International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 01 | Jan 2021 www.irjet.net p-ISSN: 2395-0072

Ship Intrusion Detection using Custom Object Detection System with YOLO Algorithm

Srujan Patel[1], Naeem Patel[1], Siddhesh Deshpande[1], Amroz Siddiqui[2]

[1]*Student, Computer Engineering Dept., Fr. C. Rodrigues Institute of Technology, Vashi, Navi Mumbai, Maharashtra, India*
[2]*Assistant Professor, Computer Engineering Dept., Fr. C. Rodrigues Institute of Technology, Vashi, Navi Mumbai, Maharashtra, India*

Abstract - *Due to the growth in the field of computer vision and deep learning which can be attributed to theoretical innovations as well as an improvement in hardware capabilities, automating tasks such as surveillance has become an area of great relevance which can leverage advancements in Artificial Intelligence to improve upon existing systems. One specific example is pertaining to the Tata Institute of Fundamental Research (TIFR) which is located on the shores of the Arabian Sea in Mumbai, India. 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 as well as people. This system uses Image Processing and Deep 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. On top of that, a secure system should be able to, with reasonable performance, detect various kinds of threats which can come in all sorts of shapes and sizes. Thus, detecting custom objects is also a useful feature of our system. Finally, the system should be easy to use and providing a good UI (user interface) has been a main goal in developing this system. We present this along with some stats regarding the training process and detection results on two classes, namely person and ship.*

Key Words: **Computer Vision, Machine Learning, Deep Learning, Convolutional Neural Networks, Ship Intrusion, YOLO**

1. INTRODUCTION

Before the advent of object recognition and automatic detection systems, surveillance systems relied on human supervision and even now, security systems haven't evolved to completely eliminate human attention. Computer Vision has one of the aims of achieving human level capabilities in seeing the world. In achieving this aim through artificial intelligence, a substantial amount of research has been done that takes a connectionist approach to this task. Deep Neural Networks, in particular, Convolutional Neural Networks have proven to be really adept at the task of object detection which is at the core of a surveillance system. Apart from the exclusivity of certain advanced technologies like Radar and Lidar to the military, using Deep Learning is fairly simple owing to its open-source nature. There are many implementations available of various algorithms along with necessary resources that expedites the task of creating a neural network based object recognition system. By nature, neural networks are mostly black boxes that are mostly invariant to the task and perform equally well on other disparate tasks. Surveillance is crucial not just in security, but also in hospitals for medical patients [1], in sports for detecting players [2] and in agriculture as well to detect anomalies the fields [3] to name a few examples. The ideal state of a surveillance system is one that involves least human intervention and notifies or signals an alarm once an intrusion is detected. These systems can use a variety of techniques like signal & image processing, computer vision, machine learning, etc. Having the right infrastructure in terms of hardware computing power, mostly GPUs (Graphics Processing Units) increases the system throughput and drastically reduces the risks of failure, proving to be a good investment. The ability of various deep learning algorithms and neural networks to utilize the GPUs to speed up computations is a huge benefit to reduce system latency [4]. There is no need of manually extracting the features because deep learning models are already trained using large sets of labelled data and neural network architectures which learn features from the given data. The availability of these deep learning resources has sparked a new method of creating an object detector known as transfer learning [5] which enables the development of these detectors without the need for immense amounts of data allowing the use of these techniques to a wide variety of applications. The main motivation in building this system with TIFR, Colaba is the fact that it is located on the shores and is at risk of intrusion by sea in water vessels. So, there is a need to put a system in place that has high reliability and is robust enough to solve this problem. Currently, Closed Circuit Television (CCTV) is used which are quite efficient but require constant human monitoring. There are limitations in the ability of humans to constantly monitor surveillance of live footage [6]. Another reason for automating this is the fact that sometimes the cameras may face resolution issues which may alter the appearance of the objects making them harder to detect.

Training humans to do this is not only expensive but may not be completely robust [7]. Comparatively, training an algorithm or a system to evolve with changing needs is easier and more cost-effective. A custom object detection system consists of 4 major steps which are image annotation, dataset splitting, training and testing/detection on videos, images or live feed. Traditionally, a user would have to perform all these tasks by employing a diverse range of methodologies and an array of different softwares. Additionally, they would need to manually edit the related files and without any simple GUI (graphical user interface). Our main contribution would be to present an end-to- end system that does all these 4 tasks in one place with just a few dependencies and a well-rounded GUI to automate as many tasks as possible.

