Healthy and Unhealthy Oil Palm Tree Detection Using Deep Learning Method

Kang Hean Heng[1], Azman Ab Malik[2], Mohd Azam Bin Osman[3], Yusri Yusop[4], Irni Hamiza Hamzah[5]
School of Computer Science, Universiti Sains Malaysia, 11800 USM, Gelugor, Malaysia[1, 2, 3]
Environmental Technology, Universiti Sains Malaysia, School of Industrial Technology, 11800 USM,Gelugor, Malaysia[4]
Cawangan Pulau Pinang, Universiti Teknologi MARA, Electrical Engineering Studies, College of Engineering, 13500 Pulau Pinang, Malaysia[5]

***Abstract*—Oil palm trees are the world's most efficient and economically productive oil bearing crop. It can be processed into components needed in various products, such as beauty products and biofuel. In Malaysia, the oil palm industry contributes around 2.2% annually to the nation's GDP. The continuous surge in demand for oil palm worldwide has created an awareness among the local plantation owner to apply more monitoring standards on the trees to increase their yield. However, Malaysia's cultivation and monitoring process still mainly depends on the labor force, which caused it to be inefficient and expensive. This scenario served as a motivation for the owner to innovate the tree monitoring process through the use of computer vision techniques. This paper aims to develop an object detection model to differentiate healthy and unhealthy oil palm trees through aerial images collected through a drone on an oil palm plantation. Different pre-trained models, such as Faster R-CNN (Region-Based Convolutional Neural Network) and SSD (Single-Shot MultiBox Detector), with different backbone modules, such as ResNet, Inception, and Hourglass, are used on the images of palm leaves. A comparison will then be made to select the best model based on the AP and AR of various scales and total loss to differentiate healthy and unhealthy oil palm. Eventually, the Faster R-CNN ResNet101 FPN model performed the best among the models, with AParea = all of 0.355, ARarea = all of 0.44, and total loss of 0.2296**

Keywords*—Component oil palm detection; deep learning models; object detection; Faster R-CNN; drone imagery analysis

I. INTRODUCTION

Palm oil is the most productive oil crop and can be processed into various products, such as soap, vegetable oil, beauty products, etc. In Malaysia, palm oil has been regarded as Malaysia's golden crop. The oil palm industry has always been one of the most important agricultural exports for the nation. In 2022 alone, palm oil contributed 66.1% of the nation's total export earnings, which amounted to MYR 44.63 billion this year [1]. Besides, the demand for palm oil continues to surge due to the growing population and the shortage of sunflower and rapeseed oils in recent years. It is estimated that the annual production will be quadrupled, reaching 240m tons by 2050 [2]. To benefit from this scenario, Malaysia's oil palm industry has to be able to increase its yield. As a result, the monitoring process of oil palm trees has become important to ensure a continuous supply of palm oil.

One of the main focus or projects involves differentiating and detecting healthy and unhealthy oil palms in the plantation. This task is particularly important because insects can transmit the disease affecting an oil palm tree to its surrounding trees. Also, it is important to provide timely treatment to the infected unhealthy oil palm trees.

However, the problem for the Malaysian oil palm industry has always been labor shortages as it heavily depends on foreign labor to carry out the monitoring task. Nevertheless, the situation worsened when coronavirus hit the globe in early 2020, which caused the border between countries to close. Foreign laborers are not able to enter Malaysia to fulfill the labor shortage faced by the industry. Fortunately, this crisis has successfully prompted the industry player to innovate and implement some automation, such as object detection techniques, to overcome the problem. With object detection techniques put in place, it can help the plantation owners to shorten the time in detecting unhealthy and healthy oil palm trees, thus increasing the efficiency of the task as the traditional way of oil palm tree monitoring is too labor-intensive and inefficient. Once this project is successful, it will provide a reference for other researchers to improvise further and develop the detection model. Besides, it also acts as a starting point for the companies in this industry to utilize this method to overcome the labor shortage, further helping them increase their crop yield moving forward. Lastly, it is also believed that this project can help to share awareness and gather the attention of other companies of different crops, apart from oil palm, to utilize the object detection technique in their crop plantation and management process.

Object detection is an important computer vision task that detects instances of visual objects of a particular class (such as humans, animals, or cars) in digital images [3]. The idea and early design of object detection started in the 1900s. In recent years, the evolution of GPU architecture and deep learning techniques have generalized the usage of object detection techniques in many sectors. Autonomous driving, defect detection, and traffic monitoring are real-world applications that use object detection models. There are two types of object detection frameworks, which are region-based, and regression/classification based. Region-based frameworks are commonly referred to as two-stage detectors, while regression/classification-based frameworks are referred to as one-stage detectors.

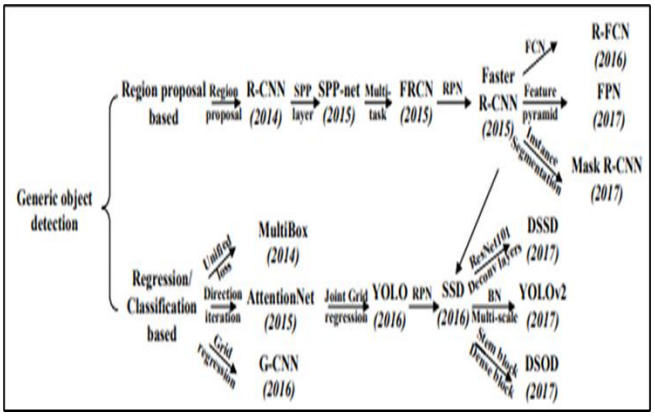

Fig. 1. Development of object detection frameworks [4].

Two-stage detectors will first generate region-based proposals, then classify each proposal into different classes. The typical examples of two-stage detectors are R-CNN, SPP-Net, Faster R-CNN, and FPN. Meanwhile, one stage detectors view object detection as a regression or classification problem, adopting a unified framework to achieve final results (categories and locations) directly [4]. Single-shot MultiBox Detector (SSD), You Only Live Once (YOLO), and CenterNet are some of the models that use regression/classification-based frameworks. Fig. 1 summarizes the development of two object detection frameworks.

