

REAL-TIME DETECTION OF PERSONAL
PROTECTIVE EQUIPMENT FOR SITE
SAFETY USING DEEP LEARNING
TECHNIQUES

MUHAMMAD HADIF BIN DZULKHISSHAM

B.ENG (HONS.) ELECTRICAL
ENGINEERING (ELECTRONICS)

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : Muhammad Hadif Bin Dzulkhissham

Date of Birth : 27 December 1997

Title : Real-Time Detection of Personal Protective Equipment
for Site Safety Using Deep Learning Techniques

Academic Session : Semester II 2021/2022

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:



(Student's Signature)

971227-10-5803

New IC/Passport Number

Date: 17/6/2022



(Supervisor's Signature)

Ikhwan Hafiz Bin Muhamad

Name of Supervisor

Date: 21/6/2022

NOTE: * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and in my opinion, this thesis is adequate in terms of scope and quality for the award of the Bachelor of Electrical Engineering (Electronics) with Honours.



(Supervisor's Signature)

Full Name : Ikhwan Hafiz Bin Muhamad

Position : *Lecturer*

Date : *21/06/2022*



STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

A handwritten signature in black ink, consisting of a vertical stroke followed by a loop and a horizontal tail, is positioned above a horizontal line.

(Student's Signature)

Full Name : Muhammad Hadif Bin Dzulkhissham

ID Number : EA18167

Date : 17/6/2022

REAL-TIME DETECTION OF PERSONAL PROTECTIVE EQUIPMENT FOR
SITE SAFETY USING DEEP LEARNING TECHNIQUES

MUHAMMAD HADIF BIN DZULKHISSHAM

Thesis submitted in fulfillment of the requirements
for the award of the
B.Eng (Hons.) Electrical Engineering (Electronics)

College of Engineering
UNIVERSITI MALAYSIA PAHANG

JUNE 2022

ACKNOWLEDGEMENTS

First and foremost, I am grateful to the Allah for the given time, a good health and wellbeing that was crucial for me to complete this thesis report. I wish to express my sincere thanks to my parents because without their tremendous understanding and encouragement during my research project, it would be impossible for me to continue my research. I would like to convey my appreciation to Sir Ikhwan Hafiz Bin Muhamad, for the invaluable advice, continuous support, and patience in guiding me during my PSM 2. I am also grateful to my fellow friends who always support and encourage me to complete my PSM 2. Their kind help and support have made my research a wonderful journey.

ABSTRAK

Kecederaan otak traumatik (akibat jatuh dan renjatan), terseliuh, patah tulang, dan kecederaan lain boleh disebabkan oleh tergelincir dan jatuh di atas tanah, gas bocor yang berbahaya untuk disedut dan pelanggaran adalah punca utama kematian di kawasan pembinaan (akibat daripada dihentak oleh objek). Jabatan Keselamatan dan Kesihatan Pekerjaan (JKKP) di Malaysia mewajibkan kontraktor untuk sentiasa menguatkuasakan dan memantau peralatan perlindungan diri (PPE) yang mencukupi untuk pekerja (contohnya, topi keledar keras dan vest) sebagai langkah pencegahan. Di samping itu, kerana wabak COVID-19 sejak dua tahun yang lalu, memakai topeng muka di kilang, jabatan, atau pejabat kerja adalah kritikal. Laporan ini membentangkan teknik pembelajaran mendalam untuk mengesan pelbagai peralatan perlindungan diri pekerja secara serentak berdasarkan algoritma pengesanan objek You-Only-Look-Once Versi 4 (YOLOv4). Keseluruhan proses latihan atau pengiraan akan dilakukan di Google Colaboratory. Dengan menggunakan kaedah ini, hasil latihan menunjukkan bahawa Min Average Precision (mAP) untuk latihan terbaik adalah 97.04% untuk mengesan pelbagai alat perlindungan perseorangan.

ABSTRACT

Traumatic brain injuries (from falls and electrocution), sprains, broken bones, and other injuries can result from slipping and falling on the ground, leaking gas that is hazardous to inhale and collisions are the primary causes of construction fatalities (resulting from being struck by objects). The Department of Occupational Safety and Health (DOSH) in Malaysia mandates contractors to always enforce and monitor adequate Personal Protective Equipment (PPE) for workers (e.g., hard helmet and vest) as a preventative measure. In addition, because of the COVID-19 outbreak over the last two years, wearing a face mask in factories, departments, or working offices is critical. This paper presents a deep learning technique for detecting multiple personal protection equipment at once based on the You-Only-Look-Once Version 4 (YOLOv4) object detection algorithm. The whole training process or computation is done in Google Colaboratory. The training result shows that the Mean Average Precision (mAP) for the best weight training is up to 97.04% for detecting multiple PPE by using this method.

TABLE OF CONTENT

DECLARATION	
TITLE PAGE	
ACKNOWLEDGEMENTS	ii
ABSTRAK	iii
ABSTRACT	iv
TABLE OF CONTENT	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF ABBREVIATIONS	x
CHAPTER 1 INTRODUCTION	1
1.1 Introduction	1
1.2 Problem Statement	2
1.3 Objective	3
1.4 Scope of works	3
1.5 Thesis Outline	4
CHAPTER 2 LITERATURE REVIEW	5
2.1 Introduction	5
2.1.1 Sensor Based Personal Protective Equipment Detection Approach	5
2.1.2 Personal Protective Equipment with Image Detection Approach	6
2.1.3 Real-Time Personal Protective Equipment Detection Using Deep Learning Approach	8

2.1.4	Proposed Method	10
2.2	Difference between Machine Learning and Deep Learning	10
2.3	YOLOv4 Overview	11
2.4	Review Conclusion	12
CHAPTER 3 METHODOLOGY		14
3.1	Introduction	14
3.2	Overall Research Process	14
3.2.1	System Environment Setup	16
3.2.2	Dataset Preparation	16
3.2.3	Hyperparameters improvement	18
3.2.4	Training Model	19
3.2.5	Validation of training weight	19
3.3	Gantt Chart	24
CHAPTER 4 RESULTS AND DISCUSSION		25
4.1	Introduction	25
4.2	Evaluation Measures	25
4.3	Mean Average Precision (mAP)	28
4.4	Average Loss	29
4.5	Testing model on real-time frame (webcam feed)	31
4.5.1	Run the detection model in different test case	32
4.6	YOLOv4 comparison (Darknet Framework vs TensorFlow Framework)	33
CHAPTER 5 CONCLUSION		35
5.1	Conclusion	35

5.2 Recommendation	36
REFERENCES	37
APPENDIX A SAMPLE APPENDIX 1	40
APPENDIX B	44
APPENDIX C	48
APPENDIX D	55

LIST OF TABLES

Table 3.1: Image dataset for each classes	17
Table 4.1: Evaluation results of the validate dataset	28
Table 4.2: Mean Average Precision for each training iteration	29
Table 4.3: Detection performance for real-time	31
Table 4.4: Mask test case	32
Table 4.5: Helmet test case	32
Table 4.6: Vest test case	33
Table 4.7: Performance comparison	33

LIST OF FIGURES

Figure 2.1: RFID and tag system attach to Personal Protective Equipment [7]	6
Figure 2.2: Overall Research Process [10]	7
Figure 2.3: YOLO working principle [18]	11
Figure 2.4: YOLOv4 block diagram [19]	12
Figure 2.5: Object Detector Comparison [20]	13
Figure 3.1: Project development flowchart	15
Figure 3.2: 17,883 images dataset for PPE	15
Figure 3.3: Example of image annotation	17
Figure 3.4: Detector.c file in darknet directory	18
Figure 3.5: YOLOv4 configuration parameter	18
Figure 3.6: Training Command Line	19
Figure 3.7: Dependencies imported	20
Figure 3.8: Source code to import darknet function	20
Figure 3.9: Source of Javascript conversion to OpenCV	21
Figure 3.10: JavaScript code to enable webcam as input	22
Figure 3.11: Source code to start webcam stream and create bounding box detection	23
Figure 3.12: Source code for decision making using OpenCV	23
Figure 3.13: Gantt Chart for Projek Sarjana Muda 2	24
Figure 4.1: Classification report for the all dataset	26
Figure 4.2: Recall Chart	26
Figure 4.3: Precision Chart	27
Figure 4.4: F1 Score Chart	27
Figure 4.5: mAP command line	28
Figure 4.6: mAP vs Loss Chart	30

LIST OF ABBREVIATIONS

PPE	Personal Protective Equipment
YOLO	You-Only-Look-Once
CNN	Convolutional Neural Network
mAP	Mean Average Precision
RFID	Radio-Frequency Identification
TP	True Positive
FP	False Positive
FN	False Negative

CHAPTER 1

INTRODUCTION

1.1 Introduction

Construction is one of the world's most important industries. However, because of the high incidence of workplace accidents and worker injuries, this industry is one of the most dangerous of all. Construction had the largest total number of fatal occupational injuries in the United States in 2016-17, compared to all other industries. According to the Bureau of Labour Statistics (BLS), 991 fatal events (or 19 percent of all fatalities) occurred over this period [1]. Furthermore, the number of nonfatal occupational injuries and illnesses in construction jobs was 79,810 in 2017, which was similarly excessive [1]. The "fatal four" – falls, being struck by an object, electrocutions, and being caught in/between – were responsible for over 60% of construction worker deaths in 2017 [2].

Malaysia has moved quickly to keep up with the times as it strives for robust economic growth and to define its own future. One of the most crucial industries for a country's development is construction. However, the building sector contributes to the high accident rate, which is in accordance with the economy's healthy expansion [3]. In 2017, the construction industry remained one of Malaysia's most important sectors, employing over 1.33 million people, accounting for 9.1% of total employment. According to the Social Security Organisation (SOCSO), there were 7,338 accidents reported in the construction sector in 2016, up from 4,330 cases reported in 2011, a 69.47 percent rise [3]. According to the Department of Occupational Safety and Health (DOSH), 106 deaths in the construction industry were reported in 2016, compared to 88 in 2015. SOCSO and DOSH statistics show that the number of fatal accident cases has increased by 231.9 percent and 125.8%, respectively, in the last five years [3]

Most of these injuries/fatalities might have been avoided if workers had worn proper personal protective equipment (PPE), particularly a hard hat and a safety vest. The Occupational Safety and Health Administration (OSHA), as well as similar agencies in other countries, mandate that all personnel working near site hazards wear appropriate personal protective equipment (PPE) to reduce the risk of being exposed to or injured by hazards [3]. It is the obligation of both employers and employees to maintain workplace safety and health under OSHA 1994. Employers and contractors who fail to create a safe and healthy working environment for their employees may be prosecuted under Section 15 of OSHA 1994, which has a maximum penalty of RM50,000 in fines, two years in prison, or both [4].

In this project, cameras are used to capture images or recordings of the site, which are then analysed to ensure PPE compliance. This method delivers more detailed information on the scene, which can be used to better understand complex building sites more quickly, precisely, and thoroughly. Multiple Personal Protective Equipment will be detected using deep learning technique.

1.2 Problem Statement

The current commercialized method of detecting and monitoring PPE compliance necessitated the installation of a sensor within the PPE. Most construction companies in Malaysia still using this approach to detect worker PPE that equipped with RFID tags passing through the gate, it will be detected by an RFID reader or an antenna system. For example, some projects using RFID tags affixed on each PPE compliance and scanning the tag with a scanner to see if the workers are correctly wearing PPE. This method is quite costly due to sensor purchasing, instalment and maintenance in each PPE on each worker [5].

1.3 Objective

This project embarks the following objectives:

- i. To develop a real-time multiple Personal Protective Equipment (PPE) detection approach using YOLOv4 object detection algorithm with Darknet framework and Python.
- ii. To achieve a sustainable result in terms of accuracy, precision, and speed in real-time computation.
- iii. To evaluate the overall performance of the algorithm used in terms of accuracy, precision, and speed.

1.4 Scope of works

Scopes of this project includes:

- i. Multiple classes of PPE will be included such as safety hat, safety mask, and vest for the dataset.
- ii. Using You-Only-Look-Once (YOLOv4) real-time object detection algorithm in Darknet Framework.
- iii. To detect if a worker is wearing all PPE appropriately.
- iv. Computation will be done in Google Colaboratory.
- v. Lighting and white background will be used for controlled environment.
- vi. Computer webcam are used for real-time detection testing purposes.

1.5 Thesis Outline

Chapter 1 explain about the whole research including expected results or outcomes.

Chapter 2 explain about the literature review of research done by other authors in implementing the object detection in same manners of detecting PPE. In this chapter also explain the significant of proposing the method that will be used in this project compared to other projects discuss in literature review.

Chapter 3 explain about the research methodology applied in this project, as well as some of the techniques and software involved in the development of the multiple PPE object detection.

Chapter 4 explain about the whole discussion of the performance of the project by carrying out analysis on the results obtained under different aspects, thus clearly seeing the outcome of this whole project.

