**AUTOMATIC HELMET VIOLATION DETECTION**

***Submitted by***

**LUHINDAAR.M 1907022**

**PRAMOTH KANNA.P 1907030**

**VEDHA.B 1907057**

***In partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

***in***

**INFORMATION TECHNOLOGY**

******

**COIMBATORE INSTITUTE OF TECHNOLOGY, COIMBATORE-641014**

***(Government Aided Autonomous Institution Affiliated to Anna University)***

**ANNA UNIVERSITY, CHENNAI 600025**

**APRIL 2022**

**COIMBATORE INSTITUTE OF TECHNOLOGY**

***(A Govt. Aided Autonomous Institution Affiliated to Anna University)***

## COIMBATORE – 641014

**BONAFIDE CERTIFICATE**

Certified that this project report titled “**AUTOMATIC HELMET VIOLATION DETECTION”** is the Bonafide work of **LUHINDAAR. M (1907022), PRAMOTH KANNA. P (1907030),** and **VEDHA. B (1907057)** in partial fulfillment for the award of the Degree of Bachelor of Technology in Information Technology of Anna University, Chennai during the academic year 2021-2022 under my supervision.

## Prof.N.K.KARTHIKEYAN, Prof.Dr.N.K.KARTHIKEYAN,

## HEAD OF THE DEPARTMENT, SUPERVISOR,

Department of Information Technology, Department of Information Technology,

Coimbatore Institute of Technology, Coimbatore Institute of Technology, Coimbatore - 641014. Coimbatore - 641014.

***Certified that the candidates were examined by us in the project work viva-vice examination held on ……………….***

## Internal Examiner External Examiner

Place:

Date:

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **LIST OF TABLES** | 5 |
|  | **LIST OF FIGURES** | 6 |
|  | **ACKNOWLEDGEMENT** | 7 |
|  | **ABSTRACT** | 8 |
| **1** | **INTRODUCTION** | 9 |
|  | 1.1 NEED FOR THE DETECTION OF HELMET | 9 |
|  | 1.2 DEEP LEARNING | 9 |
|  | 1.3 COMPUTER VISION USING OPENCV | 10 |
|  | 1.4 IMAGE CLASSIFICATION | 10 |
|  | 1.5 YOLO | 10 |
| **2** | **LITERATURE SURVEY** | 11 |
| **3** | **SYSTEM ARCHITECTURE** | 16 |
|  | 3.1 PROPOSED SYSTEM | 16 |
|  | 3.1.1 DATASET | 16 |
|  | 3.1.2 METHODOLOGY | 17 |
|  | 3.1.3 YOLOV3 | 18 |

|  |  |  |
| --- | --- | --- |
| **4** | **SYSTEM SPECIFICATION** | 19 |
|  | 4.1 HARDWARE REQUIREMENTS | 19 |
|  | 4.2 SOFTWARE REQUIREMENTS | 19 |
| **5** | **SYSTEM DESIGN** | 20 |
|  | 5.1 DATA ANNOTATION | 20 |
|  | 5.2 TRAINING MODULE | 20 |
| **6** | **IMPLEMENTATION** | 21 |
|  | 6.1 YOLOV3 ARCHITECTURE FOR THE  MODEL | 21 |
|  | 6.2 YOLOV3 TRAINING | 21 |
| **7** | **RESULT** | 25 |
|  | 7.1 SNAPSHOTS OUTPUT | 25 |
| **8** | **CONCLUSION AND FUTURE WORKS** | 28 |
|  | 8.1 CONCLUSION | 28 |
|  | 8.2 FUTURE WORKS | 28 |
| **9** | **APPENDIX** | 29 |
|  | **9.1 SOURCE CODE** | 29 |
|  | 9.1.1 FRAME EXTRACTION | 29 |
|  | 9.1.2 MODEL CREATION | 30 |
| **10** | **REFERENCES** | 38 |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table Number** | **Name of the table** | **Page Number** |
| **1** | Dataset | 16 |
| **2** | Hardware requirements | 19 |
| **3** | Software Requirements | 19 |

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **Figure Number** | **Name of the Figure** | **Page Number** |
| **1** | Sample Image of Dataset | 16 |
| **2** | System Architecture | 17 |
| **3** | CNN Architecture | 18 |
| **4** | YOLOv3 Architecture | 21 |
| **5** | Dataset | 25 |
| **6** | Frame Extraction and Data Collection for Real Time | 25 |
| **7** | Deep Learning Model with YOLOv3 | 26 |
| **8** | Implementation of the Trained Model | 27 |
| **9** | ROC Graph | 27 |
| **10** | Loss Graph | 27 |

**ACKNOWLEDGEMENT**

Our project “AUTOMATIC HELMET VIOLATION DETECTION” has been the result of motivation and encouragement from many, whom we would like to thank.

We express our sincere thanks to our Secretary **DR.R.Prabhakar** and our Principal **DR.A.Rajesweri** and our advisor **Dr.V.Selladurai** for providing us a greater opportunity to carry out our work. The following words are rather very me agree to express our gratitude to them. This work is the outcome of their inspiration and product of plethora of their knowledge and rich experience.

We record the deep sense of gratefulness to **Dr.N.K.Karthikeyan**, Head of the Department of Information Technology, for his encouragement and support during this tenure.

We equally tender my sincere gratitude to our project guide **Dr.N.K.Karthikeyan,** Department of Information Technology, for his valuable suggestions and guidance during this course.

During the entire period of study, the entire staff members of the Department of Computer Science and Engineering & Information Technology have offered ungrudging help. It is also a great pleasure to acknowledge the unfailing help we have received from our friends.

It is a matter of great pleasure to thank our parents and family members for their constant support and cooperation in the pursuit of this Endeavour.

**ABSTRACT**

In India, the number of accidents occurring each day is increasing rapidly. The two-wheelers account 25 percent of road crash deaths because of ignoring safety measures like wearing helmets while driving. A police officer cannot manage the whole traffic and look out for rule-breakers. It would be a very tough job and will need a lot of human power to cover all the areas. Our project’s main objective is to identify the two-wheeler riders who failed to wear a helmet. The system implements Deep learning and video processing techniques. The system takes a video of traffic on public roads as input. The road CCTV footage is used to detect whether a rider is wearing a helmet or not, using YOLOv3 Algorithm, a Deep Learning Technology and the respective frames are taken from the video to detect riders, riding two wheelers, who are not wearing helmets. Using this helmet detection model riders without helmet can be easily detected.

**CHAPTER-1**

**INTRODUCTION**

* 1. **Need for the Detection of Helmet**

Helmet detection means classifying the two-wheeler riders into Helmet wearing and non-helmet wearing**.** Helmet wearing group includes the riders riding various two-wheeler with their helmet wore. Non-helmet wearing group includes the riders riding various two-wheeler without wearing their helmet wore. Helmet detection is a process of classifying the riders into classes like helmet and non-helmet. Each detection goes into its respective class at the time of classification.

* 1. **Deep Learning**

Deep learning is a subset of [machine learning](https://www.ibm.com/cloud/learn/machine-learning), which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to “learn” from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. Deep learning drives many [artificial intelligence (AI)](https://www.ibm.com/cloud/learn/what-is-artificial-intelligence) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data. Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called *visible*layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made. Another process called backpropagationuses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and backpropagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.

* 1. **Computer Vision using OpenCV**

Computer Vision is an interdisciplinary field that deals with how computers can be made to gain a high-level understanding from digital images or videos. The idea here is to automate tasks that the human visual systems can do. So, a computer should be able to recognize objects such as that of a face of a human being or a lamppost or even a statue. OpenCV is Python’s very own Computer Vision Library. OpenCV is a Python library that was designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel, but later, it was supported by Willow Garage. It supports a wide variety of programming languages such as C++, Python, Java, etc. Support for multiple platforms including Windows, Linux, and MacOS. OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays. This makes it easier to integrate it with other libraries that use NumPy. For example, libraries such as SciPy and Matplotlib.

* 1. **Image Classification**

Image classification refers to the task of extracting information classes from a [multiband](https://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/raster-bands.htm#GUID-203F9D69-BEAB-4FF3-8153-336B5029F33E) raster image. The resulting raster from image classification can be used to create thematic maps. Depending on the interaction between the analyst and the computer during classification, there are two types of classification: supervised and unsupervised. This process makes use of features like edges and blends from the images to distinguish and classify images into different classes.

* 1. **You Only Look Once(YOLO)**

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. 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, 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.

**CHAPTER -2**

**LITERATURE SURVEY**

**2.1 DETECTION OF TWO-WHEELERS HELMET USING MACHINE LEARNING**

In this paper, the YOLOv5 based Helmet identification and also studied CNN. The Jupyter notebook is used to execute the python program and hence successfully compiled the program. The implemented architecture of helmet detection has been tested and implemented multiple times and verified the accuracy every time. The project can be combined with the road traffic cameras and with some modifications of software and if possible, developing an app that can be implemented to recognize helmets automatically in the real-time system. Furthermore, automatic license plate identification and make software that generates challans for not wearing helmets can be implemented.

