# SHIP INTRUSION DETECTION SYSTEM

Group members-

1. Siddhesh Deshpande (101612)
2. Srujan Patel (101641)
3. Naeem Patel (101640)

**FCRIT, Vashi**
**29/02/20**

## Introduction

- Previous batch utilized the pre-trained weight of the Yolo algorithm in order to create the ship intrusion detection system.
- In order to be able to custom train objects and detect them, we need to develop application  which would facilitate everything.
- This involves manually creating a labelling software, Data-set split module, training module as well as the testing module.

# Object Detection

FCRIT, Vashi
29/02/20

## Object Detection and Computer Vision

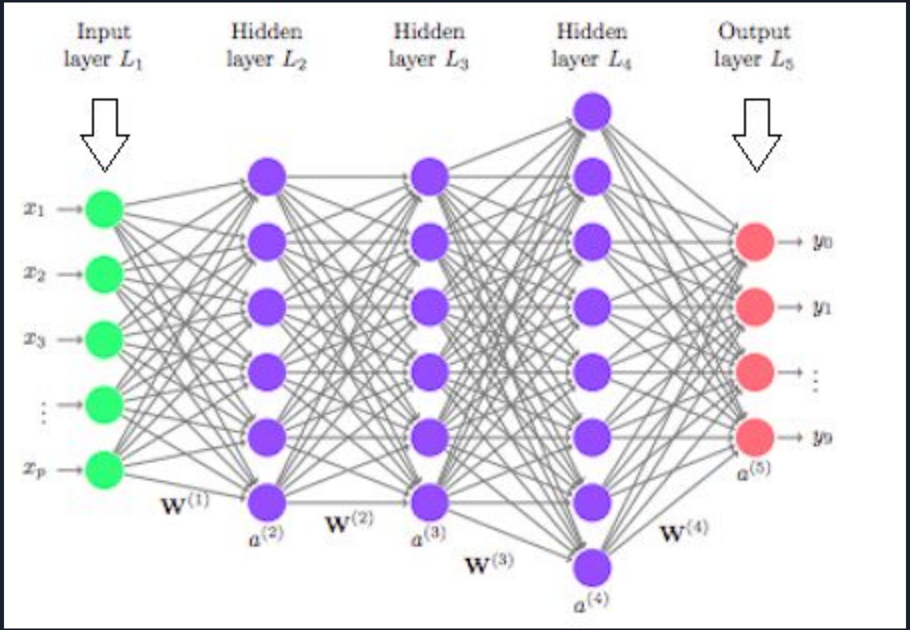
- Object detection is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos.
- From a computer vision point of view, the image is a scene consisting of objects of interest and a background represented by everything else in the image.
- Object detection and localization are two important computer vision tasks.
  - Object detection determines the presence of an object.
  - Localization determines the location of that object in the.
- The scope of this project is basically to perform object detection to detect various objects like ships etc.

**FCRIT, Vashi**
**29/02/20**

## Deep Learning

- Deep learning is a machine learning technique which teaches computers to perform activities that comes naturally to humans and by learning every day to day ventures.
- Deep learning models can achieve state-of-the-art accuracy, sometimes even surpassing human-level performance.
- Models are trained by using huge labeled data sets and neural network architectures that contain various layers.
- It uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

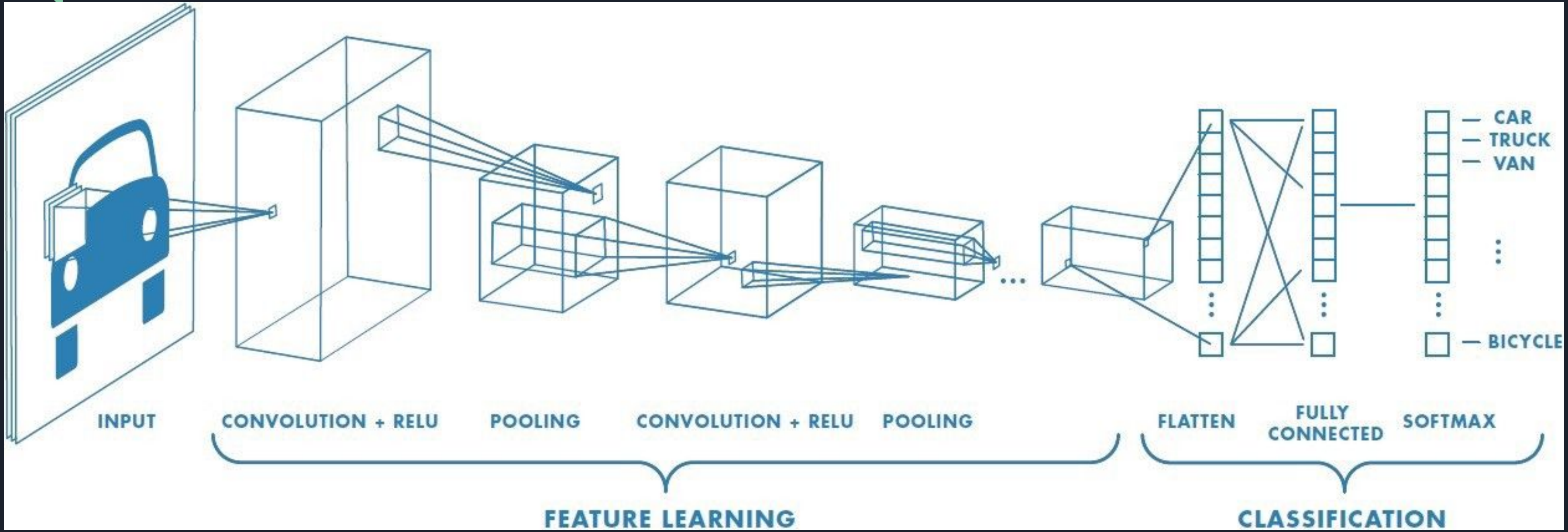
**FCRIT, Vashi**
**29/02/20**



**FCRIT, Vashi**
**29/02/20**

## Convolutional Neural Networks(CNN)

- In deep learning, a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.
- They are better for processing images as they take the spatial properties of image into consideration.



