A Project Report

On

Ship Intrusion Detection System

Submitted in partial fulfilment of the requirement of

University of Mumbai

For the Degree of

Bachelor of Engineering

in

COMPUTER ENGINEERING

Submitted by

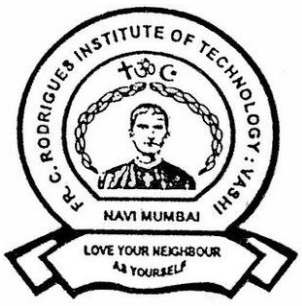
Srujan Patel
Naeem Patel
Siddhesh Deshpande

Supervised by

Mr. Amroz Siddiqui (Guide)

&

Dr. Shashikant Dugad (External Guide)



Department of Computer Engineering

Fr. Conceicao Rodrigues Institute of Technology
Sector 9A, Vashi, Navi Mumbai - 400703

UNIVERSITY OF MUMBAI

2019-2020

APPROVAL SHEET

This is to certify that the project entitled

"Ship Intrusion Detection System"

Submitted by

Srujan Patel	**101641**
Naeem Patel	**101640**
Siddhesh Deshpande	**101612**

Supervisors : _____

Project Coordinator : _____

Examiners : 1. _____

2. _____

Head of Department : _____

Date :

Place :

Declaration

We declare that this written submission for B.E. Declaration entitled "**Ship Intrusion Detection System**" represent our ideas in our own words and where others' ideas or words have been included. We have adequately cited and referenced the original sources. We also declared that we have adhere to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any ideas / data / fact / source in our submission. We understand that any violation of the above will cause for disciplinary action by institute and also evoke penal action from the sources which have thus not been properly cited or from whom paper permission have not been taken when needed.

Project Group Members:

1. Srujan Patel,101641

_____

2. Naeem Patel, 101640

_____

3. Siddhesh Deshpande, 101612

_____

Abstract

Tata Institute of Fundamental Research (TIFR) is located on the shores of the Arabian Sea. Due to its location, it faces a constant risk of intrusion by sea. Despite there being a closed circuit television (CCTV) or video surveillance system in place, there are limitations on the ability of humans to monitor footage from such a system continuously. Hence a foolproof intrusion detection system is required to address this issue. We propose an automated system that would be capable of detection of ships and similar vessels approaching the shore. This system uses Image Processing and Machine Learning for such intruder detection. On detection of intruding ships, the concerned authority (i.e. the security personnel) would receive an alert of the same. The problem that we are concerned with is detection of real time ship intrusion in a dynamic (i.e. constantly changing) environment. We propose a robust, localized, generalized video surveillance system using Deep Neural Networks. Consequently, we look to make the algorithms work faster by using parallel processing. Also, we try and apply the underlying algorithm to identify a range of diverse objects

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Just a few years ago, surveillance systems used to be state of the art technology, limited to military operations. However, nowadays surveillance systems have become the norm in modern day society. Terror and fear have caused most busy environments to be monitored in one way or the other. Security personnel and cameras are installed in shopping malls, airports, schools, sports centres, buildings, public places, etc. Surveillance systems may also be used for non-security purposes like monitoring patients in a hospital or using them to improve the ability of sports players. Intelligent video surveillance systems automatically monitor the environment with little human interference and raise alarms when anomalous behavior is detected. Monitoring tasks include object detection and tracking. Such systems are developed using various techniques from the domains of signal processing, image processing, computer vision, machine learning and artificial intelligence.

1.2 Motivation

The Tata Institute of Fundamental research, Colaba, is situated on the shores of the Arabian Sea. It is at constant risk of security breaches due to ships and other vessels, entering by sea. There is a need for a robust system to be in place to ensure the security and sanctity of the institution. Presently it does have a Closed Circuit Television (CCTV) System in place. These are quite efficient but require constant human monitoring. There are limitations in the ability of humans to constantly monitor video surveillance live footage. Humans watching a video footage for more than 20 minutes lose 95% of their ability to maintain attention sufficient to discern significant events. With two monitors, this is again halved. Clearly, in the case of a huge organization like TIFR where there are multiple cameras, monitoring live footage from all these is a mammoth task. CCTV cameras are widely adopted from small businesses to great nuclear power plants. There are other limitations as well to the use of CCTV cameras and subsequent human monitoring: figures may appear too small to be recognized, or there may be resolution issues, particularly in wide angle cameras.

1.3 Aim and Objective

Our aim is to create a system for detecting ships from CCTV footages and alert the user if a ship is detected. The task would be to create an end to end system for Annotating images, training ship detection system and of course testing our trained model. The system so designed would be capable of not only detecting ships but any object the user wishes to detect.

1.4 Report Outline

This report details our approach to the problem of detecting ships from CCTV footages and ultimately create a end-to-end system which will enable users to detect any object of interest without the need for any external software tool. We have provided a comprehensive approach to this problem which includes information about annotation of images, training time , loss and the accuracy of the trained models.

Chapter 2

Study Of the System

2.1 Related Works

2.1.1 Closed-circuit television (CCTV)

Closed-circuit television (CCTV) cameras are used to produce images or recordings for surveillance purposes. A Basic CCTV System Consists of a camera connected directly to a monitor by a co-axial cable with the power for the camera being provided from the monitor which is used to check for intrusion of ships. Although it is a very easy and simple solution, 24/7 monitoring of the video recordings is difficult due to the high risk of human error. [1]

2.1.2 RADAR (Radio Detection and Ranging)

RADAR (Radio Detection And Ranging) method is used to detect the intrusion of ships. The background of sea- shore landscape Synthetic Aperture Radar (SAR) image is dark and targets are bright making it is easy to detect the ship.[2] But when the wind is fierce, large waves will be stirred and engender strong backscattering echo. This causes many difficulties in the detection of ships. Thus the overall accuracy turns out to be poor and makes it difficult to detect the ship during bad weather condition. [3]

2.1.3 Wireless Sensor Network

Equipped with three-axis accelerometer sensors, we deploy an experimental Wireless Sensor Network (WSN) on the sea's surface to detect ships. Using signal processing techniques and cooperative signal processing, we can detect any passing ships by distinguishing the ship-generated waves from the ocean waves.[4] A three-tier intrusion detection system with which we can exploit spatial and temporal correlations of an intrusion to increase detection reliability. [5]

2.1.4 Phase saliency map and extended wavelet transform

This method for ship detection algorithm is based on phase saliency map and extended wavelet transform (PSMEWT) to solve two issued: Accurate detection of ship in complex background and How to detect accurate ship targets in the event that ship targets are comparable to false alarms. Applied extended wavelet transforms and morphological methods on multi spectral data fusion and separated sea and land areas and remove isolated holes. PSMEWT combined with ship target of regions of interest ROIs which are extracted with similar intricate background. Location and con-

tour of the ROIs were taken out from threshold segmentation method as well as texture segmentation method. [6]

2.2 Literature Review

2.2.1 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. Object detection algorithms typically use extracted features and learning algorithms to recognize instances of an object category. It is commonly used in applications such as image retrieval, security, surveillance, and advanced driver assistance systems (ADAS). Programming a computer and designing algorithms for understanding what is in these images is the field of computer vision. 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 and localization determines the location of that object in the image. [7]

2.2.2 Deep Learning

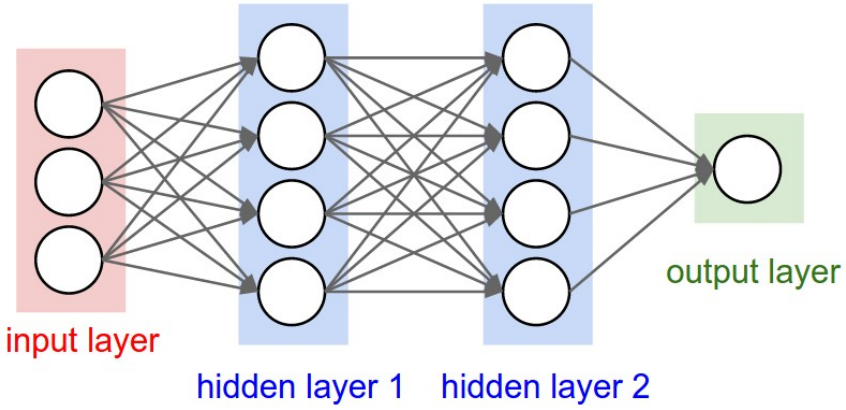
Deep learning is a machine learning technique which teaches computers to perform activities that comes naturally to humans and by learning every day to day ventures. In deep learning, a computer model learns to classify tasks directly from text, sound, images or video clips. Deep learning models can 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. Nowadays, deep learning is achieving higher levels of recognition accuracy than ever before. Recent advancements in deep learning has improved to such stage that it outperforms humans in classifying objects in images. In simple words, It is a procedure of automatically predicting and analyzing the given information by the user. [8]

2.2.3 Convolutional Neural Networks

Most deep learning methods usually use a neural network architecture, which is the reason why deep learning models are often known as deep neural networks. The term “deep” usually refers to the number of hidden layers in the neural network. Our neural network contains only 2-3 hidden layers, while deep networks can have as many as 200. There is no need of

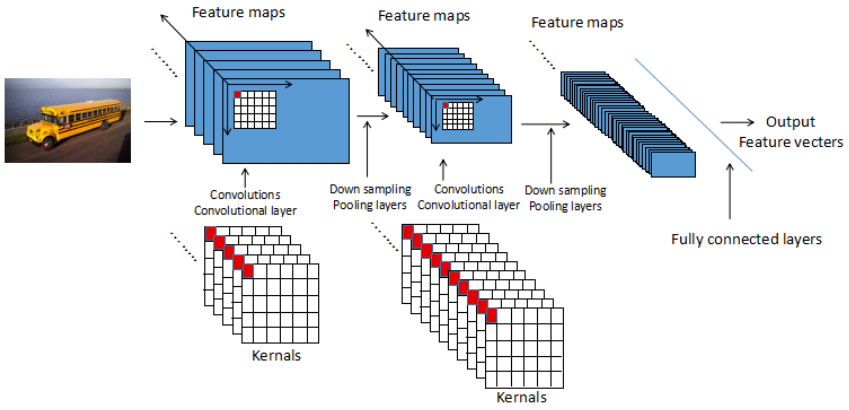
manually extracting the features because deep learning models are already trained using large sets of labeled data and neural network architectures which learn features from the given data. [8] One of the most used deep neural network is convolutional neural network (CNN). A ConvNet convolves learned features with given input data and uses convolutional layers making a well suited architecture for extracting objects directly from the images, so there is no need of identifying the features used to classify images. This way of extraction makes deep learning models to give least inaccuracy in larger context to classify objects in computer vision. [9] One of the most used deep neural network is convolutional neural network

Figure 2.1: Neural Network with interconnected hidden layers



(CNN). A ConvNet convolves learned features with given input data and uses convolutional layers making a well suited architecture for processing image datasets. ConvNet extracts features directly from the images, so there is no need of identifying the features used to classify images. This way of extraction makes deep learning models to give least inaccuracy in larger. [10]

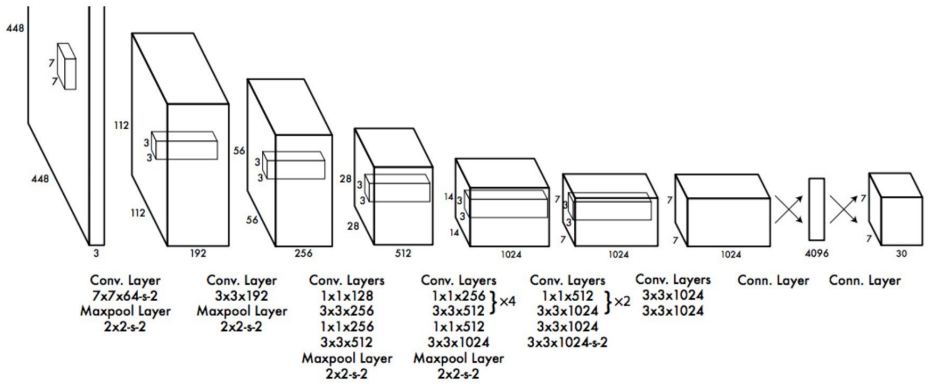
Figure 2.2: Example of convolutional layers



2.2.4 YOLO Algorithm

YOLO (You Only Look Once) real-time object detection algorithm, which is one of the most effective object detection algorithms.YOLO uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. [11] YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes. With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

Figure 2.3: YOLO Architecture

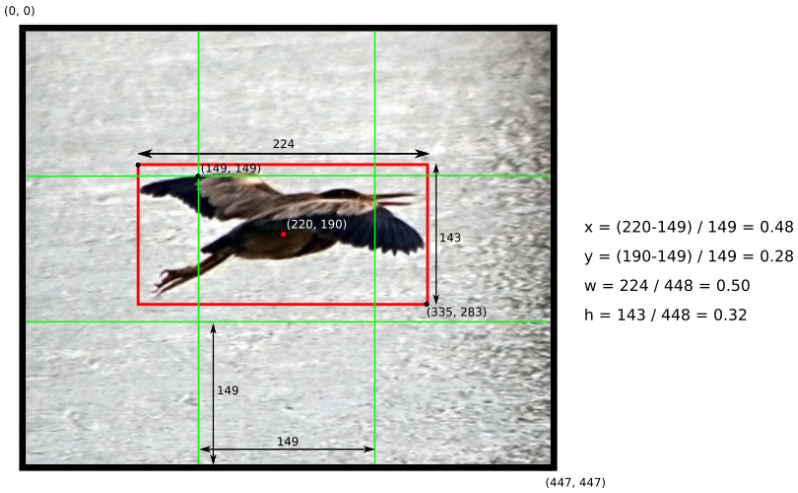


Each bounding box has -
(a) X, Y coordinates, width, height and confidence score - 5 attributes
(b) Probability distribution over classes - 80 attributes
Therefore the final shape is S x S x 255. (Where S is the grid size)