**2.2 DETECTING MOTORCYCLE HELMET USE WITH DEEP LEARNING**

In this paper, the Automated helmet use detection for motorcycle riders is a promising approach to efficiently collect large, up-to-date data on this crucial measure. When trained, the algorithm presented in this paper can be directly implemented in existing road traffic surveillance infrastructure to produce real-time helmet use data. The evaluation of the algorithm confirms a high accuracy of helmet use data, that only deviates by a small margin from comparable data collected by human observers. Observation site-specific training of the algorithm does not involve extensive data annotation. While the sole collection of data does not increase road safety by itself, it is a prerequisite for targeted enforcement and education campaigns, which can lower the rate of injuries and fatalities. The automated helmet use detection can solve the challenges of costly and time-consuming data collection by human observers.

**2.3 HELMET DETECTION AND LICENSE PLATE RECOGNITION**

In this project, A Non-Helmet Rider Detection system is developed where a video file is taken as input. If the motorcycle rider in the video footage is not wearing a helmet while riding the motorcycle, then here we are uploading an image to identify the license plate number of that motorcycle is extracted from the image and displayed. Object detection principle with YOLO architecture is used for motorcycle, person, helmet, and license plate detection. OCR is used for license plate number extraction if the rider is not wearing a helmet. Not only the characters are extracted, but also the frame from which it is also extracted so that it can be used for other purposes. All the objectives of the project are achieved satisfactorily.

**2.4 AN IMPROVED HELMET DETECTION METHOD FOR YOLOV5 ON AN UNBALANCED DATASET**

In this paper, the direct use of YOLOv5 can no longer adapt to the real production situation for target detection requirements. For the unbalanced helmet dataset, the data augmentation processes the data and then train the model with the YOLOv5 algorithm, without affecting the recognition speed, and finally get a 0.01-0.02 increase in the confidence level. The image detection and classification problems similar to helmet detection can draw on the results in this thesis to complete the further optimization of the recognition effect, which is of great significance for the rapid image detection and recognition of actual production sites.

**2.5 SAFETY HELMET WEARING DETECTION BASED ON IMAGE PROCESSING AND MACHINE LEARNING**

This paper proposed an innovative and practical safety helmet-wearing detection method based on image processing and machine learning. At first, the ViBe background modelling algorithm is exploited to detect moving objects under the view of fix surveillant camera in a power substation. After obtaining the motion region of interest, the Histogram of Oriented Gradient (HOG) feature is extracted to describe the Inner human. And then, based on the result of HOG feature extraction, the Support Vector Machine (SVM) is trained to classify pedestrians. Finally, the safety helmet detection will be implemented by colour feature recognition. Compelling experimental results demonstrated the correctness and effectiveness of this proposed method.

**2.6 SAFETY HELMET DETECTION BASED ON YOLOV5**

This work introduces the YOLOv5-based safety helmet detection method in detail, including the YOLOv5 network structure, classifier settings, and data set processing. Besides, experiments were carried out using YOLOv5 models with different parameters. As a result, a YOLOv5-based helmet detection method has achieved excellent results in test accuracy and speed. After using the pre-training weights, the map values of various models increased by about 1%. In the next work, we will collect a large number of positive and negative samples of real scenes to improve the network’s generalization ability.

**2.7 AUTOMATIC DETECTION OF BIKE-RIDERS WITHOUT HELMET USING SURVEILLANCE VIDEOS IN REAL-TIME**

In this paper, a framework for real-time detection of traffic rule violators who ride bikes without using helmets is proposed. The proposed framework will also assist the traffic police in detecting such violators in odd environmental conditions viz; hot sun, etc. Experimental results demonstrate the accuracy of 98.88% and 93.80% for the detection of bike-rider and detection of violators, respectively. The average time taken to process a frame is 11.58 ms, which is suitable for real-time use. Also, the proposed framework automatically adapts to new scenarios if required, with slight tuning. This framework can be extended to detect and report number plates of violators.

**2.8 DETECTION OF MOTORCYCLISTS WITHOUT HELMET IN VIDEOS USING CONVOLUTIONAL NEURAL NETWORK**

The proposed framework for automatic detection of motorcyclists driving without helmets makes use of adaptive background subtraction which is invariant to various challenges The use of deep learning for automatic learning of discriminative representations for classification tasks improves the detection rate and reduces the false alarms resulting in a more reliable system. The experiments on real videos successfully detect ≈ 92.87% of violators with a low false alarm rate of ≈ 0.50% on two real video datasets and thus shows the efficiency of the proposed approach.

**2.9 HELMET WEARING DETECTION FOR RIDERS’ SAFETY**

In this paper both SIFT and SURF algorithms are used for detecting the helmet. By analysing both of the algorithms in this paper, we can conclude that SURF works as faster and executes for identifying the helmet in real-time video capturing. SURF algorithms work more efficiently than the SIFT algorithm to detect the helmet. This paper will be done by comparing other algorithms like KNN, SVM, SIFT, AND SURF for detecting the helmet. Further, helmet detection can be implemented in traffic intersections, which helps to monitor people not wearing helmets and in return reduces accidents and assures riders’ safety.

**2.10 REAL-TIME HELMET DETECTION OF BIKE RIDERS**

The main objective of this paper was to develop a system for real-time detection of motorcyclists without helmets using the object detection-algorithms YOLO (You Only Look Once). The system can successfully differentiate between a person wearing a helmet and a person not wearing a helmet.

**2.11 HELMET DETECTION ON MOTORCYCLISTS USING IMAGE DESCRIPTORS AND CLASSIFIERS**

This study addresses the detection of motorcyclists without helmets on public roads. A computer vision system was proposed and classified as follows: – vehicle segmentation and classification, and – detection of helmet use. In the stage of vehicle segmentation and classification, algorithms for the background calculation and tracking of objects, descriptors, and classifiers that exhibited reasonable hit rates and low processing times were selected from the literature achieving an accuracy of 0.9778. In the stage of detection of helmet use, algorithms for the extraction of features in images and classification algorithms were employed. The proposed system also obtained satisfactory hit rates.

**2.12 DETECTION OF HELMETLESS RIDERS USING FASTER R-CNN**

This project aims to decrease the accidents caused due to not wearing helmets. It also ensures whether the law is violated or not. It stores the image of the violated people along with a cropped number plate. It is then sent through the mail to authorities. The algorithm used here is the Faster RCNN which is faster and more accurate when compared to the other one. The accuracy of the result of this project is 92%. It depends on the quality of the CCTV camera.

**2.13 HELMET DETECTION USING MACHINE LEARNING AND AUTOMATIC LICENSE PLATE RECOGNITION**

In this project, the results shown above it is evident that the YOLO object detection is well suited for real-time processing and was able to accurately classify and localize all the object classes. The proposed end-to-end model was developed successfully and has all the capabilities to be automated and deployed for monitoring. For extracting the number plates some techniques are employed by considering different cases such as multiple riders without helmets and designed to handle most of the cases. All the libraries and software used in the project are open source and hence are very flexible and cost-efficient. The project was mainly built to solve the problem of non-efficient traffic management. Hence at the end of it, it can be deployed by any traffic management department, it would make their job easier and more efficient.

**2.14 HELMET DETECTION USING ML & IOT**

In this paper, the software for the helmet detection has been thoroughly tested and implemented in a high-level language, and I have realized the ingenuity and patience with this job has to be done. In the project, the YOLO-based Helmet detection is provided, and also made a detailed study about CNN. Jupyter notebook was used to implement the program and it was successfully implemented. Our project was successfully tested in python. We also made a study application and the future scope of the project. This project can be linked with the traffic cameras and with some modifications it can be used to detect helmets in the real-time system. Further, it can be merged with the algorithm of automated license plate detectionand make a system that generates challans for those who don’t wear helmets.

**2.15 HELMET DETECTION ON MOTORCYCLISTS USING IMAGE DESCRIPTORS AND CLASSIFIERS**

This paper comes about the detection of motorcyclists without helmets. It proposes a computer vision system that is divided as follows: moving objects, segmentation, moving objects classification, and helmet use detection. It is necessary to have an image with better quality to recognize the characters. The used descriptors return lots of features and this can make the classification difficult. Considering this, attribute selection is another future work. It is very important to increase the success rates. For this reason, analysis for this kind of algorithm is required. The use of Systems of Multiple Classifiers (SMCs) will be made. SMCs with dynamic structures can be used, for example, the mixture of experts.

**CHAPTER – 3**

**SYSTEM ARCHITECHTURE**

**3.1 PROPOSED SYSTEM**

**3.1.1 Dataset**

The Dataset comes from a popular Kaggle contributor which is utilized for research taken by researchers and assistants using mobile phones. All the images are captured in different backgrounds, different lighting conditions to keep the diversity in the dataset alive. In addition to this dataset, we have included images taken from real time traffic videos. Labels of the objects are basically divided into 2 classes – with\_helmet and without\_helmet.

Following is a summary of information about the data from Kaggle Dataset.

|  |  |
| --- | --- |
| Class | Number of Images |
| Without Helmet | 250 |
| With Helmet | 200 |

Total : 450 images.

In addition to these images, around 500 images are added from our side. Post Data Augmentation around 1000 images were generated.

Following is a sample of images from Kaggle Dataset.