FCRIT, Vashi
29/02/20

# YOLO- You Look Only Once

- YOLO, “You Look Only Once”, is a neural network capable of detecting what is in an image and where stuff is, in one pass.
- It gives the bounding boxes around the detected objects, and it can detect multiple objects at a time.
- It uses single CNN for classification and clustering, therefore it is significantly faster than other methods.
- An image is divided into smaller parts (say using a 13x13 grid, which depends on the YOLO version) and each cell multiple bounding boxes.

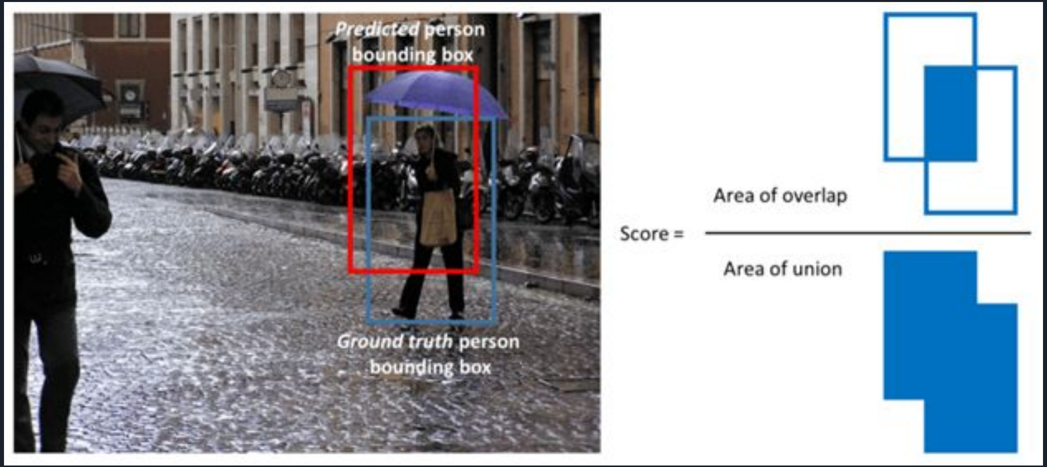
**FCRIT, Vashi**
**29/02/20**

## Yolo Algorithm

- Each bounding box has -
  - X, Y coordinates, width, height and confidence score - 5 attributes
  - Probability distribution over classes - 80 attributes
- Therefore the final shape is S x S x 255. (Where S is the grid size)

## Yolo algorithm - Intersection over union

- Used to check similarity between two bounding boxes.
- Calculates the common area between the boxes.


Predicted person bounding box
Ground truth person bounding box
Score =
Area of overlap
Area of union

**FCRIT, Vashi**
**29/02/20**

## YOLO algorithm - Non max suppression

- Used to prevent multiple prediction of same objects.
- Steps -
  - Take bounding box with highest prediction probability, based on a certain threshold.
  - Calculate IOU with other boxes, and eliminate boxes with high IOU, based on a certain threshold.
  - Take the bounding box with next highest prediction probability and repeat the process.

**FCRIT, Vashi**
**29/02/20**

## YOLO CFG File

The CFG file contains the variables that specify the configuration of the network.
It mainly consists of the following variables :

- **batch**: That many images+labels are used in the forward pass to compute a gradient and update the weights via backpropagation.
- **subdivisions**: The batch is subdivided in this many "blocks". The images of a block are ran in parallel on the gpu.
- **decay:** Maybe a term to diminish the weights to avoid having large values. For stability reasons most probably.
- **Channels:** refers to the channel size of the input image(3) for a BGR image.
- **Momentum :** is a learning parameter and as specified in the journal a momentum of 0.9 and decay of 0.0005 is used.

- **policy=steps:** Use the steps and scales parameters below to adjust the learning rate during training
- **steps=500,1000**: Adjust the learning rate after 500 and 1000 batches
- **scales=0.1,0.2:** After 500 iterations, multiply the LR by 0.1, then after 1000 multiply again by 0.2
- **angle:** augment image by rotation up to this angle (in degree)
- **filters: How many convolutional kernels there are in a layer.**
- **activation:** Activation function, relu, leaky relu, etc. See src/activations.h
- **stopbackward:** Do backpropagation until this layer only. Put it in the penultimate convolution layer before the first yolo layer to train only the layers behind that, e.g. when using pretrained weights.
- **random:** Put in the yolo layers. If set to 1 do data augmentation by resizing the images to different sizes every few batches. Use to generalize over object sizes.

## Network dimensions (YOLO)

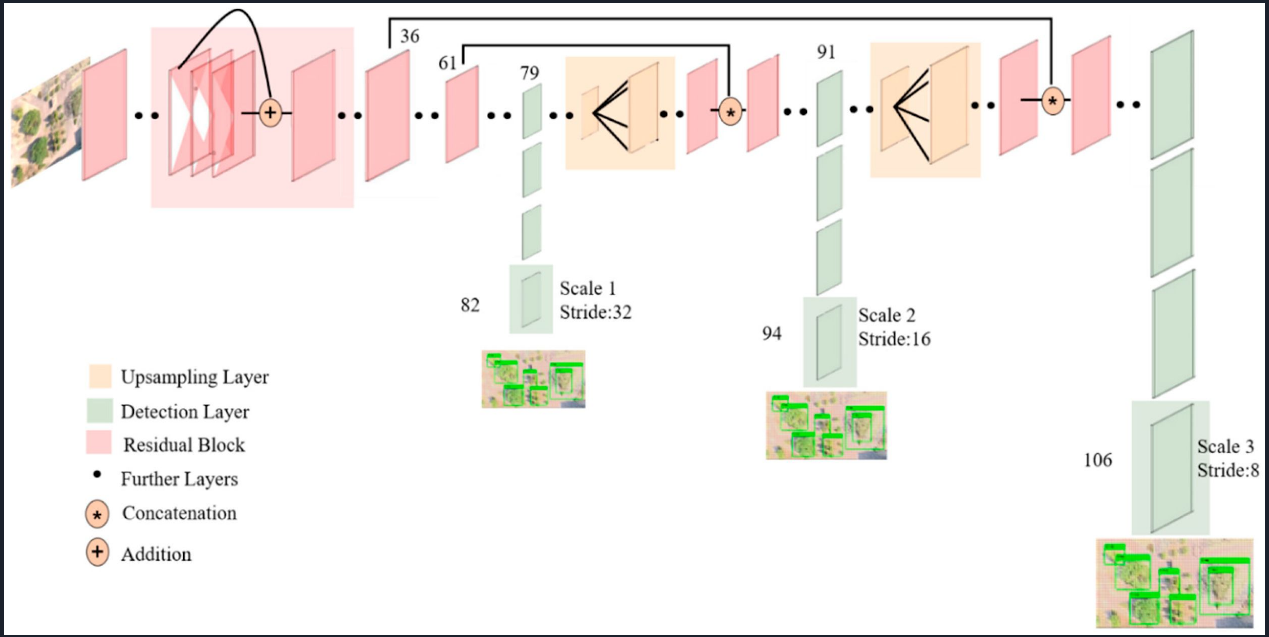
- The input is defined in the cfg file and the size decreases by the factor **stride** which is a variable defined in the cfg file.
- For example, if the input dimension is 416x416, then a stride of 32 means the next layer will have a 13x13 input and so on.
- YOLO has :
    - 75 CNN  ( Convolutional ) Layers
    - 31 other layers ( shortcut, upsample, yolo )
    - 106 layers

