OVERHEAD VIEW BASED PERSON COUNTING USING DEEP LEARNING

KAW CHEE ZHAO

UNIVERSITI TUNKU ABDUL RAHMAN

OVERHEAD VIEW BASED PERSON COUNTING USING DEEP LEARNING

KAW CHEE ZHAO

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2022

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	Zhao
Name	:	Kaw Chee Zhao
ID No.	:	1704522
Date	:	15 / 5 / 2022

APPROVAL FOR SUBMISSION

I certify that this project report entitled **“OVERHEAD VIEW BASED PERSON COUNTING USING DEEP LEARNING”** was prepared by **KAW CHEE ZHAO** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Ir Ts Dr. Tham Mau Luen
Date	:	15 / 5 / 2022

ACKNOWLEDGEMENTS

I would like to thank everyone who contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr Tham Mau Luen for his invaluable advice, guidance and his enormous patience throughout the development of the research.

ABSTRACT

Detecting people in an image or a video has become more prevalent due to the rapid advancement of technologies in the field of artificial intelligence. In conventional video surveillance systems, most of the person detection methods are based on frontal view, which may have lower accuracy stemming from the occlusion problem. This project proposes an overhead view based person counting system by enabling wider scene coverage and visibility. The entire project methodology can be divided into several phases. First, the YOLOv4 and YOLOv4-tiny object detection models are trained with the dataset of overhead camera perspective. Second, the OpenVINO Inference Engine is utilized to optimize the trained models in order to facilitate real-time implementation. Third, the accurate tracking of each detected person is performed using the deep learning based tracking framework, known as DeepSORT. Lastly, the performance of the proposed system is benchmarked based on the detection accuracy, frames per second (FPS) and counting accuracy. Based on the results obtained, the YOLOv4-tiny model is chosen as it can achieve high fps without the need of high processing power. Besides, the Centroid Tracking algorithm achieves around 38.4% to 40.4% higher fps as compared to that of the DeepSORT tracking algorithm. However, the counting accuracy of Centroid Tracking algorithm is about 22.2% lower than the DeepSORT tracking algorithm. Hence, the overall performance of the YOLOv4-tiny model integrated with DeepSORT algorithm outperforms the other tracking algorithms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AI	artificial intelligence
CNN	convolutional neural network
CT	Centroid Tracking
DS	DeepSORT
FPS	frame per second
HOG	Histogram of Oriented Gradients
KNN	K-Nearest Neighbour
LOI	line-of-interest
mAP	Mean Average Precision
R-CNN	Region-based Convolutional Neural Network
RHOG	Rotated-Histogram of Oriented Gradients
ROI	region-of-interest
SIFT	Scale-Invariant Feature Transform
SORT	Simple Online and Real-time Tracking
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine
YOLO	You Only Look Once

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In this era of digitalization, a neural network with deep learning has become more essential as it makes things easier. Most companies rely on deep learning algorithms to meet their consumer expectations. Deep learning is a type of machine learning and also a subset of artificial intelligence (AI). Machine learning enables computers to perform tasks without explicit programming whereas deep learning focuses on allowing computers to think using structure modelled on the human brain and perform complex tasks. The comparison between AI, machine learning as well as deep learning is shown in Figure 1.1.

Today, there are a variety of object detection frameworks, two of which are machine learning as well as deep learning. Object detection consists of two computer vision tasks which include classifying the image and localizing the object in the image. For image classification, prediction of the class of an object within an image is done. Meanwhile, for object localization, the objects within an image are located and their locations are indicated with bounding boxes. As such, object detection combines both the tasks which localize the objects with bounding boxes and then classify them into a list of categories. In a person counting system, tracking people is also a form of object detection, in which the target objects are people.

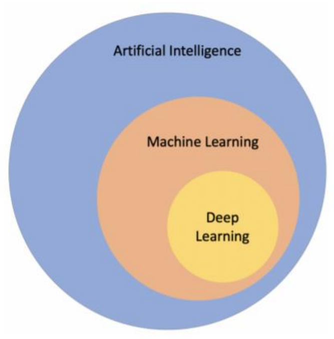


Figure 1.1: AI vs. Machine Learning vs. Deep Learning (Miraftabzadeh et al., 2019)

In recent years, automatically detecting people in images has gained importance in deep learning because of the wide variety of its applications, for example, prevention of criminal activities, behaviour analysis, as well as person counting and tracking. Different researchers have put in lots of effort so that person detection can be done with much higher accuracy. In conventional or frontal view based person detection, occlusion problems may occur when the tracked person is hidden by another person during the real-time people tracking. Hence, an overhead view based person detection is often preferred as it provides better scene coverage and visibility (Ahmad et al., 2019). The overhead view gives an elevated view of objects from above which allows people detection with a completely different perspective as compared to the frontal view based technique.

1.2 Importance of Study

Deep learning is a subfield of AI which imitates how the human brain works in processing data and decision making. Today, there are a lot of applications that are involved in deep learning such as computer vision, voice recognition, automated driving and medical research. With deep learning, complex tasks like object classification, and data prediction can be done by computers effortlessly.

The deep learning algorithm is often used for detecting and tracking objects due to its excellent detection results. In video surveillance system, person detection and tracking system using the deep learning algorithm has a wide variety of applications such as gait recognition, crime prevention, behaviour analysis of people and so on. These applications bring great benefits to society. For example, burglars or thieves can be tracked down more easily. Besides, the person counting system can also be used to ensure social distancing practices during the Covid-19 pandemic.

1.3 Problem Statement

Currently, there are a lot of libraries like Caffe and TensorFlow which support object detection. The detection methods based on deep learning such as Single Shot MultiBox Detector (SSD), You Only Look Once (YOLO), as well as Faster Region-based Convolutional Neural Network (Faster R-CNN) can achieve high accuracy and high efficiency. However, their ability to track the object detected is still far from that of human beings mainly because of occlusion which occurs when the tracked object is hidden or overlapped by another object. During real-time object tracking, the objects could be lost due to miss calculations if occlusion occurs. Another problem that shows up due to the occlusion problem is that the identities of the tracked object might be switched frequently during the tracking process. These problems will greatly deteriorate the performance of the object tracking model.

Thus, this project aims to find a more efficient real-time detection and tracking solution by using the overhead view based person detection, instead of the conventional frontal view based person detection to solve the problems arised due to occlusion.

1.4 Aim and Objectives

This project aims to use the deep learning method to detect, track and count the person based on the overhead view in a certain area in real-time. The objectives of this study include:

- To detect people from the overhead view
- To track the detected person and count
- To evaluate the performance of different tracking algorithms

1.5 Scope and Limitation of Study

The scope of this study includes designing a people tracking algorithm based on the overhead view using the deep learning detection method and then integrating it with a counting system.

The limitation of the study includes the efficiency of the prototype which depends on the processing power. The processing power of laptop may be insufficient to handle some unusual situations. For example, when there are many people appearing at the same time, the achievable frames per second (fps) may drop. Other than that, there are difficulties in using the model analysis namely OpenVINO Deep Learning Workbench, as it is a toolkit that was released in 2018. The source of information and support about the DL Workbench may be limited and hence, more efforts need to be put in to solve the problem.

1.6 Contribution of Study

This project contributes to the field of person counting based on overhead view. The deep neural networks and tracking algorithms were investigated in this project. The performance of the prototype was evaluated based on various criteria such as the total inference time, fps, and counting accuracy. This prototype is expected to be deployed in practical in the future.

1.7 Outline of the Report

This report consists of the following chapters:

- Introduction
- Literature Review
- Methodology
- Results and Discussion
- Conclusion and Recommendations

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction to Person Counting System

In video surveillance, there are two different perspectives for person detection which are frontal and overhead. Although overhead view based person detection can prevent occlusion and provide better scene coverage, it is a challenging task due to the following factors: different person body appearance, variation in poses, complex background, and uncontrolled lighting conditions (Ahmad et al., 2019). Over the last few years, a variety of top view based person detection algorithms have been developed by many researchers and these techniques can be classified into several groups which include the blob based algorithm, the machine learning based algorithm, as well as the deep learning based techniques.

A real-time object tracking algorithm is also needed for tracking and counting the detected person. There are several popular object tracking algorithms that most researchers use which include the Centroid Tracking algorithm and the Deep SORT algorithm. For the counting system, Zhao et al. (2016) stated that two existing counting systems are mostly used in two scenarios, namely counting based region-of-interest (ROI) and crossing based line-of-interest (LOI).

2.2 Blob based Algorithm

In images, a blob can be generalized as a bunch of pixel values that create a sort of colony or a huge object which is distinct from its background. These blobs can then be identified through image processing. Ahmad et al. (2019) stated that in blob based person detection algorithm, the background subtraction method is used to obtain a foreground image. However, several pre-processing techniques are performed before the background subtraction so that the noise, illumination and shadow can be removed. Then, the blob is extracted from the foreground image to be classified as one single person or another object according to its shape, motion, colour, and other feature. The overall framework of blob based algorithms is shown in Figure 2.1.

There are some basic blob features for person detection which include the blob shape, hair colour, texture and body size. Nakatani et al. (2012) stated that hair colour and hairstyle are usually different between persons. Thus, when extracting the blob feature, the total brightness of each pixel in the head area is taken into consideration. Other than that, the location of hair whorl is also one of the significant features for the identification of a person's head. An example of the extracted hair whorl feature is shown in Figure 2.2.

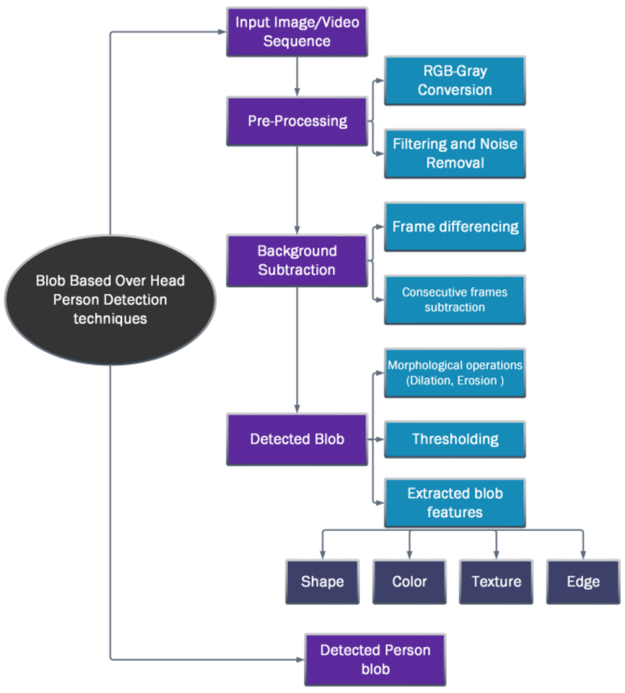

Input Image/Video Sequence
Pre-Processing
RGB-Gray Conversion
Filtering and Noise Removal
Background Subtraction
Frame differencing
Consecutive frames subtraction
Blob Based Over Head Person Detection techniques
Morphological operations (Dilation, Erosion)
Detected Blob
Thresholding
Extracted blob features
Shape
Color
Texture
Edge
Detected Person blob

Figure 2.1: Overall Framework of Blob Based Algorithms (Ahmad et al., 2019)

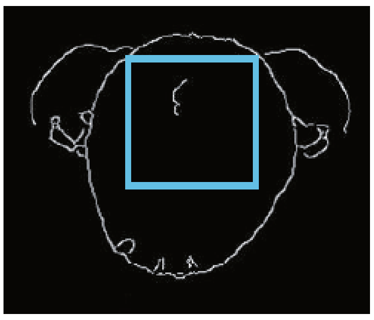


Figure 2.2: Extracted Hair Whorl Feature (Nakatani et al., 2012)

2.3 Machine Learning based Algorithm

Due to the advancement in computer vision and machine learning nowadays, various machine learning based person detection algorithms have gained popularity thanks to their high detection accuracy. For feature-based person detection, features like shape, colour, texture, direction, motion and so on are extracted from images. The images are usually split into samples for training and testing purposes. Then, machine learning classifiers, such as the AdaBoost, K-Nearest Neighbour (KNN), Support Vector Machine (SVM), and so on are used to classify those samples as person or non-person images (Ahmad et al., 2019).