2.2.4.1 NMS and IoU

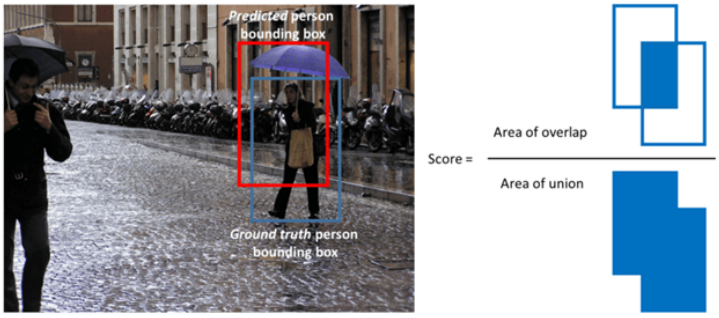
YOLO uses Non-Maximal Suppression (NMS) to only keep the best bounding box. The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold. In the code below, we set this NMS threshold to 0.6. This

Figure 2.4: YOLO Algorithm - Example of bounding box



means that all predicted bounding boxes that have a detection probability less than 0.6 will be removed. YOLO algorithm uses Intersection over Union (IoU) to check similarity between two bounding boxes.IoU is also used to calculate the common area between the boxes. After removing all the predicted bounding boxes that have a low detection probability, the second step in NMS, is to select the bounding boxes with the highest detection probability and eliminate all the bounding boxes whose Intersection Over Union (IOU) value is higher than a given IOU threshold. [12]

Figure 2.5: YOLO Algorithm - Intersection over Union (IoU)

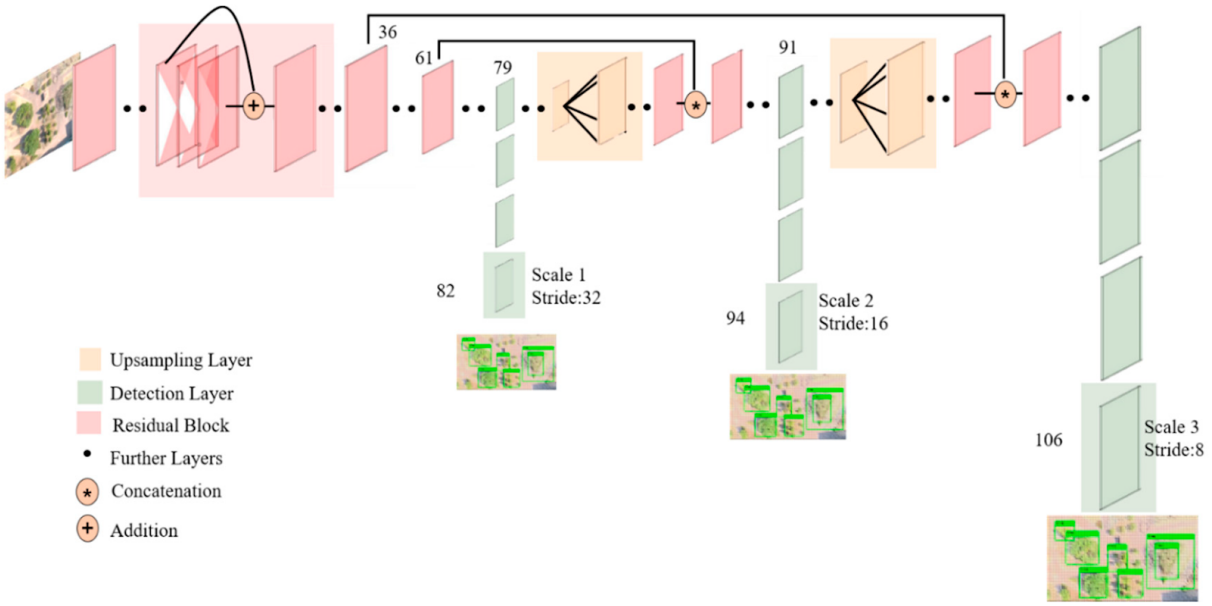


2.2.4.2 YOLO Network Dimensions

The input dimensions to the network is defined in the configurationn file and the size decreases by the factor stride which is a variable defined in the configuration 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) and total of 106 layers Tiny YOLO is another light-weight version 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 and 30 total layers.

Figure 2.6: YOLO Algorithm - Layers of the network



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

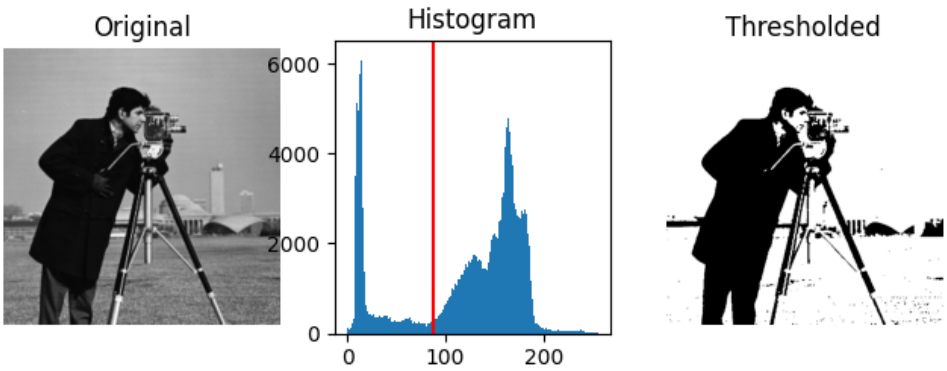
Table 1. **Darknet-53.**

2.3 Image Processing Techniques

2.3.1 Thresholding

Thresholding is the the process of image segmentation where a normal image is converted into binary image. Normal image is a color image consists of 3 channels(RGB), and each pixel has three different intensities for the three primary colors, ie red green and blue, which range from 0-255. A grayscale image on the other hand has only 1 channel, ie each pixel consists of only one value in the range 0-255.IA binary image is an image where a pixel can have only one of the two values, either 0 or 1.

Figure 2.7: Thresholding on a sample image



2.3.1.1 Binary Thresholding

Thresholding techniques are mainly used insegmentation The simplest thresholding methodsreplace each pixel in an image with a black pixel if thepixel intensity is less than some fixed constant T, else it is replace with a white pixel. Thresholding techniques are mainly used in segmentation The simplest thresholding methods replace each pixel in an image with a black pixel if the pixel intensity is less than some fixed constant T, else it is replace with a white pixel. If I (i,j) is the intensity at point (i,j) in an image, then:

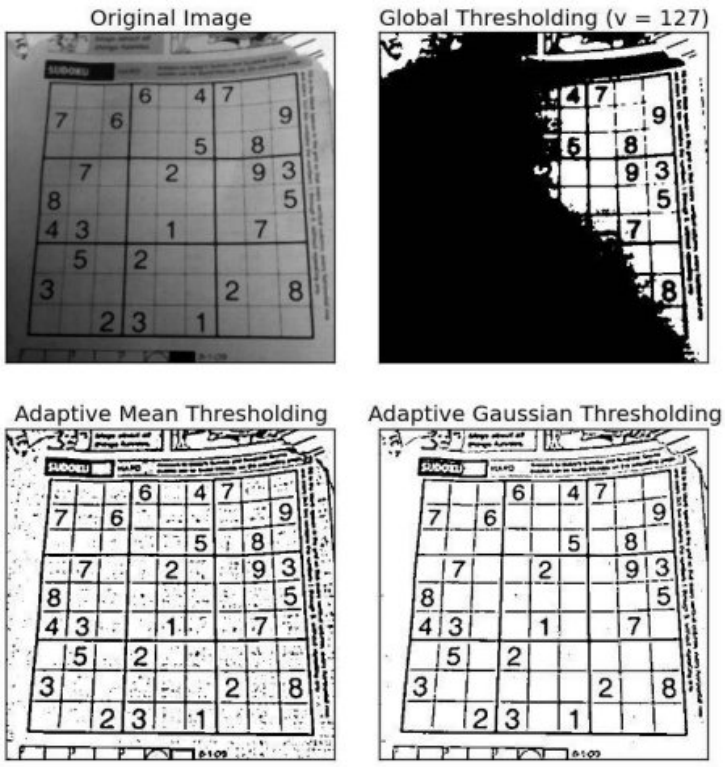
I(i,j) = 0 if I(i,j) $\leq$ T
else I(i,j) = 1
where T is some threshold value.

2.3.1.2 Adaptive Thresholding

Adaptive thresholding is a type of thresholding, where the value of a pixel is set considering it's neighbouring pixels. In many cases, some parts of the subject can remain under shadow, and different illumination may also

affect the subject It takes the illumination of the subject and background it consideration. It often produces better results than constant thresholding.

Figure 2.8: Adaptive thresholding on a sample image



2.3.1.3 OTSU Thresholding

The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), It then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

Figure 2.9: OTSU thresholding on a sample image

2.3.2 Smoothing

Images are not smooth because adjacent pixels are different. Smoothing is the process of making adjacent pixels look more similar. Smoothing strategy for a pixel is to make it the average of its neighbors.

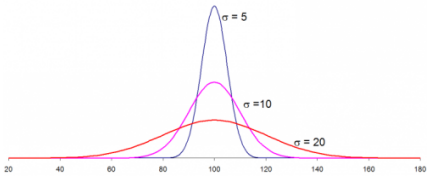
2.3.2.1 Gaussian Blur

All images contain some amount of noise. The image is first smoothed by applying a gaussian filter A gaussian filter with $\sigma = 1.4$ is :

Figure 2.10: Gaussian filter example

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figure 2.11: Gaussian curve



A gaussian filter prioritizes the middle pixel (anchor element) and gives less weights to the pixels away from it. (radially) This helps in preserving edges.

Figure 2.12: Gaussian smoothing applied to a sample image

2.3.2.2 Median Blur

The Median blur operation is similar to the other averaging methods.

The central element of the image is replaced by the median of all the pixels in the kernel area. This operation processes the edges while removing the noise. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image). Median Filter: A Matrix that is used to find the median of the pixel with respect to its neighbouring pixels.

Figure 2.13: Median Blur applied to a sample image

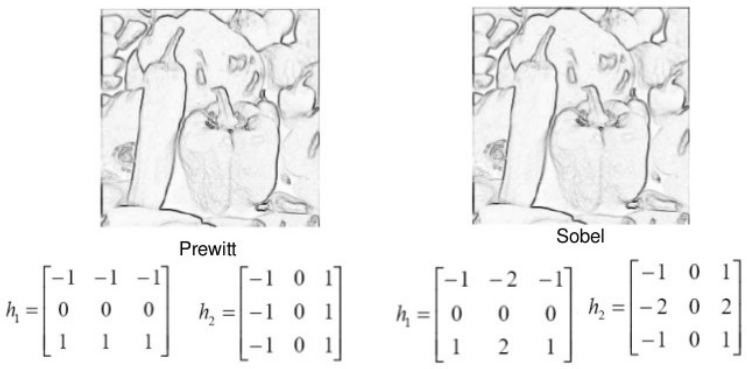
2.3.3 Edge Detection

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision. [13]

2.3.3.1 Prewitt and Sobel Edge Detection

These are first order edge detection techniques in which we convolute with the following kernels.

Figure 2.14: Prewitt and Sobel Kernels



2.3.3.2 Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages.

1. Noise Reduction
   Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

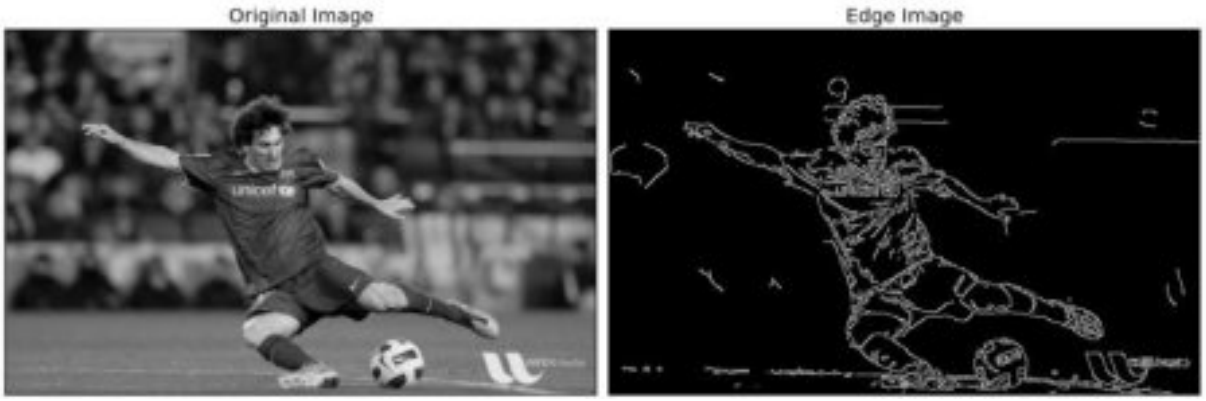
2. Finding Intensity Gradient of the Image
   Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel. Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

3. Non-maximum Suppression
   After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.

4. Hysteresis Thresholding
   This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or

non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

Figure 2.15: Canny edge detection applied to sample image



Chapter 3

Proposed System

3.1 Problem Statement

Traditionally, analog CCTV video surveillance system is used wherein a number of cameras are placed at remote locations, and connected to set of monitors placed in a single control room. These monitors are observed throughout the clock by a human operator who keeps track of intrusion by unauthorized entities. This method is not foolproof as the human operator may have to single-handedly keep track of a number of video monitors and he may get tired after a long time. This affects the ability of the human operator to detect and report intrusions. Limitations of human ability for monitoring video footages calls for an automated system for detecting and tracking ships and other similar marine vessels from a sequence of input video frames. This input may be pre-recorded or real-time. The system will work in a dynamic environment, i.e. even in the presence of background disturbances such as winds, waves, birds flying, etc. The system requires that the object to be detected is at least 5% of the input dimensions. The object will be detected at multiple scales of the object. The detection is indicated by a bounding rectangle that encloses the detected target and an alarm is raised to alert the concerned authority of the intrusion.