Chapter 5 concludes thesis with summary of the contributions and suggestions of the future research direction with regards to the issue.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, three methods were discussed. The first method is Sensor Based Approach which this approach will be discussing about physical component or material attached to PPE of a workers to make a detection. This approach gives only detect or not detect result after detection has been made. On contrary, the Vision-Based Approach which come up with two different method such as Image Detection method and Real-Time Detection method. Both methods had something on common which utilizing the computer vision to make a detection.

2.1.1 Sensor Based Personal Protective Equipment Detection Approach

The broadly existing method for Personal Protective Equipment (PPE) detection has two different categories which are sensor-based and vision-based. The general idea of this sensor-based PPE detection method is to detect the installed sensor in each PPE. Sensors such as photoresistors, optical sensors, force stretchable resistors, and touch sensors are mainly used. This general idea can be broadened widely in the system by using the internet of things (IoT) with wireless Wi-Fi modules tagged on the PPE and these sensors are embedded with near real time data processing algorithms for proper wearing detection [6]. This algorithm has been implemented in some industries such as mining industries where the Radio-Frequency Identification is used to control workers PPE, control of personnel to access mining sites and RFID solutions for tracking explosives. Each PPE component must have an RFID tag attached to regulate the Personal Protective Equipment (PPE). When a person wearing RFID-enabled PPE passes through a gate, it is recognised by an RFID reader, or an antenna system as shown in Figure 2.1. This means, to control the Personal Protective Equipment (PPE) required each PPE component

must be attached with an RFID tag [7]. Another similar approach has been investigated by Zhang et al., where Global Positioning System (GPS) were used to track or locate the workers and safety helmet [8]. However, according to Nath et al., with nowadays technology this method is considered outdated due to cost ineffective as it requires a significant investment in purchasing, installing, and maintaining the sensor network stability in real life practice. [5]

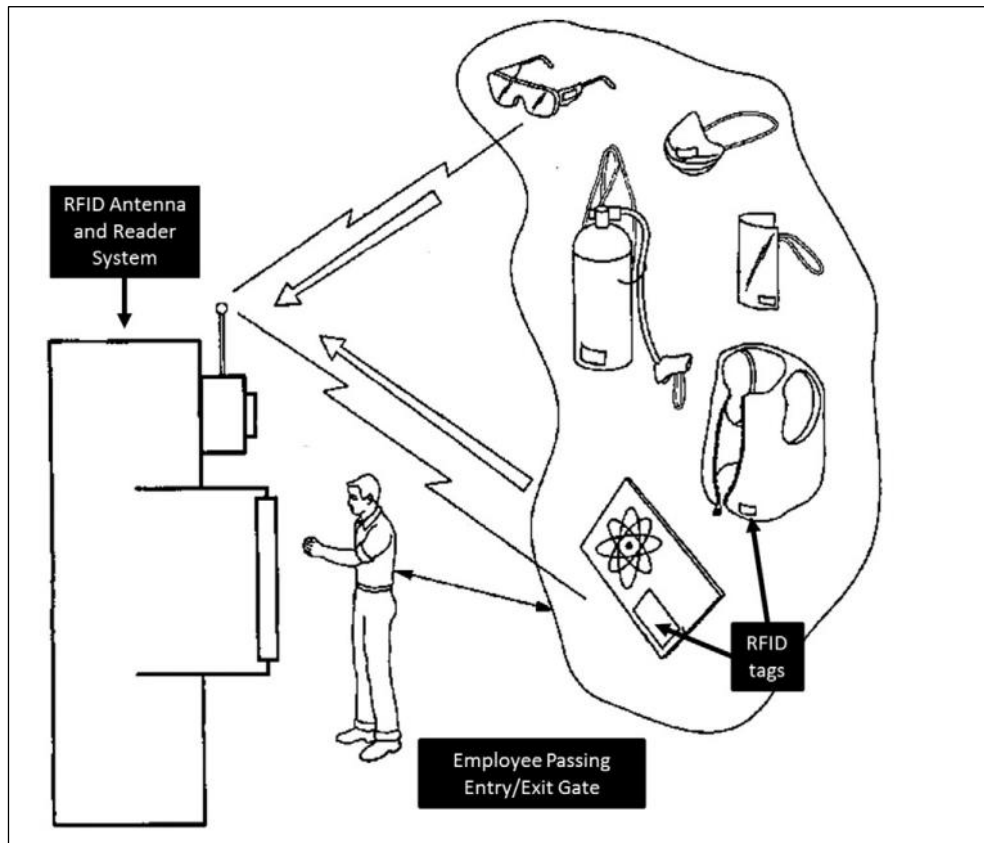


Figure 2.1: RFID and tag system attach to Personal Protective Equipment [7]

2.1.2 Personal Protective Equipment with Image Detection Approach

Image detection is one of vision-based approaches to analyse and verify PPE compliance by using cameras. This approach is indeed more sophisticated than using Sensor-Based approach since it provides richer information about the scene on construction site that has been captured through cameras [5]. However, the shortcomings of the camera (e.g., low resolution, limited range of view) used can be one of the reason this approach is not suitable for real-time implementation as it will give a bad result according to quality of

the camera. According to Asgrawal et al., this approach is using the information of the detection image to improve workplace safety practices only [9]

Using the Faster Region-based Convolutional Neural Networks (R-CNN) method, the author presents an image detection model for workers' safety situations based on PPE compliance [10]. TensorFlow was used to execute this experiment, which used 1,129 photos from the MIT Places Database (from Scene Recognition) as a training dataset and 333 anonymous dataset images from real construction sites as a training dataset. Figure 2.2 shows the overall research process of proposed method.

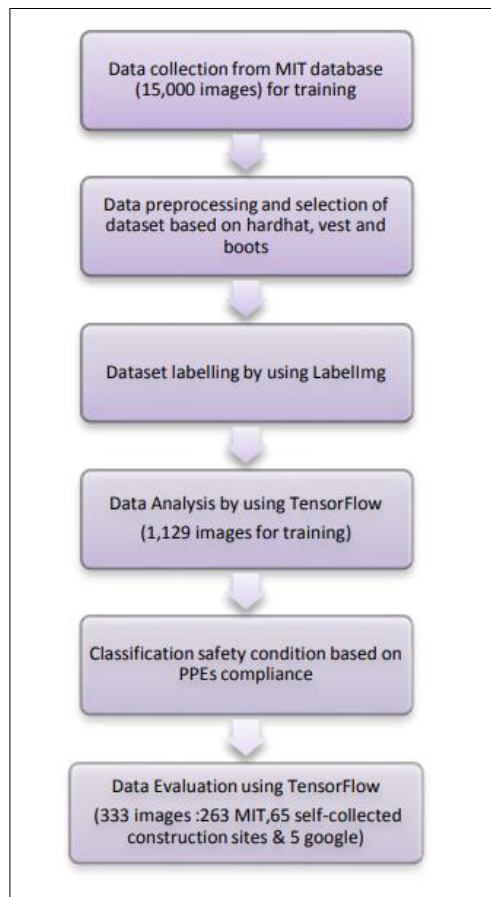


Figure 2.2: Overall Research Process [10]

Another method, the author suggested the model that outlines the creation of a computer vision system for detecting and classifying clothing in e-commerce images by using YOLOv3 and Residual Networks architectures. The DeepFashion dataset is used which contains box annotations for locations of clothes, and manually collected data for training and testing the clothes detection network and classification network. The suggested

models use bounding boxes to identify clothes and then classify the colour of the identified clothing. The experimental findings show that the suggested CNN architectures are the most efficient and effective network topologies for detecting and classifying clothing.

Another approach used Amazon Rekognition for PPE detection [9]. To detect PPE in an image, DetectProtectiveEquipment API is applied and pass an input image. The input image (in JPG or PNG format) either as raw bytes or as an object stored in an Amazon Simple Storage Service (Amazon S3) bucket. A summarization result obtained with a confidence score of above 80%. The response includes a consolidated per-image identifier (ID) summary of persons with the required PPE, persons without the required PPE, and persons where a determination could not be made. The drawback of this detection is the detection detects multiple PPE only one frame per time. It unable to detect whether the PPE is being detected properly in real time frame. Besides that, it will be time costly to detect every worker's PPE in the site because after capturing an image. The result in detect will going take some time for the detection to be made upon that image.

2.1.3 Real-Time Personal Protective Equipment Detection Using Deep Learning Approach

In modern years, deep learning technology have gotten a lot of interest in computer vision due to its capacity to self-learn relevant features from large-scale, annotated training data [11]. Fundamentally, Convolutional Neural Network (CNN) has being the most used in deep learning technology especially for image classification and object detection. For example, Zhafran et al., used CNNs to monitor directly and detect workers who do not wear PPE and the system can give a warning if there are workers who do not wear PPE completely and could be implemented into several workplaces that have the requirements for the detection [12] In his research, he obtained accuracy percentage of 79.14% and the precision of 80%. Although the system has been successfully trained and implemented, the accuracy and precision are still lower than other object detection such as YOLO.

On top of that, as the pandemic continue to cause a global health crisis all around the world, Saran et al., has developed a deep learning-based system using CNNs architecture that can detect both masked and unmasked faces and can be integrated with pre-installed CCTV cameras [13]. The model has a total of 1,727 and 320 training and validation

sample respectively. The result obtained from training has a final loss and final accuracy of 0.524 and 99.783 respectively. CNNs excel at comprehending visual data and data that is not presented in a logical order. They fall short, however, when it comes to deciphering temporal information like videos (which are simply a series of discrete pictures) and text blocks.

The author also studied about CNNs model approach and develop using transfer learning with based version of YOLOv3 to make a prediction of PPE compliance [14]. On the test dataset, the model received an F1 score of 0.96, with an average accuracy and recall rate of 96 percent. NOT SAFE, SAFE, NoHardHat, and NoJacket are the four categories in which the model predicts compliance. A total of 2,509 photos were acquired from video recordings made at various building sites and used to train the model on the web.

The model proposed by Vinh et al., as he employed the Haar cascade classifier, YOLOv3, and data from the MAFA dataset to determine whether someone is wearing a mask [15]. Pre-processing, face detection, and mask detection are the three processes of their proposed approach. Colour consistency of the input image is assured during pre-processing utilising auto white balance. The unsharp filter is then used to enhance the edges in the supplied image. The Haar cascade classifier is used to recognise faces in the processed picture frame in the following phase. Following face detection, trained YOLOv3 determines whether there is a mask covering the face. They used the MAFA dataset which is a dataset platform to collect 7000 images. A total of 5000 images are utilised to train the YOLOv3 model. Precision and recall were used to assess the performance. The accuracy of the system achieved is 90.1%. The system can work in real time with 30 fps.

Another model employing the YOLOv4 object detection algorithm that proposed by Protik et al., offered a way to detect part of the PPE for COVID-19 [16]. A mixed dataset in the solution that contains both collected and captured images with augmentation implementation applied. The training weight of detector then converted to TensorFlow format to be able to check live object detection performances. With this conversion, additional features like live object count and keeping records can be added into the system. Results from the tests and mAP of the object detector is up to 79 percent.

2.1.4 Proposed Method

Based on the literature review, the proposed method in this project employed You-Only-Look-Once (YOLO) Version 4 object detection and Darknet for implementation of the model to make a detection of multiple PPE such as safety helmet, safety mask and vest.

This system is proposed to compare the result of the latest literature review project which is using YOLOv4 and Tensorflow framework. Apparently, Darknet complement with YOLO well unlike stand-alone framework like Tensorflow. Darknet is the name of the framework YOLO is originally implemented on.

The system consists of six classes for each individual PPE such as With_Mask (WM), No_Mask (NM), With_Helmet (WH), No_Helmet (NH), With_Vest (WV), No_Vest (NV). The model then used to train the dataset and with the trained weight used to validates the detector in image and real-time (webcam-feed).

2.2 Difference between Machine Learning and Deep Learning

Machine learning is a traditional way to build object detection model. The computation speed is slower because the machine learning utilizing only CPU resulting in slower training process over a large dataset.

In term of accuracy, ML has a limitation for a complex model. The complex dataset needs the model to train the image with different complexity factor such as the image background environment, the large number of individual classes, the shape of PPE, the position of PPE, colour of PPE, colour of images etc. An error in dataset would cost the model to have a very low accuracy compared to deep learning where the model itself learn to adapt with an error.

Deep learning neural networks require a significant quantity of data to learn from because they rely on layered knowledge without human intervention. Machine learning, on the other hand, is based on a guided evaluation of data sets that are still huge but significantly smaller [17]

2.3 YOLOv4 Overview

You-Only-Look-Once (YOLO) is an object detecting system that works in real time. The primary distinction between YOLO algorithms and other object detection algorithms is that because of their speed, they can identify things in real time and make accurate predictions.

YOLO algorithm works by dividing the image into N grids, each having an equal dimensional region of $S \times S$. Each of these N grids is responsible for the detection and localization of the object it contains. The network outputs a class probability and offset values for each bounding box for each bounding box [18]. The bounding boxes with a class probability greater than a threshold value are chosen and utilised to find the item in the picture as shown in Figure 2.3.