There are two popular machine learning based algorithms which include the Histogram of Oriented Gradients (HOG) as well as the Scale-Invariant Feature Transform (SIFT). The overall framework of machine learning based algorithms is shown in Figure 2.3.

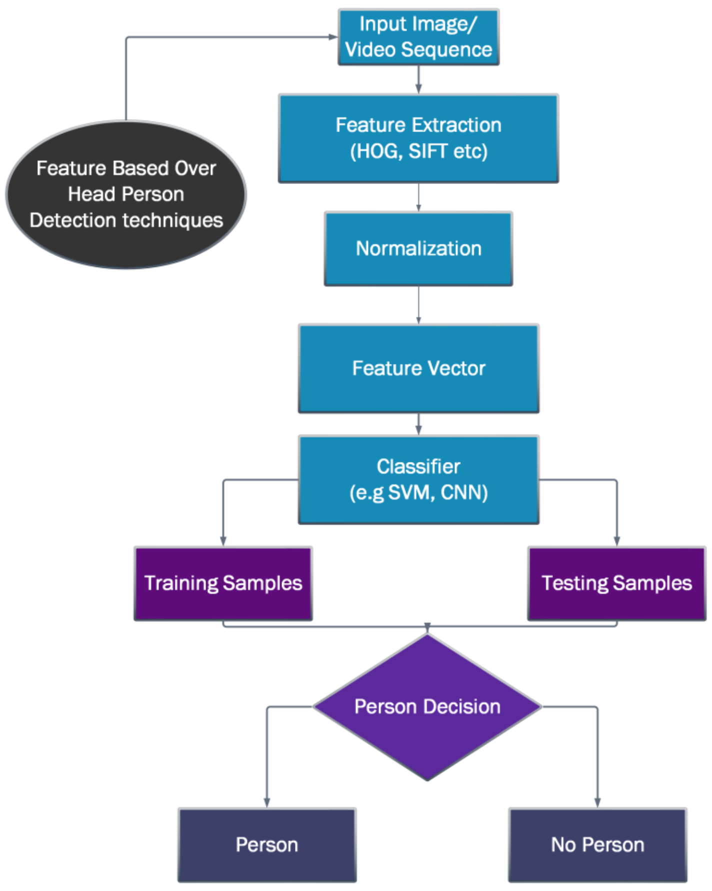

Input Image/ Video Sequence
Feature Based Over Head Person Detection techniques
Feature Extraction (HOG, SIFT etc)
Normalization
Feature Vector
Classifier (e.g SVM, CNN)
Training Samples
Testing Samples
Person Decision
Person
No Person

Figure 2.3: Overall Framework of Machine Learning Based Algorithms (Ahmad et al., 2019)

2.3.1 Histogram of Oriented Gradients (HOG)

HOG works as a feature descriptor to extract features from an image. According to Dalal and Triggs (2005), the shape and appearance of an object in a picture can be defined by the intensity gradients distribution or edge directions. The HOG algorithm works by first dividing the image into small squared cells, and creating a histogram of gradient directions for the pixel within each cell. After that, the result is normalized using a block-wise pattern and a descriptor is returned for every cell. In other words, the HOG method counts the occurrence of gradient orientation in localized segments of an image. A machine learning classifier such as SVM can be used to stack the cells into a region of squared images to be used as an image window descriptor for object detection. The process of forming the HOG descriptor is shown in Figure 2.4.

Ahmed et al. (2017) proposed an efficient Rotated-Histogram of Oriented Gradients (RHOG) method for people detection based on top view images. In the RHOG algorithm, bounding boxes of variable sizes with different orientations are used. Hence, different geometric transformations are needed to orient the bounding box based on the person’s orientation in the image. This is done to improve the overall detection rate. According to Ahmed et al. (2017), the RHOG algorithm with a detection rate of 95% performed far better than the standard HOG algorithm which only obtain a detection rate of 59%.

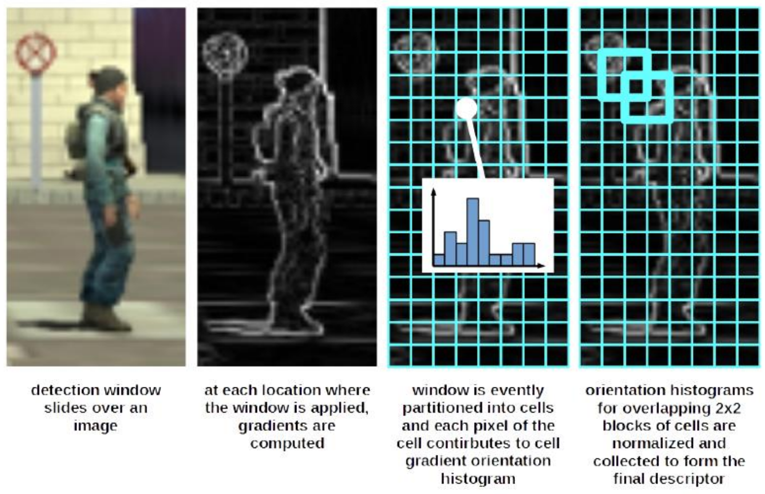

detection window slides over an image
at each location where the window is applied, gradients are computed
window is evently partitioned into cells and each pixel of the cell contirbutes to cell gradient orientation histogram
orientation histograms for overlapping 2x2 blocks of cells are normalized and collected to form the final descriptor

Figure 2.4: Process of forming HOG descriptor (Olejniczak and Kraft, 2017)

2.3.2 Scale-Invariant Feature Transform (SIFT)

According to Lowe (1999), SIFT algorithm is used to transform an image into local feature vectors for the extraction of image features. These feature vectors are invariant, which are unaffected by any rotation, translation or scaling of the image.

According to Lowe (2004), the SIFT algorithm can be divided into four steps which include detecting the feature point, localizing the feature point, assigning the orientation, as well as generating the feature descriptor. The first stage which is the feature point detection identifies the potential interest points which are invariant to scale and orientation. The next stage which is the keypoint localization locates the feature keypoints accurately according to the measures of their stability. Next, in the third stage which is the assignment of orientation, one or more orientations are assigned to each location of the keypoint according to the local image gradient directions. The last stage which is the keypoint descriptor describes the key points as a high dimensional vector which is called the SIFT key. For object detection, the SIFT key is used in a nearest-neighbour algorithm so that the objects within the image can be identified.

Ozturk et al. (2009) proposed an optical flow of SIFT algorithm to observe the orientation change in the body and head. Although the proposed algorithm works very well for one person detection, but when there are a lot of people in the scene at the same time, improvements are needed to obtain a better system performance. Overall, the proposed SIFT algorithm still gives quite promising results.

2.4 Deep Learning based Algorithm

Recently, Deep Learning based techniques that are often based on convolutional neural networks (CNN) can perform object detection without specifying the features. Unlike machine learning approach which requires the features to be defined through various methods before the image classification, deep learning approach can perform the entire detection process without the need to specify the features for image classification.

2.4.1 Convolutional Neural Network (CNN)

CNN is a feed-forward artificial neural network that is commonly used to provide accurate performance in computer vision tasks (Krizhevsky et al., 2012). CNN is often used to process 2D matrix of pixels such as images for image classification and object detection. In comparison to the traditional neural network, the CNN has deeper layers and its neurons are arranged in a volumetric way such as height, width, and depth. It is composed of three-layer types which include a convolutional layer, a sub-sampling layer, as well a fully connected layer. The convolution layer and the sub-sampling layer are connected alternatively in the middle section of the network while the fully connected layer is the last layer of the CNNs. The CNN architecture is shown in Figure 2.5.

The convolution layers apply filters for the extraction of features from the input image. Meanwhile, the function of the sub-sampling layers is to downsample or reduce the spatial dimensions such as width and height. For the fully connected layer, the probability distributions over the number of output classes is computed by applying the softmax activation function (Galvez et al., 2018).

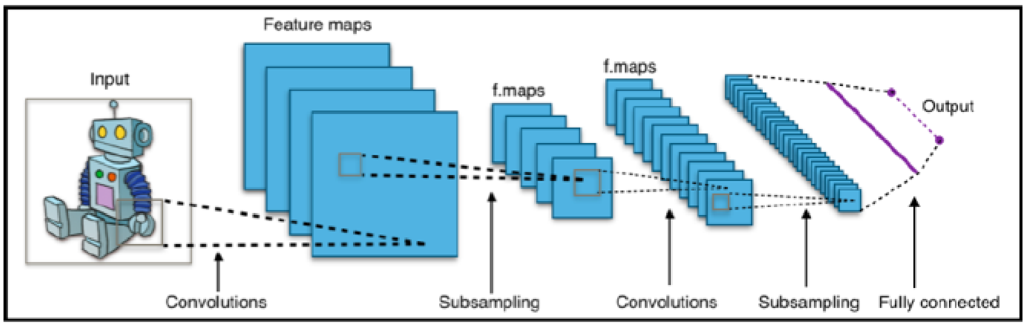


Figure 2.5: CNN Architecture (Gulli and Pal, 2017)

2.4.2 Region-based Convolutional Network (R-CNN)

R-CNN which is also known as “Regions with CNN Features” was introduced by Girshick et al. in 2014. According to Girshick et al. (2014), Convolutional Neural Networks are overly slow and relatively expensive. Hence, R-CNN tackles these problems by adopting a technique which is known as Selective Search. The Selective Search approach replaces the Exhaustive search approach which looks for objects in thousands of windows although the image is small in size. The Selective Search method searches for the object through different size windows. For each size, the adjacent pixels are grouped by colour, texture or intensity for object identification.

According to Girshick et al. (2014), the R-CNN model contains three modules. The first module is ‘Region Proposal’ which generates and extracts category-independent region proposals. In other words, it defines the number of bounding-box candidate detections by using Selective Search. The second module is ‘Feature Extraction’ which extracts features from every candidate region through the use of a large CNN. The third module is ‘Classification’ which classifies the extracted features by using linear SVM classifiers. The R-CNN model architecture is shown in Figure 2.6.

After the great success of R-CNN, the Fast Region-based Convolutional Network (Fast R-CNN) which is significantly faster in terms of training and making predictions was proposed by Girshick in 2015. In the following year, Ren et al. (2016) further improved the speed of training and detection as well as the detection accuracy by proposing the Faster R-CNN model.

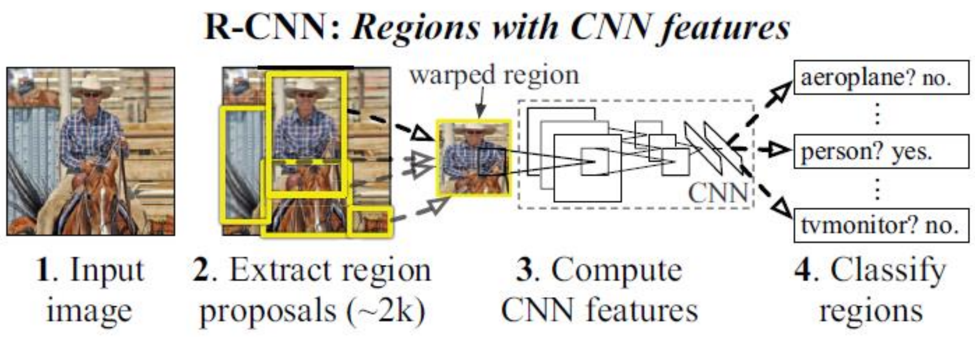


Figure 2.6: R-CNN Architecture (Girshick et al., 2014)

2.4.3 You Only Look Once (YOLO)

The YOLO model proposed by Redmon et al. (2016) can predict bounding box and class probability directly with a single neural network. Even though the YOLO model has more localization errors and lower prediction accuracy, but it can operate faster which is at 45 fps and up to 155 fps for a smaller network.