****

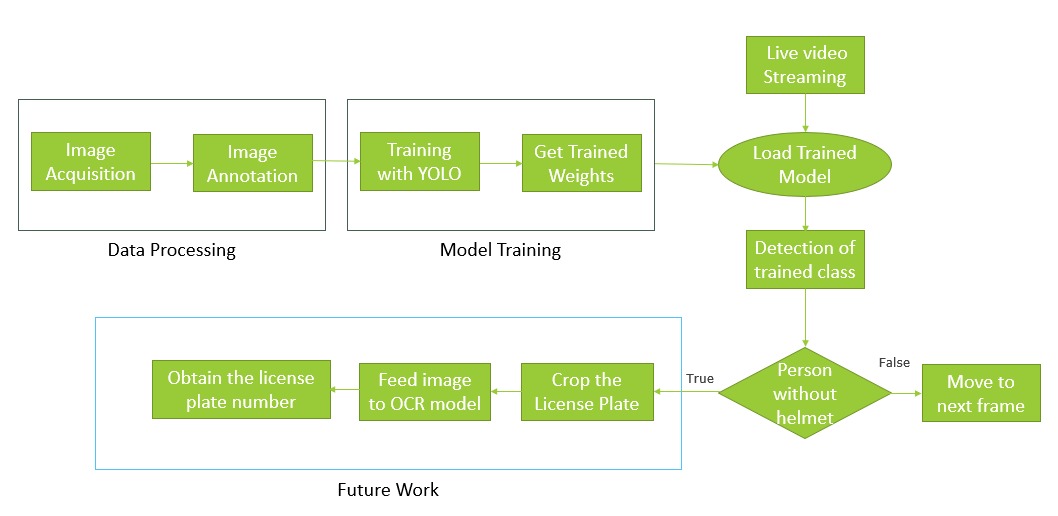
***Fig 3.1*** *Sample Image of the Dataset*

**Train dataset.** The Training Dataset consists of a data from both the Kaggle Dataset and self-collected dataset. Train Dataset has around 1000 images without applying Data Augmentation. Each class on average has around 500 images.

**Test dataset.** The Test dataset consists of randomly selected image for each class. There are about 200 images for Test Dataset without Data Augmentation.

**3.1.2 Methodology**

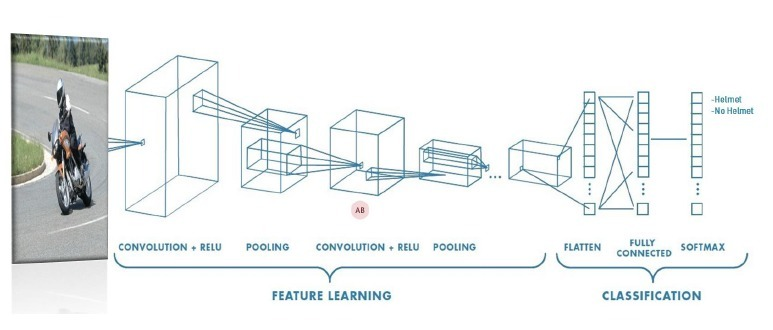
The system uses OpenCV to stream live video and capture images at regular intervals. The image captured is then passed onto the image classification model. The model is either deployed in cloud platforms or saved in local computer. The model first classifies the object into 2 different classes namely with\_helmet and without\_helmet. System for Helmet detection consists of a method which can train the network and allows the user to predict helmet violators in the video. The user data will be pre-processed on the fly before it gets fed into the system.

****

***Fig 3.2*** *System Architecture*

**3.1.3 YOLOv3**

Inspired by ResNet and FPN (Feature-Pyramid Network) architectures, YOLO-V3 feature extractor, called Darknet-53 (it has 52 convolutions) contains skip connections (like ResNet) and 3 prediction heads (like FPN) — each processing the image at a different spatial compression. Like its predecessor, Yolo-V3 boasts good performance over a wide range of input resolutions. In GluonCV’s model zoo you can find several checkpoints: each for a different input resolution, but in fact the network parameters stored in those checkpoints are identical. Tested with input resolution 608x608 on COCO-2017 validation set, Yolo-V3 scored 37 mAP (mean Average Precision). This score is identical to GluonCV’s trained version of Faster-RCNN-ResNet50, (a faster-RCNN architecture that uses ResNet-50 as its backbone) but 17 times faster. In that model zoo the only detectors fast enough to compete with Yolo-V3.



***Fig 3.3*** *CNN Architecture*

**CHAPTER-4**

**SYSTEM SPECIFICATION**

This chapter includes the System Specification of our project. The hardware and software for the system is selected by considering the factors such as CPU processing speed, Libraries availability.

**4.1 Hardware Requirements**

|  |  |
| --- | --- |
| GPU | Any Nvidia/ AMD High End GPU for faster prediction |
| Camera | Web Camera |
| PC | With Python installed |

**4.2 Software Requirements**

|  |  |
| --- | --- |
| Operating System | Windows Operating System |
| IDE | Jupyter Notebook / Google Colaboratory and Visual Studio Code |
| Language | Python |

**CHAPTER-5**

**SYSTEM DESIGN**

**5.1 Data Annotation**

Before training the model, the training datasets need to be manually annotated with the BB information for two classes (Rider with helmet, Rider without helmet). The data annotation was done using LabelImg software [18] since YOLO needs a ground truth. The software generates starting coordinates (x0, y0) and ending coordinates (x1, y1). The coordinates were normalized between 0 and 1 to fit into YOLO format (x, y, w, h) using Eqs. (5)–(8):

x = (x0 + x1) 2 ∗ W (5)

y = (y0 + y1) 2 ∗ H (6)

w = (x1 − x0) W (7)

h = (y1 − y0) H (8)

**5.2 Training Module**

The YOLOv3 includes 19 convolutional layers and 5 max-pooling layers, followed by softmax activation functions to classify the object. The input image is divided into SxS grid cells. The grid cell, which contains the center of the object is responsible for predicting the 5-bounding box (BB) coordinates (bx , by , bw, bh, c). The coordinates (bx , by ) represent the center of the object relative to the grid cell location and (bw, bh) represents the width and height of the object relative to image dimensions. The presence of object in a grid cell is given by confidence score c. The coordinates (cx, cy) represent the starting grid cell location which contains the center of a rider. The actual coordinates of the BB were normalized with relative to grid cell location using Eqs. (1)–(4):

bx = (bx − cx) cx (1)

by = ( by − cy ) cy (2)

bw = bw W (3)

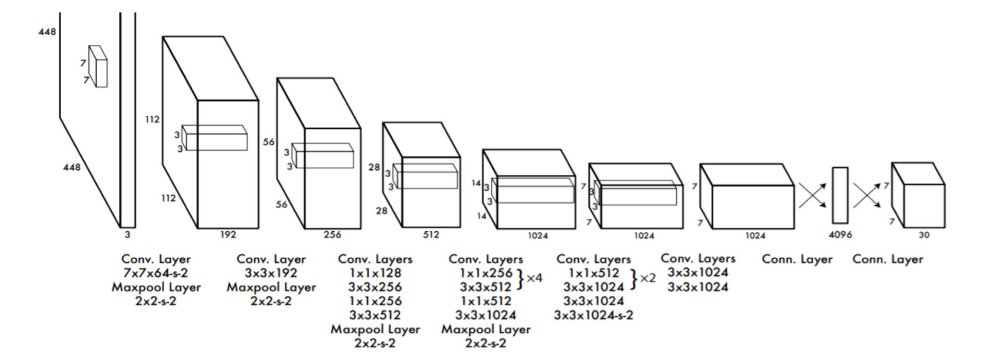
bh = bh H (4)

where W and H are the image dimensions.

**CHAPTER – 6**

**IMPLEMENTATION**

**6.1** **YOLOv3 Architecture for the Model**



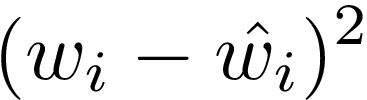
***Fig 6.1*** *YOLOv3 Architecture*

The YOLO model is made up of three key components: the head, neck, and backbone. The backbone is the part of the network made up of convolutional layers to detect key features of an image and process them. The backbone is first trained on a classification dataset, such as ImageNet, and typically trained at a lower resolution than the final detection model, as detection requires finer details than classification. The neck uses the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates. The head is the final output layer of the network which can be interchanged with other layers with the same input shape for transfer learning. The head is an **S × S × (C + B ∗ 5)** tensor and is **7 × 7 × 30** in the original YOLO research paper with a split size **S** of 7, 20 classes **C**, and 2 predicted bounding boxes **B**. These three portions of the model work together to first extract key visual features from the image then classify and bound them.

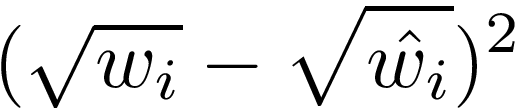
**6.2 YOLOv3** **Training**

The ImageNet 1000-class competition dataset and pre-trained 20 out of the 24 convolution layers followed by an average-pooling and fully connected layer is used. Then 4 more convolutions to the model as well as 2 fully connected layers as it has been shown that adding both convulsions and fully connected layers increases performance were added. The resolution from **244**×**244** to 448 × 448 pixels as detection requires finer details were also increased. The final layer, which predicts both class probabilities and bounding box coordinates, uses a linear activation function while the other layers use a leaky ReLU function.Data augmentation and dropout were used to prevent overfitting, with a dropout layer with a rate of 0.5, used between the first and second fully connected (preventing co-adaptation).

