



**School of Computer Sciences**

## **CDS590 – Consultancy Project & Practicum**

### **Final Report**

## **Smart Time Study Using Artificial Intelligence**

CHAN HUAN YANG

P-COM0068/19

Supervisor: Madame Maziani Binti Sabudin

Lecturer(s): Dr Nasuha Lee Abdullah

SEM 2 2020/2021

## DECLARATION

"I declare that the following is my own work and does not contain any *unacknowledged* work from any other sources. This project was undertaken to fulfill the requirements of the Consultancy Project & Practicum for the Master of Science (Data Science and Analytics) program at Universiti Sains Malaysia".

Signature	:	 _____
Name	:	_____ Chan Huan Yang
Date	:	_____ 1 - July - 2021

## **ACKNOWLEDGEMENTS**

First and foremost, I am highly grateful to my supervisors, Madame Maziani Binti Sabudin, for her invaluable advice, continuous support, and patience in my entire practicum project. Her immense knowledge and ample experience have encouraged me in my academic research and daily life. My gratitude extends to my course lecturer – Dr Nasuha Lee Abdullah, for her invaluable supervision, support and teaching during the entire course of CDS590. I would also like to thank Mr Ng Wooi Beng for his guidance and mentorship at my practicum place. My appreciation also goes out to the Lumileds Sdn Bhd Malaysia for the opportunity to conduct my practicum project at the Department of Industrial Engineering. Finally, I would like to express my gratitude to my parents and my wife. Without their tremendous understanding and encouragement in the past few months, especially in this time of the pandemic, it would be impossible for me to complete this practicum project.

## **ABSTRACT**

Artificial intelligence (AI) can create, transforming and improving many facets of engineering activities. In this project, an AI-powered time study prototype is proposed to reduce the attended time required by an IE to complete the time study analysis by automating the video analysis process. This is how our prototype works. At the first step, we identify our target events, such as process start point and endpoint, from images or video that we have recorded. The target events selected are appropriately labelled and given a unique class name. Then we use the training and testing images to train our object detection model, TensorFlow 2 EfficientDet model. The training process is on Google Colab using the GPU accelerator. The training process is stopped when the lost function no longer can be reduced. The trained model is then exported and used for inference. Whenever the model detects the target events from a video, it will draw out the inference boxes and output those events' time points and confidence scores into a CSV file. The compilation of all the target events time points is used to help IE calculate the cycle time of every sample. The proposed prototype is able to reduce the time required by an IE to complete the time study analysis by automating the event detection process from the video. The results generated using the prototype also meet the acceptable error range. Hence, it should become a reliable tool to improve IE overall productivity.

## **ABSTRAK**

Kecerdasan buatan (AI) mampu mencipta, mengubah dan memperbaiki banyak aspek aktiviti kejuruteraan. Dalam projek ini, prototaip kajian masa berlandaskan AI dicadangkan untuk mengurangkan waktu yang dihadiri oleh IE untuk menyelesaikan analisis kajian waktu dengan mengotomatisasi proses analisis video. Inilah cara prototaip kami berfungsi. Pada langkah pertama, kami mengenal pasti peristiwa sasaran kami, seperti titik permulaan dan titik akhir proses, dari gambar atau video yang telah kami rakam. Peristiwa sasaran yang dipilih dilabel dengan tepat dan diberi nama kelas yang unik. Kemudian kami menggunakan gambar latihan dan penujian untuk melatih model pengesanan objek kami iaitu model TensorFlow 2 EfficientDet. Proses latihan dilakukan di Google Colab yang menggunakan pemecut GPU. Proses latihan dihentikan apabila fungsi kehilangan tidak dapat dikurangkan lagi. Model yang dilatih kemudian dieksport dan digunakan untuk membuat kesimpulan. Setiap kali model mengesan peristiwa sasaran dari video, ia akan melakar kotak inferensi dan mengeluarkan titik waktu peristiwa dan skor keyakinan ke dalam fail CSV. Penyusunan semua titik masa peristiwa sasaran digunakan untuk membantu IE mengira masa kitaran setiap sampel. Prototaip yang dicadangkan dapat mengurangkan masa yang diperlukan oleh IE untuk menyelesaikan analisis kajian waktu dengan mengotomatisasi proses pengesanan peristiwa dari video. Hasil yang dikeluarkan menggunakan prototaip juga memenuhi julat ralat yang dapat diterima. Oleh itu, ia harus menjadi alat yang boleh dipercayai untuk meningkatkan produktiviti IE secara keseluruhan.

## TABLE OF CONTENTS

DECLARATION .....	ii
ACKNOWLEDGEMENTS .....	iii
ABSTRACT.....	iv
ABSTRAK.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES .....	vii
LIST OF FIGURES .....	viii
LIST OF ABBREVIATIONS AND SYMBOLS .....	x
1. INTRODUCTION... ..	1
2. RELATED WORKS .....	7
3. RESEARCH METHODOLOGY.....	13
4. RESULTS AND DISCUSSION .....	26
5. CONCLUSION & LESSON LEARNED.....	33
6. REFERENCES .....	35
APPENDICES .....	37

## **LIST OF TABLES**

Table 4.1: User acceptance testing of the project.....	26
Table 4.2: Total time required of the time study (before vs after using the prototype).....	29
Table 4.3: The error rate of the prototype.....	31

## LIST OF FIGURES

Figure 1.1: Lumileds worldwide offices and plants location.....	1
Figure 1.2: Lumileds Malaysia Sdn Bhd in Penang.....	2
Figure 1.3: Lumileds product images.....	3
Figure 1.4: Lumileds time study sample.....	5
Figure 2.1: Overview of object recognition computer vision tasks.....	7
Figure 2.2: Performance of Object Detectors on MS COCO dataset (Zaidi et al., 2021).....	11
Figure 3.1: Data science methodology.....	13
Figure 3.2: Proposed solution.....	14
Figure 3.3: The current (as of Jan 21) top 5 classifiers on the ImageNet task.....	15
Figure 3.4: Time study candidates.....	17
Figure 3.5: Different part of WBB.....	18
Figure 3.6: Display panel of WBB and the event description.....	19
Figure 3.7: Sample calculation of time study of WBB.....	20
Figure 3.8: The Labelimg used to annotate the images.....	22
Figure 3.9: Summary of Image Annotation.....	23
Figure 3.10. TensorBoard output to visualize our training procedure.....	25
Figure 4.1: The inference boxes show the events class that is found in the video.....	27
Figure 4.2. The CSV file which logs down all the time points of all the event classes (0 – no detected, 1 – detected).....	27



Figure 4.3: Visualize the log data in chart form (green – tileout, red – alarm message).....	28
Figure 4.4: Visualize the log data in chart form (blue – blasting, red – alarm message).....	28
Figure 4.5: The attended time required of a time study (before vs after).....	30
Figure 4.6. The image with motion blur.....	32

## **LIST OF ABBREVIATIONS AND SYMBOLS**

AI	- Artificial intelligence
IE	- Industrial engineer
TF	- Tensor Flow
SOTA	- State-of-The-Art
WBB	- Wet Bead Blasting

# CHAPTER 1

## INTRODUCTION

### 1.1. Background

Lumileds is a lighting company that designs, manufactures, and distributes LEDs, light bulbs, and other lighting related products. It is a global lighting solutions company that enables customers worldwide to have differentiated automotive, mobile, IoT, and lighting applications solutions. As a joint venture between Philips Lighting and Agilent Technologies, Lumileds was founded in November 1999. Lumileds became a business unit within Philips Lighting following Philips' acquisition in 2005 and became known as Philips Lumileds Lighting Company. In December 2016, Philips decided to sell most of its stake in Lumileds to Apollo Global Management. Since then, Lumileds operates as a private company, with Apollo Global Management controlling an 80.1% stake while Philips owns the remaining 19.9%. In the automotive lighting, general lighting, and consumer business markets, Lumileds is the leading global light engine company serving customers. Lumileds operates in 31 countries with 8500 workers and over \$2Billion in annual sales. Headquartered in San Jose, California, Lumileds has operations in Singapore, Malaysia, and the Netherlands and has distribution offices worldwide.



Figure 1.1: Lumileds worldwide offices and plants location.

Lumileds Malaysia Sdn Bhd has two manufacturing plants in Penang, Malaysia. Both the plants are focusing on doing the backend processes of LED production. Once the wafers' front-end processes have been completed in the Lumileds plant in Europe and Singapore, they are sent to the back-end-of-line in Penang. These backend processes include attaching LEDs onto the metal frame, testing the LEDs, sawing the package, and packaging the individual LEDs package. Below is the detail of contact of Lumileds Malaysia Sdn Bhd.

Practicum Organization : Lumileds Malaysia Sdn Bhd

Practicum Department : Industrial Engineering

Address : No.3, Lintang Bayan Lepas 8, Kawasan  
Perindustrian, Bayan Lepas Fasa 4, Mukim 12,  
11900 Penang.

Official Website : <https://www.lumileds.com/>

Phone : +6046434096



Figure 1.2: Lumileds Malaysia Sdn Bhd in Penang.

Lumileds offers lighting related products for automotive, illumination, and speciality applications. For automotive applications, Lumileds products are widely used for car lighting, truck lighting, and motorcycle lighting. Speciality products from

Lumileds are designed to provide cutting-edge solutions for Camera Flash, Display, Infrared (IR), and ultraviolet (UV) applications. The speciality products have wide applications spanning across mobile, consumer electronics, industrial, and automotive markets. Meanwhile, the illumination products from Lumileds are used in various lighting solutions, including architectural, entertainment, high impact retail, horticultural, and sports lighting. Figure 1.3 below shows the images of some of the end products manufactured by Lumileds.

Application	Product			
Automotive	 Luxeon F	 Versat	 SignalSure	 LED Bulbs and Modules
Specialty	 Luxeon IR Domed Line	 Luxeon Flash	 Display LED	 Luxeon UV U Line
Illumination	 Luxeon C Color Line	 Luxeon CZ Color Line	 Luxeon XR-5050 SQR	 Luxeon CoB

Figure 1.3: Lumileds product images.

## 1.2. About Industrial Engineering

Traditionally, industrial engineering is an engineering discipline that developing, improving, and implementing integrated systems of the workforce, resources, knowledge, information, and equipment to optimize a complex system, system, or organization. In Lumileds, the primary responsibility of IE is slightly different from the traditional role of IE. Below is the primary responsibility of IE in Lumileds:

- Perform time study. Time study is a direct and continuous observation of a task, using a videotape camera or machine log to record the time taken to

accomplish a job. It has often been used when there are repetitive work cycles of short to long duration.

- **Resource capacity planning.** Resource capacity planning is a process to determine the ability of the resource in the production line, service department, or function to meet a specified demand over a period of time. The resource could be in the workforce, machine, tool, and floor space. The key metrics used in capacity planning, such as process cycle time, process throughput (unit per hour), resource performance, are obtained via the time study activity.
- **Layout planning.** Layout planning is a process to arrange all the resources that consume space within a facility. These resources might include a work centre, a machine, a cabinet, a workstation, or even a department.
- **Productivity Improvement.** Productivity is defined as a measure of the efficiency of a person completing a task correctly. Productivity improvement can be made via continuous improvement projects to identify the non-value-added step in a process and subsequently eliminate them.