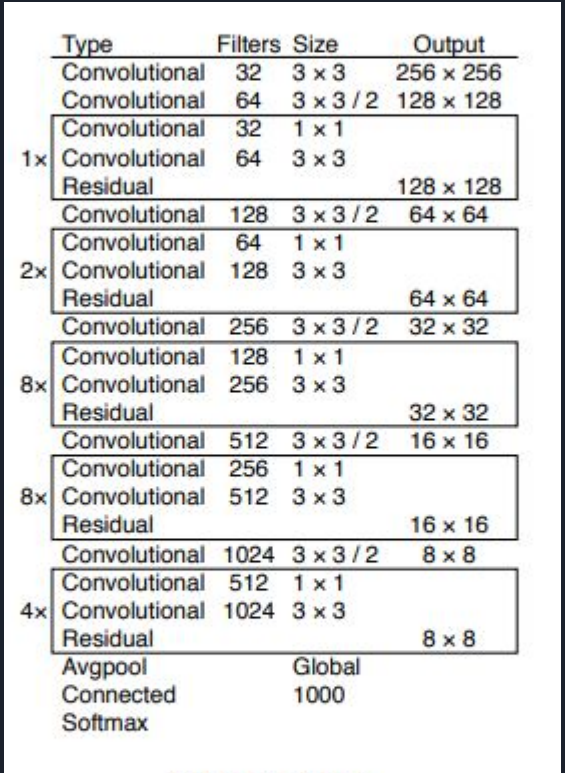
**FCRIT, Vashi**
**29/02/20**

## Network dimensions (Tiny YOLO)

- It is based off of the Darknet reference network  and is much faster but less accurate than the normal YOLO model.
- Tiny YOLO has :
    - 22 CNN (convolutional) layers
    - 8 others (max, route) layers
    - 30 total layers

**FCRIT, Vashi**
**29/02/20**





| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| | Convolutional | 32 | 1 × 1 | |
| 1× | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| | Convolutional | 64 | 1 × 1 | |
| 2× | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| | Convolutional | 128 | 1 × 1 | |
| 8× | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| | Convolutional | 256 | 1 × 1 | |
| 8× | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| | Convolutional | 512 | 1 × 1 | |
| 4× | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

Table 1. **Darknet-53.**

**FCRIT, Vashi**
**29/02/20**

## Changes to cfg file for custom object detection

Make the following changes in training.cfg -

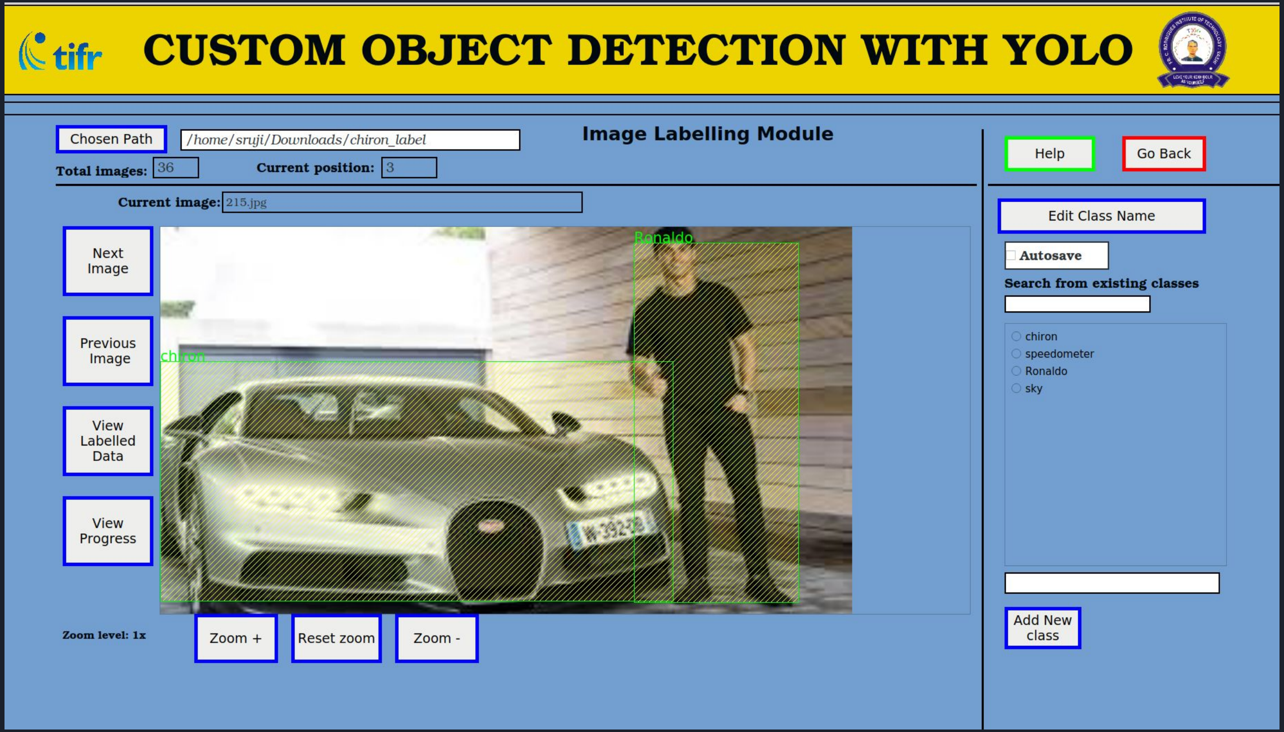
- Line 3: set batch=24, this means we will be using 24 images for every training step
- Line 4: set subdivisions=8, the batch will be divided by 8 to decrease GPU VRAM requirements.
- Line 603/127(for tiny): set filters=(classes + 5)*3
- Line 610/135(for tiny): set classes=_, the number of categories we want to detect
- Line 689/171(for tiny): set filters=(classes + 5)*3
- Line 696/177(for tiny): set classes=_, the number of categories we want to detect
- Line 776: set filters=(classes + 5)*3
- Line 783: set classes=_, the number of categories we want to detect