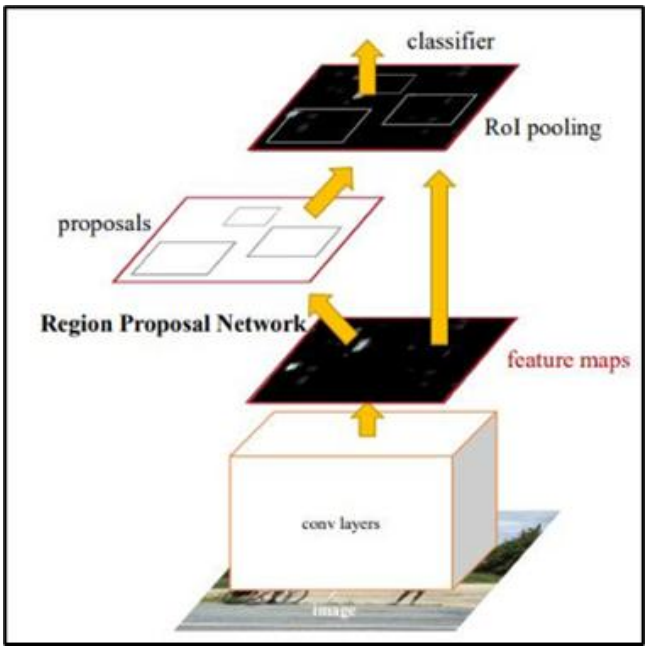

Fig. 2. RPN module [5].

Faster R-CNN is the abbreviation of Faster Region-based Convolutional Neural Network. It combines the algorithm of RPN (Region Proposal Network) and Fast R-CNN as shown in Fig. 2. It can be viewed as an updated version of Fast R-CNN, where, RPN replaces the Selective Search method. RPN is a fully convolutional network, which slides over the convolutional feature map and simultaneously predicts object boundary and object-ness scores at each position [4]. As a result, the model performs better in speed and accuracy and consumes fewer computational resources. In a paper published in [6], the authors applied Faster R-CNN on high resolution imagery for automatic detection and health classification of oil palm trees. There are two backbones selected for Faster R-CNN, which are ResNet-50 and VGG 16. The model with ResNet-50 as the backbone obtained F1 scores of 95.09%, 92.07%, and 86.96%, respectively, for oil palm tree detection, healthy tree identification, and unhealthy tree identification. Overall, the ResNet-50 model yielded a better F1 score than the VGG-16 model.

RetinaNet is a one-stage object detection model first published in a paper by Lin et al. The development of RetinaNet is aimed at overcoming the problem of class imbalance and robust estimation in Single-Shot MultiBox Detector (SSD) during training by utilizing a focal loss function. Class imbalance problem in SSD leads to many easy negatives appearing when the detector is evaluating the object locations, overwhelming the loss function and causing a degeneration in the model performance. With focal loss function, it helps to down-weight those easy negatives and thus focus training on hard negatives as shown in Fig. 3.

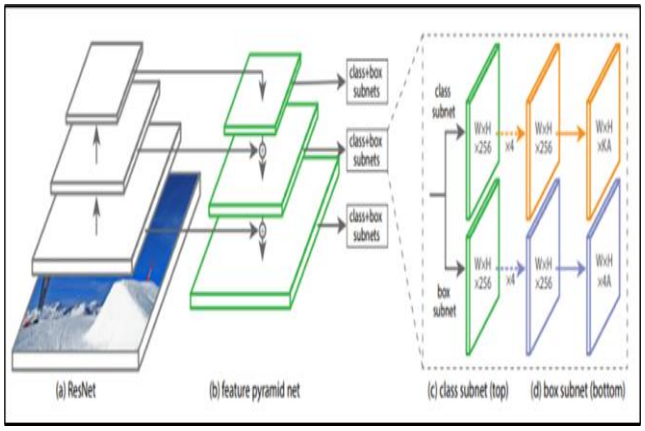

Fig. 3. RetinaNet architecture [7].

An optimized palm tree inventory model based on a RetinaNet object detector and high-resolution RGB images is proposed in 2021 to classify and locate palm trees in different scenes with different appearances and ages [8]. The dataset has a spatial resolution of 25cm. The detection model for palm tree inventory achieved a precision of 89.3% on the validation dataset and 76.9% on the test dataset. Both precision values are on the mAP@IoU = 0.50 category. CenterNet is an anchorless object detection model published [9], entitled "Objects as Points" in 2019. The idea of anchorless in this model is to replace Non-Maximum Suppression (NMS) algorithm used in SSD or YOLO. NMS algorithm functions as a filter to select one single bounding box out of the many overlapping bounding boxes in the post process. For example, YOLOv3 generates more than 7k bounding boxes in its prediction for each image, mostly considered garbage predictions, and the NMS algorithm will need to run pairwise checks for those overlapping bounding boxes [10]. Fig. 4 shows a CenterNet algorithm. This method increases the complexity of the model when the number of bounding boxes (predictions) increases. It also forces the model to consume more computational power and time in clearing those irrelevant predictions.

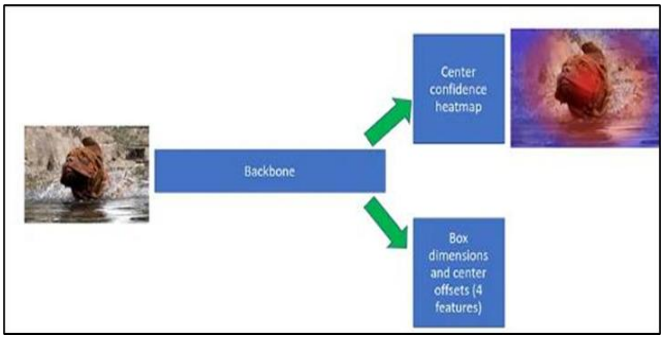

Fig. 4. CenterNet algorithm [10].

Meanwhile, CenterNet predicts a box center for an object and uses the center point to regress the value for its box dimensions and offsets, directly removing irrelevant predictions without any decoding process. An anchor-free deep learning model, CenterNet, is used to detect individual crown locations and regions from dense 3D terrestrial laser scans [11]. There are 1181 crowns from twelve plots. The author used eight plots for training, and four plots for testing. The model is trained over 40k iterations. The maximum training F1-score and IoU were 0.881 and 0.670, while testing result showed a F1-score of 0.754 and IoU of 0.583. The result also showed that a taller, larger, smoother, less crowded, and less overlapped tree was found easier to be detected by the model. Table I shows a comparison of object detection models for two stage and one stage of model. ResNet, Inception, and HourGlass Network are all convolutional neural networks widely used for image classification tasks. However, each contains its design specification to optimize the deep neural network as it goes deeper.

