Overhead Person Detection on Mobile Platforms

Tong Siew Wen

Department of Mechatronics Engineering

In partial fulfilment of the
Requirements for the Degree of
Bachelor of Engineering at
Monash University

Under the supervision of:
Dr. Soon Foo Chong (Monash Malaysia)
Dr. Shahnewaz Chowdhury (ELID Sdn. Bhd.)

Semester 2, 2021

Copyright notice

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

Abstract

Person detection is an important step in in visual surveillance. This final year project aims to train a detector, which is part of the anti-tailgate system, that is able to accurately detect the presence of humans from an overhead view when deployed on a mobile platform. This report aims to summarize what is done in this one-year period, as well as the findings and outcome of this project. The overhead person detection model architecture used is MobileNetV2 SSD due to its lightweight characteristics. The model is trained using a self-collected dataset that undergoes various augmentations via traditional augmentation techniques and via StyleGAN2-ADA. A precision of 90% and recall of 70% is achieved with a deployment speed greater than 30 FPS when the trained model is tested in the target environment.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Student signature: 

Student name: Tong Siew Wen Date: 17/9/2021

Student ID: 29095093

Acknowledgments

I would like to show my appreciation to ELID Sdn. Bhd. to offer this project to me as my Final Year Project, and dedicated resources to enable smooth completion of the project. I would also like to thank Monash University Malaysia to provide me financial support in this project. Last but not least, I wish to deep and sincere gratitude to both my supervisors, Dr. Soon Foo Chong (Monash University Malaysia) and Dr. Shahnewaz Chowdhury (ELID Sdn. Bhd.) for their constant support and invaluable guidance throughout the project. The completion of this project would not have been possible without the assistance they have provided. Thanks for everything.

Table of Contents

List of figures

List of Tables

List of abbreviation

Abbreviations	Definition
AR	Average Recall
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAGAN	Data Augmentation Generative Adversarial Network
EC	ELID Cafeteria (dataset name)
EDE	ELID DPD Entrance (dataset name)
EDI	ELID DPD Indoor (dataset name)
EDOB	ELID DPD Outdoor Bright (dataset name)
EDOD	ELID DPD Outdoor Dark (dataset name)
FPS	Frame per second
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradient
IoU	Intersection Over Union
mAP	mean Average Precision
MC-GAN	Multi-Conditional Generative Adversarial Network
RCNN	Region-based Convolutional Neural Networks
SSD	Single Shot Detector
SVM	Support Vector Machines
TPU	Tensor Processing Unit
TVMPC	Top View Multiple Person Cafeteria (dataset name)
TVPR	Top View Person Reidentification (dataset name)
YOLO	You Only Look Once

1.0 Introduction

1.1 Project background

Tailgating, also known as piggybacking, is defined to be the action of tagging along with an authorized person into a restricted area. Tailgating can be seen as a behavior that violates physical access control policies, as an authorized person can illegally enter critical infrastructure facilities that should be safeguarded. Physical access systems that purely rely on card readers or/and movement sensors are far from adequate to prevent tailgating [1]. This is because most of the automated electronics access control in the market do not monitor and do not have control over the number of people entering and leaving the facilities [2]. An analysis by United States government demonstrates that undercover agents from the US Federal Aviation Administration (FAA) successfully sneaked through security measures at major airports 68% of their tries, where one of the methods used is by tailgating staff through doors into controlled areas [3]. Hence, to tackle this problem, ELID aims to produce an anti-tailgate system, named ELIDEye [4], where its function is shown below:

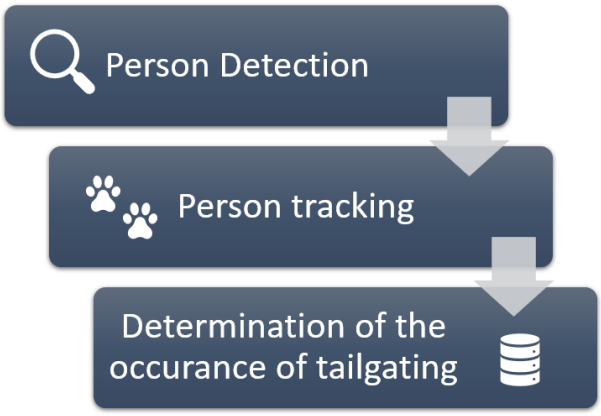


Figure 1: Process detecting tailgating by ELIDEye product

Also, since the strike of the COVID19 pandemic since 2019, the government has enforced several standard operating procedures (SOP) to prevent the spreading of the virus. One of the SOP enforced is that the number of people in a premise is restricted [5, 6]. Since ELIDEye is able to detect and track the people entering and exiting a premise, it is possible to include the feature of people counting which will aid companies and organizations in adhering the SOP [7]. Using artificial intelligence to perform people

counting is much more cost effective as compared to the current solution implemented by most companies in Malaysia, i.e., using labor to perform manual people counting. Hence, with this newly added feature, the workflow of ELIDEye is as follows:

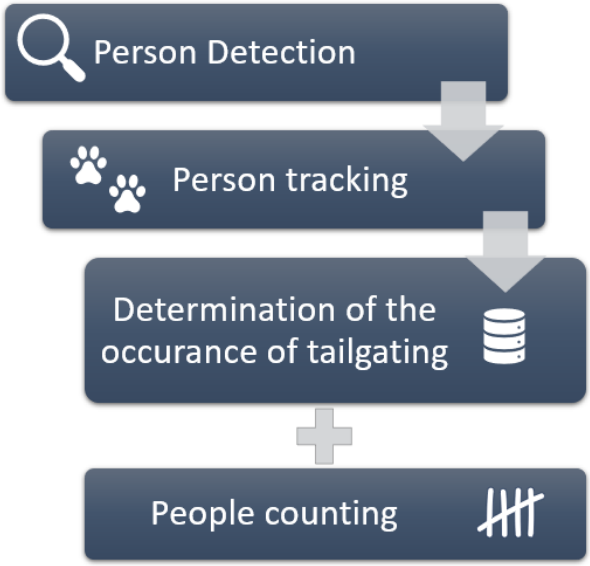


Figure 2: Workflow of ELIDEye product with the addition of people counting feature

In this final year project, the focus will be on person detection. For the previous version of ELIDEye, traditional machine learning methods are used, i.e., using HOG as the feature extractor and SVM as the classifier [8, 9], which is found less accurate when being deployed in unseen environments. Hence, a large amount of training data is required to be collected from that unseen environment to train the model from scratch before being able to be deployed there. From the insights given by ELID, the whole process is lengthy and would take more than two days. Even worse, some effort and cooperation are needed from the customer to validate the training. This has caused ELID's customers to be unpleasant as they hope that the product can be deployed as soon as possible after installation and need minimal effort from them even if on-site training is required.

Therefore, ELID would like to explore person detection using a deep learning mechanism to solve the problem described above. Since the application of person detection is gaining more and more attention, especially in video surveillance, access control, etc. a great amount of research is done, which results in a great choice of mature architectures that are available to use. However, there are still challenges exist in person detection such as

a large variety of human features due to change in pose, illumination, occlusion, etc. [10]. As compared to frontal view, using the overhead view of people is able to solve the occlusion problem but there are much lesser features when people are viewed from above [11]. Since ELID's main goal is to prevent tailgating, it is obvious that overhead view would be a more suitable choice. This is because any miss detection due to occlusion is not acceptable, else tailgating could not be detected.

Most existing detection achieves great performance, given that the training dataset and the testing dataset have very similar distributions [12]. However, this assumption is not true in real-life applications, as the scene that the overhead camera is deployed at the customer site will be different from the scene in the training dataset. Scene complexity and variation will also cause the performance of the detection model to drop during actual deployment [12, 13]. Hence, data collection and retraining at the customer site is necessary. This leads to another problem: It is expensive to perform data collection and labeling every time the model is deployed at a new site [13, 14]. Also, as mentioned, ELID would like to have the object detection model to be deployed as soon as possible on site. Thus, this project aims to also implement transfer learning to reduce the need to recollect data and minimize the training duration.

To have a good-performing deep learning model, it is important to have high-quality training that is sufficiently large and varied and should represent the reality [15]. At the initial stage of this project, it is realized that the size and variation of the training dataset are not sufficient, causing a bottleneck in the performance. Hence, to tackle this problem, this project also aims to investigate the effect of introducing image augmentation in the training data on the performance of the person detection model. As traditional augmentation methods are very limited in terms of invariances, Generative Adversarial Network (GAN) is implemented to learn a much larger invariance space so as to introduce more variations to the training dataset [16]. Since the introduction of GAN [17], it has been a very active topic of machine learning research in recent years, as it is unsupervised which implicitly learns an underlying distribution.

1.2 Terminologies Definition for COCO object detection evaluation metrics

Precision/recall is introduced to replace the area under curve (AUC) measure of the receiver operating characteristic (ROC) curve used in VOC2006 for the classification task. This replacement aims to improve the sensitivity and interpretability of the metrics, as well as to increase the visibility to performance at low recall [18]. Since the terminology of mAP and AR will be frequently used throughout the report, their definitions and method of computation are explained below to better understand the report content.

1.2.1 General definition of precision and recall

Precision is defined as the proportion of correct predictions (true positives) out of all predictions made, while recall is defined as the proportion of correct predictions out of all ground truth instances. In other words, precision and recall can be represented using the following equation:

$$Precision = \frac{TP}{FP + TP}, \qquad Recall = \frac{TP}{FN + TP}$$

Equation 1: Precision and recall in object detection

where TP refers to True Positives (correct predictions), FP refers to False Positives (wrong predictions), and FN refers to False Negatives (missed predictions).

If the Precision and Recall are plotted against each other across different confidence threshold values, a precision/recall curve is computed. An example of a precision/recall curve is shown below:

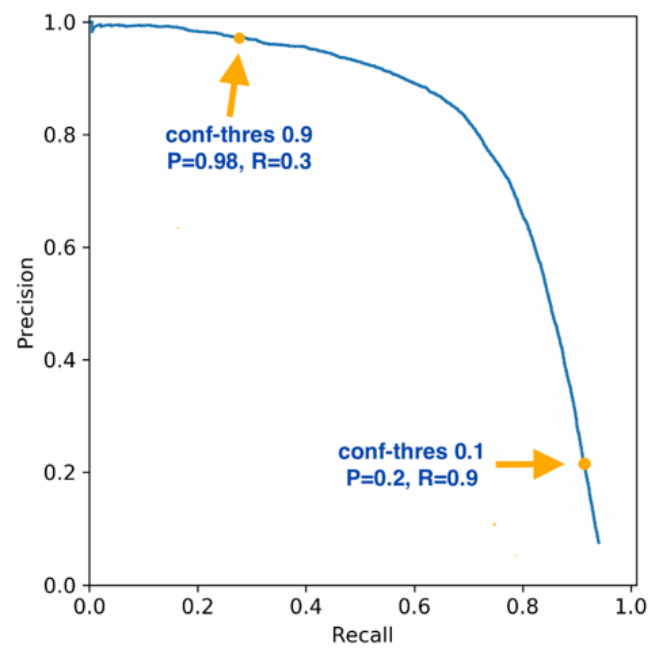


Figure 3: Precision/Recall Curve.

Adapted from:[19]

As it is hard to use the precision/recall curve to perform the comparison between different models, Average Precision (AP) and Average Recall (AR) are introduced, incorporated with Intersection over Union (IoU).

1.2.1 Intersection over Union (IoU)

IoU measures the overlap (intersection over union) of two boundaries [20], as represented using the equation below:

$$IOU = \frac{A \cup B}{A \cap B}$$

Equation 2: Intersection over Union

By computing IoU of the ground truth bounding box (A) and the predicted bounding box (B), it can measure the accuracy of an object detection model. Hence, the object detection model will aim to achieve a high IoU i.e., great overlapping of the ground truth and predicted bounding boxes. IoU is also used to define whether the prediction is a true or false positive. Normally, an IoU greater than 0.5 acts as an indicator of an accurate prediction [20].

1.2.2 mean Average Precision (mAP)

Average Precision (AP) is defined as the mean precision at a set of eleven equally spaced recall levels from 0 to 1. To smoothen the precision/recall curve, interpolation is performed where the precision of each recall level to taken to be the highest precision to the right of that recall level. AP can be computed via the area under the precision-recall curve which is interpolated [18]. It shows how accurate are the predictions of the model [20, 21]. In COCO detection evaluation, AP is also averaged over IoU values from 0.5 to 0.95 with 0.05 intervals. AP is calculated for one class, hence the mean Average Precision (mAP) is the mean of the AP for all classes [22]. For instance, mAP@IoU50 is the mean Average Precision of the model with IoU 0.5. Since our problem is a single class problem, the value for mAP will be the same as AP.

1.2.3 Average Recall (AR)

Similar to Average Precision in COCO detection evaluation, Average Recall (AR) is the mean of recalls between IoU values from 0.5 to 1. In other words, AR summarizes the recall over multiple IoU thresholds for a given number of detections. For instance, AR10 is the AR given ten detections per image. With the incorporation of IOU, AR is able to reflect the proposal recall and the localization accuracy simultaneously [21].

1.3 Overview of the solution proposed

In this project, SSD is chosen for its convenience as the whole pipeline from training to compiling to Edge TPU file is supported in TensorFlow Object Detection API [23]. In the original research paper of SSD, VGG16 is used as the backbone model, but it is also mentioned that using other networks as the backbone model would also produce promising results [24]. As this application is designed as an offline standalone device, the model needs to be able to be deployed in real-time on mobile platforms. This has resulted in a lot of restrictions as networks that are deep are computationally expensive [25], thus they are not suitable for this project. Hence, after comparing different lightweight networks, MobileNetV2 is chosen as it requires much lesser memory and computational power with satisfying accuracy [26]. Literature review and comparison of different object detection architectures are discussed in later sections.

Also, StyleGAN2-ADA and DAGAN are chosen as the architecture to perform data augmentation via GAN, due to the less training complexity and high suitability to this application. Further discussion on the GAN architectures and the rationale behind choosing these two GAN models can be found in later sections. The performance of both GAN models will be compared against each other, and the architecture that is able to generate more realistic images will be chosen to perform data augmentation on the top view person dataset for the object detection training.

Transfer Learning will be performed by initializing the model weights with the MobileNetV2 SSD model from the TensorFlow Model Zoo, where the model is pre-trained on the COCO dataset. Then, during training, the initial layers of the object detector model will be frozen, and its performance and training speed will be compared to the model that is trained from scratch.

The overview of the approach can be illustrated as follows:

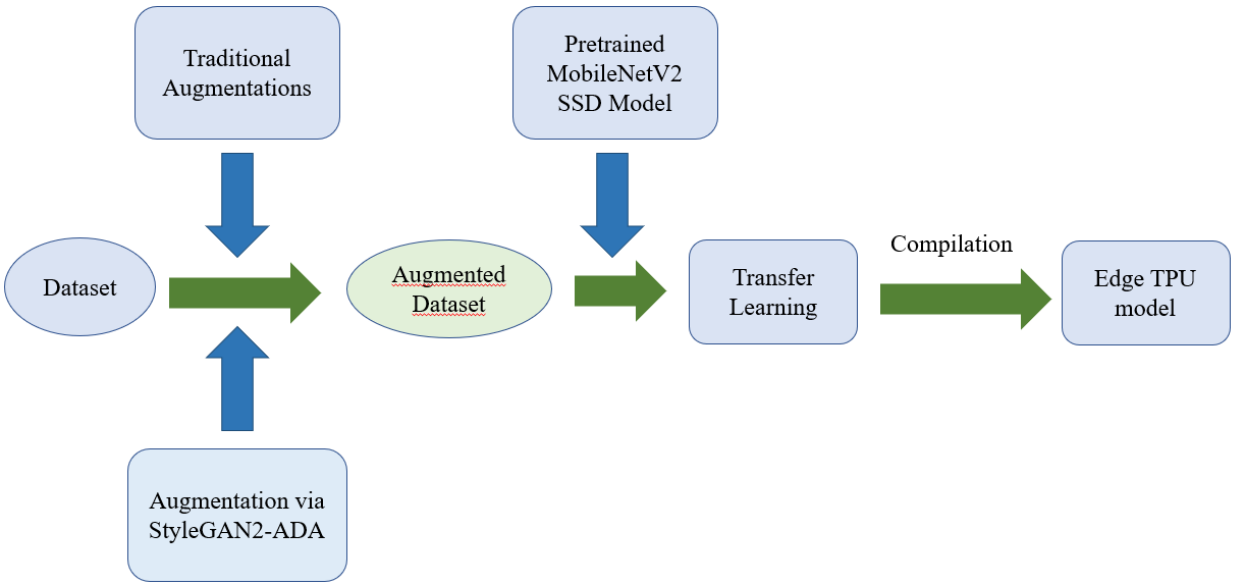


Figure 4: Overview of solution

2.0 Objectives & Research Scope

Main goal:

Train a reliable overhead person detection model for mobile platforms in unconstrained environments.

Objectives:

1. The model can be deployed on Google Coral, and Edge Tensor Processing Unit, with at least 30 frames per second (real-time).
2. The detection of the model should be sufficient for accurate tracking. Hence, the model should have precision and recall of 80% on unseen environments.
3. Extensive testing should be done on the model to validate the model performance.
4. Implement transfer learning to shorten the training time and increase accuracy.
5. Perform image augmentation via traditional data augmentation methods and Design Generative Adversarial Network (GAN). This is to improve the generalization of the datasets to train the model. The images generated should be realistic and able to be used as training samples to the model.

The scope/limitation of the project is as follows:

1. The model should be able to detect humans from one-floor height, i.e., 8 to 14 feet above the ground, as that will be the installation height of the overhead camera.
2. The device used to deploy the model is Raspberry Pi 4 Model B, and Google Coral is used to accelerate the processing operation.
    a. Hence, the architecture needs to be compatible with Google Coral, i.e., able to be compiled to an Edge TPU model.
3. The overhead camera is Raspberry Pi 8MP Camera Module V2.
4. The number of people that the model should be able to detect is 1 to 5 people.
5. The model will be deployed on a mobile device, which should be able to run the program independently offline.

3.0 Literature Review

3.1 Overhead Person Detection

Overhead person detection is a branch of object detection. Object detection provides insightful information for semantic understanding of images and videos [27], as it outputs not only the class of the object but also the location of the object in the form of bounding boxes. The initial approach to tackle object detection is to have a sliding window that browses through the image, which classification is performed in each sliding window. The feature extractor and the classifier are also separate architectures. A popular example would be the R-CNN model [28] which is state-of-the-art in 2012, adopting selective search [29] for region proposal generation, CNN module for feature extraction, and support vector machine (SVM) as classifier [28]. SPP-Net tries to solve the fixed-sized bounding boxes constraint in R-CNN by employing spatial pyramid pooling [30], then Fast R-CNN further improves from that by proposing to merge the feature extractor and the classifier into a single model, in order to speed up training and detection [31]. Faster RCNN introduced an additional Region Proposal Network (RPN) to share full-image convolutional features with the network so that the training and detection speed could be improved [32]. All these models are known to be region-based proposals as they include a preprocessing step for generating object proposals [20, 27]. To further increase the inference speed, one-stage detections frameworks are introduced, with YOLO [33] and SSD [24] being the most representative models.

3.1.1 You Only Look Once (YOLO)

YOLO architecture consists of 24 convolution layers followed by 2 fully connected layers. Compared to region proposal classification networks, YOLO looks at the image as a whole. This is achieved by dividing the image into grids who which will be responsible to perform the predictions. Each prediction comes with the confidence of the model, which is the product of the probability of an object present and the IoU of the bounding box [33]. This can be visualized in Figure 5.

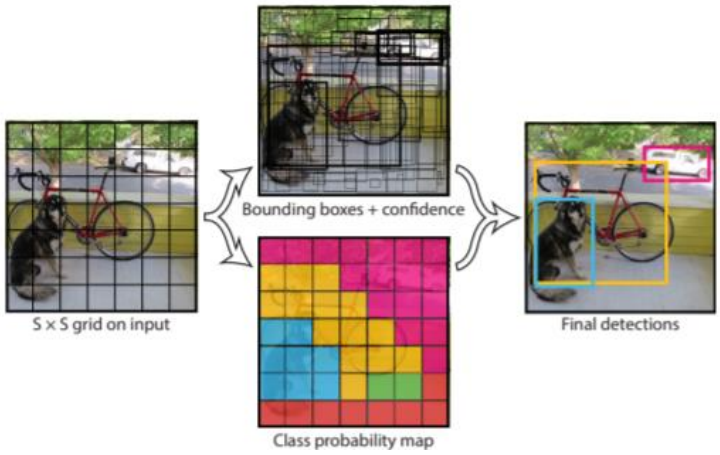


Figure 5: YOLO introduced the new approach to model detection as a regression problem.

Adapted from: [33]

YOLO has a different approach from previous work, as it treats object detection as a regression problem, instead of a classification problem. Hence, its loss function is composed of multiple sum-squared errors, which penalizes boxes for wrong classifications only if the bounding box consists of an object. YOLO is significantly faster than the state of art at that time, with an accuracy slightly lower. On the VOC2007 test, YOLO achieved 45 FPS with mAP 63.4%, compared to Faster R-CNN 7 FPS with mAP 73.2% [33].

The architecture of YOLO is as illustrated below:

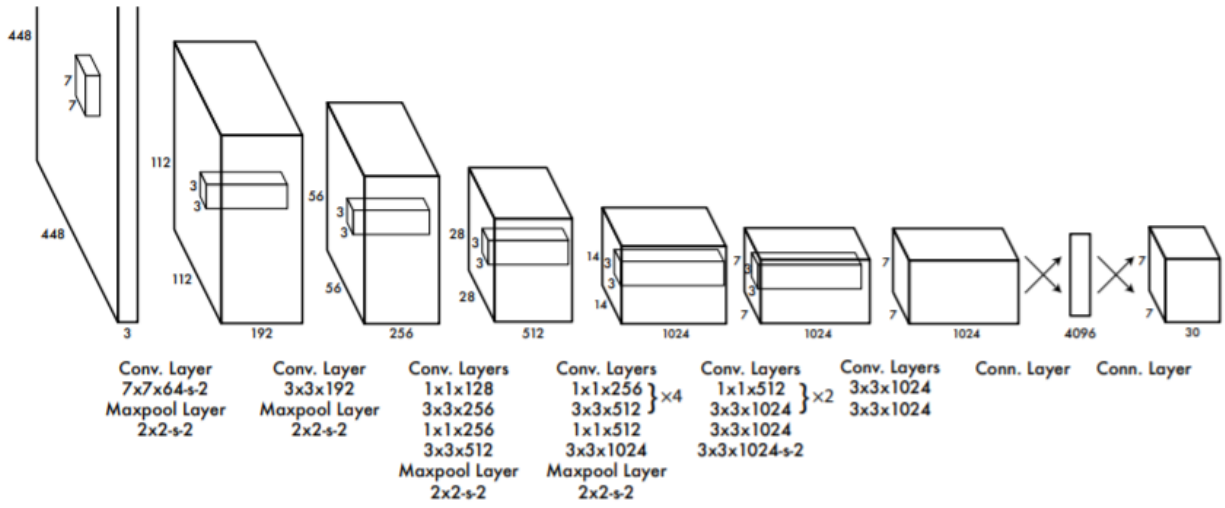


Figure 6: YOLO architecture.