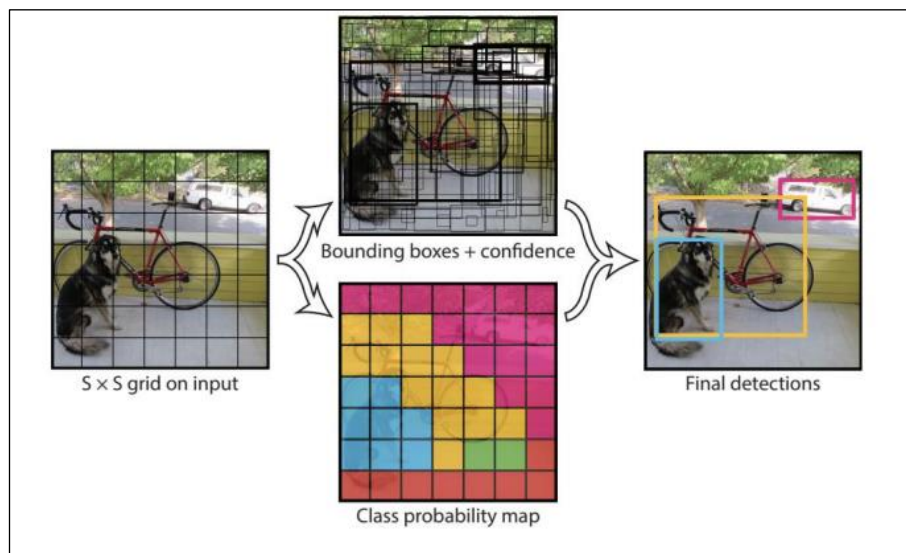


Figure 2.3: YOLO working principle [18]

Backbone, neck, and head parts are included in the YOLOv4 model as shown in Figure 2.4. The feature extractor model CSPDarknet53 is employed in the backbone component of the model [19]. In the neck component of the model, Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN) are employed. For example, in YOLOv4, modified PAN was utilised for segmentation. They used the concatenation operation instead of the addition procedure to modify PAN. The SPP is used to perform maximum pooling over a feature map. The accuracy of the model is improved by combining feature maps with

the concatenation technique. The head component of YOLOv4 was kept the same as it was in YOLOv3.

YOLOv4 concentrated on enhancing the existing version's accuracy and speed [20]. They employed two sorts of strategies to enhance accuracy and speed: bag of freebies (BoF) and bag of specials (BoS). BoF approaches assist to enhance accuracy without increasing the cost of inference. The inventors of the BoF category have developed a new data augmentation approach called Mosaic. In the mosaic data augmentation approach, they blended four separate photos into one single image. The DropBlock regularisation technique is utilised in YOLOv4. DropBlock is a method of organised dropout. This allows YOLOv4 to have better accuracy.

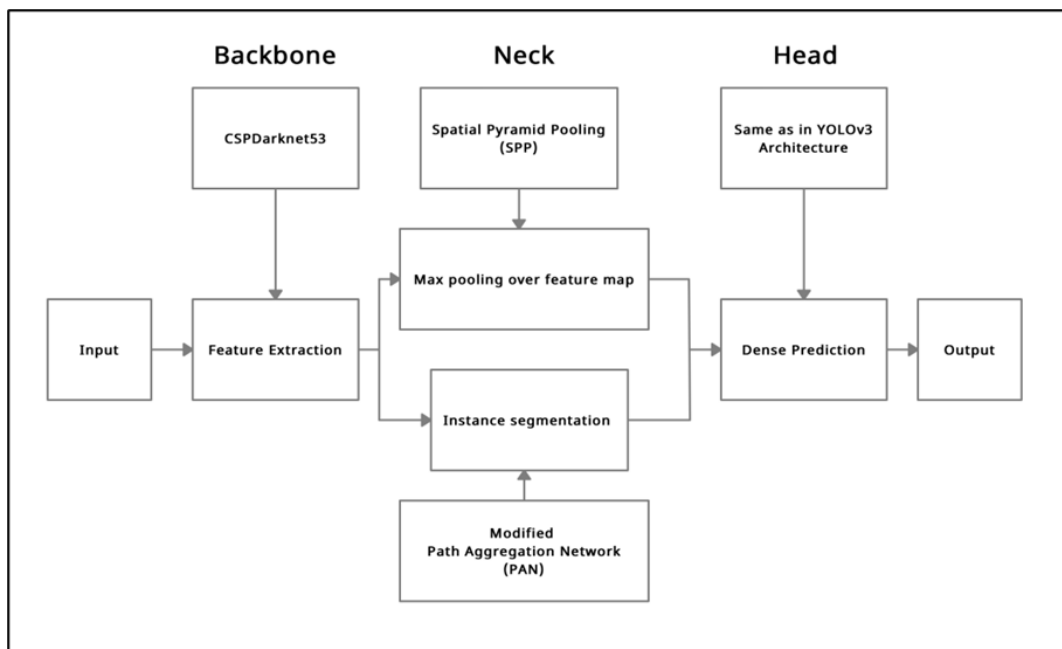


Figure 2.4: YOLOv4 block diagram [19]

2.4 Review Conclusion

YOLO has advantage of speed, in addition to higher prediction accuracy and a superior Intersection over Union in bounding boxes (when compared to real-time object detectors). YOLO is a lot quicker algorithm than its competitors, reaching speeds of up to 45 frames per second.

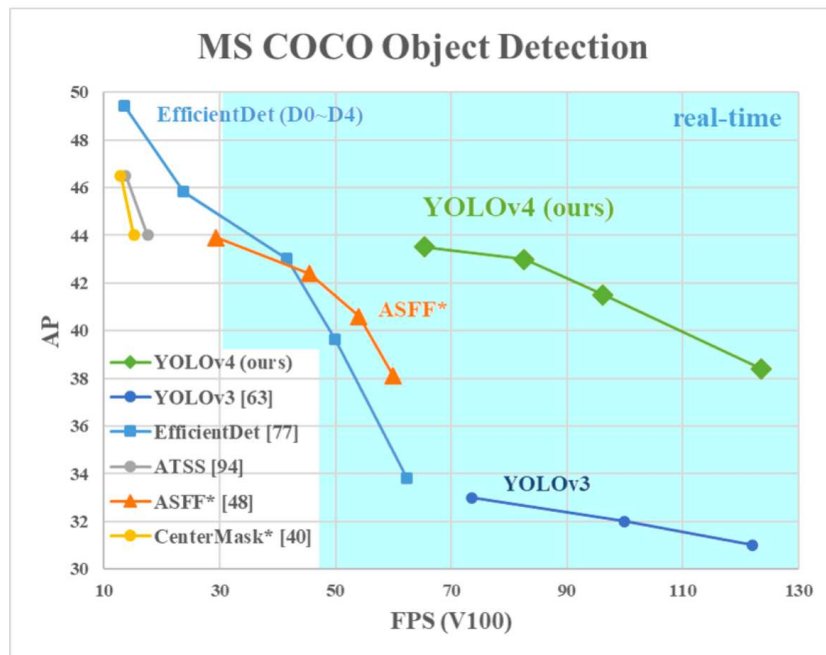


Figure 2.5: Object Detector Comparison [20]

Based on Figure 2.5, YOLOv4 achieved 0.435 average precision (AP), running at 62 frames per second (FPS). YOLOv4 runs twice faster than EfficientDet with comparable performance. Improving YOLOv3's AP and FPS by 10% to 12% respectively. YOLOv4 is very fast in real time object detection while maintaining accuracy. With MS COCO dataset it gives AP of 43.5% at 62 FPS using Tesla V100 GPU [20]. Therefore, YOLO Version 4 object detection will be used in this project for implementation of the model to make a detection of multiple PPE simultaneously such as safety helmet, safety vest and safety mask.

CHAPTER 3

METHODOLOGY

3.1 Introduction

Object detection is a more sophisticated kind of image classification in which a neural network predicts and highlights things in an image using bounding boxes. It is linked to computer vision and image processing. Hence, this chapter is focusing on method of how the detection can be done using object detection algorithm called You-Only-Look-Once (YOLO), Darknet Framework and Python. The fundamental idea of this system is to make detection in real-time so that it can be implement in real industries. The computation and training process will be done in cloud notebook server called Google Colaboratory.

3.2 Overall Research Process

The overall process shown in Figure 3.1. The project will start from collecting the sample of dataset from many sources from the internet as well as manually collected by using video recording. The sources for dataset is obtained from Kaggle, Prajnasb Github, X-zhangyang Github, Joseph Nelson Roboflow, Pictor-PPE Github. Total sample images of PPE collected 17,883. Some example of the images dataset obtained is shown in Figure 3.2.

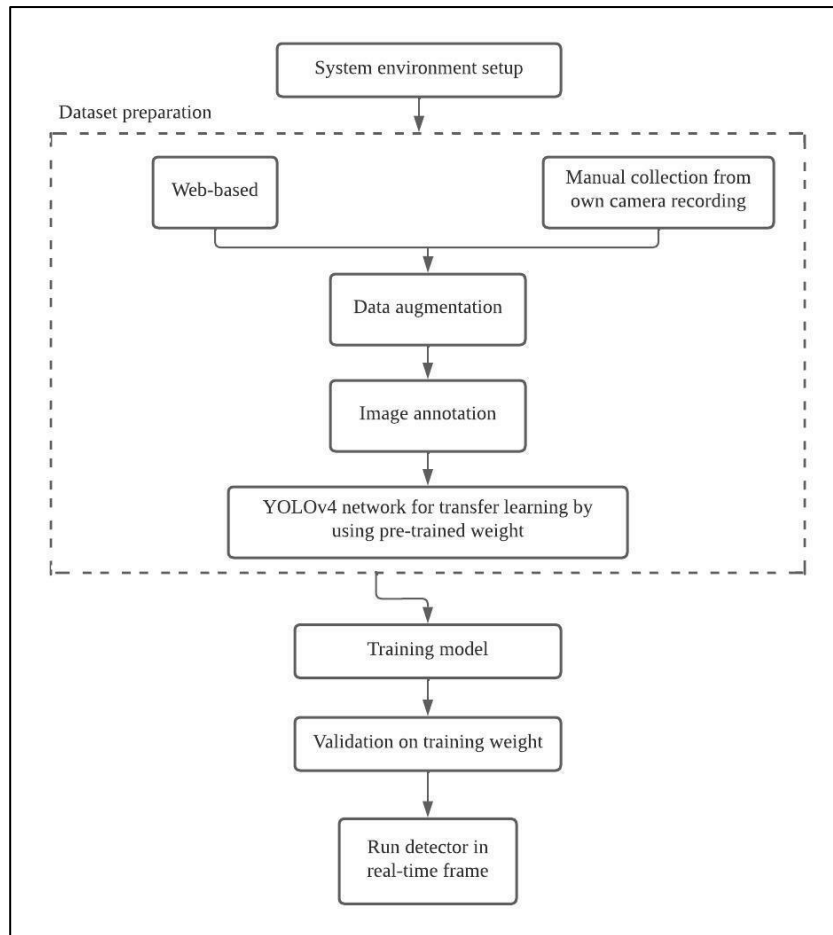


Figure 3.1: Project development flowchart



Figure 3.2: 17,883 images dataset for PPE

3.2.1 System Environment Setup

The basic environment setup consists of darknet, python3, OpenCV. The image annotation is done using the OpenLabelling software tool. The computation was done on a Google Colab notebook. In the Darknet framework, the YOLOv4 model is employed for transfer learning. By adjusting the filter size, the final output layer is updated to output six classes: no_mask, no_helmet, no_vest, mask, helmet and vest.

3.2.2 Dataset Preparation

The collection and preparation of data to enable the validation of the model was a crucial aspect of training the machine learning algorithm. The most time-consuming and crucial component is the dataset preparation, as it allows for fast training and accurate identification by the algorithm. Data is gathered through both manually collecting and from the web-based collection.

To begin, data was collected manually of a person wearing mask or without mask. After that, the frames from the videos are retrieved as images. Other alternative of collecting sample dataset is by obtaining on the internet sites. To add variation to the training data, a few images with different background were kept.

The data was labelled using OpenLabelling tool, a graphical image annotation tool once the dataset was gathered. With bounding boxes, the photos were labelled according to the six classes (no_mask, no_helmet, no_vest, mask, helmet and vest). YOLO_darknet files in .txt were used to save annotations as shown in Figure 3.3. These files were used for training the object detection in YOLO environment.

After all datasets have been labelled, data augmentation was conducted using basic augmentations such as flipping, rotating 30 degrees right, and 30 degrees left of the images that have been labelled. Applied transformation for both the images and labels together. The study's final data set included 17,883 images. The full coding of the image augmentation is at **Appendix D**.