2. LITERATURE SURVEY

2.1 Existing Systems

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 [7]. CCTV systems that require manual human attention clearly lack complete security but if coupled with automated systems and additional modules that improve the reliability, could be a good solution for surveillance systems. In fact, CCTV systems have been employed to good use for traffic monitoring. [8] demonstrates a pedestrian monitoring system. Further the use of face recognition systems through CCTVs [9] encourage the need to build intrusion detection systems for other use cases as well (ships in our case).
2) Radar: RADAR (Radio Detection and Ranging) method is used to detect the intrusion of ships [10]. The background of seashore landscape Synthetic Aperture Radar (SAR) image is dark and targets are bright making it is easy to detect the ship [11]. 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. Furthermore, installation of these systems require a fairly large investment and these systems are more restrictive too in terms of viewing angles. Also, the complexity of these systems provide little scope of applying the latest computer vision based algorithms which are gaining popularity for surveillance tasks.
3) Wireless Sensor Networks: Using signal processing techniques and cooperative signal processing, detection of any passing ships can be done by distinguishing the ship-generated waves from the ocean waves. A three-tier intrusion detection system which can exploit spatial and temporal correlations of an intrusion can be used to increase detection reliability. The main limitations of these systems are that they require relatively dense networks, especially to achieve higher detection accuracy with small boats because of the high noise of the sea [12].
4) Phase Saliency Map and extended wavelet transform: This method for ship detection is based on phase saliency map and extended wavelet transform (PSMEWT) to solve two issues: Accurate detection of ships in complex background and how to detect accurate ship targets in the event that ship targets are comparable to false alarms. In this method, multi-spectral information is first adopted in the sea–land segmentation to reduce the detection time consumption. Then, a visual phase saliency map based on the extended wavelet transform is constructed to highlight the difference between ships and the background to locate ship candidates. In the process of removing false alarms, aside from morphological and geometric features, other advanced methods are used to more effectively eliminate false alarms, and the SVM classifier is adopted to conduct offline training at the same time. However, there will be relatively more false targets when this method is applied to small ships. The small size (only 10–20 pixels) of the ships in the images make it difficult to determine whether it is a ship through optical remote sensing images [13].

2.2 Our Approach

1) Object Detection and Computer Vision: Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. [14] 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. Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do [15]–

[17]. 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.

2) Deep Learning: Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. Because the computer gathers knowledge from experience, there is no need for a human computer operator formally to specify all of the knowledge needed by the computer. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones; a graph of these hierarchies would be many layers deep hence the name "deep" learning [18]. Over the last years deep learning methods have been shown to outperform previous state-of-the-art machine learning techniques in several fields, with computer vision being one of the most prominent cases. The ambition to create a system that simulates the human brain fueled the initial development of neural networks. In 1943, McCulloch and Pitts [19] tried to understand how the brain could produce highly complex patterns by using interconnected basic cells, called neurons. The McCulloch and Pitts model of a neuron, called a MCP model, has made an important contribution to the development of artificial neural networks. These include LeNet [20] and Long Short-Term Memory [21], leading up to today's "era of deep learning." One of the most substantial breakthroughs in deep learning came in 2006, when Hinton et al. [22] introduced the Deep Belief Network, with multiple layers of Restricted Boltzmann Machines, greedily training one layer at a time in an unsupervised way. Guiding the training of intermediate levels of representation using unsupervised learning, performed locally at each level, was the main principle behind a series of developments that brought about the last decade's surge in deep architectures and deep learning algorithms [23].

3) Convolutional Neural Networks: Convolutional Neural Networks (CNNs) were inspired by the human visual system's structure, and by the models of it proposed in [24]. The first computational models based on these local connectivity between neurons and on hierarchically organized transformations of the image are found in Neocognitron [25], which describes that translation invariance is achieved when neurons with the same parameters are applied on patches of the previous layer at different locations, Yann LeCun and his collaborators later designed Convolutional Neural Networks employing the error gradient and attaining very good results in a variety of pattern recognition tasks [23], [26]–[28]. A CNN comprises of three main layers: convolutional layers, pooling layers and fully connected layers as demonstrated in Figure 1.