## YOLO Darknet Application

FCRIT, Vashi
29/02/20

## Labeling Module

- The app basically loads training images and allows users to annotate it by drawing bounding boxes which enclose the ROI ( region of interest) which can be an object (bus,car,book,etc.) that we want to predict later.
- The app should load a training directory and allow the user to go through all the images in that directory and draw bounding boxes around the ROI.
- The user should also be able to add and specify classes to which the ROIs belong.
- The app must then store the image details in YOLO format.
- The app should also not overwrite the classes.txt file in the situation when a new class is added. (One that was not specified before).



Chosen Path: /home/sruji/Downloads/chiron_label

Total images: 36 Current position: 3

Current image: 215.jpg

Autosave

Search from existing classes

- chiron
- speedometer
- Ronaldo
- sky

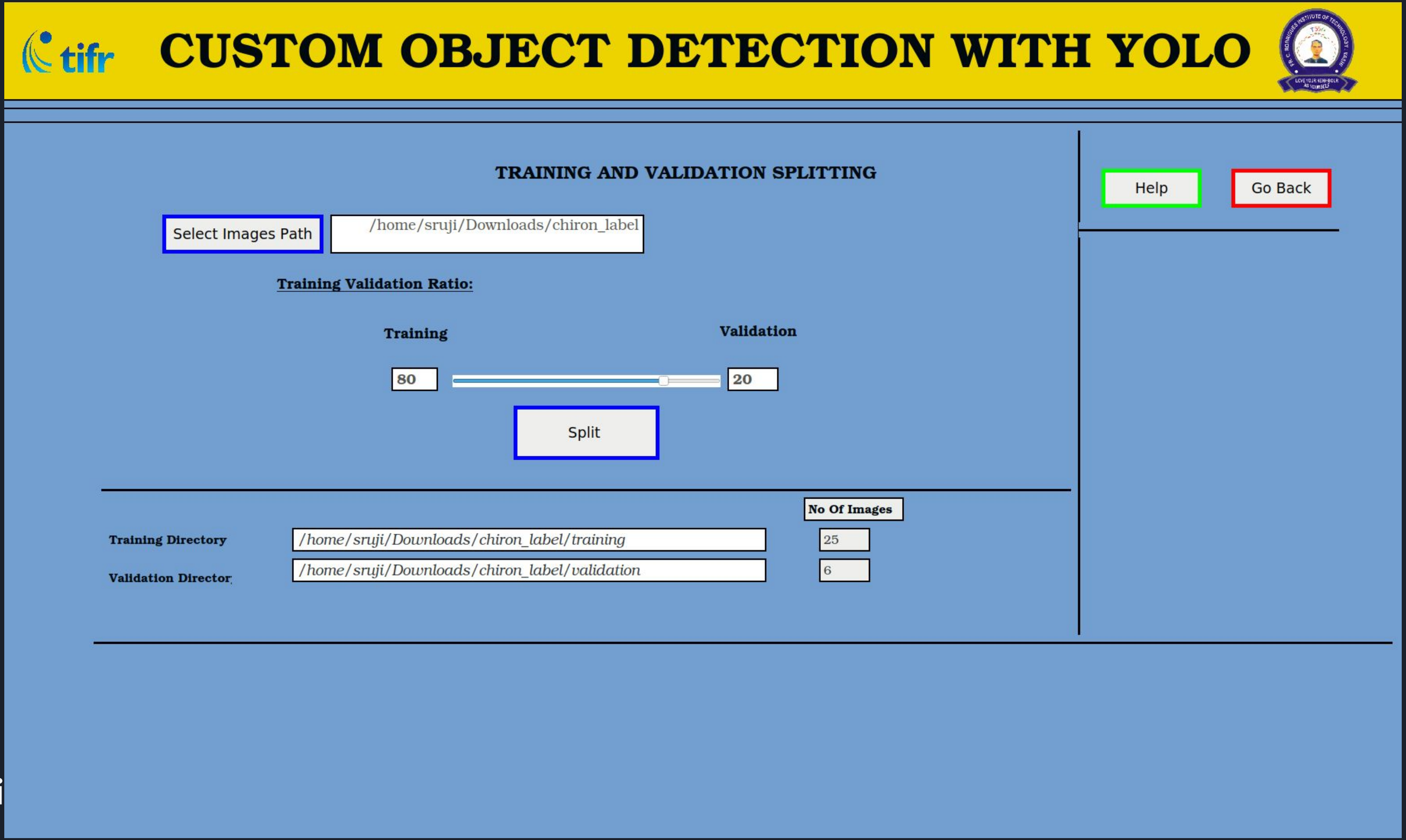
Zoom level: 1x

FCRIT, Vashi
29/02/20

## Data set Split module

- Once we understand how validation dataset works, we can make out why there is a need for the Dataset Split module.
- This takes input the training and the validation split and then randomly samples the data. This sampled data is now stored in a folder called as the Validation and the remaining are then stored in training directory.
- The amount of Validation depends on the ratio provided by the user. The Default is 80:20, but the user is free to change it according to his/her requirement.