The loss function is the simple squared sum, but it must be modified. Without modification, the model will weight localization error, the difference between predicted and true bounding box coordinates, and class prediction error the same. Additionally, when a grid cell doesn’t contain an object, its confidence score tends towards 0 which can overpower the gradients from other cells that do contain objects. Both issues are solved by using two coefficients, λcoord and λnoobj, which multiply the loss for the coordinates and the object losses respectively. These are set to λcoord = 5 and λnoobj = 0.5, increasing the weight of detection and decreasing the importance of no object loss. Finally, to weight small bounding box equality as much as large boxes, the width and height difference is square-rooted rather than used directly. This makes sure that the error is treated the same as in large and small boxes, which would otherwise discourage the model from predicting large boxes. For example, if the predicted width of the bounding box is 10 and the actual width is 8, and this equation is used



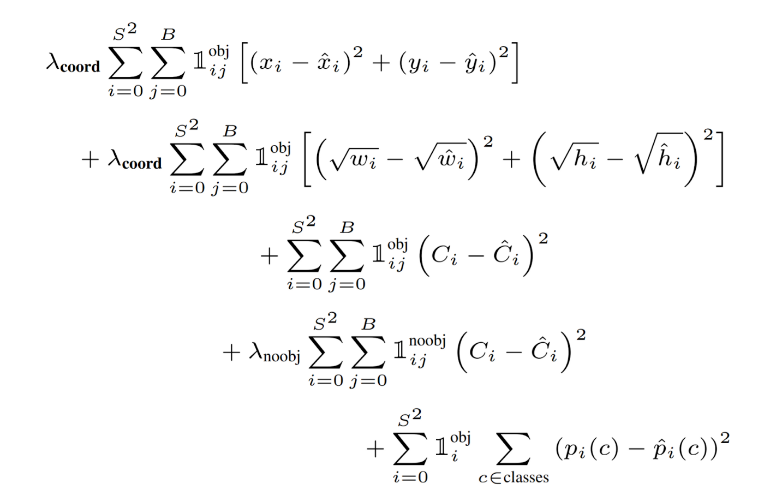
So the loss is 4. However, a difference of 2 out of the true 98 is negligible compared to a difference of 2 out of 8. Therefore, the loss between 10 and 8 should be much larger than the loss between 100 and 98. So this equation was used instead



Using this new equation, the loss for 10 and 8 is 0.111 while the loss for 100 and 98 is 0.010. So the fact that 0.111 is much smaller than 4 doesn’t matter, but what does matter is that the difference between loss for the large and small widths is 0% for the squared difference while the difference is 90.99% for the squared rooted difference.

Each grid cell predicts multiple bounding boxes, but only one bounding box is responsible for detecting the object. The responsible bounding box is determined by choosing the predicted bounding box with the highest IOU.

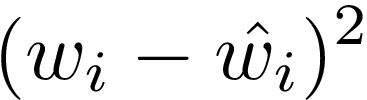
Loss function:



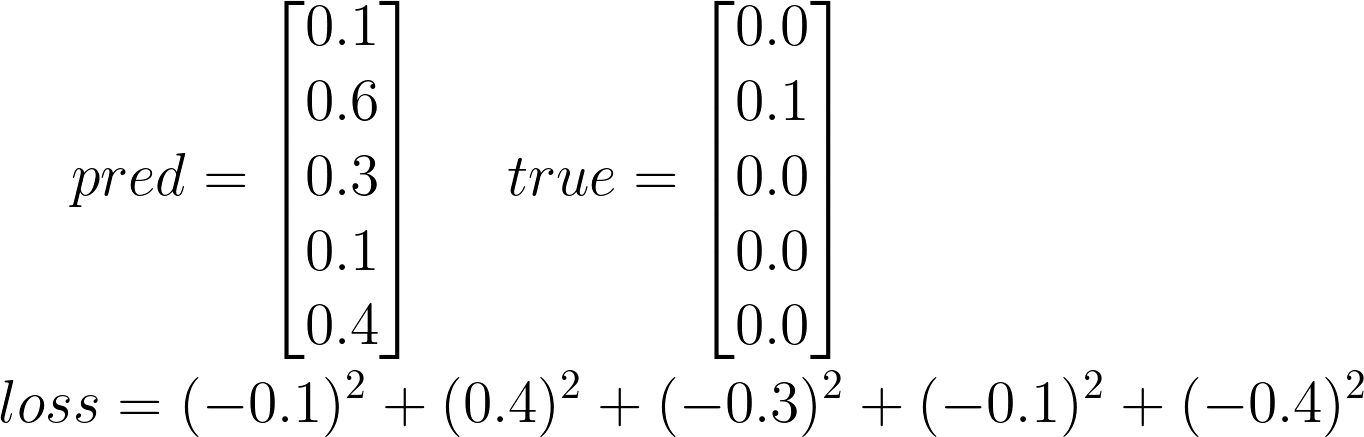
The double summation merely means to sum across all of the grid cells (an S × S square) and all of the bounding boxes B.



This is an identity function, set to 1 if there is an object in cell i, and bounding box j is responsible for the prediction.



This represents the squared difference between the actual x coordinate and the predicted coordinate in cell i. This is repeated for both x and y coordinates, finding the squared difference between the overall midpoint. Finally, the identity function is `0` when there is no object or the current bounding box isn’t the responsible one. In other words, we only calculate the loss for the best bounding box. So the first line of the equation is the sum of squared differences between the predicted and actual midpoints of the objects in all grid cells which have an object in them and are the responsible bounding box. The second line is the sum of the squared differences between the square roots of the predicted and actual widths and heights in all grid cells which have an object in them. These are square rooted for reasons explained earlier. The third line is just the squared difference between the predicted class probabilities and the actual class probabilities in all cells that contain an object. The fourth line is the same but for all cells that don’t have an object. These two lines are summed across all bounding boxes because each bounding box also predicts a confidence score in addition to coordinates. The last line: the first summation goes through every grid cell which has an object in it. Then, for that single grid cell, the squared difference between the predicted class vector and the actual vector is found.

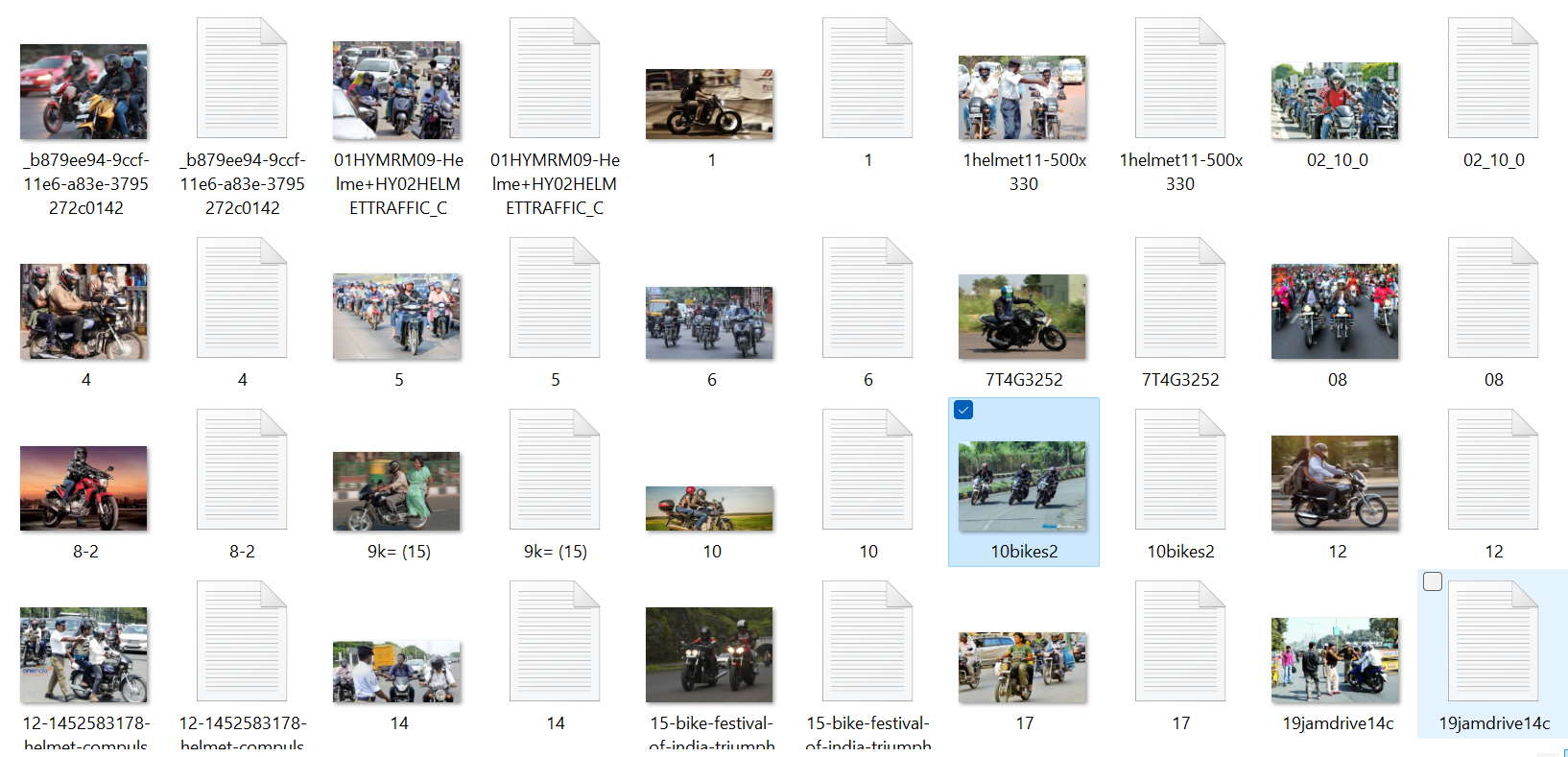


Finally, the last line is the squared difference between the predicted and actual class for all cells that have an object. This is calculated just across grid cells and not across each bounding box as each cell predicts only a single classification regardless of the number of bounding boxes it also predicts. Finally, all of these are summed together, and the first two lines are multiplied by a coordinate coefficient to weight them more heavily and line four is multiplied by a smaller no object coefficient to weight it less.