### **1.3. Problem Background**

Time study is a direct and continuous observation of a task from a video to record the time taken to complete a process. Cycle time is the total elapsed time in which a unit of work is acted upon to bring it closer to its desired output. To calculate a process cycle time, time study has often been used when there are repetitive work cycles of short to long duration. Below are the steps of how to perform a time study:

- **Observe the video** on how the work takes shape from one step to another, how the machine moves the work from one step to the next.
- **Determine the exact points** at which the process begins and ends. It must be a precise moment at which we decide the process is complete and the timer will stop. For example, is the process complete when the machine display panel shows the message "Complete"? Or is it finished when it shows a message "Unload Material Successfully"?

- Begin by breaking the entire process apart into however many separate steps we determine there to be. Arrive at a number and stick to that. Our next step will be to determine the precisely fixed endpoints and starting points of each step. The importance of these fixed bookend points is that they will help give our data clarity and reliability later on.
- When a unit of work reaches the starting point we've decided on, set it as our start point. Pause the video when the process reaches the endpoint we decided on. Record the start and end time from the video repeatedly for the subsequent unit of works until we've collected as large a sample size as we intended. Do this for all the steps we have listed until we have a vast pool of sample time for every step recorded.
- Calculate the cycle time duration for every step by subtracting the endpoint with the start point of every sample collected and summarize them in a time study table as shown in the figure below.

Process: Plasma					
Machine: Plasma#01					
Product: 0012					
index	step description	from video		duration(s)	sample
		start point	end point		
1	machine running	0:00:05	0:01:30	85	1
2	change over	0:01:30	0:01:42	12	1
3	machine running	0:01:42	0:03:06	84	2
4	change over	0:03:06	0:03:18	12	2
5	machine running	0:03:18	0:04:47	89	3
6	change over	0:04:47	0:04:59	12	3
7	machine running	0:04:59	0:06:20	81	4
8	change over	0:06:20	0:06:31	11	4
9	machine running	0:06:31	0:07:59	88	5
10	change over	0:07:59	0:08:10	11	5
11	machine running	0:08:10	0:09:31	81	6
12	change over	0:09:31	0:09:44	13	6
13	machine running	0:09:44	0:11:12	88	7
14	change over	0:11:12	0:11:25	13	7
15	machine running	0:11:25	0:12:48	83	8
16	change over	0:12:48	0:12:58	10	8
17	machine running	0:12:58	0:14:25	87	9
18	change over	0:14:25	0:14:37	12	9
19	machine running	0:14:37	0:16:02	85	10
20	change over	0:16:02	0:16:14	12	10

The results of the statistical analysis for the time study		
Statistic	machine running	change over
Mean	85.10	11.80
Mode	81.00	12.00
Median	85.00	12.00
Minimum	81.00	10.00
Maximum	89.00	13.00
Count	10.00	10.00

index	step description	duration(s)	sample
2	change over	12	1
4	change over	12	2
6	change over	12	3
8	change over	11	4
10	change over	11	5
12	change over	13	6
14	change over	13	7
16	change over	10	8
18	change over	12	9
20	change over	12	10
1	machine running	85	1
3	machine running	84	2
5	machine running	89	3
7	machine running	81	4
9	machine running	88	5
11	machine running	81	6
13	machine running	88	7
15	machine running	83	8
17	machine running	87	9
19	machine running	85	10

Figure 1.4: Lumileds time study sample.

## 1.4. Problem Statement

Time study is a time-consuming and tedious task because IE has to manually identify each sample's start point and endpoint from the video. The time required for an IE to complete a time study is positively correlated to the variables such as process cycle time, the number of process step needs to be determined, and the number of samples. From the management perspective, IE should spend more time doing more value-

added tasks such as capacity planning, layout planning, and productivity improvement projects rather than just sitting in front of a laptop to observe the video and write down the cycle time of each sample.

### **1.5. Objectives of Project**

The project aims to reduce the attended time required by an IE to complete the time study analysis by automating the video analysis process. The expected outcomes of the project are as following:

- To create a time study prototype that can detect and identify the critical event, such as the start point and endpoint of a given process from a video and summarize the results into a CSV file.
- The prototype must reduce the IE attended time of time study analysis by at least 80%.
- The prototype results must be correct (allow error margin of +/- 3%).

### **1.6. Benefit of Project**

The project will improve IE productivity by reducing the attended time required by IE to complete the time study analysis. By doing so, IE will have more time to do more value-added tasks such as capacity planning, layout planning, and productivity improvement project, which can help the organization operate more efficiently.



## CHAPTER 2

### RELATED WORKS

#### 2.1 Introduction

Computer vision is an interdisciplinary scientific field that studies how to develop a technique that can help the computer to "see" and understand the high-level content from digital images or videos so that it can be used to automate tasks that the human visual system is doing. Object recognition is one of the most common applications of computer vision. According to Brownlee (2019), it is possible to break down the recognition problem into image classification, object localization, and object detection.

- Image classification. It is an algorithm that predicts the type or class of an object in an image.
- Object localization. It is an algorithm that locates the presence of objects in an image and indicates their location with a bounding box.
- Object detection. It is an algorithm that locates the presence of objects with a bounding box and types or classes of the found objects in an image.

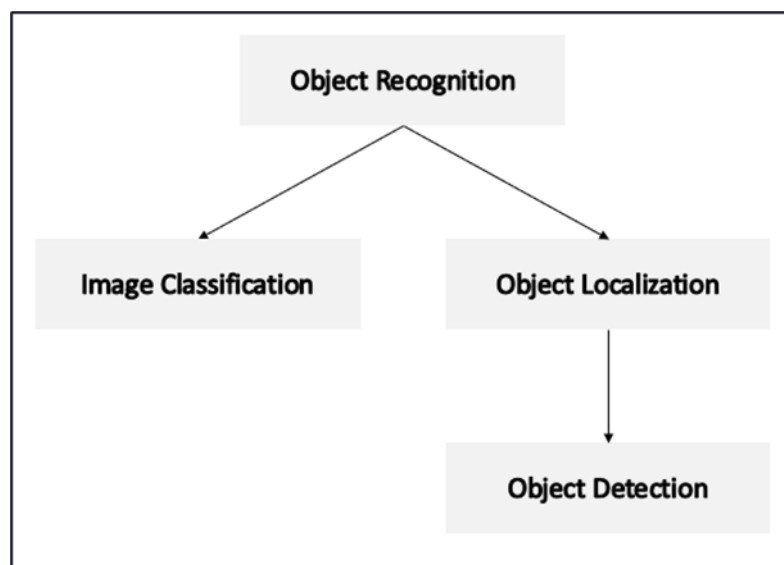


Figure 2.1: Overview of object recognition computer vision tasks.

Sharma (2019) summarizes the differences between image classification, object localization, and object detection as follows:

- Image classification helps us to classify what is contained in an image.
- Image localization will specify the location of a single object in an image.
- Object detection specifies the location of multiple objects in the image.

However, for some of the practitioners, such as (Russakovsky et al., 2015), they often mean "object detection" when they refer to "object recognition". They use the term object recognition broadly to encompass both image classification (a task requiring an algorithm to determine what object classes are present in the image) and object detection (a task requiring an algorithm to localize all objects present in the image).

## 2.2 Type of Object Detector

To identify the presence of a known set of objects from an image or a video stream, we need an object detector model. We can divide pre-existing deep learning-based image object detectors into two major categories: two-stage detector and one-stage detector. The two-stage detector has a more complicated pipeline. In the first stage, the model proposes and generates a set of regions of interest (RoI) by screening out all the positive samples. In the second stage, the algorithm performs regional classification and location refinement on the RoI generated in the previous stage (Du, Zhang, & Wang, 2020). Common two-stage algorithms include R-CNN, Fast R-CNN, and Faster R-CNN.

Girshick, Donahue, Darrell, and Malik (2014) was the first to propose region-based CNN (R-CNN). Their first stage of the model is to apply high-capacity CNN to bottom-up region proposals to localize and segment objects. The second stage is training large CNNs when labelled training data is scarce. Girshick (2015) further enhances his previous R-CNN with a newer version named Fast R-CNN, which is much more efficient and faster.

Ren, He, Girshick, and Sun (2016) introduced a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network

that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which Fast R-CNN uses for detection. They further merged RPN and Fast R-CNN into a single network by sharing their convolutional features using the recently popular terminology of neural networks with 'attention' mechanisms. The RPN component tells the unified network where to look. This whole pipeline is named Faster R-CNN.

Unlike a two-stage detector, a one-stage detector skips the region of interest proposal stage. It treats object detection as a simple regression problem by taking an input image and learning the class probabilities, and bounding box coordinates. Common one-stage algorithms are YOLO, SSD, CenterNet, and EfficientDet.

Liu et al. (2016) presented the first deep network-based object detector that does not resample pixels or features for bounding box hypotheses and is as accurate as approaches that do. Their approach is named Single Shot MultiBox Detector (SSD), which discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is relatively more straightforward than methods requiring object proposals because it eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component.

YOLO (Redmon, Divvala, Girshick, & Farhadi, 2016) treated the object detection task as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

CenterNet by Duan et al. (2019) is an efficient solution that explores the visual patterns within each cropped region with minimal costs. The framework was built

based on a representative one-stage keypoint-based detector named CornerNet (Law & Deng, 2018). CenterNet detects each object as a triplet, rather than a pair, of key points, which improves both precision and recall.

Tan, Pang, and Le (2020) proposed a new family of object detectors called EfficientDet with a weighted bi-directional feature pyramid network (BiFPN), which allows easy and fast multiscale feature fusion. EfficientNet also has a compound scaling method that uniformly scales the resolution, depth, and width for all backbone, feature network, and box/class prediction networks simultaneously.

In general, we can say that the two-stage detectors have high localization and object recognition accuracy, whereas the one-stage detectors achieve high inference speed. Furthermore, the one-stage detectors propose predicted boxes from input images directly without the region proposal step. Thus they are time efficient and can be used for real-time devices.

## 2.3 Related Work

We know that object detection techniques can be used in video highlight detection and video summarization. However, to our knowledge, there are no publicly available implementations or datasets, such as time study using object detection that could be used to compare our work. Despite this different goal, we review object detection of other real-world application methods because all those topics are highly related to our project objectives.

Mandal and Adu-Gyamfi (2020) deploy several state-of-the-art object detection and tracking algorithms to detect and track different classes of vehicles in their regions of interest (ROI). The goal of correctly detecting and tracking vehicles' ROI is to obtain an accurate vehicle count. Multiple object detection models coupled with different tracking systems are applied to access the best vehicle counting framework. The models address challenges associated with varying weather conditions, occlusion, and low-light settings and efficiently extract vehicle information and trajectories through its computationally rich training and feedback cycles. The automatic vehicle counts resulting from all the model combinations are validated and compared against the manually counted ground truths of over 9 hours of

traffic video data obtained from the Louisiana Department of Transportation and Development. Experimental results demonstrate that the combination of CenterNet and Deep SORT and YOLOv4 and Deep SORT produced the best overall counting percentage for all vehicles.

Zaidi et al. (2021) have shown how two-stage and single-stage detectors developed over their predecessors. While the two-stage detectors are generally more accurate, they are slow and cannot be used for real-time applications like self-driving cars or security. However, this has changed in the last few years, where one stage detectors are equally accurate and much faster than the former. They compared the performance of both single and two-stage detectors on PASCAL VOC 2012 and Microsoft COCO datasets. The performance of object detectors is influenced by several factors like input image size and scale, feature extractor, GPU architecture, number of proposals, training methodology, loss function etc., which makes it difficult to compare various models without a common benchmark environment.