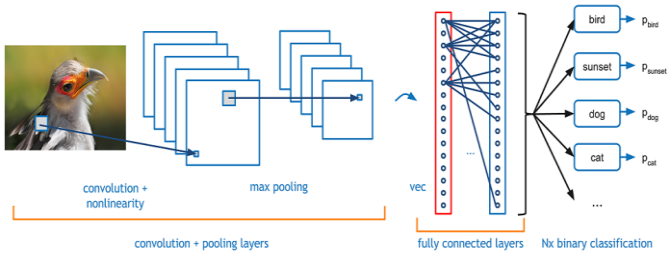

Fig -1: Example of Convolutional Layers

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 than other algorithms. 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. The one we used, YOLOv3 [29] is really powerful with a good tradeoff between accuracy and speed. There is also another version of YOLO known as 'tiny-YOLO' which is a scaled down version of YOLO which gives faster results on low-end systems with slightly worse results. The benefits of YOLO have been applied to various applications due to the availability of pre-trained weights trained on the COCO dataset and reasonable training times. For instance, [30] have demonstrated the use of YOLO in counting traffic. [31] have used the YOLO algorithm to monitor the growth of apples in orchards. This demonstrates the viability of YOLO as being a reliable object detection algorithm in diverse applications. The architecture of the network used in YOLO is shown in Figure 2.

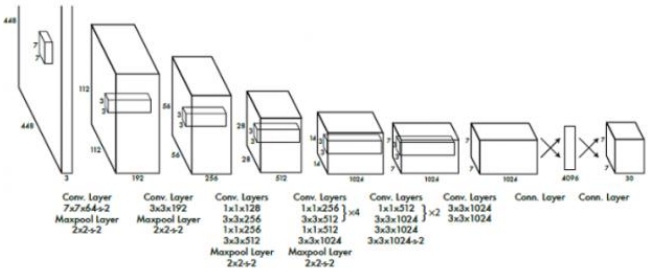

Fig -2: YOLO Architecture

3. SYSTEM DETAILS

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. The main goal of the system is to provide a workflow to the user with a GUI for performing the task necessary in building a custom object detector using the YOLO algorithm and using it for detection on images, videos and live footage. The system should allow the user to annotate images for training, then split it into training and validation sets, then train it and finally obtain weights that will help in detection. For our specific use case we have chosen to work on detection of ship and person. A high-level block diagram of the system is illustrated in Figure 3. In terms of implementation, we have used the Qt framework for C++ to develop our software. We have also used the OpenCV [32] library for image manipulation related tasks. The main OS used by us was Ubuntu but the application can be made to work in any Unix-like environment. For distribution, we used Appimages to package our software. In particular, linuxdeployqt. The main modules that the system is divided into will be elaborated upon in the following sections.

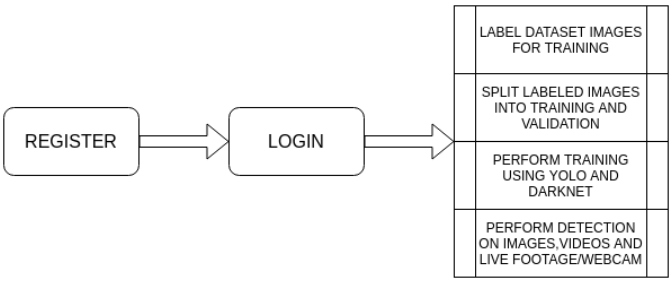

Fig -3: Block Diagram

3.1 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.

3.2 LABELING/IMAGE ANNOTATION MODULE

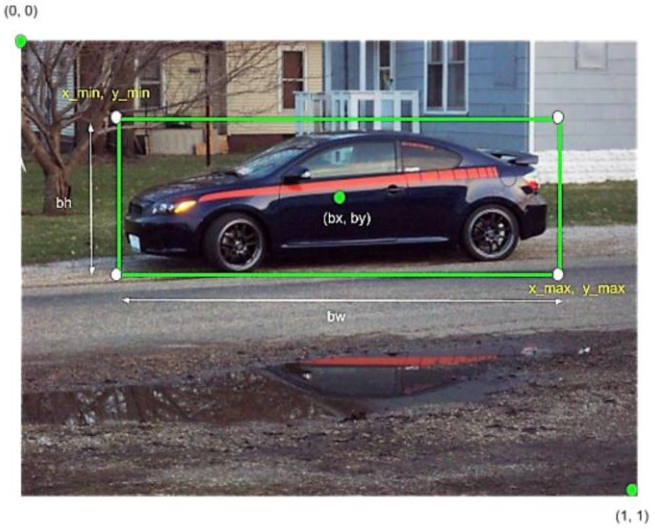

Fig -4: The associated .txt file will have the following contents (if car is class 0): 0 0.0833 0.0833 0.0556 0.0417

Although there are many tools available for annotating images like [33]. These need to be installed separately and there are some features that they lack. To overcome this and integrate the workflow for custom object detection, we came up with our own implementation of an image annotation tool. 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)

An example of how the annotations are stored is illustrated in Figure 4. 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. 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. Drawing bounding boxes is intuitive with the use of mouse. The app will ask for confirmation every time a bounding box is drawn. To skip this, the user can check the" Auto-save" 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.