Adapted from: [33]

However, YOLO has its limitation as each grid can only perform one prediction, and struggles when dealing with objects of unusual aspect ratio [33]. Inspired by SSD, YOLO_v2 attempts to solve this limitation by introducing anchor boxes in the bounding boxes, where the fully connected layers are replaced with anchor boxes prediction. Batch normalization is also introduced to increase the convergence of the model, which improves the mAP by 2% [34]. The classifier is also trained with images with higher resolution [34] as it is observed that features used to perform detection as relatively coarse, which affects the performance of the model [33].

3.1.2 Single Shot Detector (SSD)

The architecture of SSD is as shown in the image below:

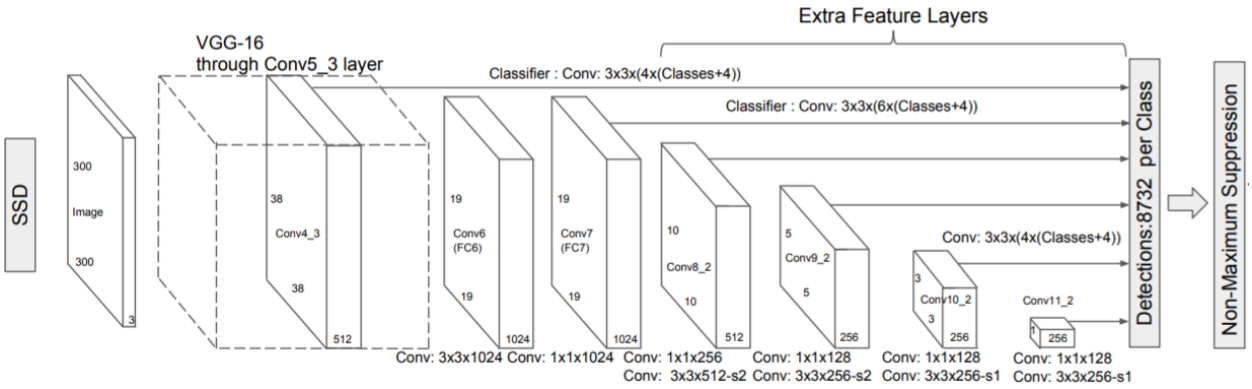


Figure 7: SSD architecture.

Adapted from:[24]

Different from YOLO that handcrafted its CNN network, SSD uses existing CNN networks as its 'backbone' with the last layer stripped off and replaced with additional convolutional layers. In the paper, the backbone used is VGG16 that is pre-trained on the ILSVRC CLS-LOC dataset. Inspired by anchor boxes in Faster R-CNN, SSD not only separates each image into grids similar to YOLO but has taken one step further to make use of default boxes to perform prediction. This can effectively produce tight bounding boxes which better match the shape of the object. An example of default boxes is illustrated in the image below:

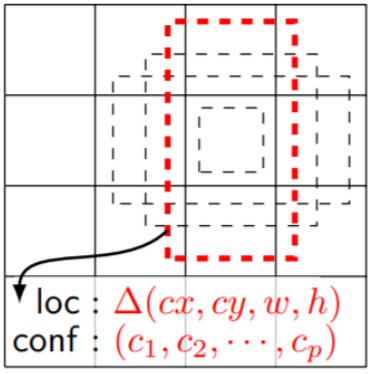


Figure 8: Default boxes with different aspect ratios are used to evaluate confidence for all classes at each image grid.

Adapted from: [24]

Even better, SSD introduced the idea of performing prediction at multiple layers, instead of just the last layer, so that the model has the ability to detect objects of different sizes. With that, the bounding box will have a variety of sizes and aspect ratios to better fit the object. SSD outperforms YOLO in terms of accuracy but runs slightly slower. On VOC2007 test, SSD achieved 74.3% mAP at 46 FPS [24].

3.1.3 Comparison between these object detection architectures

Table 1: Comparison between different object detection architecture. Adapted from: [24, 33, 34]

Object detection architectures	mAP	FPS
YOLO	63.4%	45
YOLOv2 288 × 288	69.0%	91
YOLOv2 544 × 544	78.6%	40
SSD300	74.3%	46

The table above summarizes and compares the performance of Faster R-CNN, YOLO, YOLO_v2, and SSD on the VOC2007 test dataset. YOLO is not chosen as it has a lower accuracy and inference speed compared to the other two architectures. Unfortunately, not many resources are available for quantizing and compiling YOLOv2 into Edge TPU model. Hence SSD will be chosen to be the architecture for this project since the pipeline to train, quantize and compile SSD is readily available online as well as the API by TensorFlow [23]. Backbone CNN networks will be reviewed and compared in the following section.

3.2 Convolutional Neural Networks Backbone for object detection (SSD)

Convolutional Neural Networks (CNN) is the most representative deep learning model as they are able to exploit the fundamental properties of natural signals which include translation invariance, local connectivity, and compositional hierarchies [20, 27]. In 2011, with the increase GPU computing speed and power and the introduction of ImageNet [35], it is made easier to train larger and deeper CNNx, huge improvement is achieved when AlexNet [36] won the 2012 ImageNet computer vision test [37] with an astonishing accuracy of 84.6% and being able to perform computer vision tasks that are previously deemed impossible. In 2014, VGG-16 surpassed AlexNet with an accuracy of 91.9% [38] by again using larger and deeper networks. However, as the need to have CNN models deployed on mobile platforms in real-time increases, intensive research is done to design a CNN model that is lightweight with minimal accuracy compromisation. In 2014 and 2015, GoogLeNet [25] and ResNet [39] is introduced respectively, with a much smaller size compared to VGG-16 [38] and also achieved a higher accuracy. More models are introduced which surpass the performance of these models or uses lesser computational power, but their design is a combination of main features from previous models. Since the aim of the project is to train a reliable model for real-time application on mobile platforms, only lightweight CNN models will be discussed and compared in detail to be the backbone of SSD, which are inception Nets [25] and MobileNets [26, 40].

The comparison of the mentioned CNN networks is visualized in the graphics below:

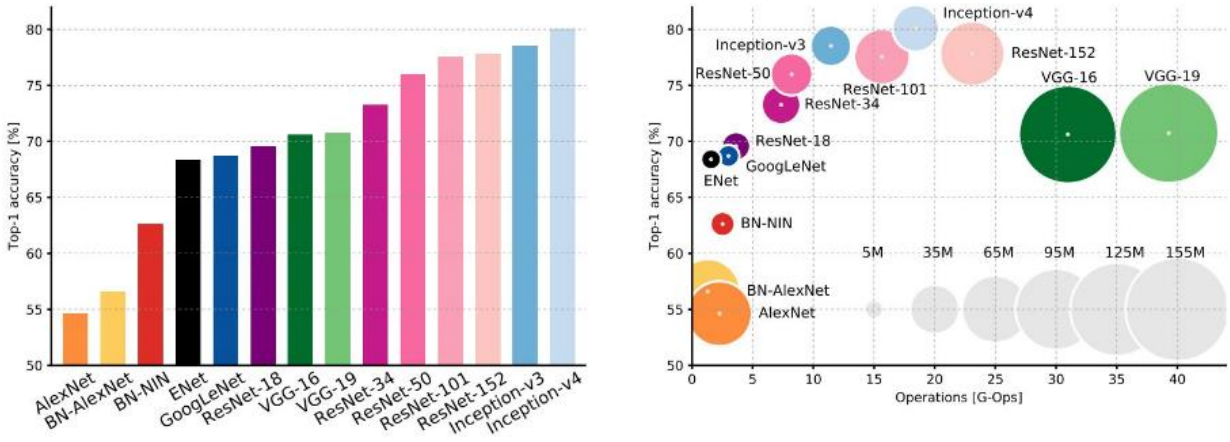


Figure 9: An analysis of Deep Neural Network Models for Practical Applications.

Adapted from:[41]

3.3.1 Inception Nets

The inception model is first introduced in GoogLeNet [25]. InceptionV1 architecture introduced the idea of having parallel filters of multiple sizes so that the model can classify objects regardless of their size in the image, without needing to increase the depth of the model. To make convolution more efficient and less computationally expensive, the input goes through 1x1 convolution to compress the channels before going through 3x3 or 5x5 convolution. To prevent the vanishing gradient problem, two auxiliary classifiers are placed at the middle layer of the model, where the loss from each auxiliary classifier is taken into consideration in the total loss (auxiliary loss as a weightage of 0.3) [25].

The architecture of Inception v1 (GoogleNet) is as shown in the picture below:

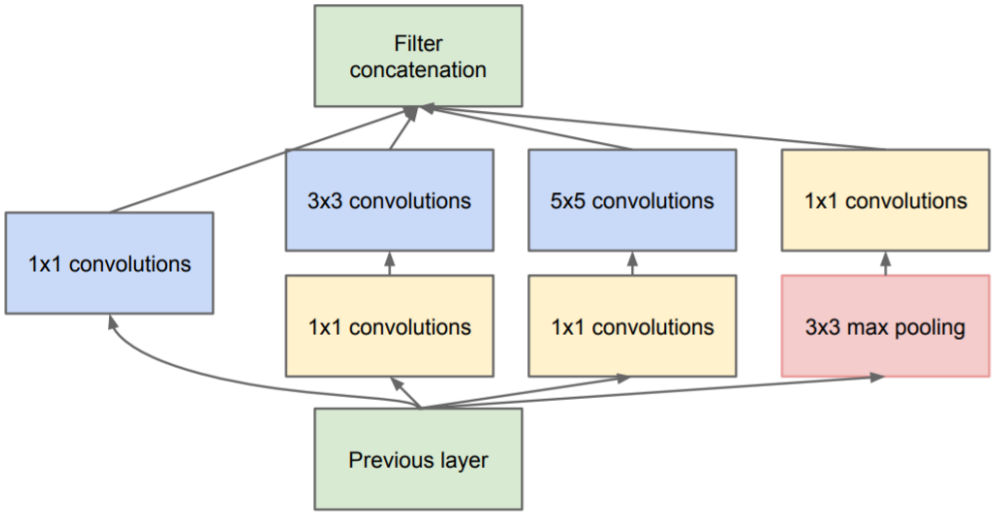


Figure 10: Inception V1 module with dimension reductions.
Adapted from:[25]

Inception V2 and V3 are introduced to improve from Inception V1. Inception V2 aims to reduce the representational bottleneck, as it is thought that the performance of the network would improve if the convolution operations did not alter the input dimension drastically. Inception V2 also explores methods to improve the efficiency of convolution operations so that it requires less computational power. The solution proposed by Inception V2 is by splitting a convolution with a big kernel size into two or more stacked

convolutions with smaller kernel sizes that will produce the same output size. This solution not only saves more computational power but also boosts the performance of the network [42]. The architecture is as shown below:

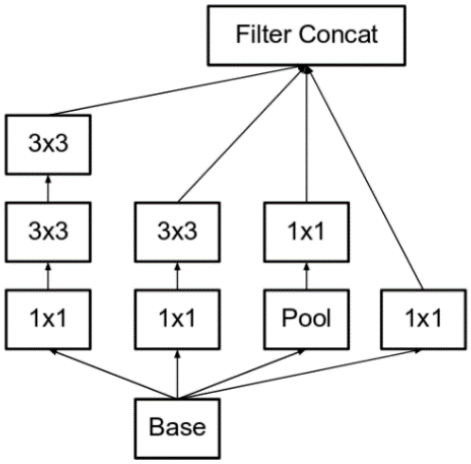


Figure 11: Architecture of Inception V2. 5x5 convolution is replaced by two 3x3 convolutions. Adapted from: [42]

They also found that splitting an nxn convolution into 1xn and nx1 convolutions to be 33% computationally cheaper, where the architecture is as shown below:

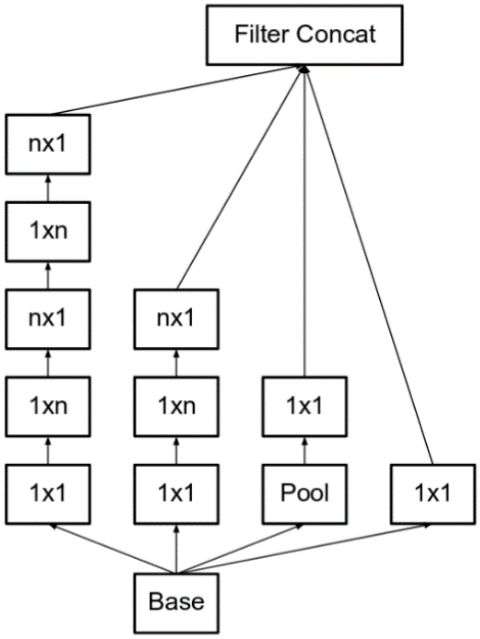


Figure 12: Inception V2 architecture. Splitting nxn convolutions into 1xn and nx1 convolutions. Adapted from: [42]

On the other hand, Inception V3 tries to improve the training process, as it is found that the auxiliary classifiers only contribute near the end of the training process, and acts more like a regularizer similar to BatchNorm and DropOut operations. Hence, RMSProp Optimiser, BatchNorm, and Label Smoothing are added to tackle the problem mentioned [42].

6.3.2 MobileNets

In MobilenetV1, depthwise separable convolution is suggested, which is to turn standard convolution into depthwise convolution followed by pointwise convolution.

An example of the depthwise convolution is illustrated below:

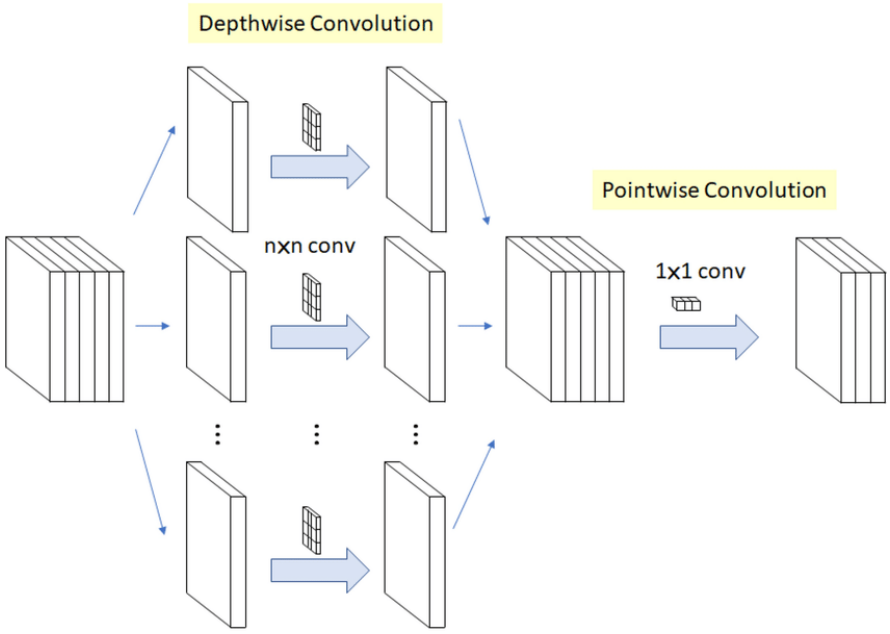
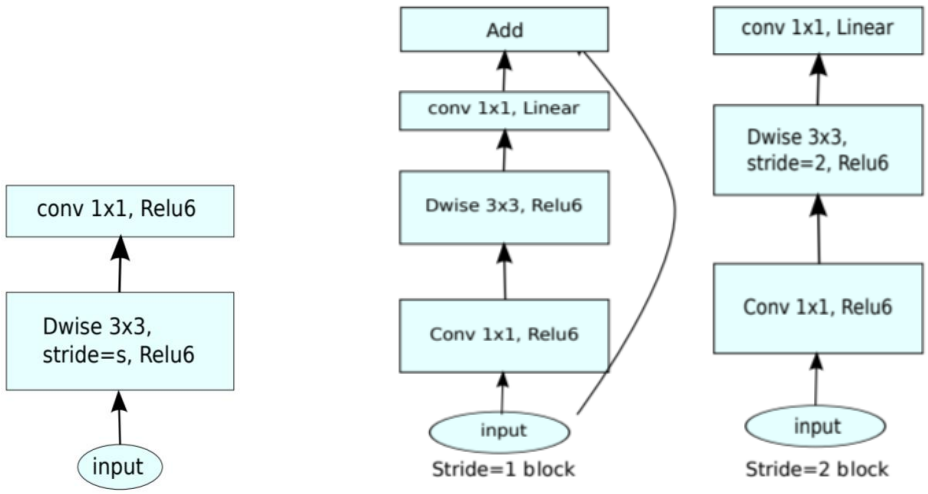


Figure 13: Illustration of depthwise separatable convolution. Each input channel is convolved with an nxn filter in parallel, where the output is concatenated, followed by a pointwise convolution to achieve the desired number of output channels.
Adapted from: [43]

By doing this, the hyperparameters of the model and the computation power required are significantly reduced compared to standard convolution. When the kernel is 3×3, the computation power is reduced 8 to 9 times with only a small reduction in accuracy. Two additional hyperparameters, Width Multiplier α and Resolution Multiplier ρ are also included to give flexibility to the model [40].

In MobileNetV2, other than using the ideas suggested in MobileNetV1, the idea of bottleneck layers is introduced. The bottleneck layers are designed to be linear as it is observed that non-linearity will cause a loss of information. Different from residual blocks

in Resnet, the shortcuts are placed in between bottleneck layers, in order to tackle the vanishing gradient problem which is common in large deep learning networks. Hence, the building block of MobileNetV2 consists of pointwise convolution that expands the channel of the previous layer, followed by depthwise convolution to extract features and lastly another pointwise convolution that compresses the channels of the previous layer, as shown in the image below:



(a) *(b)*

Figure 14: Comparison between the bottleneck module in MobineNetV1 (a) and in MobineNetV2 (b). Adapted from: [26]

With this, the number of parameters of the model is efficiently decreased, hence the memory usage is also reduced [26].

6.2.3 Comparison between these backbone CNN networks

The table below compares the number of parameters and number of layers of different deep learning architectures, and can be shown in the table below:

Table 2: Comparison between different backbone CNN network

Deep learning architecture	Number of parameters ($\times 10^6$)	Time per inference on CPU + USB Accelerator (ms) [44] (before adding SSD)	Performance (mAP)
InceptionV1 (GoogleNet)	6.8 [25]	3.4	43.9% (ILSVRC 2014) (mAP @ .5) [25]
MobileNetV1	4.2 [40]	2.4	19.3% (COCO dataset) (mAP @ [.5,.95]) [40]
MobileNetV2	3.2 [26]	2.6	22.1% (COCO dataset) (mAP @ [.5,.95]) [26]

Referring to the table above, it can be observed that MobileNetV1 is fastest when deployed on CPU + USB Accelerator, while MobileNetV2 requires the least number of parameters, i.e., requires the least memory. The accuracy of InceptionV1 (GoogleNet) could not be compared against the MobileNets as the performance is evaluated via different evaluation metrics. However, since InceptionV1 (GoogleNet) requires greater memory and computational power, MobileNets are preferable. More specifically, MobileNetV2 is a better choice compared to MobileNetV1 as it has a greater accuracy and lesser parameters, with slightly greater inference time which is negligible. [45] has also utilized MobileNetV2 + SSD on Raspberry Pi 4 with Google Coral USB for real-time object detection, and successfully achieve an accuracy of at least 70% via testing real-time in different environments.

3.3 Other related works

Many works have been attempted for overhead person detection. Authors in [10] developed DPD Net to perform person detection via depth overhead images. DPD Net consists of two fully convolutional encoder-decoder blocks built with residual layers, where the encoder proposes a pixel-wise confidence map, which is then fed to the decoder to refine the confidence map. The architecture of DPDNet is illustrated below:

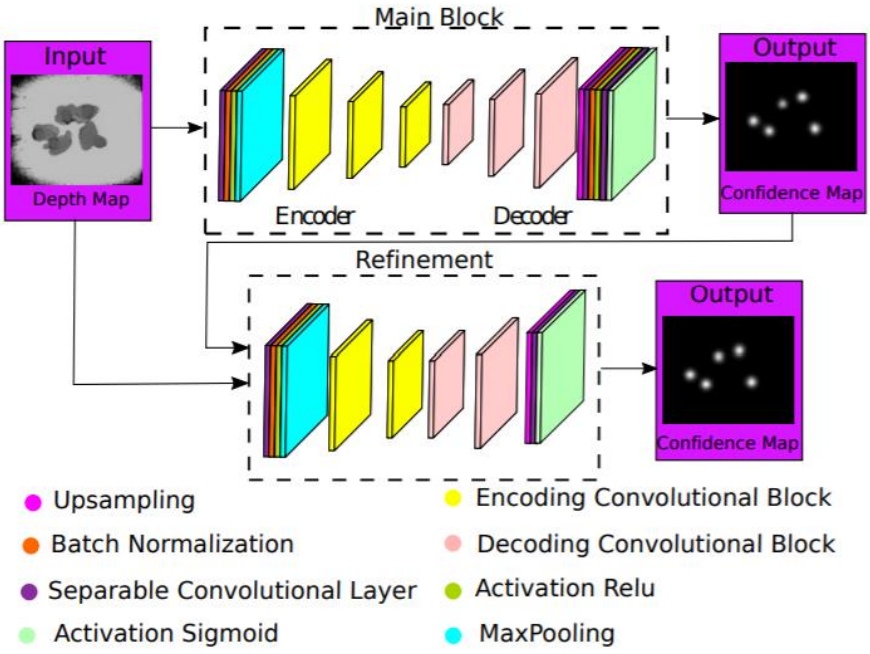


Figure 15: Architecture of DPDNet.
Adapted from: [10]

DPD Net is trained on the GOTPD1 database, then evaluated on other datasets which include the MIVIA database, Zhang 2012 database, and some self-recorded data, and it is able to achieve great performance of 99% accuracy. However, for more complicated datasets such as the TVHEADS dataset, the model requires fine-tuning using 80% of the TVHEADS dataset, before it is able to achieve an accuracy of 99% [10]. From the sample images shown, although hard scenarios such as people walking close together and people carrying large objects, the background is fairly simple and not cluttered, which simplifies the detection process. Also, depth images are not suitable to be used in this

project as ELIDEye has to track each individual that is detected, which is not viable since each unique person ID could not be identified if depth images are used.

Authors in [46] also used depth images and designs a two-stage method to detect human heads. The first stage utilizes the geometric property of a human head, which includes radius and depth information, compares it with the depth images, and proposes as many detections as it can. In the second stage, a convolutional network is applied to extract the features and output descriptor of the input image, as illustrated below:

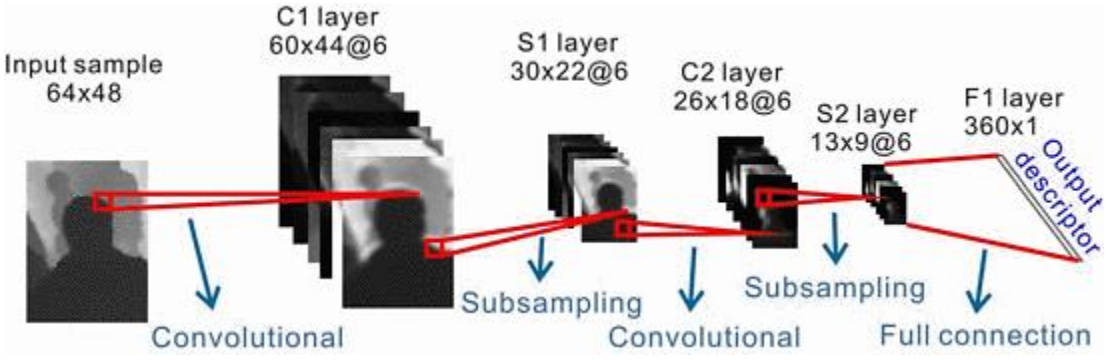


Figure 16: Architecture of the convolutional network that extracts the feature of the input sample. Adapted from: [46]

These output descriptors will then be fed to an SVM classifier, in order to reduce and validate the detections proposed in the first stage. This architecture is evaluated on self-collected data and achieves an average miss rate of 0.046 and false-positive-per-image of 20.48 [46]. This is a bad performance which reflects that this architecture is not feasible. Also, as mentioned earlier, this project will work on RGB images due to the need for tracking after performing detection.