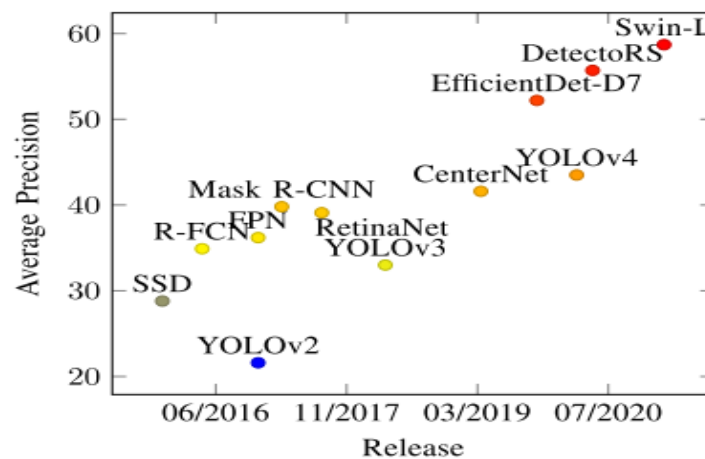


Figure 2.2: Performance of Object Detectors on MS COCO dataset (Zaidi et al., 2021).

Liao, Zou, Shen, Liu, and Du (2021) applied the object detection algorithm to detect cigarette end from the video. After analyzing a variety of target detection algorithms, they considered that EfficientDet can achieve high efficiency under a wide range of resource constraints and can easily carry out multiscale feature fusion so that it has high speed and accuracy in the process of target detection. In this paper, Efficientnet is used as the backbone feature extraction network. A set of fixed scaling

coEfficients is used to scale the network's depth, width, and resolution for preliminary feature extraction to obtain three effective feature layers. Then, to improve the prediction level and enhance the feature extraction, the three effective feature layers are transferred to bifpn with cross-scale connection optimization. The prediction results can be used as the evaluation results of the model. The evaluation results show that EfficientDet has high mAP.

To further improve wheat ear identification and detection counting accuracy under a field environment, Cao et al. (2020) proposed using the EfficientDet algorithm to detect the wheat ear image. The idea is to frame out the wheat in the wheatear image and then count the detected target number to realize the automatic counting of wheat ears. EfficientDet-D3 model is adopted in this paper, and adamw algorithm is used to train the model, with a learning rate of  $1e5$ . The experimental results show that the model can quickly and accurately recognize wheat ear images with different densities under various lighting conditions. The final accuracy rate reaches 92.92%, and the test time of the single sheet is 0.2s. We carry out an ear counting test on 20 wheatear images with different densities, and the average accuracy rate reaches 95.30%. This method can detect wheat ear image quickly, and the detection effect is good.

## CHAPTER 3

### RESEARCH METHODOLOGY

#### 3.1 Introduction

This chapter will provide an overview of the foundational methodology that will act as a guiding strategy for problem-solving. This methodology, which is independent of particular technologies or tools, should provide a framework for proceeding with the methods and processes that will be used to obtain answers and results (Rollins, 2015). Our methodology would consist of 5 steps which are data requirement, data collection, data preparation, modelling, and evaluation. Below is the process flow of the proposed methodology.

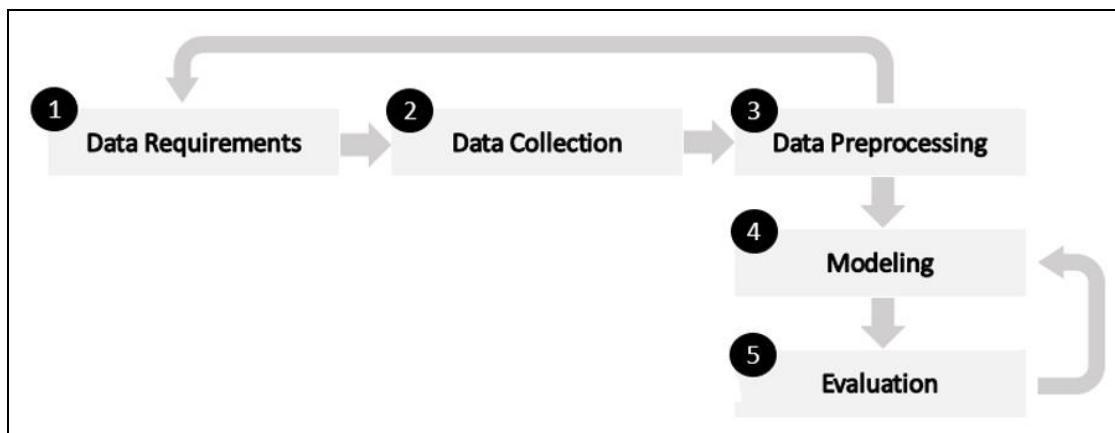


Figure 3.1: Data science methodology.

This is how our proposed solution works. First, we will identify our target events, such as process start point and endpoint, from images or video that we have recorded. The target events selected (from images) will be appropriately labelled and given a unique class name. Then we will use the training and testing images to train our object detection model, which is TensorFlow 2 EfficientDet model. The training process will be done on Google Colab using the GPU accelerator. The training process is stopped when the lost function no longer can be reduced. Now we will have a well-trained model that able to recognize our target events from the images. The trained model is then exported and used for inference. Whenever the model detects the target events from a video, it will draw out the inference boxes and output those

events' time points and confidence score into a CSV file. The compilation of all the happening time of all the target events can be used to calculate the cycle time of every sample.

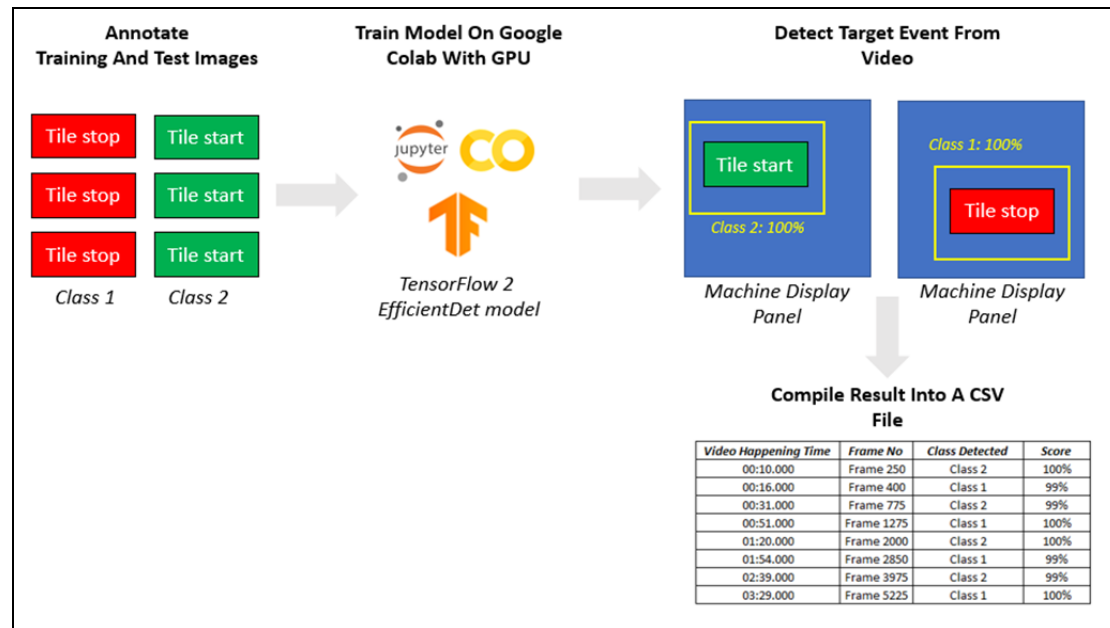


Figure 3.2: Proposed solution.

### 3.2 TensorFlow 2 EfficientDet Model

Tensorflow (TF) is a deep learning framework that powers many of the state-of-the-art (SOTA) models in natural language processing (NLP), speech synthesis, semantic segmentation, and object detection. TensorFlow object detection API is an open-sourced collection of object detection models used by deep learning enthusiasts and by different experts in the field. Currently (as of Jan 2021), six different SOTA models are available in the TensorFlow 2 Detection Model Zoo. They are CenterNet (HourGlass/ResNet), EfficientDet, SSD (MobileNet/ResNet), Faster R-CNN (ResNet/Inception ResNet), Mask R-CNN (Inception ResNet), and ExtremeNet. Although the SOTA models were pre-trained on the COCO 2017 dataset, the Google TensorFlow team make them extremely easy to be trained from scratch using a custom dataset by editing the model config file. Using TensorFlow Object Detection API models instead of implementing the SOTA models on our own gives us more time to focus on the data, which is another crucial factor in achieving the high performance of object detection models.



The main reason we choose EfficientDet as our project model because it is currently the most performant convolutional neural network for classification. Image Classifiers are typically benchmarked on ImageNet, an image database organized according to the WordNet hierarchy, containing hundreds of thousands of labelled images. As we can see in Figure 3.3, 4 out of the top 5 approaches to the ImageNet task are based on EfficientNet.

RANK	MODEL	TOP 1 ACCURACY	TOP 5 ACCURACY	NUMBER OF PARAMS	EXTRA TRAINING DATA	PAPER	CODE	RESULT	YEAR
1	Meta Pseudo Labels (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels	<a href="#">GitHub</a>	<a href="#">ImageNet</a>	2021
2	Meta Pseudo Labels (EfficientNet-B6-Wide)	90%	98.7%	390M	✓	Meta Pseudo Labels	<a href="#">GitHub</a>	<a href="#">ImageNet</a>	2021
3	EfficientNet-L2-475 (SAM)	88.61%		480M	✓	Sharpness-Aware Minimization for Efficiently Improving Generalization	<a href="#">GitHub</a>	<a href="#">ImageNet</a>	2020
4	ViT-H/14	88.55%		632M	✓	An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale	<a href="#">GitHub</a>	<a href="#">ImageNet</a>	2020
5	FixEfficientNet-L2	88.5%	98.7%	480M	✓	Fixing the train-test resolution discrepancy: FixEfficientNet	<a href="#">GitHub</a>	<a href="#">ImageNet</a>	2020

Figure 3.3: The current (as of Jan 21) top 5 classifiers on the ImageNet task.

EfficientDet currently exists in 8 base variations, D0 to D7, with increasing size and accuracy. Choosing an EfficientDet model involves a trade-off between accuracy and performance, which depends on the use case. For our case, we will just use the "EfficientDet D0 512x512", which has the shortest inference time as our starting model.

### 3.3 Google Colab

Google Colaboratory is a free online cloud-based Jupyter notebook environment that allows us to train our machine learning and deep learning models on CPUs, GPUs, and TPUs. Training a deep learning model usually would take numerous hours on a CPU. We've all faced this issue with our local machines. GPUs and TPUs, on the other hand, can train these models in a matter of minutes or seconds. The main reason we choose Google Colab to train our EfficientNet simply because it gives us a decent GPU for free, which we can continuously run for 12 hours. It's an incredible online

browser-based platform that allows us to build our large deep learning models without burning a hole in our pockets.

### 3.3 Data Requirements

In the beginning, we identified which machine will be chosen for the time study. The selected machine must satisfy the following criteria:

- It doesn't have a machine log to record down the detail of the process. Hence the time study only can be done by using a manual video recording method.
- It has a display panel attached to the machine, which will show the real-time event of the process. For example, the display panel will show which process step the material currently at? How many materials have been processed? What is the machine status? Is the machine running or stop?