Table 3.1:Image dataset for each classes

Classes	Total images
no_mask	2500
no_helmet	3000
no_vest	3383
mask_detected	3000
helmet_detected	3000
vest_detected	3000
Total images	17,883
Training set (90%)	16,095
Test set (10%)	1,788

A training set is used to develop a model in a dataset, whereas a test (or validation) set is used to test the model. The test (validation) set excludes data points from the training set. Usually, a dataset is divided into a training set, a validation set (some people use ‘test set’ instead) in each iteration. In deep learning, the objective of model built to predict the test data. Therefore, the training data to fit the model and testing data to test it. The bigger the ratio of dataset to train is better [21].

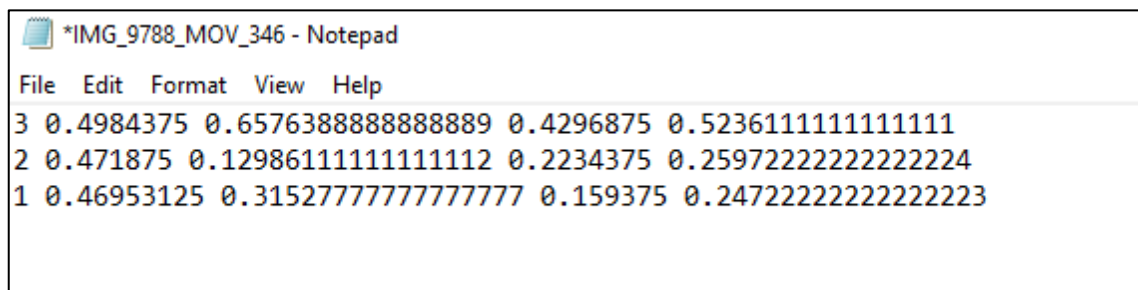
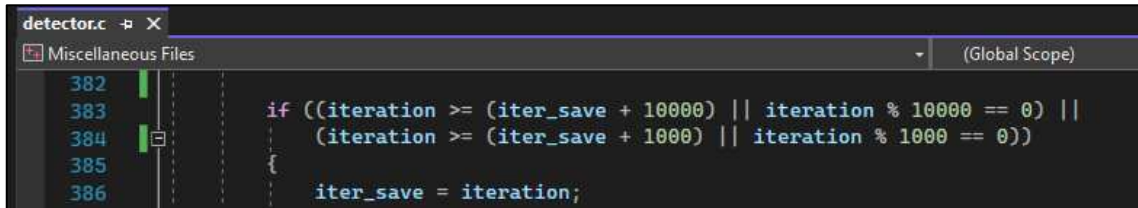


Figure 3.3: Example of image annotation

The YOLO label file contains 5 values as shown in Figure 3.3. The first value is that of the class index id. The remaining four values in the file are object coordinates and the height and width of the bounding box.


3.2.3 Hyperparameters improvement



```
382  
383     if ((iteration >= (iter_save + 10000) || iteration % 10000 == 0) ||  
384         (iteration >= (iter_save + 1000) || iteration % 1000 == 0))  
385     {  
386         iter_save = iteration;
```

Figure 3.4: Detector.c file in darknet directory

By adding this additional code in detector.c file in darknet directory as shown in Figure 3.4, it will enable the training iteration to be more than 10,000 as the higher the number of classes, the higher the training iterations and to save a weight for every 1000 iterations.



```
gear yolov4-custom.cfg ●  
C: > Users > User > Downloads > gear yolov4-custom.cfg  
1 [net]  
2  
3 batch=32  
4 subdivisions=16  
5 width=416  
6 height=416  
7 max_batches = 12000  
8 steps=9600,10800  
9 learning_rate=0.001
```

Figure 3.5: YOLOv4 configuration parameter

The batch size, which is the number of images required to train a single forward and backward pass, is one of the most essential hyperparameters. As shown in Figure 3.5, the batch size will load 32 images for one iteration and average error will be calculated and then update weights every 32 images.

To train the network faster, the subdivision value will be used to split batch into 16 mini-batches. Hence, the network will produce 2 images to be sent for processing. This process will be performed 16 times until the batch is completed and a new iteration will start with 32 new images.

Learning rates allow model to converge slowly and steadily over complex and high quantity dataset. To controls how quickly the model is adapted to the estimated error on each weight updated. Angle, saturation, exposure, and hue are random image characteristics adjusted during training. All in default value.

Higher batch sizes may not always result in high accuracy. Learning rate and optimizer utilised will also have an impact. Simply reducing the learning rate and batch size will help the network to train more effectively, especially when fine-tuning [22]

The number of times the algorithm views the full data set is equal to the number of epochs. One epoch is completed when the algorithm has seen all samples in the dataset. One iteration was completed each time a batch of data was completed through the neural network.

Since the model consists of three sets of PPES that involves six classes alternately (i.e., helmet_detected and no_helmet), the iteration or max_batches are 12,000 in one epoch which create twelve consecutive weights per 1000 iteration.

3.2.4 Training Model

The YOLOv4 network is used for transfer learning. Pre-trained YOLOv4 weights, which have been learned up to 137 convolutional layers, were used instead of training a model from scratch.

After all the dataset preparation is completed, and the pre-trained has been implemented, the training was executed by command line as shown in Figure 3.6.

```
!./darknet detector train data/obj.data cfg/yolov4-  
custom.cfg yolov4.conv.137 -dont_show -map
```

Figure 3.6: Training Command Line

For best results, training should stop when the average loss is less than 0.05 if possible or at least below 0.3. The training process took around 24 hours to complete without any interruption and every 1000 iteration, the computation will create a training weight and save in training folder to test real-time scenario (webcam feed).

3.2.5 Validation of training weight

After training process has completed, the training weight will be saved in training folder in Google Drive. As the best weight will be validate using real-time detector.

3.2.5.1 Run detector in real-time frame

Running YOLOv4 on webcam stream is a more complex than images. Start off a video stream using webcam or any recording device in safety site (CCTV) as input. Then run each frame through YOLOv4 model and create an overlay image that contains bounding box of detections. Next, the overlay bounding box image will back onto the next frame webcam stream.

To validate in real-time (webcam feed), all important dependencies need to be imported beforehand as shown in Figure 3.7. These dependencies enabling all the functions used for real-time detection in Google Colab environment.

```
# import dependencies
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from google.colab.patches import cv2_imshow
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
import matplotlib.pyplot as plt
%matplotlib inline
```

Figure 3.7: Dependencies imported

```
# import darknet functions to perform object detections
from darknet import *
# load in YOLOv4 architecture network
network, class_names, class_colors =
load_network("cfg/yolov4-custom.cfg", "data/obj.data",
             "/mydrive/yolov4/training/yolov4-custom_best.weights")
width = network_width(network)
height = network_height(network)
```

Figure 3.8: Source code to import darknet function

As shown in Figure 3.8, importing darknet is a crucial part to perform object detections in this Colab environment as it will enable the YOLOv4 architecture network load file that store in the training storage.

```

# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
    """
    Params:
        js_reply: JavaScript object containing image from webcam
    Returns:
        img: OpenCV BGR image
    """
    # decode base64 image
    image_bytes = b64decode(js_reply.split(',')[1])
    # convert bytes to numpy array
    jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
    # decode numpy array into OpenCV BGR image
    img = cv2.imdecode(jpg_as_np, flags=1)

    return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string to be
# overlaid on video stream
def bbox_to_bytes(bbox_array):
    """
    Params:
        bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
    Returns:
        bytes: Base64 image byte string
    """
    # convert array into PIL image
    bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
    iobuf = io.BytesIO()
    # format bbox into png for return
    bbox_PIL.save(iobuf, format='png')
    # format return string
    bbox_bytes = 'data:image/png;base64,{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8')))

    return bbox_bytes

```

Figure 3.9: Source of Javascript conversion to OpenCV

OpenCV has an easy integration when working with YOLO. However, OpenCV alone cannot directly enable the webcam feed to work as an input. Therefore, JavaScript is used beforehand to convert the direct input feed of the webcam to OpenCV images as shown in Figure 3.9.

```
# JavaScript to properly create live video stream using webcam as input
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
      stream.getVideoTracks()[0].stop();
      video.remove();
      div.remove();
      video = null;
      div = null;
      stream = null;
      imgElement = null;
      captureCanvas = null;
      labelElement = null;
    }
  ''')
```

Figure 3.10: JavaScript code to enable webcam as input

The video stream is properly defined as shown in Figure 3.10 to create real-time detection using webcam as an input.

```

def video_frame(label, bbox):
    data = eval_js('stream_frame("{}","{}".format(label, bbox))')
    return data

# start streaming from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialize bounding box to empty
bbox = ''
count = 0
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    frame = js_to_image(js_reply["img"])

    # create transparent overlay for bounding box
    bbox_array = np.zeros([480,640,4], dtype=np.uint8)

    # call darknet helper on video frame
    detections, width_ratio, height_ratio = darknet_helper(frame, width, height)

    # loop through detections and draw them on transparent overlay image
    for label, confidence, bbox in detections:
        left, top, right, bottom = bbox2points(bbox)
        left, top, right, bottom = int(left * width_ratio), int(top * height_ratio),
        int(right * width_ratio), int(bottom * height_ratio)
        bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_colors[label], 2)
        bbox_array = cv2.putText(bbox_array, "{} {:.2f}".format(label, float(confidence)),
            (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
            class_colors[label], 2)

```

Figure 3.11: Source code to start webcam stream and create bounding box detection

Finally, the video stream will be executed by using the code shown in Figure 3.11 along with the bounding box detection by using OpenCV images that has been converted before.

```

font = cv2.FONT_HERSHEY_SIMPLEX
color_1 = (255,0,0) #RED
color_2 = (0,255,0) #GREEN

if ('mask_detected' and 'helmet_detected' and 'vest_detected') in label:
    text = cv2.putText(bbox_array, "ALLOW", (50,440), font, 1, color_2, 2)
else:
    if ('no_mask' or 'no_helmet' or 'no_vest') in label:
        text = cv2.putText(bbox_array, "NO ENTRY", (50,440), font, 1, color_1, 2)

```

Figure 3.12: Source code for decision making using OpenCV

Adding python source code in real-time detection cell code as shown in Figure 3.12 to allow decision making display using OpenCV.

3.3 Gantt Chart

PSM2 Tasks	Duration(W) Academic Week	Mar-21			Apr-21				May-21				Jun-21					
		1/3	8/3	15/3	22/3	29/3	5/4	12/4	19/4	26/4	3/5	10/5	17/5	24/5	31/5	7/6	14/6	21/6
1.0 Project Development (Objective 2)	7																	
1.1 Collect and Label Images For Multiple PPE																		
1.2 Training Models (YOLOv4 Object Detection Algorithm)																		
1.3 Detect Multiple Objects In Real-Time (Safety Helmet, Safety Mask)																		
2.0 Project Development (Objective 3) - Analysis	7																	
2.1 Testing model in different test case																		
2.2 Analyzing Mean Average Precision (mAP) and Average Recall for Object Detection models																		
2.3 Accuracy improvement based on mAP and AR																		
3.0 PSM2 Thesis Writing	12																	
3.1 PSM1 Report Correction + Additional Info	7																	
3.2 Thesis First Draft Submission	1																	
3.3 Thesis Correction + Completion	6																	
3.4 Thesis Second Draft Submission	1																	
3.5 Thesis Final Correction + Completion	4																	
3.6 Thesis Submission to SV	1																	
3.7 Thesis Submission to 2nd Evaluator	1																	
4.0 PSM2 Seminar (EXSELEN)	3																	
4.3 Presentation	1																	

Figure 3.13: Gantt Chart for Projek Sarjana Muda 2

To ensure the project planning can be completed within the time frame. The process is translated into Gantt Chart as shown in Figure 3.13. For Projek Sarjana Muda 2, the main objective is to create a real-time detection of multiple PPE by collecting and labelling a huge amount of image datasets for safety mask, safety vest and safety helmet. After dataset preparation is complete, the model will be trained using YOLOv4 and Darknet framework in Colab environment.

The effectiveness of the algorithm will be assessed in project development (analysis) in objective 3 by testing the model in different test case, evaluating the Mean Average Precision (mAP), Average Loss and Average Recall for the model. The main accuracy improvement will be identified upon the evaluation.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter will fully discuss about the result of training computation which consists of Mean Average Precision (mAP) and Average Loss for each training weight. The best mAP for training will be used to validate the detector. Also, will be evaluating the model's performances by precision, recall and F1-Score value. Finally at the end of this chapter, the system will demonstrate the detector in real-time (webcam feed).