**FCRIT, Vashi**
**29/02/20**



CUSTOM OBJECT DETECTION WITH YOLO

TRAINING AND VALIDATION SPLITTING

/home/sruji/Downloads/chiron_label

Training Validation Ratio:

| Training | Validation |
|---|---|
| 80 | 20 |

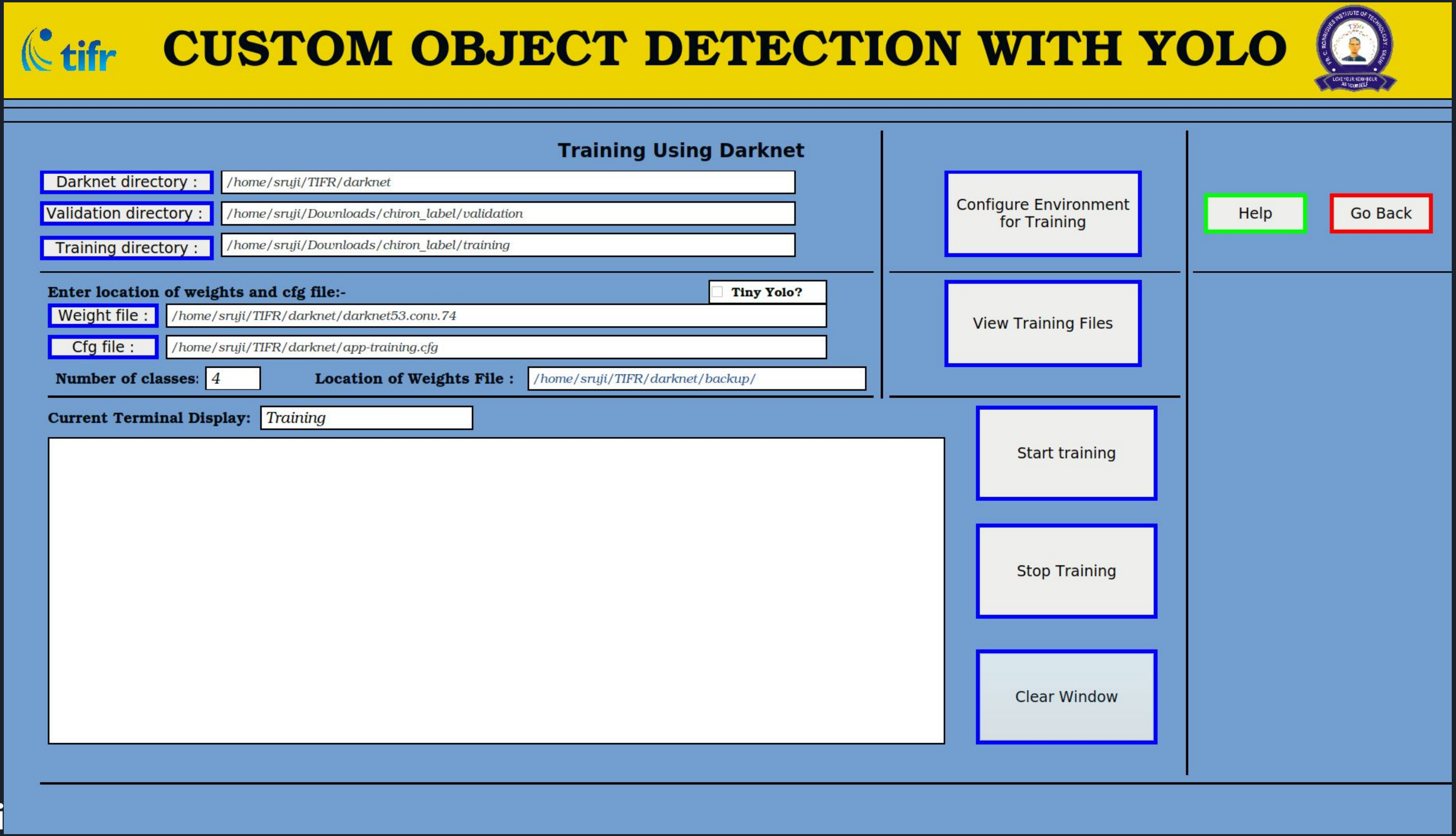
| | | No Of Images |
|---|---|---|
| Training Directory | /home/sruji/Downloads/chiron_label/training | 25 |
| Validation Director | /home/sruji/Downloads/chiron_label/validation | 6 |

FCRIT, Vashi
29/02/20

## Training using Darknet

- Custom Training involves providing a set of directories to the darknet. This involves : -
    - **Darknet Directory** - This is the directory where the Darknet has been installed. The user is expected to run the make in this directory prior to use.
    - **Validation Directory -** This is the directory created by the Data set split Module . The user need to provide the path to this directory.
    - **Training Directory -** This is the directory where all our training data will be located. This is the directory which is created by the data-split application.
    - **Weight File** - This is the starting point for our training. Our model will transfer learn taking the weight of a pre-trained model as the starting point for our training.
    - **Cfg File -** This file contains the information about the weight files like the number of layers, classes and hidden layers.

**FCRIT, Vashi**
**29/02/20**



# CUSTOM OBJECT DETECTION WITH YOLO

**Training Using Darknet**

Darknet directory : /home/sruji/TIFR/darknet

Validation directory : /home/sruji/Downloads/chiron_label/validation

Training directory : /home/sruji/Downloads/chiron_label/training

Enter location of weights and cfg file:-

Tiny Yolo? [ ]