Basically, we need the video footages that record down the whole process event shown in the machine's display panel. IE will use the messages or animations shown in the display panel to calculate the cycle time of each step. A total of three different processes were selected for possible time study candidates. They are as following:

- Wet Bead Blasting (WBB). It is a process used to clean the workpieces.
- Platelet Attach (PA). It is a process used to attach platelet onto the workpieces.
- Automatic Visual Inspection (AVI). It is a process used to inspect workpieces.

All the above three machines were shortlisted because they are among the bottleneck processes in Lumileds and need consistent attention on the machine throughput. However, after extensive research on all the three machines had been done, we only selected the WBB process as our project's subject. We found out that both the PA and AVI have the logfile sufficient for IE to calculate cycle time without having to record the display panel.




	<i>Wet Bead Blasting (WBB)</i>	<i>Platelet Attach (PA)</i>	<i>Automatic Visual Inspection (AVI)</i>
Machine			
Display Panel	Yes	Yes	Yes
Logfile	No	Yes	Yes

Figure 3.4: Time study candidates.

### 3.3.1 Process Flow of WBB

WBB is a machine used to clean the workpieces by blasting a high-pressure solution onto a workpiece from a different predefined direction. The complete process flow of a workpiece in WBB machine is described below:

1. The operator raises the WBB door and places a workpiece onto the input area.
2. The conveyor moves the workpiece from the input area to the turntable.
3. The turntable rotates and transfers the workpiece into the blasting chamber.
4. After the blasting process, the turntable rotates again to transfer the workpiece out from the blasting chamber.
5. The conveyor moves the workpiece from the turntable to the cooling chamber.
6. The conveyor moves the workpiece from the cooling chamber to the output area.
7. The operator collects the workpiece from the output area.

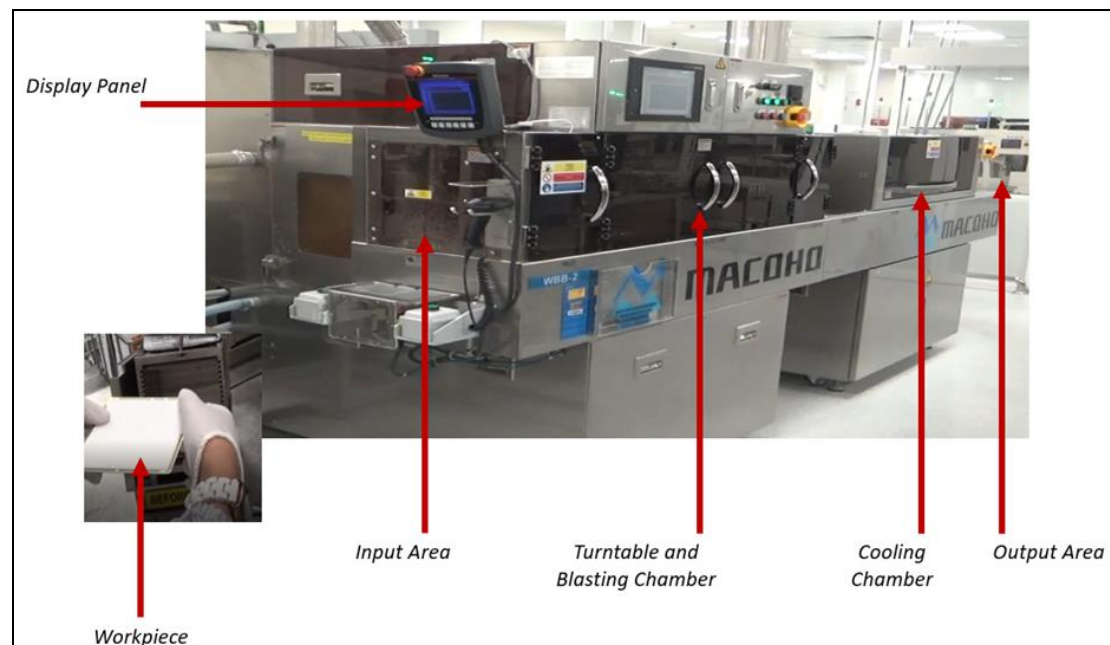





Figure 3.5: Different part of WBB.

As we can see from Figure 11, the machine covers of WBB are black. Hence, it is pretty difficult for us to record what is happening inside the machine. We can understand what is happening on the workpiece inside the WBB by looking at the display panel that shows the real-time event of the process (see Figure 3.6).

<i>Display Panel</i>	<i>Event Description</i>
	A workpiece at the input area
	A workpiece at the turntable (before blasting)
	A workpiece inside the blasting chamber

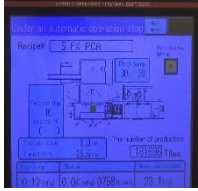
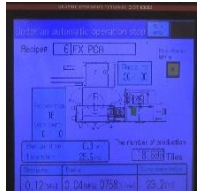


	A workpiece at the turntable (after blasting)
	A workpiece at the cooling chamber
	A workpiece at the output area
	Machine alarm message

Figure 3.6: Display panel of WBB and the event description.

### 3.3.2 Time Study of WBB

The time study of WBB is done by recording the machine display panel. IE then analyzes the video to get the three crucial event time from it. The three important events are:

- What is the blasting time of a workpiece?
- How long is the alarm time (if any) during the process?
- What is the cycle time (throughput) of the process?

IE calculate the blasting time of a workpiece by counting how long the display panel shows the image "A workpiece inside the blasting chamber" from the video. Like the blasting time, the alarm time is also calculated by counting how long the display panel shows the image "Machine alarm message". However, for the cycle time

(throughput), it is calculated by using the time difference between each subsequent workpiece reach the output area.

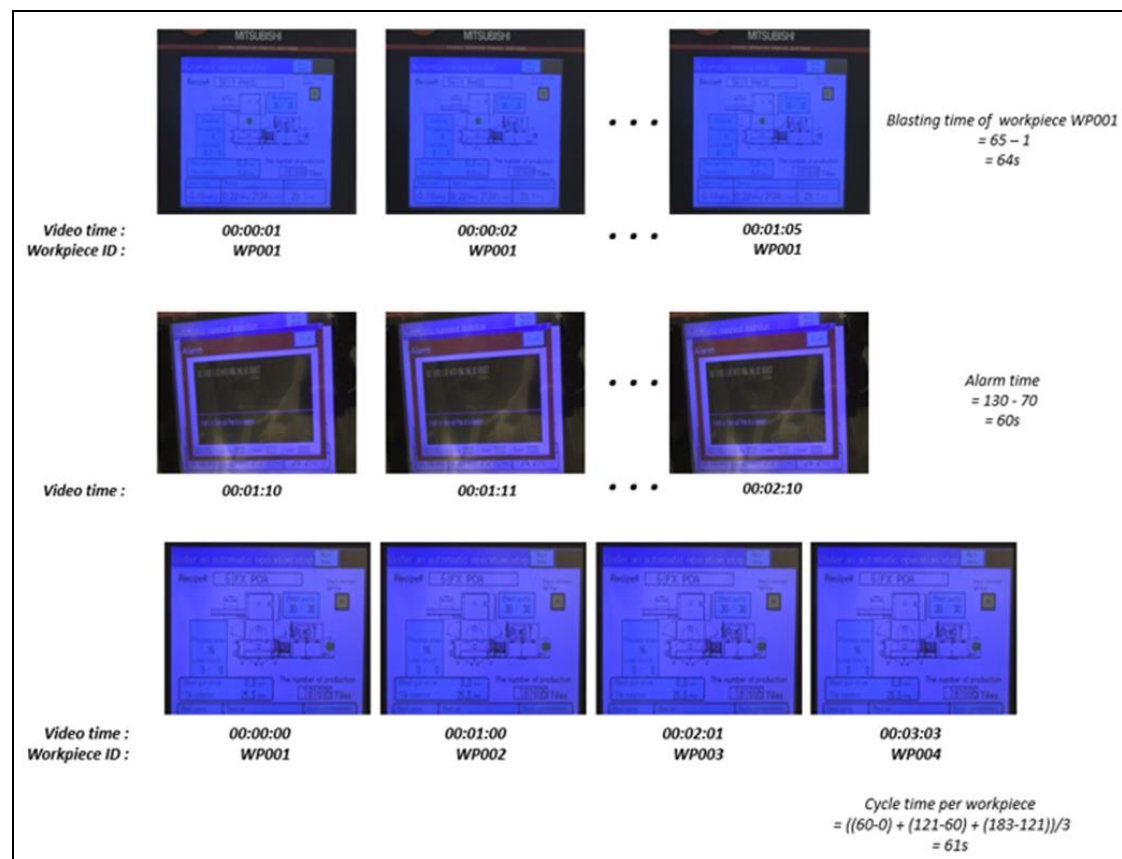


Figure 3.7: Sample calculation of time study of WBB.

### 3.4 Data Collection

In the data collection phase, we must think about how does the computer sees the images. It is essential to understand the machine's display panel environment, lighting conditions, or type of cam recorder as they would vary a lot in their background, image quality, lighting, etc. We tried to get the most variable and diverse training dataset when recording the WBB display panel. Here are some guidelines:

- get images from different angles
- change lighting conditions
- take images with good quality and in focus
- change recording distance

We set our image/video requirements as follows:

- The video recording has a resolution of 1440 x 1080 pixels.
- The video recording has a frame rate of 25 frames/second.
- The video recorded covers a complete cycle of a process of a batch.

Most research papers and consumer use cases tend to use low-resolution images for training deep learning models, often as small as 256 x 256 pixels. Many resize the ImageNet data set images down to this resolution. The challenge in keeping the images large is that the deep learning model size explodes. However, we recorded the video with a larger resolution (1440 x 1080 pixels) to have a better accuracy while having the flexibility to scale down the video resolution if the training process takes a too long time.

### **3.5 Data Preparation**

The data preparation phase comprises all activities used to construct the data set that will be used in the modelling stage. Our model in the next stage will be based on the TensorFlow object detection API. TensorFlow object detection API needs a special format for all input data, called TFRecord. It is Tensorflow's own binary storage format. According to Gamauf (2018), by using a binary file format for the storage of our large dataset, we can significantly improve our import pipeline's performance and the training time of our model. This is because binary data takes up less space on disk, takes less time to copy, and can be read much more efficiently from disk. Pure performance isn't the only advantage of the TFRecord file format. Most importantly, it is optimized for use with Tensorflow in multiple ways. Below are the important steps that we performed in sequence in this stage:

- test-train split
- image annotation
- data transformation and label map creation

#### **3.5.1 Test-Train Split**

According to Ripley (2007), a test set is a set of examples used only to assess the performance of a fully specified classifier. Meanwhile, a training set is a set of examples used for learning and fit the classifier's parameters. We used an 70/30 split as it is quite a commonly occurring ratio, often referred to as the Pareto principle.

### 3.5.2 Image Annotation

Our images need to be annotated for object detection. Image annotation is the process that regions for all objects of interest that might be presented in our datasets are manually defined as bounding boxes, and ground truth labels are set for each and every box. We used an open-source graphical image annotation tool called the LabelImg which can be downloaded from the website <https://tzutalin.github.io/labelImg/> to label object bounding boxes in our images. Annotations were then saved as XML files in PASCAL VOC format, the format that suits our purposes.