Authors in [11] on the other hand used Aggregated Channel Features (ACF) with Adaboost Classifier to perform top view person detection and counting. Training data are self-collected by installing an overhead camera. To ensure the model is able to run real-time, the detection scale and camera resolution are optimized, while utilizing asynchronous multithreaded design for the deployment of the model. However, also due to the asynchronous multithreaded design, a great amount of memory is needed to be able to perform computation as image data that are in the queue need to be stored until

it is processed. The authors tackle this problem by designing the allocation of memory to enable the reuse of memory. The allocation of memory proposed is illustrated below:

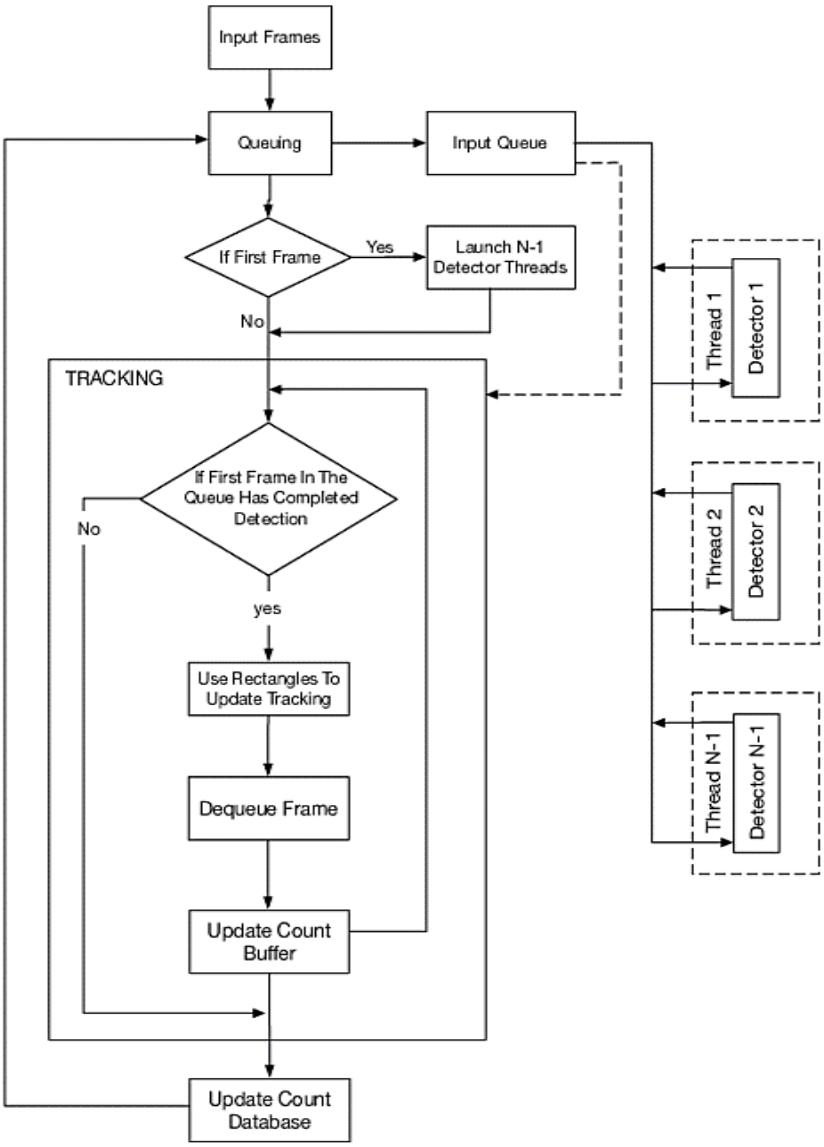


Figure 17: Multithreaded design for memory allocation

Adapted from: [11]

In good lighting and static background, the model is able to achieve an accuracy of 78% but is only able to achieve 50% accuracy otherwise. Also, the inference time is dependent on the number of people in the scene, that is the more people the greater the inference time [11]. A good model should have an inference time that is independent of the number of detection [10], hence this architecture is not suitable for this project.

3.4 Transfer Learning

Transfer Learning is defined to be the transfer of knowledge from a task that is related and learned by the model [47]. It is believed that transfer learning is most effective when the features learned by the base network from the base dataset are general so that this general knowledge can be transferred and improved to solve a niche problem [48]. In other words, transfer learning is based on the assumption that the same parameters are shared between the tasks or domains [14]. The first few layers of most deep neural networks that are trained on natural images are not learning features that are specific to the target task. Instead, general features are learned and the first few layers act like Gabor filters or color globs [48], as shown below:

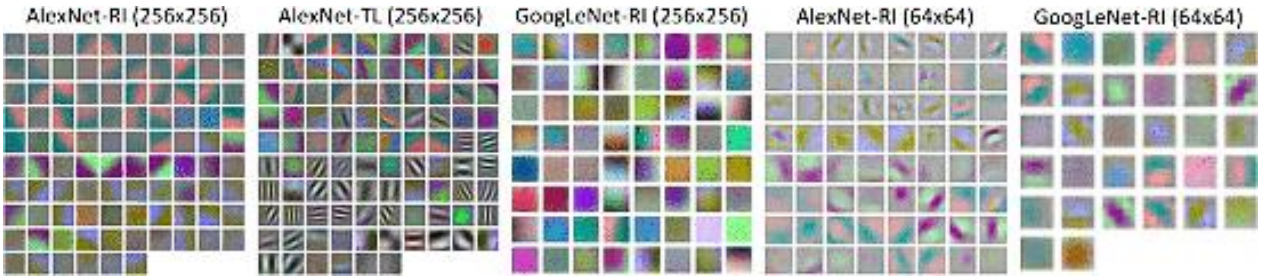
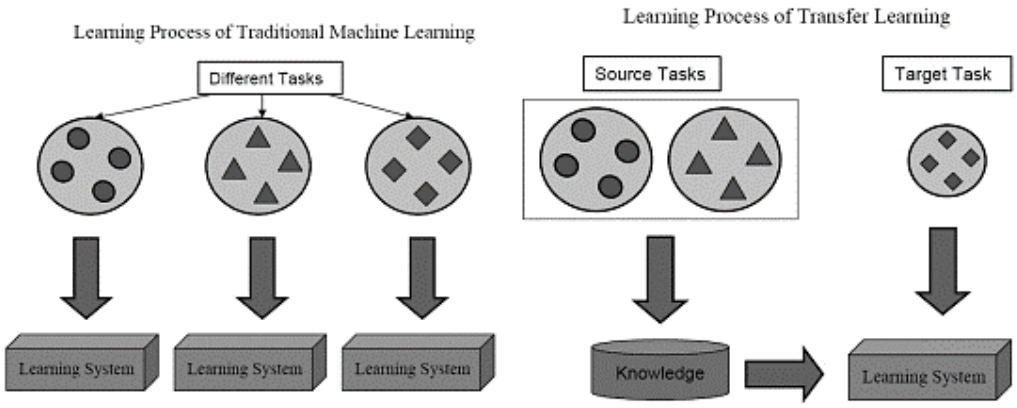


Figure 18: Visualization of first layer convolution filters of CNNs.
Adapted from: [49]

It is also found that while transfer learning is beneficial for most of the cases, applying transfer learning might bring a negative impact to the model performance if the source and target domains are too dissimilar [48]. Different from multitask learning which tries to learn to perform the source and target tasks simultaneously, transfer learning focuses on learning to perform the target task [14]. The overview of transfer learning can be illustrated in Figure 19.



(a) *(b)*

Figure 19: Comparison between traditional machine learning (a) and transfer learning (b). Adapted from: [14]

There are three categories in transfer learning, which are inductive transfer learning, transductive transfer learning, and unsupervised transfer learning. Inductive transfer learning is for applications where the target task and the source task are different but related, regardless of whether the domains are the same or not. Transductive transfer learning is for applications where the task remains the same, but the target domain is of different data distributions of the source domain. Unsupervised transfer learning, as the name goes, is for applications where tasks might be different but labeled data is unavailable [14]. These three categories can be further split into subcategories, as shown in the mind map below:

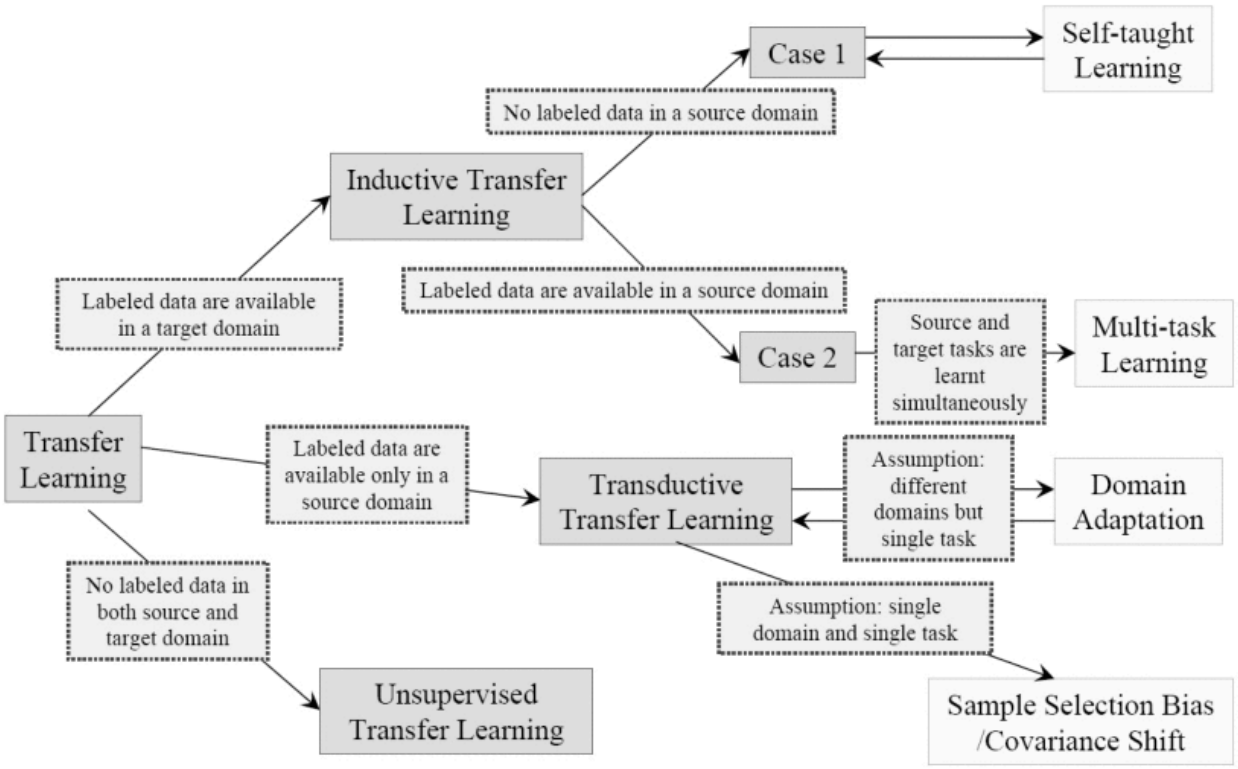


Figure 20: Different settings of transfer learning.
Adapted from: [14]

For this project, the pre-trained MobileNetV2 SSD model is trained on the COCO dataset, which is a generic dataset with a total of 2.5 million labeled instances in 328k images over 91 objects types [50]. The target task, on the other hand, is a single class detection to be trained on labeled top view human datasets. Hence, it can be deduced that inductive transfer learning will be the most suitable transfer learning method to be implemented in this project.

Transfer learning has been implemented for various tasks and proved to be successful. Authors in [51] propose to approach Alzheimer's neurological disorder detection via transfer learning. The proposed method can be summarized in the illustration below:

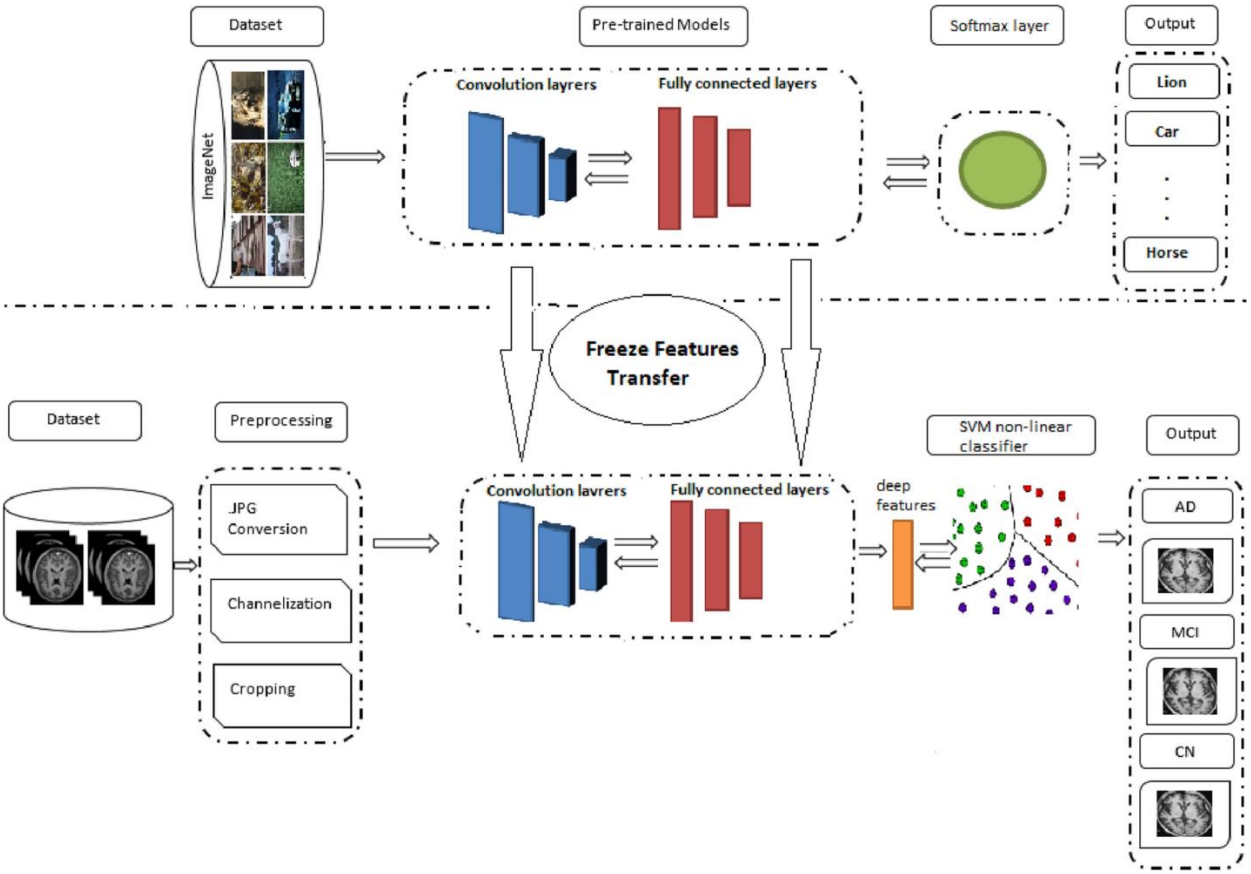


Figure 21: Flow of freeze features approach proposed.
Adapted from: [51]

As shown in the illustration above, the authors freeze the weights of pre-train models to extract visual and salient features of the Magnetic Resonance Imaging (MRI) images. Freezing the weights of the first few layers of the pre-trained model give the convenience to capture generic image features, before the features are feed to the non-linear SVM classifier. The authors claim that this approach is less computationally complex hence gives faster training as compared to other fine-tune approaches. After employing the proposed method on eleven CNN architectures such as AlexNet, GoogLeNet, VGG-16, VGG19, ResNet18, ResNet50, ResNet101, MobileNetV2, InceptionV3, Inception-ResNet-V2, and DenseNet201, all models are able to perform well and achieve an accuracy of 73% to 99% [51]. Freezing the weights of the first few layers of the model is a simple yet effective approach to ensure that the model is able to learn from a good starting point and able to converge with high accuracy. This method is also able to be implemented via TensorFlow API by indicating which layers to freeze in the config file.

The authors in [49] have a very similar approach as [51] to tackle thoraco-abdominal lymph node (LN) detection and interstitial lung disease (ILD) detection. Instead of freezing the weights, they modified the learning rate of all CNN layers excluding the last layer to have a learning rate ten times smaller. This proposal is experimented using different CNN architecture which includes AlexNet, CifarNet, GoogLeNet, Overfeat, and VGG net. It is concluded that transfer learning from a big generic dataset such as ImageNet has consistently achieved better performance, as compared to if the CNN models are trained from scratch using a small dataset [49]. This method is also simple and is based on the same theory as freezing the model weights. However, due to the restrictions in the TensorFlow API, the learning rate that is defined in the config file is imposed on the whole model, and there does not exist a setting to individually set the learning rates for each layer in the model. This method will be feasible if the model is self-built, but not possible to be done in TensorFlow API.

Collecting new data to retrain a model every time the target task or target domain changes is very expensive. To tackle this problem, TrAdaBoost is introduced. TrAdaBoost is a transfer learning framework that only requires a small amount of the target dataset which will be merged with the source dataset during training. It is proven that TrAdaBoost is able to effectively transfer the knowledge from the source dataset to the target dataset, allowing the model to converge and achieve high accuracy in the unseen data [13]. Inspired by TrAdaBoost, authors in [12] proposed a similar approach to improve pedestrian detection in unseen environments. Although existing works on pedestrian detection are reported to be able to give great accuracy, the model might not be able to perform well in real applications. This is due to varying environment conditions such as lighting, pedestrian variation, site background, etc. Some papers suggest rebuilding a new model in the unseen environment or improve the model dynamically in the unseen environment, but this method is found to be computationally expansive in practice [12].

A two-step strategy is proposed in [12]. First, samples from the training dataset that is similar to the unseen environment are screened based on manifold learning via Isomap algorithm, as it is believed that there exists some correlation between the training dataset

and the unseen set. These screened samples will be merged with the new dataset collected from the unseen environment and would take up most of the entire training dataset, thus increasing the size of the unseen dataset. Second, a new classification model based on transfer learning is proposed, named ITLAdaBoost. Different from AdaBoost, rather than only updating the weights of incorrectly predicted samples, ITLAdaBoost added a few modifications include the feature of weighted training for seen and unseen data. If a seen data is incorrectly classified, it is assumed that this is due to the seen data is conflicting with the unseen data, hence its weight will be reduced. On the other hand, if unseen data is incorrectly predicted, the model will be forced to focus more on this data during training by increasing its weight. With this proposed method, the classifier is able to achieve a better performance in unseen environments [12]. This method is much complicated as it involves multiple steps to achieve. Also, ELID is only concerned with its performance on the deployment site as the overhead camera location will be fixed, and the environment will likely not have great changes as the application is in indoor settings. Hence, this method might not be suitable for this project.

From the literature review, it can be deduced that transfer learning is suitable for this project, as the pre-trained network is trained on a much more general yet similar dataset, making knowledge transfer possible to happen. From all transfer learning methods that are reviewed, inductive transfer learning suits this project most as the source and target task are dissimilar. It will be implemented by freezing the first few layers of the pre-trained model, which can be easily done in TensorFlow API. However, experiments need to be done to find out how many layers should be frozen to give the optimal solution between fast training and high accuracy.

3.5 GAN in data/image augmentation

Having insufficient training data is a common problem in deep learning. Although many regularisation methods are introduced such as dropout [52] and batch normalization [53] to prevent overfitting and encourage the generation of the model, it is always thought that having sufficient large and varied data samples will improve the network performance better [54]. Ever since GAN models are proven to be able to generate realistic images, a lot of research is done on incorporating GAN to perform data augmentation which is previously not achievable via traditional methods.

The first GAN model is introduced in 2014 by Ian Goodfellow and his colleagues using the idea of having two ‘players’ competing with each other in a min-max two-person zero-sum game, where these two ‘players are the generator and the discriminator [17]. The generator takes noise as input and generates samples, aiming to fool the discriminator by creating realistic samples. The discriminator receives real and fake examples and aims to be able to distinguish between the two. The authors in [17] have proved that by having this competition, the generator is able to learn the data distribution well and the generated samples are indistinguishable from real data [55]. The GAN architecture in [17] used fully connected layers and hence not translation invariance and requires huge memory if the depth and width of the model are increased. Therefore, DCGAN which stands for “Deep Convolutional GAN”, is introduced [56]. As the name suggests, instead of using fully connected layers, DCGANs use stride convolutions to allow spatial up-sampling operations for generations and down-sampling operations for discriminations.

3.5.1 Multi-Conditional GAN (MC-GAN)

Conditional GANs are introduced as a better approach to multi-modal data generation, where both the generator and discriminator networks are made class-conditional [57]. Since this project is a single-class problem, having a class-conditional generation will not be helpful. Instead, as it is hoped that the newly created human instances can have the same or similar context as the background. In other words, for this project, it is hoped that the GAN model is conditioned on the background context instead of the class information. MC-GAN, i.e. Multi-Conditional GAN [58], does exactly what is desired.

The architecture of MC-GAN is illustrated below:

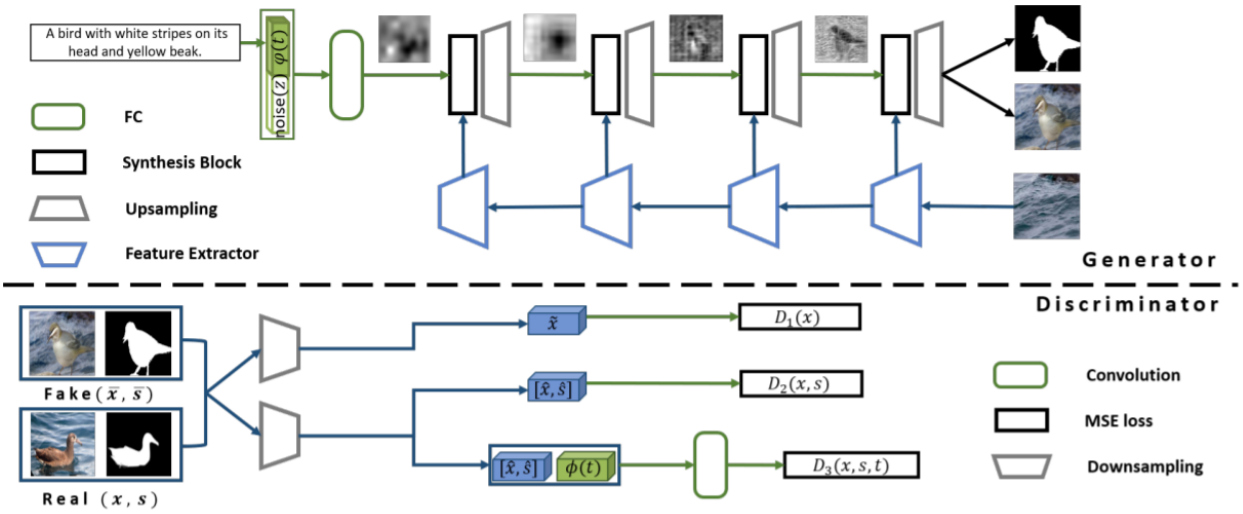


Figure 22: MC-GAN architecture.