3.2 Scope

The project envisages a system that will detect any ships and other water vessels that trespass into the coastal area. This system can be generalized to detect any other class of objects. Detection is achieved using Image Processing and Machine Learning techniques. The system is an automation solution that can detect ships round the clock and notify the administrator of any Intrusion. This is an improvement over conventional manual inspection of video feeds. The efficacy of detection depends on the quality of the footage from the CCTV. A good camera will ensure accurate detection. Also, it is assumed that the system is provided with some footage for 'Training' which is still background of the surveillance environment, free from any ship or other noise. The system can have the following for interaction:A GUI for end user interaction. Email for sending alerts. To provide results with optimum accuracy, we suggest the following: The CCTV should be rigidly fixed in the environment. The footage should not be shaky or have other disturbances. It follows that a good training set leads to a robust classifier

3.3 Proposed System

3.3.1 Approach

YOLO uses a single CNN network for both classification and localising the object using bounding boxes. Here, a single convolutional network simultaneously predicts class probabilities and multiple bounding boxes. YOLO trains on full images and directly optimizes imizes the detection performance. This special model has shown several benefits over traditional methods of object detection as it instead of selecting interesting parts of an image, predicts classes and bounding boxes for the whole image in single run of the algorithm. It is commonly used for real-time time object detection. Our job is to predict a class of an object and the bounding box specifying object location.
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 detection module.

3.3.2 Login, Main Menu and Profile Module

Registration allows users to register themselves on the system using a valid email and password. Users can log in to the system once they are registered. Only registered users can log in to the system. If unregistered users try to log in, an "Invalid User" notification pops up and the unregistered users are prompted to register before logging in. On the login page, the user can click the help button for a description of the application.
Once logged in, the user is presented with the Main Menu, from here, he can view instructions on how to proceed. The user can go to the required module, for training or testing, a dropdown menu will further provide more options to perform the required tasks.A user can view their profile which contains the file/folder paths that the user may have previously selected for the application, these paths can be copied to clipboard as well. On using a module and returning, the session is refreshed and the current state of that module is reverted to initial state.

3.3.3 Labeling / Image Annotation Module

This module 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. This module loads a training directory and allows the user to go through

all the images in that directory and draw bounding boxes around the ROI. The user can also add and specify classes to which the ROIs belong.
This module stores the image details in YOLO format which is saved as a .txt file. The file contains object number and object coordinates on this image, for each object in new line:

<object-class><x><y><width><height>

where,

- **Object class** - integer number of object from 0 to (classes-1)
- **<x><y><width><height>** - float values relative to width and height of image, it can range from 0.0 to 1.0
- **Attention**: x and y are center of rectangle (are not top-left corner)

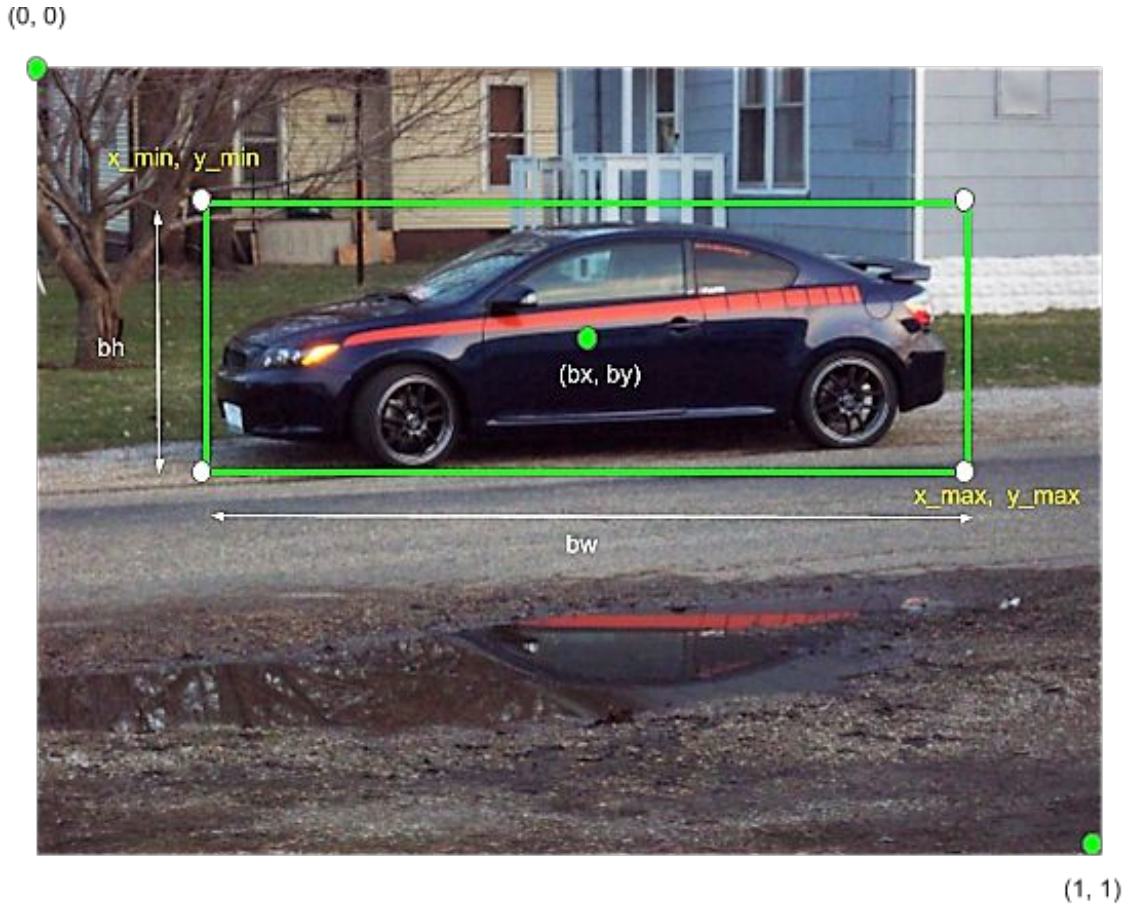
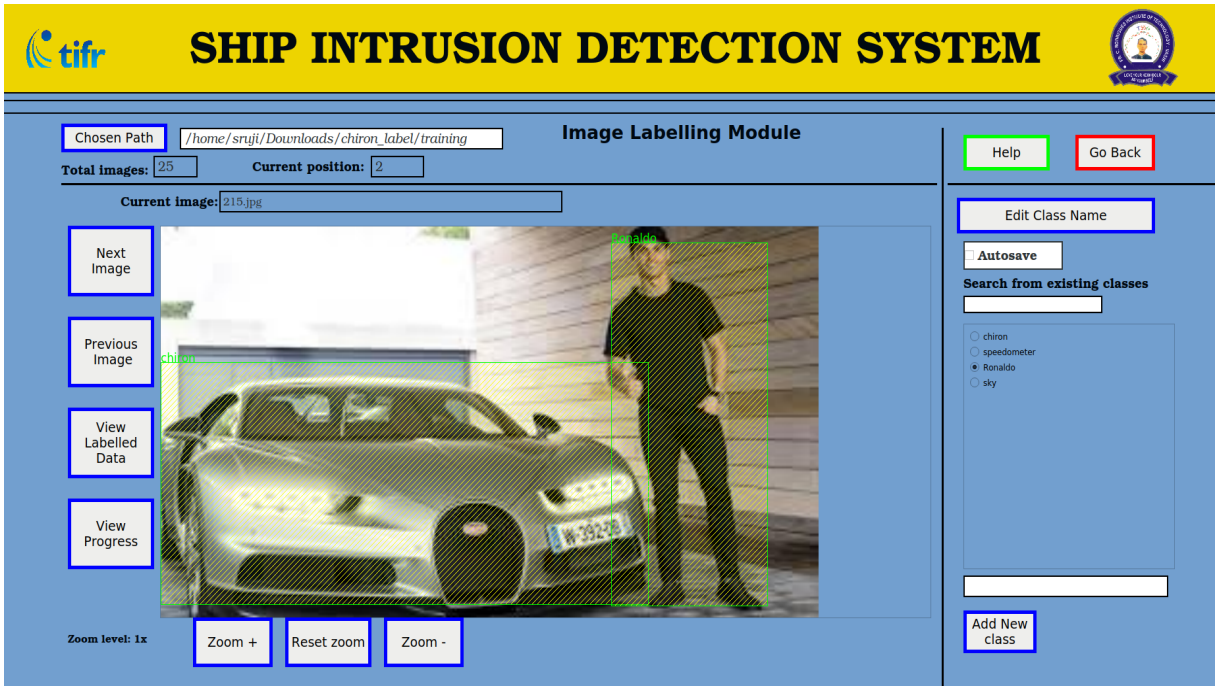


Figure 3.1: The associated .txt file will have the following contents (if car is class 0): 0 0.0833 0.0833 0.0556 0.0417

We have added various functionalities in this module to make the process as user-friendly and robust as possible.There is a help section to guide the user in case of confusion.All the user needs to select first is the dataset

Figure 3.2: Labeling Main Page



directory.All other options are disabled except the 'help' and 'go back' until the user selects the correct directory. The correct directory means that the directory should contain at least one image, if the directory does not contain any image, then the user is notified and prompted to select the correct directory.

There is no cap on the number of classes that the user can add, the user can add their class and even edit the class name. While drawing bounding box, the user can even search from the list of existing classes to minimize effort in scrolling and then selecting the class. The class names are saved in the 'classes.txt' file which contains each class name on each line. The classes are 0-indexed.

Once the directory is selected, the first image will load and the user can then go through the images using 'next image' and 'previous image' button. The app will save previous annotations and they will load if a session is resumed. The app also provides an option to view the corresponding .txt file generated for the current image.

Drawing bounding boxes is intuitive with the use of mouse. The app will ask for confirmation everytime a bounding box is drawn. To skip this, the user can check the "Autosave" option to remove these confirmation messages. To delete a bounding box, the user can right click and delete and to edit the class of a bounding box in case of a mistake ,the user has

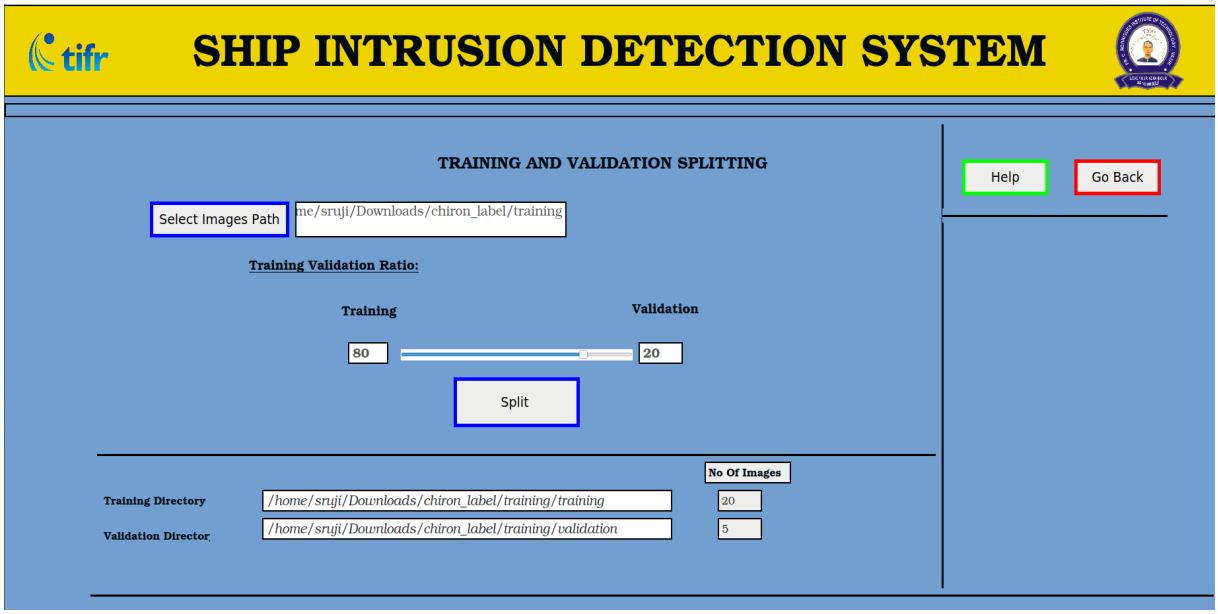
to select the correct class after pressing edit on right click.
Apart from these, the user can zoom-in and zoom-out in case the item to be annotated is too small or too large. The user can also reset zoom and view the zoom level.
The app also displays the total number of images and the current image number and name. The user can also view the progress of the labeling of the dataset. This displays the percentages and names of the images that have been labeled and not yet labelled. It also displays the percentage and names of images that have been copied to the training and validation directories in case the user has already split the dataset before. Once, the user decides to go back, the session is refreshed and the user can start afresh. If the user wants to continue with the same session, they can check the profile section which saves the previously selected directory locations that can even be copied to clipboard.

3.3.4 Dataset Split Module

Training Dataset is The sample of data used to fit the model. This is the actual dataset that we use to train the model (weights and biases in the case of Neural Network). The model sees and learns from this data.
Validation Dataset is The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. We as machine learning engineers use this data to fine-tune the model hyperparameters. Hence the model occasionally sees this data, but never does it "Learn" from this. We use the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.
This module takes as 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 using the slider according to his/her requirement.
After selecting the Image Directory, the user can split and the number of images in each set (training and validation will be displayed alongwith the path).
This module does not move the images, it copies the images into training

and validation subdirectories inside the main dataset directory.
The user can also go back if they wish or view the instructions by clicking the help button.

Figure 3.3: Dataset split Module



3.3.5 Training Module

YOLO allows us to train the model on our own dataset to detect our own classes. The actual training of the network happens in this module. To train the network, their are certain inputs required and our application then generates the required files when the user clicks the configure environment button. The user can select the tiny-yolo checkbox to train using tiny-yolo instead of normal YOLO. Custom Training involves providing a set of directories and files. These include :-

3.3.5.1 Darknet Directory

This is the directory where Darknet has been installed. The user is expected to run the make in this directory prior to use.

3.3.5.2 Validation and Training Directory

Validation Directory is the directory created by the Data set split Module . The user need to provide the path to this directory. Training Directory is the directory where all our training data will be located. This is the directory too is created by the data-split application.

3.3.5.3 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. If starting training on a new dataset, the use of the darknet.conv.74 weight file is recommended. If the user has already trained before, they can resume training with those weights as well. By default, darknet saves the weight file every 100 iterations and then every 1000 iterations after 1000th iteration.

3.3.5.4 Cfg File

The Configuration (cfg) file contains the variables that specify the configuration of the network. The user has to provide the standard cfg file that is available on the official darknet site. It is named yolov3.cfg or the tiny-yolo variant.
The .cfg 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.