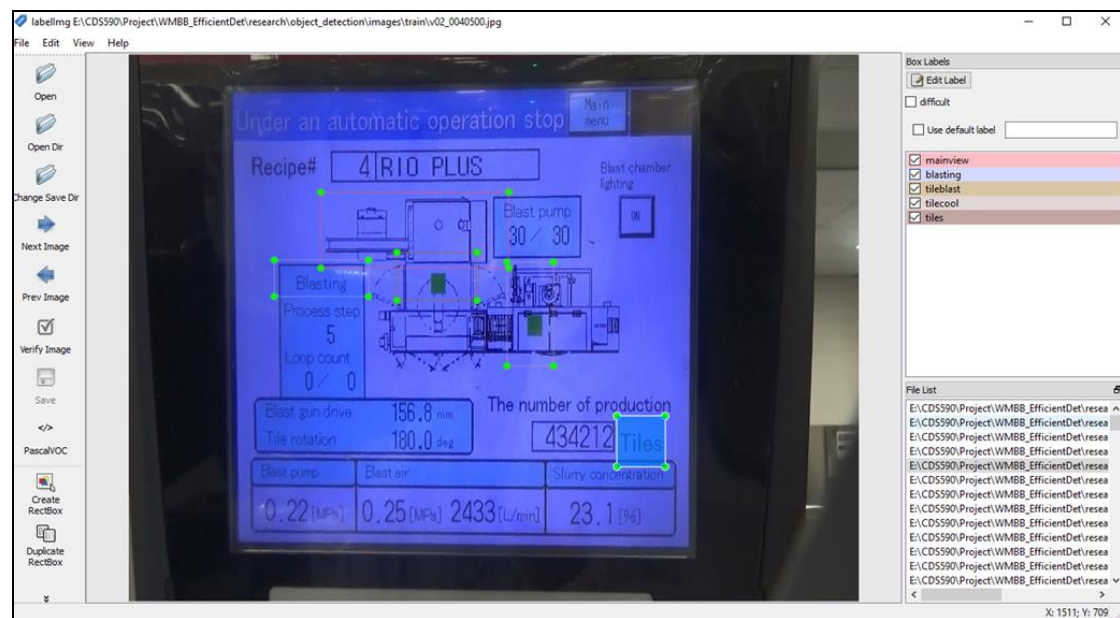


Figure 3.8: The Labelimg used to annotate the images.

A total of seven different classes were annotated from the 211 images. The summary of image annotation is described in Figure 3.9.



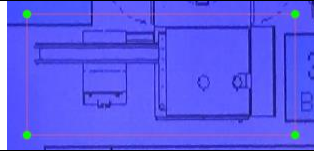


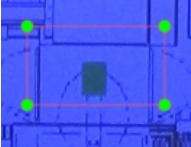
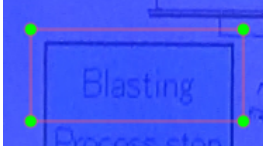

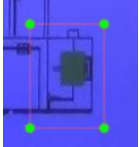
Class	Train Set	Test Set	Purpose	Annotation
mainview	119	52	To ensure display panel in the correct page	
alarm	29	11	To detect alarm status	
tilein	33	13	To detect workpiece (tile) at the input area	
tileblast	110	48	To detect workpiece (tile) at the blasting area	
blasting	91	42	To detect machine in blasting status	
tilecool	30	14	To detect workpiece (tile) at the cooling area	
tileout	35	15	To detect workpiece (tile) at the output area	

Figure 3.9: Summary of Image Annotation.

### 3.5.3 Data Transformation and Label Map Creation

Our goal is to transform each of our annotated datasets (training and testing) in XML format into the TFRecord format. Finally, a Label Map is created. It is a simple .txt file (.pbtxt to be exact). It links labels to some integer values. The TensorFlow Object Detection API needs this file for training and detection purposes.

## 3.6 Modelling

We build our object detection model using the "EfficientDet D0 512x512" from TensorFlow 2 Detection Model Zoo. The pre-train model can be downloaded from [here](#):

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/tf2\\_detection\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md). We can easily re-train the model from scratch by using our custom dataset by modifying the model configs file. We will train object detection API using Google Colab with GPU. Below is the general step we need to do:

1. Install TensorFlow object detection API.
2. Set up Object Detection directory and python virtual environment at once.
3. Gather and label pictures.
4. Generate training and testing dataset.
5. Create train tfrecord and test tfrecord files.
6. Create label map and configure training.
7. Setup Google Colab for TensorFlow 2 EfficientDet model training.
8. Start model training on Google Colab.
9. Export inference graph
10. Try out our object detector for new video.

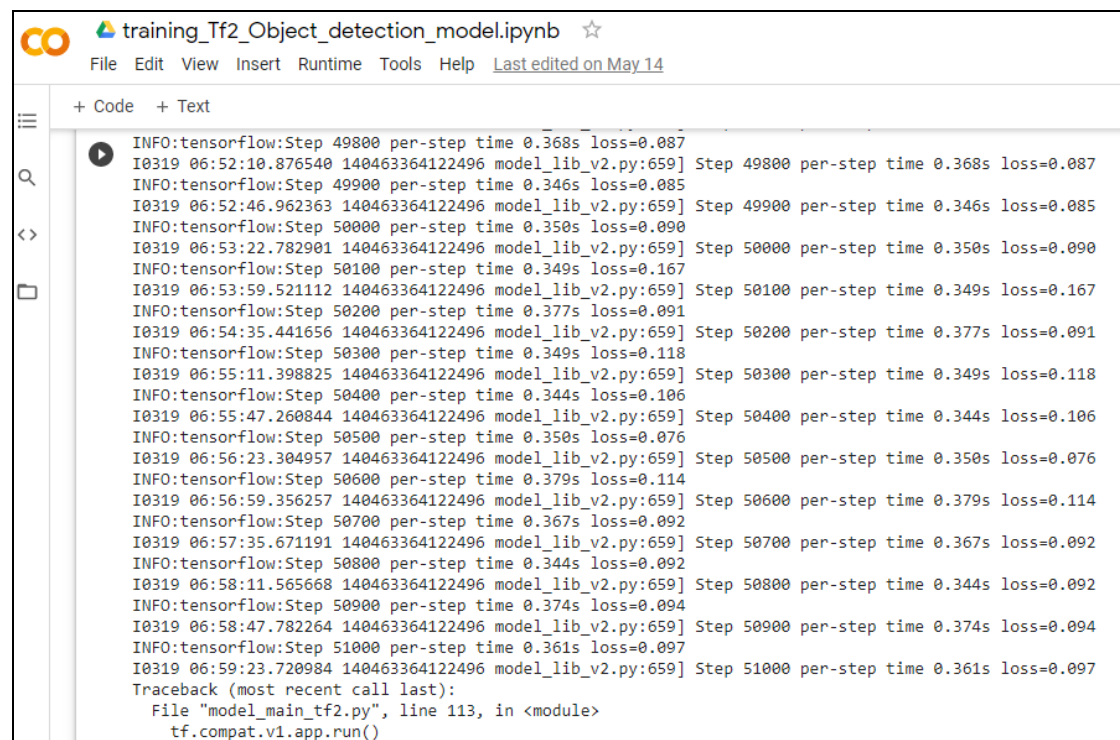
EfficientDet provides a set of hyperparameters that allow for the alteration of both the network architecture and the training strategy. Some of the important hyperparameters include "classification\_loss", "learning\_rate\_base", and "warmup\_learning\_rate" which can be fine-tuned in the pipeline.config file. The training process can be visualized using the TensorBoard. The events file generated as part of the evaluation will display the evaluation metrics on a TensorBoard.

### **3.7 Evaluation**

We evaluated the model's quality in the evaluation phase and checked whether it addresses our project problem statement fully and appropriately. Doing so requires computing various diagnostic measures and other outputs, such as tables and graphs, using a testing set for a predictive model. If the evaluation result is not satisfying, we

may need to go back to the modelling phase again and set up our model with a different setting and hyperparameter.

To examine our training success, we output the TensorBoard showing how our model's loss function has decreased over the training procedure. The lower the loss, the better. After training for 9-10 hours of about 51000 epochs, we could see total loss get down to 0.08.



```

training_Tf2_Object_detection_model.ipynb ☆
File Edit View Insert Runtime Tools Help Last edited on May 14

+ Code + Text

INFO:tensorflow:Step 49800 per-step time 0.368s loss=0.087
I0319 06:52:10.876540 140463364122496 model_lib_v2.py:659] Step 49800 per-step time 0.368s loss=0.087
INFO:tensorflow:Step 49900 per-step time 0.346s loss=0.085
I0319 06:52:46.962363 140463364122496 model_lib_v2.py:659] Step 49900 per-step time 0.346s loss=0.085
INFO:tensorflow:Step 50000 per-step time 0.350s loss=0.090
I0319 06:53:22.782901 140463364122496 model_lib_v2.py:659] Step 50000 per-step time 0.350s loss=0.090
INFO:tensorflow:Step 50100 per-step time 0.349s loss=0.167
I0319 06:53:59.521112 140463364122496 model_lib_v2.py:659] Step 50100 per-step time 0.349s loss=0.167
INFO:tensorflow:Step 50200 per-step time 0.377s loss=0.091
I0319 06:54:35.441656 140463364122496 model_lib_v2.py:659] Step 50200 per-step time 0.377s loss=0.091
INFO:tensorflow:Step 50300 per-step time 0.349s loss=0.118
I0319 06:55:11.398825 140463364122496 model_lib_v2.py:659] Step 50300 per-step time 0.349s loss=0.118
INFO:tensorflow:Step 50400 per-step time 0.344s loss=0.106
I0319 06:55:47.260844 140463364122496 model_lib_v2.py:659] Step 50400 per-step time 0.344s loss=0.106
INFO:tensorflow:Step 50500 per-step time 0.350s loss=0.076
I0319 06:56:23.304957 140463364122496 model_lib_v2.py:659] Step 50500 per-step time 0.350s loss=0.076
INFO:tensorflow:Step 50600 per-step time 0.379s loss=0.114
I0319 06:56:59.356257 140463364122496 model_lib_v2.py:659] Step 50600 per-step time 0.379s loss=0.114
INFO:tensorflow:Step 50700 per-step time 0.367s loss=0.092
I0319 06:57:35.671191 140463364122496 model_lib_v2.py:659] Step 50700 per-step time 0.367s loss=0.092
INFO:tensorflow:Step 50800 per-step time 0.344s loss=0.092
I0319 06:58:11.565668 140463364122496 model_lib_v2.py:659] Step 50800 per-step time 0.344s loss=0.092
INFO:tensorflow:Step 50900 per-step time 0.374s loss=0.094
I0319 06:58:47.782264 140463364122496 model_lib_v2.py:659] Step 50900 per-step time 0.374s loss=0.094
INFO:tensorflow:Step 51000 per-step time 0.361s loss=0.097
I0319 06:59:23.720984 140463364122496 model_lib_v2.py:659] Step 51000 per-step time 0.361s loss=0.097
Traceback (most recent call last):
  File "model_main_tf2.py", line 113, in <module>
    tf.compat.v1.app.run()
  
```

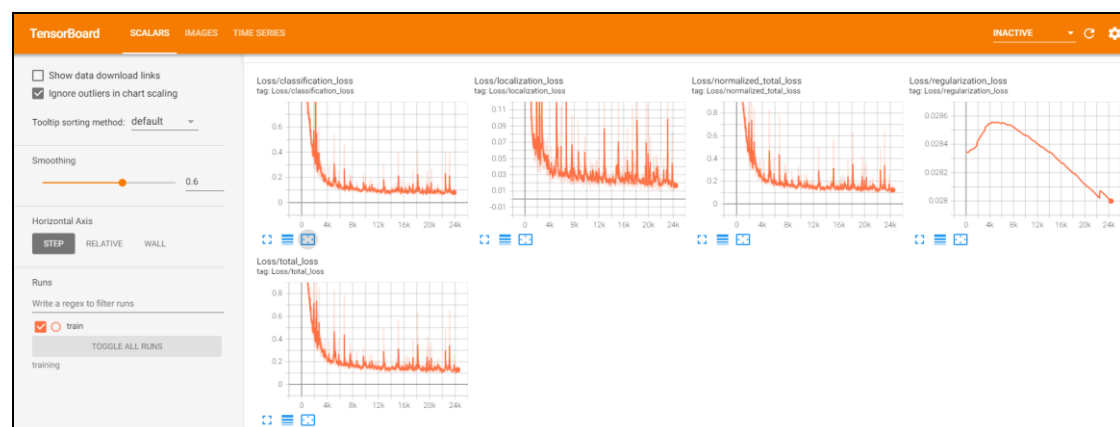


Figure 3.10. TensorBoard output to visualize our training procedure.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

This chapter focuses on discussion and results obtained from the object detector prototype that is developed using TensorFlow2 and Google Colab. The gathered acceptance criteria from the IE are used in validating whether our prototype meet the project's objectives or not. Based on the criteria, the IE team give us a list of acceptance criteria cases. The requirements and the test results are shown in Table 4.1.