4.2 Evaluation Measures

Precision, Recall, and F1-score are the measures we use to assess the model's correctness. When PPE is accurately detected with IOU (Intersection Over Union) between ground truth and bounding box anticipated to be greater than a threshold, True Positive results. False Positive occurs when the identity is incorrect, which means that the incorrect class may be detected, or when the IOU is less than 0.5. False Negative is caused by misidentification, which means the thing appears but is not recognised. The following are the definitions for Precision, Recall, and F1-score:

$$Precision = \frac{TP}{TP+FP}; \quad Recall = \frac{TP}{TP+FN}; \quad F1 = \frac{2*Precision*Recall}{Precision+Recall}$$

Where : TP = True Positive (detected object region fit its groundtruth)

FP = False Positive (detected object region does not fit its groundtruth)

FN = False Negative (groundtruth not detected)

```
detections_count = 10646, unique_truth_count = 5044
class_id = 0, name = mask_detected, ap = 94.71%      (TP = 437, FP = 81)
class_id = 1, name = no_mask, ap = 99.25%          (TP = 1104, FP = 39)
class_id = 2, name = no_helmet, ap = 98.16%        (TP = 877, FP = 74)
class_id = 3, name = no_vest, ap = 97.56%          (TP = 1025, FP = 108)
class_id = 4, name = helmet_detected, ap = 97.73%  (TP = 925, FP = 70)
class_id = 5, name = vest_detected, ap = 94.80%    (TP = 411, FP = 43)

for conf_thresh = 0.25, precision = 0.92, recall = 0.95, F1-score = 0.93
for conf_thresh = 0.25, TP = 4779, FP = 415, FN = 265, average IoU = 77.47 %
```

Figure 4.1: Classification report for the all dataset

The classification report shown in Figure 4.1, includes total detections count, total ground truth count, TP, FP and AP of each class. The model's accuracy was 97.04 percent, with an average precision of 0.92, a recall of 0.95, and an F1 score of 0.93. Taking this into consideration, the model predicted with a 97 percent accuracy across all datasets.

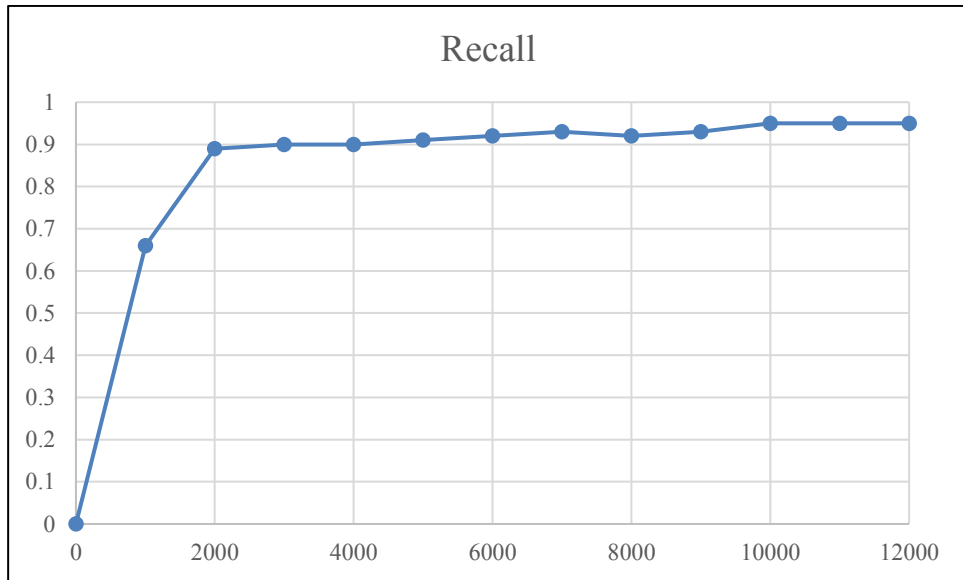


Figure 4.2: Recall Chart

The "false negative rate," or the ratio of true object detections to the total number of objects in the data set, is measured by recall. If the recall score is close to 1.0, the model will positively recognise almost all the objects in the dataset. Based on graph shown in Figure 4.2, the recall gradually increases directly proportional to the number of trainings to approach 1.0.

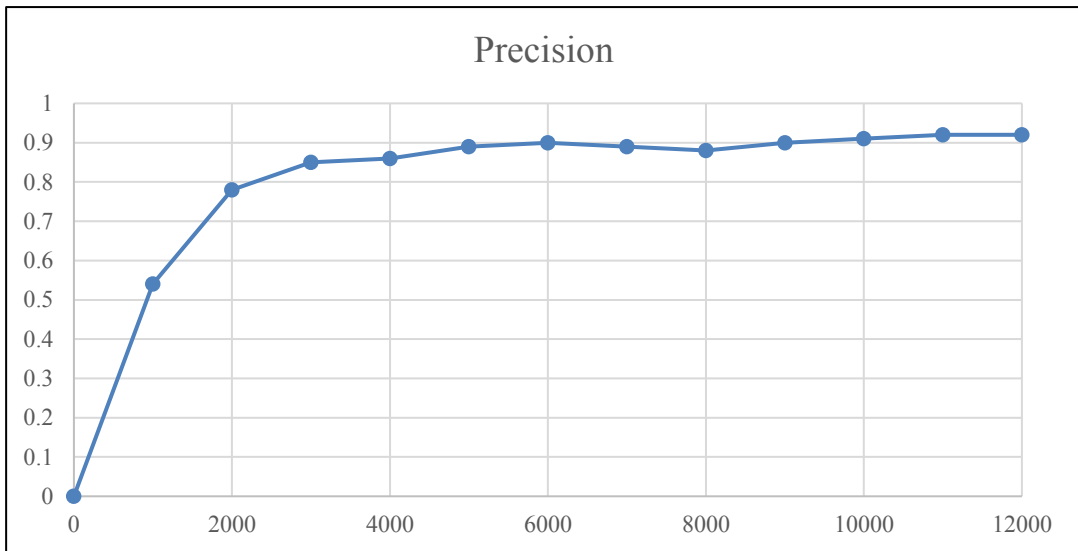


Figure 4.3: Precision Chart

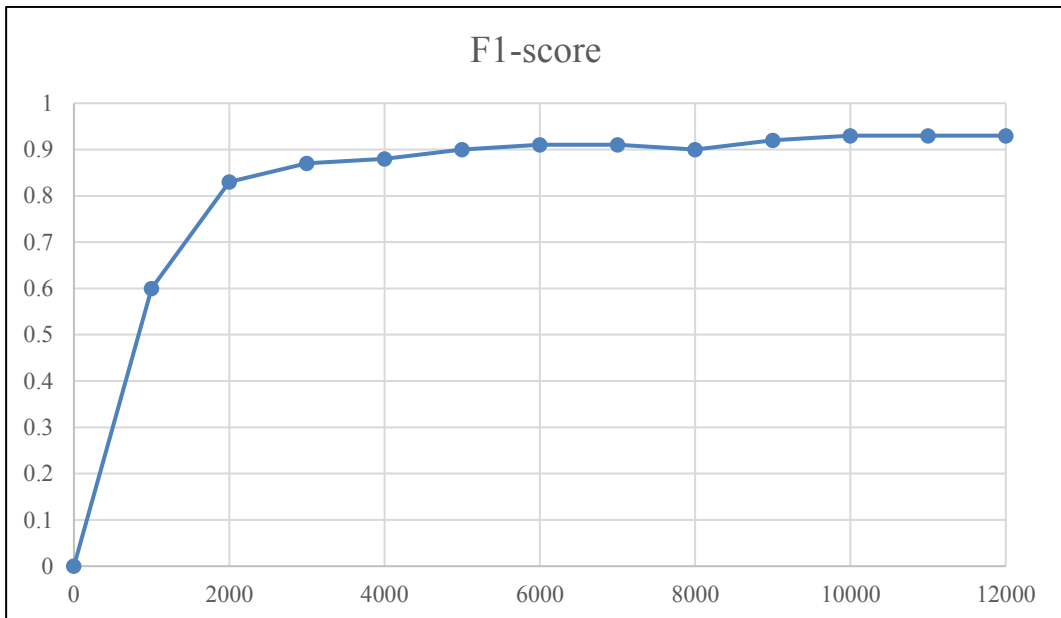


Figure 4.4: F1 Score Chart

Table 4.1: Evaluation results of the validate dataset

Iterations	Precision	Recall	F1 Score	IoU
1000	0.54	0.66	0.60	35.46%
2000	0.78	0.89	0.83	59.03%
3000	0.85	0.90	0.87	66.37%
4000	0.86	0.90	0.88	65.98%
5000	0.89	0.91	0.90	69.20%
6000	0.90	0.92	0.91	70.63%
7000	0.89	0.93	0.91	69.33%
8000	0.88	0.92	0.90	69.58%
9000	0.90	0.93	0.92	72.53%
10,000	0.91	0.95	0.93	76.35%
11,000	0.92	0.95	0.93	77.32%
12,000	0.92	0.95	0.93	77.47%

4.3 Mean Average Precision (mAP)

The mean average precision (mAP) is used to evaluate object detection models like YOLO. The mAP calculates a score by comparing the ground-truth bounding box to the detected box. The higher the score, the better the model's detection accuracy. To evaluate this model, theoretically mAP is obtained for each training weight through each training process.

To check the mAP for each training weight, the line of code shown in Figure 4.5 is used.

```
!./darknet detector map data/obj.data cfg/yolov4-
custom.cfg /mydrive/yolov4/training/yolov4-custom_best.weights -
points 0
```

Figure 4.5: mAP command line

Table 4.2: Mean Average Precision for each training iteration

Iterations	mAP
1000	56.01%
2000	88.79%
3000	92.31%
4000	90.83%
5000	94.22%
6000	94.61%
7000	95.24%
8000	94.34%
9000	95.49%
10000	96.88%
11000	97.03%
12000	97.04%

4.4 Average Loss

The dataset's training method yielded loss training, which reflects the training's performance, and was completed in 12,000 iterations for six classes. The loss value reflects the model's quality; in this case, the loss result should be the lowest score. Every training stage assesses the loss value. Weight loss at each phase is referred to as train loss. The smaller the number, the better, as the outcome is derived using the loss function, where the inaccuracy of the weight distributions is represented.

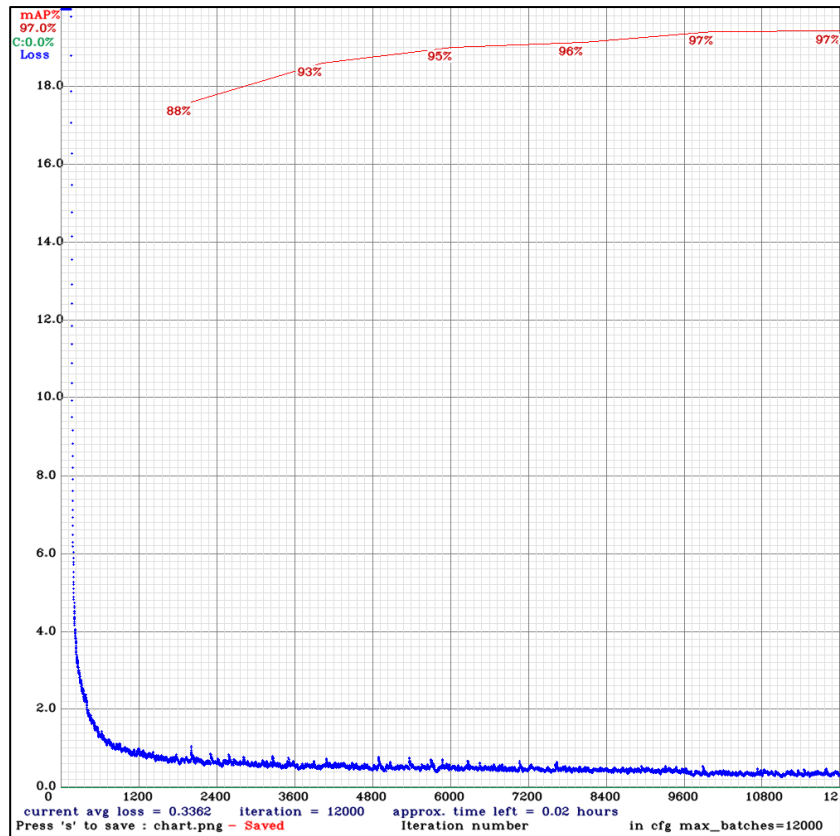


Figure 4.6: mAP vs Loss Chart

The training performance illustrated in Figure 4.6 has shown that the average loss of this algorithm is 0.3362. The ideal average loss for good detector should be in between 0.03 – 0.5. Average loss plays significant role on determining the effectiveness of the detector. As the mAP rises up through each training process according to Table 4.2, the best weight recorded at iteration 12,000. The higher the mAP the better it is for object detection.

4.5 Testing model on real-time frame (webcam feed)

Table 4.3: Detection performance for real-time

No	Classes	Decision	Detection result	Prediction speed
1	Complete PPE (Mask, Helmet, Vest)	“ALLOW TO ENTER SITE”	Successful	4.95ms
2	(No_Mask, Helmet, Vest)	“NO ENTRY”	Successful	4.83ms
3	(Mask, No_Helmet, Vest)	“NO ENTRY”	Successful	4.98ms
4	(No_Mask, No_Helmet, Vest)	“NO ENTRY”	Successful	5ms
5	(Mask, No_Helmet, No_Vest)	“NO ENTRY”	Successful	4.94ms
6	(No_Mask, Helmet, No_Vest)	“NO ENTRY”	Successful	4.95ms
7	(Mask, Helmet, No_Vest)	“NO ENTRY”	Successful	4.92ms
8	(No_Mask, No_Helmet, No_Vest)	“NO ENTRY”	Successful	4.88ms

Table 4.3 shows the performance of real-time using webcam feed to detect the worker wearing complete PPE appropriately. The result images are all attached in **Appendix B**.