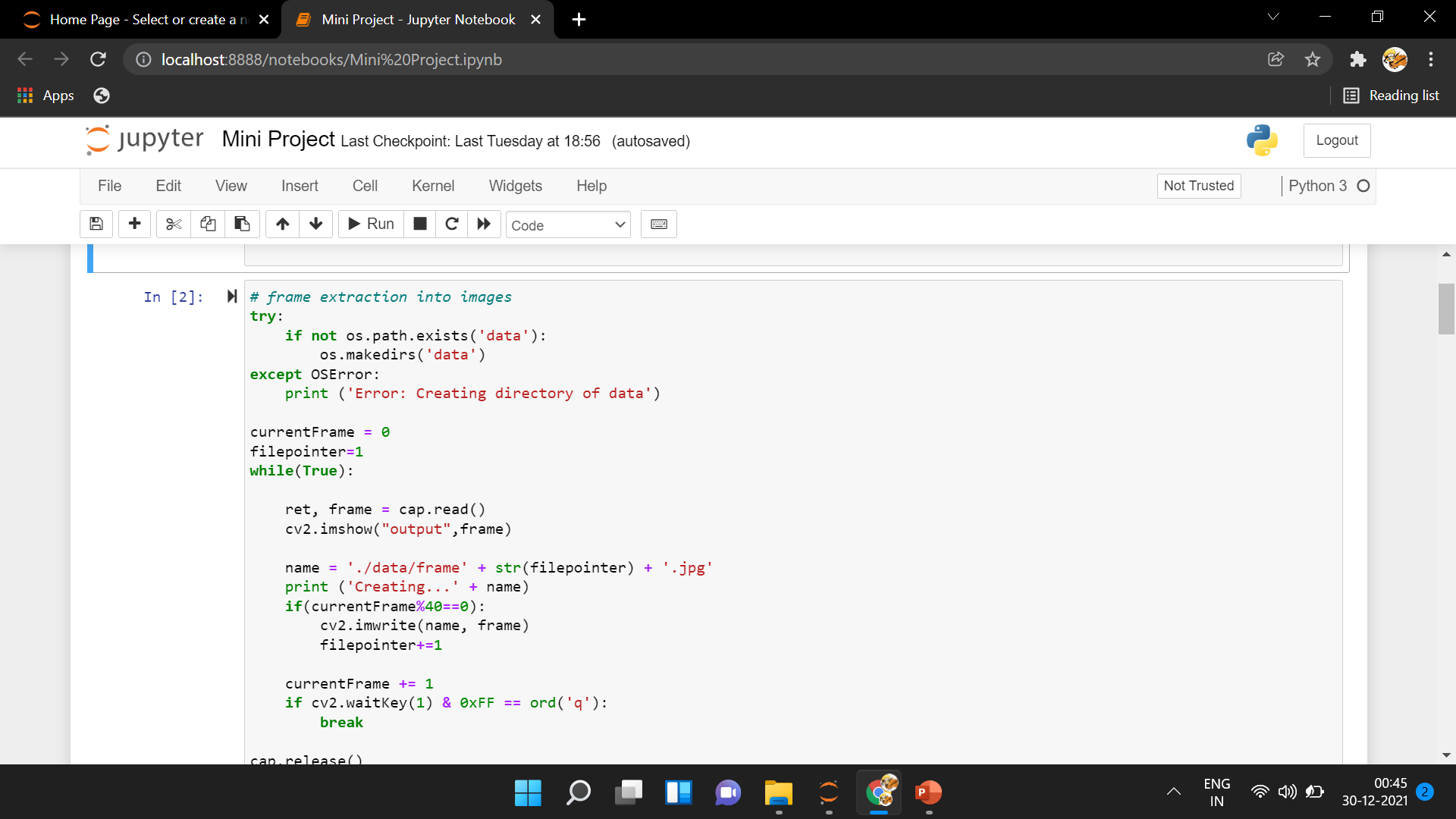
**CHAPTER – 7**

**RESULT**

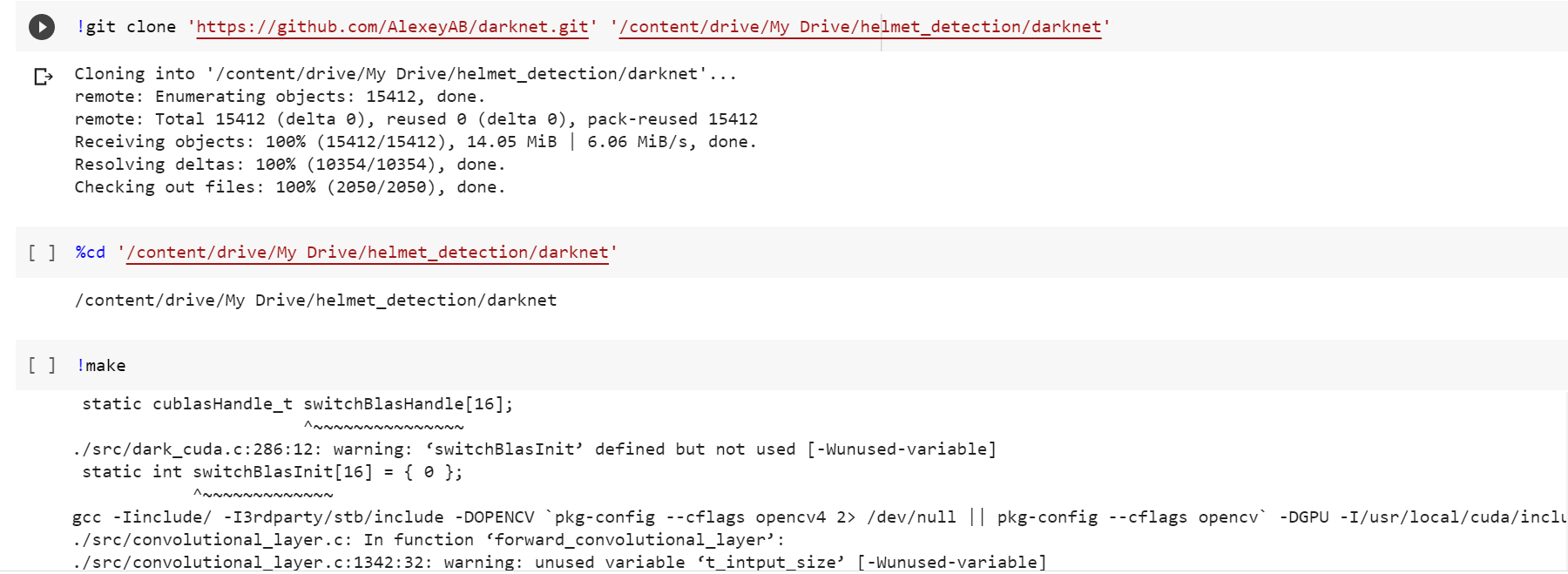
**7.1 SNAP SHOTS OUTPUT**

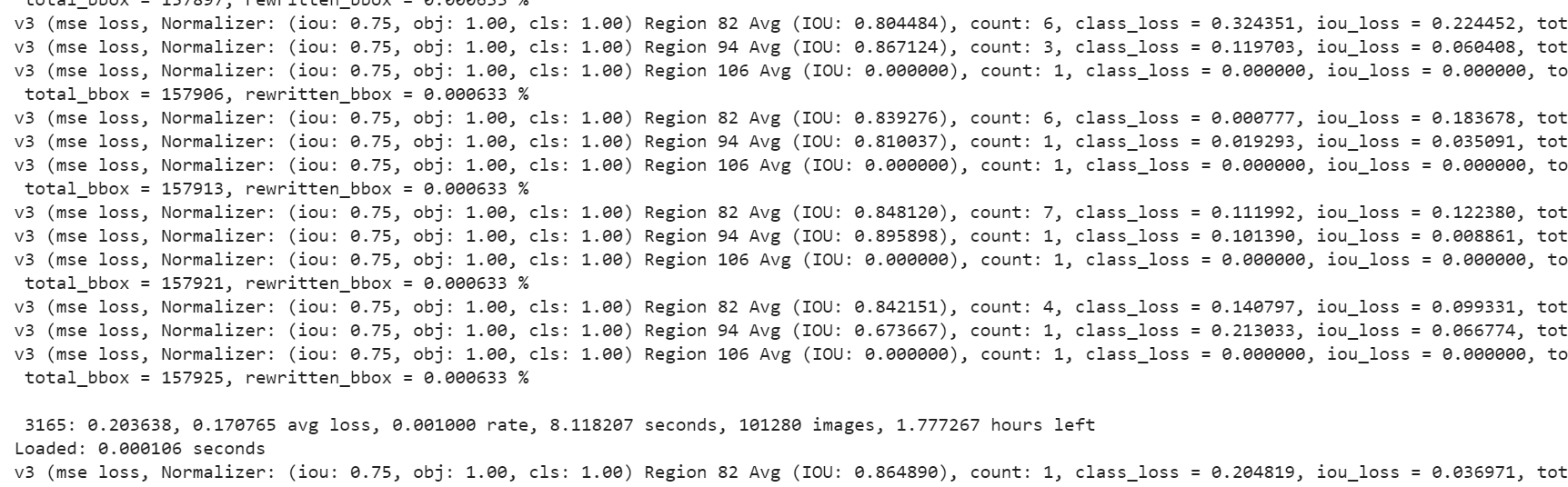