TABLE I. COMPARISON OF OBJECT DETECTION MODELS

Model		Innovations	Strength	Limitations
Two Stage	Faster R CNN	The RPN module helps to generate high-quality region proposals and saves time compared to the Selective Search method.	Higher accuracy	Complicated training with high memory consumption
One Stage	RetinaNet	Focal loss function – down weight the easy negatives and focus on hard negatives	Stable training for class imbalance	Lower accuracy compared to a two-stage detector

Before ResNet was invented, researchers tended to design deep learning networks by increasing their layers and depth as shown in Fig. 5. However, it comes to a limitation where the train and test errors increase when it goes even deeper. In a paper published by [12], it is believed that this degradation in performance is not caused by overfitting, and indicated that not all networks could be optimized easily by making it deeper. As a result, a deep residual learning framework has been proposed by [12] to address the degradation problem. ResNet learns residual functions with reference to the layer inputs instead of learning unreferenced functions. Thus, it makes it possible to train a much deeper network while minimizing the error value.

Next, Inception Network as shown in Fig. 6 is designed to decrease the computational cost and burden to run a deep neural network. Besides, it also helps to better optimize parallel computing. The network performs max pooling and a convolution on an input with three different sizes of filters (1x1, 3x3, and 5x5), rather than stacking them in sequence. As the network progress, it will grow wider and not deeper. This also help to reduce vanishing gradient problem [14]. Meanwhile, HourGlass Network consists of multiple stacked hourglass modules, which allow for repeated bottom-up, top-down inference [15]. It is specially designed for predicting human poses. The advantage of this network is that it captures and consolidates information across all scales of the image [15].

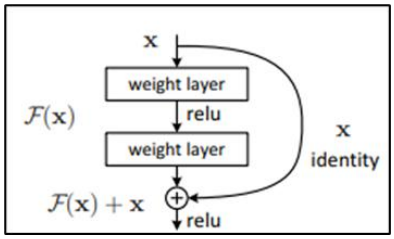

Fig. 5. Residual learning [12].

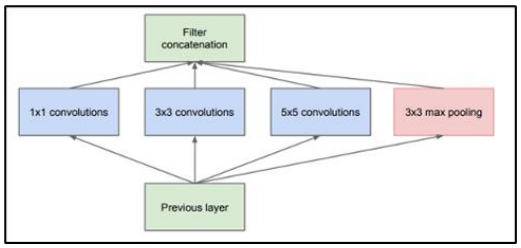

Fig. 6. Inception module [13].

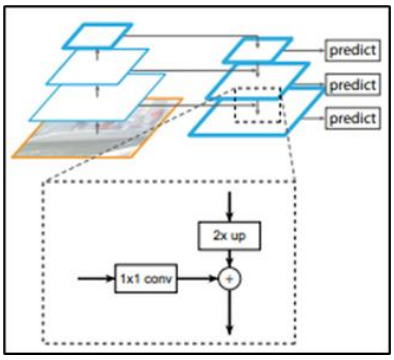

Fig. 7. Building block of FPN [16].

The Feature Pyramid Network is published by Lin et al. in 2017. FPN can be seen as an updated version of ConvNet's pyramidal feature hierarchy. FPN takes a single-scale image of an arbitrary size as input, and outputs proportionally sized feature maps at multiple levels, in a fully convolutional fashion. This process is independent of the backbone convolutional architecture [16]. As shown in Fig. 7, the building block involves a bottom-up pathway, a top-down pathway, and lateral connections. The result from the paper showed that average precision of a Faster R-CNN ResNet50 with FPN, goes up by 7.6%, from 26.3% to 33.9%. The evaluation of the model is made on COCO minival set.

A standard implementation, which can be used as a benchmark among different datasets is published in 2020. One of the famous metrics used for model evaluation is called average precision (AP) and average recall (AR). Before going deeper into how metrics works, it is also wise to understand the concept of intersection over union (IOU). It is a measurement based on Jaccard Index, a coefficient of similarity for two sets of data [17]. For object detection model, IOU measures the overlapped area between two bounding boxes, one from prediction, and one of ground-truth, divided by the area of union between them. With a given threshold, if the IOU is bigger than the threshold, then the detection is considered valid and vice versa. With IOU in place, AP and AR can be evaluated in different variations as mentioned in Table II.

TABLE II. PERFORMANCE METRICS [17]

Metrics	Meaning
AP@50:5:95	AP of 10 IOUs varying from range of 50% to 95% with steps of 5%.
AP50	AP of IOU of 50%
AP75	AP of IOU of 75%
AP-S	AP for small sized objects (area < 322 pixels)
AP-M	AP for medium sized objects (322 < area < 962 pixels)
AP-L	AP for large sized objects (area > 962 pixels)
AR@50:5:95	AR of 10 IOUs varying from range of 50% to 95% with steps of 5%.
AR-S	AR for small sized objects (area < 322 pixels)
AR-M	AR for medium sized objects (322 < area < 962 pixels)
AR-L	AR for large sized objects (area > 962 pixels)

The loss function is a common but essential metric in deep learning. It provides a numerical representation of how well a model performs on a particular task. It helps the researchers to evaluate and fine-tune the model to achieve a better result. If the model predicts poorly, the loss function will output a higher number and vice versa. In object detection, the loss function can be categorized into two parts; one is classification loss, another one is localization loss. The former is applied to train the classification head for determining the target object type, and the latter is used to train another head for regressing a rectangular box to locate the target object [18]. Once these two categories adds on, it is called a total loss for the object detection model.

II. RESEARCH METHOD

A standard object detection workflow has been presented to create a deep learning-based object detector in a paper published in [19] and the workflow as shown in Fig. 8. The workflow starts with the dataset acquisition process, followed by data annotation. The images will then be augmented into different sizes and split into training and test sets according to a selected ratio. Next, the training and test set will be fitted and trained in an object detection algorithm. Once the training is done, the best model will be saved. Finally, the best model will be used to infer the test dataset or new images for evaluation purposes. The evaluation process will be based on the selected metrics, such as mAP, AP, and AR. If the model performs well, it can be deployed into a real-case scenario.