3.3 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 use the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly. Splitting the dataset into two folders (training and validation) manually requires copy pasting the images through the OS or the command line which is again a bit tedious. Since, the separation should be random for optimum results, the process is again complicated to select certain images. For solving this, we have added this module which randomly splits the dataset and the user has to just give the ratio of training to validation images. 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 their requirement. This module does not move the images, it copies the images into training and validation subdirectories inside the main dataset directory.

3.4 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. Without this part, the user would have to manually modify necessary files and configure the environment and give command line instructions to start training a model. All this is automated with this module. To train the network, there 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:

- 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. Installation instructions are available online.
- 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.
- 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.
- 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.

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 separate 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.

3.5 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 along with 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 minimum inference time.

4. RESULTS AND DISCUSSION

Using our application, we created a ship and person intrusion detector. All the processes were done using our application except training which 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 steps discussed in the following sections.

4.1 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 [34] along with some images scraped from the web. The details about our dataset are given in Table 1.

4.2 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 at least 500-600 images can be annotated in 2 hours. 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 -1: Dataset Information for Detector

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

4.3 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. In- stead of doing all this manually, we decided to use the GPUs that Google provides on their cloud named 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 Table 2. 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 contexts, we compute the AP for each class and average them. But in others, they mean the same thing. For example, in the context of COCO dataset, there is no difference between AP and mAP.

Table -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%

4.4 TESTING/DETECTION

The detection module of the application can detect the trained classes with the help of the weights, .names and the .cfg (configuration) 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 used some metrics. To understand these metrics, we use these measures:

1) True Positives (TP): Images with objects present and detected correctly by the model.
2) False Positives (FP): Images not containing objects but objects detected by the model.
3) False Negatives (FN): Images with objects which are not detected by the model.
4) True Negatives (TN): Images not containing objects and not detected by the model.
5) Accuracy: It is the fraction of predictions that the model got right. Formally,

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

Or,

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

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

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

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

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

8) F1 score: F1 score conveys the balance between the precision and the recall.

$$F1Score = \frac{2 * precision * recall}{precision + recall}$$

Table 3 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. We used a script that detected detections with a threshold confidence value of 0.5. 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 -3: Training Results

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. CONCLUSIONS

The integration of various components required in the process of building a custom object detector including image annotation, dataset splitting, training the model and making detections will make the process much easier for anyone looking to do it using YOLO for their needs. Furthermore, a user does not need to have in-depth knowledge of how to set things up. Almost all the instructions are given in the application which should be sufficient for using all the other features as well. The applications of object detection are ubiquitous and having a tool that enables a user to build a custom object detector with minimum effort is critical in such cases. There are going to be lots of new frameworks, tool, algorithms that are going to come up as a result of research in the fields of computer vision and artificial intelligence and coupled with increasing availability of powerful computational hardware like GPUs, it's very important to have systems in place that can allow users to extract benefits of this. With that in mind, we have proposed this system that is an end-to-end solution for using state-of-

the-art algorithms to build a custom object detector for ship intrusion detection. The screenshots of the main UI pages of the system along with detection results on some images can be found on this link. Our application is rudimentary in the sense that emphasis is put on a system that uses only one algorithm and has a fixed workflow which is nonetheless effective but there is scope for expanding this. There can always be more flexibility in that different neural network architectures and methodologies can be added to the system. Further, the Custom Object Detection System can be provided as a service with all the steps like training, labeling, testing being done on the cloud and the user doesn't need to put a lot of resources. Leveraging existing cloud resources offered, the system can be turned into a web application or mobile application with constructing the detector for a specific object (class) in a short amount of time.

ACKNOWLEDGEMENT

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 like to show our appreciation to Mr Amroz Siddiqui 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.

REFERENCES

[1] A. Zainuddin, Z. Khalidin, M. S. Mohd Taufik, and A. F. Mohd Mansor, "Patient monitoring system using computer vision for emotional recognition and vital signs detection," 11 2020.M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[2] W. Lu, J. Ting, J. J. Little, and K. P. Murphy, "Learning to track and identify players from broadcast sports videos," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 7, pp. 1704– 1716, 2013.