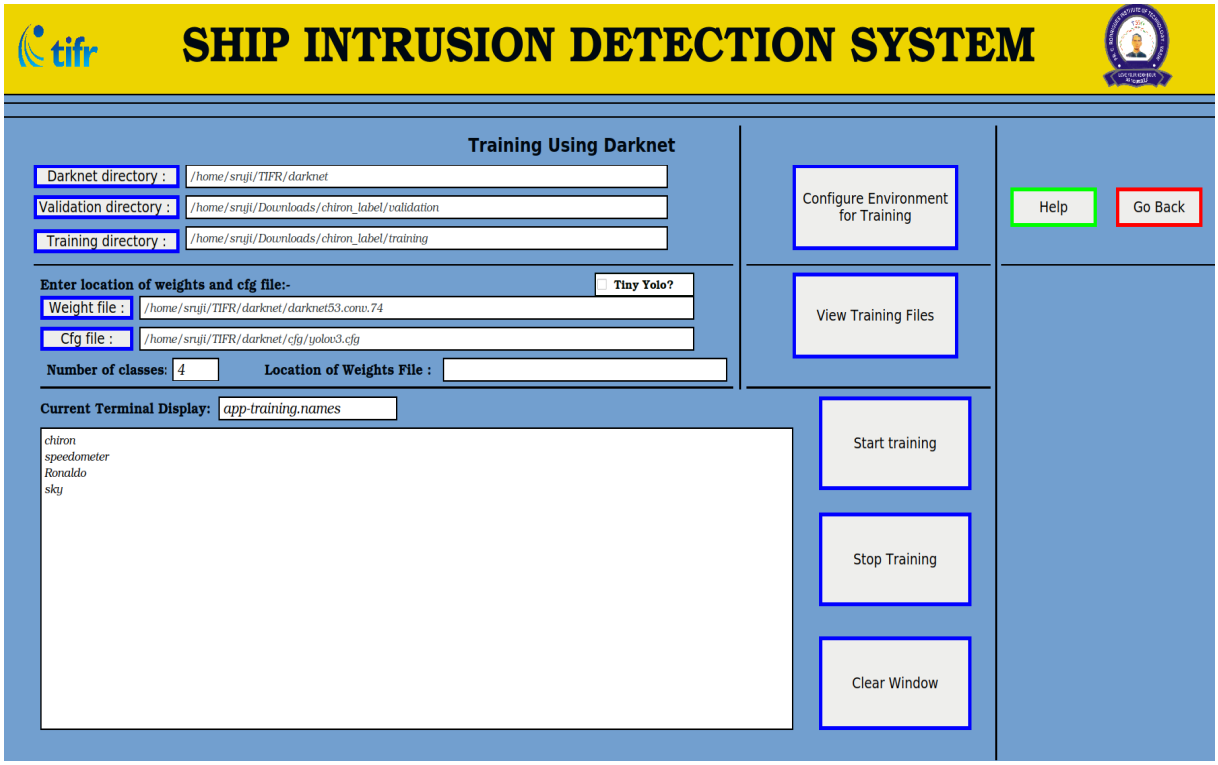
To train the network for our dataset, we need to make the following changes in the .cfg file :
- Line 3 : set batch=24, this means we will be using 24 images for every training step
- Line 4: set subdivisions=8, the batch will be divided by 8 to decrease GPU VRAM requirements.
- Line 603/127(for tiny): 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

Once the inputs are given, the user has to configure environment for training, without this all the other options are disabled. After generation of required files, the user can view the generated files on the output window in the page. The files that are generated are :
- app-training.data : This file contains the supplementary information like the number of classes, the location of the other generated files and the path of the backup directory where the weight file generated from training is stored.
- app-training.cfg : This file is the modified cfg file that is customized to our dataset.
- app-training.names : This file contains each class name on a seperate line.
- app-training-test.txt : This contains the path to all the images in the validation set.
- app-training-train.txt : This contains the path to all the images in the training set.

Once these files are generated then we can start training, the output

comes in batches from the terminal, so there is a little latency. The user can stop the training as well which kills the process. Furthermore, the output window can be cleared by the clear window button's clicking.
The output of training provides some float numbers like IoU and the average loss, we have to see the average loss and decide if we are satisfied with the training. If enough iterations are run, then the app displays the location of the weight file which can be then used to perform detection or further train the model. The app draws a bounding box around the detected object.

Figure 3.4: Training Module



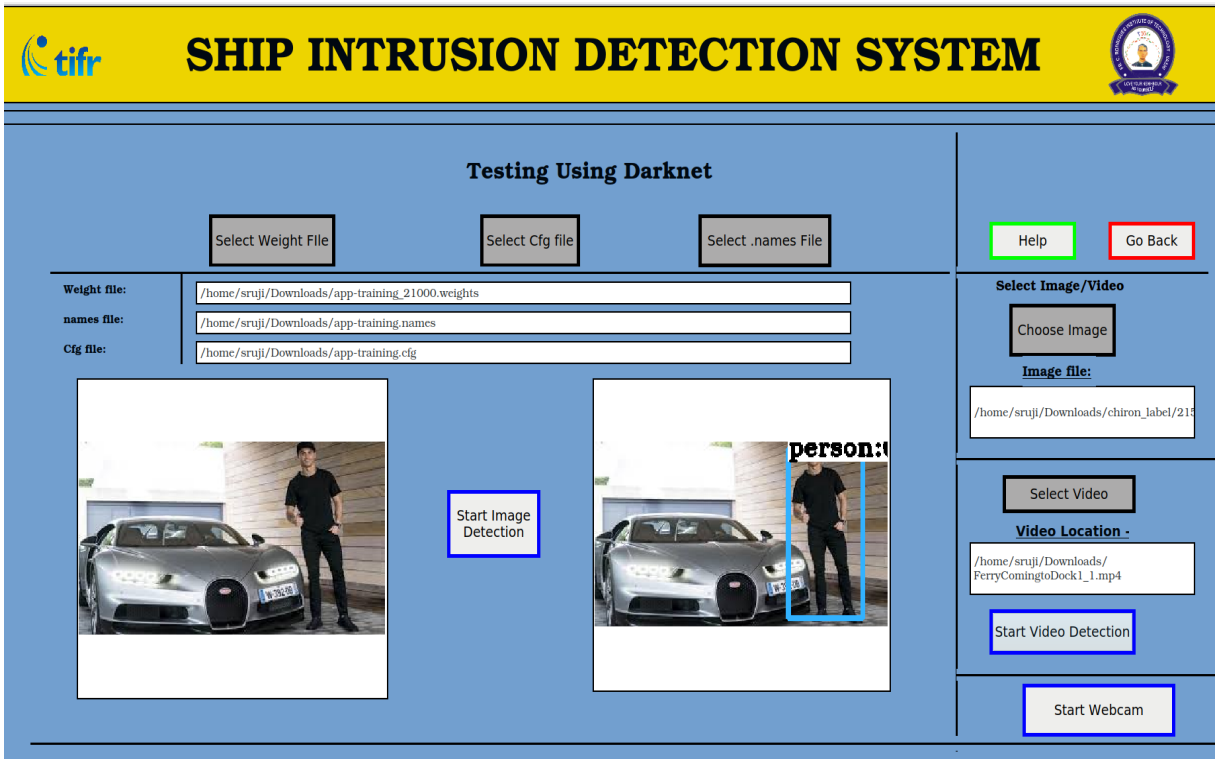
3.3.6 Testing / Detection Module

After we have trained the model, we now need to test it.Testing or detection can be done on images, videos, and webcam (if available) In order to test our custom trained object, we again need to provide a set of files. 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.

In the video detection module, the user can see the detection alongwith the related information like inference time which is the time taken to load a frame, the user can also see the objects detected and the frames skipped in case of fast-forwarding. As mentioned above, the user can fast-forward, pause and play on the video detection. The user can go to the default speed as well. It is recommended that the detection is done on GPUs for maximum inference time.

Figure 3.5: Testing/Detection Module



Chapter 4

Design Of the System

4.1 Requirement Engineering

4.1.1 Requirement Elicitation

Table 4.1: Requirement Elicitation

Sr.No	Type	Short Description
F1	Functional	User should be able to label Images
F2	Functional	The system can start training using Darknet
I1	Informational	The system displays the previous activity of the user
F3	Functional	User can split the data into training and testing
I2	Informational	The system consist of guides to help the user
F4	Functional	User can test the trained model on images/Video
I3	Informational	The system should display the status of training
F5	Functional	User can test using the Webcam
F6	Functional	User can register and sign up
I4	Informational	The system consist of guides to help the user
I5	Informational	The system should display the labelling progress
I6	Informational	The system should refresh the user session
F7	Functional	The user should be able to switch to any web camera

4.1.2 Software lifecycle model

The Software lifecycle model used was Agile Framework. The entire System was completed in an iterative manner wherein each phase modules were developed and at the end of the lifecycle are integrated. Reasons to use agile:-

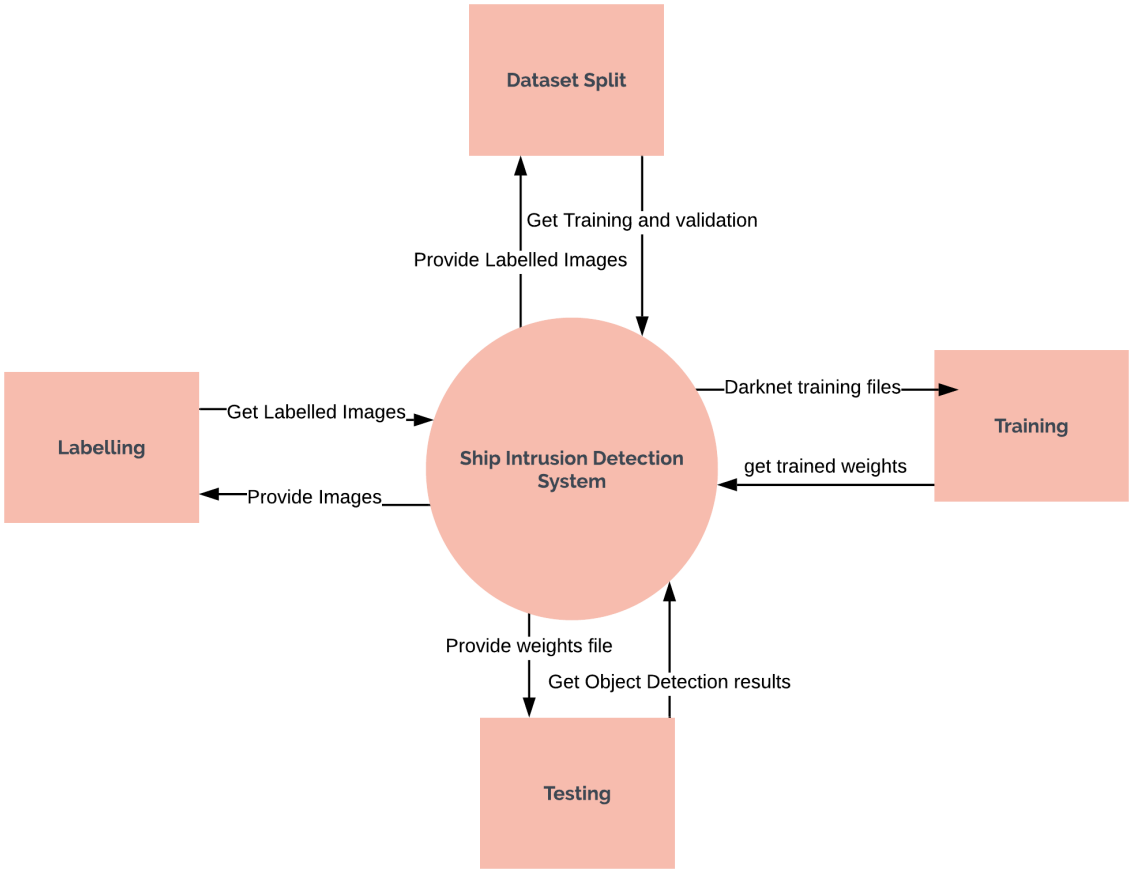
Faster Time to Market - Agile frameworks emphasize the Minimum Viable Product (MVP) and the Minimal Marketable Product (MMP). These are two Lean concepts that allow for a small, safe investment that can be delivered quickly.
Stakeholder Satisfaction Stakeholders are users, customers, or anyone that has a vested interest in the software. In Agile approaches, stakeholders are heavily engaged in the process and their feedback is constantly solicited. To ensure that we're adding value every step of the way, we ask the stakeholders their opinions during iterations.

Organizational Transparency Agile is a total, holistic process that creates organizational transparency and camaraderie. People are honest and work towards a common goal and achieve what they have set out to achieve. The political landscape is lowered. The open office environments that are encouraged flatten the organizational hierarchy and create a greater sense of community.