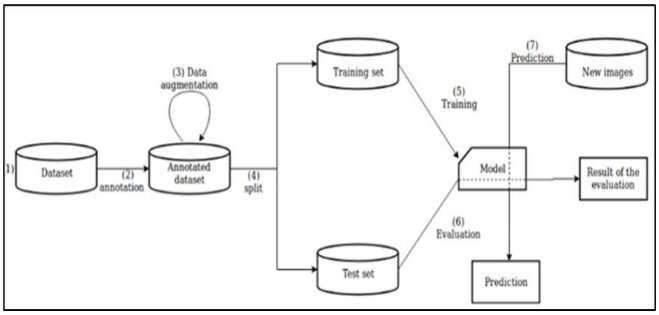

Fig. 8. Workflow for object detection project [19].

Fig. 9 illustrates the overall lifecycle of the object detection project, and it will only cover performance evaluation. The model's deployment for real-case scenarios will depend on the client's needs. The cycle consists of 6 stages: problem understanding, data collection, data annotation and augmentation, data transformation, modelling, and model evaluation. During the early stage of the problem understanding, one discussion session was held with the client, and the problem faced was identified. The problem is due to the client's lack of domain knowledge to develop a proper object detection model. Next, several meetings were held to discuss the desired outcome and expectations from the client. The goal was then set: to develop a prototype model which can make a good prediction on healthy and unhealthy oil palm trees on aerial images captured by drone.

The next stage in the project lifecycle is data collection. It is also known as data acquisition in the object detection workflow. The client provides the image dataset for this project. It consists of 90 images, which are collected from an oil palm plantation in Perak, based on the coordination stored in the metadata of the image. The images are collected during sunny and cloudy weather with different aerial angles. Multiple trees can be seen in an image with tree crowns overlapping each other. Some are unhealthy trees as shown in Fig. 10, and some are healthy trees as shown on Fig. 11. The health of the oil palm trees can be differentiated using crown color. Most healthy trees have green crowns, while the unhealthy ones tend to have yellow and brown crowns [20]. All the images are captured through a drone and stored in 32-bit PNG format. The image size ranges from 8 MB to 12 MB, while the image size is fixed at 2982-pixel x 1694-pixel. All the images have a resolution of 144 dots per inch (DPI).

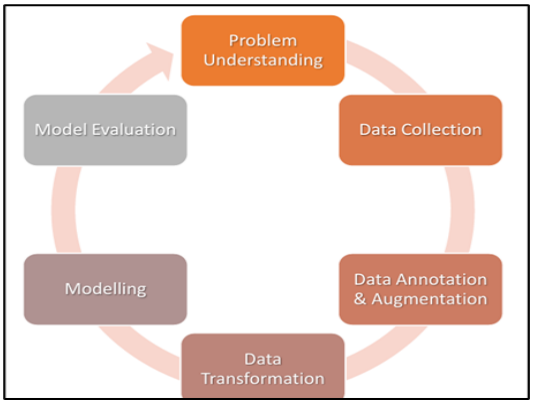

Fig. 9. Project lifecycle.

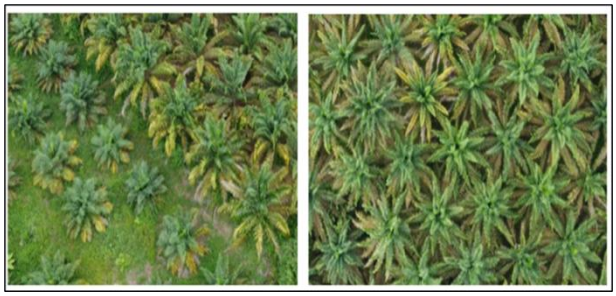

Fig. 10. Tree crown of unhealthy oil palm tree.

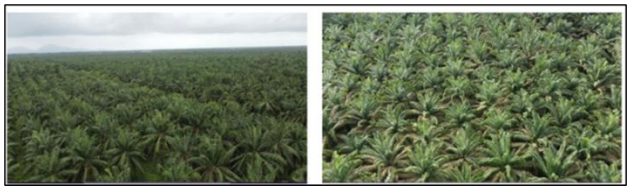

Fig. 11. Tree crown of healthy oil palm tree.

Next, the images were manually annotated using an open-source tool called 'Labellmg' as shown in Fig. 12. The software is written in Python format and uses Qt for its graphical interface [21]. The bounding box was drawn on the visible objects. Each bounding box represents the object's coordinate in the image with four axis: xmin, xmax, ymin, and ymax. Two classes, 'Healthy' and 'Unhealthy', are used to annotate each object in the images. The annotations are saved in Pascal VOC XML format.

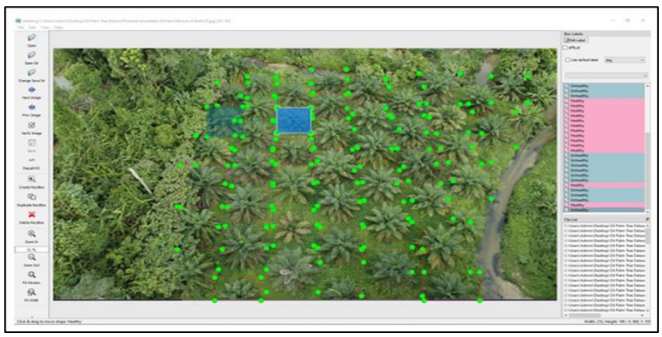

Fig. 12. LabelImg.

The images were once trained using one of the selected object detection models. However, the image size is too big for the GPU to handle. The GPU on Google Colab is the Nvidia Tesla T4, which has 16GB of memory. As a result, the images must be rescaled and reduced in size. The images were augmented or rescaled to 1024 x 1024 pixels. The bounding boxes also shrank according to the rescale ratio using a python script found in a GitHub repository. Italo José codes the script [22].