The workflow of the YOLO model is first dividing the input image into an S × S grid. Whenever an object's centre is within a cell of the grid, then the cell will be responsible for the detection of the object. The bounding box and its confidence score are also predicted by the grid cell. The confidence score shows the degree whereby the object is certain to be inside the predicted bounding box. The grid cell also predicts the conditional class probabilities. Then, the bounding boxes with confidence scores are integrated with the class probability map to produce a final set of bounding boxes and class labels (Redmon et al., 2016). The general workflow of the YOLO model is shown in Figure 2.7.

Over the years, the YOLO model was further improved with its versions named YOLOv2, YOLOv3 and YOLOv4, whereby the YOLOv4 model is the updated YOLO model with improved performance in terms of detection speed and accuracy. The YOLOv3 and YOLOv4 models use Darknet53 for the extraction of features. This makes the models to be more accurate as compared to the YOLOv2 model which uses Darknet19 with only 19 convolutional layers as the feature extractor.

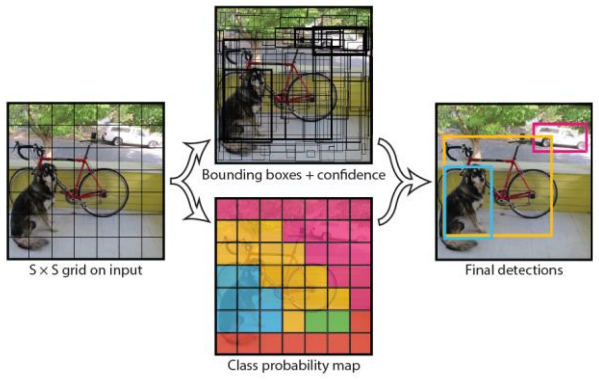


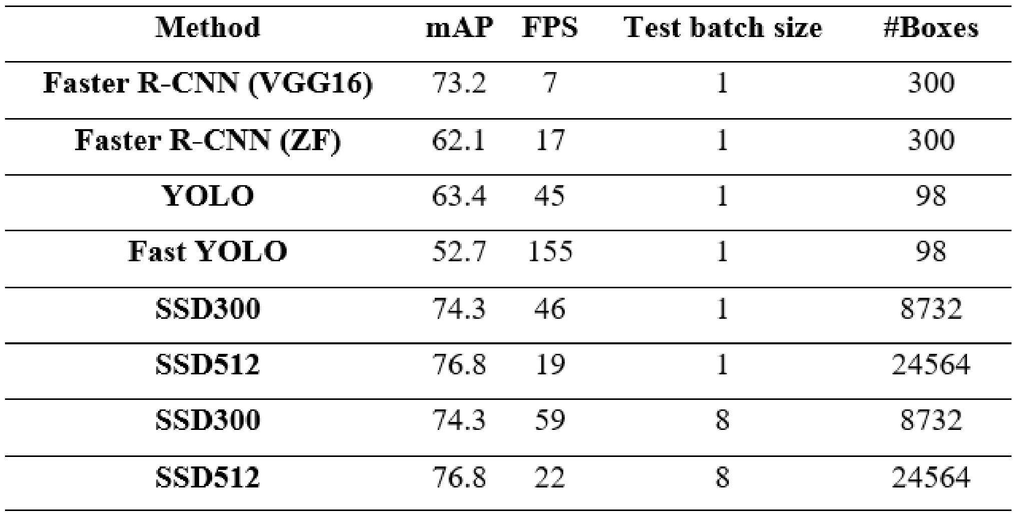
Figure 2.7: General Workflow of YOLO model (Redmon et al., 2016)

2.4.4 Single Shot MultiBox Detector (SSD)

Liu et al. (2016) proposed a single shot detection model which is known as Single Shot MultiBox Detector (SSD) and stated that it is faster and significantly more accurate than the previous YOLOv2 model. The comparison of the performance of several different detection models is shown in Table 2.1. Unlike YOLO, the SSD model does not have a constant aspect ratio for grid cells. Instead, it uses a distinct aspect ratio with multi boxes to achieve higher detection accuracy.

The SSD object detection works by first extracting the feature maps, and then applying convolution filters for object detection. First, an image with ground truth boxes for every object is inputted into the SSD. A small set of default boxes with different aspect ratios is evaluated in a convolutional fashion in some feature maps with different scales such as 8 × 8 and 4 × 4 feature maps as shown in Figure 2.8 (b) and (c) respectively. Then, the confidence scores and the shape offsets for all categories of objects are predicted for each default box. The SSD framework is shown in Figure 2.8.

Table 2.1: Comparison between Different Detection Models (Liu et al., 2016)



Method	mAP	FPS	Test batch size	#Boxes
Faster R-CNN (VGG16)	73.2	7	1	300
Faster R-CNN (ZF)	62.1	17	1	300
YOLO	63.4	45	1	98
Fast YOLO	52.7	155	1	98
SSD300	74.3	46	1	8732
SSD512	76.8	19	1	24564
SSD300	74.3	59	8	8732
SSD512	76.8	22	8	24564

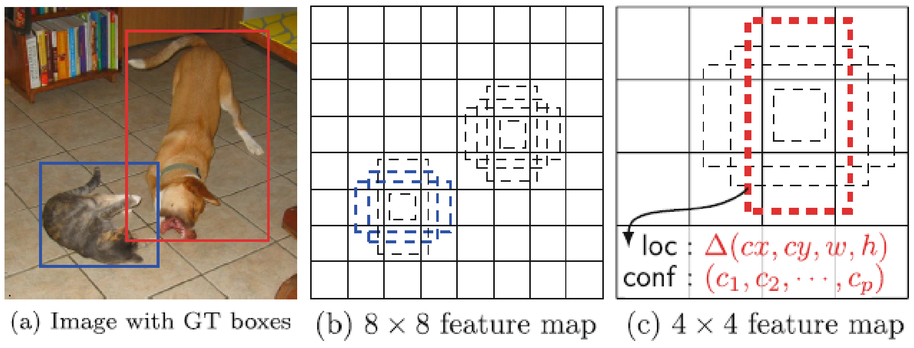


Figure 2.8: Framework of SSD (Liu et al., 2016)

2.5 Review of Real-Time Object Tracking Algorithm

After the object detection, a tracking system is needed to track the object for counting purposes. The overall process of object tracking begins with taking an initial set of bounding box coordinates, then generating a unique ID for each of the bounding boxes, and lastly tracking the objects while maintaining their unique IDs. There are several popular object tracking algorithms that most researchers use which include the Centroid Tracking algorithm and the DeepSORT algorithm.

2.5.1 Centroid Tracking Algorithm

After detecting the person, the information of the classified bounding box is utilized for counting the total number of people within the image. For every classified bounding box detected, it will be assigned a count number or a unique ID. Then, the bounding box coordinates are determined and the centroids are calculated by using the diagonal points of the bounding box as shown in Figure 2.9. The equation for the calculation of the centroid is shown in equation (2.1) below:

$$x_C = \frac{x_1 + x_2}{2}, y_C = \frac{y_1 + y_2}{2} \tag{2.1}$$

where

x_1, y_1 and x_2, y_2 represent the diagonal points of the bounding box

x_C and y_C represent the centroid of the bounding box

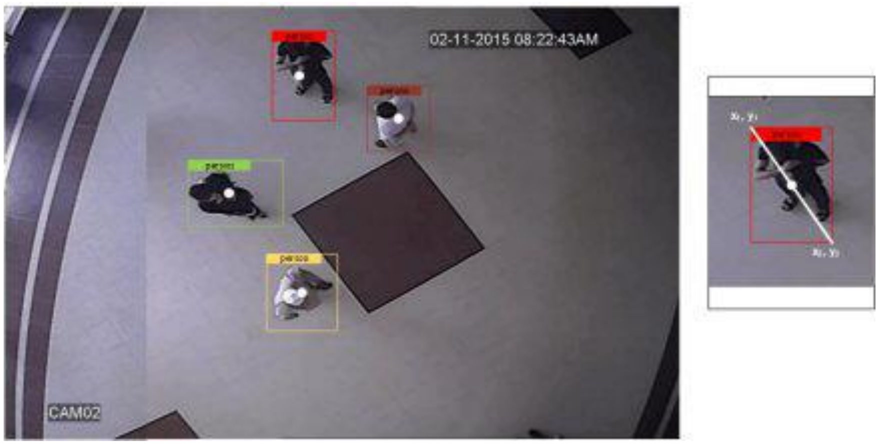


Figure 2.9: Locating Centroids for each Classified Bounding Box (Ahmad et al., 2019)

For object tracking, the centroids are calculated for every subsequent frame in the video stream. However, rather than assigning a new ID to the detected object, the Euclidean distances are computed between existing objects and the new bounding boxes detected. Then, the x and y coordinates for the existing objects are updated while the new object detected is assigned with a unique ID. The tracking process continues until the object leaves the field of view or disappears, the object will be deregistered from the system.

2.5.2 DeepSORT Algorithm

Wojke et al. (2017) stated that Simple Online and Real-Time Tracking with a Deep Association Metric (DeepSORT) is one of the most robust and fastest object tracking algorithms. It was developed based on the Simple Online and Real-time Tracking (SORT) approach which focuses on associating object detections on each frame (Bewley et al., 2016). The SORT approach makes use of the CNNs for more accurate object detection. It also implements two traditional methods in motion prediction and data association, namely the Hungarian algorithm and the Kalman filter as its tracking components.

Bewley et al. (2016) stated that SORT is over twenty times faster than other state-of-the-art online tracking approaches. However, according to Wojke et al. (2017), the major drawback of SORT is that the tracked object identities might be switched frequently during the tracking process due to the occlusions. In order to solve this issue, DeepSORT integrates a deep appearance-based metric derived from the convolutional network (CNN), rather than using only the motion-based metrics in data association. A large person re-identification dataset is used for training the CNN applied in DeepSORT to discriminate the pedestrians. This results in increasing the system robustness against occlusions, as well as reducing the number of identity switches by approximately 45% (Wojke et al., 2017). The architecture of the DeepSORT algorithm is shown in Figure 2.10.

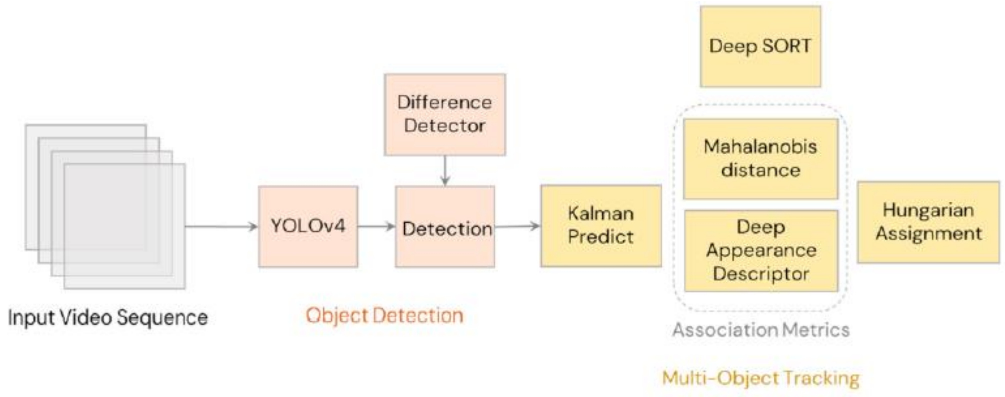


Figure 2.10: DeepSORT Architecture (Parico and Ahamed, 2021)

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, both the hardware and software required to design an overhead view based person counting system using deep learning are introduced.