Adapted from:[58]

The generator encodes a text sentence input describing the object to be generated as input, concatenates it with a noise vector, and applies the product to a fully connected layer to form a seed feature map. Background feature is also extracted from the base image to place the generated object using convolution and batch normalization [53, 58]. To incorporate background information of the base image to the output image, the concept of a synthesis block is introduced. The architecture of a synthesis block is illustrated in Figure 23.

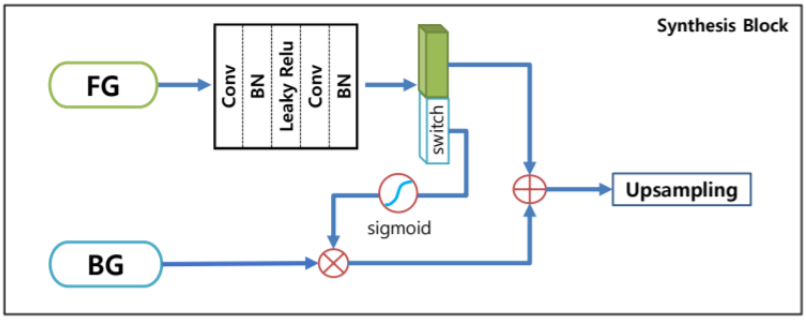


Figure 23: Architecture of the synthesis block in MC-GAN.
Adopted from: [58]

As shown in the figure above, a series of synthesis blocks will take in both the foreground feature (seed feature map or feature map from previous layers) and the background feature to generate images with their corresponding segmentation mask. Hence, the discriminator will then not only need to identify fake images but also fake segmentation masks and mismatching text descriptions [58]. MC-GAN is proven to be able to generate realistic images which match the text description and have the base image background context when trained using the Caltech-200 bird dataset [59] and the Oxford-102 flower dataset [60].

3.5.2 CycleGAN

CycleGAN is initially introduced to tackle the problem of requiring paired training data in image-to-image translation. Image translation is a computer vision problem that aims to generate an image with controlled modification of another given image. Before CycleGAN is introduced, image pairs are required to train the model, which is hard to prepare and sometimes impossible to obtain. Hence, CycleGAN presents an unsupervised training method that uses a collection of images from a source and target domain that need not be related in any way.

CycleGAN made it possible by having two generators and discriminators that interact with each other. One generator takes images from the source domain and learns to augment the images to the target domain, while its discriminator aims to differentiate whether or not the images it gets are from the target domain. The second generator and discriminator then do the opposite. The key difference of CycleGAN is the inclusion of cycle consistency loss which ensures that the modification can be cycled, by calculating the L1 norm [61] of the input photo and the generated photo. What this means is if an image is passed to the first generator, then the output is passed to the second generator, the final output of the second generator should be the same as the original picture [62].

The concept of cycle consistency loss is very well illustrated in the image below:

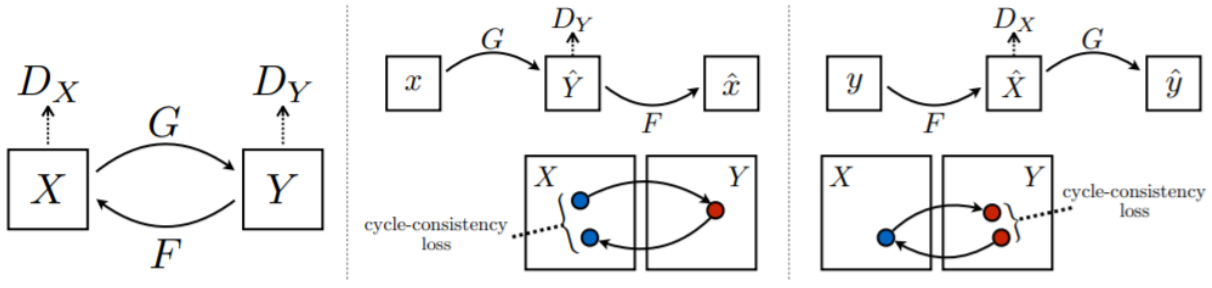


Figure 24: (a) the relationship of the two generators, (b) concept of cycle consistency loss. Adapted from: [62]

CycleGAN is also used to perform data augmentation in computed tomography (CT) segmentation tasks [63]. CycleGAN is trained to transform contrast CT images to non-contrast images [63], where contrast refers to the intake of substances that cause the particular organ or tissue seen more clearly [64]. Different augmentation method is tested, including no augmentation, standard augmentation, histogram equation augmentation, and CycleGAN augmentation, then segmentation performance of the models trained on dataset undergoing these augmentations are compared. It is shown that CycleGAN augmentation improves the performance in general, especially for the kidney model for non-contrast segmentation [63].

3.5.3 StyleGANs

While GAN is able to generate realistic images, they are not able to change explicit features. StyleGAN is an extension to the progressive GAN architecture aiming to regulate the output. Similar to the baseline progressive growing GAN, StyleGAN generates the simulated image sequentially via the incremental expansion of the model to start from a small resolution and up-sampling it to a huge resolution, but StyleGAN uses bilinear sampling instead of nearest neighbor sampling. The idea of Adaptive Instance Normalization (AdaIn) is introduced in StyleGAN. AdaIn originates from batch normalization and extends from Instance Normalization, where it receives the feature from previous layers, and controls the style of the images generated using the Affine Transformation module inputted [65].

Also, instead of using a point from a latent space as input, StyleGAN uses two sources of randomness to generate images, which are the standalone mapping network and noise vectors. The mapping network aims to encode the input latent vector into an intermediate latent space. The input latent vector should consist of the probability density of the training data and is generated starting from a learned constant. On the other hand, the noise vectors are just a single-channel picture with uncorrelated Gaussian noise, which are added at the last to induce stochastic details to the generated output images without affecting the overall perception of the image [65].

StyleGAN is proved to be able to generate realistic images with controllable variation and style transfer when trained on Celeba-HQ and FFHQ datasets. It is found that varying the noise image introduces changes to the generated image without affecting the overall appearance and pose, which is different from previous GAN models where changing the noise vector will result in totally different generated images. This indicates that the StyleGAN managed to separate different aspects of the images, hence able to modify stochastic variations and high-level attributes while maintaining the overall identity [65]. Similar to CycleGAN [62], StyleGAN [65] is also used to perform data augmentation in CT images in [66]. Authors in [66] propose to use a pre-trained StyleGAN network to

transform the Magnetic Resonance Imaging (MRI) images to CT images using limited sample images. It is proved that via this method, the CT motion artifacts classification task has achieved an improvement of 5.59%.

StyleGAN2 [67] is an advancement from StyleGAN. To remove droplet artifacts generated by StyleGAN, Adaptive Instance Normalization (AdaIN) is reconstructed as Weight Demodulation in StyleGAN2. This is because it is believed that AdaIN destroys information found in the relative feature magnitudes, causing the generator to create a strong, localized spike which results in the droplet artifacts. The addition of noise and bias is also shifted to be outside the style blocks to give more predictable results. This change of architecture is found to be able to also ease the parallelization of the computational path, leading to a 40% increase in training speed [67]. The change of architecture is visualized in the image below:

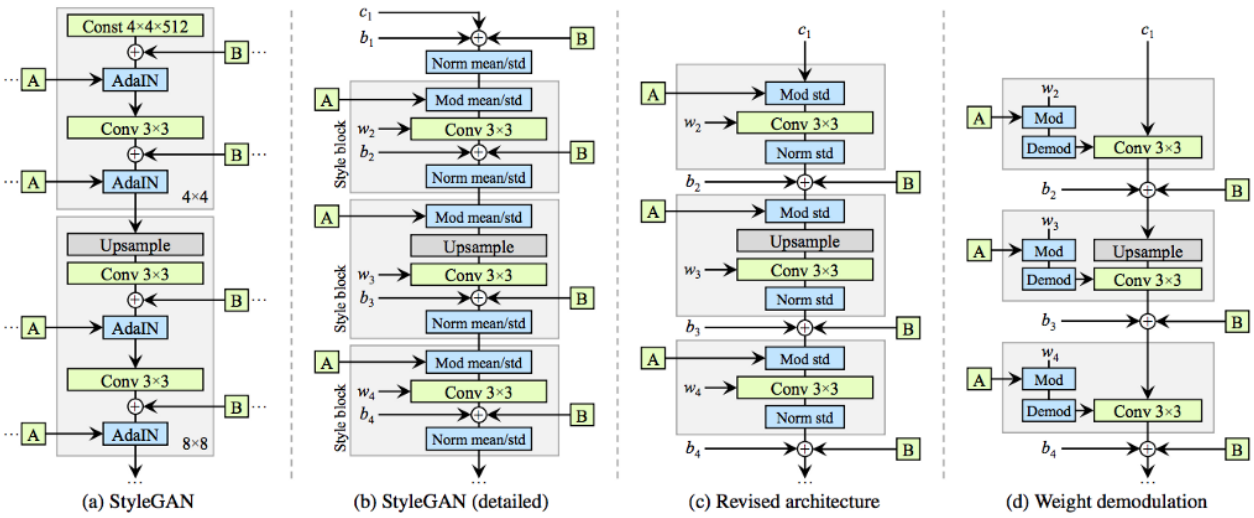


Figure 25: Change of architecture from StyleGAN to StyleGAN2.
Adapted from:[67]

Progressive growing is also removed to tackle the issue with StyleGAN having a strong location preference for certain features. Inspired by MSG-GAN [68], StyleGAN2 uses multiscale image generation via skip connections between lower resolution maps to the

output layer of the generator, similar to ResNet [67]. This proposal is illustrated in the image below:

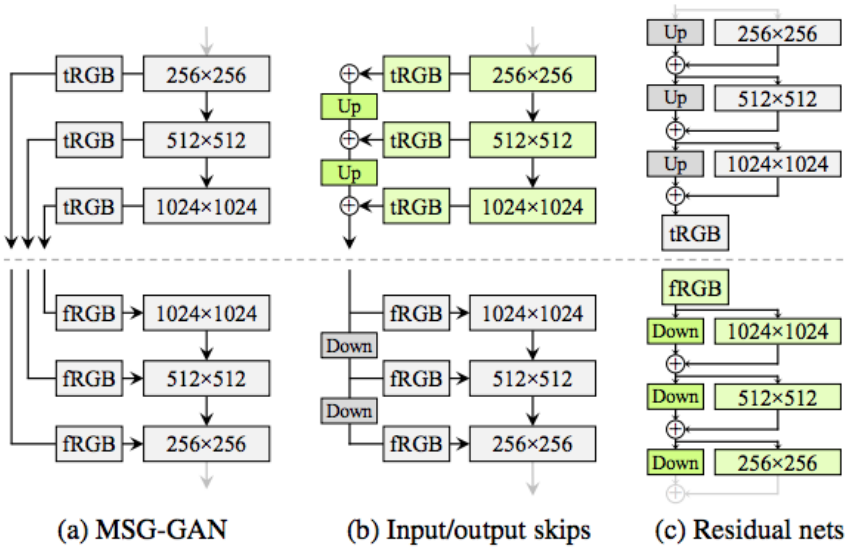


Figure 26: Inspired by MSG-GAN (a), StyleGAN2 eliminates progressive growing (b) with architecture similar to ResNet (c).
Adapted from: [67]

Path Regular Regularization is also introduced in StyleGAN2 by adding an additional loss term to the generator every 16 steps. This is to smoothen the latent space interpolation i.e., smoother transitioning from one image to another when the source vector changes [67].

StyleGAN2-ADA [69] in short, is StyleGAN with Adaptive Discriminator Augmentation (ADA). StyleGAN2-ADA aims to solve the problem with the overfitting issue of the discriminator in StyleGAN2 when the training dataset size is small. This issue is very critical as an overfitted discriminator will memorize all the training data and rejects all other images that are generated, causing the generator to diverge due to having very little feedback from the discriminator [69]. Data augmentation, which is commonly used to deal with overfitting problems, could not be used directly in GAN training as the generator will learn the augmented distribution and not give us the expected result. Hence, the technique called Stochastic Discriminator Augmentation to prevent the augmentations "leaking" to the generator [69], where its architecture is shown below:

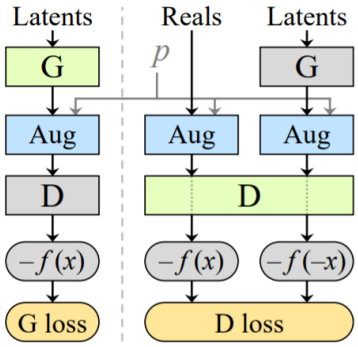


Figure 27: Stochastic discriminator augmentations.
Adapted from:[69]

However, this approach requires the distortions to be represented by an invertible transformation of the probability data distributions. Moreover, two plausible overfitting heuristics are also proposed to measure overfitting, which are rv that compares discriminator performance on training dataset against the testing dataset, and rt that computes the portion of the training data that discriminator takes as real images. These two heuristics are able to detect overfitting and determine how much augmentation is required. It is also demonstrated in the paper that StyleGAN2-ADA is able to give better performance in data-limited scenarios [69].

3.5.4 Data Augmentation GAN (DAGAN) [16]

Authors in [16] utilize GAN to perform data augmentation as well, hence DAGAN is introduced where its architecture can be found below:

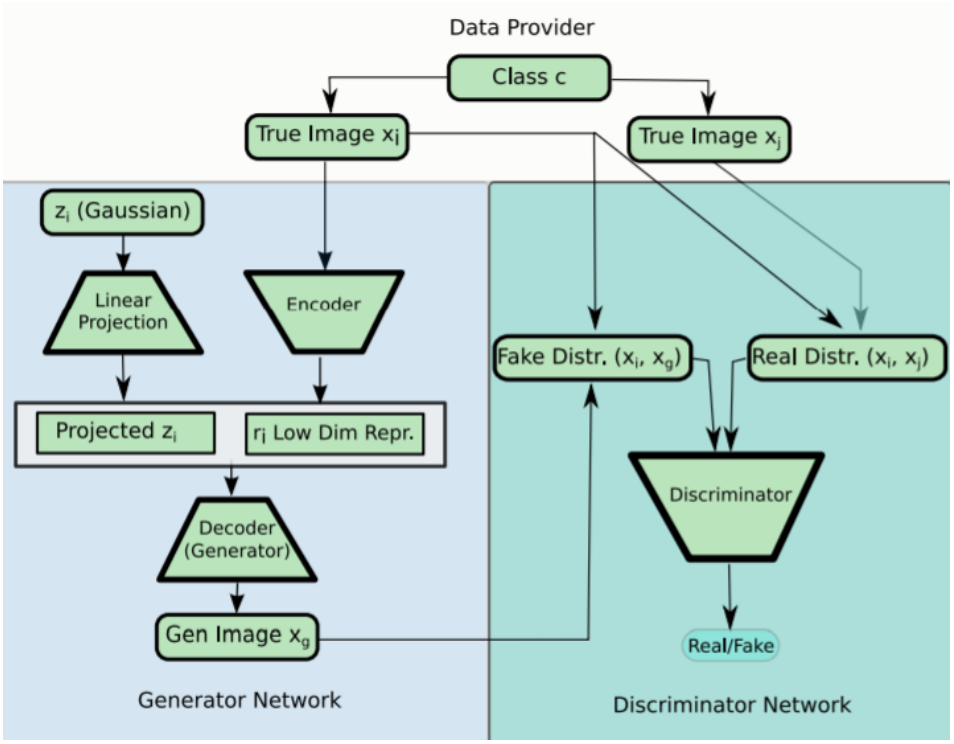


Figure 28: Architecture of DAGAN.
Adapted from: [16]

As shown in the image above, a few modifications are done from the vanilla GAN model. First, the input to the generator consists of some low dimensional information encoded from the original image to be augmented, concatenated with a randomized noise vector. Since the aim is to augment the image, it should be validated that the generated image is different from the original image, else the network will function as an auto-encoder. Thus, the original image is also fed to the discriminator to ensure that the generated image is related but different from the original real image. DAGAN also differentiates itself from CGANs as class information is not given. Instead, another real image from the same class is provided to the discriminator, so that the discriminator can learn to generalize to be consistent for all classes [16].

The generator is named as UResNet, as it is built using a combination UNet and ResNet. On the other hand, the discriminator follows the architecture of DenseNet, but replaced batch normalization with layer normalization as layer normalization will break the assumptions of the WGAN objective function. DAGAN is trained with Omniglot dataset, EMNIST dataset, and VGG-Face dataset. The results proved that DAGAN is able to generate realistic images that are different from the original real images yet still in the same class as the original images [16].

3.5.5 Comparisons between these GAN architectures

MC-GAN matches the requirement for the solution the most, however, in order to train an MC-GAN model, text descriptions of all human instances need to be prepared which requires a great amount of effort and time. Also, good performance is not guaranteed as MC-GAN is only tested with bird and flower dataset, while top view humans are much complicated. Taking these factors into consideration, MC-GAN might not be suitable due to the time limitation of this project.

CycleGAN on the other hand requires two different image domains to train on. An example application of CycleGAN is translating Monet and photo, zebras and horses, summer and winter, etc. However, there are not two domains to be used in this project, as the aim is to augment the human instances, e.g., change of hairstyle or pose, hence CycleGAN is also not very suitable for this project.

For StyleGANs, since StyleGAN2-ADA has the best performance and is suitable for small datasets, only StyleGAN2-ADA will be considered. StyleGAN2-ADA only requires the original images as training datasets and will learn the data distribution and generate images within that distribution. Even better, as the available number of images for top view human is quite little, StyleGAN-ADA is able to overcome this problem as mentioned in the paper, hence it is very suitable to perform data augmentation for this project.

Last but not least, DAGAN aims to perform data augmentation on the training images, hence matches with the objective of the project. However, in the paper, DAGAN is tested on simpler datasets such Omniglot dataset and EMNIST dataset [16]. Although it is proven that it works on more complicated datasets such as the VGG dataset [16], top view person images are much complicated and DAGAN might fail to learn the data distribution. This is because, in the VGG dataset, the face features are at fixed locations whereas, in top view human images, the body parts of the human will vary according to their pose and location relative to the overhead camera. Nevertheless, as the training pipeline of DAGAN is very simple, it is worth experimenting with it.

Hence, in short, both StyleGAN2-ADA and DAGAN will be experimented, and the architecture that is able to give the most realistic generated images will be chosen for the next stage i.e., to generate human instances to be incorporated in the training images for the Object Detector training. Further details will be explained in the Methodology section.

4.0 Methodology

4.1 Overview of methodology

The methodology can be summarized in the flow chart below:

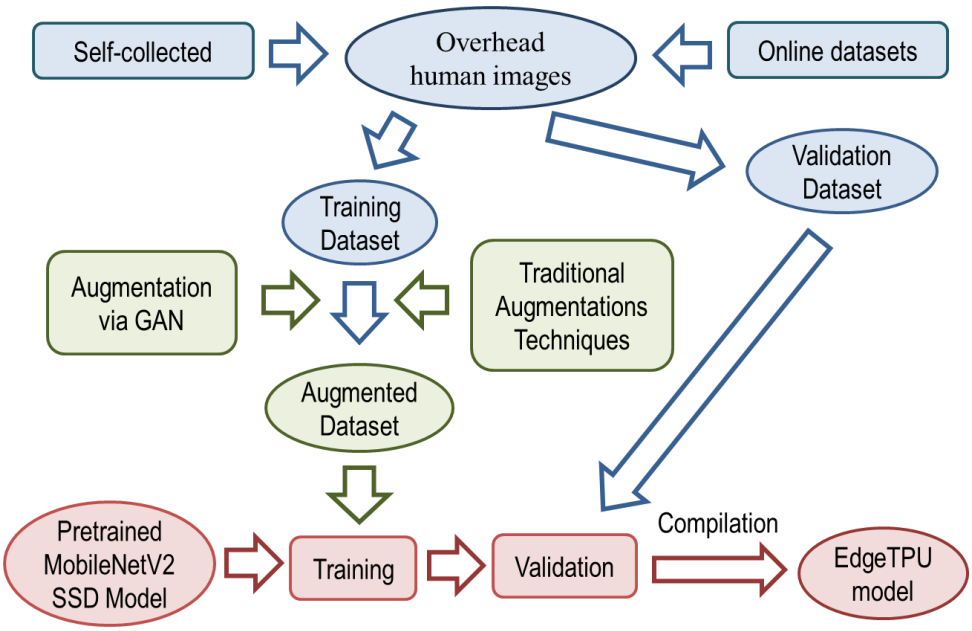


Figure 29: Methodology (flow of project)

The object detection model that is chosen is MobileNetV2 SSD [24, 26]. The model will be trained via supervised learning, thus labeled data is required i.e., top view images of humans with bounding box annotations. As TensorFlow API is used for the whole pipeline, data need to be preprocessed to fit the input requirements of the model, as well as to be compatible with the pipeline codes. Then, the images are split into training and validation datasets.

Data augmentation is also performed to increase the variation and size of the dataset so that the model trained is able to generalize better and have a better performance during validation and testing. In this project, various efforts are done in data augmentation to maximize its effect on improving the performance of the object detection model. If the GAN models can generate realistic images that are different from the original set of

training images, the generated human instances will be incorporated into the original images in the training dataset. After that, augmentation combinations will be applied to each image in the training dataset. Augmentation combinations refer to groups of one or more traditional augmentation techniques, which will be explained in detail in the later sections.

After the data and the environment to perform training is prepared, the object detection model will be trained for a few iterations, then compiled to an Edge TPU model. Then, testing is conducted to validate whether the model is able to achieve the target performance. If the trained model is not satisfactory, then the model will need to be retrained, after changing corresponding model hyperparameters, increasing or augmenting the training images. This process of preparation, training and testing will need to be repeated many times until the performance of the model is satisfactory i.e., able to give have accuracy and recall of 80% on unseen environments.

4.2 Data Collection

In data collection, images can be self-collected and can also be gathered through online datasets by other researchers. Image can be self-collected by installing an overhead camera which is programmed to capture images whenever it detects motion. This can be done via performing background segmentation for every frame and captures an image if the foreground area is greater than the preset threshold.

An example of the installation of the overhead camera is as shown below:

Figure 30: Installation of the overhead camera (circled in yellow).

For images without annotations, bounding boxes will need to be drawn manually via LabelImg application. For online datasets that come with annotations, if then the annotations need to be processed and converted to the required format i.e. [xmin, ymin, xmax, ymax, class] if the annotations are not in that format. For self-collected images, images that are collected with the same background will be grouped to be a dataset. The datasets that are collected will be discussed in detail in later sections.