4.5.1 Run the detection model in different test case

Table 4.4: Mask test case

Class	Different Test Condition	Detection class	Detection accuracy	Detection speed
Mask	Wear correctly	mask_detected	88.05%	4.94ms
	No mask	no_mask	92.73%	4.53ms
	Not wear correctly	no_mask	73.26%	4.99ms
	White cloth	N/A	N/A	N/A
	Hand covered	N/A	N/A	N/A
	Far distance detection	mask_detected	70%	4.81ms
	Holding a mask	N/A	N/A	N/A

Table 4.5: Helmet test case

Class	Different Test Condition	Detection class	Detection accuracy	Detection speed
Helmet	Wear correctly	helmet_detected	80.30%	4.90ms
	No helmet	no_helmet	88.52%	4.83ms
	Wearing cap	N/A	N/A	N/A
	White headwear	helmet_detected	75.44%	4.90ms
	Not wear correctly	helmet_detected	87.07%	4.92ms
	Holding a helmet	N/A	N/A	N/A
	Far distance detection	helmet_detected	95.66%	4.99ms

Table 4.6: Vest test case

Class	Different Test Condition	Detection class	Detection accuracy	Detection speed
Vest	Wear correctly	vest_detected	98.69%	4.92ms
	No vest	no_vest	57.45%	4.94ms
	Green Shirt	N/A	N/A	N/A
	Unzip vest	no_vest	51.46%	4.95ms
	Holding a vest	vest_detected	99.2%	4.80ms
	Far distance detection	vest_detected	97.89%	4.99ms

These test cases for mask, helmet and vest to make an evaluation of detection model in term of precision, accuracy and speed with different test case. The result images attached in **Appendix C**.

4.6 YOLOv4 comparison (Darknet Framework vs TensorFlow Framework)

The performance comparison between proposed method using YOLOv4 and Darknet Framework with YOLOv4 and TensorFlow Framework [16] is shown in Table 4.7.

Table 4.7: Performance comparison

Metric	Algorithm	
	YOLOv4 (Darknet)	YOLOv4 (TensorFlow)
Mean Average Precision (mAP)	97.04%	79%
Average Loss	0.3362	2.9671
Average Precision	0.92	0.78
Average Recall	0.95	0.80
F1-Score	0.93	0.79

Precision refers to the accuracy of a model's predictions. The effectiveness of a model to detect positive cases is measured by recall. Precision and recall are both considered in the F1-Score, which is a weighted average of the two. The F1-score provides insight into a model's false positive and false negative predictions.

Based on Table 4.7, the model using YOLOv4 and Darknet Framework has better overall performances compared with YOLOv4 and TensorFlow Framework.

CHAPTER 5

CONCLUSION

5.1 Conclusion

This study applies deep learning-based computer vision algorithms to detect essential processes that keep construction sites safe and running smoothly. However, this study act only as a study purpose. This study also shows how safety compliance may be automatically recognised by employing a trained model to analyse data from locations using YOLOv4, a state-of-the-art object detection method. The study also illustrated how taught algorithms may be customised to specific circumstances via transfer learning. Techniques like these are critical for assuring the framework's scalability and applicability.

For this project, all PPE were detected by using YOLOv4 algorithm approach individually which consist of six classes (no_mask, no_helmet, no_vest, mask, helmet and vest). The training weight is basically trained inference graph that gradually increase upon number of iterations, which will be later used to perform the object detection. The training result shows that the Mean Average Precision (mAP) for the best weight training is up to 97.04% for all PPE classes. This weight then applied to a detector for real-time application (webcam feed). To evaluate the detector effectiveness in term of precision and accuracy, the detector is put into tested whether the detector able to detect a certain PPE in different test cases. The result shows a significant improvement in precision where the detection able to differentiate properly. And based on the result of vest and helmet test case, there are some detecting error which the detector detecting the vest only without having to wear it by the person and able to detect non-helmet (white headwear). This is due to of the position, similar shape and datasets trained. Average detection speed is 4.90ms. Hence, all the objectives for PSM 2 has been achieved successfully.

5.2 Recommendation

Based on current results, the improvement that can be made is to add or change to other PPE (e.g., safety boot, safety goggle, etc) according to site requirement. Furthermore, to further improvise the detection error in helmet and vest test case, the slight improvement in labelling the dataset can be made such as include images with vest and helmet at different, scales, rotations, lightings, from different sides, different body posture, on different backgrounds. Also, can include images with non-labelled PPE that do not want to be detected which a negative sample without bounded box.

This model also can be associated with microcontroller to produce an output for hardware such as motor, siren, etc upon every detection made. First, the model needs to be imported first in individual device because Google Colab unable to link with any external hardware of microcontroller.

REFERENCES

- [1] “Table 4. Fatal occupational injuries for selected industries, 2016-20 - 2020 A01 Results.” <https://www.bls.gov/news.release/cfoi.t04.htm> (accessed Feb. 05, 2022).
- [2] “Commonly Used Statistics | Occupational Safety and Health Administration.” <https://www.osha.gov/data/commonstats> (accessed Feb. 05, 2022).
- [3] A. R. A. Hamid *et al.*, “Causes of fatal construction accidents in Malaysia,” *IOP Conference Series: Earth and Environmental Science*, vol. 220, no. 1, p. 012044, Jan. 2019, doi: 10.1088/1755-1315/220/1/012044.
- [4] “GUIDELINES ON OCCUPATIONAL SAFETY AND HEALTH ACT 1994 (ACT 514) DEPARTMENT OF OCCUPATIONAL SAFETY AND HEALTH MINISTRY OF HUMAN RESOURCES MALAYSIA,” 2006, Accessed: Feb. 05, 2022. [Online]. Available: <http://dosh.mohr.gov.my/>
- [5] N. D. Nath, A. H. Behzadan, S. Paal, and A. Professor, “DEEP LEARNING FOR SITE SAFETY: REAL-TIME DETECTION OF PERSONAL PROTECTIVE EQUIPMENT DEEP LEARNING FOR SITE SAFETY: REAL-TIME DETECTION OF PERSONAL 1 PROTECTIVE EQUIPMENT 2,” 2020.
- [6] X. Yang, Y. Yu, S. Shirowzhan, S. Sepasgozer, and H. Li, “Automated PPE-Tool pair check system for construction safety using smart IoT,” *Journal of Building Engineering*, vol. 32, Nov. 2020, doi: 10.1016/j.jobe.2020.101721.
- [7] M. K. N. Mahmad, M. A. Z. Mohd Remy Rozainy, and N. Baharun, “Applications of Radio Frequency Identification (RFID) in Mining Industries,” in *IOP Conference Series: Materials Science and Engineering*, Jun. 2016, vol. 133, no. 1. doi: 10.1088/1757-899X/133/1/012050.
- [8] S. Zhang, J. Teizer, N. Pradhananga, and C. M. Eastman, “Workforce location tracking to model, visualize and analyze workspace requirements in building information models for construction safety planning,” *Automation in Construction*, vol. 60, pp. 74–86, Dec. 2015, doi: 10.1016/J.AUTCON.2015.09.009.

- [9] “Automatically detecting personal protective equipment on persons in images using Amazon Rekognition | AWS Machine Learning Blog.” <https://aws.amazon.com/blogs/machine-learning/automatically-detecting-personal-protective-equipment-on-persons-in-images-using-amazon-rekognition/> (accessed Feb. 05, 2022).
- [10] M. Mohd Saudi *et al.*, “Image Detection Model for Construction Worker Safety Conditions using Faster R-CNN,” *IJACSA International Journal of Advanced Computer Science and Applications*, vol. 11, no. 6, 2020, Accessed: Feb. 04, 2022. [Online]. Available: www.ijacsa.thesai.org
- [11] Z. Kolar, H. Chen, and X. Luo, “Transfer learning and deep convolutional neural networks for safety guardrail detection in 2D images,” *Automation in Construction*, vol. 89, pp. 58–70, May 2018, doi: 10.1016/J.AUTCON.2018.01.003.
- [12] F. Zhafran, E. S. Ningrum, M. N. Tamara, and E. Kusumawati, “Computer Vision System Based for Personal Protective Equipment Detection, by Using Convolutional Neural Network,” *IES 2019 - International Electronics Symposium: The Role of Techno-Intelligence in Creating an Open Energy System Towards Energy Democracy, Proceedings*, pp. 516–521, Sep. 2019, doi: 10.1109/ELECSYM.2019.8901664.
- [13] “Face Mask Detection — using Multi Stage CNN Architecture | by Anand Saran | Medium.” <https://anandsarank.medium.com/multi-stage-cnn-architecture-for-face-mask-detection-63cae21724a9> (accessed Feb. 05, 2022).
- [14] V. S. K. Delhi, R. Sankarlal, and A. Thomas, “Detection of Personal Protective Equipment (PPE) Compliance on Construction Site Using Computer Vision Based Deep Learning Techniques,” *Frontiers in Built Environment*, vol. 6, p. 136, Sep. 2020, doi: 10.3389/FBUIL.2020.00136/BIBTEX.
- [15] T. Q. Vinh and N. T. N. Anh, “Real-Time Face Mask Detector Using YOLOv3 Algorithm and Haar Cascade Classifier,” in *Proceedings - 2020 International Conference on Advanced Computing and Applications, ACOMP 2020*, Nov. 2020, pp. 146–149. doi: 10.1109/ACOMP50827.2020.00029.

- [16] A. A. Protik, A. H. Rafi, and S. Siddique, “Real-time Personal Protective Equipment (PPE) Detection Using YOLOv4 and TensorFlow,” Aug. 2021. doi: 10.1109/TENSYMP52854.2021.9550808.
- [17] “What’s the difference between Machine Learning and Deep Learning? - viso.ai.” <https://viso.ai/deep-learning/deep-learning-vs-machine-learning/> (accessed Jun. 16, 2022).
- [18] “YOLO: Real-Time Object Detection Explained.” <https://www.v7labs.com/blog/yolo-object-detection> (accessed Jun. 16, 2022).
- [19] “YOLO v4: Optimal Speed & Accuracy for object detection | by Andrej Anka | Towards Data Science.” <https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50> (accessed Feb. 04, 2022).
- [20] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, Accessed: Feb. 04, 2022. [Online]. Available: <https://arxiv.org/abs/2004.10934v1>
- [21] “What is the difference between training and test dataset? | by Sajid Lhessani | Analytics Vidhya | Medium.” <https://medium.com/analytics-vidhya/what-is-the-difference-between-training-and-test-dataset-d20820e5f632> (accessed Jun. 16, 2022).
- [22] I. Kandel and M. Castelli, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset,” *ICT Express*, vol. 6, no. 4, pp. 312–315, Dec. 2020, doi: 10.1016/J.ICTE.2020.04.010.

APPENDIX A

SAMPLE APPENDIX 1

```
# import dependencies
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from google.colab.patches import cv2_imshow
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
import matplotlib.pyplot as plt
%matplotlib inline

# import darknet functions to perform object detections
from darknet import *
# load in our YOLOv4 architecture network
network, class_names, class_colors = load_network("cfg/yolov4-custom.cfg", "data/obj.data", "/mydrive/yolov4/training/yolov4-custom_best.weights")
width = network_width(network)
height = network_height(network)

# darknet helper function to run detection on image
def darknet_helper(img, width, height):
    darknet_image = make_image(width, height, 3)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (width, height),
                              interpolation=cv2.INTER_LINEAR)

    # get image ratios to convert bounding boxes to proper size
    img_height, img_width, _ = img.shape
    width_ratio = img_width/width
    height_ratio = img_height/height
```

```
# run model on darknet style image to get detections
copy_image_from_bytes(darknet_image, img_resized.tobytes())
detections = detect_image(network, class_names, darknet_image)
free_image(darknet_image)
return detections, width_ratio, height_ratio

# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
    """
    Params:
        js_reply: JavaScript object containing image from webcam
    Returns:
        img: OpenCV BGR image
    """
    # decode base64 image
    image_bytes = b64decode(js_reply.split(',')[1])
    # convert bytes to numpy array
    jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
    # decode numpy array into OpenCV BGR image
    img = cv2.imdecode(jpg_as_np, flags=1)

    return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string to be overlaid on video stream
def bbox_to_bytes(bbox_array):
    """
    Params:
        bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
    Returns:
        bytes: Base64 image byte string
    """
```

```

# convert array into PIL image
bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
iobuf = io.BytesIO()
# format bbox into png for return
bbox_PIL.save(iobuf, format='png')
# format return string
bbox_bytes = 'data:image/png;base64,{}'.format((str(b64encode(iobuf.getvalue())), 'utf-8'))

return bbox_bytes

# JavaScript to properly create our live video stream using our webcam as input
def video_stream():
    js = Javascript('''
var video;
var div = null;
var stream;
var captureCanvas;
var imgElement;
var labelElement;

var pendingResolve = null;
var shutdown = false;

function removeDom() {
    stream.getVideoTracks()[0].stop();
    video.remove();
    div.remove();
    video = null;
    div = null;
    stream = null;
    imgElement = null;
    captureCanvas = null;
    labelElement = null;
}