This project is divided into several parts as shown in Figure 3.1. Firstly, the dataset required to train the YOLOv4 and YOLOv4-tiny object detection models which include the images of people based on overhead view and the images of human faces were obtained from online sources. Then, all the dataset was annotated in YOLO format by using LabelImg. Next, the Darknet was installed to train both the YOLOv4 and YOLOv4-tiny models. After that, the OpenVINO Inference Engine was utilized to optimize the performance of the models. Tracking algorithms which include Centroid Tracking and DeepSORT tracking algorithms were implemented into the code to track and count the person detected by the detection model. Lastly, the performance of the prototype was evaluated based on the total inference time, fps as well as counting accuracy.

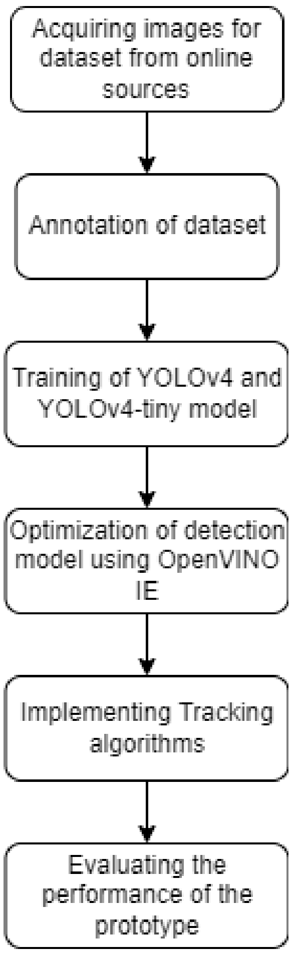

Acquiring images for dataset from online sources
Annotation of dataset
Training of YOLOv4 and YOLOv4-tiny model
Optimization of detection model using OpenVINO IE
Implementing Tracking algorithms
Evaluating the performance of the prototype

Figure 3.1: Workflow of the Project

3.2 Overview of System Requirement

In this project, the hardware required include a laptop and a USB stick. Meanwhile, the required software include Rufus, Ubuntu 18.04 LTS, OpenCV, TensorFlow 2.0, Darknet, and OpenVINO toolkit.

3.2.1 Ubuntu 18.04 LTS

The first step to start the project is by installing the Linux operating system on the laptop. The Ubuntu 18.04 LTS which is one of the popular Linux distributions is chosen. The latest version of Ubuntu 21.04 is not preferred to avoid software incompatibility issues.

Since the laptop comes preinstalled with Windows 10 Home, the Ubuntu 18.04 LTS operating system is installed along with (dual booting) Windows 10. The steps to dual boot the Ubuntu 18.04 LTS along with Windows 10 is shown in Figure 3.2. Firstly, the disk partition for Ubuntu installation is created in the Windows menu. Around 200 GB of free space is allocated to Ubuntu. Then, the ISO image file for Ubuntu 18.04 LTS is downloaded from the official Ubuntu website. After that, the USB is plugged in and a bootable Ubuntu USB is created by using the Rufus software. Once the live Ubuntu USB is ready, the next step is to boot from it and then the installation procedure of Ubuntu can be started. When the installation process is complete, the laptop is restarted and the laptop can finally be booted into either Ubuntu or Windows through the GRUB boot options.

After installing the Ubuntu 18.04 LTS operating system, the necessary software and packages are ready to be installed on the machine.

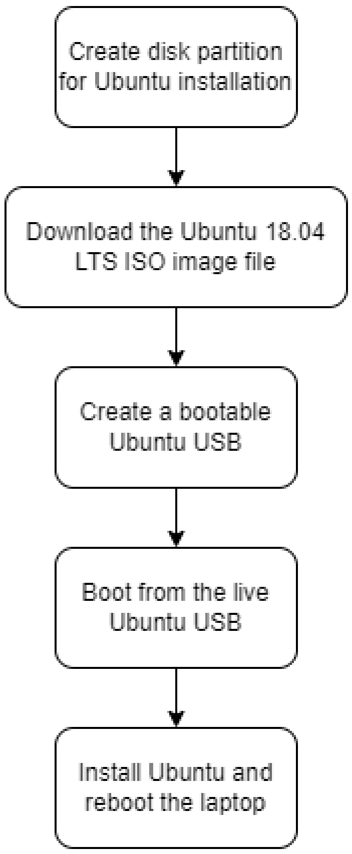

Create disk partition for Ubuntu installation
Download the Ubuntu 18.04 LTS ISO image file
Create a bootable Ubuntu USB
Boot from the live Ubuntu USB
Install Ubuntu and reboot the laptop

Figure 3.2: Dual Booting Ubuntu 18.04 LTS along with Windows 10

3.2.2 OpenCV

OpenCV is a machine learning library which consists of functions for real-time computer vision tasks like image processing, frame capturing and so on. The codes for installing OpenCV on Ubuntu 18.04 LTS are attached in Appendix A. The steps to install OpenCV on Ubuntu 18.04 LTS are as follows:

1. First, the required OpenCV dependencies are installed.
2. Then, the OpenCV is downloaded from its official source.
3. After completing the download, a temporary build directory is created and the OpenCV build is set up with CMake.
4. Next, the OpenCV is compiled using the make command.
5. Finally, the OpenCV is installed.

3.2.3 TensorFlow

TensorFlow developed by Google is a free open-source platform which is important for machine learning. In this project, TensorFlow is used as the framework for the Deep SORT object tracking algorithm. The codes for installing TensorFlow in the virtual machine are attached in Appendix B. TensorFlow is installed in a Python virtual environment and the installation steps are as follows:

1. First, a virtual environment is created.
2. Once the virtual environment is activated, the ‘pip’ is upgraded to the latest version and the TensorFlow package which includes GPU support is installed.

3.2.4 Darknet

Darknet is an open-source framework which is used for the implementation of neural networks such as the YOLO algorithm. In our project, Darknet is used to train the YOLOv4 and YOLOv4-tiny deep learning models.

The steps to install the Darknet are shown in Figure 3.3. Firstly, the Darknet git repository created by Alexey AB is cloned. Then, the dependencies of Darknet which include CUDA and cuDNN are installed. Next, the makefile is configured based on the Nvidia GPU architecture of my laptop. After that, the 'make' command is run and then an executable file named 'darknet' is created.

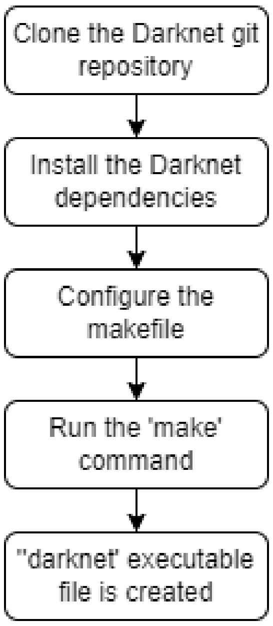

Clone the Darknet git repository
Install the Darknet dependencies
Configure the makefile
Run the 'make' command
"darknet' executable file is created

Figure 3.3: Steps to Install Darknet

3.2.5 OpenVINO Toolkit

The Open Visual Inference and Neural Network Optimization (OpenVINO) toolkit is a free and cross-platform software provided by Intel. The toolkit is simple to use and it helps to optimize the YOLOv4 and YOLOv4-tiny deep learning models for faster inference time on any Intel hardware such as Intel Neural Compute Stick 2 (NCS2), Intel CPU, Intel GPU, and so on.

The workflow of the OpenVINO toolkit is shown in Figure 3.4. Firstly, the trained YOLOv4 or YOLOv4-tiny deep learning model is fed into the Model Optimizer of the toolkit. The YOLO models will be optimized with techniques such as quantization, fusion, freezing and so on. Then, an Intermediate Representation (IR) of the model which includes the .xml and .bin files will be generated. The Intermediate Representation is then fed into the Inference Engine to check the compatibility of the model based on the framework and hardware used. In our project, the framework used is TensorFlow while the hardware used includes Intel CPU and Intel GPU, which are all supported by OpenVINO. Lastly, the model can be executed on the devices through C++ or Python scripts.

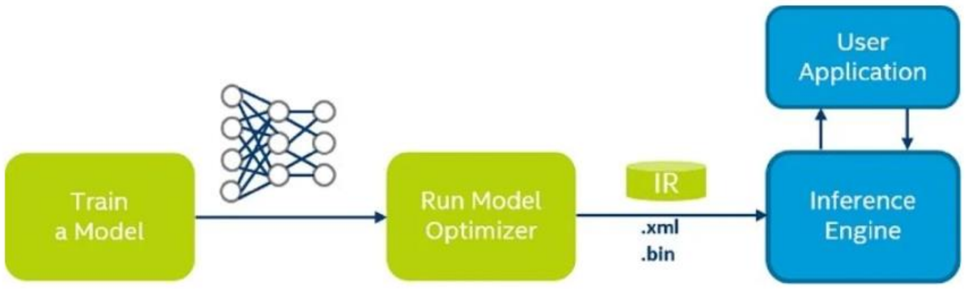


Figure 3.4: Workflow of OpenVINO toolkit

3.3 Dataset Preparation

In implementing an object detection model, a sufficient dataset is required before the training and testing of the model. It is important to have enough datasets with object annotations and labelling so that the neural network can learn the pattern of the detected object and classify them accordingly.

In this project, there are two use cases where the first use case (Use Case 1) is to detect, track and count the number of people entering the geofencing. Meanwhile, the second use case (Use Case 2) is simulating the scenario in a bus where the total number of people boarding and exiting the bus is recorded. For Use Case 1, only the images of people based on the overhead view are required to train the deep learning model. Meanwhile, for Use Case 2, the person images based on the overhead view and also the images of human faces are needed. This is because the detection of the human face is applied at the entrance of the bus while the detection of the person based on the overhead view is applied at the exit.

The dataset is obtained from open sources such as Kaggle, and CVonline. For Use Case 1, the dataset consists of 1000 images of people based on the overhead view. Meanwhile, for Use Case 2, the dataset contains 500 images of people based on the overhead view and 500 images of the human face. Since some of the object annotations in the dataset are missing or miss-aligned, the particular dataset is re-annotated by using labelImg. The labelling file is then stored in the desired YOLO format. The example of object annotation for overhead view person image is shown in Figure 3.5. The images from the dataset are split into a ratio of 8:2, whereby 80% of the images are utilized to train the model while 20% rest of the images are used to test the model.

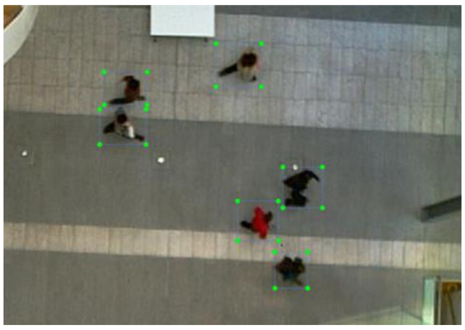


Figure 3.5: Object Annotation for Overhead View Person Image

3.4 Deep Learning Model Framework

3.4.1 Selection of Deep Learning Model

There are several Deep Learning based object detection algorithms as discussed earlier in Chapter 2. In this project, the YOLOv4 model is used for person detection. According to Bochkovskiy et al. (2020), the accuracy and the fps of the YOLOv4 model improve by 10% and 12% respectively as compared to the previous YOLOv3 model. The comparison between different deep learning detection methods is shown in Figure 3.6.