4.3 Data Processing

4.3.1 Traditional Data Augmentation

Having insufficient size and variation for the training data imposes a great challenge in this project. This is because most person detection datasets are of frontal view such as Caltech Pedestrian Dataset [70], Penn-Fudan Database [71], and INRIA Person Dataset [9]. This results in a lack of variation of the training images since there are not many different backgrounds that can be used to collect training data. It is also hard to gather training data outside ELID due to privacy considerations. Hence, to increase the size of the training dataset, a few augmentation techniques are included, as summarized below:

Table 3: Self-written augmentations

Augmentation	Remarks/description	Number of variations
Rotation	Rotates image 90, 180, or 270 degrees clockwise.	3
Flip	Flip image vertically or horizontally	2
Hue	Shift hue of the image	2
Saturation	Increase or decrease image saturation	2
Brightness	Increase or decrease image brightness	2
Contrast	Increase or decrease image contrast	2
Noise	Add salt and pepper noise to the image	1
Blur	Blur image (Gaussian Blur)	1
Random erase*	Randomly erase a portion of the bounding box (human instances)	1
Grid mask*	Add grid mask of random grid size to image	1
Aspect ratio*	Change the aspect ratio of the image	1
Background rotation*	The foreground (humans) is extracted using the bounding box information, the background is rotated, then the foreground (humans) is added back in (not rotated).	1

* These augmentations techniques are added in later stages of the project.

It is hoped that by applying augmentation techniques, it can increase the variation and number of training images. This can be done by applying the augmentation before training, where the augmented image is assumed to be a new image / new training data. To increase the variation further, an image can be applied to one or more self-written augmentation techniques. For instance, an image can go through rotation, noise addition, followed random erase (combination 1). It can also go through flip, hue shift, followed by grid mask addition (combination 2). As both combinations (combination 1 and 2) of the augmentation techniques are different, it is believed that the augmented images are considered to be different from the model. This method effectively transformed the original image into two different augmented images. In other words, the size of the dataset can be increased via this method.

However, the sequence of the augmentation technique performed on the image will not result in a different augmented image. For instance, if an image goes through noise addition, random erase, followed by rotation (combination 3), the augmented image will be identical to the same image that goes through combination 1. Also, different variations of the same augmentation technique should not appear in the same combination, as it is equivalent to not applying any augmentations. For instance, increasing brightness then decreasing brightness would not result in a different image compared to the image before applying the augmentations.

Knowing these constraints of creating augmentation combinations, the number of possible combinations formed with n augmentation techniques per combination is computed and tabulated in Table 4.

Table 4: Number of possible combinations formed for n augmentation techniques.

Augmentation Techniques	Maximum number of augmentation techniques per combination (n)											
	one	two	three	four	five	six	seven	eight	nine	ten	eleven	twelve
Rotation	3	48	345	1470	4125	8004	10959	10590	7080	3120	816	96
Flip	2	28	174	632	1486	2364	2578	1904	912	256	32	-
Hue	2	24	126	380	726	912	754	396	120	16	-	-
Saturation	2	20	86	208	310	292	170	56	8	-	-	-
Brightness	2	16	54	100	110	72	26	4	-	-	-	-
Contrast	2	12	30	40	30	12	2	-	-	-	-	-
Noise	1	5	10	10	5	1	-	-	-	-	-	-
Blur	1	4	6	4	1	-	-	-	-	-	-	-
Random erase	1	3	3	1	-	-	-	-	-	-	-	-
Grid mask	1	2	1	-	-	-	-	-	-	-	-	-
Aspect ratio	1	1	-	-	-	-	-	-	-	-	-	-
Background rotation	1	-	-	-	-	-	-	-	-	-	-	-
Total	19	163	835	2845	6793	11657	14489	12950	8120	3392	848	96
cumulative total	19	182	1017	3862	10655	22312	36801	49751	57871	61263	62111	62207

As shown in the table above, a total of 103679 possible combinations can be formed if the combination can have one to twelve augmentation techniques. Hence, if it is desired to transform an image to k augmented images, k combinations will be chosen from this pool of augmentation combinations to be performed on the image.

4.3.2 Data Augmentation via GAN

Before using GAN to augment images, the GAN model needs to be trained. Referring to the papers on GAN that are reviewed, the target is always in the middle and occupies most of the space in the training images, which is not the case for the images collected for object detection. Hence, the human instances in the training images are cropped out using the bounding box annotations. To have some background information, the cropped images are a little wider than the bounding boxes. Then, images are filtered to keep only those that are not occluded to ease the learning by the GAN model and resized to be 256x256.

Two GAN models will be trained to perform image augmentation, i.e. DAGAN [16] and StyleGAN2-ADA [69]. DAGAN is coded for multiclass problems; hence the code needs to be modified a little for the data preparation as this project is a single class problem. Following the guideline in their GitHub page [72], the number of generator inner layers is chosen to be 3, the number of discriminator inner layers is chosen to be 5, the z dimension is chosen to be 100 and the dropout rate is chosen to be 0.5. On the other hand, StyleGAN2-ADA does not require any modification of the code. Following the guideline in their GitHub [73], mirroring of the datasets in the x-direction is enabled, while the discriminator augmentation pipeline is chosen to be pixel blitting, geometry, and color transformation. The StyleGAN model is trained from the pre-trained model on the FFHQ dataset [65].

After training, if the GAN models are able to generate realistic images that are different from the original set of training images, the generated human instances will be incorporated into the original training images. In other words, the original dataset will have one or more original human instances, and an additional synthetic human generated by the GAN model. The bounding boxes annotations of the images need to be updated to include the bounding boxes of the synthetic human instances. After that, traditional augmentation combinations are applied to these images, as described in the previous section.

4.3.3 Other data preprocessing

Following TensorFlow API, the bounding box annotations in XML format will then need to be compiled into a CSV file, then converted to tfrecords to be compatible for training in TensorFlow API. The data also needs to be split into training data and testing data, where the testing data need to be wisely chosen so that the performance metrics can reflect the actual performance of the model in unfamiliar environments and difficult scenarios.

4.4 Training of the object detection model

After setting up a docker environment that consists of TensorFlow Object Detection API [74], as well as the checkpoint and config file of the pre-trained MobileNetV2 SSD model [75], the training can start. If it is desired to perform transfer learning, the layers to be frozen need to be indicated in the config file. Other than reducing the batch size to suit the GPU capability, other model hyperparameters are remained to be the same as the pre-trained MobileNetV2 SSD model.

During training, the performance of the model can be monitored via TensorBoard. The interface of TensorBoard is as shown below:

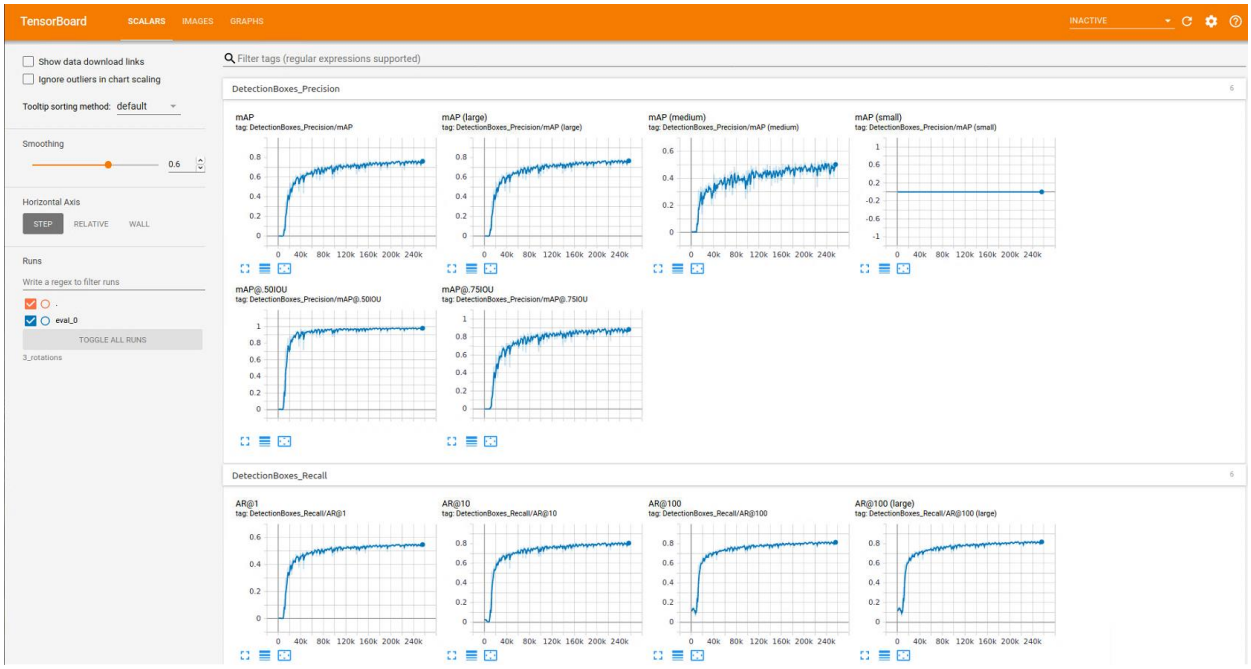


Figure 31: Interface of TensorBoard for COCO evaluation metrics

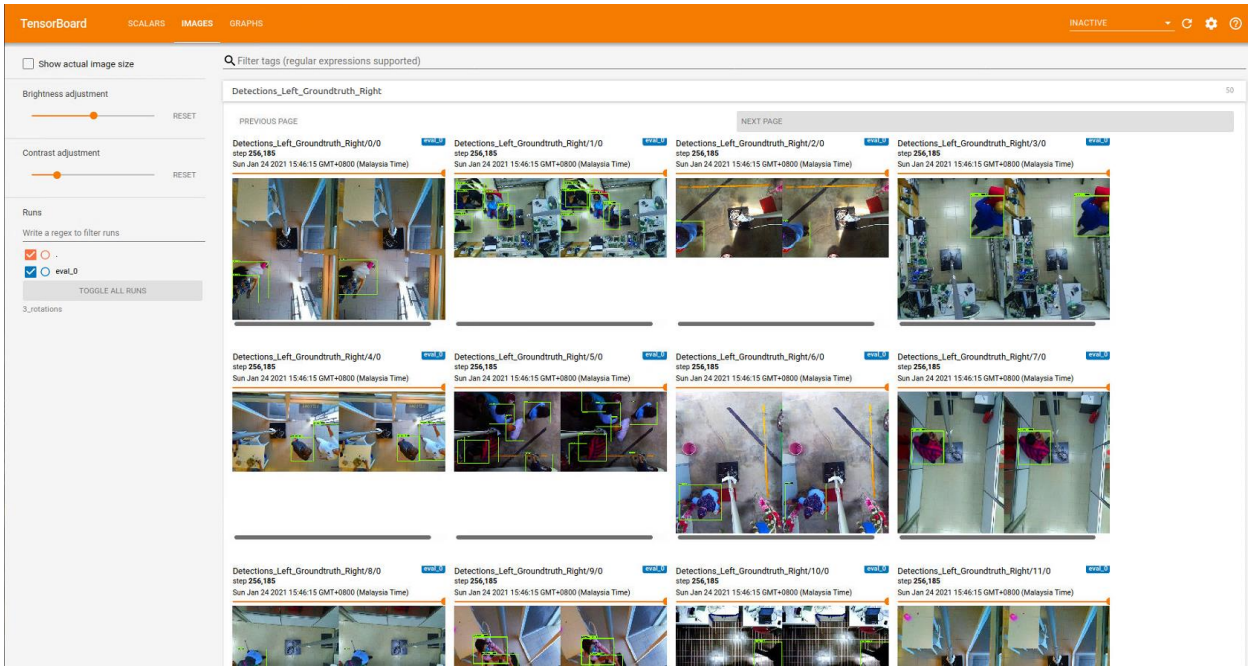


Figure 32: Interface of TensorBoard for visualization of model performance on test images

If the performance is not satisfactory, the training dataset needs to be modified or the hyperparameters of the model which include the number of freeze layers, learning rate, regularization weight, etc. need to be tuned.

4.5 Compiling, testing, and deploying of the trained model

After training the model, it needs to be compiled to an Edge TPU model following the following procedure which is taken from the documentation of google coral:

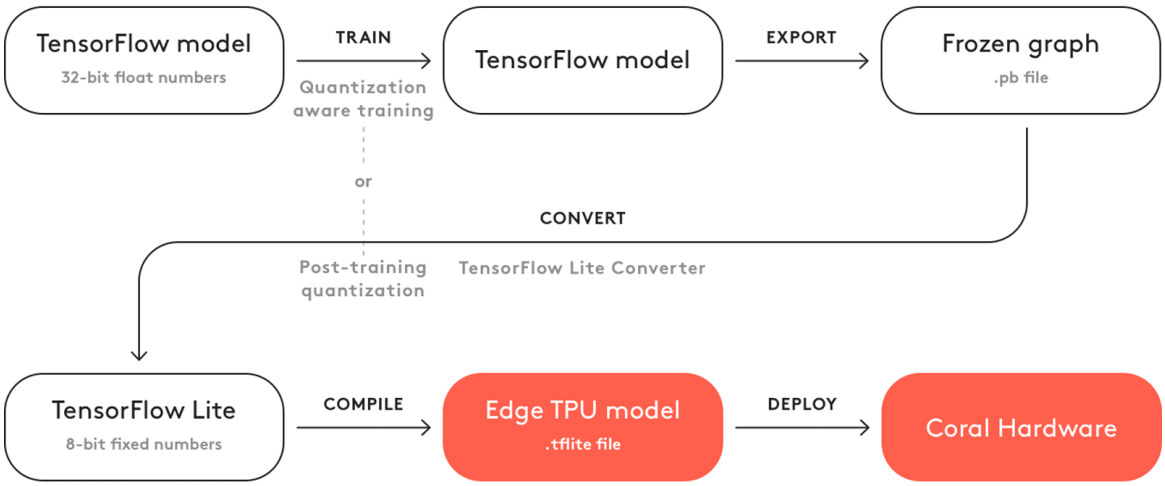


Figure 33: The basic process to create a model that is compatible with the Edge TPU. Adapted from [76]

As shown, the trained model will need to be converted to a frozen graph, quantized to be a TensorFlow Lite model with an 8-bit fixed number, then only compiled into an Edge TPU model. Fortunately, this complicated process is automated in TensorFlow API, hence all that needs to be done is to run the necessary scripts. The Edge TPU model will then need to be deployed on the Google Coral and tested real-time on the Raspberry Pi overhead camera, to ensure that the real-time performance of the model is satisfactory.

5.0 Experimental results

5.1 Datasets collected

Following the steps outlined in the previous section, a total of 7984 images with 14245 annotations are collected and labeled. The data collection via online resources is harder than expected. This is because the datasets for person detection available online are mostly of frontal views, such as Caltech Pedestrian Dataset [70] and INRIA Person dataset [9]. Online datasets that are collected are the Top View Person Reidentification dataset (TVPR) [77] and Top View Multi-Person Cafeteria (TVMPC) [78]. The details of the self-collected datasets and the online datasets are described in the following sub-sections.

5.1.1 Self-collected datasets

Overhead cameras are located at different locations at ELID to collect images whenever there is a change in the scene is detected. The image resolutions are 640x480 and 1280x960 where the former takes two-third of each dataset. The camera gain is set to 100, camera exposure is set to 6 and the image saturation is increased to 80, while the other settings are remained as their default values. All images collected in ELID are also self-labeled and require great amount of time and effort, hence the number of self-collected images is also limited. The datasets collected are grouped and named based on the location the images are taken.

Details of datasets collected in ELID are summarized in the table below:

Table 5: Self-collected datasets

Dataset Name	Total images	Total annotations	Sample images
ELID Cafeteria (EC)	1310	2408	

ELID DPD Entrance (EDE)	2225	2921	
ELID DPD Indoor 1 (EDI1)	90	148	
ELID DPD Indoor 2 (EDI2)	109	282	
ELID DPD Indoor 3 (EDI3)	323	1052	
ELID DPD Outdoor Bright (EDOB)	1049	1536	
ELID DPD Outdoor Dark (EDOD)	754	1227	

5.1.2 Online datasets

Top View Person Reidentification (TVPR) dataset, as the name suggests, is a dataset for person reidentification in scene monitoring. It consists of 640x480 RGB-D images collected from the top using Asus Xtion Pro Live [77]. Since this dataset is for person reidentification, the annotations given are not bounding box coordinates, but instead the identity of the person captured in each image. Hence, bounding boxes are added via LabelImg application. As the images are taken at a much higher FPS, the images that are very similar to each other are also filtered to prevent the model from overfitting to these images.

Similar to the TVPR dataset, the Top View Multiple Person Cafeteria (TVMPC) dataset [78] is not for person detection, but for person tracking purposes. Hence, although the bounding boxes are labeled, the bounding boxes are not accurate and there are some missed human instances that are not labeled. As the scene taken is while people are queuing for food, there are very little change from one image to the next. Hence, the images are filtered and relabeled.

Details of online datasets collected after filtering and labelling are summarized below:

Table 6: Self-collected datasets

Dataset Name	Total images	Total annotations	Sample images
Top View Person Reidentification dataset (TVPR) [77]	1893	1893	
Top View Multi-Person Cafeteria (TVMPC) [78]	231	2778	

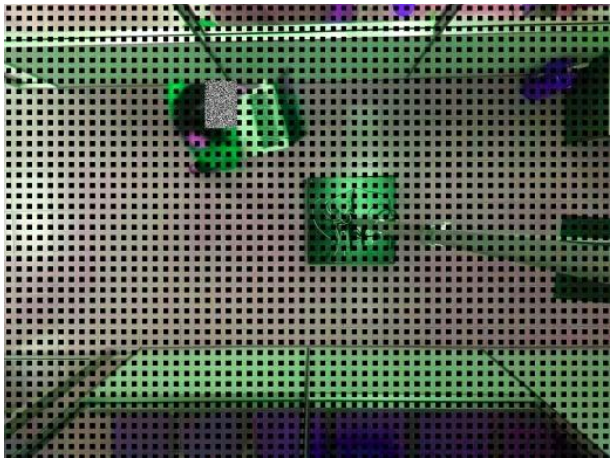
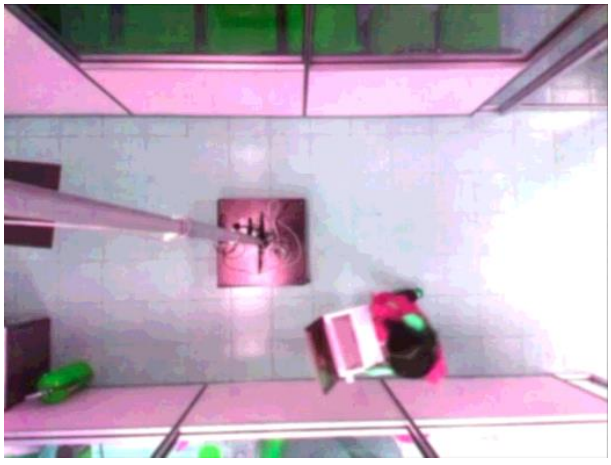
5.2 Traditional augmentation techniques

As described in the methodology, it is hoped that by applying unique traditional augmentation combinations, the variation and size of the training dataset can be increased so that the trained model can generalize better and prevent itself from overfitting. The augmentation combinations can have one to twelve augmentation techniques, and each augmentation creates a new unique image.

This statement is shown in the table below which consists of example images augmented using different combinations of traditional augmentation techniques.

Table 7: Sample of images after applying self-written augmentations.

Augmentation techniques applied	Sample image (using one of the image from EDE dataset)
Original image	

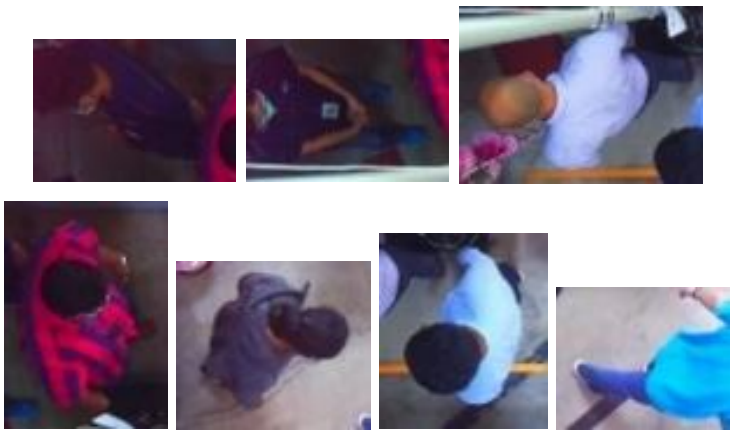
• Applied Gaussian Blur • Added salt and pepper noise • Changed image aspect ratio	
• Flip image horizontally • Shifted image hue • Increase image saturation • Add grid mask • Random erase	
• Flip image vertically • Shifted image hue • Increase image brightness • Applied Gaussian Blur	

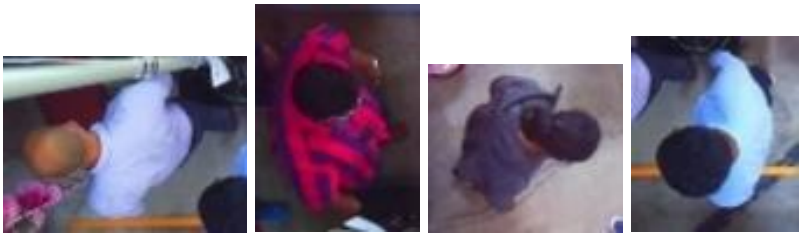
As image that is applied with augmentation combinations consisting of one augmentation techniques does not has huge difference with the original image, the original image will not be included in the training dataset. Hence, the number of augmentation combination applied to each image can be equate to the multiplicator of the size of the dataset. In order words, applying ten augmentation combinations on one image will create ten new unique augmented images which are considered different from each other.

5.3 Augmentation via GAN

As mentioned in the methodology, the images need to be cropped slightly larger than the bounding box, then filtered to be the training dataset. An illustration of the process is tabulated below:

Table 8: Illustration of the flow to prepare the training dataset for GAN

Description	Sample image (using one of the images from the EDOB dataset)
Original image	
Cropped human instances	

After filtering the human instances that are incomplete or occluded.	
After resizing to 256x256	

Both the EDE and EDOB datasets are used to train the GAN models, and the details of the training dataset are summarized as follows:

Table 9: Number of cropped human instances from EDE and EDOB dataset

Dataset	Number of cropped images / human instances
EDE	1724
EDOB	609

The performance of DAGAN and StyleGAN2-ADA in augmenting the images are discussed in the sections below.

5.3.2 DAGAN

The authors included the creation of visual outputs of the DAGAN model in the code shared on GitHub [79]. In the visual output images, the first column is the original image, and the second column onwards are the augmented images that are generated by the DAGAN model that given the original image in the same row is inputted.