Table 4.1: User acceptance testing of the project.

Number	Acceptance Requirement	Critical	Test Result
1	The prototype must execute to end of job.	Yes	Yes
2	The results of prototype must be correct.	Yes	Yes
3	The prototype must reduce IE attended time of time study by at least 80%.	Yes	Yes

In summary, the prototype can help IE detect the predefined three crucial event times from the video and subsequently visualize the results in an Excel spreadsheet. Doing so allows the IE to improve productivity by reducing the attended time required by IE to complete the time study analysis.

#### 4.2 Prototype Overview

Now that we have a trained TensorFlow2 object detector. Whenever the model detects the target events from a video, it will draw out the inference boxes and output those events' occurring time and confidence score into a CSV file. The compilation of the time points of all the target events can be used to calculate the cycle time of every sample.

Below are a few screenshots showing how the prototype looks like:

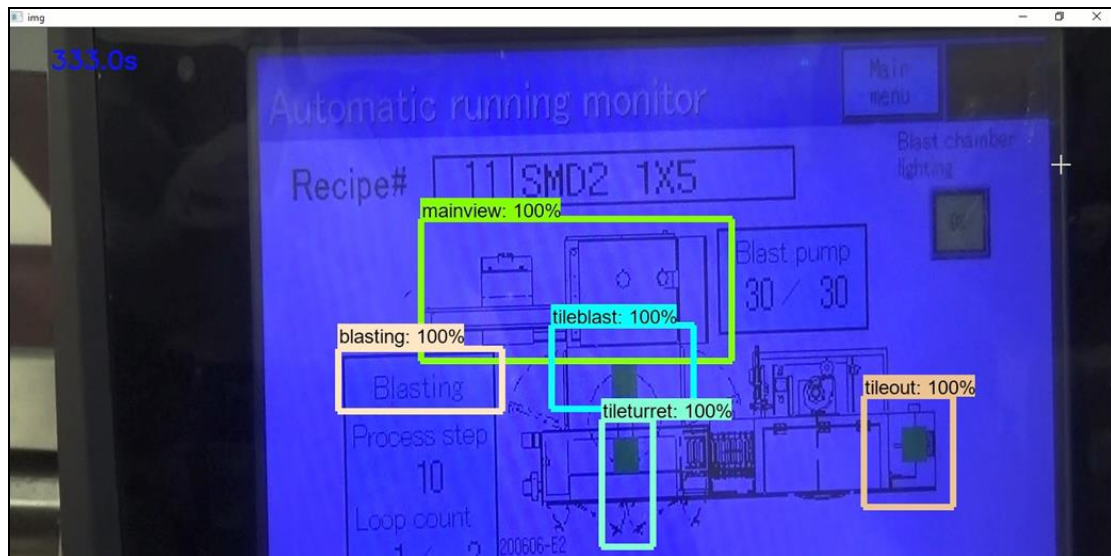


Figure 4.1: The inference boxes show the events class that is found in the video.

time_point	mainview	tileblast	tileturret	tiles	tilemid	blasting	tilecool	alarm	tileout	tilein	source
42	1	1	1	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
43	1	1	1	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
44	1	1	1	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
45	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
46	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
47	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
48	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
49	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
50	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
51	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
52	1	1	1	1	0	0	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
53	1	1	1	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
54	1	1	1	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
55	1	1	1	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
56	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
57	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
58	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
59	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
60	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
61	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS
62	1	1	0	1	0	1	0	0	0	0	E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS

Figure 4.2. The CSV file which logs down all the time points of all the event classes (0 – no detected, 1 – detected)

The prototype is able to help IE detect the three crucial event times from the video and subsequently visualize the results in an Excel spreadsheet. The three important events are as following:

- What is the blasting time of a workpiece?
- How long is the alarm time (if any) during the process?
- What is the cycle time (throughput) of the process?

From Figure 4.3, we can see that the red colour indicates that our prototype detects alarm messages from the video. Meanwhile, from Figure 4.4, the blue colour indicates that our prototype detects blasting messages from the video. And also, from both Figure 4.3 and Figure 4.4, we can observe a cyclic pattern from the charts, which can help IE to find out how long the cycle time of each process is.

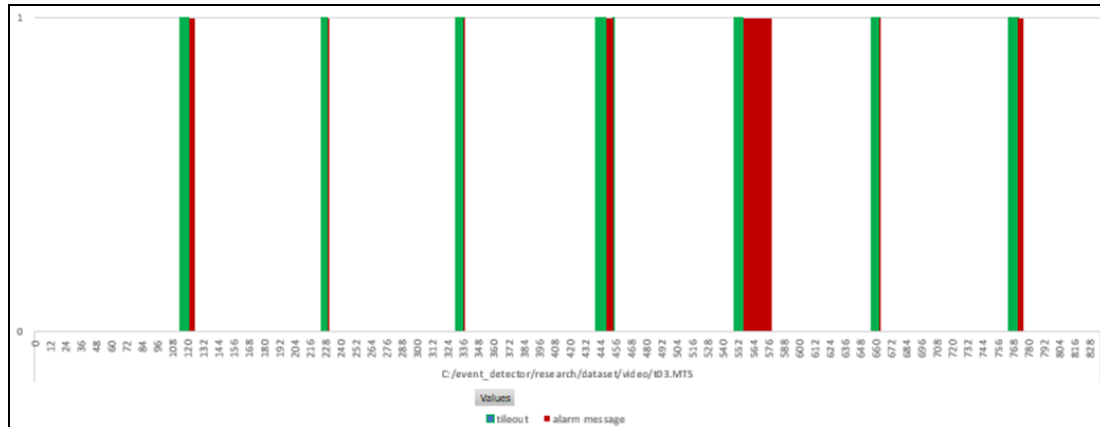


Figure 4.3: Visualize the log data in chart form (green – tileout, red – alarm message)

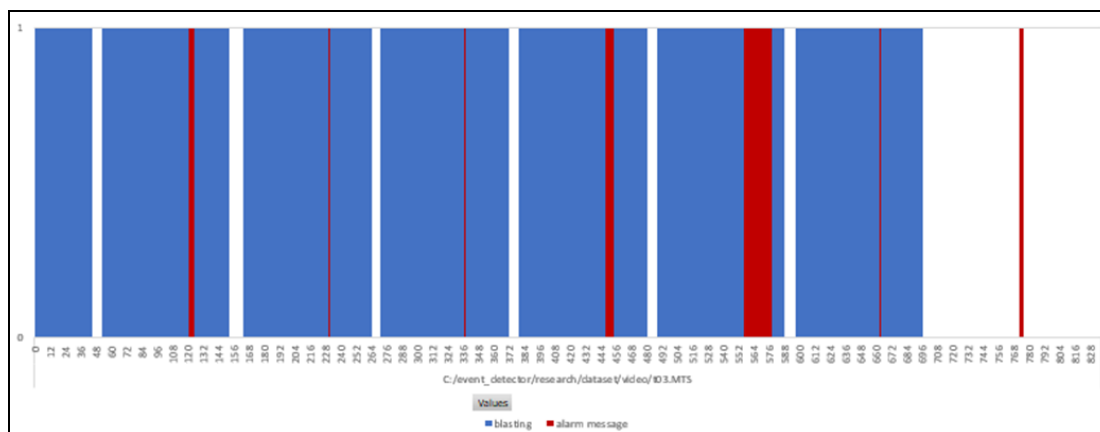


Figure 4.4: Visualize the log data in chart form (blue – blasting, red – alarm message)

### 4.3 Attended Time

One of the project's objectives is to reduce the time study analysis process that needs IE direct involvement (attended time). To obtain the baseline data for this study, ten different time study activities were collected in March 2021. The data were also used as a reference for the current state (sometimes called benchmark data) and are compared with the future state (using prototype).

As we can see from Table 4.2, the current way of time study requires IE to spend a lot of time (about 90% of total analysis time) observing the video and write down the time points into Excel. However, IE can eliminate this video observation time by using the prototype and let our prototype log down the time points.

Table 4.2: Total time required of the time study (before vs after using the prototype).

Process:		WBB	WBB	WBB	WBB	WBB	WBB	WBB	WBB	WBB	WBB
Product:		Product 01	Product 02	Product 03	Product 04	Product 05	Product 06	Product 07	Product 08	Product 09	Product 10
Workpiece/Batch:		20	20	20	20	20	20	20	20	20	20
Process of Time Study (Before)											
Done by	Sample 01	Sample 02	Sample 03	Sample 04	Sample 05	Sample 06	Sample 07	Sample 08	Sample 09	Sample 10	
Open media player "MPC-HC" and load target video	IE	12	14	8	9	14	12	11	10	13	15
Observe the video and write down the event time point in Excel	IE	1510	1725	1925	2458	3141	3350	3750	4149	4549	4948
Calculate the cycle time of each sample and get the average	IE	301	335	400	276	320	320	318	345	289	375
Total time required (s)	IE	1823	2074	2333	2743	3475	3682	4079	4504	4851	5338
Process of Time Study (After Using Prototype)											
Done by	Sample 01	Sample 02	Sample 03	Sample 04	Sample 05	Sample 06	Sample 07	Sample 08	Sample 09	Sample 10	
Open the prototype and run	IE	13	10	14	7	12	12	14	9	10	14
Prototype detect the target event from video and compile the time points into CSV	Prototype	6030	6870	7600	9762	12467	13401	14999	16597	18195	19793
Calculate the cycle time of each sample and get the average	IE	70	81	78	55	67	72	75	80	79	89
Total time required (s)	Prototype	6030	6870	7600	9762	12467	13401	14999	16597	18195	19793
Total time required (s)	IE	83	91	92	62	79	84	89	89	89	103

Figure 4.5 further summarizes how much attended time IE can save by using the prototype to help them do the time study analysis. The study's key results included preliminary evidence indicating that more than 95% of video observation time is saved and converted into unattended time.