The image dataset is then split randomly into training and validation sets with a ratio of 8:2 as shown in Fig. 13. Once the splitting process is complete, the respective annotations are split into two folders, training and validation set accordingly. The dataset will need to train in different models. Some models are stored in different model zoos of different platforms, such as TensorFlow, PyTorch, and Detectron2. The model zoo is a repository where companies or open-source institutions store machine learning and deep learning models. Object detection models, stored in TensorFlow and PyTorch model zoo, are developed by Google and Linux Foundation (previously Meta AI). Those models are trained on the PASCAL VOC dataset, and the annotation needs to be in XML format. Meanwhile, models in the Detectron2 platform are maintained by Meta AI, and the models are trained on the COCO dataset. Thus, the annotation needs to be in JSON format. As a result, the annotations in XML format are converted into JSON format using a script found in the GitHub repository. A person develops the script with a pseudonym called fam_taro [23]. Meanwhile, a script developed by Dat Tran in GitHub is being used to generate TFRecord format for TensorFlow object detection models [24]. The Fig. 14 shows the dataset directory structure for the project.

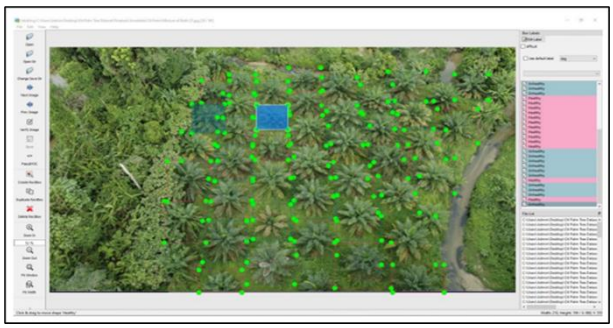

Fig. 13. Original size image vs resized image.

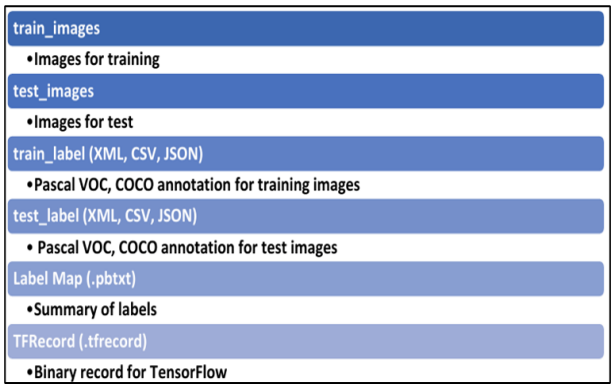

Fig. 14. Dataset directory structure.

TABLE III. MODEL SPECIFICATION

Specification	Faster R-CNN		RetinaNet		CenterNet
Model Zoo	TensorFlow	Detectron2	TensorFlow	Detectron2	TensorFlow
Annotation Format	XML	JSON	XML,	JSON	XML
Backbone	Inception-ResNet	ResNet-50/101	ResNet-50/101		HourGlass
FPN	No	Yes	Yes		No
Anchor	Yes		Yes		No
Input Image Size	640 x 640		640 x 640		512 x 512

Several models are being selected for training. The justification of the advantages and disadvantages of each model is written in previous discussion. Table III summarizes the specification of the selected models.

Besides, several settings need to be configured in the config file of each model before the training process starts. Table IV summarizes the basic settings in the config file.

TABLE IV. CONFIG SETTINGS

Settings	Options
num_classes	2/3*
Path to train_images/ test_images/ labelmap	Path to the working directory in Google Drive
fine_tune_checkpoint	Checkpoint of pre-trained model downloaded from the model zoo of the respective platform
fine_tune_checkpoint_type	Detection
batch_size	2/4
num_steps/iter/epoch	50000/3600/100

Many pre-trained object detection models are available in the model zoo of different platforms. However, every platform requires a library version to be installed before the model can be trained on the custom dataset. In addition, the TensorFlow library, Colab, and Drive are all developed by Google. As a beginner, it is reasonable to consider models from the TensorFlow library as the first choice due to compatibility issues. Unfortunately, TensorFlow models are computationally expensive to train and consume much time. A Faster R-CNN model took up to 9 hours to train. Due to cost concerns, other alternatives, such as PyTorch's TorchVision library and Detectron2 library, are being considered. The models in PyTorch take much lesser time and are easier to train. PyTorch's Faster R-CNN model only took an hour to train on the same model. However, only a limited variety of models are available to train on the platform, which is the main disadvantage of the PyTorch platform.

For Detectron2 library, it contains a variety of models that are available for training. Most models are trained with the 3x schedule (~37 COCO epochs). A Faster R-CNN model only takes an hour to train on the same dataset. It has much more model variation and consumes less time in training. As a result, the Detectron2 library is selected for the rest of the training process to produce a standardized result.

Object detection has recently been deemed feasible due to the rapid development of GPU parallel computing. GPU parallel computing allows the model to split the training process into thousands of tasks and process it simultaneously. It is much more efficient and timesaving than the CPU, which only processes one task simultaneously. Unfortunately, there is no GPU hardware available on the laptop. Specification of project setup as shown in Table V. One of the alternatives available online is the Cloud GPU Platform. Some examples of these platforms are Google Colaboratory, Paperspace, Microsoft Azure, and Kaggle.

In order to stay competitive, every platform offers similar GPU devices and only differs by the subscription plan. As a result, it is really up to user preferences and usage. For this project, Google Colaboratory has been selected due to several reasons. The user interface is also not much different from Jupyter Notebook. Lastly, it allows the users to link to their Google Cloud storage, keeping the whole process on the cloud. Thus, it is easier and more flexible for beginners and students.

TABLE V. SPECIFICATION OF PROJECT SETUP

Types	Specification
CPU	Nvidia Tesla T4 (16GB memory)
Storage	Google Drive (100GB)
IDE	Google Colab (Python)
RAM/ Disk Space	12.7 GB/ 107.7 GB

III. RESULTS AND DISCUSSION

The models selected are CenterNet HourGlass104 512x512 and Faster R-CNN Inception ResNet V2 640x640. The basic settings of the model as shown in Table VI and Table VII, the results, and the inference images as shown in Fig. 15 and Fig. 16.