A DAGAN model is trained using cropped images from the EDE dataset for continuously two days on the GPU using the same settings described in the paper. The visual output of the DAGAN model trained on cropped images from the EDE dataset is as shown below:

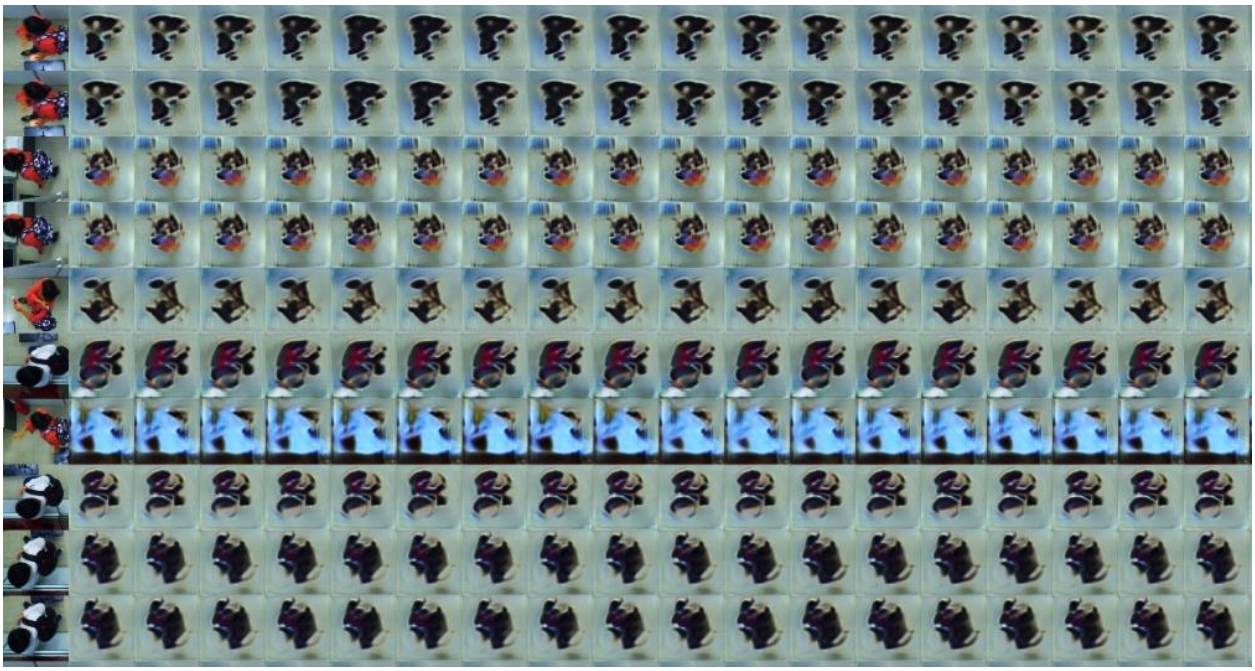


Figure 34: visual output of the DAGAN model trained on cropped images from the EDE dataset

The results are barely acceptable, as the generated images do not resemble overhead humans and do not relate with the original image in any way. Also, it can be observed that mode collapse [55] has occurred as the generated images is the same for each row from the second column onwards, instead of having a slightly different generated image when moving from one column to another. The deduction made for this failure is that overhead person images are much complicated then the datasets used in the research paper, which includes the Omniglot dataset, the EMNIST dataset, and the VGG-Face dataset. For instance, the VGG-Face dataset has the face features at fixed locations, e.g., noses at the middle of the image and mouth below it, as illustrated in Figure 35.

Figure 35: Sample images from VGG-Face dataset. Face features are at fixed locations on the image. Adapted from: [80]

In contrast, the overhead humans from the EDE dataset images have different orientations, e.g., the head is not always at the middle, the human is facing different directions, the hand positions are varied across images, etc. as illustrated below:

Figure 36: Sample overhead human images. Features are varied and at different locations on the image.

Hence, more convolutional layers are added to the DAGAN model, in order to hopefully help the DAGAN model to learn the complicated features and improve its performance. The visual output after retraining of the DAGAN model with additional convolutional layers is shown in Figure 37.

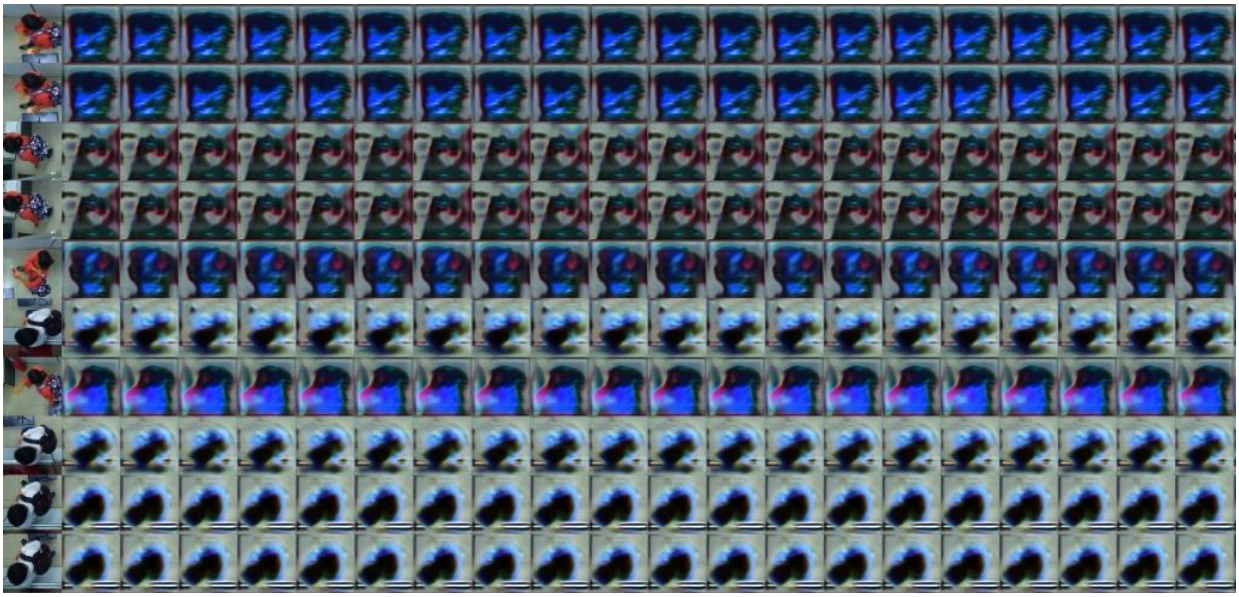


Figure 37: visual output of the DAGAN model after adding extra convolutional layers.

As shown in the figure above, adding convolutional layers did not help the DAGAN model to learn the features better. Hence, another possible reason for the poor performance is that the dataset size is too small for the DAGAN model to be able to learn and interpolate the data distribution. Thus, the performance of DAGAN on the EDOB dataset is not tested since the EDOB dataset has a smaller dataset size, so it can be foreseen that the performance of the DAGAN model on EDOB dataset will be worse.

Due to the underperforming of the DAGAN model in augmenting overhead human instances, the generated human instances will be of low quality which will affect the performance of the object detection model. Thus, it is decided not to use in, it is decided not to use in the later stages i.e., augmenting the human instances and to include them in the training of the object detection model.

5.3.3 StyleGAN2-ADA

The authors of StyleGAN2-ADA also included the creation of visual outputs of the DAGAN model in the code shared on GitHub [73]. Different from DAGAN, the visual output is a photo college of 1024 generated images via interpolation of the learned data distribution. A StyleGAN2-ADA model is trained using cropped images from the EDE dataset for continuously four hours on the GPU using the same settings described in the paper. The visual output of the trained model is as shown below:

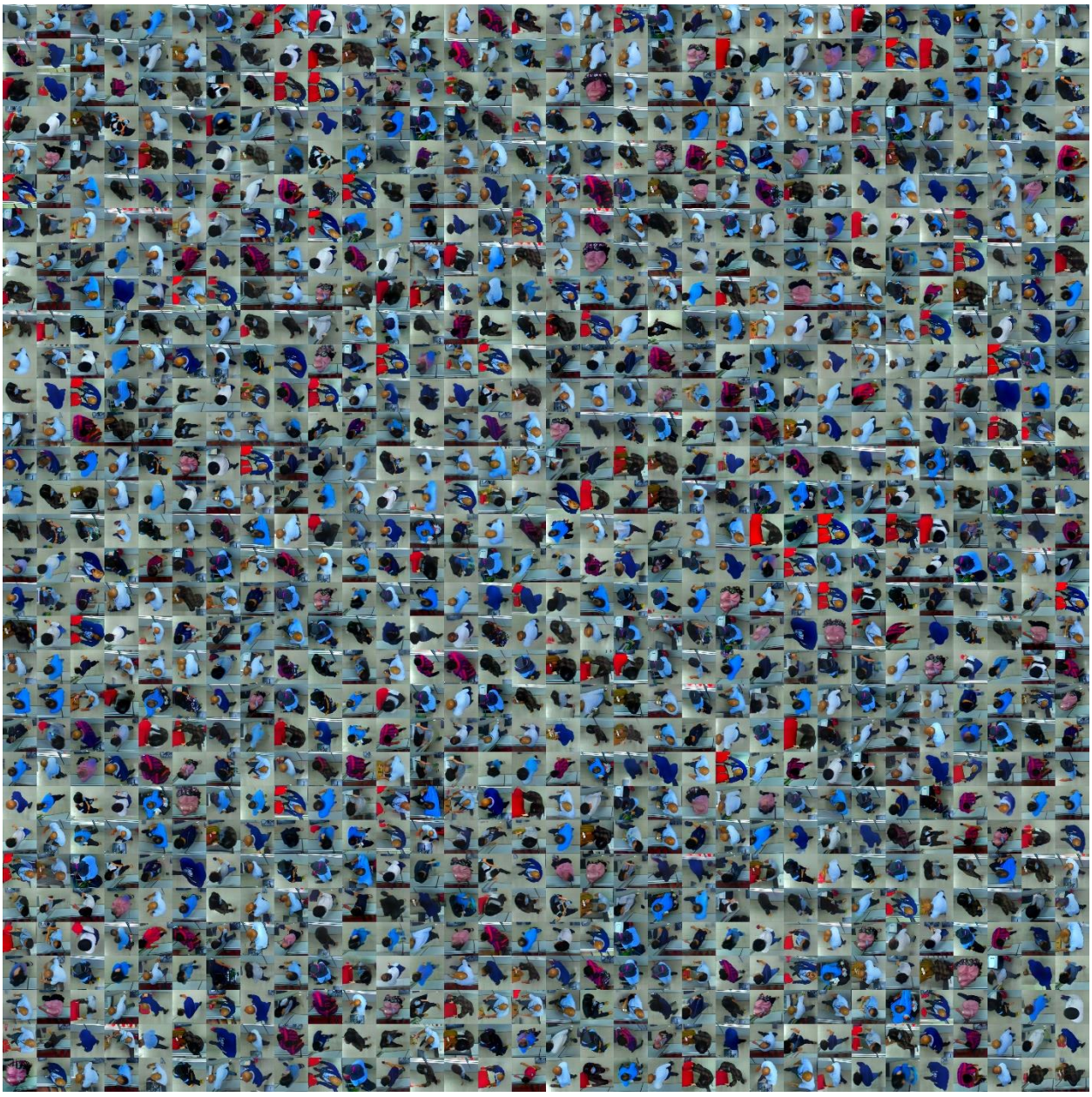


Figure 38: The visual output of the SytleGAN2-ADA model trained on cropped images from the EDE dataset.

Figure 39: Portion of the generated images (zoomed in) by StyleGAN2-ADA on cropped images from EDE dataset.

Although the training time of the StyleGAN2-ADA model is shorter than the training time of the DAGAN model, the performance of the StyleGAN2-ADA model is much better. A great contribution goes to the model initialization using the weights of the pre-trained StyleGAN2-ADA model trained on the FFHQ dataset. If the StyleGAN2-ADA model is trained from scratch with randomly initialized weights, the training duration required to achieve a similar performance will be much longer.

Aside from shorter training time, there is also no sight of mode collapse, as all the generated images are different from each other. The generated human instances also look realistic, and when compared to the original images, it is observed that only small details like the hand or leg postures and hairstyles are changed. This result is exactly what is desired, as it performed augmentations that are not possible via traditional augmentation techniques.

However, there are some generated images that are imperfect, e.g., having no head or two heads, head and body facing opposite directions, disjointed limbs, etc. Some examples are illustrated below:

(a) *(b)* *(c)* *(d)*

Figure 40: (a)(b) Two heads, (c) Missing head, (d) head and body facing different directions

Among all defects, the most common defect is having two heads. The actual reason for this flaw is not known, as all human instances in the training images only have one head. A possible reason that leads to this defect is the varied head positions in the training images which might have caused confusion to the model.

As the performance of the StyleGAN2-ADA model is satisfactory, it will be used to augment human instances in the training images of the object detection model. Further details on how it is performed and the effect of including augmentation via GAN models will be covered in later sections.

5.4 Object Detection

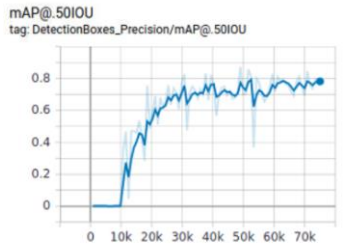
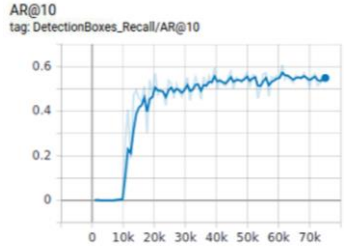
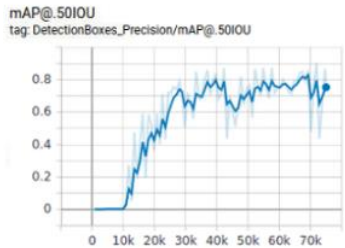
The effect of data distribution, augmentation methods, and transfer learning is investigated in this final year project. The findings are presented in the subsections below.

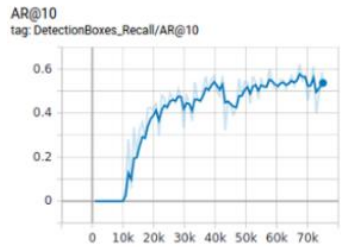
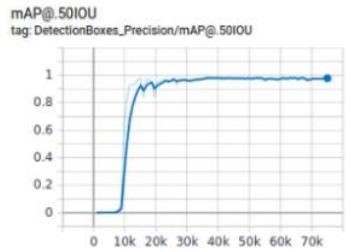
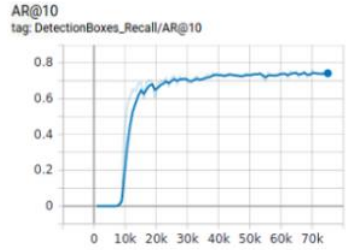
It is also worth noting that the precision is measured using mean Average Precision for IoU greater than 0.5 (mAP@IOU50 because 0.5 IoU is accurate enough to be used for the subsequent stage, i.e., tracking of each detected individual). On the other hand, recall is measured using Average Recall given 10 detections per image (AR10) as from the images collected, the number of human instances is from 1 to 20, hence AR10 is the most suitable choice compared to AR1 and AR100.

5.4.1 Effect of data distribution

Each dataset represents a different data distribution due to different background contexts of the training images. The model is trained using different combinations of datasets, to simulate the effect of training using different data distribution. To ensure fair comparison, the validation set needs to be consistent, and hence all validation set are chosen from the EDOB dataset. The results of the experiments are tabulated in Table 10.

Table 10: Model performance when trained with datasets of different data distributions.

No.	Situation	Experiment settings		Results
		Training set	Validation set	
1	Data distribution of the training set is general and might not overlap with the data distribution of the validation set.	Datasets involved: All datasets except EDOB dataset No. of images: 87311 No. of annotations: 144572 *Each image goes through ten augmentation combinations	Datasets involved: EDOB dataset No. of images: 1049 No. of annotations: 1536	mAP@IOU50  0.7816 AR@10  0.5487
2	Data distribution of the training set is general and has overlap with the data distribution of the validation set.	Datasets involved: All datasets + 500 images from EDOB dataset No. of images: 87361	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1297	mAP@IOU50  0.7533

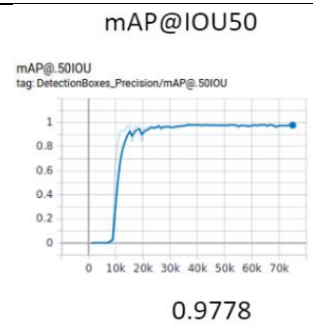
		No. of annotations: 145560 *Each image goes through ten augmentation combinations	*500 training images that are randomly selected from EDOB dataset will not be included in the validation set	AR@10  0.5373
3	Data distribution of the training set consists of only the data distribution of the validation set.	Datasets involved: 500 images from EDOB dataset No. of images: 500 No. of annotations: 738 *original/raw images are used as training images.	Datasets involved: EDOB dataset No. of images: 549 No. of annotations: 797 *500 training images that are randomly selected from EDOB dataset will not be included in the validation set	mAP@IOU50  0.9778 AR@10  0.7429

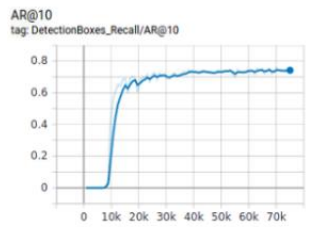
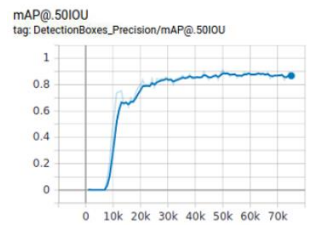
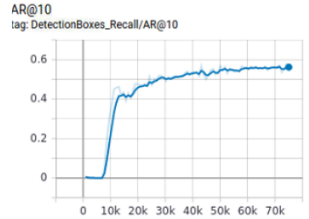
5.4.2 Effect of augmentations

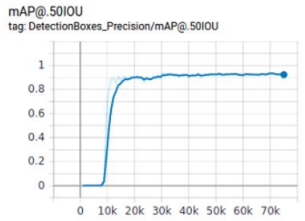
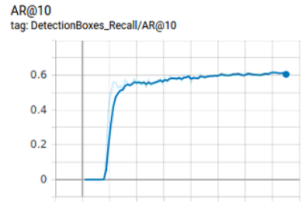
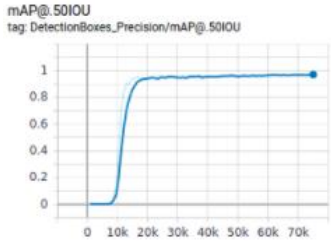
Since collecting and labelling images will require great effort, ELID hopes to have a training mechanism that can minimize the number of images need to be collected without compromising much accuracy. Therefore, the following hypothesis is made: “If the images from the target environment are augmented during preprocessing, lesser images are needed to achieve the same results with the model trained with raw images.”

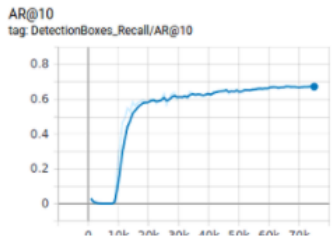
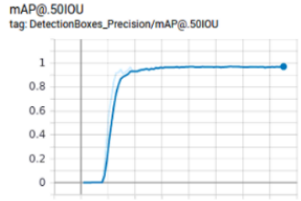
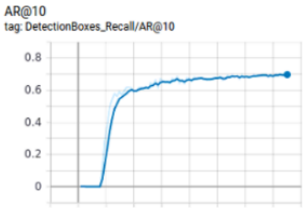
The training dataset is augmented using traditional augmentation techniques as well as using Generative Adversarial Networks in different ways, and the results are tabulated below.

Table 11: Important findings and stepstones on experiments on different augmentation settings on EDOB dataset.

No.	Situation	Experiment settings		Results
		Training set	Validation set	
3	Using a greater number of training images without any augmentations.	Datasets involved: 500 images from EDOB dataset No. of images: 500 No. of annotations: 738 *Original / raw images are used as training images.	Datasets involved: EDOB dataset No. of images: 549 No. of annotations: 797 *500 training images that are randomly selected from EDOB dataset will not be included in the validation set.	

				AR@10  0.7429
4	Reduce the number of raw images but maintained the same number of training images as Experiment 3.	Datasets involved: 100 images from EDOB dataset No. of images: 500 No. of annotations: 675 *Each image goes through five augmentation combinations	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1401 *100 training images that are randomly selected from EDE dataset will not be included in the validation set.	mAP@IOU50  0.868 AR@10  0.5615

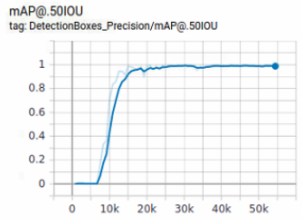
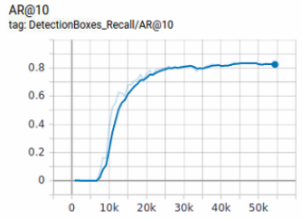
5	Maintain the same number of raw images as Experiment 4 but have a more training images by applying more augmentation combinations on the raw images.	Datasets involved: 100 images from EDOB dataset No. of images: 4000 No. of annotations: 5400 *Each image goes through forty augmentation combinations	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1401 *100 training images that are randomly selected from EDE dataset will not be included in the validation set.	mAP@IOU50  0.9233 AR@10  0.6056
6	Increase the number of raw images used, while maintaining the number of augmentation combinations applied.	Datasets involved: 200 images from EDOB dataset No. of images: 8000 No. of annotations: 11760 *Each image goes through forty augmentation combinations	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1242 *200 training images that are randomly selected from EDOB dataset will not be included in the validation set.	mAP@IOU50  0.9705

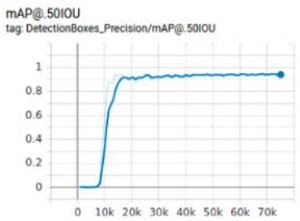
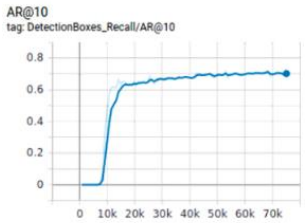
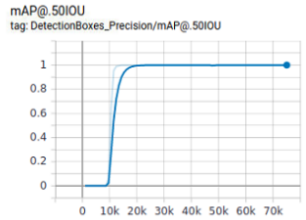
				AR@10  0.6736
7	Maintaining the setting in Experiment 6, but new augmentation techniques are added to increase the number of augmentation techniques.	Datasets involved: 200 images from EDOB dataset No. of images: 8000 No. of annotations: 11760 *Each image goes through forty augmentation combinations ***New augmentation techniques are included.**	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1242 *200 training images that are randomly selected from EDOB dataset will not be included in the validation set.	mAP@IOU50  0.9707 AR@10  0.6975

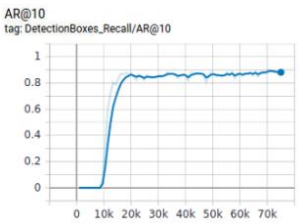
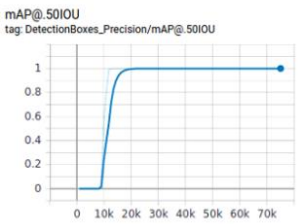
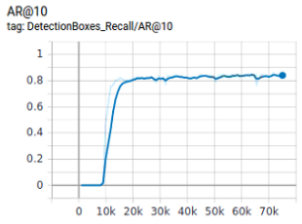
From the table above, it can be observed that randomly selecting 200 images and applying forty augmentation combinations on the training images gives the best performance. To verify and validate this findings, EDE dataset and

TVPR dataset are tested under the same setting. For each dataset, the performance of the model trained using great number of training images no augmentation is compared against the performance of the model trained using lesser number of training images applied with forty augmentation combinations. The results are tabulated below:

Table 12: Validation of findings on other datasets

No.	Situation	Experiment settings		Results
		Training set	Validation set	
8	Using a greater number of training images from the EDE dataset without any augmentations.	Datasets involved: 1780 images from EDE dataset No. of images: 1780 No. of annotations: 11760 *Original / raw images are used as training images.	Datasets involved: EDOB dataset No. of images: 445 No. of annotations: 597 *Remaining images that are not included in the training set is placed in the validation set.	mAP@IOU50  0.9876 AR@10  0.8245