Weight file : /home/sruji/TIFR/darknet/darknet53.conv.74

Cfg file : /home/sruji/TIFR/darknet/app-training.cfg

Number of classes: 4

Location of Weights File : /home/sruji/TIFR/darknet/backup/

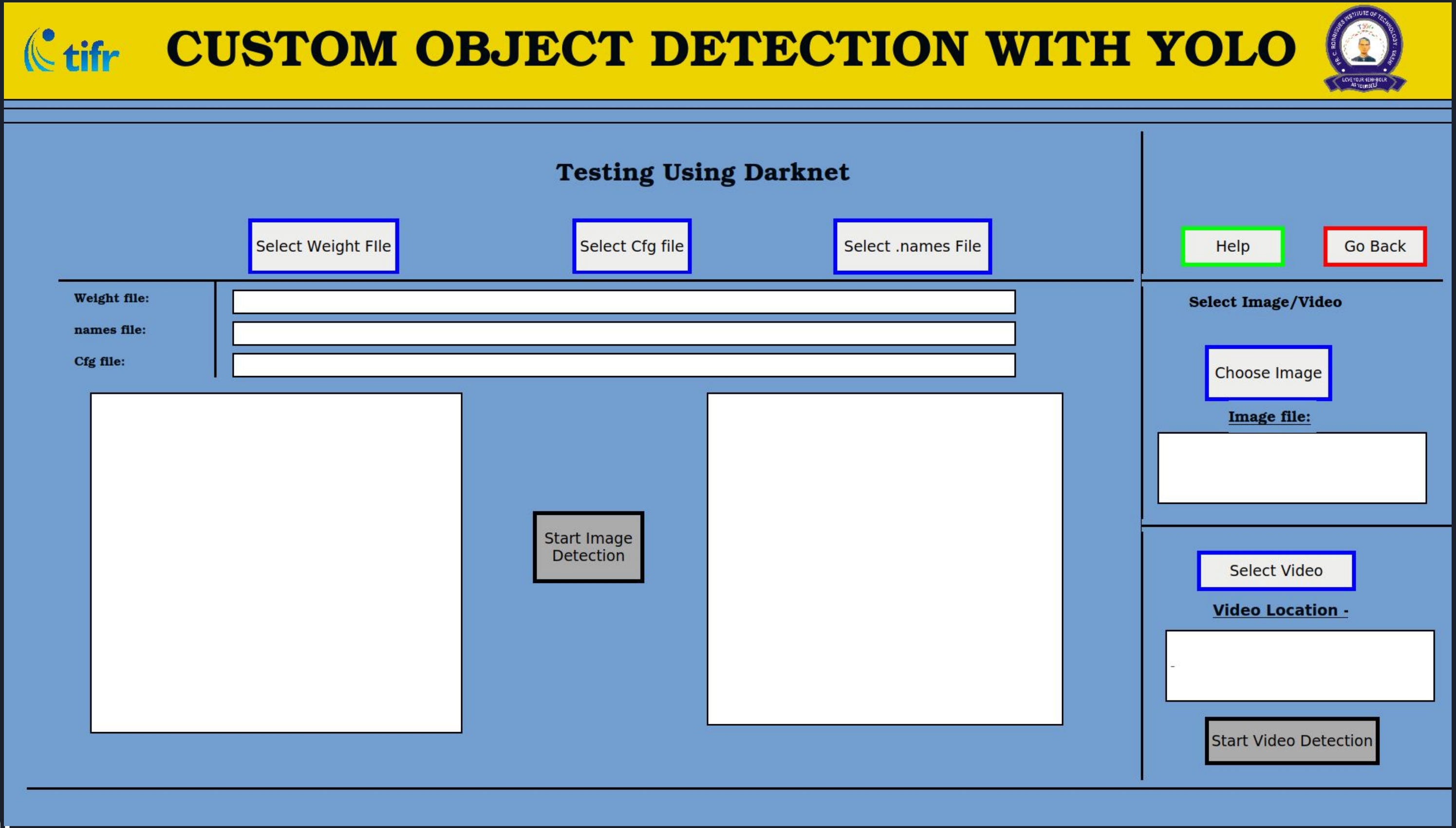
Current Terminal Display: Training

FCRIT, Vashi
29/02/20

## Testing using Darknet

- After we have trained the model, we now need to test it. In order to test our custom trained object, we again need to provide a set of directories. These are :-
    - **Weight File -** This is the weight file which contains our network along with the weights for all the neuron . This is the output file from the training module.
    - **Image/Video** - We provide the image/video on which we want to test our model.
    - **.names file -** This is the data file which contains the name of each class that we want the model to recognize.
    - **Cfg File -** This file contains the modified cfg file from the training. This file contains information about the number of layers, filters, input layer etc.

**FCRIT, Vashi**
**29/02/20**



FCRIT, Vashi
29/02/20

# Ship and Person Intrusion Detection Using Custom Dataset

FCRIT, Vashi
29/02/20

## Procedure

1 - Collect Dataset

2 - Label / Annotate images in dataset

3 - Split Dataset

4 - Train ( we used colab )

5 - Test on Image/Video

FCRIT, Vashi
29/02/20

## Dataset

**Total no. of images used :** **13221**

**Training images :** **12697**

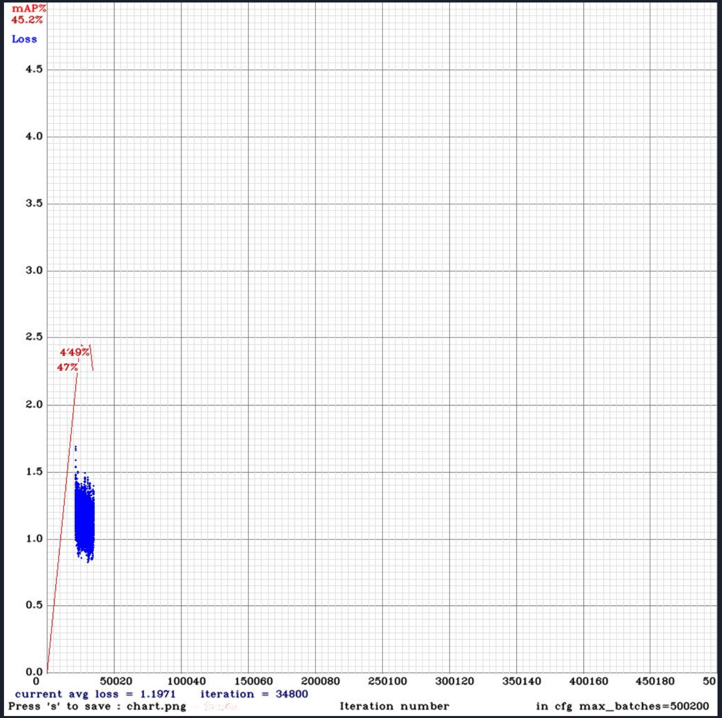
**Validation images :** **524**

**Testing images :** **143**

**No. of classes :** 2 (Ship,Person)

**FCRIT, Vashi**
**29/02/20**

# Training Stats for YOLO (on Colab)



current avg loss = 1.1971 iteration = 34800
Press 's' to save : chart.png
in cfg max_batches=500200