However, Bochkovskiy et al. (2020) recommended that the YOLOv4 model shall be used on a conventional GPU with a minimum of 8GB VRAM for the best performance. Since the YOLOv4 requires high processing power, the compressed version of YOLOv4, which is the YOLOv4-tiny model is introduced into this project. The YOLOv4-tiny model is usually used to obtain faster training and detection speed. This is because its network structure is much simpler and has lesser parameters than the YOLOv4 model. This allows the YOLOv4-tiny model to be more feasible for the development on mobile or embedded devices (Jiang et al., 2020). The comparison of the YOLOv4 and the YOLOv4-tiny model is shown in Table 3.1 below. According to Jiang et al. (2020), the YOLOv4-tiny model obtains a lower mean average precision (mAP) than that of the YOLOv4 model. The lower the mAP, the less accurate the detection model is. However, the YOLOv4-tiny obtains relatively high fps, which is around 6.5 times greater as compared to the YOLOv4 model which has only 41 fps. Besides, the mAP of the YOLOv4-tiny model can be further improved by feeding in more datasets for transfer learning on the customised classes of objects.

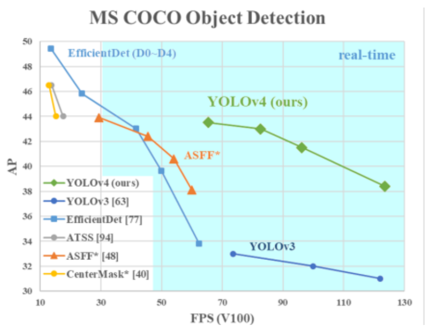
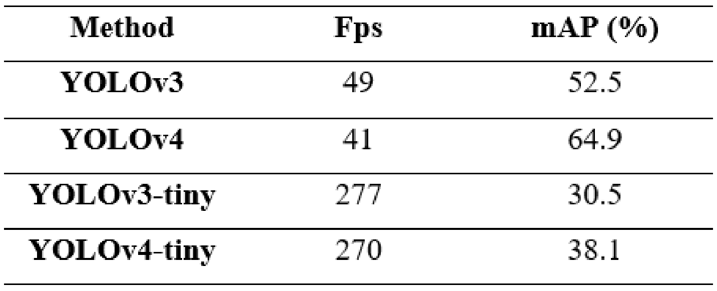


Figure 3.6: Comparison of YOLOv4 models and Other Object Detection Models

Table 3.1: Comparison of Different Detection Methods (Jiang et al., 2020)



Method	Fps	mAP (%)
YOLOv3	49	52.5
YOLOv4	41	64.9
YOLOv3-tiny	277	30.5
YOLOv4-tiny	270	38.1

3.4.2 Training of Deep Learning Model

In this project, the deep learning models selected which include the YOLOv4 and YOLOv4-tiny models are trained using Darknet environment. Since there are two use cases in our project, both the YOLOv4 and YOLOv4-tiny models are trained on a single class, as well as on two classes for Use Case 1 and Use Case 2 respectively. Thus, a total of four YOLO models will be trained.

The steps to train an object detection model using Darknet are shown in Figure 3.7. Firstly, the custom dataset prepared in the previous section is moved to the Darknet directory. Then, configuration is done on the config files, .data file and .names file with respect to each use case. Next, instead of training a deep learning model from scratch, the pre-trained YOLOv4 and YOLOv4-tiny weights are downloaded for transfer learning. After getting everything prepared, the training of the YOLO models can be started.

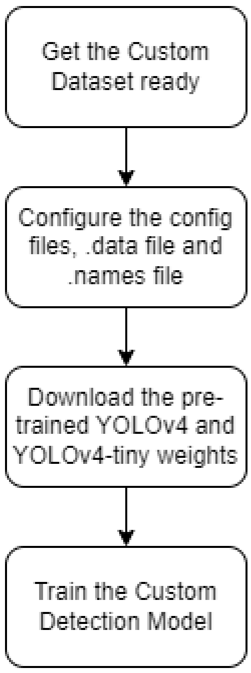

Get the Custom Dataset ready
Configure the config files, .data file and .names file
Download the pre-trained YOLOv4 and YOLOv4-tiny weights
Train the Custom Detection Model

Figure 3.7: Steps to Train a Custom Detection Model using Darknet

3.5 Implementation of Tracking Algorithm

After training the custom detection model, a tracking system is then required to track the person for counting purposes. As discussed in the previous chapter, there are two tracking algorithms that will be implemented in this project which include the Centroid Tracking and the DeepSORT algorithm. The performance of each algorithm will be evaluated afterwards.

3.5.1 Centroid Tracking Algorithm

For the Centroid Tracking algorithm, the information of the classified bounding box is used to track the person who enters the geofencing. For every classified bounding box detected, it will be assigned a count number or a unique ID. Then, the bounding box coordinates are determined and the centroids are calculated by using the diagonal points of the bounding box. In the subsequent frame, if the centroid distance between the old object and the new object exceeds the maximum distance set, a new unique ID is assigned to the new object, and the old object will be deregistered. Whenever a new object ID is registered, the total number of people present in the scene is increased by 1. Figure 3.8 shows the flowchart of the Centroid Tracking algorithm.

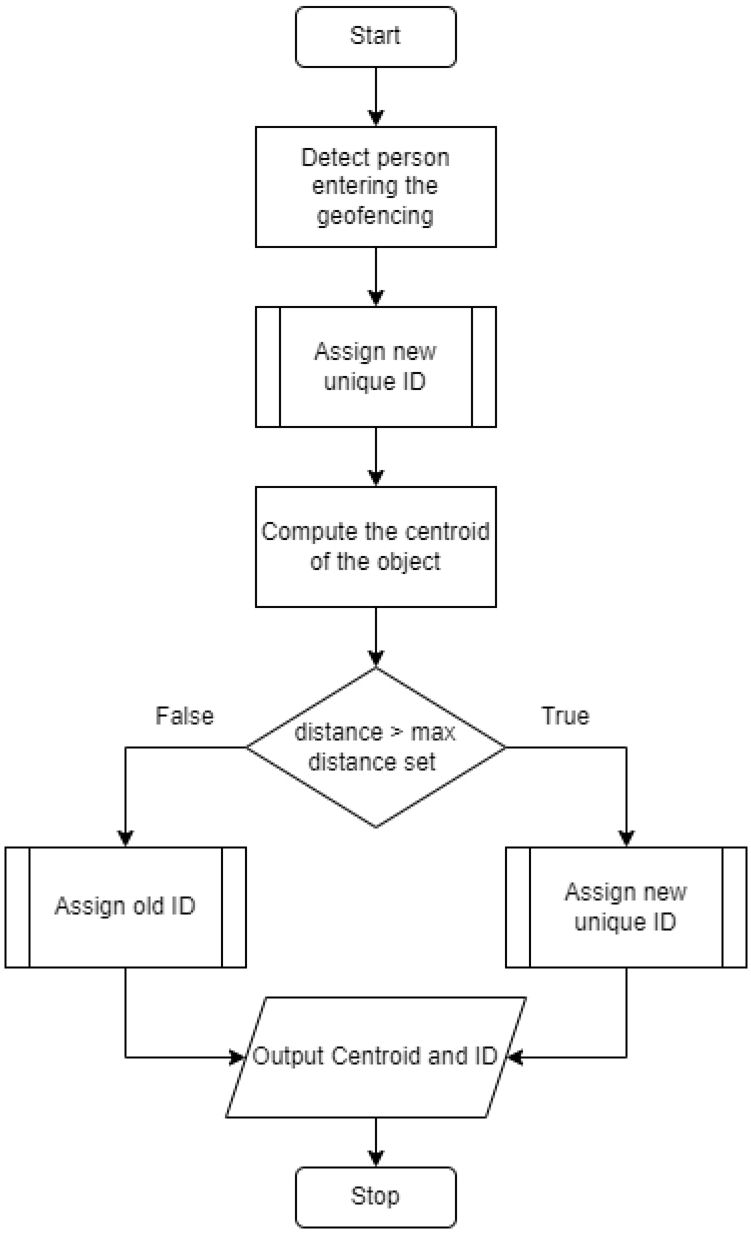

Start
Detect person entering the geofencing
Assign new unique ID
Compute the centroid of the object
distance > max distance set
False
True
Assign old ID
Assign new unique ID
Output Centroid and ID
Stop

Figure 3.8: Flowchart of Centroid Tracking Algorithm

3.5.2 DeepSORT Tracking Algorithm

For the DeepSORT tracking algorithm, a deep appearance-based metric derived from CNN is applied, rather than using only the motion-based metrics as in the SORT algorithm. The overall architecture of the DeepSORT tracking algorithm is shown in Figure 3.9. The architecture consists of three main modules which include the Kalman Filter (KF) based estimation, data association, as well as track management. After the object is detected by the YOLO models, the Kalman Filter predicts the object locations in the current frame. The predicted object locations by KF, together with the object detections by the YOLO models, are the inputs of the data association module. To enhance the data association, a CNN is used by DeepSORT to compute the appearance features which allows the re-identification of tracks even after a long period of occlusion (Pereira et al., 2022). Finally, the array of tracks which contains the bounding boxes and track IDs will be the output of this DeepSORT tracking algorithm. Whenever there is a new track ID, the total number of people in the scene is increased by 1.