4.1.3 Requirement Analysis

4.1.3.1 Data Flow Diagram

Figure 4.1: Data Flow Diagram of the System



4.1.3.2 Use Case Diagram

Figure 4.2: Use Case Diagram of Overall System

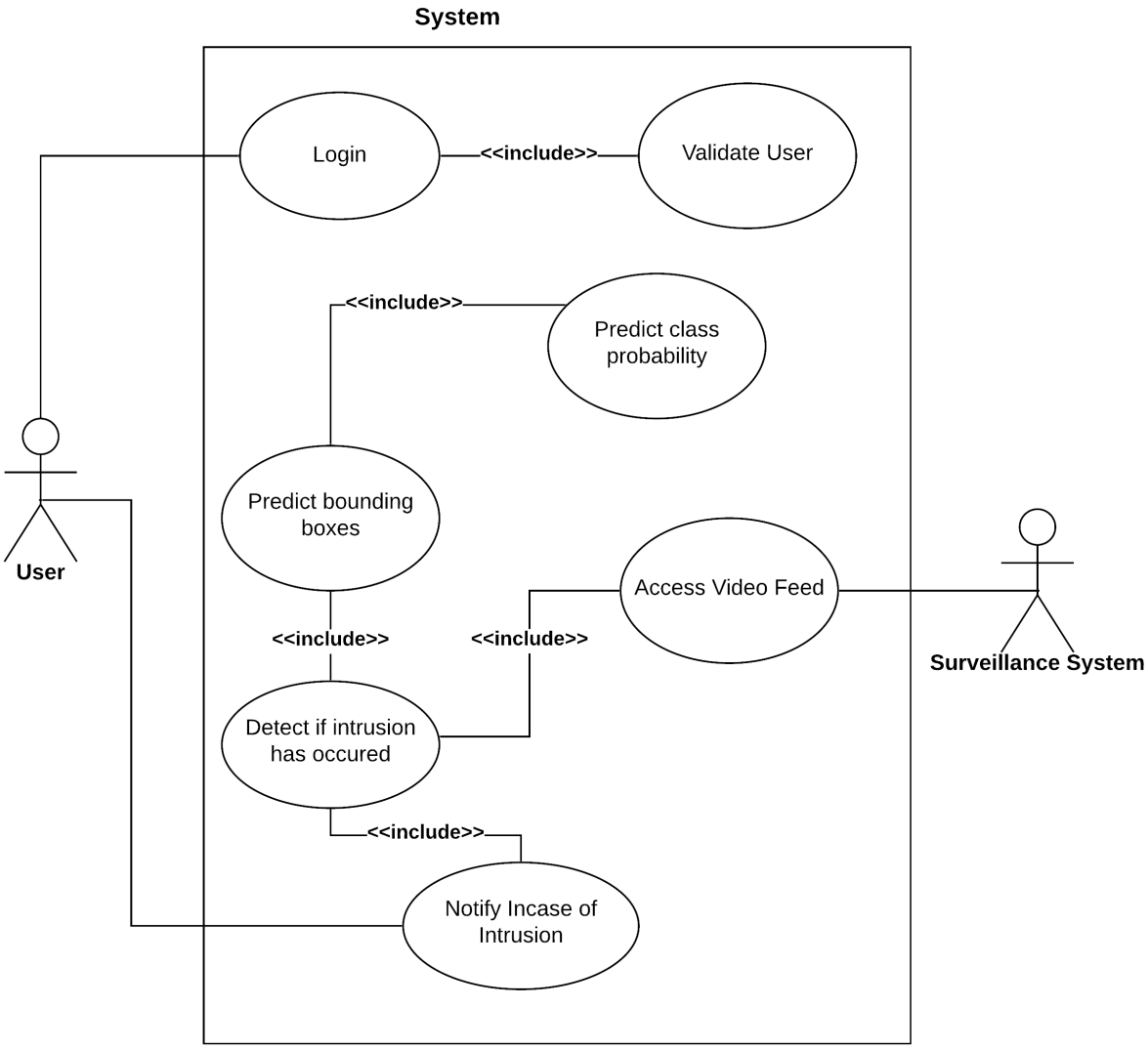


Figure 4.3: Use Case Diagram of Labeling Module

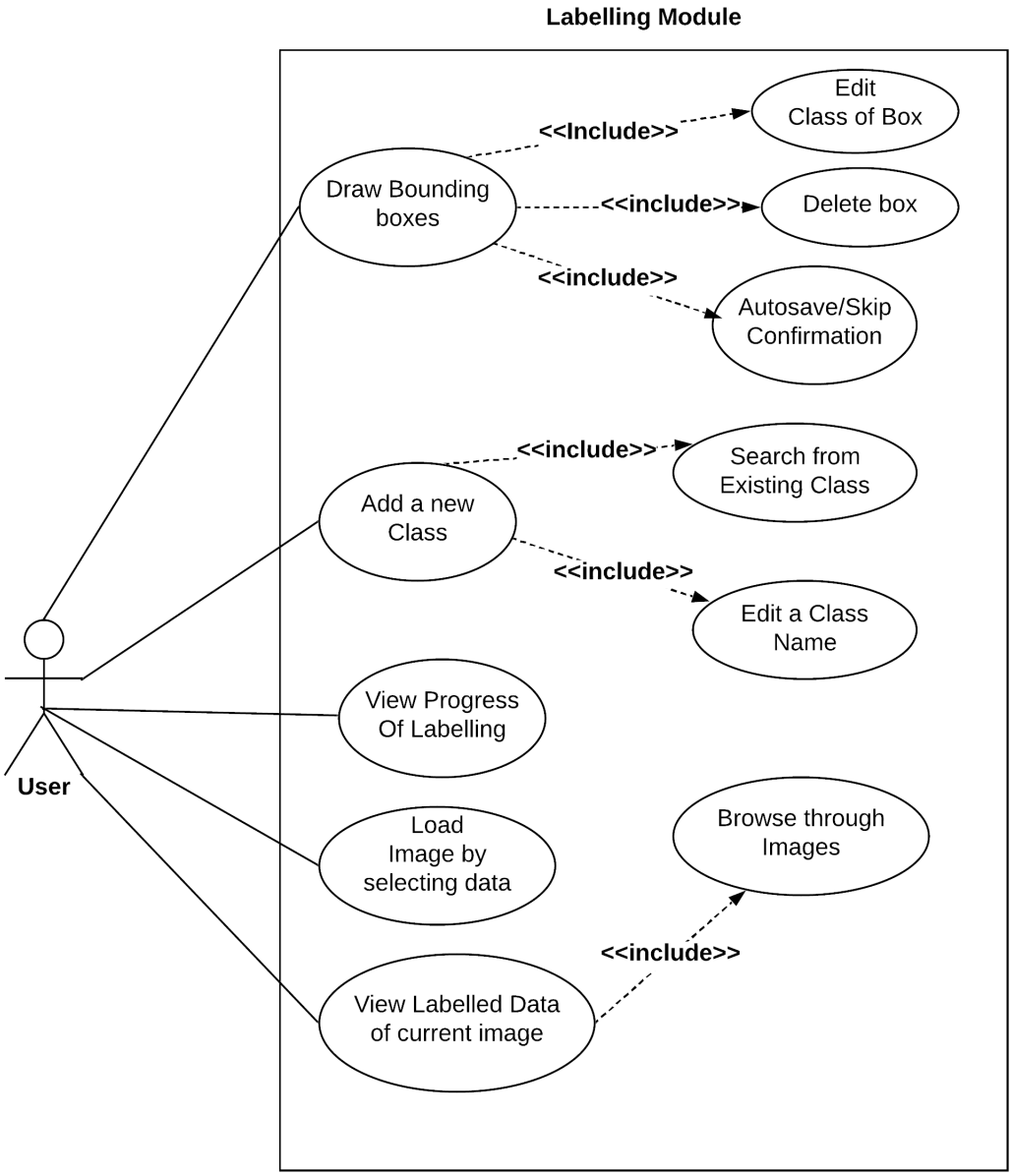


Figure 4.4: Use Case Diagram of Dataset Split Module

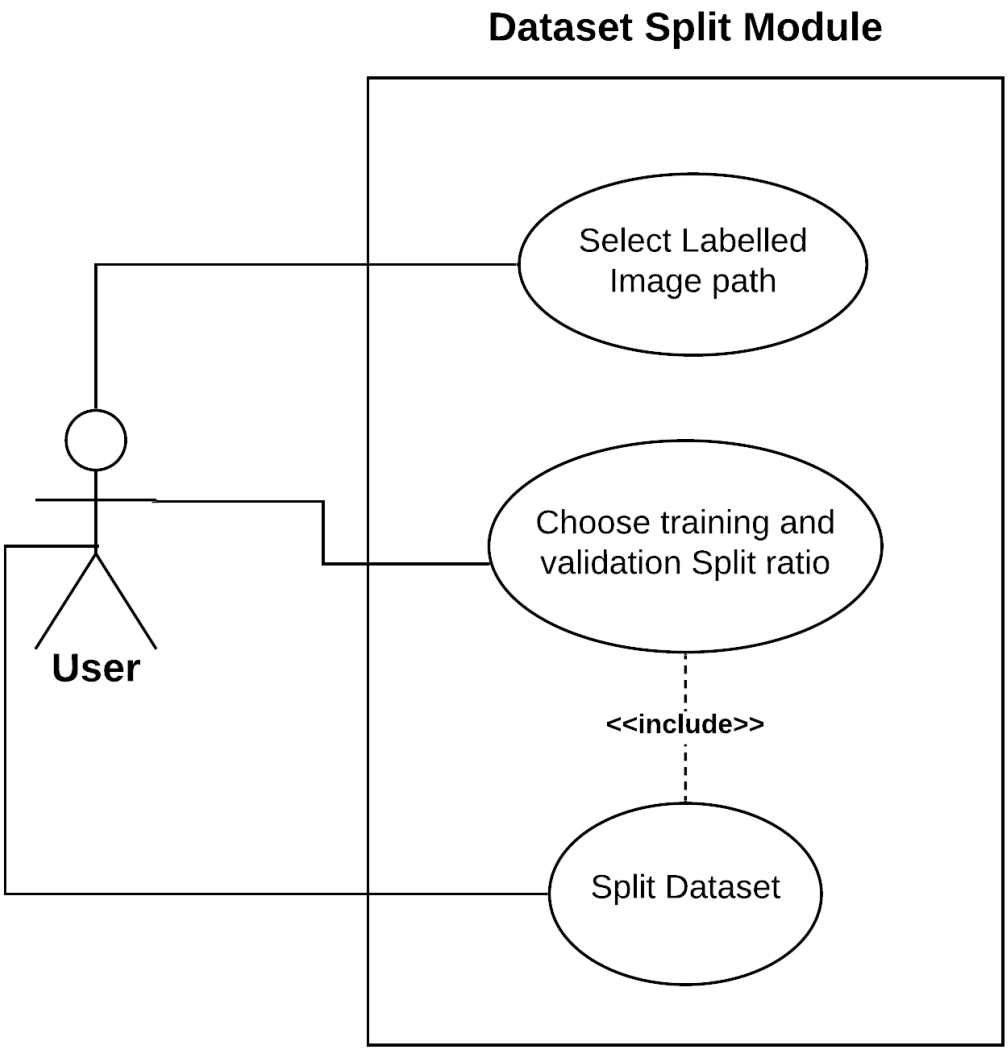


Figure 4.5: Use Case Diagram of Detection Module

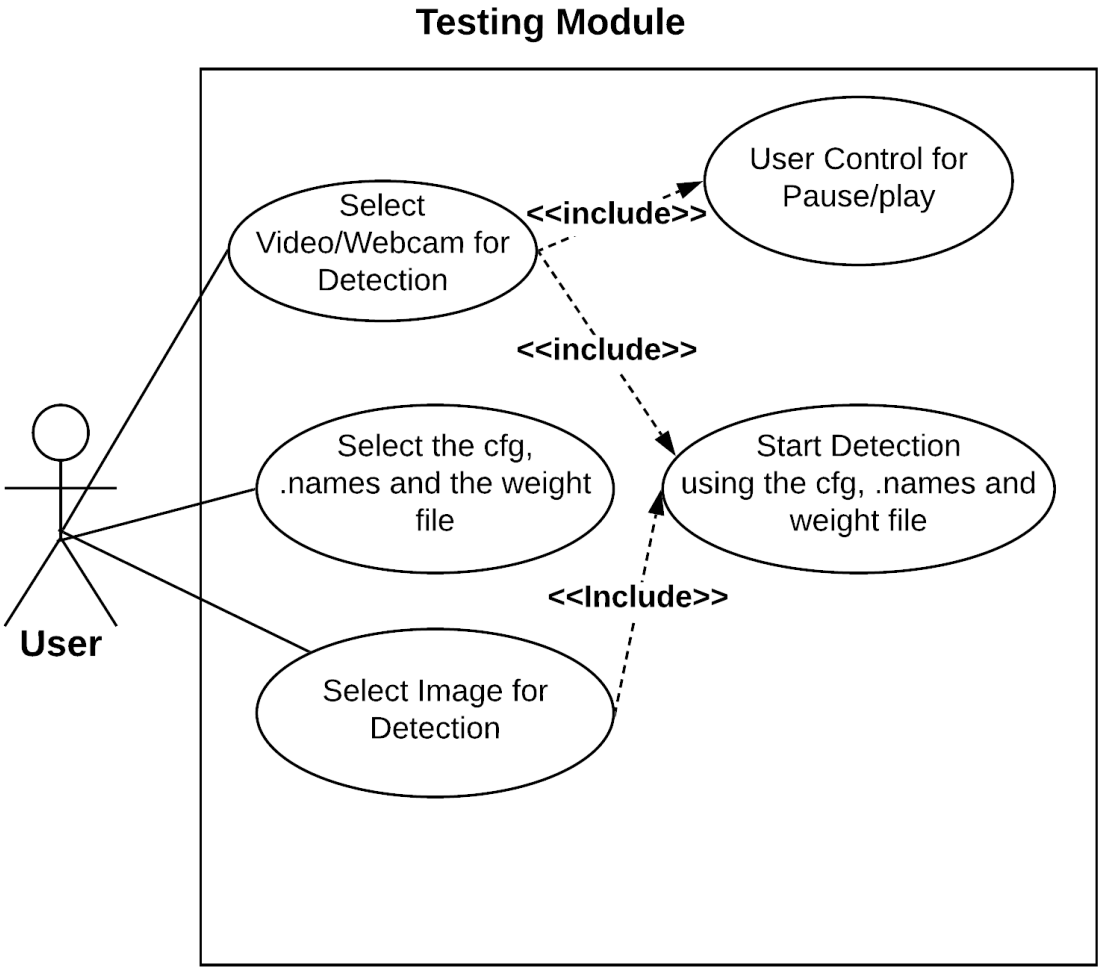
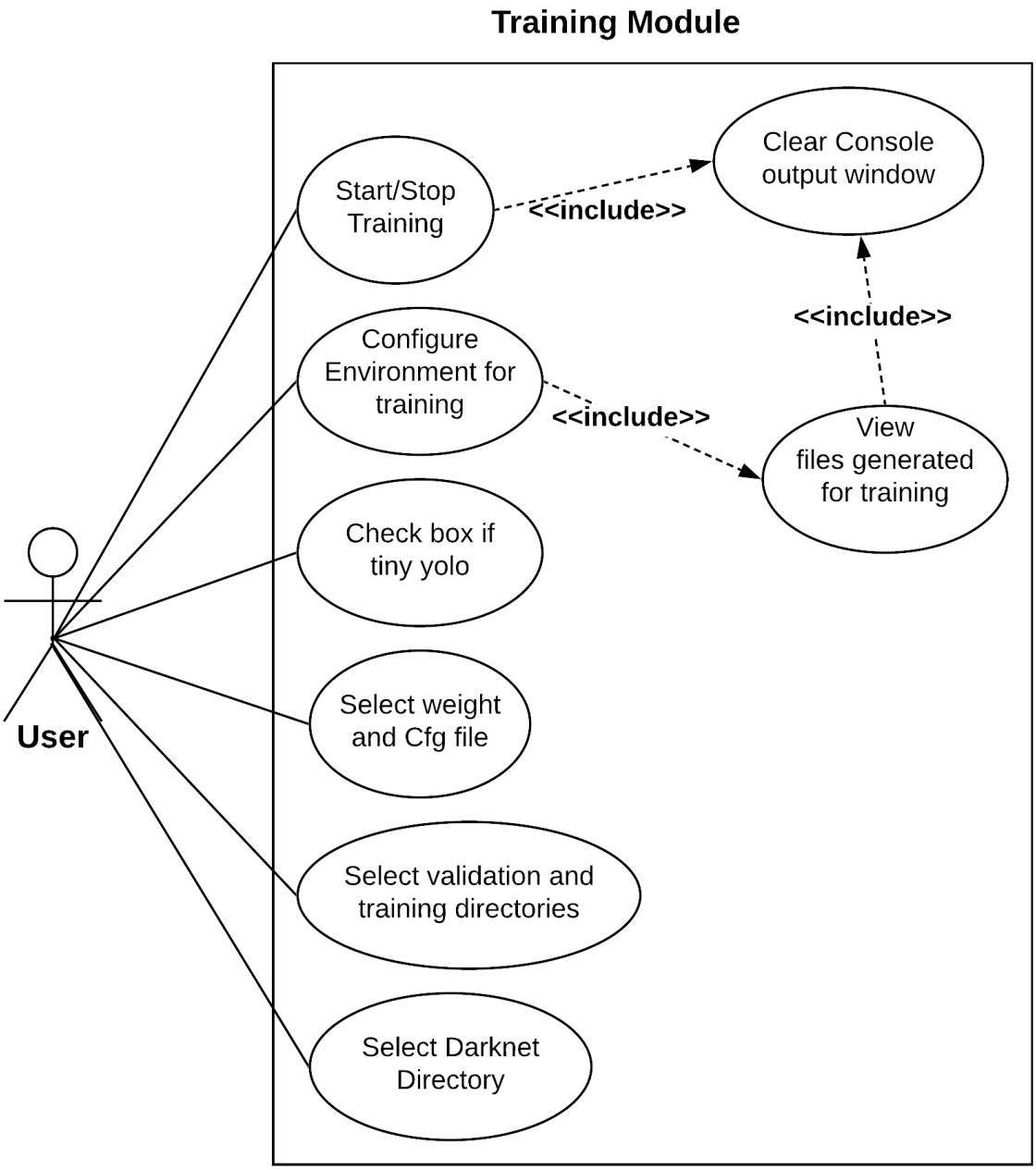