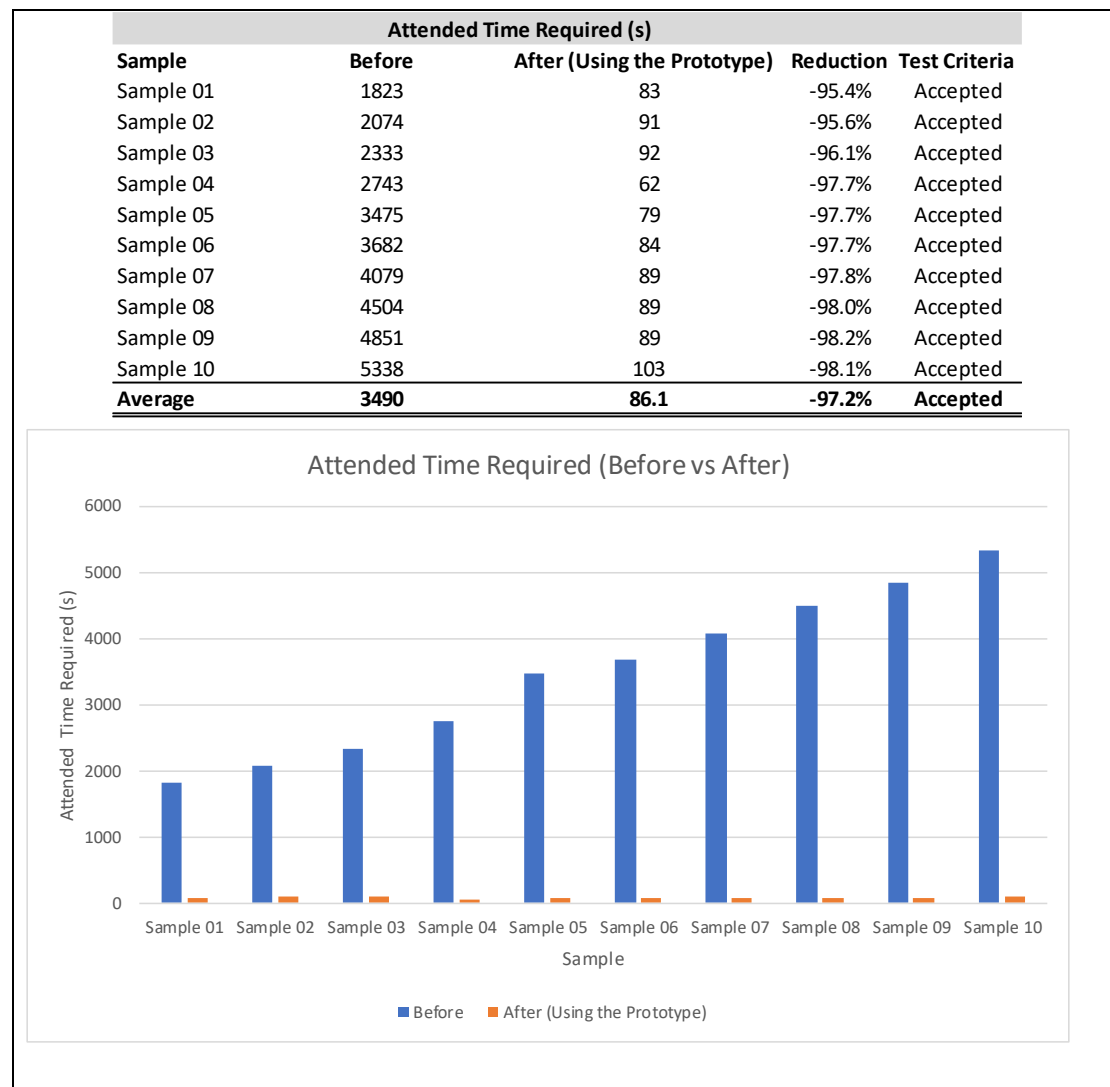


Figure 4.5: The attended time required of a time study (before vs after).

The before and after stats prove that the prototype can improve IE productivity by reducing the attended time required by IE to complete the time study analysis. By doing so, IE will have more time to do more value-added tasks such as capacity planning, layout planning, and productivity improvement project, which can help the organization operate more efficiently.

#### 4.4 Time Study Result Accuracy

One of the IE's acceptance requirements is that the prototype time study results must be accurate with an error rate within  $\pm 3\%$ . The error rate of the time study is calculated using the formula below:



$$\text{Error} = \frac{\text{Cycle Time Value Calculated Using Prototype} - \text{True Value}}{\text{True Value}}$$

To obtain the baseline data for this study, ten different time study activities were collected in March 2021. An experienced IE did all the time study, and his cycle time value will be regarded as the true value when calculating the error of our prototype. Table 4.3 summarizes the error rate between the prototype results value vs the true value. The study's results indicate that the average error rate of our prototype is only around -0.1%.

Table 4.3: The error rate of the prototype.

Sample	Class	True Value (s)	Prototype Value (s)	Error %	Test Criteria
Sample 1	Alarm	17	17	0.0%	Accepted
Sample 2	Alarm	18	18	0.0%	Accepted
Sample 3	Alarm	21	21	0.0%	Accepted
Sample 4	Alarm	25	25	0.0%	Accepted
Sample 5	Alarm	33	32	-3.0%	Accepted
Sample 6	Alarm	33	33	0.0%	Accepted
Sample 7	Alarm	37	37	0.0%	Accepted
Sample 8	Alarm	43	43	0.0%	Accepted
Sample 9	Alarm	47	47	0.0%	Accepted
Sample 10	Alarm	49	49	0.0%	Accepted

Sample	Class	True Value (s)	Prototype Value (s)	Error %	Test Criteria
Sample 1	Blasting	92	92	0.0%	Accepted
Sample 2	Blasting	103	103	0.0%	Accepted
Sample 3	Blasting	116	116	0.0%	Accepted
Sample 4	Blasting	147	148	0.7%	Accepted
Sample 5	Blasting	189	189	0.0%	Accepted
Sample 6	Blasting	201	200	-0.5%	Accepted
Sample 7	Blasting	225	225	0.0%	Accepted
Sample 8	Blasting	249	249	0.0%	Accepted
Sample 9	Blasting	274	274	0.0%	Accepted
Sample 10	Blasting	297	297	0.0%	Accepted

Sample	Class	True Value (s)	Prototype Value (s)	Error %	Test Criteria
Sample 1	Throughput	301	301	0.0%	Accepted
Sample 2	Throughput	343	343	0.0%	Accepted
Sample 3	Throughput	380	379	-0.3%	Accepted
Sample 4	Throughput	488	488	0.0%	Accepted
Sample 5	Throughput	623	623	0.0%	Accepted
Sample 6	Throughput	670	671	0.1%	Accepted
Sample 7	Throughput	749	749	0.0%	Accepted
Sample 8	Throughput	829	829	0.0%	Accepted
Sample 9	Throughput	909	911	0.2%	Accepted
Sample 10	Throughput	989	989	0.0%	Accepted

However, it is worth checking why some of the prototype results are slightly different from the true value provided by the experienced IE. For example, we investigated the class "Alarm" of Sample 7. The true value provided by the IE is 33s, but our prototype can only detect a total of 32s alarm time from the video. Further investigation showed that the difference in 1s is caused by our object detector not detecting the alarm message from this image (Figure 4.6). As we can see from Figure 4.6, the alarm message is not clear because an operator accidentally distorts our camera when the recording is in progress. The distortion caused the image captured not clear and has some motion blur. We have tried several ways to overcome it. For example, we lower the score threshold of the object detector from 0.95 to 0.90. Score thresholds are computed assuming that the model never returns predictions with a score lower than this value. By lowering the score threshold, the detector now more

robust to detect those not clear images. However, this method also comes with the drawback that our detector is also now more vulnerable to detect unwanted noise.

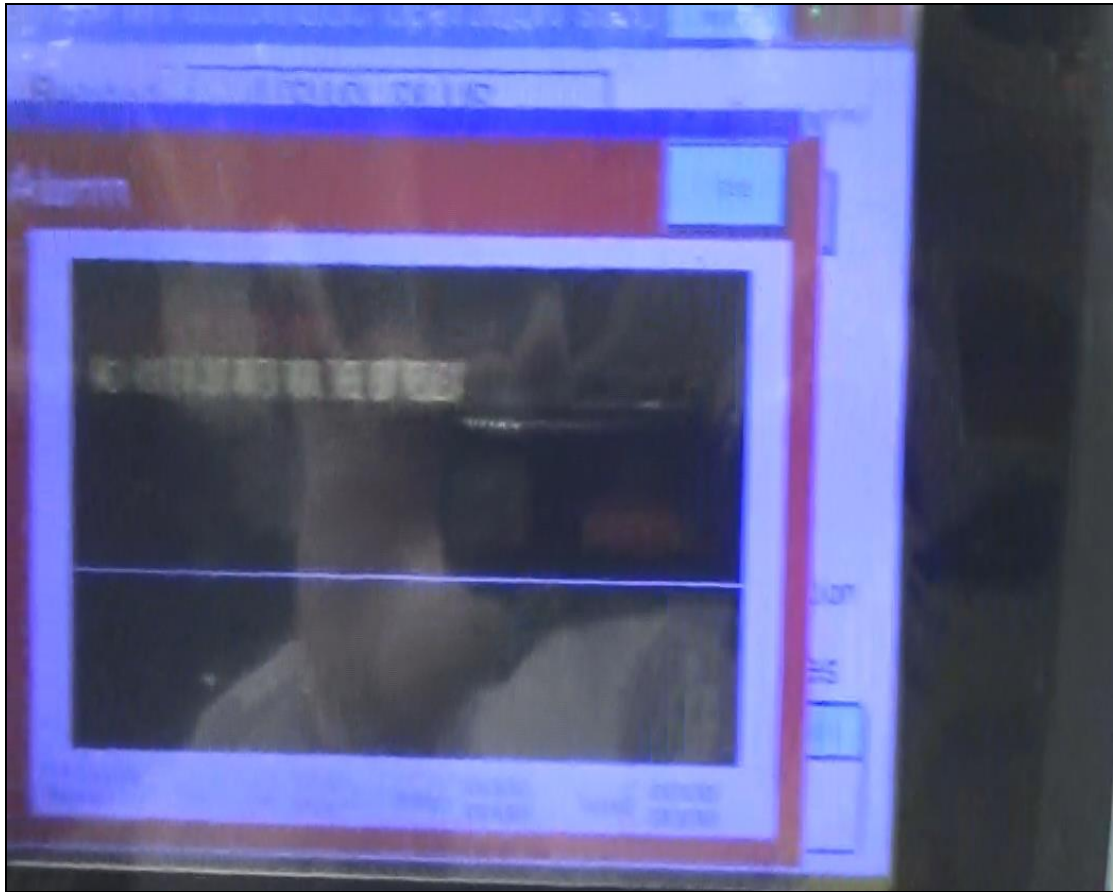


Figure 4.6. The image with motion blur.

## **CHAPTER 5**

### **CONCLUSION & LESSON LEARNED**

#### **5.1 Introduction**

In this project, a time study prototype that can detect and identify the critical event from a video was successfully developed. This prototype utilized the free TensorFlow object detection API for implementing the SOTA models and Google Colab to train our object detection models.

Implementing this artificial intelligence-powered prototype can reduce the time required by an IE to complete the time study analysis by automating the video analysis process. IE no longer have to spend a lot of time observing the video and write down the time points into Excel. The results generated using the prototype also meet the acceptance requirement, and it should become a reliable tool to improve IE overall productivity. By having this tool, IE will have more time to do more value-added tasks such as capacity planning, layout planning, and productivity improvement project, which can help the organization operate more efficiently.