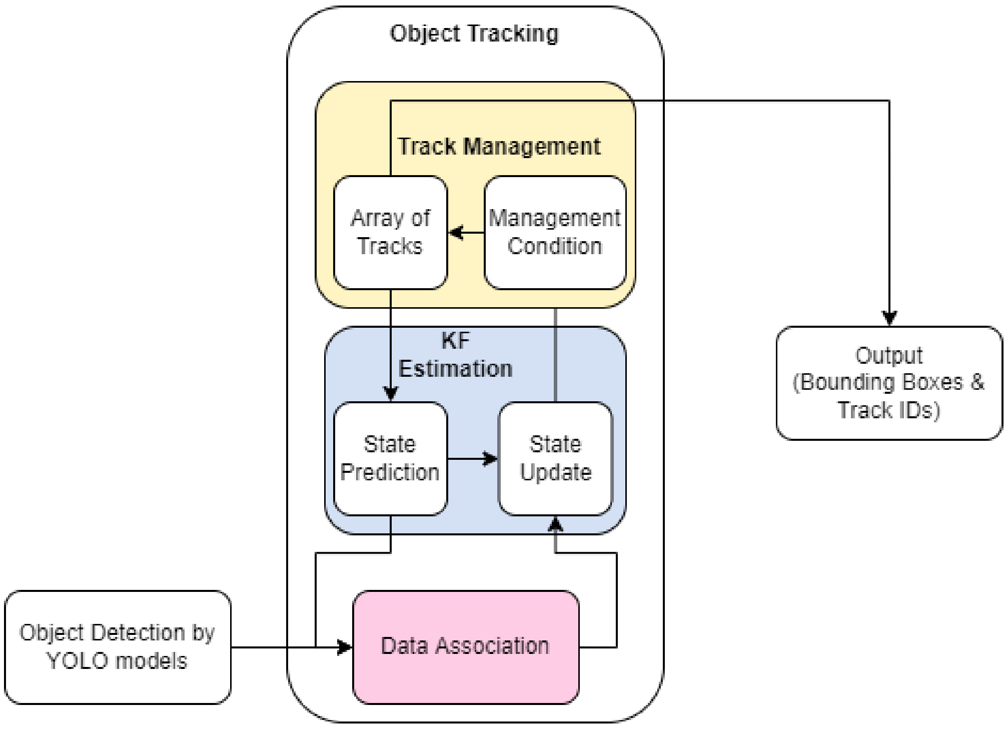


Figure 3.9: Overall Architecture of DeepSORT Tracking Algorithm

3.6 Evaluation of Performance

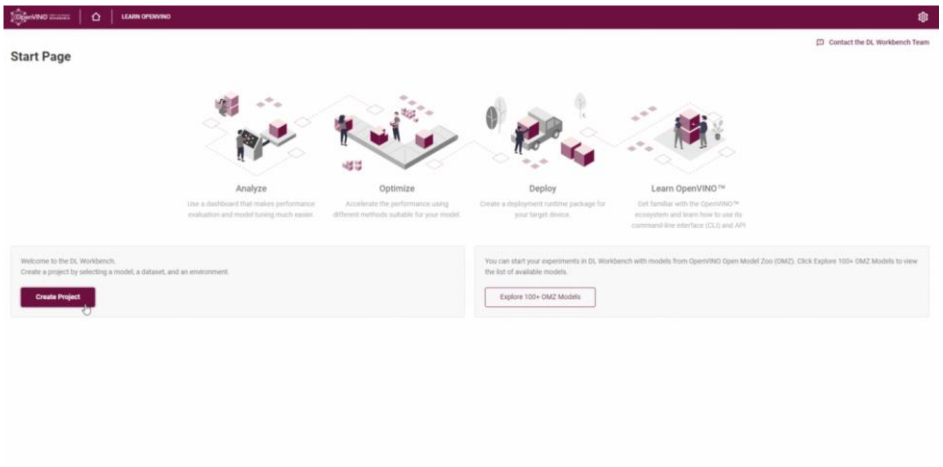
After getting the tracking algorithms ready, the performance of each prototype will be evaluated. The evaluation of performance is done based on the total inference time, fps, mAP for object detection as well as the detection accuracy.

Since there are two use cases in our project, the performance of each prototype will be evaluated separately for each use case. For example, the performance of the YOLOv4 model integrated with the DeepSORT algorithm will be evaluated separately for Use Case 1 and 2. After the performance evaluation, the results are recorded and discussed in the following Chapter 4.

3.6.1 OpenVINO Deep Learning (DL) Workbench

Deep Learning (DL) Workbench is a web-based graphical environment which allows us to visualize, fine-tune and analyse the performance of deep learning models on Intel CPU, Intel GPU, Intel Neural Compute Stick 2 (NCS2), and so on.

In this project, the OpenVINO DL Workbench of version 2021.4.582 is used to evaluate the IR models' performance. The conversion of YOLO to IR models through the OpenVINO toolkit is discussed earlier in Section 3.2.5. The result of the DL Workbench is recorded and discussed in Chapter 4. Figure 3.10 shows the User Interface of the OpenVINO DL Workbench.



Use a dashboard that makes performance evaluation and model tuning much easier.

Accelerate the performance using different methods suitable for your model.

Create a deployment runtime package for your target device.

Get familiar with the OpenVINO™ ecosystem and learn how to use its command-line interface (CLI) and API

Welcome to the DL Workbench.
Create a project by selecting a model, a dataset, and an environment.

You can start your experiments in DL Workbench with models from OpenVINO Open Model Zoo (OMZ). Click Explore 100+ OMZ Models to view the list of available models.

Figure 3.10: User Interface of the OpenVINO DL Workbench

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the performance of this prototype which consists of the object detection models and the tracking algorithm was evaluated based on mean average precision (mAP) for object detection, frames per second (fps), and the detection accuracy. Besides, the performance of the prototype was evaluated based on two use cases. The Use Case 1 is detecting, tracking, and counting the number of people entering the geofencing. Figure 4.1 shows an example result for use case 1. Meanwhile, the Use Case 2 is simulating the scenario in a bus where the number of people boarding and exiting the bus is tracked and counted. Figure 4.2 shows an example result for use case 2. The formula to calculate fps and mAP are shown in equations (4.1) and (4.2) respectively.

$$fps = \frac{input\ frame}{end\ time - start\ time} \tag{4.1}$$

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{4.2}$$

where

fps = frames per second

mAP = mean average precision, %

AP_i = average precision of class i

N = number of classes

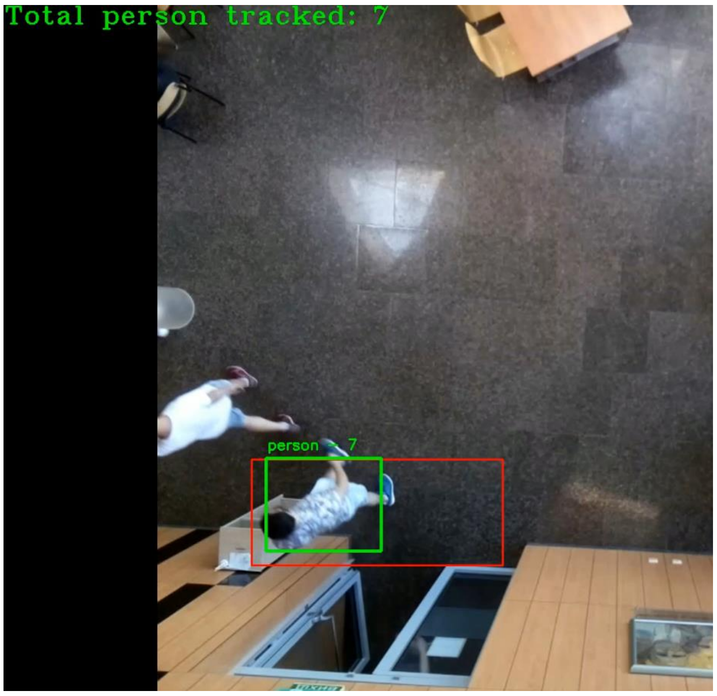


Figure 4.1: Example Result for Use Case 1

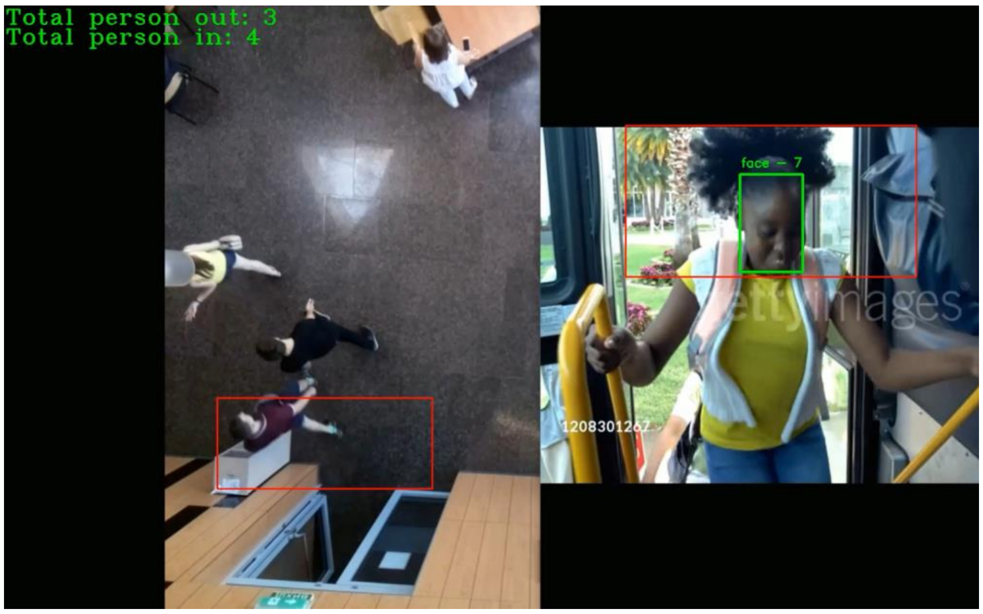


Figure 4.2: Example Result for Use Case 2

The evaluation of the performance consists of three sections as follows:

1. Performance of the Detection Model
2. Performance of the Tracking Algorithm
    - Performance based on Total Inference Time
    - Performance based on fps
    - Performance based on Counting Accuracy
3. Results of OpenVINO DL Workbench

4.2 Performance of the Detection Model

In this project, the pre-trained YOLOv4, and YOLOv4-tiny models by Alexey Bochkovskiy were downloaded and further trained for 6000 iterations on the class of object named “person”. For the second use case, both the pre-trained detection models were also further trained for 6000 iterations as well, but on two classes namely “person” and “human face”. After training for 6000 iterations, both the trained YOLOv4 and YOLOv4-tiny weight files with the best mAP recorded were chosen as our detection models.

Table 4.1 shows the performance of the YOLOv4 and YOLOv4-tiny models in terms of mAP as well as total detection time for use case 1. Based on Table 4.1, the mAP of the YOLOv4 model is 86.48 % which is 1.56 % higher than that of the YOLOv4-tiny model. However, the total detection time for the YOLOv4 model takes 13 seconds longer than the YOLOv4-tiny model. Table 4.2 tabulates the performance of both the detection models for use case 2. Similarly, for use case 2, the mAP of the YOLOv4 model is greater than that of the YOLOv4-tiny model, but the total detection time of the YOLOv4 model is around 4 times longer as compared to the YOLOv4-tiny model. Figure 4.3 compares the mAP of YOLOv4 and YOLOv4-tiny models under both use cases. Based on Figure 4.3, the mAP achieved by both YOLOv4 and YOLOv4-tiny model for use case 1 is greater than the use case 2. This is because for use case 1, all the dataset are used to train on the same single class, but for use case 2, the dataset is divided to train on 2 new classes which include person from overhead view and human faces.

Table 4.1: Performance of the Detection Models for Use Case 1

Case 1	MAP (%)	Total detection time (s)
YOLOv4	86.48	16
YOLOv4-tiny	84.92	3

Table 4.2: Performance of the Detection Models for Use Case 2

Case 2	MAP (%)	Total detection time (s)
YOLOv4	84.20	16
YOLOv4-tiny	80.33	4

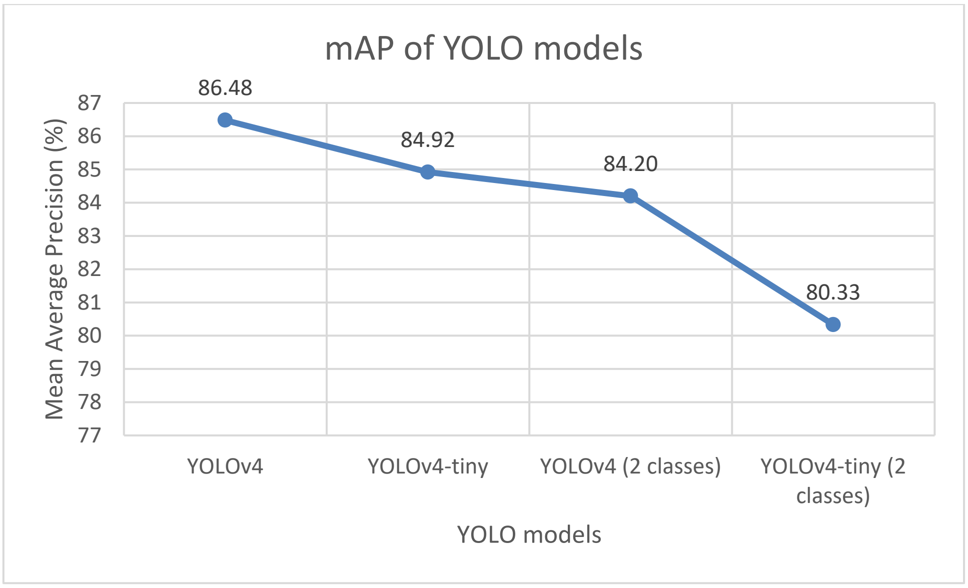


Figure 4.3: mAP of YOLO models for Both Cases

4.3 Performance of the Tracking Algorithm

In this section, both the trained YOLOv4 and YOLOv4-tiny models were integrated with one of the tracking algorithms which include the Centroid Tracking (CT) and DeepSORT (DS) tracking algorithms. The performance of the YOLO models with different tracking algorithms was evaluated based on the total inference time, fps, and the counting accuracy which is computed in terms of the number of miscounts. Other than that, the performance of these solutions was also evaluated based on the processing unit used. The processing units include the laptop’s CPU (i5-6300 HQ) and GPU (Intel HD Graphics 530).