9	Same setting as Experiment 7 to reduce the number of raw images.	Datasets involved: 200 images from EDE dataset No. of images: 8000 No. of annotations: 9520 *Each image goes through forty augmentation combinations	Datasets involved: EDE dataset No. of images: 445 No. of annotations: 597 *200 training images that are randomly selected from EDE dataset will not be included in the validation set	mAP@IOU50  0.9393 AR@10  0.7026
10	Using a greater number of training images from the TVPR dataset without any augmentations.	Datasets involved: 1523 images from EDE dataset No. of images: 1523 No. of annotations: 1523 *Original / raw images are used as training images.	Datasets involved: EDOB dataset No. of images: 370 No. of annotations: 370 *Remaining images that are not included in the training set is placed in the validation set.	mAP@IOU50  1.0

				AR@10  0.8822
11	Same setting as Experiment 7 to reduce the number of raw images.	Datasets involved: 200 images from TVPR dataset No. of images: 4000 No. of annotations: 1523 *Original / raw images are used as training images.	Datasets involved: TVPR dataset No. of images: 1693 No. of annotations: 1693 *Remaining images that are not included in the training set is placed in the validation set.	mAP@IOU50  1.0 AR@10  0.8411

Then, the effect of the augmentation by GAN on the performance of the model is studied. As mentioned in the previous section, only StyleGAN2-ADA will be utilised to perform augmentation on the human instances as DAGAN is not able to create realistic images. As some of the generated images by StyleGAN2-ADA is flawed, manual filtering needs to be

performed. After filtering, these generated images will be pasted on the raw images before the augmentation combinations are applied. A code is written to automate the searching for empty areas on the image, i.e., image areas where there are no human instances, to paste of the generated human instances. A few examples images from the EDE and EDOB dataset after the generated human instances are pasted as illustrated below:

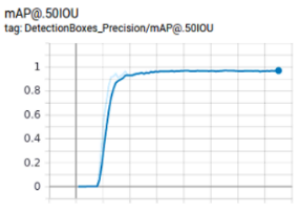
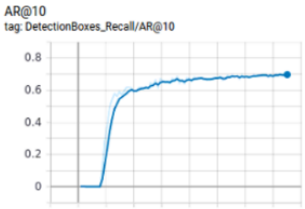
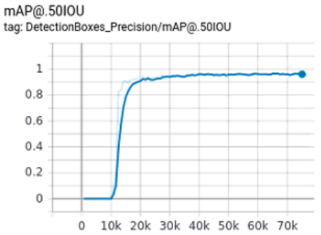
(a) *(b)* *(c)* *(d)*

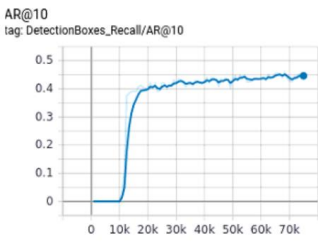
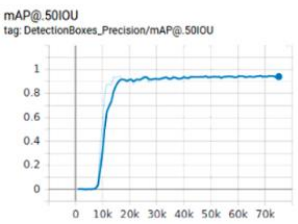
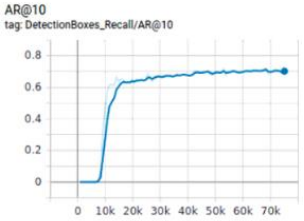
Figure 41: sample images after pasting the generated human instances from EDE dataset [(a),(b)] and EDOB dataset [(c),(d)]

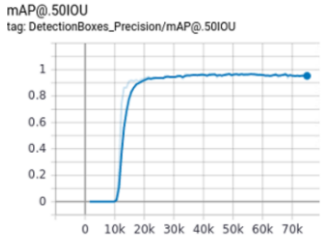
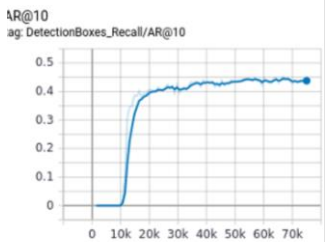
Then, the added human instances need to be labelled using LabelImg, then the training pipeline can proceed as usual, i.e., applying the traditional augmentation combinations and other data pre-processing, then the actual training of the object detection model.

The model performance after training with the training set that consists of generated human instances is tabulated below and is compared with the best performing model which is Experiment 7 for the EDOB dataset and Experiment 9 for the EDE dataset.

Table 13: Validation of findings on other datasets

No.	Situation	Experiment settings		Results
		Training set	Validation set	
7	No augmentations via StyleGAN2-ADA are applied on the training images from the EDOB dataset.	Datasets involved: 200 images from EDOB dataset No. of images: 8000 No. of annotations: 11760 *Each image goes through forty augmentation combinations	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1242 *200 training images that are randomly selected from EDOB dataset will not be included in the validation set.	mAP@IOU50  0.9707 AR@10  0.6975
12	Augmentations via StyleGAN2-ADA are applied on the training images from the EDOB dataset.	Datasets involved: 200 images from EDOB dataset No. of images: 8000 No. of annotations: 11760	Datasets involved: EDOB dataset No. of images: 849 No. of annotations: 1242 *200 training images that are randomly selected from EDOB	mAP@IOU50  0.9607

		*Each image goes through forty augmentation combinations ***Each image is added a generated human instance by StyleGAN2-ADA**	dataset will not be included in the validation set.	AR@10  0.4464
9	No augmentations via StyleGAN2-ADA are applied on the training images from the EDE dataset.	Datasets involved: 200 images from EDE dataset No. of images: 8000 No. of annotations: 9520 *Each image goes through forty augmentation combinations	Datasets involved: EDE dataset No. of images: 445 No. of annotations: 597 *200 training images that are randomly selected from EDE dataset will not be included in the validation set	mAP@IOU50  0.9393 AR@10  0.7026

13	Augmentations via StyleGAN2-ADA are applied on the training images from the EDE dataset.	Datasets involved: 200 images from EDE dataset No. of images: 8000 No. of annotations: 9520 *Each image goes through forty augmentation combinations ***Each image is added a generated human instance by StyleGAN2-ADA**	Datasets involved: EDE dataset No. of images: 445 No. of annotations: 597 *200 training images that are randomly selected from EDE dataset will not be included in the validation set	mAP@IOU50  0.9526 AR@10  0.4377

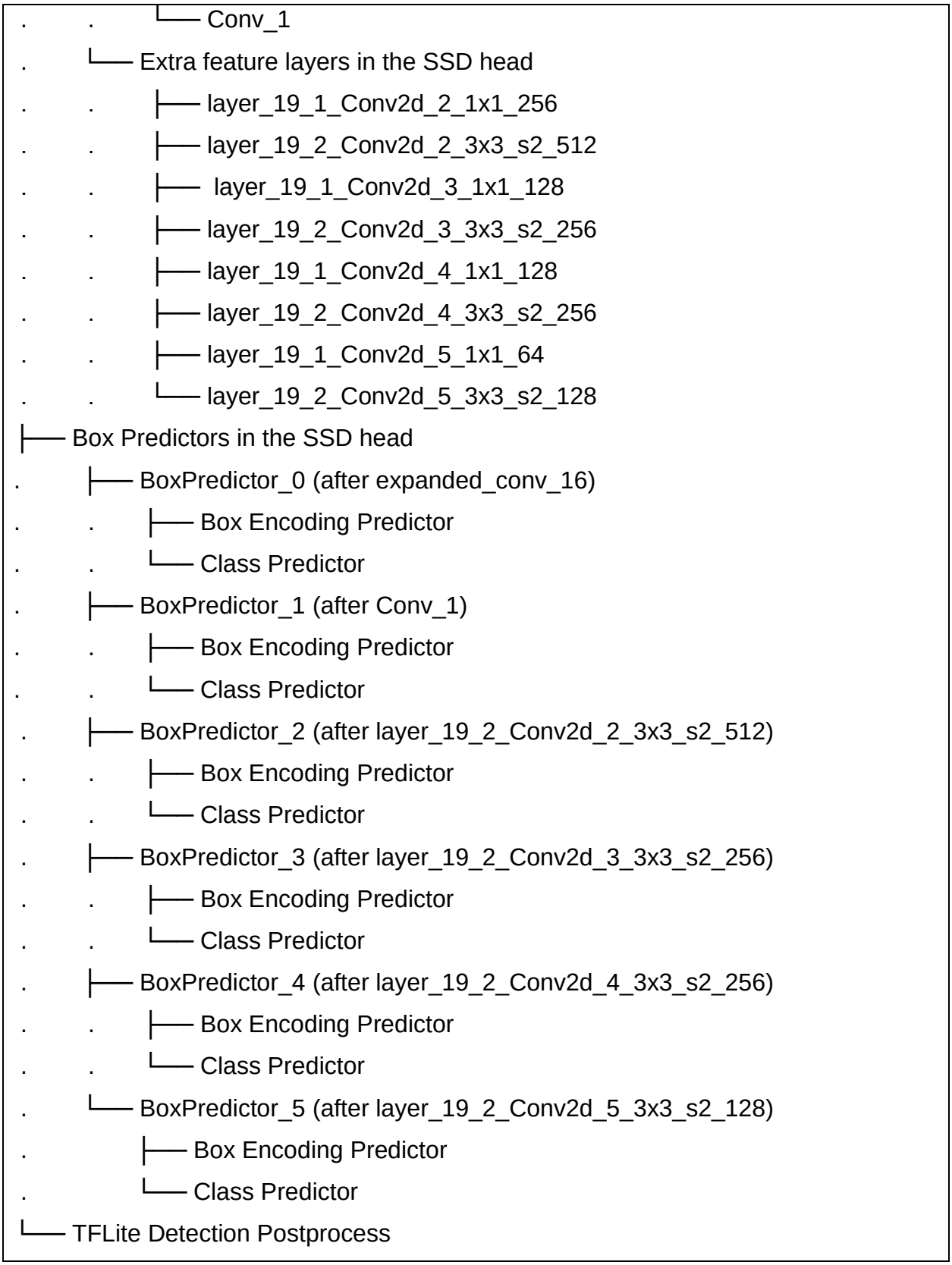
As shown, incorporating augmentation via StyleGAN2-ADA in the training set does not help in improving the model performance, but deteriorated the model performance instead. This is out of expectation, and possible reasons of this finding will be discussed in the later section. Due to the negative impact augmentation via StyleGAN2-ADA have on the model performance, it will be excluded in future experiments.

5.4.3 Effect of transfer learning

As mentioned in the objectives and methodology, it is desired to investigate the effect of transfer learning in improving the model performance, as well as accelerating the training duration of the object detection model.

The layer-by-layer architecture of the MobileNetV2 SSD model is detailed below, where the name of each layer is taken from the file “tflite_graph.pbtxt” in the Tensorflow API [23]. For the feature extractor (MobileNetV2), only convolutional layers are listed out.

```
MobileNetV2 SSD
├── Feature Extractor
.       ├── MobileNetV2 backbone
.       .       ├── Conv
.       .       ├── expanded_conv
.       .       ├── expanded_conv_1
.       .       ├── expanded_conv_2
.       .       ├── expanded_conv_3
.       .       ├── expanded_conv_4
.       .       ├── expanded_conv_5
.       .       ├── expanded_conv_6
.       .       ├── expanded_conv_7
.       .       ├── expanded_conv_8
.       .       ├── expanded_conv_9
.       .       ├──  expanded_conv_10
.       .       ├──  expanded_conv_11
.       .       ├── expanded_conv_12
.       .       ├── expanded_conv_13
.       .       ├── expanded_conv_14
.       .       ├── expanded_conv_15
.       .       ├── expanded_conv_16
```


. . └── Conv_1
. └── Extra feature layers in the SSD head
. . ├── layer_19_1_Conv2d_2_1x1_256
. . ├── layer_19_2_Conv2d_2_3x3_s2_512
. . ├── layer_19_1_Conv2d_3_1x1_128
. . ├── layer_19_2_Conv2d_3_3x3_s2_256
. . ├── layer_19_1_Conv2d_4_1x1_128
. . ├── layer_19_2_Conv2d_4_3x3_s2_256
. . ├── layer_19_1_Conv2d_5_1x1_64
. . └── layer_19_2_Conv2d_5_3x3_s2_128
├── Box Predictors in the SSD head
. ├── BoxPredictor_0 (after expanded_conv_16)
. . ├── Box Encoding Predictor
. . └── Class Predictor
. ├── BoxPredictor_1 (after Conv_1)
. . ├── Box Encoding Predictor
. . └── Class Predictor
. ├── BoxPredictor_2 (after layer_19_2_Conv2d_2_3x3_s2_512)
. . ├── Box Encoding Predictor
. . └── Class Predictor
. ├── BoxPredictor_3 (after layer_19_2_Conv2d_3_3x3_s2_256)
. . ├── Box Encoding Predictor
. . └── Class Predictor
. ├── BoxPredictor_4 (after layer_19_2_Conv2d_4_3x3_s2_256)
. . ├── Box Encoding Predictor
. . └── Class Predictor
. └── BoxPredictor_5 (after layer_19_2_Conv2d_5_3x3_s2_128)
. ├── Box Encoding Predictor
. └── Class Predictor
└── TFLite Detection Postprocess

The architecture of the MobileNetV2 SSD model in expended view is shown below, where the layers are named after the list above.

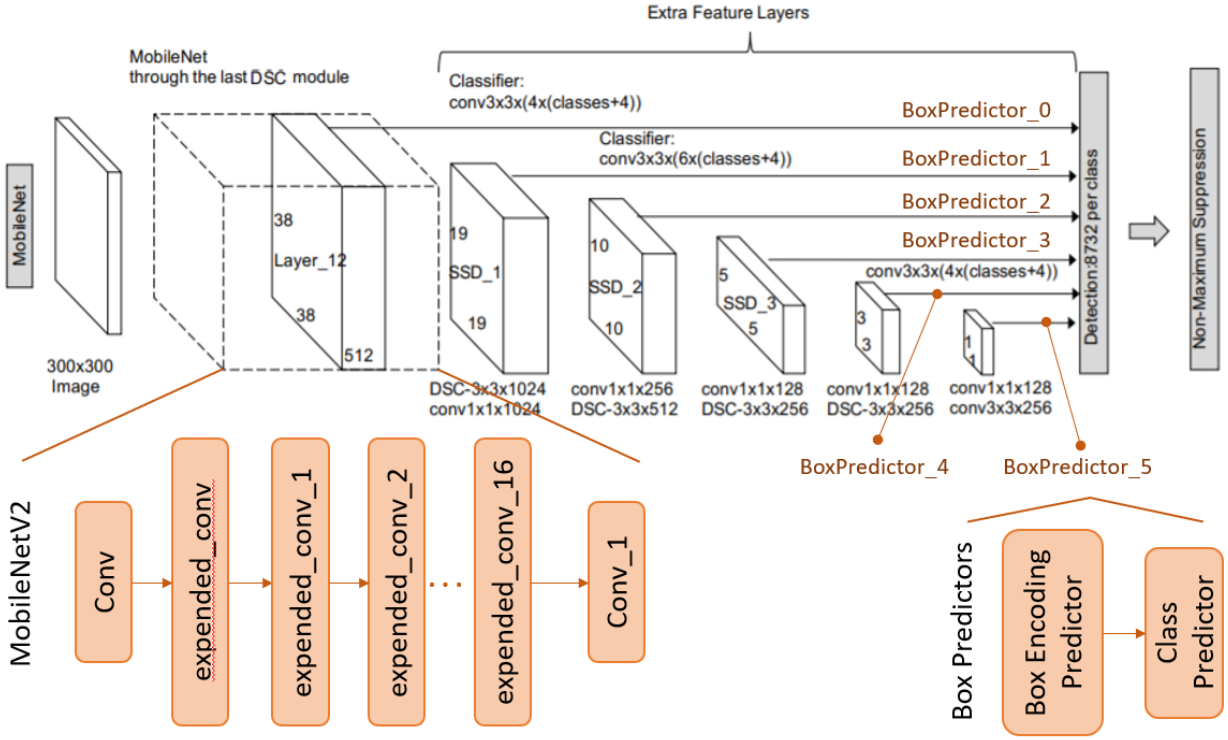
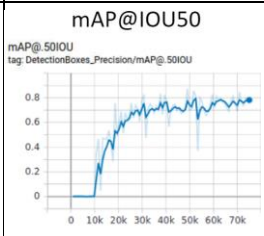
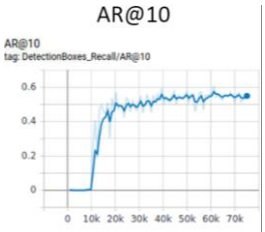
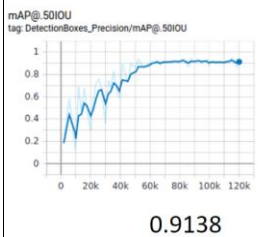
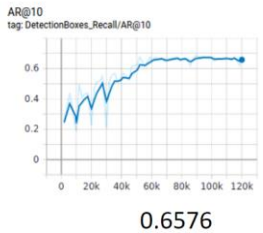


Figure 42: Architecture of MobileNetV2 SSD. Expended_conv layers are the bottleneck layer blocks of MobileNetV2. Modified from [81]

Transfer learning is implemented by freezing the layers of the object detection model during training. Hence, the number of frozen layers of the object detection model during training is varied to get the optimum setting for transfer learning. The precision, recall, and training duration are recorded for each experiment performed.

The most desirable outcome is that by training a general model, the trained model is able to perform well in unseen environments. Hence, the performance of the general model with and without transfer learning is compared, and the results are tabulated below. The training set and the validation set are the same as Experiment 1. The training set consists of 87311 images with 144572 annotations from all datasets except the EDOB dataset, while the validation set consists of 1049 images with 1536 annotations from the EDOB dataset.

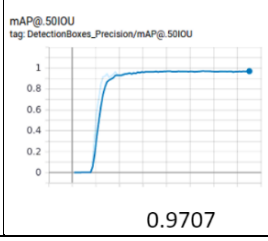
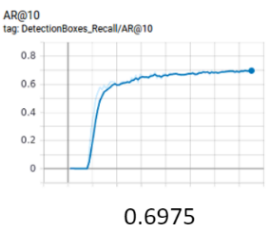
Table 14: Training a general model with versus without transfer learning

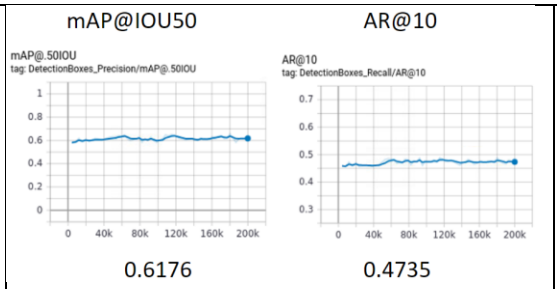
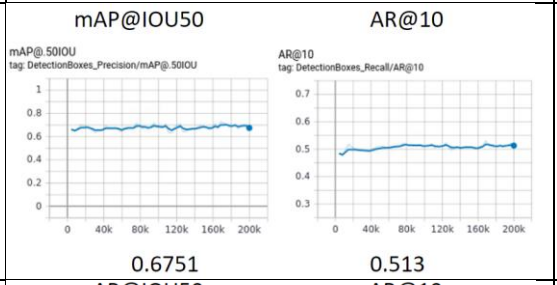
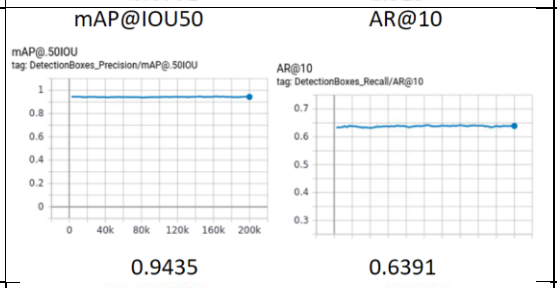
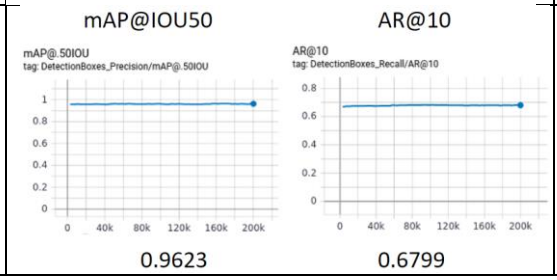
No.	Experiment settings	Model Performance	Training duration per 100 iterations (s)
1	Without transfer learning	mAP@IOU50  0.7816 AR@10  0.5487	70
14	With transfer learning, where layers up to expended_conv_7 are frozen.	mAP@IOU50  0.9138 AR@10  0.6576	30

From the table above, it can be observed that with transfer learning, the general model performs better. However, the performance is still not as good as the model that is trained on the training images with the same environment that it will be deployed in. Thus, the focus shifts to investigating the optimal setting for transfer learning to train this type of model.

The number of layers frozen during training is varied, and the results are tabulated below:

Table 15: Training the model with different number of layers frozen

No.	Experiment settings	Model Performance	Training duration per 100 iterations (s)
7	Without transfer learning	mAP@IOU50  0.9707 AR@10  0.6975	70

15	With transfer learning, where the whole feature extractor and the box encoding predictors of all box predictors are frozen.		15
16	The whole feature extractor is frozen.		18
17	Layers up to expanded_conv_10 are frozen.		20
18	Layers up to expanded_ conv_7 are frozen.		25

From the table above, it is deduced that freezing the layers up to expanded_conv_7 during training gives the closest performance to the model when trained without transfer learning applied, while being able to shorten the training duration by a factor close to three. Hence, it is the optimal setting for transfer learning. Further discussion about the results obtained will be covered in later sections.

5.5 Real time deployment speed when deployed on Google Coral

The model is deployed real time on raspberry pi, and the FPS of the first 500 frames is calculated as 1/(time per frame) and plotted as shown below:

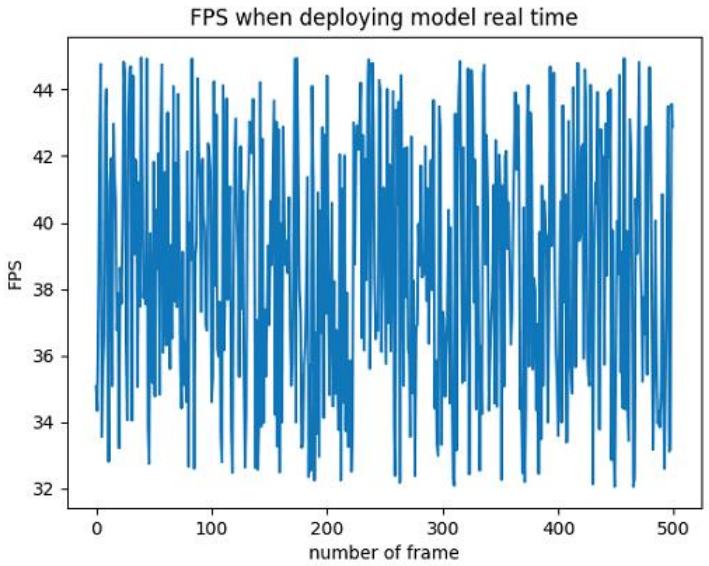


Figure 43: FPS of first 500 frames when deploying model on Raspberry Pi with the USB accelerator (google coral)

As shown in the plot, the FPS is always greater than 30 FPS, hence the first objective achieved.