****

***Fig 7.1*** *Dataset*

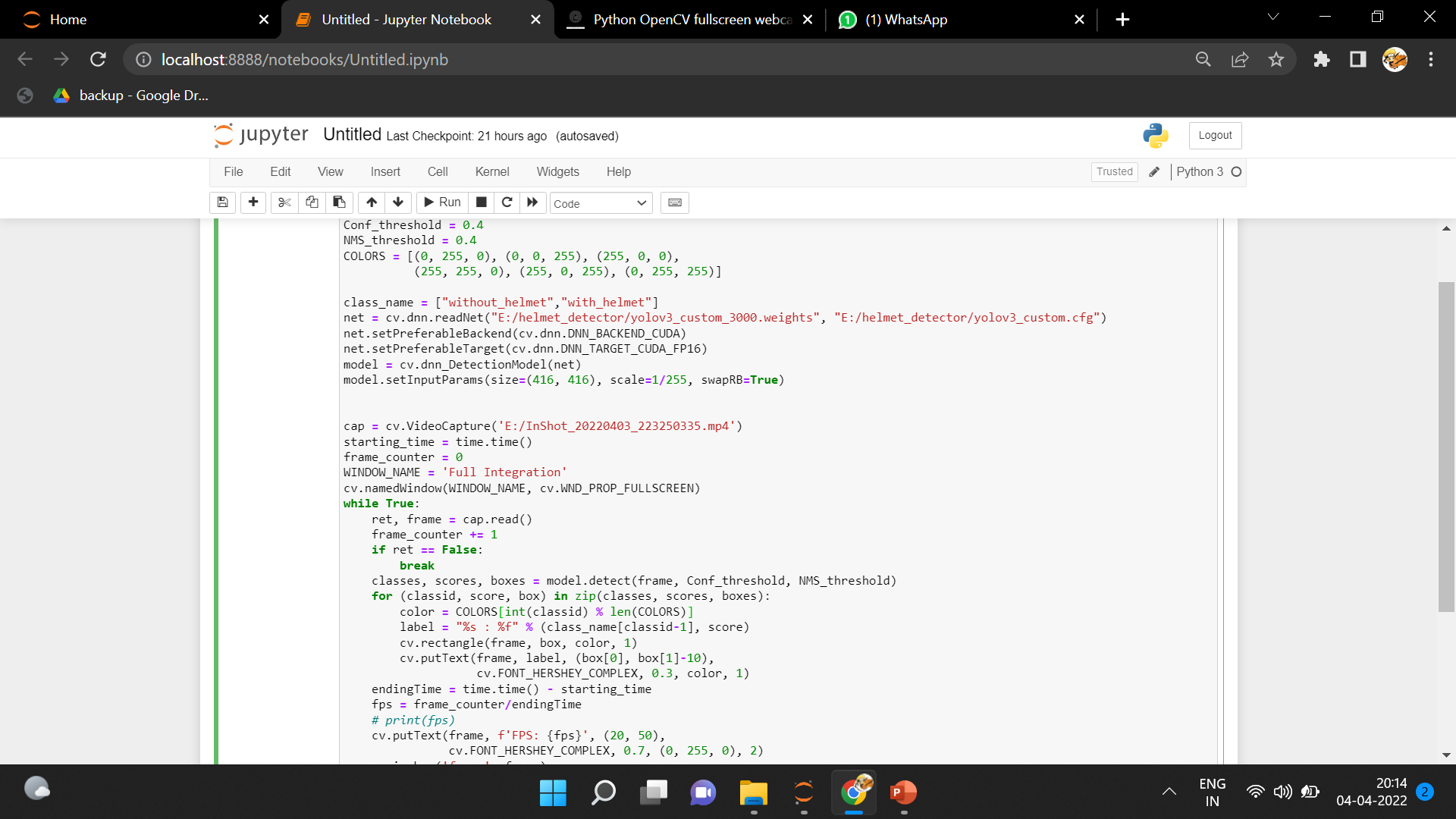
**

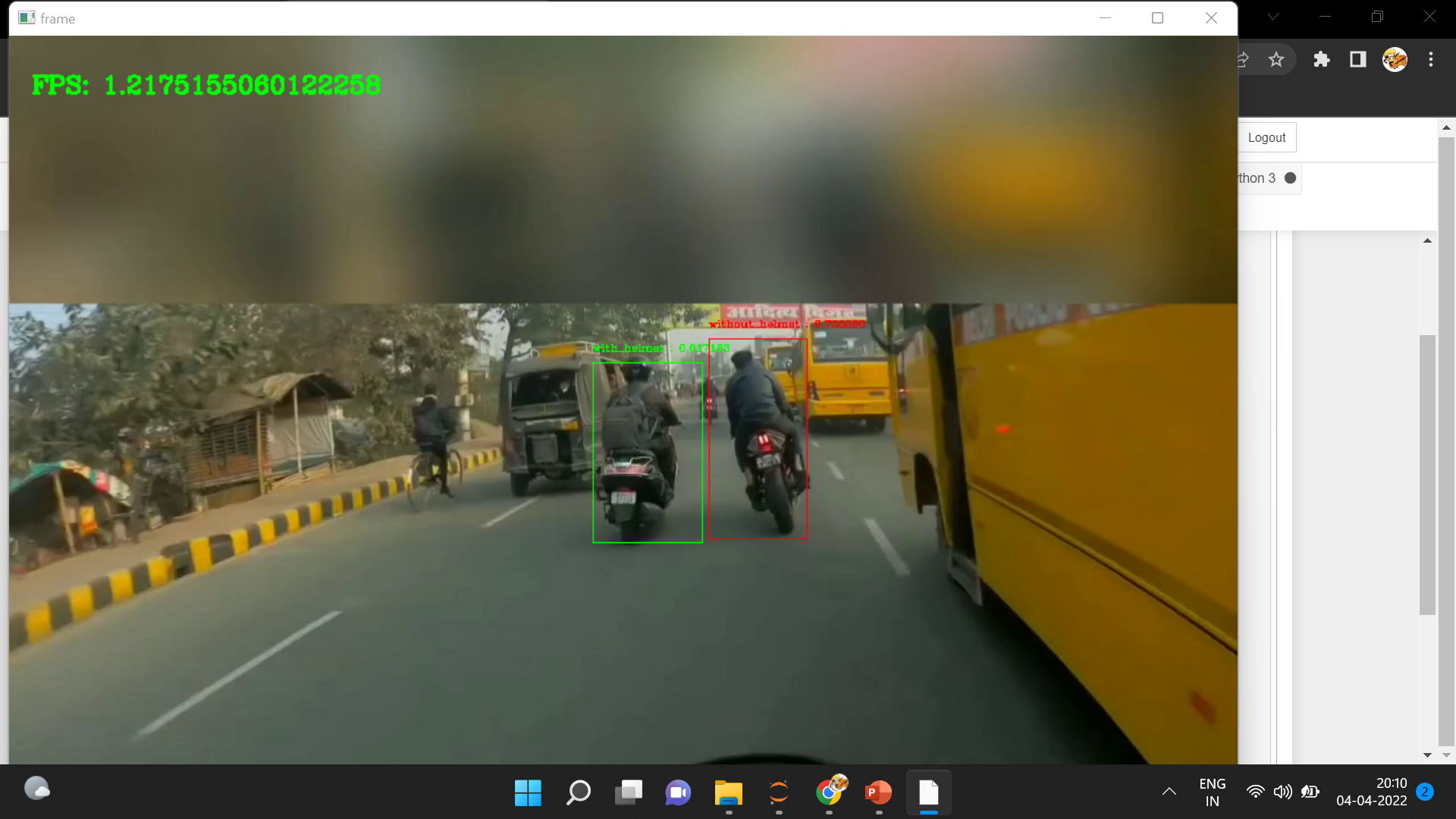
***Fig 7.2*** *Frame Extraction and Data Collection for Real Time*