TABLE VI. BASIC SETTINGS FOR TENSORFLOW MODELS

Settings	Options
num_classes	2
fine_tune_checkpoint_type	Detection
batch_size	2
num_steps	50000

TABLE VII. RESULT FOR CENTERNET & FASTER R-CNN

Model	CenterNet HourGlass104 512x512	Faster R-CNN Inception ResNet V2 640x640
Average Precision (AP) @ [IoU-0.50:0.95 | area= all]	0.372	0.351
Average Precision (AP) @ [IoU=0.50:0.95 | area= small]	-1	-1
Average Precision (AP) @ [IoU=0.50:0.95 | area= medium]	0.194	0.157
Average Precision (AP) @ [IoU=0.50:0.95 | area= large]	0.408	0.405
Average Recall (AR) @ [IoU=0.50:0.95 | area= all]	0.542	0.475
Average Recall (AR) @ [IoU=0.50:0.95 | area= small]	-1	-1
Average Recall (AR) @ [IoU=0.50:0.95 | area= medium]	0.366	0 275
Average Recall (AR) @ [IoU=0.50:0.95 | area= large]	0.576	0.516
Total Loss	5.196	1.788

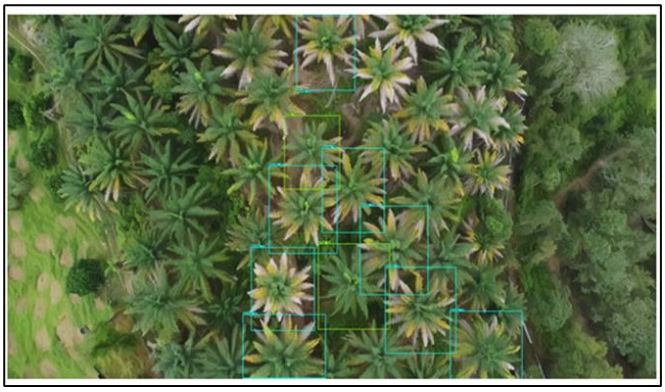

Fig. 15. Inference image of centernet hourglass104.

Fig. 16. Inference image of faster R-CNN inception ResNet V2.

In this evaluation, the inference image is a new test image with an image size of 3840 x 2160 pixels. In the image, most of the oil palm trees are considered medium or large objects, as the area of the big tree crown is above 962 pixels, and the medium one is around 322 and 962 pixels. In Table VI, the evaluation metrics shows that CenterNet with HourGlass104 backbone achieved a higher average precision (AP) for all, medium and large area categories, compared to Faster R-CNN with Inception ResNet backbone. A higher value of AP indicates that when the algorithm detects an object as healthy or unhealthy, it has a high possibility that it is correct. Furthermore, the metrics also show that the average recall rate (AR) of all, medium, and large area categories are higher in the CenterNet model than in the Faster R-CNN model. It means that the CenterNet model has fewer false negative predictions. Meanwhile, there are no small tree crowns under 322 pixels in the image that can be detected. Thus, the metrics show a value of 1 in both AP and AR for the small area category. Lastly, CenterNet achieved 5.196 in total loss, while the Faster R-CNN model only achieved 1.788. This problem can be seen in the inference image as well. Many tree crowns are left undetected in CenterNet's inference image, compared to the one in the Faster R-CNN model. This scenario also means that the CenterNet failed to predict many tree crowns, which means fewer false negative instances are being predicted (higher AR). However, those predicted ones are highly likely to be correct (higher AP). Thus, the metrics proved that the CenterNet model is not robust enough to locate the tree crowns and has a high localization loss compared to Faster R- CNN. In conclusion, both models did not produce a satisfactory result and took a very long time to train around 5 hours. As a result, a decision has been made to stop the training process for other TensorFlow models. The models in the Dectectron2 platform will be used as an alternative.

Four pre-trained models are being selected from the Detectron2 Model Zoo. The models selected are RetinaNet ResNet-50, RetinaNet ResNet-101 FPN, Faster R-CNN ResNet-50 FPN, and Faster R-CNN ResNet-101 FPN. The basic settings of the model as shown in Table VIII and Table IX, the result, and the inference images as shown in Fig. 17, Fig. 18 and Fig. 19.

TABLE VIII. BASIC SETTINGS FOR DETECTRON2 MODELS

Settings	Options
num_classes	2
inns_per_batch (batch_size)	2
max_iter	3600

TABLE IX. RESULT FOR DETECTRON2 MODELS

Model	RetinaNet ResNet50 FPN	RetinaNet ResNet101 FPN	Faster R CNN ResNet50 FPN	Faster R CNN ResNet101 FPN
Average Precision (AP) @ [loU-0.50:0.95 | area= all]	0.23	0217	0.317	0.353
Average Precision (AP) @ [loU-0.50:0.95 | area= small]	-1	-1	-1	-1
Average Precision (AP) @ [loU-0.50:0.95 | area= medium]	0.050	0.071	0.167	0.145
Average Precision (AP) @ [loU-0.50:0.95 | area= large]	0.263	0.284	0.388	0.4
Average Recall (AR) @ [loU-0.50:0.95 | area= all]	0.261	0.284	0.422	0.425
Average Recall (AR) @ [loU-0.50:0.95 | area= small]	-1	-1	-1	-1
Average Recall (AR) @ [loU-0.50:0.95 | area= medium]	0.062	0.08	0.223	0.196
Average Recall (AR) @ [loU-0.50:0.95 | area= large]	0.3	0.324	0.466	0.472
Total Loss	0.3329	0.2768	0.559	0.4414

Fig. 17. Inference image of retinaNet ResNet50 FPN.

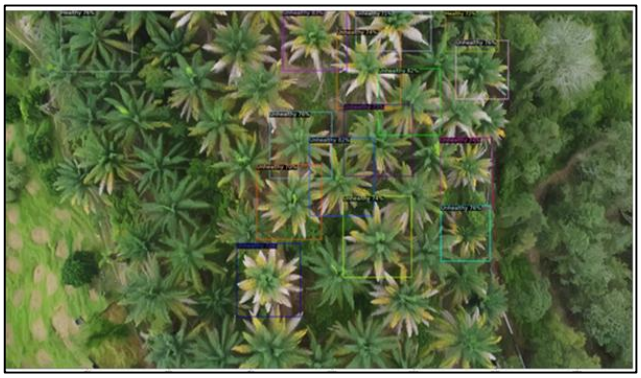

Fig. 18. Inference image of faster R-CNN ResNet50 FPN.

Fig. 19. Inference image of faster R-CNN ResNet101 FPN.