#### **5.2 Challenge and Lesson Learned**

TF2OD API configuring can be one of the most complex and equally rewarding tasks if we want to leverage the power of plug and play already trained deep learning models and quickly train, and with some bit of enhancements, we can deploy it. There are many tutorial/guides available on the Internet to set up and use TF2OD API to perform object detection in images/video. However since it's so new and documentation is pretty sparse, it can be tough to get up and running quickly. Reading other guides and tutorials we found that they glossed over specific details which took me a few hours to figure out on my own. Hence, we still faced many difficulties installing the TF2OD API correctly because it comes with many dependencies, and we need to make sure that we set the Tensorflow directory structure correctly.

One of the valuable tutorials that help us a lot is the Youtube video created by Jay Bhatt <https://www.youtube.com/watch?v=a1br6gW-8Ss&list=LL&index=7&t=534s>. In the video, Jay Bhatt goes over the entire setup process and explain every step to get things working. He even makes a PowerShell script to help the viewer install and set up all the prerequisite dependencies or software.

Arguably, the best-known disadvantage of neural networks is their "black box" nature. Simply put, we don't know how or why our TF2OD came up with a particular output. For example, when we put an image of an "alarm" message into a TF2OD, it detects nothing. It is challenging to understand what caused it to arrive at this prediction. For a deep neural network, as there is no proper defined mathematical model, the learning is spread across the hidden units, and the learning process is considered a Black Box there. Determining the training method/hyperparameters for TF2OD is a black art with no theory to guide us. Many hyperparameters have to be tuned to get to a place in the error space where the error is small enough so that the model will be good enough. The training process of TF2OD is very slow, and adding the tuning of the hyperparameters into that makes it even slower. What we do in this project is search through the Internet to check the parameters/configurations used by other practitioners. Then we used them as our guidelines for the model hyperparameter tuning.

## REFERENCES

- Brownlee, J. (2019). A Gentle Introduction to Object Recognition With Deep Learning.
- Du, L., Zhang, R., & Wang, X. (2020). *Overview of two-stage object detection algorithms*. Paper presented at the Journal of Physics: Conference Series.
- Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., & Tian, Q. (2019). *Centernet: Keypoint triplets for object detection*. Paper presented at the Proceedings of the IEEE International Conference on Computer Vision.
- Gamauf, T. (2018). Tensorflow Records? What they are and how to use them.
- Girshick, R. (2015). *Fast r-cnn*. Paper presented at the Proceedings of the IEEE international conference on computer vision.
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. Paper presented at the Proceedings of the IEEE conference on computer vision and pattern recognition.
- Law, H., & Deng, J. (2018). *Cornernet: Detecting objects as paired keypoints*. Paper presented at the Proceedings of the European Conference on Computer Vision (ECCV).
- Liao, J., Zou, J., Shen, A., Liu, J., & Du, X. (2021). *Cigarette end detection based on EfficientDet*. Paper presented at the Journal of Physics: Conference Series.
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). *Ssd: Single shot multibox detector*. Paper presented at the European conference on computer vision.
- Mandal, V., & Adu-Gyamfi, Y. (2020). Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis. *Journal of Big Data Analytics in Transportation*, 1-11.
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You only look once: Unified, real-time object detection*. Paper presented at the Proceedings of the IEEE conference on computer vision and pattern recognition.
- Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6), 1137-1149.
- Ripley, B. D. (2007). *Pattern recognition and neural networks*: Cambridge university press.
- Rollins, J. (2015). Why we need a methodology for data science.

- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Bernstein, M. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3), 211-252.
- Sharma, P. (2019). Image Classification vs. Object Detection vs. Image Segmentation.
- Tan, M., Pang, R., & Le, Q. V. (2020). *Efficientdet: Scalable and efficient object detection*. Paper presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.
- Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2021). A Survey of Modern Deep Learning based Object Detection Models. *arXiv preprint arXiv:2104.11892*.

## APPENDIX A

### Prototype System Code

```
# Imports Main Library
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import pathlib
import cv2
import pandas as pd
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display

# Import the object detection module.
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

# patch tf1 into `utils.ops`
utils_ops.tf = tf.compat.v1

# Patch the location of gfile
tf.gfile = tf.io.gfile

# Loader
def load_model(model_name):
    base_url = 'http://download.tensorflow.org/models/object_detection/'
    model_file = model_name + '.tar.gz'
    model_dir = tf.keras.utils.get_file(fname=model_name, origin=base_url +
model_file, untar=True)
    model_dir = pathlib.Path(model_dir)/"saved_model"
    model = tf.saved_model.load(str(model_dir))
    return model

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = "../object_detection/images/labelmap.pbtxt"
category_index =
label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)
```

```

# model_name = 'ssd_mobilenet_v1_coco_2017_11_17'
# detection_model = load_model(model_name)
detection_model =
tf.saved_model.load('./object_detection/inference_graph/saved_model')

# Check the model's input signature, it expects a batch of 3-color images of type
uint8:
print(detection_model.signatures['serving_default'].inputs)

# And returns several outputs:
detection_model.signatures['serving_default'].output_dtypes
detection_model.signatures['serving_default'].output_shapes

# -----
# Event Detector Class
class event_detector(object):
    def __init__(self):
        self.category_size = len(category_index)
        self.report = {}
        self.output = {}
        self.event = [9]*self.category_size
        self.eventlog = {}

    def run_inference_for_single_image(self, model, image, name):
        image = np.asarray(image)
        # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
        input_tensor = tf.convert_to_tensor(image)
        # The model expects a batch of images, so add an axis with `tf.newaxis`.
        input_tensor = input_tensor[tf.newaxis,...]

        # Run inference
        model_fn = model.signatures['serving_default']
        output_dict = model_fn(input_tensor)
        # print(output_dict)
        # All outputs are batches tensors.
        # Convert to numpy arrays, and take index [0] to remove the batch dimension.
        # We're only interested in the first num_detections.
        num_detections = int(output_dict.pop('num_detections'))
        output_dict = {key:value[0, :num_detections].numpy() for key,value in
output_dict.items()}
        output_dict['num_detections'] = num_detections

        # detection_classes should be ints.
        output_dict['detection_classes'] =
output_dict['detection_classes'].astype(np.int64)
        # print(output_dict['detection_classes'])
        # Handle models with masks:
        if 'detection_masks' in output_dict:
            # Reframe the the bbox mask to the image size.

```



```

        detection_masks_reframed = tf.cast(detection_masks_reframed > 0.8, tf.uint8)
        output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

    return output_dict

def get_inference_summary(self, model, image_np, threshold, name):
    output_dict = self.run_inference_for_single_image(model, image_np, name)
    n = len(output_dict['detection_scores'])
    current_event = [0]*self.category_size

    for i in range(n):
        l = []
        if output_dict['detection_scores'][i] >= threshold:
            current_event[output_dict['detection_classes'][i]-1] = current_event[output_dict['detection_classes'][i]-1] + 1
        else:
            break
    self.eventlog[name] = current_event

    if self.event != current_event:
        print("{}: {} -> {}".format(name, self.event, current_event))
        self.event = current_event
        self.output[name] = output_dict
        self.report[name] = current_event + ['NaN']

    # get main view and export recipe
    img_output_path = "./dataset/output/"
    class_detected = output_dict['detection_classes']
    class_score = output_dict['detection_scores']
    class_boxes = output_dict['detection_boxes']

    class_selected = 1
    class_location = []

    for i in range(len(class_score)):
        if class_score[i] >= threshold and class_detected[i] == class_selected:
            class_location = class_boxes[i]
            break

    cv2.imwrite(img_output_path + "" + str(name).replace(".", "_") + ".jpg",
cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB))
    try:
        img = image_np
        (im_width, im_height, _) = img.shape
        xmin, ymin, xmax, ymax = class_location

```

```

        (xmin, xmax, ymin, ymax) = (round(xmin * im_width), round(xmax *
im_width),
                                   round(ymin * im_height), round(ymax * im_height))

        extra_x = round((xmax-xmin)/1.5)
        extra_y = round((ymax-ymin)/2)

        img2 = img[(xmin-extra_x):xmin, ymin:(ymax+extra_y)]
        img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
        img2 = cv2.cvtColor(img2, cv2.COLOR_RGB2BGR)
        self.report[name] = current_event + [str(name)+"_recipe.jpg"]
        cv2.imwrite(img_output_path + "" + str(name).replace(".", "_")
+"_recipe.jpg", img2)
    except:
        print ("No Main View Detected")
        pass

    return output_dict

def get_inference_image(self, model, image_np, threshold, name):
    output_dict = self.get_inference_summary(model, image_np, threshold, name)
    final_img = vis_util.visualize_boxes_and_labels_on_image_array(image_np,
                                                                    output_dict['detection_boxes'],
                                                                    output_dict['detection_classes'],
                                                                    output_dict['detection_scores'],
                                                                    category_index,

instance_masks=output_dict.get('detection_masks_reframed', None),
                                                                    use_normalized_coordinates=True,
line_thickness=8,
                                                                    min_score_thresh=threshold)

    return(final_img)

# -----
# Event Detector Method
def run_report(video_path, start_frame, sampling_interval, end_frame, threshold):
    detector = event_detector()
    source = cv2.VideoCapture(video_path) # from video
    start_frame = start_frame
    n = sampling_interval # select sampling number, if 25 means 1s, 50 means 2s
    (based on our video format 25 frames per second)
    count = 0

    while count < end_frame:

        # extracting the frames
        ret, img = source.read()
        if img is None:

```

```

        break
    if (count % n == 0 and count >= start_frame):
        img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
        detector.get_inference_summary(detection_model, img, threshold, count/n)

    count = count + 1

return detector

def run_video(video_path, start_frame, sampling_interval, end_frame, threshold,
vid_frame_rate = 25):
    detector = event_detector()
    source = cv2.VideoCapture(video_path) # from video
    start_frame = start_frame
    n = sampling_interval # select sampling number, if 25 means 1s, 50 means 2s
    (based on our video format 25 frames per second)
    count = 0

    # text setting
    font = cv2.FONT_HERSHEY_SIMPLEX
    org = (50, 50)
    fontScale = 1
    color = (255, 0, 0)
    thickness = 2

    while count < end_frame:
        # extracting the frames
        ret, img = source.read()
        if img is None:
            break
        if (count % n == 0 and count >= start_frame):
            img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
            final_img = detector.get_inference_image(detection_model, img, threshold,
count/n)
            final_img = cv2.cvtColor(final_img,cv2.COLOR_RGB2BGR)
            final_img = cv2.putText(final_img, "{}s".format(count/vid_frame_rate), org,
font, fontScale, color, thickness, cv2.LINE_AA)
            cv2.imshow('img',final_img)

            if cv2.waitKey(1) == ord('q'):
                break

        count = count + 1

    source.release()
    cv2.destroyAllWindows()

return detector

```

```

# -----
# Run inference

# create detector object
video = "E:/CDS590/Project/WMBB_EfficientDet/research/dataset/video/t03.MTS"
obj1 = run_video(video, 0*25, 25, 99999, 0.9, vid_frame_rate = 25)

# generate event_change in dataframe
header = [category_index[i]['name'] for i in category_index] + ['recipe_img']
df = pd.DataFrame.from_dict(obj1.report, orient='index', columns=header)
df['source'] = video
df = df.reset_index().rename(columns={"index": "time_point"})
df.to_csv("./dataset/output/event_change.csv", index=False)

# generate event_log in dataframe
header2 = [category_index[i]['name'] for i in category_index]
df2 = pd.DataFrame.from_dict(obj1.eventlog, orient='index', columns=header2)
df2['source'] = video
df2 = df2.reset_index().rename(columns={"index": "time_point"})

df2.to_csv("./dataset/output/event_log.csv", index=False)

# -----
print(">>>>>>> Run event detector successfully")

```