```

```

function onAnimationFrame() {
    if (!shutdown) {
        window.requestAnimationFrame(onAnimationFrame);
    }
    if (pendingResolve) {
        var result = "";
        if (!shutdown) {
            captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);
            result = captureCanvas.toDataURL('image/jpeg', 0.8)
        }
        var lp = pendingResolve;
        pendingResolve = null;
        lp(result);
    }
}

async function createDom() {
    if (div !== null) {
        return stream;
    }

    div = document.createElement('div');
    div.style.border = '2px solid black';
    div.style.padding = '3px';
    div.style.width = '100%';
    div.style.maxWidth = '600px';
    document.body.appendChild(div);

```

```

const modelOut = document.createElement('div');
modelOut.innerHTML = "<span>Status:</span>";
labelElement = document.createElement('span');
labelElement.innerText = 'No data';
labelElement.style.fontWeight = 'bold';
modelOut.appendChild(labelElement);
div.appendChild(modelOut);

video = document.createElement('video');
video.style.display = 'block';
video.width = div.clientWidth - 6;
video.setAttribute('playsinline', '');
video.onclick = () => { shutdown = true; };
stream = await navigator.mediaDevices.getUserMedia(
  {video: { facingMode: "environment"}});
div.appendChild(video);

imgElement = document.createElement('img');
imgElement.style.position = 'absolute';
imgElement.style.zIndex = 1;
imgElement.onclick = () => { shutdown = true; };
div.appendChild(imgElement);

const instruction = document.createElement('div');
instruction.innerHTML =
  '<span style="color: red; font-weight: bold;">' +
  'When finished, click here or on the video to stop this demo</span>';
div.appendChild(instruction);
instruction.onclick = () => { shutdown = true; };

video.srcObject = stream;
await video.play();

```

```

captureCanvas = document.createElement('canvas');
captureCanvas.width = 640; //video.videoWidth;
captureCanvas.height = 480; //video.videoHeight;
window.requestAnimationFrame(onAnimationFrame);

return stream;
}
async function stream_frame(label, imgData) {
  if (shutdown) {
    removeDom();
    shutdown = false;
    return '';
  }

  var preCreate = Date.now();
  stream = await createDom();

  var preShow = Date.now();
  if (label != "") {
    labelElement.innerHTML = label;
  }

  if (imgData != "") {
    var videoRect = video.getClientRects()[0];
    imgElement.style.top = videoRect.top + "px";
    imgElement.style.left = videoRect.left + "px";
    imgElement.style.width = videoRect.width + "px";
    imgElement.style.height = videoRect.height + "px";
    imgElement.src = imgData;
  }
}

```



```

var preCapture = Date.now();
var result = await new Promise(function(resolve, reject) {
  pendingResolve = resolve;
});
shutdown = false;

return {'create': preShow - preCreate,
        'show': preCapture - preShow,
        'capture': Date.now() - preCapture,
        'img': result};
}
''')

display(js)

def video_frame(label, bbox):
    data = eval_js('stream_frame("{}","{}").format(label, bbox)')
    return data

# start streaming video from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialize bounding box to empty
bbox = ''
count = 0
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    frame = js_to_image(js_reply["img"])

# call darknet helper on video frame
detections, width_ratio, height_ratio = darknet_helper(frame, width, height)

# loop through detections and draw them on transparent overlay image
for label, confidence, bbox in detections:
    left, top, right, bottom = bbox2points(bbox)
    left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int(right * width_ratio), int(bottom * height_ratio)
    bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_colors[label], 2)
    bbox_array = cv2.putText(bbox_array, "{} [ {:.2f} ]".format(label, float(confidence)),
                            (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, class_colors[label], 2)

    font = cv2.FONT_HERSHEY_SIMPLEX
    color_1 = (255,0,0) #RED
    color_2 = (0,255,0) #GREEN

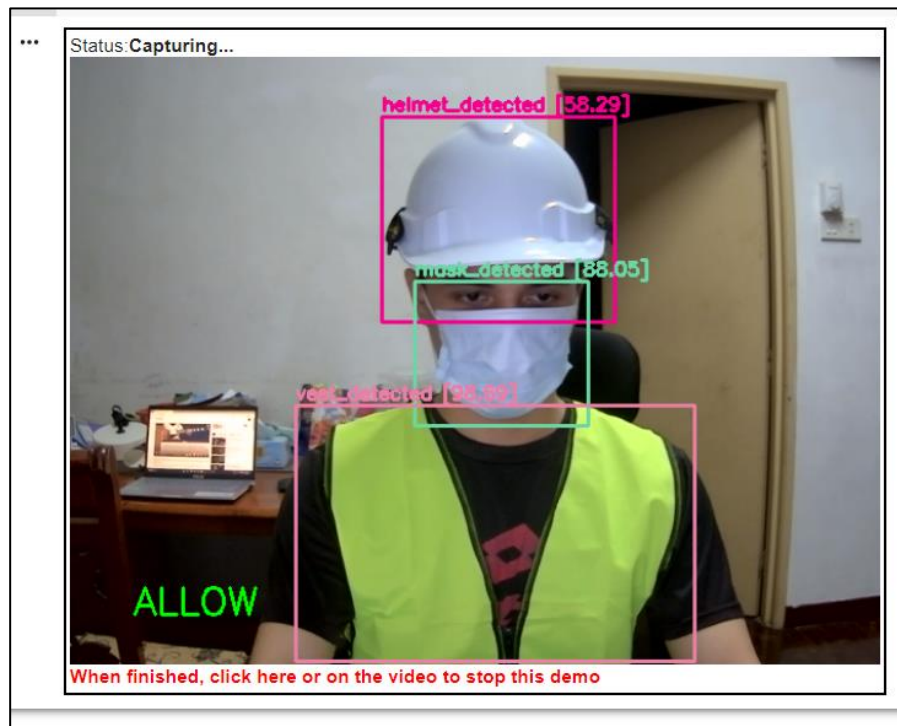
    if "no_mask" or "no_helmet" or "no_vest" in label:
        bbox_array = cv2.putText(bbox_array, "NO ENTRY", (50,440), font, 1, color_1, 2)

    if ('mask_detected' and 'helmet_detected' and 'vest_detected') in label:
        bbox_array = cv2.putText(bbox_array, "ALLOW TO ENTER", (50,440), font, 1, color_2, 2)

bbox_array[:, :, 3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
# convert overlay of bbox into bytes
bbox_bytes = bbox_to_bytes(bbox_array)
# update bbox so next frame gets new overlay
bbox = bbox_bytes

```

APPENDIX B

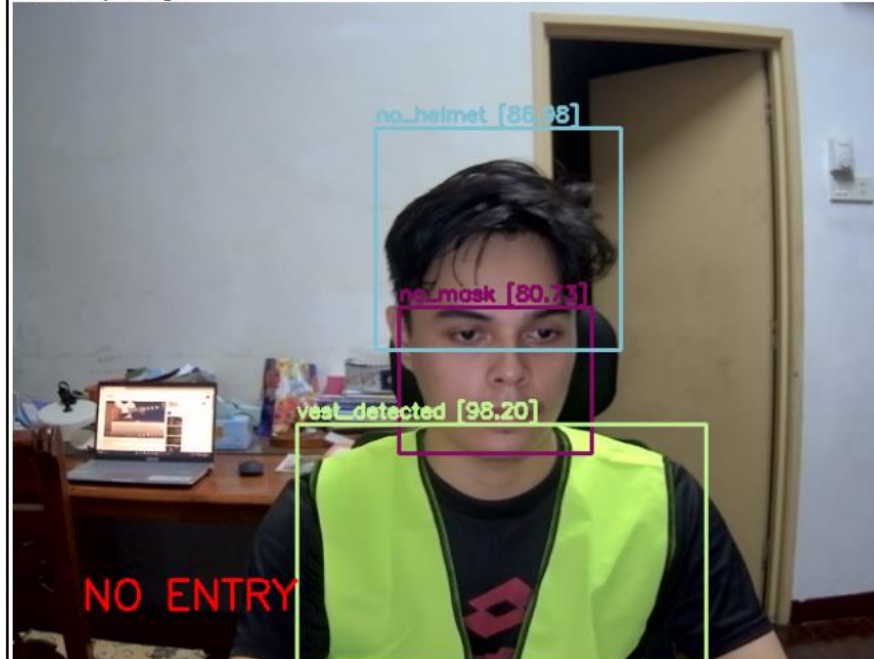


Status: Capturing...



When finished, click here or on the video to stop this demo

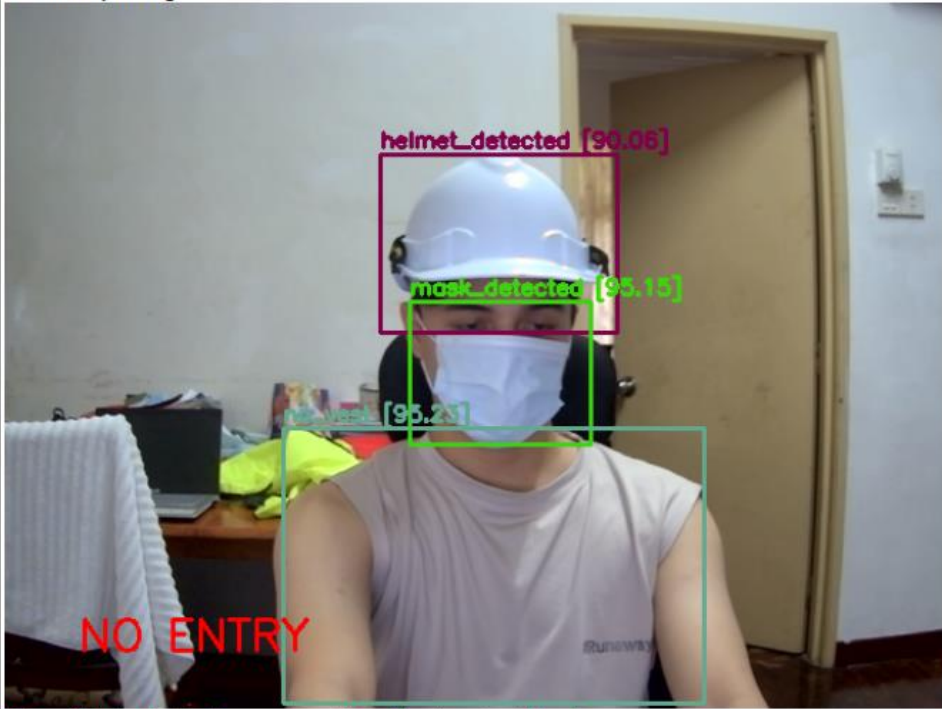
Status: Capturing...



When finished, click here or on the video to stop this demo

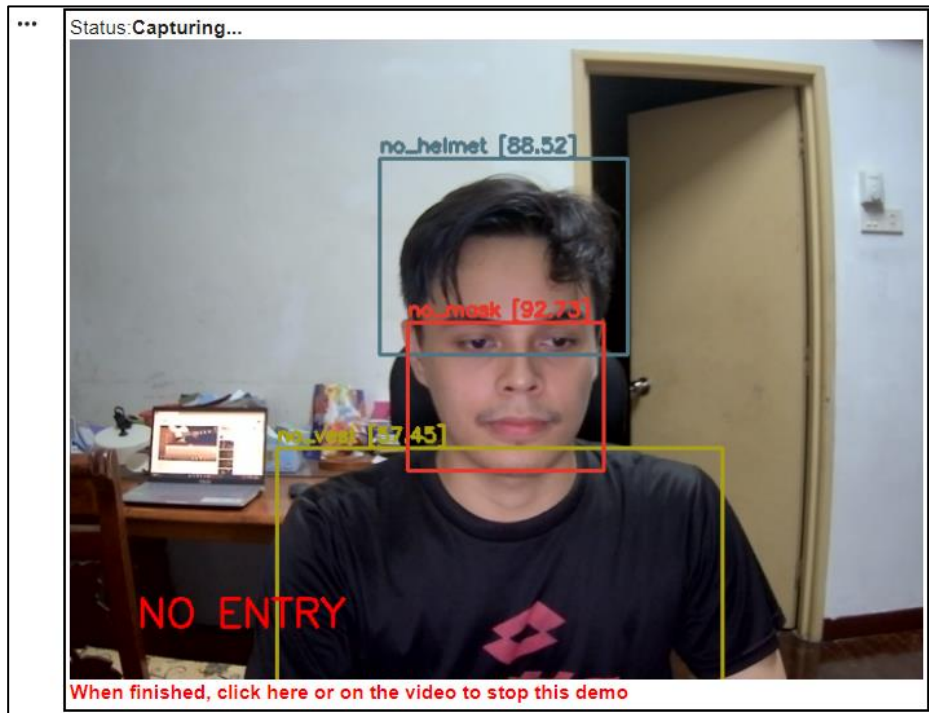


Status: Capturing...