The batch size and the number of max iterations have been selected for the fine tuning process. These two parameters are essential and will significantly impact the result if the value is fine-tuned correctly. Batch size refers to the number of training images utilized in one iteration, while max_iter refers to the maximum cycle for the training process. The settings of the model as shown in Table X and Table XI, the result and the inference images as shown in Fig. 20, Fig. 21, Fig. 22, Fig. 23 and Fig. 24.

TABLE X. FINE-TUNING FOR FASTER R-CNN RESNET101 FPN

Settings	Options
num_classes	2
ims_per_batch (batch_size)	2/ 4
max_iter	3600/ 7200/ 9000/ 10800

TABLE XI. RESULT UNDER DIFFERENT SETTINGS

Settings	Benchmark	A: 3600 iter, batch size: 4	B: 7200 iter, batch size: 2	C: 9000 iter, batch size: 2	D: 10800 iter, batch size: 2
Average Precision (AP) @ [IoU-0.50:0.95 | area= all]	0.353	0.329	0.342	0.355	0.333
Average Precision (AP) @ [IoU-0.50:0.95 | area= small]	-1	-1	-1	-1	-1
Average Precision (AP) @ [IoU-0.50:0.95 | area= medium]	0.145	0.158	0.165	0.169	0.184
Average Precision (AP) @ [IoU-0.50:0.95 | area= large]	0.4	0.364	0.382	0.393	0.362
Average Recall (AR) @ [IoU-0.50:0.95 | area= all]	0.425	0.393	0.418	0.440	0.421
Average Recall (AR) @ [IoU-0.50:0.95 | area= small]	-1	-1	-1	-1	-1
Average Recall (AR) @ [IoU-0.50:0.95 | area= medium]	0.196	0.206	0.212	0.222	0.259
Average Recall (AR) @ [IoU-0.50:0.95 | area= large]	0.472	0.430	0.463	0.487	0.452
Total Loss	0.4414	0.3803	0.2853	0.2296	0.2020

Fig. 20. Benchmark from preliminary result.

Fig. 21. Batch size of 4 and 3600 iterations.

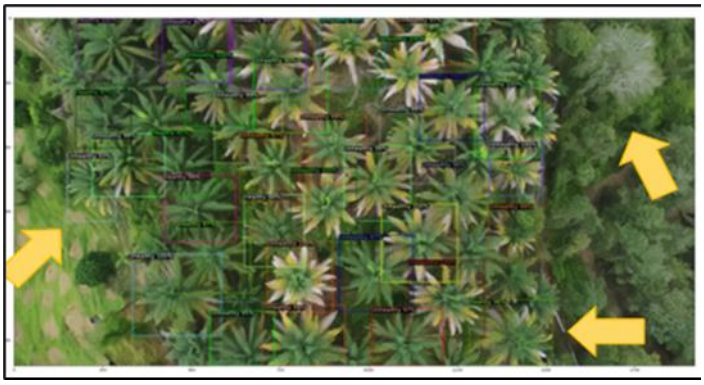

Fig. 22. Batch size of 2 and 7200 iterations.

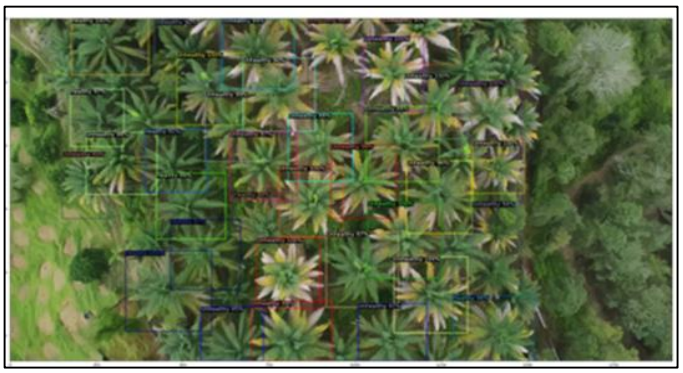

Fig. 23. Batch size of 2 and 9000 iterations.

Fig. 24. Batch size of 2 and 10800 iterations.

The fine-tuning process starts with model A by increasing the batch size to 4 while maintaining the max_iter value at 3600. These settings are made to explore the impact on model performance by increasing the batch size. The process is then continued with another three models of B, C, and D with different iterations, which are 7200, 9000, and 10800, while the batch size is kept constant at 2. The reason is to understand the effect of different numbers of max iteration on the model performance. Table X shows the performance metrics of a benchmark model from the preliminary stage and four other fine-tuned models. The model did not perform significantly better by increasing the batch size to 4. The performance metrics show that model A has a lower total loss and a slight increase in value for AP and AR in the medium area category, while the other metrics of model A dropped when compared with the benchmark model. The inference image also did not show any significant improvement in detection, as there is still a false positive detection on the top right corner of the image. Next, model B has settings of 7200 iterations and batch size of 2. The table shows that there is only a slight increase in value for AP and AR in the medium area category and a lower value of total loss (0.2853). Besides that, it did not show any significant increase in other performance metrics. However, there is an improvement in model B when comparing the inference image with the one of the benchmark model. It is shown with an arrow at the area of improvement. The false positive detection on the top right corner of the image is gone, and the undetected trees at the left side and the bottom part of the image are now detected. For model C, the max iteration is increased to 9000 iterations, and the batch size is kept constant at 2. The result shows that model C achieved the highest AP value of 0.355 for all area category, and the highest AR value of 0.44 and 0.487 for all and large area categories, respectively. Meanwhile, the total loss of model C is 0.2296, which is the second lowest among the models. The inference image of model C has no differences when compared to the one of model B. Lastly, the max iteration is set to 10800, and the batch size is kept constant at 2 for model D. Even though model D achieved the lowest value of total loss and the highest value in AP and AR for the medium area category, the value for other performance metrics dropped compared to model C of 9000 iterations. The value of AP for all and large area categories hits the lowest among all models. The inference image of model D also started to show the symptoms of overfitting as the false positive detection reappeared in the top right corner of the image. In conclusion, the Faster R-CNN ResNet101 FPN model with 9000 iterations and batch size of 4 performs the best. This experiment also shows that a two-stage detector, like the Faster R-CNN model, is better at classifying and locating the object than a one-stage detector, like RetinaNet and CenterNet. In addition, it also proved that a deeper backbone module would yield a better result.