4.3.1 Performance based on Total Inference Time

Figures 4.4 and 4.5 show the total inference time of each YOLO model when integrated with different tracking algorithms for Use Case 1 and 2, respectively. Based on Figures 4.4 and 4.5, for both use cases, the total inference time for YOLOv4-tiny models is a lot shorter than that of the YOLOv4 models, regardless of the tracking algorithms used.

Besides, the effect of different processing units is significant as well. For YOLOv4 models, the total inference time of GPU (Intel HD Graphics 530) is around 3 times shorter than CPU (i5-6300HQ). Meanwhile, for YOLOv4-tiny models, the total inference time of GPU is about 2 times lower than that of CPU. Based on Figure 4.5, for both YOLOv4 and YOLOv4-tiny models, the total inference time recorded when integrated with Centroid Tracking is slightly shorter as compared to integrating the DeepSORT tracking algorithm.

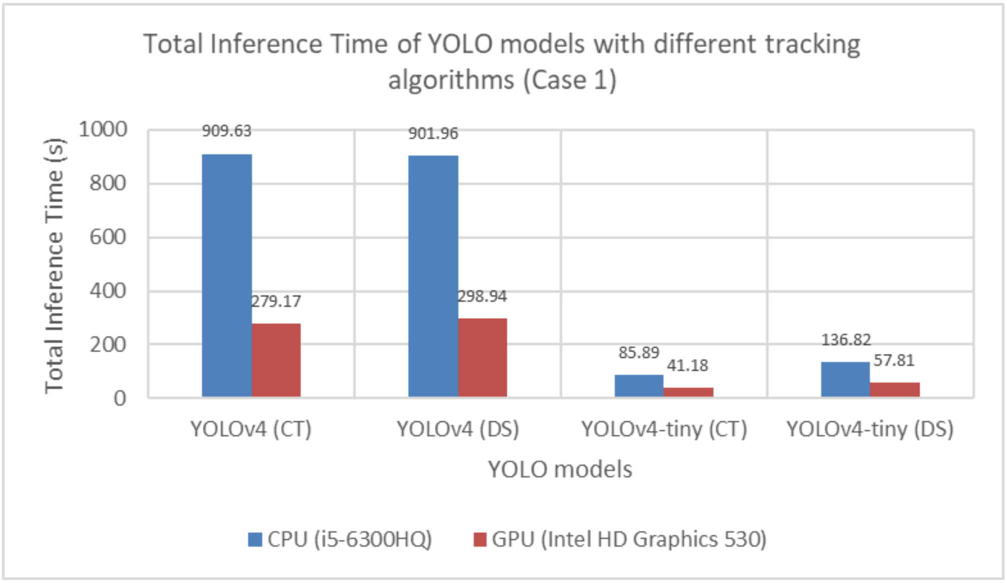


Figure 4.4: Total Inference Time of YOLO models with different Tracking Algorithms (Use Case 1)

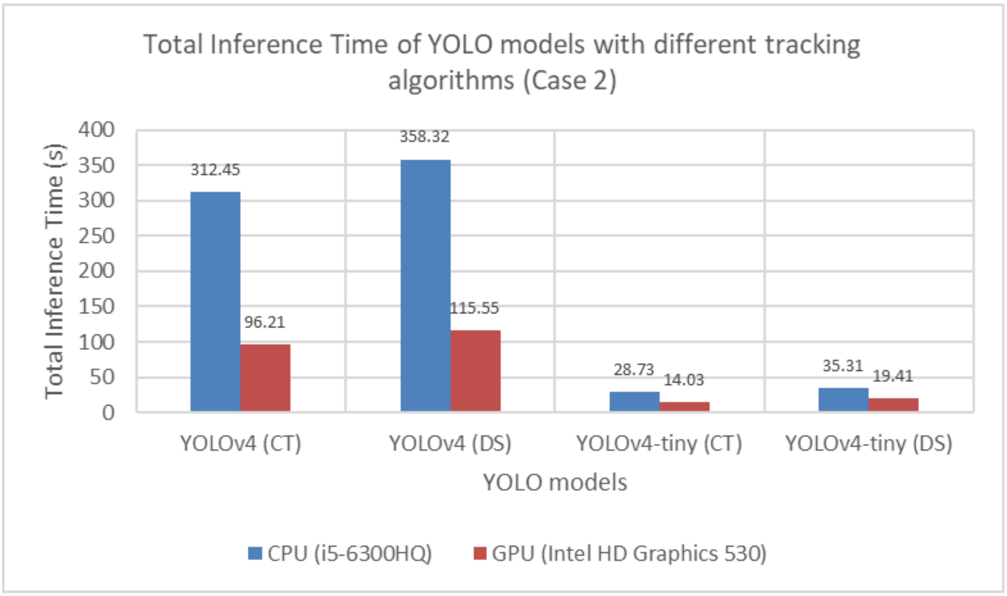


Figure 4.5: Total Inference Time of YOLO models with different Tracking Algorithms (Use Case 2)

4.3.2 Performance based on fps

Frames per second (fps) measures how fast the YOLO models can detect the person and generate the desired output. Figures 4.6 and 4.7 illustrate the fps achieved by the YOLO models with different tracking algorithms for Use Case 1 and 2 respectively. For both use cases, the highest fps is recorded by the YOLOv4-tiny detection model integrated with the Centroid Tracking algorithm. For use case 1, it can achieve 21.03 fps when running with GPU and 10.08 fps when running with CPU. Meanwhile, for use case 2, the highest fps recorded when running with GPU is 22.24 fps and 10.86 fps with CPU. For both use cases, the fps achieved by the YOLOv4 model is significantly smaller as compared to that of the YOLOv4-tiny models. In terms of the processing unit, for YOLOv4 models, the fps achieved when running with GPU is around 3 times faster than CPU. Meanwhile, for YOLOv4-tiny models, the fps obtained when running with GPU is about 2 times faster than that of CPU.

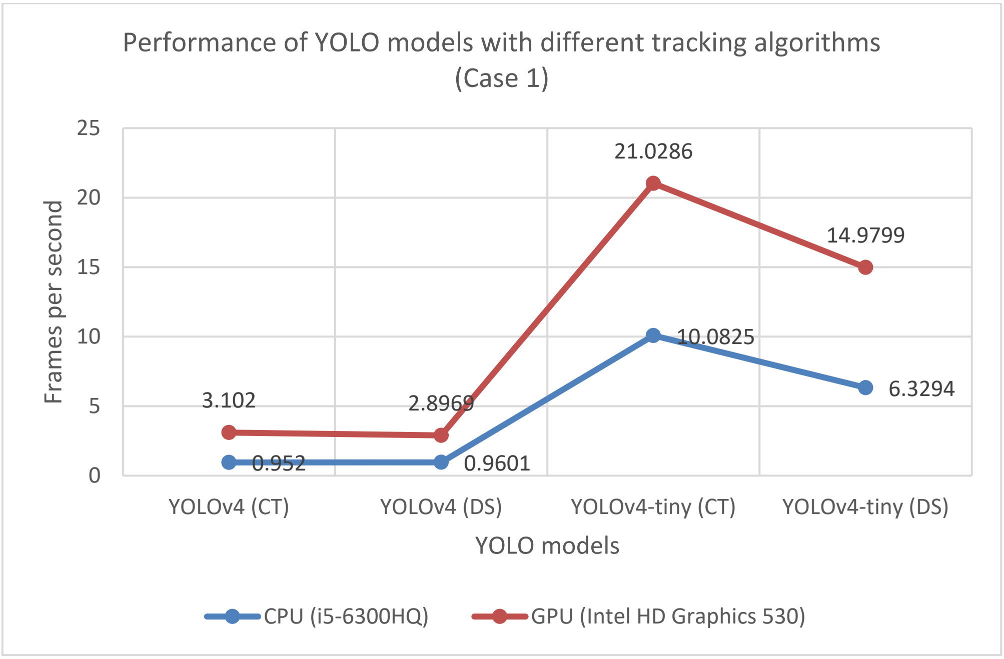


Figure 4.6: Fps Achieved by YOLO models with different Tracking Algorithms (Use Case 1)

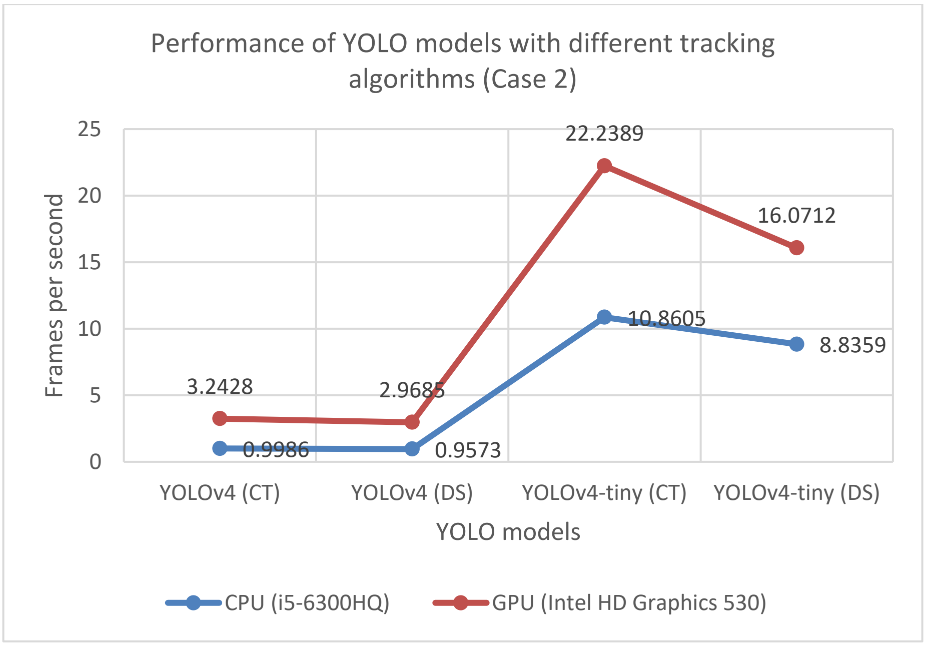


Figure 4.7: Fps Achieved by YOLO models with different Tracking Algorithms (Use Case 2)

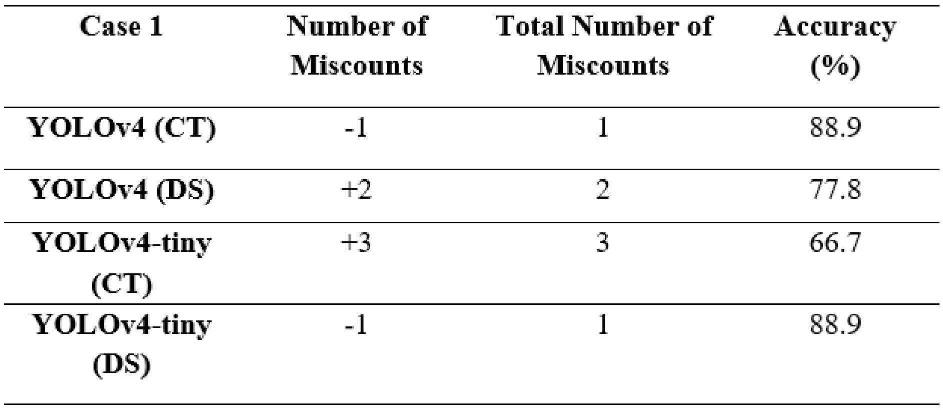
4.3.3 Performance based on Counting Accuracy