Figure 4.6: Use Case Diagram of Training Module



4.1.3.3 Cost Analysis

CCTV Cameras This system is reliant on images from the CCTV footages for detecting objects. So there is cost a that will be incurred for setting up the CCTV cameras of high resolution.

Training & Testing In order to train the Neural network, high configuration Computers with Graphic Processing Unit(GPU) will be required. Also in order to test or run the neural network on the video frames, GPU is

expected if higher frame rate is required to achieve

4.1.3.4 Software Requirements

- Operating System : Linux Ubuntu 18.04
- Front End : Qt Creator v5.7 and above
- Back-End : OpenCV 3 or 4
- C++ Installation
- SQLite

4.1.3.5 Hardware Requirements

- Master Computer at the security room with specifications:
  1. Intel Pentium IV Processor or above
  2. $\geq$ 2GB RAM
  3. $\geq$ 5 GB HDD Memory
  4. Internet Connectivity
- Surveillance Camera:
  1. Brand - Logitech
  2. Connection type - USB
  3. Video Resolution - 640 x 480
  4. Frame Rate - 30 fps
  5. Focus Type - Fixed

4.2 System architecture

4.2.1 UI/UX diagram

Figure 4.7: Login Menu user Interface

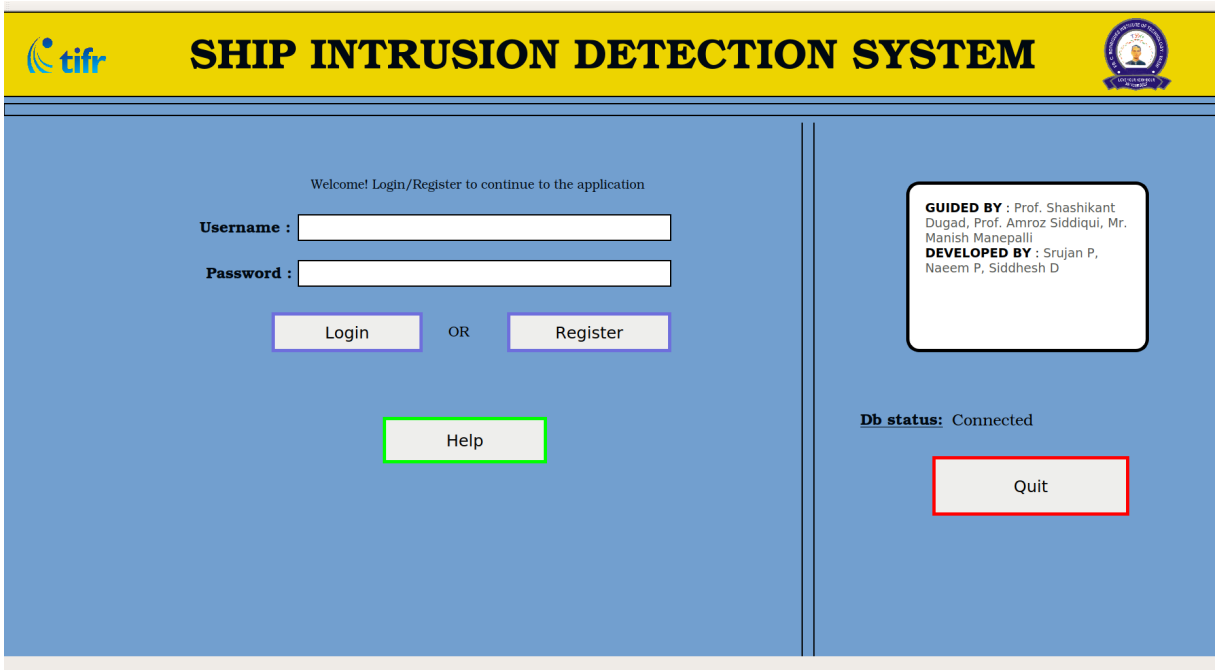


Figure 4.8: Main Menu user Interface

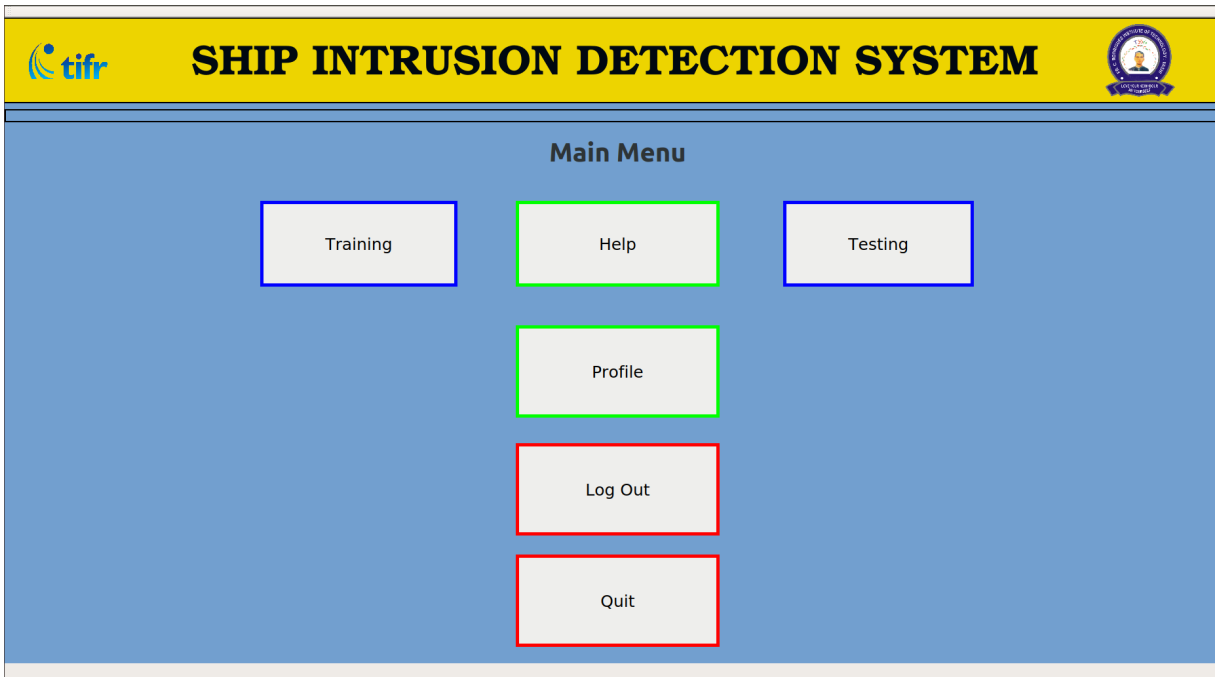


Figure 4.9: Data Split user Interface

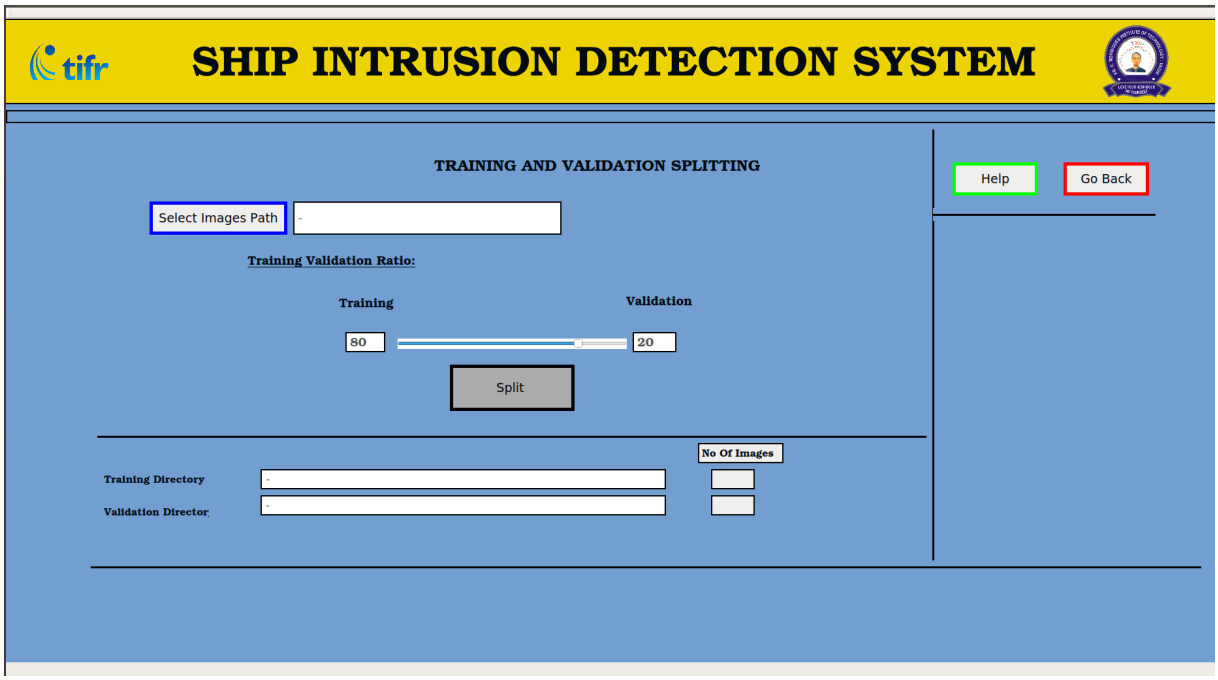


Figure 4.10: Training user Interface

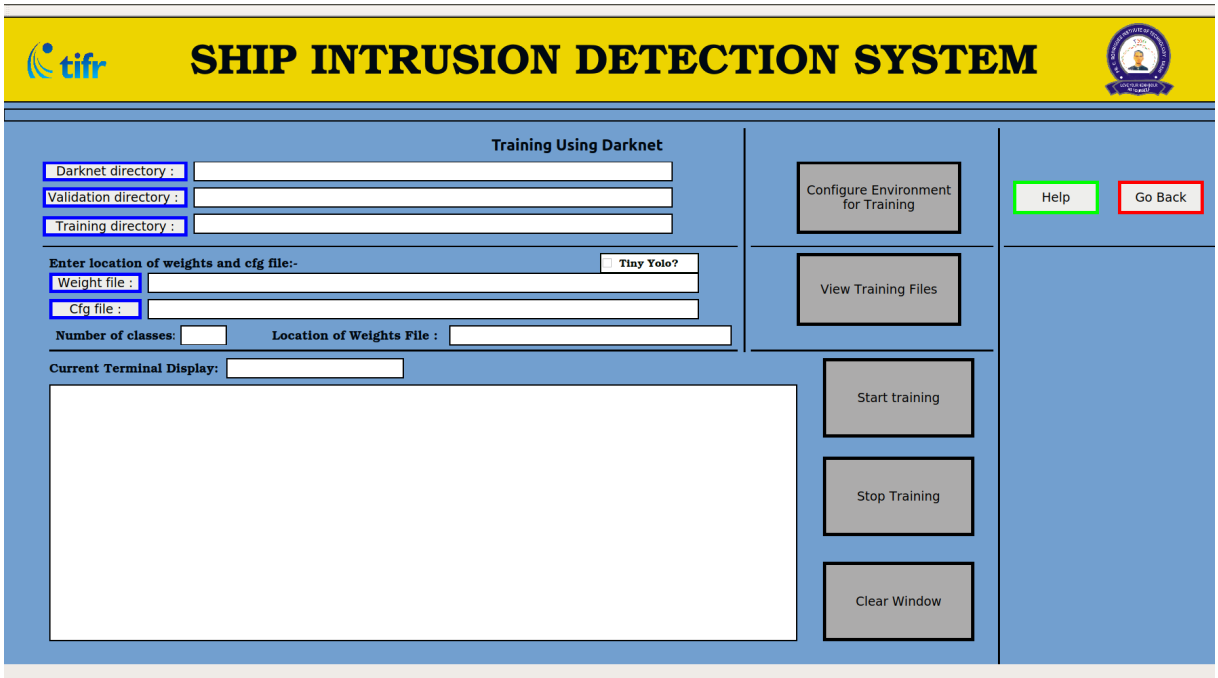
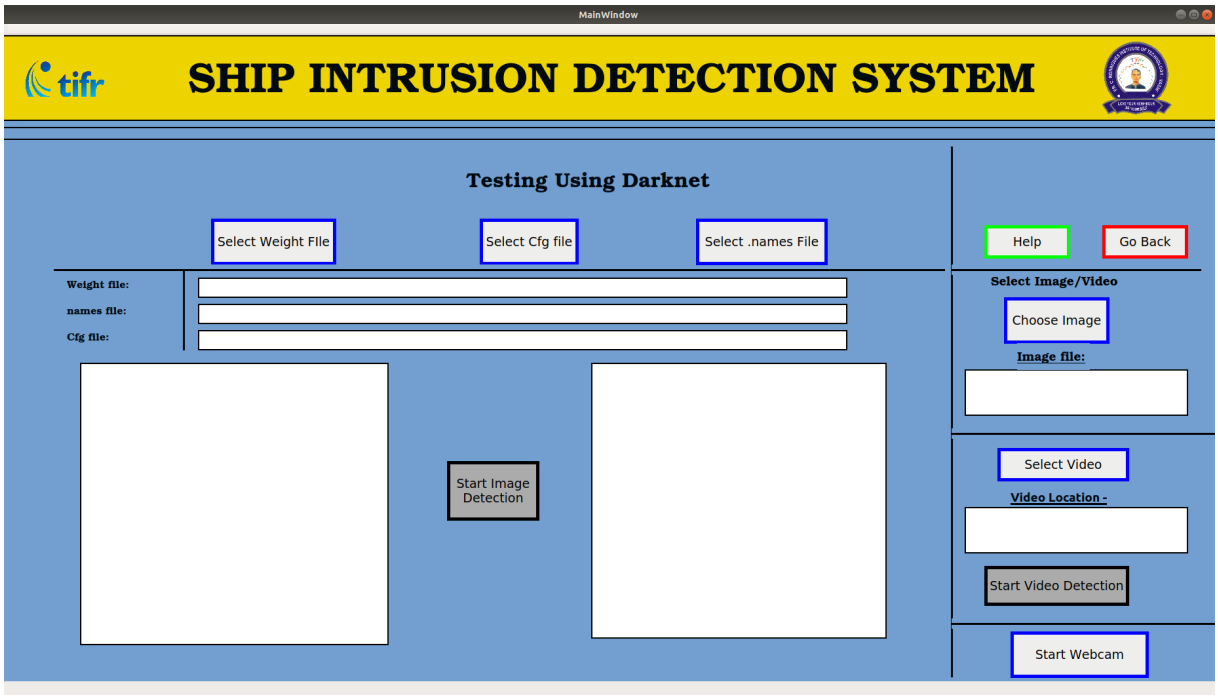


Figure 4.11: Testing User Interface



4.2.2 Block Diagram

Figure 4.12: Flow Diagram of YOLO Darknet Application

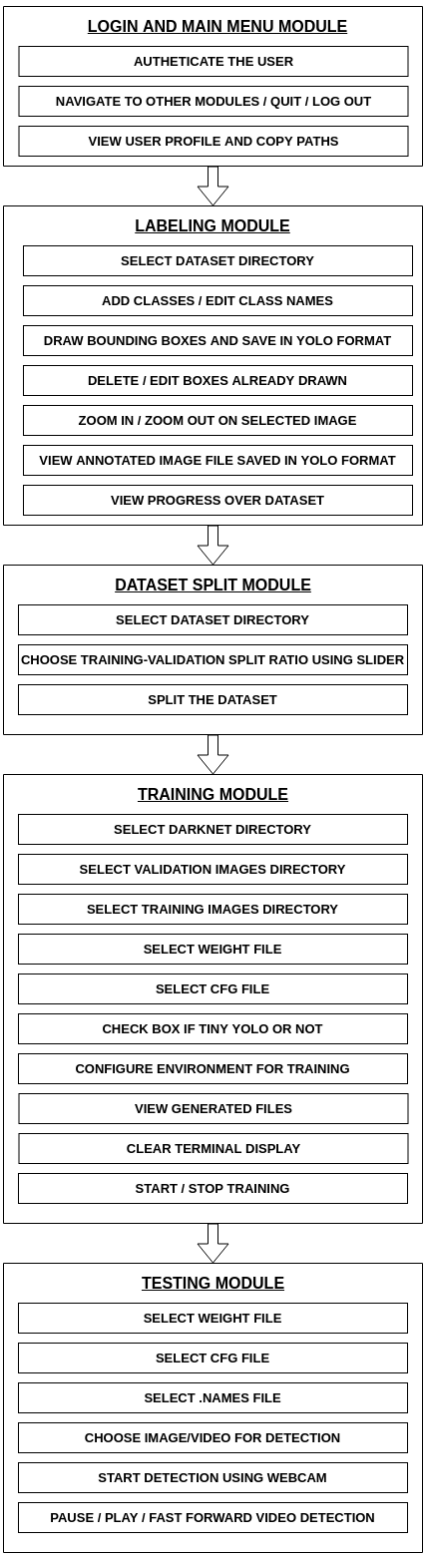
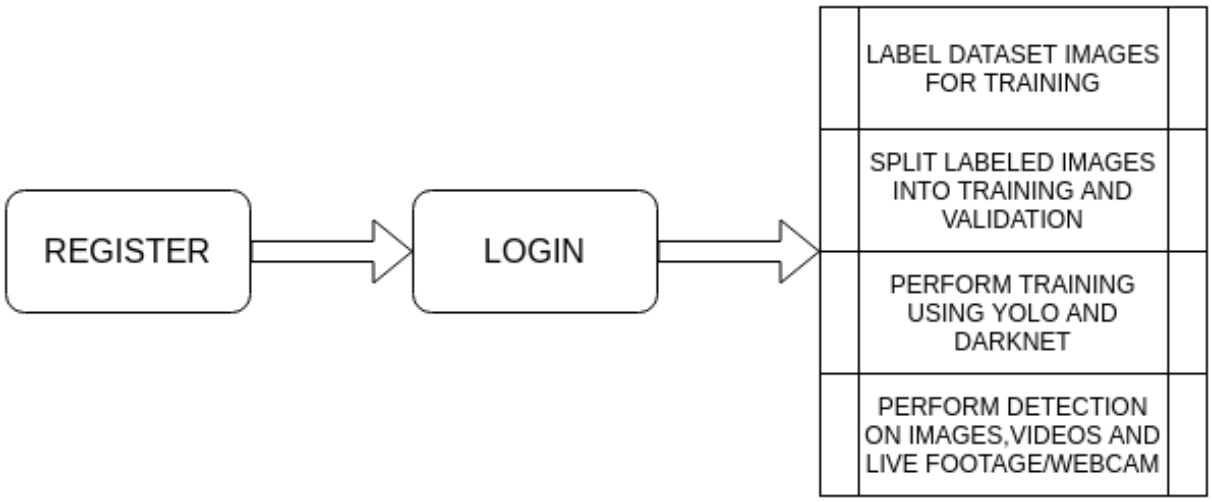


Figure 4.13: Block Diagram of YOLO Darknet Application



Chapter 5

Result and Discussion

5.1 Observations

Using our application, we created a ship and person intrusion detector. Most of the process was done using the application, but training was done on Google Colab due to lack of dedicated GPU. It took us about 2 to 3 days to create the detector without a dedicated good enough GPU but we estimate that this process can be finished in even lesser time given a dedicated GPU with good compute capability. Before using a GPU, CUDA should be installed and darknet should be installed with the GPU and CUDNN options set to 1. Online tutorials are available on how to do this.
Regardless, our process involved the following steps :

5.1.1 Collection of Data

Dataset collection is the first step, we need a dataset that has enough images to get a good detector. Furthermore, the dataset should have good variety for the classes that we want to build the detector for. Variety means images in which the classes are having different sizes, orientations, colors, etc. The more variety and size of the dataset, the more robust our detector is. There should also be some negative images which do not contain the classes.
For our process, we took the dataset from the COCO dataset along with some images scraped from the web.

5.1.2 Labeling/ Image Annotation

This is the next step in our procedure. This involves labeling our dataset and drawing bounding boxes around the classes in our image, our application provides a labeling module that does this efficiently and provides the users with many extra features to make the process as smooth as possible. Using our application the .txt files are generated that contain the image annotations in the YOLO format. The user can also view their progress through the application. For a large dataset, annotation process may be time consuming and using our application, we estimate that atleast 500-600 images can be annotated in a day.
We, however, used the annotations already given in the COCO dataset to generate the .txt files in the YOLO format which saved us some time.

Table 5.1: Dataset information for the detector

Total no. of images used	13221
Training images	12697
Validation images	524
Testing images	143
No. of classes	2

5.1.3 Dataset Splitting