6.0 Discussion

Initially, one of the datasets (EDOB dataset) is chosen to be the validation images to act as the target environment, and all remaining images collected are used as training images, as shown in Experiment 1. Augmentation combinations are applied ten times on the training set to increase the training set size and variation by a factor of 10. However, the performance is not satisfactory, especially the Average Recall which is only around 55% which indicates only half of the human instances will be detected by the model. This is because the data distribution of the training data is different from the data distribution of the validation data (i.e. the unseen environment), due to not having a large and varied enough training set that can represent a more general data distribution.

Experiment 14 tries to improve the performance of the model by applying transfer learning, where layers up to expended_conv_7 are frozen. The model is trained and evaluated on the same image set, and it can be observed that the performance of the model has improved a lot i.e., an increment of 13% in mAP@0.5IoU and 11% in AR@10. This is because the earlier layers of the pre-trained MobileNetV2 SSD model are trained on a large and varied dataset (COCO dataset), hence the earlier layers are well trained to pick up the low-level features. This gives a good starting point for the training of the model on overhead human detection, as now the model will only need to learn the high-level features. Nevertheless, although the performance of the general model has improved by applying transfer learning, it is still not sufficient as the recall is still not over 80% as stated in the objective.

Since it is not easy to gather a great number of images for overhead images as mentioned previously, it is decided to include some images from the target environment in the training set, so that the training set covers some data distribution of the target environment, which is what is done in Experiment 2. In Experiment 2, the training set is applied with ten augmentation combinations, then 500 images that are randomly selected from the EDOB dataset (representing the target environment) are added to the training set.

However, no significant improvement is observed. This is because 500 is a relatively small number compared to the training set, hence it will not affect the performance much.

In order to prove that the deduction made above is valid, Experiment 3 is conducted, where the training set consists of only the 500 images from the EDOB dataset (representing the target environment), and the model is validated using the remaining images from the EDOB dataset. The performance from this experiment acts as a reference to evaluate and compare the performance of other experiments. As expected, since the training and validation set are of the same environment, the performance is much better as compared to Experiment 1 and Experiment 2, with mAP close to 100% and AR close to 80%. However, it should be kept in mind that this model is not robust to changes in the target environment e.g., changing floor color/pattern, adding new unseen human-like objects, etc., hence this model might not be suitable to be used in practice. Instead, data augmentation should be heavily applied to the training set to prevent the model from overfitting to the background context and hence ensuring the robustness of the model.

After discussing with ELID, as the product will only be deployed at a static site and it can be assumed that the environment wouldn't have great changes being an indoor application, it is decided that both training and validation set will be using the images collected from the target environment. As such, ELID will need to collect images from the targeted site and train the model using those images. Since collecting and labeling images will require great effort, ELID hopes to have a training mechanism that can minimize the number of images that need to be collected without compromising accuracy.

As mentioned in previous sections, it is believed that augmentations are able to generate new samples that are different from their original image. This means that lesser images (which will act as the original images) need to be collected to achieve a similar performance as a model trained with more original images. Thus, later experiments aim to achieve similar performance in Experiment 3 with lesser raw images by applying different augmentations settings.

In Experiment 4, the number of raw images is reduced to 100, where each image is applied five augmentation combinations. This results in the same number of training images with Experiment 3 i.e., 500 training images. The performance drops 11% in mAP@0.5IoU and 18% in AR@10. This shows that the model is underfitting due to insufficient training data. Hence in Experiment 5, the number of augmentation combinations applied is increased from 5 to 40. This helps in increasing the number of training images to 4000. As a result, both the precision and recall improved by 5% However, this improvement is small although the number of augmentation combinations applied is increased by a factor of 8. This is because the changes made by traditional augmentation techniques are limited. In other words, the human instances in the augmented images remain the same as the raw images i.e., the posture, clothing, hairstyle, etc. are unchanged. Hence, the trained model has not learnt with enough human instances, causing it to not be able to perform well in detecting human instances with unseen postures.

From the deduction of Experiment 4, the number of raw images is increased to 200 while maintaining the number of augmentation combinations applied. The performance of the model is improved, where the precision increased by 5% to 97% while the recall increased by 10% to 67%. Then, in experiment 7, new augmentation techniques such as random erase, aspect ratio, grid mask, and background rotation are introduced. The addition of new augmentation techniques has improved the recall by another 2%. Experiment 7 deems to have the optimal settings as its performance is closest to that of in Experiment 3 i.e., requiring 200 images from the target environment that will undergo Transformation40, resulting in a total of 8000 images in the training set.

To prove that this setting is able to give satisfactory performance for different environments, it is also tested with different datasets, which include the EDE dataset (Experiment 8 and 9) and the TVPR dataset (Experiment 10 and 11). The results assert the hypothesis made, and the model is able to give satisfactory results for all the datasets tested. When Experiment 8 is compared against Experiment 9, it can be seen that using

the optimum setting caused a drop in model performance (-5% in mAP@0.5IoU and -10% in AR@10). However, this drop is acceptable considering the training set size of Experiment 9 is approximately one-tenth of the training set size in Experiment 8. For the TVPR dataset, although the training set size is also shrunken by one-tenth, the drop in performance is much smaller, where only the recall decreased by 4%. This might be because the TVPR dataset is simpler for the model since each image will only contain one human instance, and the position of the human instances is similar across the dataset. Hence, the model does not require many training samples for the TVPR dataset to achieve a good performance.

The findings from Experiment 3 to Experiment 11 provide a lot of insights on the effect of traditional augmentation techniques on the model performance, as well as the limitation of traditional augmentation techniques. In short, it is shown that with the application of traditional augmentation combinations, the raw images need to give the desired performance can be reduced. The number of raw images required would be around 200, but ELID should adjust this value based on the scene complexity to achieve the best results while minimizing collection time for the raw images.

Although traditional augmentation techniques are able to improve the model performance and require a smaller number of raw images, augmentation via StyleGAN2-ADA did not achieve similar results. Instead, incorporating generated human instances by StyleGAN2-ADA into the raw images caused a huge drop in model performance. This might be due to the method of these synthetic human instances are added. As shown in the sample images in the previous section, it can be seen that there are obvious boundaries between the synthetic human instances with the background image. This might have caused the model to take the ‘shortcut’ by detecting humans from detecting sudden changes in the images which is the boundary of the pasted human instance. Another possible cause is that the synthetic human instances are actually very distinct from the real human instances in the perspective of the object detection model. It is found in previous researches that CNN models are biased towards texture compared to the shape when learning the features of an object [82]. This is proved as there are adversarial attacks

which alters the image in such a way that it is undisguisable in human's eye, but could alter the output the CNN model in an unexpected way [83]. Thus, these synthetic human instances might be considered as hard examples to the model, causing the model trying hard to learn to detect them. These two possible reasons will cause the divergence of the model and the reduction of the model performance.

Besides, augmentation via StyleGAN2-ADA is also not very practical. This is because the StyleGAN2-ADA model needs to be trained every time the background context of the target environment changes. Two problems will arise such as the increment of training duration and insufficiency of training data. The training of the StyleGAN2-ADA took around 4 to 6 hours to be able to generate images that are realistic and usable. This is opposed to what ELID aims for: to minimize the time required to prepare the object detection model for deployment at the site. Also, in the actual application, only around 200 raw images will be collected, indicating that the number of human instances that can be used to train the StyleGAN2-ADA model will reduce. This would lead to a longer training duration for StyleGAN2-ADA or/and reduced quality of the generated images, as StyleGAN2-ADA only guarantees good results for datasets with at least a few thousand training images [69]. If more raw images need to be collected to ensure the quality of the generated images by StyleGAN2-ADA, the total required time before deployment of the model will be longer. Therefore, taking all these factors into consideration, it is decided not to include augmentation via GAN in the training pipeline.

After that, to accelerate the training, transfer learning is incorporated in Experiment 15 to Experiment 18. The model is initialized with the weights of the pre-trained MobileNet2 SSD model, then some layers of the model are frozen during training. The rationale for freezing the earlier layers of the model is that given the low-level features learned in the previous task are the same as the target task, the earlier layers need not be trained anymore since their weights are already optimized. The more layers are frozen, the lesser the computation is required during training since backpropagation and parameter optimization of the frozen layers are not required, therefore the training will speed up. However, if the assumption that the features learned by the frozen layers are useful to

the target task does not hold, then the model performance will drop. Therefore, Experiment 15 to 18 aims to find the optimal setting for transfer learning, i.e., up to which layer should be frozen during training that can give a satisfactory performance while minimizing the training duration.

In Experiment 15, only the class predictors of all box predictors in SSD are not frozen. This is a common setting that was done in transfer learning for classification problems, where only the classifier layers are changed and trained. However, the results show that this setting is not preferable for this project, as the model performance dropped a lot compared to Experiment 7 (without transfer learning), i.e. mAP@0.5IoU dropped by 35.3% and AR@10 dropped by 22.6%. Therefore, in Experiment 16, only the feature extractor (MobileNetV2) is frozen, or in other words, only the SSD head is trained. The performance improved from Experiment 15, but still not satisfactory. Hence, the number of layers frozen is further reduced in Experiment 17 and 18. It is then concluded that freezing layers up to expended_conv_7 (the ninth convolutional block in the MobileNetV2 feature extractor) is the optimum setting. With this setting, the training duration is shortened by a factor of close to three, while only having a small compromisation in the precision and recall.

7.0 Project Impact

Documentation on the pipeline is prepared for ELID to implement it in their products, from collecting images on site, pre-processing the images, applying augmentations to the training images, training the MobileNetV2 SSD model via transfer learning to compiling and deploying the trained model as an Edge TPU model. ELID is satisfied with the results, hence the findings of this final year project are utilized by ELID in their current version for ELIDEye. Taking the advantage of rising needs of people counting due to the pandemic, ELIDEye is also adapted to not only detect tailgating but also count and restrict the number of people entering a space. ELID targets to install ELIDEye in large factories and manufacturing plants, banks, shopping malls, office buildings and government organizations, since these places would require the features provided by ELIDEye. Up till now, up to 40 units of the current version of ELIDEye will be installed in Renesas Semiconductor KL SDN BHD, and many orders are in discussion with various companies. It is believed that multiple deals can be secured as more and more companies are reopening as the government lifts the restrictions, and ELIDEye is able to aid them in following the standard operating procedure (SOP) to restrict the number of people entering their premises.

8.0 Conclusion

Results show that training and validating the model in the same environment performs much better as compared to training in a few different environments and validating the model in an unseen environment. It is also concluded that replacing raw images with augmented images helps in improving the performance as it adds variance to the training set. However, augmentation via StyleGAN2-ADA does not help in improving the model performance is not practical in actual application. Last but not least, transfer learning is applied by initializing the model weights using pre-trained model weights and freezing the model layers during training. The application of transfer learning is proved to be able to increase the training speed while not affecting the model performance. A precision of over 90% and recall of almost 70% is achieved with a deployment speed greater than 30 FPS when the trained model is tested in the target environment.

9.0 Recommendations for future work

The limitation of StyleGAN2-ADA could possibly be solved if the GAN model is changed to MC-GAN. This is mainly because the generated human instances are merged with the background context inputted to the model. This prevents the model taking ‘shortcut’ to take boundaries in images as a hint to the presence of human. As the generated image by MC-GAN does not contain the background context of the images it was trained on, the MC-GAN model only needs to be trained once. After that, the model can be applied to generate new synthetic samples by using the background image of the site which the model will be deployed on. This solves the problem of long training duration each time the site environment changes. The reason MC-GAN is not attempted in this project is due to the great amount of annotating effort required prior to train the MC-GAN mode. As the input for the MC-GAN model is a text description of the human instance and the background image, each human instances in the training set need to be described manually, i.e., from the hairstyle, type and color of the clothes wore, all the way to the hand and leg posture. Although MC-GAN is seen to be a great alternative to perform augmentation, it might require a large training set and the performance of the model is not guaranteed.

Also, to ensure that the collected images are useful in the learning of the model to detect overhead view humans, the MobileNetV2 SSD model pre-trained on COCO dataset can be used to help filter the collected images. If the model is able to accurately detect all human instances in that image, that images will not be included in the training set. On the other hand, for images where the human instances are missed or detected inaccurately by the model, the bounding boxes will be labelled manually and will be used to train the object detection model.

10.0 References

[1] C. Cheh, U. Thakore, B. Chen, W. G. Temple, and W. H. Sanders, "Leveraging Physical Access Logs to Identify Tailgating: Limitations and Solutions," in *2019 15th European Dependable Computing Conference (EDCC)*, 17-20 Sept. 2019 2019, pp. 127-132, doi: 10.1109/EDCC.2019.00032.

[2] T. W. Chan, V. V. Yap, and C. S. Soh, "Embedded based tailgating/piggybacking detection security system," in *2012 IEEE Colloquium on Humanities, Science and Engineering (CHUSER)*, 3-4 Dec. 2012 2012, pp. 277-282, doi: 10.1109/CHUSER.2012.6504324.

[3] M. Kettle, "Inspectors walk through US airport security," in *The Guardian*, ed, 1999.

[4] "EV100 ELIDEye." Elid Sdn Bhd. https://www.elid.com/index.php/products/video-analytic/9-elid-articles/218-elideye-ev100 (accessed 25 March, 2021).

[5] "SOP MOVEMENT CONTROL ORDER (MCO)." Ministry of Health Malaysia. https://covid-19.moh.gov.my/faqsop/sop-perintah-kawalan-pergerakan-pkp (accessed 6th October, 2021).

[6] P. Sunil. "Sep '21 SOPs for selected sectors in Malaysia: Workplace capacity, re-opening of businesses, and more." Human Resources Online. https://www.humanresourcesonline.net/sep-21-sops-for-selected-sectors-in-malaysia-workplace-capacity-re-opening-of-businesses-and-more (accessed 6th October, 2021).

[7] S.-K. Jarraya, M.-H. Alotibi, and M.-S. Ali, "A Deep-CNN Crowd Counting Model for Enforcing Social Distancing during COVID19 Pandemic: Application to Saudi Arabia's Public Places," *Computers, Materials \& Continua,* vol. 66, no. 2, pp. 1315--1328, 2021. [Online]. Available: http://www.techscience.com/cmc/v66n2/40674.

[8] K. Chong, M. P. Ooi, M. S. Chowdhury, and Y. C. Kuang, "Distributed human detection on mobile platform," in *2013 IEEE International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, 25-27 Nov. 2013 2013, pp. 1-6, doi: 10.1109/ICSIMA.2013.6717932.

[9] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 20-25 June 2005 2005, vol. 1, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[10] D. Fuentes-Jimenez *et al.*, "DPDnet: A robust people detector using deep learning with an overhead depth camera," *Expert systems with applications,* vol. 146, p. 113168, 2020, doi: 10.1016/j.eswa.2019.113168.

[11] P. Maheshwari, D. Alex, S. Banerjee, S. Behera, and S. Panda, "Top View Person Detection and Counting for Low Compute Embedded Platforms," in *ACM International Conference Proceeding Series*, ed: ACM, 2018, pp. 35-43.

[12] X. Cao, Z. Wang, P. Yan, and X. Li, "Transfer learning for pedestrian detection," *Neurocomputing,* vol. 100, pp. 51-57, 2013/01/16/ 2013, doi: https://doi.org/10.1016/j.neucom.2011.12.043.

[13] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," presented at the Proceedings of the 24th international conference on Machine learning, Corvalis, Oregon, USA, 2007. [Online]. Available: https://doi.org/10.1145/1273496.1273521.

[14] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering,* vol. 22, no. 10, pp. 1345-1359, 2010, doi: 10.1109/TKDE.2009.191.

[15] F. H. K. d. S. Tanaka and C. Aranha, "Data Augmentation Using GANs," *ArXiv,* vol. abs/1904.09135, 2019.

[16] A. Antoniou, A. Storkey, and H. Edwards, "Data Augmentation Generative Adversarial Networks," 11/12 2017.

[17] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems,* vol. 27, 2014.
[18] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision,* vol. 88, no. 2, pp. 303-338, 2010.
[19] ultralytics. "Precision-Recall Curve." https://github.com/ultralytics/yolov3/issues/898 (accessed 14th September, 2021).
[20] L. Liu *et al.*, "Deep learning for generic object detection: A survey," *International journal of computer vision,* vol. 128, no. 2, pp. 261-318, 2020.
[21] J. Hosang, R. Benenson, P. Dollár, and B. Schiele, "What makes for effective detection proposals?," *IEEE transactions on pattern analysis and machine intelligence,* vol. 38, no. 4, pp. 814-830, 2015.
[22] "Detection Evaluation." https://cocodataset.org/#detection-eval (accessed 14th September, 2021).
[23] "TensorFlow Object Detection API - GitHub." https://github.com/tensorflow/models/tree/master/research/object_detection (accessed 5th April, 2021).
[24] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*, 2016: Springer, pp. 21-37.
[25] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1-9.
[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 18-23 June 2018 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.
[27] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems,* vol. 30, no. 11, pp. 3212-3232, 2019.
[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580-587.
[29] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision,* vol. 104, no. 2, pp. 154-171, 2013.
[30] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence,* vol. 37, no. 9, pp. 1904-1916, 2015.
[31] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440-1448.
[32] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems,* vol. 28, pp. 91-99, 2015.
[33] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779-788.
[34] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263-7271.
[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, 2009: Ieee, pp. 248-255.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems,* vol. 25, pp. 1097-1105, 2012.
[37] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision,* vol. 115, no. 3, pp. 211-252, 2015.
[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.
[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.
[40] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861,* 2017.
[41] S. Das. "CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more…." https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 (accessed 9th Spetember, 2021).
[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818-2826.
[43] I. Junejo and N. Ahmed, "Depthwise Separable Convolutional Neural Networks for Pedestrian Attribute Recognition," *SN Computer Science,* vol. 2, 04/01 2021, doi: 10.1007/s42979-021-00493-z.
[44] "Edge TPU performance benchmarks." https://coral.ai/docs/edgetpu/benchmarks/ (accessed 29 March, 2021).
[45] A. Ghosh, S. A. A. Mahmud, T. I. R. Uday, and D. M. Farid, "Assistive Technology for Visually Impaired using Tensor Flow Object Detection in Raspberry Pi and Coral USB Accelerator," in *2020 IEEE Region 10 Symposium (TENSYMP)*, 5-7 June 2020 2020, pp. 186-189, doi: 10.1109/TENSYMP50017.2020.9230630.
[46] J. Zhao, G. Zhang, L. Tian, and Y. Q. Chen, "Real-time human detection with depth camera via a physical radius-depth detector and a CNN descriptor," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, 10-14 July 2017 2017, pp. 1536-1541, doi: 10.1109/ICME.2017.8019323.
[47] E. S. Olivas, J. D. M. Guerrero, M. Martinez-Sober, J. R. Magdalena-Benedito, and L. Serrano, *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques*. IGI Global, 2009.
[48] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *arXiv preprint arXiv:1411.1792,* 2014.
[49] H. Shin *et al.*, "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning," *IEEE Transactions on Medical Imaging,* vol. 35, no. 5, pp. 1285-1298, 2016, doi: 10.1109/TMI.2016.2528162.
[50] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, 2014: Springer, pp. 740-755.
[51] S. Naz, A. Ashraf, and A. Zaib, "Transfer learning using freeze features for Alzheimer neurological disorder detection using ADNI dataset," *Multimedia Systems,* 2021/05/04 2021, doi: 10.1007/s00530-021-00797-3.
[52] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research,* vol. 15, no. 1, pp. 1929-1958, 2014.
[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015: PMLR, pp. 448-456.

[54] H. Shi, L. Wang, G. Ding, F. Yang, and X. Li, *Data Augmentation with Improved Generative Adversarial Networks*. 2018, pp. 73-78.
[55] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine,* vol. 35, no. 1, pp. 53-65, 2018.
[56] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434,* 2015.
[57] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784,* 2014.
[58] H. Park, Y. Yoo, and N. Kwak, "Mc-gan: Multi-conditional generative adversarial network for image synthesis," *arXiv preprint arXiv:1805.01123,* 2018.
[59] B. S. Welinder P., Mita T., Wah C., Schroff F., Belongie S., Perona, P., "Caltech-UCSD Birds 200," California Institute of Technology, 2010. [Online]. Available: http://www.vision.caltech.edu/visipedia/CUB-200.html
[60] M. Nilsback and A. Zisserman, "Automated Flower Classification over a Large Number of Classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 16-19 Dec. 2008 2008, pp. 722-729, doi: 10.1109/ICVGIP.2008.47.
[61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[62] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223-2232.
[63] V. Sandfort, K. Yan, P. J. Pickhardt, and R. M. Summers, "Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks," *Scientific reports,* vol. 9, no. 1, pp. 1-9, 2019.
[64] "Computed Tomography (CT or CAT) Scan of the Abdomen." Johns Hopkins Medicine. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/computed-tomography-ct-or-cat-scan-of-the-abdomen (accessed 31 August, 2021).
[65] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401-4410.
[66] K. Su, E. Zhou, X. Sun, C. Wang, D. Yu, and X. Luo, "Pre-trained StyleGAN Based Data Augmentation for Small Sample Brain CT Motion Artifacts Detection," Cham, 2020: Springer International Publishing, in Advanced Data Mining and Applications, pp. 339-346.
[67] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110-8119.
[68] A. Karnewar and O. Wang, "Msg-gan: Multi-scale gradients for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7799-7808.
[69] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, "Training generative adversarial networks with limited data," *arXiv preprint arXiv:2006.06676,* 2020.
[70] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 20-25 June 2009 2009, pp. 304-311, doi: 10.1109/CVPR.2009.5206631.
[71] L. Wang, J. Shi, G. Song, and I.-f. Shen, "Object Detection Combining Recognition and Segmentation," Berlin, Heidelberg, 2007: Springer Berlin Heidelberg, in Computer Vision – ACCV 2007, pp. 189-199.
[72] A. Antoniou. "Data Augmentation Generative Adversarial Networks (DAGAN)." https://github.com/AntreasAntoniou/DAGAN (accessed 1st August, 2021).
[73] NVlabs. "StyleGAN2-ada." https://github.com/NVlabs/stylegan2-ada (accessed 15th August, 2021).

[74] Google. "Set up the Docker container." https://coral.ai/docs/edgetpu/retrain-classification/#set-up-the-docker-container (accessed.
[75] Google. "Object detection." https://coral.ai/models/object-detection/ (accessed 1st April, 2021).
[76] "TensorFlow models on the Edge TPU: Compatibility overview." https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview (accessed 5th April 2021.
[77] D. Liciotti, M. Paolanti, E. Frontoni, A. Mancini, and P. Zingaretti, "Person Re-identification Dataset with RGB-D Camera in a Top-View Configuration," Cham, 2017: Springer International Publishing, in Video Analytics. Face and Facial Expression Recognition and Audience Measurement, pp. 1-11.
[78] I. Prodaiko. "Person re-identification in a top-view multi-camera environment." https://github.com/ucuapps/top-view-multi-person-tracking (accessed 13th May, 2021).
[79] A. Antoniou. "DAGAN." https://github.com/AntreasAntoniou/DAGAN (accessed.
[80] G. Mai, K. Cao, P. C. Yuen, and A. K. Jain, "Face Image Reconstruction from Deep Templates," *ArXiv,* vol. abs/1703.00832, 2017.
[81] Y. Zhang, H. Peng, and P. Hu, "Cs341 final report: Towards real-time detection and camera triggering," *Standford, CA, USA, Tech. Rep. CS,* vol. 341, 2017.
[82] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," *arXiv preprint arXiv:1811.12231,* 2018.
[83] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572,* 2014.