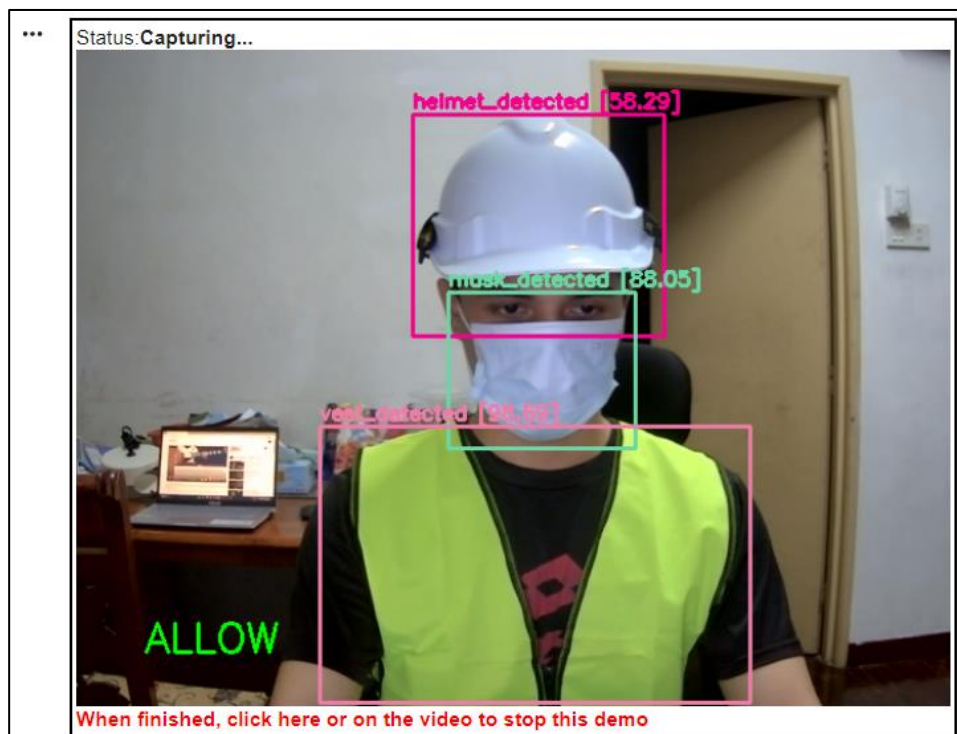


When finished, click here or on the video to stop this demo

APPENDIX C

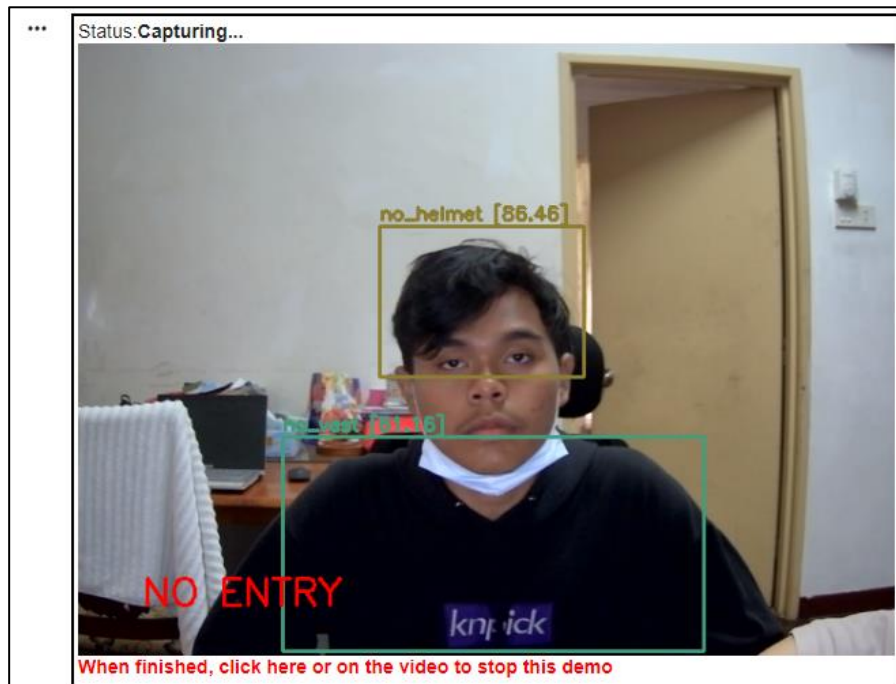


Test case 1: Not wearing all PPE



Test case 2: Wearing all PPE correctly

Mask test case



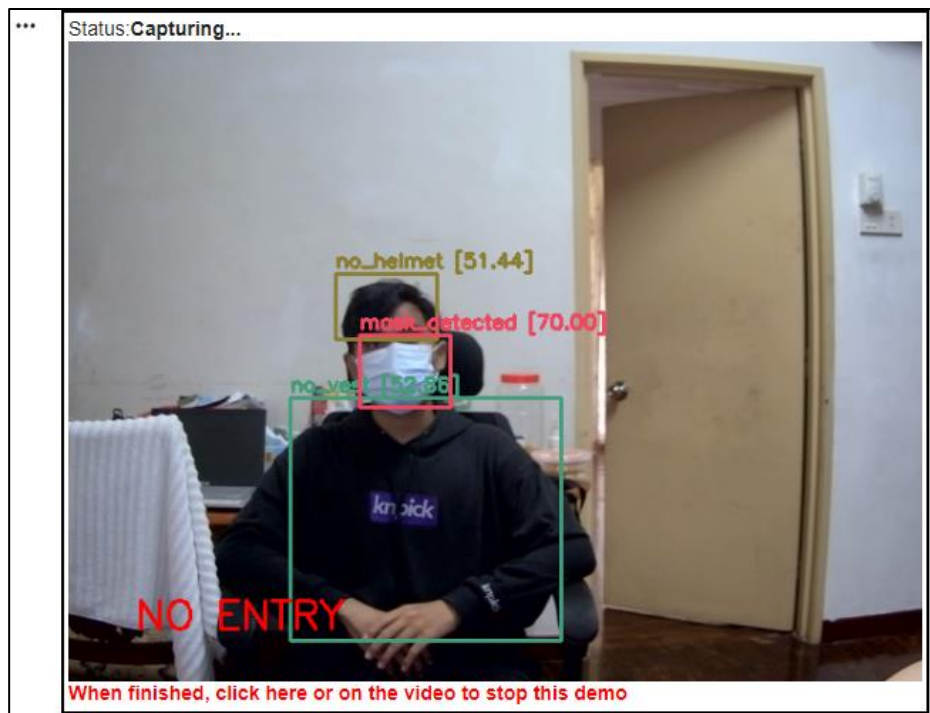
Test case 3: Not wearing mask properly



Test case 4: Using cloth



Test case 5: Hand cover



Test case 6: Far distance detection



Test case 7: Holding a mask

Helmet test case



Test case 8: Wear cap/other headwear



Test case 9: Not wearing helmet properly



Test case 10: Holding a helmet



Test case 11: Far distance detection

Vest test case



Test case 12: Unzipped vest



Test case 13: Far distance detection



Test case 14: Holding a vest

APPENDIX D

```
import argparse
from tqdm import tqdm
#convert from Yolo_mark to opencv format
def yoloFormattocv(x1, y1, x2, y2, H, W):
    bbox_width = x2 * W
    bbox_height = y2 * H
    center_x = x1 * W
    center_y = y1 * H
    voc = []
    voc.append(center_x - (bbox_width / 2))
    voc.append(center_y - (bbox_height / 2))
    voc.append(center_x + (bbox_width / 2))
    voc.append(center_y + (bbox_height / 2))
    return [int(v) for v in voc]

# Convert from opencv format to yolo format
# H,W is the image height and width
def cvFormattoYolo(corner, H, W):
    bbox_W = corner[3] - corner[1]
    bbox_H = corner[4] - corner[2]
    center_bbox_x = (corner[1] + corner[3]) / 2
    center_bbox_y = (corner[2] + corner[4]) / 2
    return corner[0], round(center_bbox_x / W, 6),
           round(center_bbox_y / H, 6),
           round(bbox_W / W, 6),
           round(bbox_H / H, 6)

class yoloRotatebbox:
def __init__(self, filename, image_ext, angle):
    assert os.path.isfile(filename + image_ext)
    assert os.path.isfile(filename + '.txt')

    self.filename = filename
    self.image_ext = image_ext
    self.angle = angle

# Read image using cv2
self.image = cv2.imread(self.filename + self.image_ext, 1)
```

```
else:
    continue
for angle in angels:
    im = yoloRotatebbox(image_name, image_ext, angle)
    bbox = im.rotateYolobbox()
    image = im.rotate_image()
    # to write rotateed image to disk
    cv2.imwrite(image_name+'_' + str(angle) + '.jpg', image)
    file_name = image_name+'_' + str(angle) + '.txt'
    #print("For angle "+str(angle))
    if os.path.exists(file_name):
        os.remove(file_name)
    # to write the new rotated bboxes to file
    for i in bbox:
        with open(file_name, 'a') as fout:
            fout.writelines(
                ' '.join(map(str, cvFormattoYolo(i, im.rotate_image().shape[0], im.rotate_image().shape[1]))) + '\n')
```

```

# Read image using cv2
self.image = cv2.imread(self.filename + self.image_ext, 1)

rotation_angle = self.angle * np.pi / 180
self.rot_matrix = np.array(
    [[np.cos(rotation_angle), -np.sin(rotation_angle)],
     [np.sin(rotation_angle), np.cos(rotation_angle)]]
)
def rotateYolobbox(self):
    new_height, new_width = self.rotate_image().shape[:2]
    f = open(self.filename + '.txt', 'r')
    f1 = f.readlines()
    new_bbox = []
    H, W = self.image.shape[:2]
    for x in f1:
        bbox = x.strip('\n').split(' ')
        if len(bbox) > 1:
            (center_x, center_y, bbox_width, bbox_height) =
            yoloFormattoCv(float(bbox[1]), float(bbox[2]), float(bbox[3]), float(bbox[4]), H, W)
            upper_left_corner_shift = (center_x - W / 2, -H / 2 + center_y)
            upper_right_corner_shift = (bbox_width - W / 2, -H / 2 + center_y)
            lower_left_corner_shift = (center_x - W / 2, -H / 2 + bbox_height)
            lower_right_corner_shift = (bbox_width - W / 2, -H / 2 + bbox_height)
            new_lower_right_corner = [-1, -1]
            new_upper_left_corner = []
            for i in (upper_left_corner_shift, upper_right_corner_shift, lower_left_corner_shift,
                    lower_right_corner_shift):
                new_coords = np.matmul(self.rot_matrix, np.array((i[0], -i[1])))
                x_prime, y_prime = new_width / 2 + new_coords[0], new_height / 2 - new_coords[1]
                if new_lower_right_corner[0] < x_prime:
                    new_lower_right_corner[0] = x_prime
                if new_lower_right_corner[1] < y_prime:
                    new_lower_right_corner[1] = y_prime
                if len(new_upper_left_corner) > 0:
                    if new_upper_left_corner[0] > x_prime:
                        new_upper_left_corner[0] = x_prime
                    if new_upper_left_corner[1] > y_prime:
                        new_upper_left_corner[1] = y_prime
            else:
                new_upper_left_corner.append(x_prime)
                new_upper_left_corner.append(y_prime)
                # print(x_prime, y_prime)
                new_bbox.append([bbox[0], new_upper_left_corner[0], new_upper_left_corner[1],
                                new_lower_right_corner[0], new_lower_right_corner[1]])
    return new_bbox
def rotate_image(self):
    """
    Rotates an image (angle in degrees) and expands image to avoid cropping
    """
    height, width = self.image.shape[:2] # image shape has 3 dimensions
    image_center = (width / 2,
                   height / 2)
    rotation_mat = cv2.getRotationMatrix2D(image_center, self.angle, 1.)
    # rotation calculates the cos and sin, taking absolutes of those.
    abs_cos = abs(rotation_mat[0, 0])
    abs_sin = abs(rotation_mat[0, 1])
    # find the new width and height bounds
    bound_w = int(height * abs_sin + width * abs_cos)
    bound_h = int(height * abs_cos + width * abs_sin)

    rotation_mat[0, 2] += bound_w / 2 - image_center[0]
    rotation_mat[1, 2] += bound_h / 2 - image_center[1]
    # rotate image with the new bounds and translated rotation matrix
    rotated_mat = cv2.warpAffine(self.image, rotation_mat, (bound_w, bound_h))
    return rotated_mat
if __name__ == "__main__":
    angels=[45,90,135,180,225,270,315]
    for filename in tqdm(os.listdir()):
        file =filename.split(".")
        if(file[-1]=="jpg"):
            image_name=file[0]
            image_ext="."+file[1]
        else:

```