**

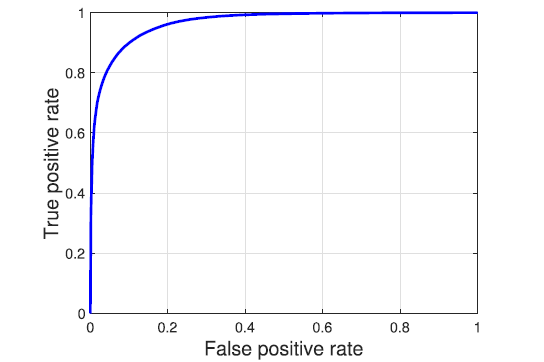
**

***Fig 7.3*** *Deep Learning Model with YOLOv3*

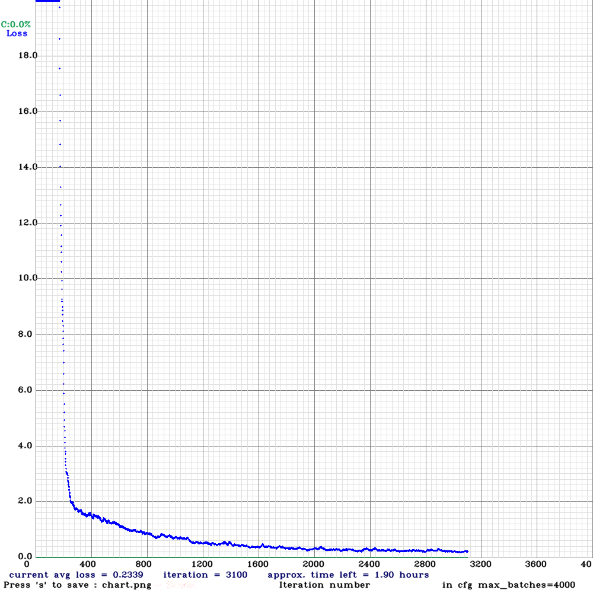




***Fig 7.4*** *Implementation of the Trained Model*



***Fig 7.5*** *ROC Graph*

**

***Fig 7.6*** Loss *Graph*

**CHAPTER-8**

**CONCLUSION AND FUTURE WORK**

**8.1 CONCLUSION**

The obtained value of accuracy is 82% with a better detection of the riders without helmet. From the results shown above it is evident that the YOLO object detection is well suited for real-time processing and was able to accurately classify and localize all the object classes. The proposed end-to-end model was developed successfully and has all the capabilities to be automated and deployed for monitoring. All the libraries and software used in our project are open source and hence is very flexible and cost efficient. The project was mainly built to solve the problem of non-efficient traffic management. Hence at the end of it we can say that if deployed by any traffic management departments, it would make their job easier and more efficient.

**8.2 FUTURE WORKS**

The model can further be improved with better training for the image classifier. Due to lack of resources the accuracy obtained is limited. This model can also be used in such a way that a motorbike does not start unless the rider wears a helmet. By incorporating a portable model to the ignition switch the power supply can be killed until a helmet is detected. This is one of the future developments that can be made to this model. Even though the vehicle cost will increase a bit than its standards, safety can be ensured. The system can further be automated by using an image to text classifier to identify number plates and generate a fine automatically.

**CHAPTER-9**

**APPENDIX**

**9. SOURCE CODE**

**9.1.1 Frame Extraction:**

import cv2

import numpy as np

import os

import matplotlib.pyplot as plt

%matplotlib inline

cap = cv2.VideoCapture('E:/video1.mp4')

# frame extraction into images

try:

if not os.path.exists('datass'):

os.makedirs('datass')

except OSError:

print ('Error: Creating directory of data')

currentFrame = 0

filepointer=1

while(True):

ret, frame = cap.read()

cv2.imshow("output",frame)

if(currentFrame%10==0):

name = './datass/frapp' + str(filepointer) + '.jpg'

print ('Creating...' + name)

cv2.imwrite(name, frame)

filepointer+=1

currentFrame += 1

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

**9.1.2 Model Creation (Model -1)**

#!/usr/bin/env python3

"""

Python 3 wrapper for identifying objects in images

Running the script requires opencv-python to be installed (`pip install opencv-python`)

Directly viewing or returning bounding-boxed images requires scikit-image to be installed (`pip install scikit-image`)

Use pip3 instead of pip on some systems to be sure to install modules for python3

"""

from ctypes import \*

import math

import random

import os

class BOX(Structure):

\_fields\_ = [("x", c\_float),

("y", c\_float),

("w", c\_float),

("h", c\_float)]

class DETECTION(Structure):

\_fields\_ = [("bbox", BOX),

("classes", c\_int),

("best\_class\_idx", c\_int),

("prob", POINTER(c\_float)),

("mask", POINTER(c\_float)),

("objectness", c\_float),

("sort\_class", c\_int),

("uc", POINTER(c\_float)),

("points", c\_int),

("embeddings", POINTER(c\_float)),

("embedding\_size", c\_int),

("sim", c\_float),

("track\_id", c\_int)]

class DETNUMPAIR(Structure):

\_fields\_ = [("num", c\_int),

("dets", POINTER(DETECTION))]

class IMAGE(Structure):

\_fields\_ = [("w", c\_int),

("h", c\_int),

("c", c\_int),

("data", POINTER(c\_float))]

class METADATA(Structure):

\_fields\_ = [("classes", c\_int),

("names", POINTER(c\_char\_p))]

def network\_width(net):

return lib.network\_width(net)

def network\_height(net):

return lib.network\_height(net)

def bbox2points(bbox):

"""

From bounding box yolo format

to corner points cv2 rectangle

"""

x, y, w, h = bbox

xmin = int(round(x - (w / 2)))

xmax = int(round(x + (w / 2)))

ymin = int(round(y - (h / 2)))

ymax = int(round(y + (h / 2)))

return xmin, ymin, xmax, ymax

def class\_colors(names):

"""

Create a dict with one random BGR color for each

class name

"""

return {name: (

random.randint(0, 255),

random.randint(0, 255),

random.randint(0, 255)) for name in names}

def load\_network(config\_file, data\_file, weights, batch\_size=1):

"""

load model description and weights from config files

args:

config\_file (str): path to .cfg model file

data\_file (str): path to .data model file

weights (str): path to weights

returns:

network: trained model

class\_names

class\_colors

"""

network = load\_net\_custom(

config\_file.encode("ascii"),

weights.encode("ascii"), 0, batch\_size)

metadata = load\_meta(data\_file.encode("ascii"))

class\_names = [metadata.names[i].decode("ascii") for i in range(metadata.classes)]

colors = class\_colors(class\_names)

return network, class\_names, colors

def print\_detections(detections, coordinates=False):

print("\nObjects:")

for label, confidence, bbox in detections:

x, y, w, h = bbox

if coordinates:

print("{}: {}% (left\_x: {:.0f} top\_y: {:.0f} width: {:.0f} height: {:.0f})".format(label, confidence, x, y, w, h))

else:

print("{}: {}%".format(label, confidence))

def draw\_boxes(detections, image, colors):

import cv2

for label, confidence, bbox in detections:

left, top, right, bottom = bbox2points(bbox)

cv2.rectangle(image, (left, top), (right, bottom), colors[label], 1)

cv2.putText(image, "{} [{:.2f}]".format(label, float(confidence)),

(left, top - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

colors[label], 2)

return image

def decode\_detection(detections):

decoded = []

for label, confidence, bbox in detections:

confidence = str(round(confidence \* 100, 2))

decoded.append((str(label), confidence, bbox))

return decoded

# https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/

# Malisiewicz et al.

def non\_max\_suppression\_fast(detections, overlap\_thresh):

boxes = []

for detection in detections:

\_, \_, \_, (x, y, w, h) = detection

x1 = x - w / 2

y1 = y - h / 2

x2 = x + w / 2

y2 = y + h / 2

boxes.append(np.array([x1, y1, x2, y2]))

boxes\_array = np.array(boxes)

# initialize the list of picked indexes

pick = []

# grab the coordinates of the bounding boxes

x1 = boxes\_array[:, 0]

y1 = boxes\_array[:, 1]

x2 = boxes\_array[:, 2]

y2 = boxes\_array[:, 3]

# compute the area of the bounding boxes and sort the bounding

# boxes by the bottom-right y-coordinate of the bounding box

area = (x2 - x1 + 1) \* (y2 - y1 + 1)

idxs = np.argsort(y2)

# keep looping while some indexes still remain in the indexes

# list

while len(idxs) > 0:

# grab the last index in the indexes list and add the

# index value to the list of picked indexes

last = len(idxs) - 1

i = idxs[last]

pick.append(i)

# find the largest (x, y) coordinates for the start of

# the bounding box and the smallest (x, y) coordinates

# for the end of the bounding box

xx1 = np.maximum(x1[i], x1[idxs[:last]])

yy1 = np.maximum(y1[i], y1[idxs[:last]])

xx2 = np.minimum(x2[i], x2[idxs[:last]])

yy2 = np.minimum(y2[i], y2[idxs[:last]])

# compute the width and height of the bounding box

w = np.maximum(0, xx2 - xx1 + 1)

h = np.maximum(0, yy2 - yy1 + 1)

# compute the ratio of overlap

overlap = (w \* h) / area[idxs[:last]]

# delete all indexes from the index list that have

idxs = np.delete(idxs, np.concatenate(([last],

np.where(overlap > overlap\_thresh)[0])))

# return only the bounding boxes that were picked using the

# integer data type

return [detections[i] for i in pick]

def remove\_negatives(detections, class\_names, num):

"""

