfg backup/yolov3-custom_1000.weights fish2.jpg
from IPython.display import Image
Image('predictions.jpg')
```

```
(predictions:335): Gtk-WARNING **: 23:17:15.960: cannot open display:
```

```
[14] !./darknet detector demo data/obj.data cfg/yolov3-custom-test.cfg backup/yolov3-custom_1000.weights Test_ROV_video_h264_decim.mp4 -out_filename results.avi
     #Executing code for a video
```

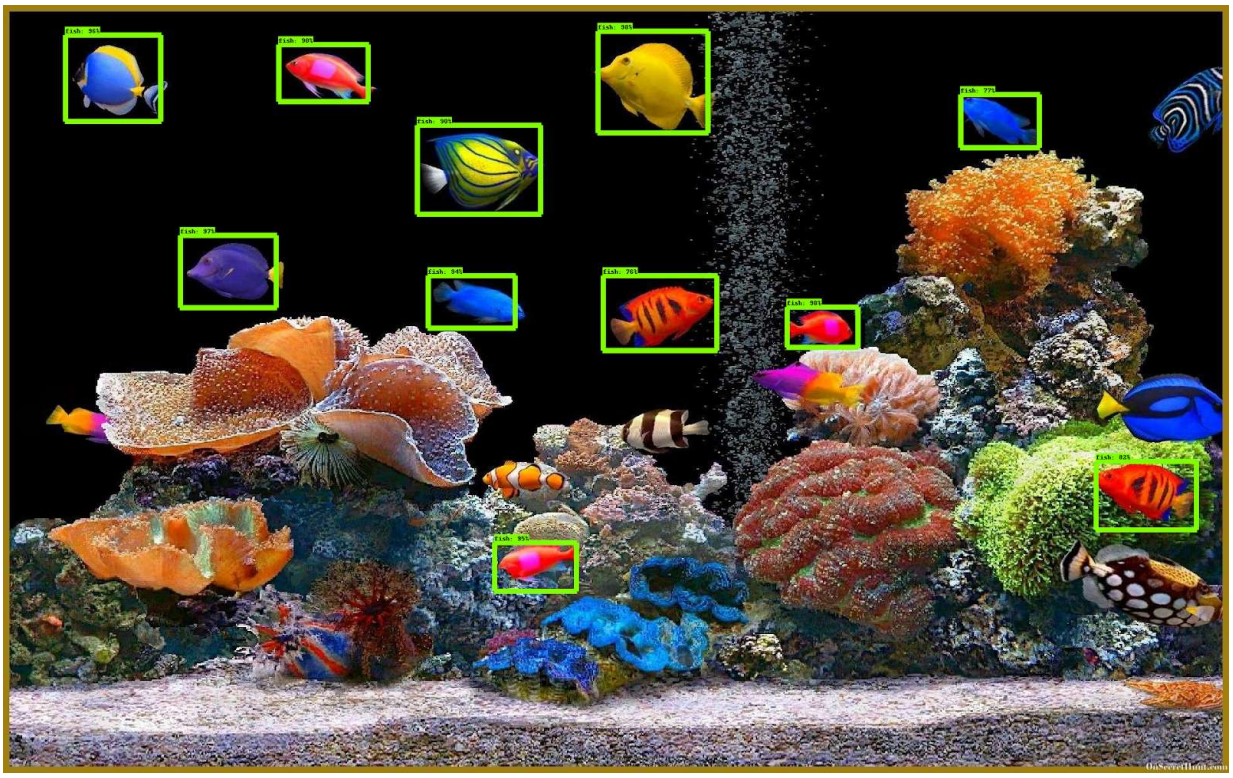
7. Results

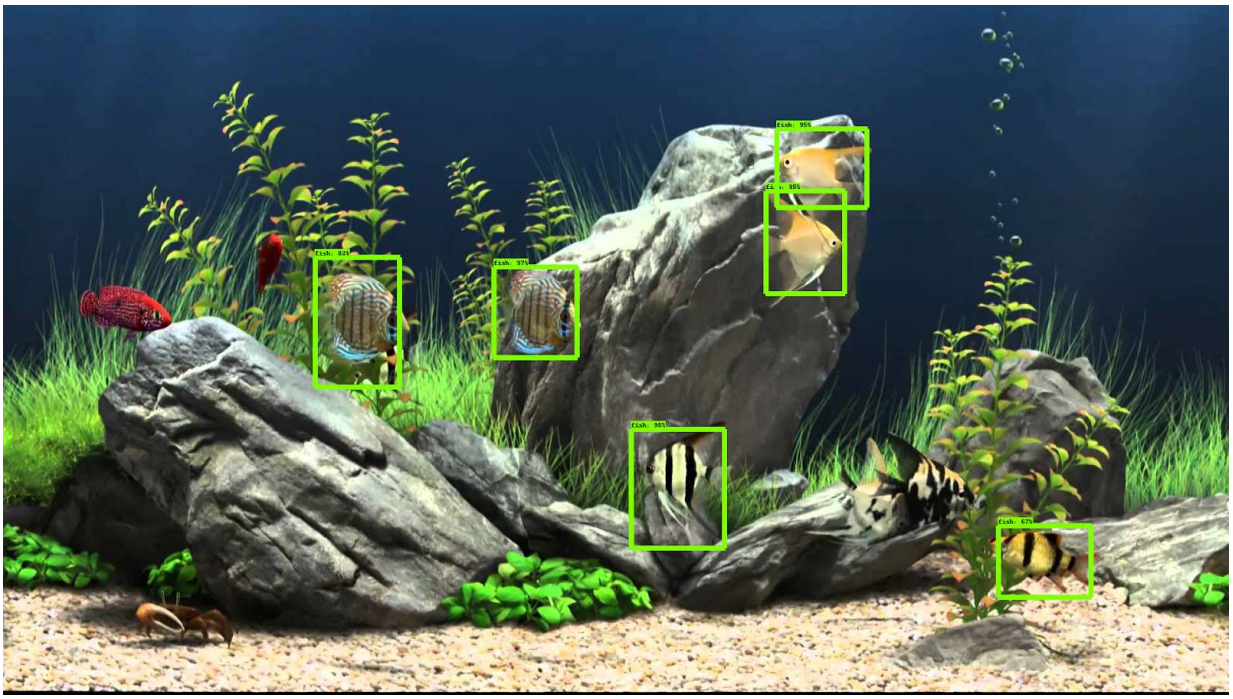
7.1

Below are results for Faster RCNN model implementations.

Sample Data

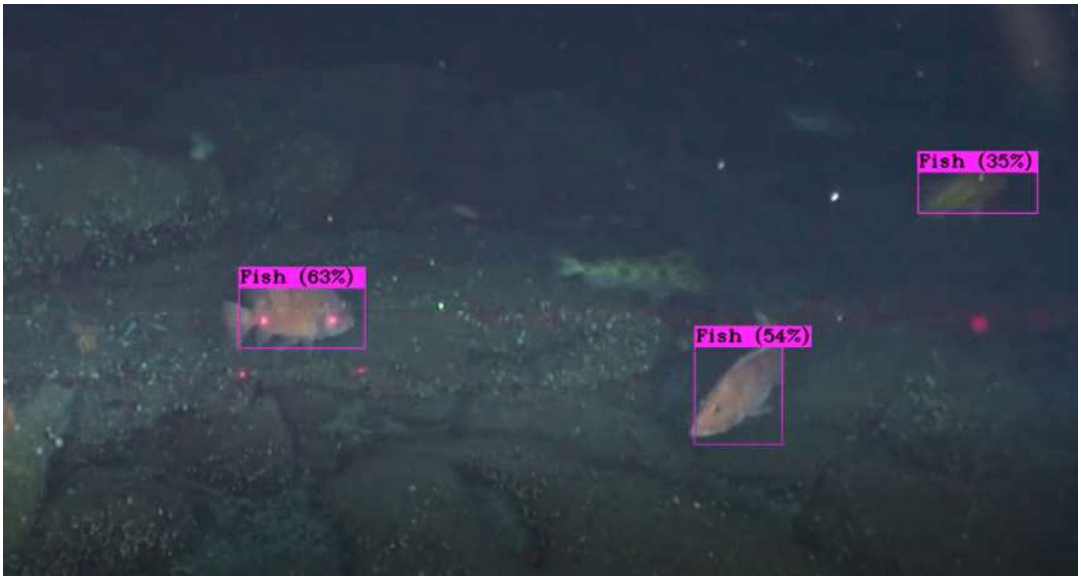
Results(Faster RCNN)





7.2

Below is a snapshot of the video implementation using YOLOv3

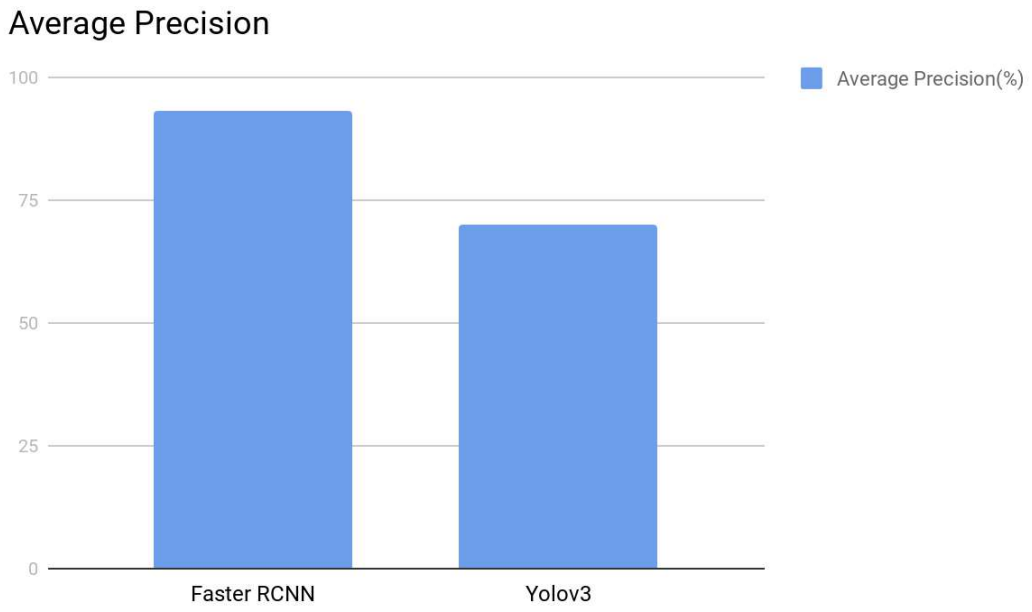


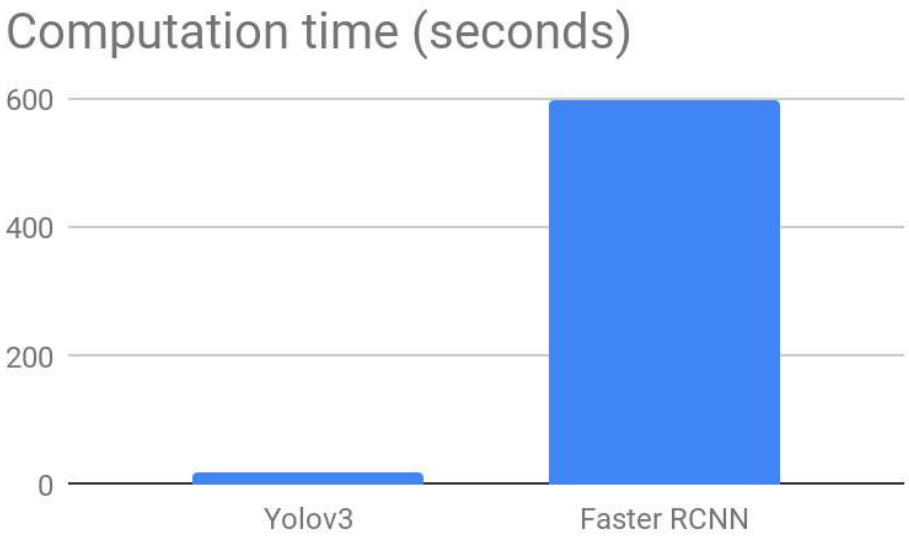
Detection of Sharks Implementation

8. Performance

Yolov3 had achieved an average precision of 70% while executing at 45 frames per second for the video implementation and 90% precision for images.
RCNN had achieved average precision of 93% and took 0.6 seconds per image on average