Tables 4.3 and 4.4 tabulate the total number of miscounts generated by the prototype and the accuracy for use case 1 and 2 respectively. The number of miscounts recorded may be in positive or negative value. The ground truth values for use case 1 and use case 2 are 9 people and 11 people respectively. If the number of miscounts is positive, this means that there is an extra number of people recorded as compared to the ground truth value. Otherwise, if the number of miscounts recorded is negative, the value shows that there are how many people that are not being detected by the prototype. The lesser the total number of miscounts recorded, the more accurate the prototype is. The accuracy is calculated by using equation (4.3).

$$Accuracy = \left(1 - \frac{Total\ number\ of\ Miscounts}{Ground\ Truth\ Value}\right) \times 100\% \tag{4.3}$$

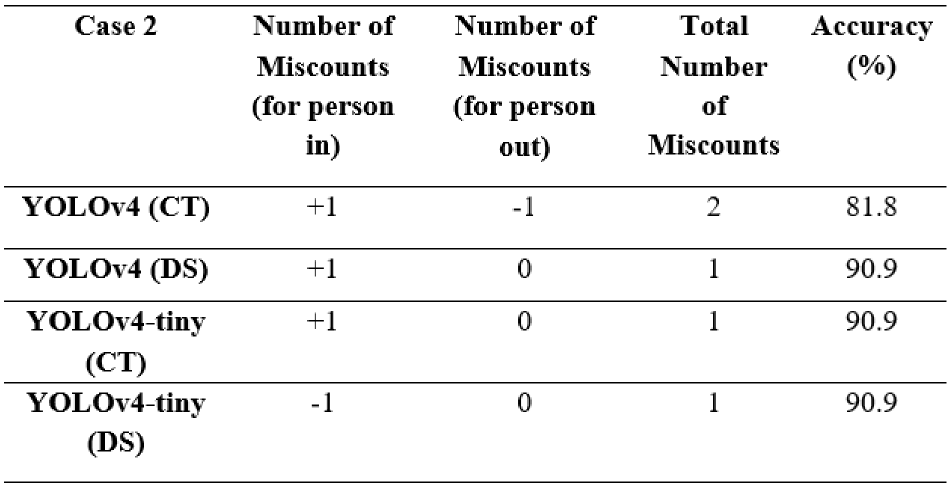
Based on Table 4.3, the YOLOv4-tiny integrated with the Centroid Tracking algorithm has the highest number of miscounts which is a total of 3 miscounts. According to Section 4.3.2, although the YOLOv4-tiny model with Centroid Tracking algorithm achieves the highest fps, however, the counting accuracy is relatively low as compared to other solutions. On the other hand, although the YOLOv4-tiny integrated with the DeepSORT tracking algorithm achieves only the second highest fps, but it manages to record a smaller total number of miscounts with the accuracy of 88.9% and 90.9% for use case 1 and use case 2 respectively.

Table 4.3: Total Number of Miscounts and Accuracy for Use Case 1



Case 1	Number of Miscounts	Total Number of Miscounts	Accuracy (%)
YOLOv4 (CT)	-1	1	88.9
YOLOv4 (DS)	+2	2	77.8
YOLOv4-tiny (CT)	+3	3	66.7
YOLOv4-tiny (DS)	-1	1	88.9

Table 4.4: Total Number of Miscounts and Accuracy for Use Case 2



Case 2	Number of Miscounts (for person in)	Number of Miscounts (for person out)	Total Number of Miscounts	Accuracy (%)
YOLOv4 (CT)	+1	-1	2	81.8
YOLOv4 (DS)	+1	0	1	90.9
YOLOv4-tiny (CT)	+1	0	1	90.9
YOLOv4-tiny (DS)	-1	0	1	90.9

4.4 Results of OpenVINO DL Workbench

Deep Learning (DL) Workbench is a web-based graphical environment which allows us to visualize, fine-tune and analyse the performance of deep learning models on Intel Neural Compute Stick 2 (NCS2), Intel CPU, Intel GPU, and so on. After the installation of DL Workbench on the local system, the OpenVINO Intermediate Representation (IR) model is imported, and the validation dataset is uploaded for analysis. After modifying all the configuration settings, the result of the model analysis on DL Workbench in terms of fps is shown in Figure 4.8.

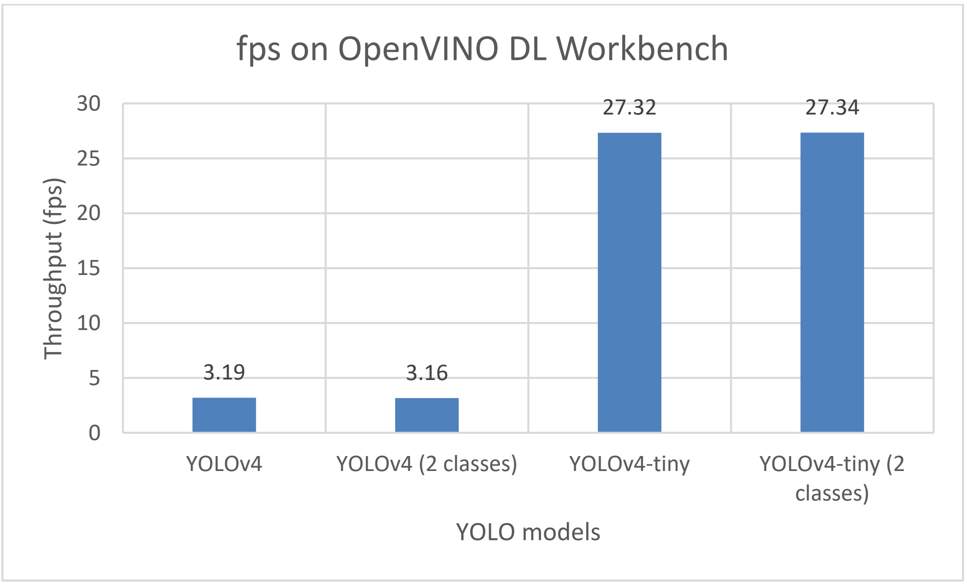


Figure 4.8: Result of Model Analysis on DL Workbench

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this project, a person tracking and counting system based on overhead view was developed. Transfer learning of the YOLO model was done by using a custom dataset which includes the images of person based on the overhead view and the images of the human face. This prototype consists of a deep learning detection model integrated with one of the tracking algorithms. The YOLOv4-tiny detection model is selected due to its lightweight property and its ability to achieve high fps without the need for high processing power. Even though the Centroid Tracking algorithm achieves around 38.4% to 40.4% higher fps as compared to that of the DeepSORT tracking algorithm, its counting accuracy is around 22.2% lower than the DeepSORT tracking algorithm. Hence, the overall performance of the YOLOv4-tiny model integrated with the DeepSORT algorithm is better as compared to the others.

5.2 Recommendations for future work

The bounding boxes generated by the YOLOv4-tiny detection model were slightly larger than the normal and accurate bounding boxes. Hence, further training of the YOLOv4-tiny detection model is recommended to increase the detection accuracy.

Other than that, a front-end development can be done to improve the user experience. For example, a web-based user interface or dashboard can be created for this prototype.

REFERENCES

Ahmad, M., Ahmed, I., & Adnan, A., 2019. Overhead View Person Detection Using YOLO. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON).*

Ahmed, I. and Adnan, A., 2017. A robust algorithm for detecting people in overhead views. *Cluster Computing,* 21(1), pp. 633–654.

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B., 2016. Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP).*

Bochkovskiy, A., Wang, C., & Liao, H.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection.

Dalal, N., & Triggs, B., 2005. Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).*

Galvez, R. L., Bandala, A. A., Dadios, E. P., Vicerra, R. R. P., & Maningo, J. M. Z., 2018. Object Detection Using Convolutional Neural Networks. *TENCON 2018 - 2018 IEEE Region 10 Conference.*

Girshick, R., 2015. Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV).*

Girshick, R., Donahue, J., Darrell, T., & Malik, J., 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition.*

Gulli, A. and Pal, S., 2017. *Deep Learning with Keras.* Birmingham: Packt Publishing.

Jiang, Z., Zhao, L., Li, S., & Jia, Y., 2020. Real-time object detection method for embedded devices.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C., 2016. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, pp. 21-37.

Lowe, D. G., 1999. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision.*

Lowe, D. G., 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), pp. 91–110.

Miraftabzadeh, S. M., Foiadelli, F., Longo, M., & Pasetti, M., 2019. A Survey of Machine Learning Applications for Power System Analytics. *2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe).*

Nakatani, R., Kouno, D., Shimada, K. & Endo, T., 2012. A Person Identification Method Using a Top-view Head Image from an Overhead Camera. *2012 Journal of Advanced Computational Intelligence and Intelligent Informatics*, 16(6).

Olejniczak, M. & Kraft, M., 2017. Taming the HoG: The Influence of Classifier Choice on Histogram of Oriented Gradients Person Detector Performance. *International Conference on Artificial Intelligence and Soft Computing*, 552-560.

Ozturk, O., Toshihiko Yamasaki, & Kiyoharu Aizawa., 2009. Tracking of humans and estimation of body/head orientation from top-view single camera for visual focus of attention analysis. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops.*

Parico, A.I.B. & Ahamed, T., 2021. Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT. *Sensors*, 21(14), pp. 4803.

Pereira, R., Carvalho, G., Garrote, L., & Nunes, U.J., 2016. Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics. *Applied Sciences*, 12(3), 1319.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Ren, S., He, K., Girshick, R., & Sun, J., 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), pp. 1137–1149.

Wojke, N., Bewley, A., & Paulus, D., 2017. Simple online and realtime tracking with a deep association metric. *2017 IEEE International Conference on Image Processing (ICIP).*

Zhao, Z., Li, H., Zhao, R. and Wang, X., 2016. Crossing-line crowd counting with two-phase deep neural networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9912 LNCS, pp.712–726.

APPENDICES

Appendix A: Code for Installing OpenCV

```
$ sudo apt-get install build-essential cmake unzip pkg-config
$ sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
$ sudo apt-get install libjasper-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libv4l-dev libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk-3-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python3-dev
$ sudo apt-get install openexr libtbb2 libtbb-dev libdc1394-22-dev
$ mkdir ~/opencv_build && cd ~/opencv_build
$ git clone https://github.com/opencv/opencv.git
$ git clone https://github.com/opencv/opencv_contrib.git
$ cd ~/opencv_build/opencv
$ mkdir build && cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
   -D CMAKE_INSTALL_PREFIX=/usr/local \
   -D INSTALL_C_EXAMPLES=ON \
   -D INSTALL_PYTHON_EXAMPLES=ON \
   -D OPENCV_GENERATE_PKGCONFIG=ON \
   -D
OPENCV_EXTRA_MODULES_PATH=~/opencv_build/opencv_contrib/mo
dules \
   -D BUILD_EXAMPLES=ON ..
$ make -j8
$ sudo make install
```

Appendix B: Codes for Installing TensorFlow

```
$ sudo apt install python3-venv
$ mkdir my_tensorflow
$ cd my_tensorflow
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install --upgrade pip
$ pip install --upgrade tensorflow-gpu
```