Remove all classes with 0% confidence within the detection

"""

predictions = []

for j in range(num):

for idx, name in enumerate(class\_names):

if detections[j].prob[idx] > 0:

bbox = detections[j].bbox

bbox = (bbox.x, bbox.y, bbox.w, bbox.h)

predictions.append((name, detections[j].prob[idx], (bbox)))

return predictions

def remove\_negatives\_faster(detections, class\_names, num):

"""

Faster version of remove\_negatives (very useful when using yolo9000)

"""

predictions = []

for j in range(num):

if detections[j].best\_class\_idx == -1:

continue

name = class\_names[detections[j].best\_class\_idx]

bbox = detections[j].bbox

bbox = (bbox.x, bbox.y, bbox.w, bbox.h)

predictions.append((name, detections[j].prob[detections[j].best\_class\_idx], bbox))

return predictions

def detect\_image(network, class\_names, image, thresh=.5, hier\_thresh=.5, nms=.45):

"""

Returns a list with highest confidence class and their bbox

"""

pnum = pointer(c\_int(0))

predict\_image(network, image)

detections = get\_network\_boxes(network, image.w, image.h,

thresh, hier\_thresh, None, 0, pnum, 0)

num = pnum[0]

if nms:

do\_nms\_sort(detections, num, len(class\_names), nms)

predictions = remove\_negatives(detections, class\_names, num)

predictions = decode\_detection(predictions)

free\_detections(detections, num)

return sorted(predictions, key=lambda x: x[1])

if os.name == "posix":

cwd = os.path.dirname(\_\_file\_\_)

lib = CDLL(cwd + "/libdarknet.so", RTLD\_GLOBAL)

elif os.name == "nt":

cwd = os.path.dirname(\_\_file\_\_)

os.environ['PATH'] = cwd + ';' + os.environ['PATH']

lib = CDLL("darknet.dll", RTLD\_GLOBAL)

else:

print("Unsupported OS")

exit

lib.network\_width.argtypes = [c\_void\_p]

lib.network\_width.restype = c\_int

lib.network\_height.argtypes = [c\_void\_p]

lib.network\_height.restype = c\_int

copy\_image\_from\_bytes = lib.copy\_image\_from\_bytes

copy\_image\_from\_bytes.argtypes = [IMAGE,c\_char\_p]

predict = lib.network\_predict\_ptr

predict.argtypes = [c\_void\_p, POINTER(c\_float)]

predict.restype = POINTER(c\_float)

set\_gpu = lib.cuda\_set\_device

init\_cpu = lib.init\_cpu

make\_image = lib.make\_image

make\_image.argtypes = [c\_int, c\_int, c\_int]

make\_image.restype = IMAGE

get\_network\_boxes = lib.get\_network\_boxes

get\_network\_boxes.argtypes = [c\_void\_p, c\_int, c\_int, c\_float, c\_float, POINTER(c\_int), c\_int, POINTER(c\_int), c\_int]

get\_network\_boxes.restype = POINTER(DETECTION)

make\_network\_boxes = lib.make\_network\_boxes

make\_network\_boxes.argtypes = [c\_void\_p]

make\_network\_boxes.restype = POINTER(DETECTION)

free\_detections = lib.free\_detections

free\_detections.argtypes = [POINTER(DETECTION), c\_int]

free\_batch\_detections = lib.free\_batch\_detections

free\_batch\_detections.argtypes = [POINTER(DETNUMPAIR), c\_int]

free\_ptrs = lib.free\_ptrs

free\_ptrs.argtypes = [POINTER(c\_void\_p), c\_int]

network\_predict = lib.network\_predict\_ptr

network\_predict.argtypes = [c\_void\_p, POINTER(c\_float)]

reset\_rnn = lib.reset\_rnn

reset\_rnn.argtypes = [c\_void\_p]

load\_net = lib.load\_network

load\_net.argtypes = [c\_char\_p, c\_char\_p, c\_int]

load\_net.restype = c\_void\_p

load\_net\_custom = lib.load\_network\_custom

load\_net\_custom.argtypes = [c\_char\_p, c\_char\_p, c\_int, c\_int]

load\_net\_custom.restype = c\_void\_p

free\_network\_ptr = lib.free\_network\_ptr

free\_network\_ptr.argtypes = [c\_void\_p]

free\_network\_ptr.restype = c\_void\_p

do\_nms\_obj = lib.do\_nms\_obj

do\_nms\_obj.argtypes = [POINTER(DETECTION), c\_int, c\_int, c\_float]

do\_nms\_sort = lib.do\_nms\_sort

do\_nms\_sort.argtypes = [POINTER(DETECTION), c\_int, c\_int, c\_float]

free\_image = lib.free\_image

free\_image.argtypes = [IMAGE]

letterbox\_image = lib.letterbox\_image

letterbox\_image.argtypes = [IMAGE, c\_int, c\_int]

letterbox\_image.restype = IMAGE

load\_meta = lib.get\_metadata

lib.get\_metadata.argtypes = [c\_char\_p]

lib.get\_metadata.restype = METADATA

load\_image = lib.load\_image\_color

load\_image.argtypes = [c\_char\_p, c\_int, c\_int]

load\_image.restype = IMAGE

rgbgr\_image = lib.rgbgr\_image

rgbgr\_image.argtypes = [IMAGE]

predict\_image = lib.network\_predict\_image

predict\_image.argtypes = [c\_void\_p, IMAGE]

predict\_image.restype = POINTER(c\_float)

predict\_image\_letterbox = lib.network\_predict\_image\_letterbox

predict\_image\_letterbox.argtypes = [c\_void\_p, IMAGE]

predict\_image\_letterbox.restype = POINTER(c\_float)

network\_predict\_batch = lib.network\_predict\_batch

network\_predict\_batch.argtypes = [c\_void\_p, IMAGE, c\_int, c\_int, c\_int,

c\_float, c\_float, POINTER(c\_int), c\_int, c\_int]

network\_predict\_batch.restype = POINTER(DETNUMPAIR)

**CHAPTER – 10**

**REFERENCES**

1. M.V.D. Prasad, E. Kiran Kumar, S.V.N.P Vamsi Krishna, M. Santosh Kumar, P. Sri Harsha, SK. Hasane Ahammad - “Detection of Two Wheelers Helmet Using Machine Learning” – 2020 IEEE *Turkish Journal of Physiotherapy and Rehabilitation (2020).* IEEE, 2020.
2. Felix Wilhelm Siebert, Hanhe Lin - “Detecting motorcycle helmet use with deep learning*”* – 2017 IEEE *Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017.
3. Thirunavukkarasu.M, Bugade Amoolya, Bulusu Vyagari Vaishnavi - “Helmet Detecting and license plate recognition “- IJCSMC, 2021.
4. Rui Geng, Yixuan Ma, Wanhong Huang- “An improved helmet detection method for YOLOv5 on an unbalanced dataset” - 2019 IEEE *16th International Conference on Networking, Sensing and Control (ICNSC)*, IEEE, 2019.
5. Jie Li, Huanming Liu, Tianzheng Wang - “ Safety helmet wearing detection based on image processing and machine learning” - 2017 IEEE Ninth International Conference on Advanced Computational Intelligence (ICACI) - IEEE, 2017.
6. Fangbo Zhou, Huailin Zhao, Zhen Nie - “ Safety Helmet Detection Based on YOLOv5 “ - 2021 IEEE *International Conference on Power Electronics, Computer Applications (ICPECA)* - IEEE, 2021.
7. Kunal Dahiya, Dinesh SINGH, C. Krishna Mohan - “Automatic Detection of bike riders without helmet using surveillance videos in real time” – *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020.

[8] C. Vishnu, Dinesh Singh, C. Krishna Mohan and Sobhan Babu – “Detection of Motorcyclists without Helmet in Videos using Convolutional Neural Network” – IEEE, 2020

[9] Madhuchhanda Dasgupta, Oishila Bandyopadhyay, Sanjay Chaterji, "Automated Helmet Detection for Multiple Motorcycle Riders using CNN", *Information and Communication Technology (CICT) 2019* IEEE, 2019.

[10] C.Y. Wen, S.H. Chiu, J.J. Liaw and C.P. Lu “The Safety Helmet Detection for ATM’s Surveillance System via the Modified Hough transform”. *Security Technology, IEEE 37th Annual International Carnahan Conference, pp. 364–369.* IEEE (2003).

[11] Romuere R.V e Silva, Kelson R.T Aires, Rodrigo de M.S Veras - “Helmet Detecting using Image Descriptors and Classifiers” – IEEE, 2019.

[12] Meenu R, Sinta Raju, Smrithi P Paul, Swathy Sajeev, Alphonsa Jhony

“Detection of Helmetless rider using faster RCNN” -IEEE, 2020.

[13] Lokesh Allamki, Manjunath Panchakshari, Ashish Sateesha, K. S Pratheek - “Helmet Detection using Machine Learning and Automatic License Plate Recognition” IJERT, 2019.

[14] M.V.D. Prasad, S.V.N.P. Vamshi Krishna, M. Santhosh Kumar, P. Sri Harsha - “Helmet Detecting using ML & IoT” - IJERT, 2020.

[15] Romuere Rodrigues Veloso e Silva, Kelson Romulo Teixeira Aires, Rodrigo

de Melo Souza Veras - ” Helmet Detection on Motorcyclists Using Image

Descriptors and Classifiers” - 27th SIBGRAPI Conference on Graphics, Patterns

and Images – IEEE 2017.