This step involves splitting the dataset into training and validation sets. This can easily be accomplished using our application. The user can choose their ratio and simply click the split button to split the dataset. Ideally, 80-20 split is used but COCO recommends a larger training set as compared to validation and testing set. The table above gives some idea about the dataset and the split we used for our detector.

5.1.4 Training

The main part of building a detector is training. Training the neural network requires a lot of computations which are usually done using good NVIDIA GPUs. These are expensive and require additional packages or softwares installed for their use which require a proper environment. Instead of doing all this manually, we decided to use the GPUs that Google provides on their colab platform. The GPUs provided are powerful and all the necessary installations are easy to carry out. The downside is that the runtime is refreshed every 12 hours meaning that a lot of redundant work needs to be done. On top of that sometimes, colab causes the browser to crash and gets disconnected if the browser is inactive for more than 20-30 minutes. Anyhow, for lack of a better option, we used colab to train. We used the initial darknet53.conv.74 to start training using YOLO and other weight file for tiny yolo as well. The training statistics are provided in the following table.

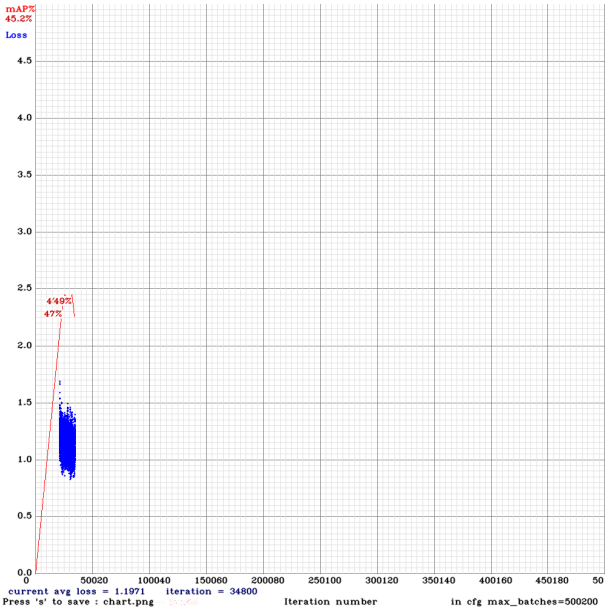
Table 5.2: Training statistics

	YOLO	**tiny-YOLO**
No. of iterations	34000	104000
Time taken (hours)	17.5	20.5
Average Loss	1.19	1.92
mAP(Mean Average Precision)	45.2 %	31 %

Darknet gives us a graph that visualizes the training process and helps us give some idea about what stage the training is at. If we are satisfied, then

we can stop the training or we can continue. It gives us the average loss which is the error (the less is better) and the mAP. Now, the AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1.mAP (mean average precision) is the average of AP. In some context, we compute the AP for each class and average them. But in some context, they mean the same thing. For example, under the COCO context, there is no difference between AP and mAP.

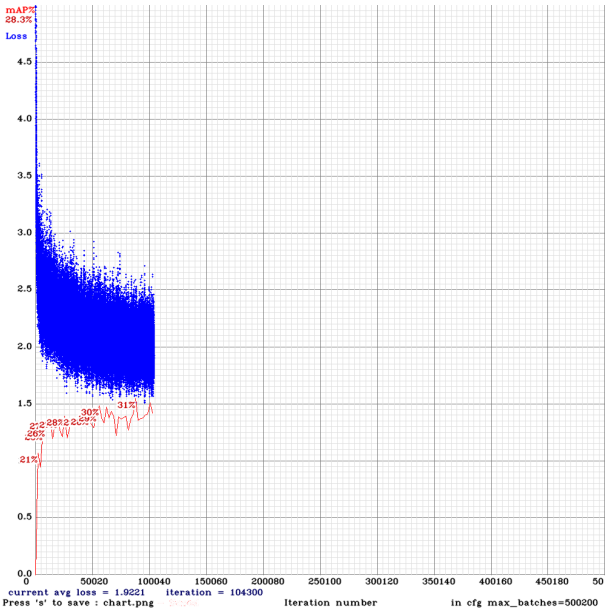
Figure 5.1: Training chart for YOLO



5.1.5 Testing / Detection

The detection module of the application can detect the trained classes with the help of the weights, .names and the .cfg (configuation) files. If any class is detected, a bounding box is drawn around the detected class and the confidence with which that object is detected is also displayed. In case of image detection the detected image is displayed with the initial input image provided. However, with webcam and video detection, a new page is displayed with more features like pause/play and fast-forward. In the video detection page, the inference time, the classes detected and the frames skipped (in case of fast-forwarding) is also shown. To gain a better idea of how our model works we use some metrics. To understand these metrics we need to know about certain things which are

Figure 5.2: Training chart for tiny-YOLO



- **True Positives (TP) :** These are the images which contain the required class and are predicted as having the required class. For eg, an image containing a ship was predicted as containing a ship.
- **False Positives (FP) :** These are the images which contain the required class and are predicted as not having the required class. For eg, an image having a ship was predicted as not having a ship
- **True Negatives (TN) :** True Negatives (TN): These are the images which do not contain the required class and are predicted as not having the required class. For eg, an image not having a ship was predicted as not having a ship.
- **False Negatives (FN) :** These are the images which do not contain the required class but are predicted as having the required class. For eg, an image not containing a ship was predicted as containing a ship.