Adding two classes for fish and sharks for the YOLOv3 video implementation improved the precision and could be a useful application for coast guards to warn people of harmful predators. There were no false positives in the sharks implementation.
RCNN was much more precise for images but did not work real time as it was too slow.




Computation time (seconds)
600
400
200
0
Yolov3
Faster RCNN

9. CONCLUSION AND FUTURE SCOPE

We believe that this project proves that detection and tracking of fish underwater is feasible and this technology can be implemented in various fields. The faster- RCNN model was more accurate for image detection with high precision , however the YOLO implementation performed better in the video implementation, however with lower accuracy. The only limitation to furthering our accuracy is the quality of the dataset which we feel was still subpar with what can be achieved. We were able to increase accuracy using our own images with bounding boxes that we drew with the labelling toolkit.

Some of the applications could be

- Tracking abundance of fish for fisheries
- Checking biodiversity in coral reefs
- Tracking Fish migration patterns and prediction
- Alerting shores of sharks and other predators

10. References

[1]Cutter, G.; Stierhoff, K.; Zeng, J. (2015) "Automated detection of rockfish in unconstrained underwater videos using Haar cascades and a new image dataset: labeled fishes in the wild," IEEE Winter Conference on Applications of Computer Vision Workshops

[2]Research on accuracy of various models

https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[3]Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun

[4]Project on underwater fish detection on github

https://github.com/kwea123/fish_detection

[5]Dataset Acquired from Open Images Dataset by cvd foundation

https://github.com/cvdfoundation/open-images-dataset#download-images-with-bounding-boxes-annotations

[6] YOLOv3 algorithm created by the following website author

https://pjreddie.com/darknet/yolo/