[3] H. Tian, T. Wang, Y. Liu, X. Qiao, and Y. Li, "Computer vision technology in agricultural automation ——a review," Information Processing in Agriculture, vol. 7, 09 2019.

[4] D. Steinkraus, I. Buck, and P. Simard, "Using gpus for machine learning algorithms," 10 2005, pp. 1115 – 1120 Vol. 2.

[5] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," 2018.

[6] S. A. Velastin, "Cctv video analytics: Recent advances and limitations," in Visual Informatics: Bridging Research and Practice, H. Badioze Zaman, P. Robinson, M. Petrou, P. Olivier, H. Schr¨oder, and T. K. Shih, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 22–34.

[7] A. Costin, "Security of cctv and video surveillance systems: Threats, vulnerabilities, attacks, and mitigations," 10 2016.

[8] J. K. Park, K. S. Han, and S. Y. Yun, "Intensity classification background model based on the tracing scheme for deep learning based cctv pedestrian detection," in 2018 IEEE 9th International Conference on Mechanical and Intelligent Manufacturing Technologies (ICMIMT), 2018, pp. 224–228.

[9] Ting Shan, Shaokang Chen, C. Sanderson, and B. C. Lovell, "Towards robust face recognition for intelligent-cctv based surveillance using one gallery image," in 2007 IEEE Conference on Advanced Video and Signal Based Surveillance, 2007, pp. 470–475.

[10] J. Barnum, "Ship detection with high-resolution hf skywave radar," IEEE Journal of Oceanic Engineering, vol. 11, no. 2, pp. 196–209, 1986.

[11] C. Wang, S. Jiang, H. Zhang, F. Wu, and B. Zhang, "Ship detection for high-resolution sar images based on feature analysis," IEEE Geoscience and Remote Sensing Letters, vol. 11, no. 1, pp. 119–123, 2014.

[12] H. Luo, K. Wu, Z. Guo, L. Gu, and L. M. Ni, "Ship detection with wireless sensor networks," IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 7, pp. 1336–1343, 2012.

[13] T. Nie, B. He, G. Bi, Y. Zhang, and W. Wang, "A method of ship detection under complex background," ISPRS International Journal of Geo-Information, vol. 6, p. 159, 05 2017.

[14] S. Dasiopoulou, V. Mezaris, I. Kompatsiaris, V. Papastathis, and M. Strintzis, "Knowledge-assisted semantic video object detection," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 15, pp. 1210 – 1224, 11 2005.

[15] D. H. Ballard and C. M. Brown, Computer Vision, 1st ed. Prentice Hall Professional Technical Reference, 1982.

[16] T. Huang, "Computer vision: Evolution and promise," 1996.

[17] M. Sonka, V. Hlavac, and R. Boyle, Image processing, analysis and and machine vision (3. ed.)., 01 2008.

[18] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning. MIT press Cambridge, 2016, vol. 1, no. 2.

[19] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115–133, 1943.

[20] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a backpropagation network," in Advances in neural information processing systems, 1990, pp. 396–404.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural computation, vol. 18, no. 7, pp. 1527–1554, 2006.

[23] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," Computational intelligence and neuroscience, vol. 2018, 2018.

[24] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual

cortex," The Journal of physiology, vol. 160, no. 1, p. 106, 1962.

[25] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in Competition and cooperation in neural nets. Springer, 1982, pp. 267–285.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[27] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[28] M. Tygert, J. Bruna, S. Chintala, Y. LeCun, S. Piantino, and A. Szlam, "A mathematical motivation for complex-valued convolutional networks," Neural computation, vol. 28, no. 5, pp. 815–825, 2016.

[29] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," arXiv preprint arXiv:1804.02767, 2018.

[30] J. Lin and M. Sun, "A yolo-based traffic counting system," in 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI), 2018, pp. 82–85.

[31] Y. Tian, G. Yang, Z. Wang, H. Wang, E. Li, and Z. Liang, "Apple detection during different growth stages in orchards using the improved yolo-v3 model," Computers and electronics in agriculture, vol. 157, pp. 417–426, 2019.

[32] G. Bradski and A. Kaehler, Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[33] D. Tzutalin, "Labelimg–labelimg is a graphical image annotation tool. github repository," 2020.

[34] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick, "Microsoft coco: Common objects in context," in European conference on computer vision. Springer, 2014, pp. 740–755.