With these definitions we can now go over the metrics that we have used for evaluating the trained model.

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

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ no.\ of\ predictions}$$

or,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

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.

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

The following table shows the values obtained for each of the metrics. These values have been obtained by testing on new images which the model has previously not seen. The values have been given for three models. The first one shows the values for the pre-trained YOLO model, those weights are available online and have been trained for about 500000 iterations. The next is our custom-trained model on YOLO which has been trained for 34000 iterations. And last one is the custom trained model on tiny-YOLO which has been trained for 104000 iterations. Note that the tiny yolo model is much faster and lightweight but gives a lower accuracy. The YOLO model can be improved if run for more iterations.

Table 5.3: Evaluation for different models

Metric	**YOLO (Pre-Trained)**	**YOLO (Custom-Trained)**	**Tiny-YOLO (Custom-Trained)**
Accuracy	94.37 %	82.52 %	68.06 %
Precision	98.77 %	98.39 %	97.67 %
Recall	91.95 %	71.76 %	48.28 %
F1-Score	95.24 %	82.99 %	64.62 %

5.1.6 Results on Videos and Images

Figure 5.3: Detection of Ships

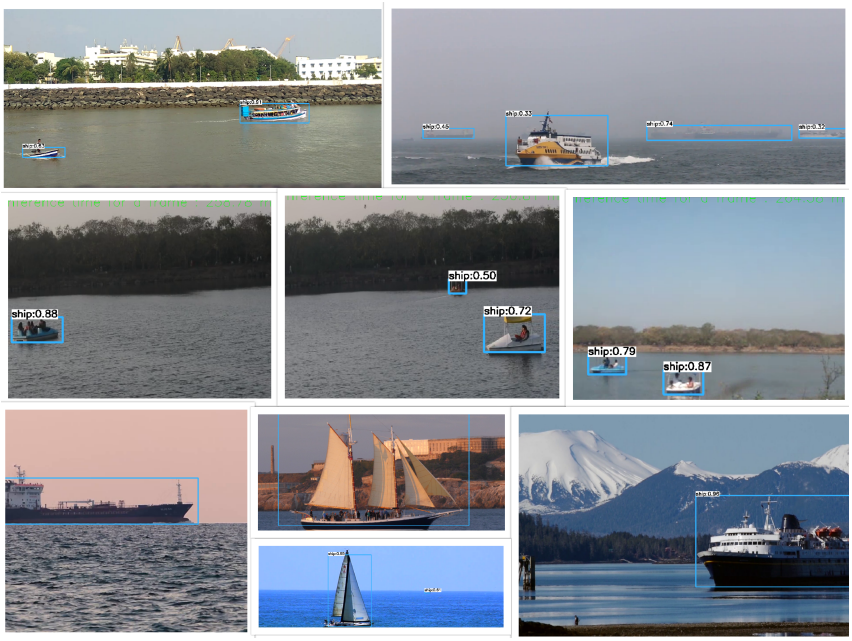
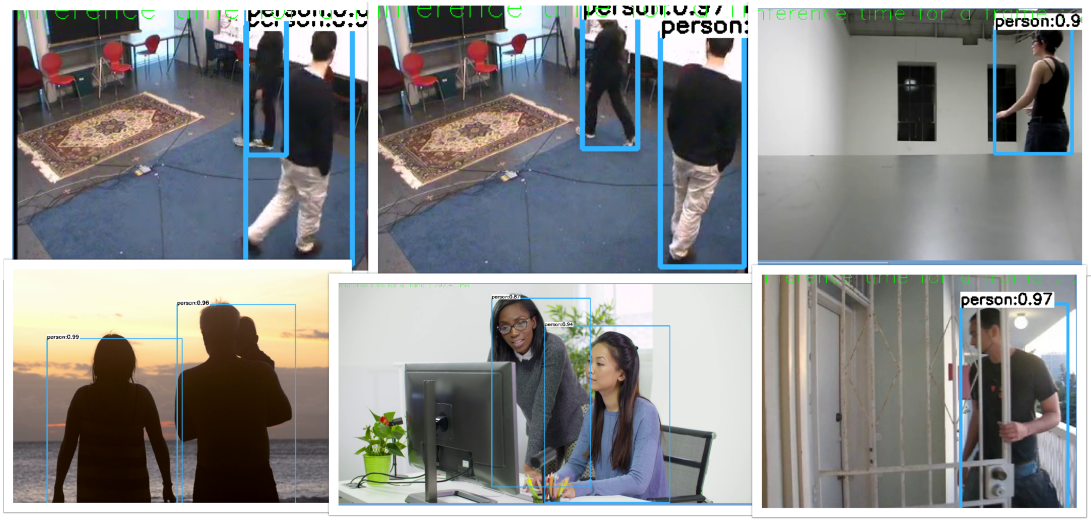


Figure 5.4: Detection of Person(s)



5.2 Important Code Snippets

5.2.1 Searching External Cameras

```
void Login::searchCameras()
{
    cv::VideoCapture temp_camera;
    int maxTested = 10;
    vector<int> cams;
    cams.push_back(0);
    for (int i = 1; i < maxTested; i++)
    {
      cv::VideoCapture temp_camera(i);
      bool res = (!temp_camera.isOpened());
      temp_camera.release();
      if (!res)
      {
        cams.push_back(i);
      }
    }

    QString avail_cam;
    vector<int>::iterator ptr;
    for(ptr=cams.begin();ptr<cams.end();ptr++)
    {
        avail_cam = avail_cam + QString::number(*ptr)
          + "\n";
    }

     while ( QLayoutItem* item = ui->verticalLayout_2
       ->layout()->takeAt( 0 ) )
         {

        //Q_ASSERT( ! item->layout() ); // otherwise
           the layout will leak

            delete item->widget();

            delete item;
        }
```

```
QVBoxLayout *buttonLayout = new QVBoxLayout;
QButtonGroup *grp = new QButtonGroup();

//radio button group for displaying the webcams
   available
for(int i = 0; i < cams.size(); i++)
{
        QRadioButton *btn = new QRadioButton("
           Camera " + QString::number(cams[i]));
        btn->setStyleSheet(" font: 75 15pt \"URW
           Bookman L\";");
        buttonLayout->addWidget(btn);
        grp->addButton(btn);
        grp->setId(btn, cams[i]);
 }
 buttonLayout->addStretch();
 this->selectCameraButtonGroup = grp;


 QWidget* scrollAreaContent = new QWidget;
 scrollAreaContent->setLayout( buttonLayout );
 QScrollArea* scrollArea = new QScrollArea;
 scrollArea->setHorizontalScrollBarPolicy( Qt::
    ScrollBarAlwaysOff );
 scrollArea->setVerticalScrollBarPolicy( Qt::
    ScrollBarAsNeeded );
 scrollArea->setWidgetResizable( true );
 scrollArea->setWidget( scrollAreaContent );


this->ui->verticalLayout_2->addWidget(scrollArea)
   ;
 //this->ui->verticalLayout->addLayout(
    buttonLayout);


}
```

5.2.2 Postprocessing Frames using Opencv

```
void postprocess(Mat& frame, const vector<Mat>& outs)
{
    vector<int> classIds;
    vector<float> confidences;
    vector<Rect> boxes;
    int test=0,b=3;
    if(test%b==0){
        for (size_t i = 0; i < outs.size(); ++i)
        {
        // Scan through all the bounding boxes output
            from the network and keep only the
        // ones with high confidence scores. Assign
           the box's class label as the class
        // with the highest score for the box.
        float* data = (float*)outs[i].data;
        for (int j = 0; j < outs[i].rows; ++j, data
          += outs[i].cols)
        {
            Mat scores = outs[i].row(j).colRange(5,
               outs[i].cols);
            Point classIdPoint;
            double confidence;
            // Get the value and location of the
               maximum score
            minMaxLoc(scores, 0, &confidence, 0, &
               classIdPoint);
            if (confidence > confThreshold)
            {
                int centerX = (int)(data[0] * frame.
                   cols);
                int centerY = (int)(data[1] * frame.
                   rows);
                int width = (int)(data[2] * frame.
                   cols);
                int height = (int)(data[3] * frame.
                   rows);
                int left = centerX - width / 2;
                int top = centerY - height / 2;

                classIds.push_back(classIdPoint.x);
```

```
                    confidences.push_back((float)
                        confidence);
                    boxes.push_back(Rect(left, top, width
                        , height));
                height1=height;
                width1=width;
                }
            }
            }
    }//b++;

    // Perform non maximum suppression to eliminate
        redundant overlapping boxes with
    // lower confidences
    vector<int> indices;
    NMSBoxes(boxes, confidences, confThreshold,
        nmsThreshold, indices);
    for (size_t i = 0; i < indices.size(); ++i)
    {
        int idx = indices[i];
        Rect box = boxes[idx];
        drawPred(classIds[idx], confidences[idx], box
            .x, box.y,
                box.x + box.width, box.y + box.
                    height, frame);
    }
}
```

5.3 Testing

Table 5.4: Testing Report

Functionality	**Test Cases Executed**	**Test Cases passed**	**Remark**
Sign Up in the Main Menu	10	9	Pass
Login in the Main Menu	10	10	Pass
Help Section in the Main Menu	8	9	Pass
Edit Option in Labelling	13	11	Pass
Drawing Bounding Box in Labelling	15	12	Pass
Generating the training files for Darknet	13	11	Pass
Viewing the progress bar in Labelling	11	11	Pass
Zooming feature in Labelling	14	12	Pass
Deleting Labelled boxes	17	16	Pass
Splitting labelled images into validation and Training	18	15	Pass
Displaying the object detected and the inference times for each frame in testing	20	18	Pass
Starting the training once all the necessary files are provided	19	16	Pass
Viewing the terminal output on the application			Pass
Clear the previous User session	20	19	Pass
Switching between web cameras	10	8	Pass

Chapter 6

Conclusion & Future Scope

Ship Intrusion Detection System is an automated video surveillance system developed such that it can be deployed and used in any given dynamic environment. The system developed is intelligent, user-friendly and robust. The system uses concept of deep neural networks and You only look once algorithm as classifier. A combination of Computer Vision and Machine Learning has been used in this system to detect the intrusions and to make it an accurate system. It is a user friendly system which has a flexible and easy to use interface. We understand that user of this system may not have technical expertise. Therefore the system is designed in such a way that the user is not burdened with the technical jargon and still be used with ease. The system is equipped with which features which aid the user in controlling the system and to help the user understand the system better. This system is designed in such a way that it can monitor any given area and detect any intrusions 24x7 for any environment. Thus giving a sense of security and reliability to the user.

References

[1] Don Stephens. Using intrusion detection on your security camera. https://www.cctvcameraworld.com/using-intrusion-detection-on-security-camera.

[2] Yuanyuan Wang, Chao Wang, Hong Zhang, Dong Yingbo, and Sisi Wei. A sar dataset of ship detection for deep learning under complex backgrounds. *Remote Sensing*, 11, 03 2019.

[3] Nannan Li, Xinyu Wu, Huiwen Guo, Dan Xu, Yongsheng Ou, and Yen-Lun Chen. Anomaly detection in video surveillance via gaussian process. *International Journal of Pattern Recognition and Artificial Intelligence*, 29:150426191333005, 04 2015.

[4] Shuxin Li, Zhilong Zhang, Biao Li, and Chuwei Li. Multiscale rotated bounding box-based deep learning method for detecting ship targets in remote sensing images. In *Sensors*, 2018.

[5] K. Anupriya and T. Sasilatha. *Ship Intrusion Detection System - A Review of the State of the Art: Second International Conference, ICSCS 2018, Kollam, India, April 19–20, 2018, Revised Selected Papers*, pages 147–154. 09 2018.

[6] T. N. Hannevik, Ø. Olsen, A. N. Skauen, and R. Olsen. Ship detection using high resolution satellite imagery and space-based ais. In *2010 International WaterSide Security Conference*, pages 1–6, Nov 2010.

[7] MathWorks Inc. https://in.mathworks.com/discovery/object-detection.html.

[8] Caglar Gulcehre. http://deeplearning.net/.

[9] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, Yuanyuan yu, Gary Bradski, Andrew Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. volume 19, pages 281–288, 01 2006.

[10] Hongping Cai, Qi Wu, Tadeo Corradi, and Peter Hall. The cross-depiction problem: Computer vision algorithms for recognising objects in artwork and in photographs. *ArXiv*, abs/1505.00110, 2015.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[13] Abdulbasit Alazzawi, Husam Alsaadi, Abidaoun Shallal, and Saad Albawi. Edge detection-application of (first and second) order derivative in image processing. 12 2015.

Acknowledgement

Success of a project like this involving high technical expertise, patience and massive support of guides, is possible when team members work together. We take this opportunity to express our gratitude to those who have been instrumental in the successful completion of this project. We would like to appreciate the constant interest and support of our mentor **Prof. Shashikant Dugad** in our project and aiding us in developing a flair for the field of Computer Vision and Machine Learning. We would always cherish the journey of transforming the idea of our project into reality. We would like to show our appreciation to **Mr Amroz Siddhique** for his tremendous support and help, without whom this project would have reached nowhere. We would also like to thank our project coordinator **Mrs. Rakhi Kalantri** for providing us with regular inputs about documentation and project timeline. A big thanks to our HOD **Dr. Lata Ragha** for all the encouragement given to our team. We would also like to thank our principal, **Dr. S. M. Khot**, and our college, **Fr. C. Rodrigues Institute of Technology, Vashi**, for giving us the opportunity and the environment to learn and grow.

Project Group Members:

1. Srujan Patel, 101641

_____

2. Naeem Patel, 101640

_____

3. Siddhesh Deshpande, 101612

_____

Appendix A : Timeline Chart

Figure 6.1: Timeline chart for project