**<u>No. of iterations :</u>** **21000**

**<u>Time Taken :</u>** **14.5 hours**

**<u>Average Loss :</u>** **1.2**

**<u>mAP (Mean Average Precision) :</u>** **40.6%**

**FCRIT, Vashi**
**29/02/20**

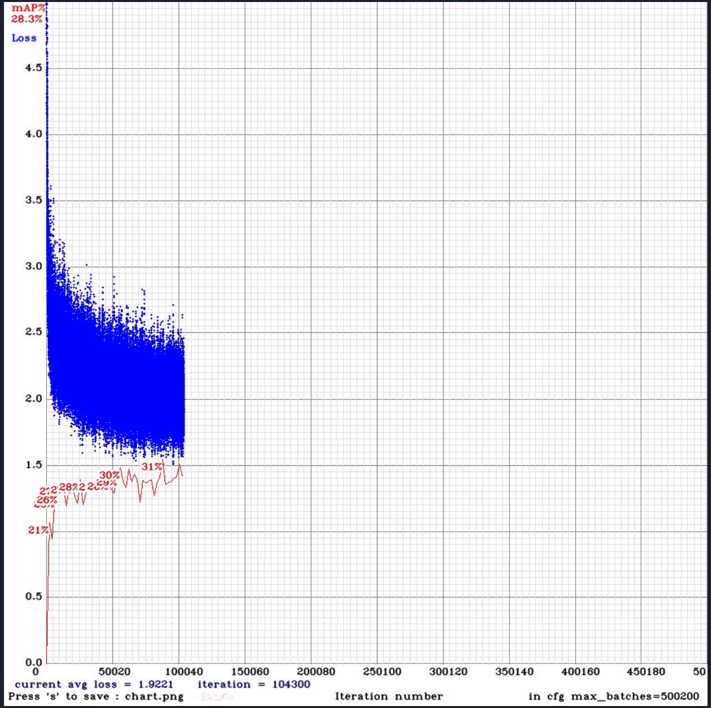
## Training Stats for TinyYOLO (on Colab)

<u>No. of iterations :</u> 104000

<u>Time Taken :</u> 20.5 hours

<u>Average Loss :</u> 1.92

<u>mAP (Mean Average Precision) :</u> 31%



```
current avg loss = 1.9221    iteration = 104300
Press 's' to save : chart.png                 in cfg max_batches=500200
```

FCRIT, Vashi
29/02/20

## Evaluation Metrics Used

1. Accuracy : It is the fraction of predictions that the model got right. Formally -

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ Number\ of\ predictions} \qquad Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

2. Precision : It tells us what proportion of positive identifications was actually correct.

$$Precision = \frac{TP}{TP + FP}$$

3. Recall : It tells us what proportion of actual positives was identified correctly.

$$Recall = \frac{TP}{TP + FN}$$

4. F1-Score : F1 score conveys the balance between the precision and the recall.

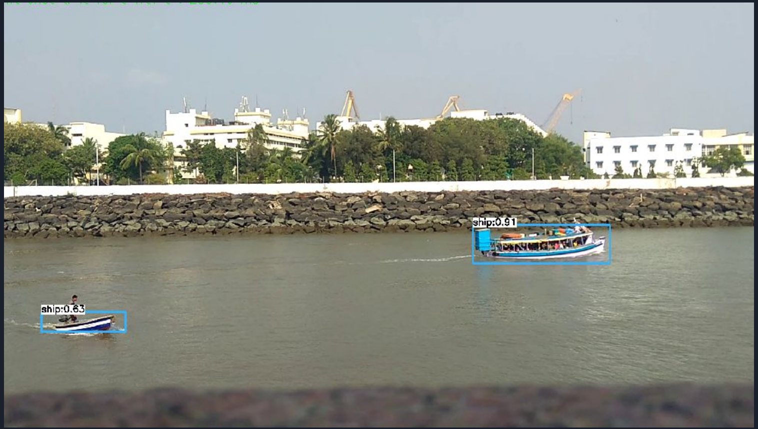
$$F1\ Score = \frac{2 * precision * recall}{precision + recall}$$

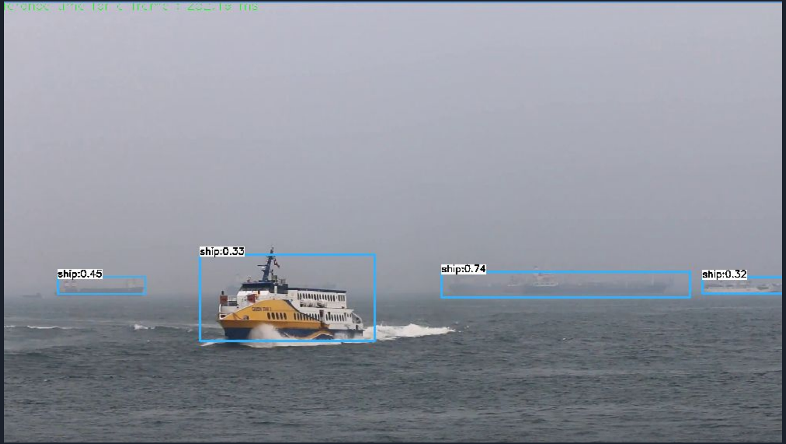
**FCRIT, Vashi**
**29/02/20**

## Results

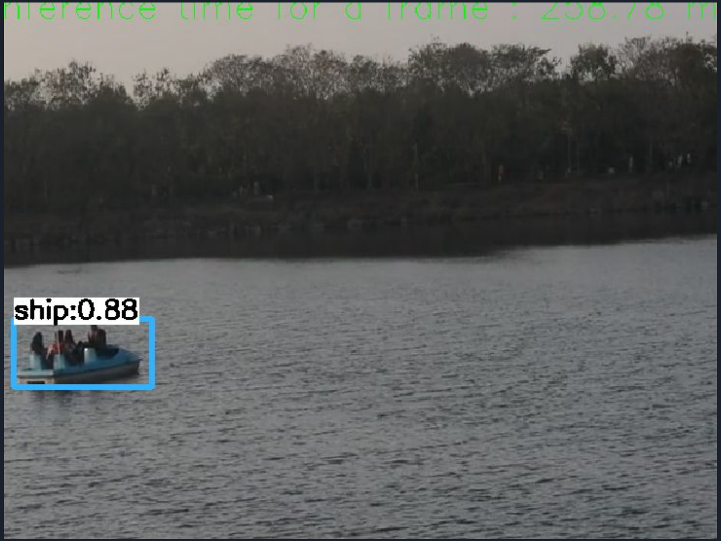
| Metrics | Pre-trained YOLO | Custom-Trained YOLO | Custom-Trained tiny-YOLO |
|---|---|---|---|
| **Accuracy** | 94.37% | 82.52% | 68.06% |
| **Precision** | 98.77% | 98.39% | 97.69% |
| **Recall** | 91.95% | 71.76% | 48.28% |
| **F1-Score** | 95.24% | 82.99% | 64.62% |