There are some challenges throughout the project execution. The data annotation process was a challenging one. The oil palm trees are tough to label as it contains much ambiguity. Some tree crowns are yellowish because of sunlight reflection, but it does not mean the tree is unhealthy. Besides, oil palm tree crowns overlap, making it hard to determine the correct boundary for each crowns during annotation. Thus, constant communication is made with the clients on this issue so that a standardized dataset can be produced. Lastly, another challenge is the time consumption in training the TensorFlow models. The longest time to train a TensorFlow is around nine hours for the Faster R- CNN model. It is also tough to train the model as the training process disconnected from the server several times, and it will be retrained again. Thus, object detection models from alternative platforms like PyTorch and Detectron2 are being used and the training time significantly reduces to less than two hours.

IV. CONCLUSION

As mentioned in Section III, the Faster R-CNN ResNet101 FPN model performed the best among all the models. It is then fine-tuned to achieve a better result. Batch size and max iteration are the parameters used to fine tune the model. As a result, the Faster R-CNN ResNet101 FPN model with 9000 iterations and batch size of 2 achieved the best performance compared to its

benchmark model with 3600 iterations and batch size of 2. Suggestion for future work, to explore the method of taking a picture by combine using multi modal fusion and sensor. The data might be useful by combining sensors such as lidar, hyperspectral, and SAR (Synthetic Aperture Radar) with RGB imagery to improve detection under varying the environment from different perspectives.

REFERENCES

[1] A. Azuar, "Malaysia's palm oil, products exports up 55.2% for 6M22," The Malaysian Reserve, Aug. 9, 2022. [Online]. Available: https://themalaysianreserve.com/2022/08/09/malaysias-palm-oil-products-exports-up-55-2-for-6m22/ Accessed: Dec. 27, 2024.

[2] P. Tullis, "How the world got hooked on palm oil," The Guardian, Feb. 19, 2019. [Online]. Available: https://www.theguardian.com/news/2019/feb/19/palm-oil-ingredient-biscuits-shampoo-environmental Accessed: Dec. 27, 2024

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," arXiv preprint, arXiv:1905.05055, pp. 1–22, May 2019. doi: 10.48550/arXiv.1905.05055

[4] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," IEEE Trans. Neural Netw. Learn. Syst., vol. 30, pp. 3212–3232, Dec. 2019, doi: 10.1109/TNNLS.2018.2876865.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," arXiv preprint, arXiv:1506.01497v3, pp. 1–14, Jan. 2016. doi: 10.48550/arXiv.1506.01497

[6] K. Yarak, A. Witayangkurn, K. Kritiyutanont, C. Arunplod, and R. Shibasaki, "Oil palm tree detection and health classification on high-resolution imagery using deep learning," Agriculture, vol. 183, 2021.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," arXiv preprint, arXiv:1708.02002v2, pp. 1–10, Jan. 2018. doi: 10.48550/arXiv.1708.02002

[8] M. Culman, S. Delalieux, and K. Van Tricht, "Palm tree inventory from aerial images using RetinaNet," in Proc. 2020 Mediterranean and Middle-East Geoscience and Remote Sensing Symp. (M2GARSS), Tunis, 2020, pp. 314–317, doi: 10.1109/M2GARSS.2020.317.

[9] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," arXiv preprint, arXiv:1904.07850, 2019. doi: 10.48550/arXiv.1904.07850

[10] U. Almog, "CenterNet, explained," Towards Data Science, Apr. 10, 2021. [Online]. Available: https://towardsdatascience.com/a7386f368962. Accessed: Dec. 27, 2024.

[11] Z. Xi and C. Hopkinson, "Detecting individual-tree crown regions from terrestrial laser scans with an anchor-free deep learning model," Can. J. Remote Sens., vol. 46, pp. 228–242, 2020.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv preprint, arXiv:1512.03385v1, pp. 1–12, Dec. 2015. doi: 10.48550/arXiv.1512.03385

[13] C. Szegedy et al., "Going deeper with convolutions," arXiv preprint, arXiv:1409.4842v1, pp. 1–12, Sep. 2014. doi: 10.48550/arXiv.1409.4842

[14] DeepAI, "Inception module," DeepAI, Jan. 5, 2023. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/inception-module. Accessed: Dec. 27, 2024.

[15] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," arXiv preprint, arXiv:1603.06937v2, pp. 1–17, Apr. 2016. doi: 10.48550/arXiv.1603.06937

[16] T.-Y. Lin et al., "Feature pyramid networks for object detection," arXiv preprint, arXiv:1612.03144v2, pp. 1–10, Mar. 2017. doi: 10.48550/arXiv.1612.03144

[17] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in Proc. 2020 Int. Conf. Syst., Signals and Image Process. (IWSSIP), Niterói, 2020, pp. 237–242, doi: 10.1109/IWSSIP.2020.347.

[18] S. Jiang, H. Qin, B. Zhang, and J. Zheng, "Optimized loss functions for object detection: A case study on nighttime vehicles," arXiv preprint, arXiv:2011.05523v2, pp. 1–15, Nov. 2020. doi: 10.48550/arXiv.2011.05523

[19] A. Casado and J. Heras, "Guiding the creation of deep learning-based object detectors," arXiv preprint, arXiv:1809.03322v1, pp. 1–6, Sep. 2018. doi: 10.48550/arXiv.1809.03322

[20] K. Yarak, A. Witayangkurn, K. Kritiyutanont, C. Arunplod, and R. Shibasaki, "Oil palm tree detection and health classification on high-resolution imagery using deep learning," Agriculture, vol. 183, 2021.

[21] L. Studio, "labelImg," GitHub, Jan. 5, 2023. [Online]. Available: https://github.com/heartexlabs/labelImg. Accessed: Dec. 27, 2024.

[22] I. Jose, "Resize_dataset_pascalvoc," GitHub, Jan. 1, 2023. [Online]. Available: https://github.com/italojs/resize_dataset_pascalvoc. Accessed: Dec. 27, 2024.

[23] fam_taro, "voc2coco," GitHub, Jan. 1, 2023. [Online]. Available: https://github.com/yukkyo/voc2coco. Accessed: Dec. 27, 2024.

[24] D. Tran, "generate_tfrecord.py," GitHub, Jan. 1, 2023. [Online]. Available: https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py. Accessed: Dec. 27, 2024.