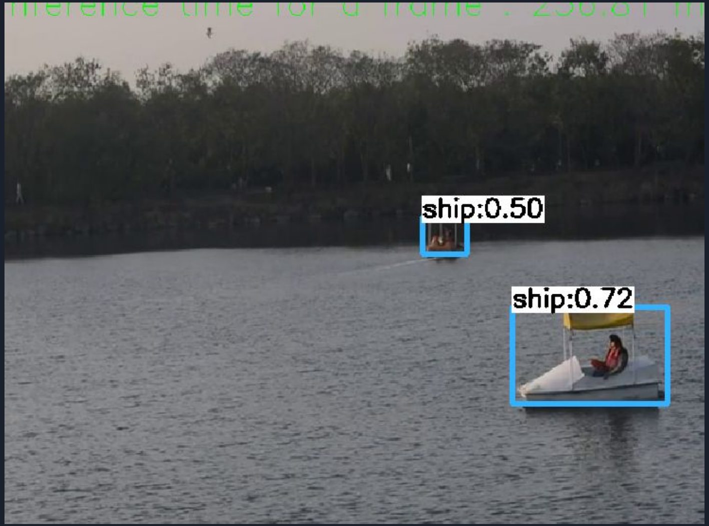
**FCRIT, Vashi**
**29/02/20**



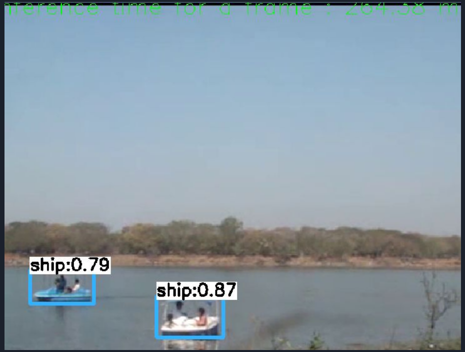


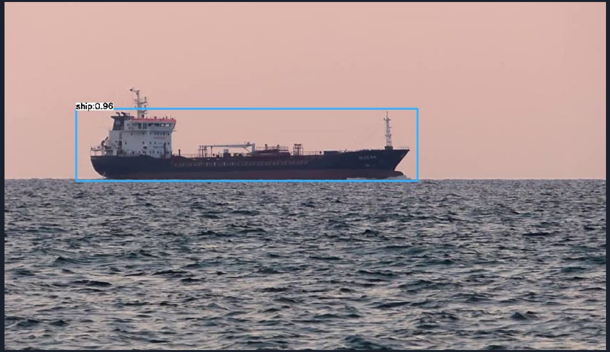
**FCRIT, Vashi**
**29/02/20**

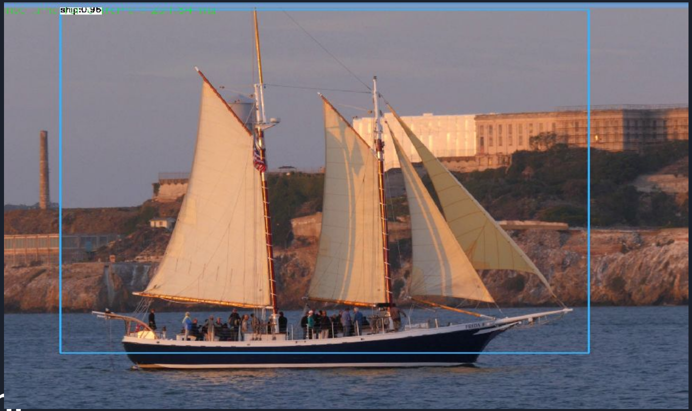


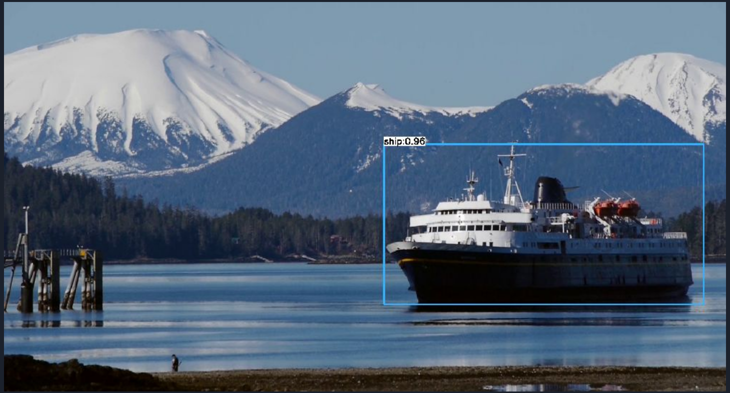


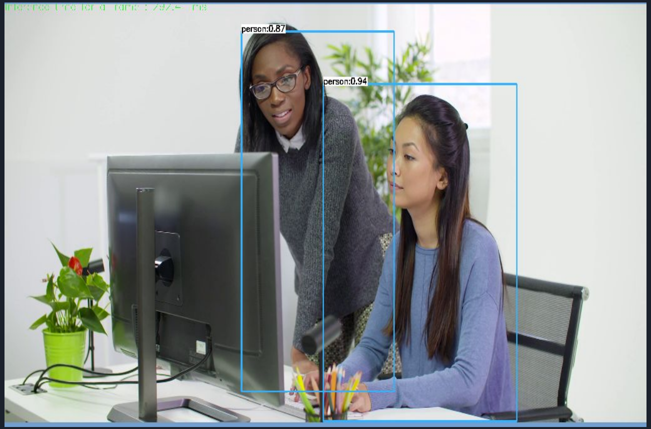
**FCRIT, Vashi**
**29/02/20**


nference time for a frame : 264.38 m
ship:0.79
ship:0.87


ship:0.96


ship:0.96


ship:0.96



**FCRIT, Vashi**
**29/02/20**

# THANK YOU

FCRIT, Vashi
29/02/20