Singapore Institute of Technology - University of Glasgow
Joint Degree in Computing Science Degree Programme

CSC3001 Capstone Project

Please complete the following form and attach it to the Capstone Report submitted.

Capstone Period: 06 SEP 2021 to 22 JUL 2022
Assessment Trimester: Final Trimester
Project Type: Academic

Academic Supervisor Details:

Name: Peter CY Yau
Designation: Associate Faculty
Email Address: PeterCY.Yau@glasgow.ac.uk
Contact Number: N.A

Student Particulars & Declaration:

Name of Student: Wilbur Lua Kai Heng
Student ID: 1901839

I hereby acknowledge that I have engaged and discussed with my **Academic Supervisor** on the contents of this Final Capstone Report and have sought approval to release the report to the Singapore Institute of Technology and the University of Glasgow.

Signature [illegible]

Date: 22/07/2022

END OF FORM

Singapore Institute of Technology - University of Glasgow
Joint Degree in Computing Science Degree Programme

Final Capstone Report
Real-Time COVID-19 Face Mask Detection: Edge Intelligence with CNN and Single-shot detectors

For **Final Trimester** from 09 MAY 2022 to 22 JUL 2022

Wilbur Lua Kai Heng
Student ID: 1901839

Academic Supervisor: Peter CY Yau

Submitted as part of the requirement for CSC3001 Capstone Project

Acknowledgements

This project would not have been possible without the support of many people. I would like to thank my academic supervisor, Professor Peter Yau, for his guidance, patience, encouragement, and committed assistance during my capstone project. I would also like to thank Professor Fauzi and Mr Remy for their help with the loan of the hardware used in this project.

Table of Contents

Table of Figures

1. Introduction

According to World Health Organization (WHO), in 2022, there will be 520 million confirmed cases, causing 6.2 million deaths worldwide [1]. Research studies by the Infectious Diseases Society of America have shown that countries with stringent regulations have seen drastically reduced numbers of local community transmissions [2]. There is no doubt that the novel COVID-19 virus has impacted the way of interaction between humans. Wearing face masks remains heavily mandated in most countries to minimise social interactions. Extensive studies in [1] and [2] have suggested that wearing face masks drastically reduces the community's asymptomatic transmissions and is directly correlated with the mortality rates. Enforcement of Face mask regulations is difficult as constant monitoring is required. Lapses can cause a chain reaction that leads to exponential viral spread within the community. Therefore, an automatic face-mask detection system can assist health authorities in enforcing public health safety by providing constant monitoring and surveillance, providing an unbiased view of the public's behaviours.

This research extensively reviewed current literature on deep learning, convolutional neural networks and COVID-19 Face Mask detection (FMD) system implementations[3]. Deep learning was predominantly the preferred method for real-time detection over traditional machine learning algorithms such as Support Vector Machine (SVM) and Decision Tree (DT). Usage of deep neural networks such as VGG-16 [4], InceptionV3 [5], and ResNet-50 [6]are popular architectures used to apply transfer learning and train image classification models for face masks. The most popular approach is using object detectors, and existing object detector models can be classified into two different types; single-stage detectors and two-stage detectors. Single-stage detectors such as the YOLO object detection family[7] treat object detection as a regression problem and perform classification, localisation, and detection of multiple objects within one forward propagation of the neural network. At the same time, two-stage detectors like R-CNN [8]and Faster R-CNN [9] are Region Proposal Networks (RPN) that first run a region proposal stage before running the classifier on regions of interest (ROI). Benchmarks studies revealed that Single-shot detectors like YOLOv4[10]were able to obtain similar average precision (AP) compared to Faster R-CNN [9] while achieving 2-3 times more performance in terms of FPS [11] A survey done on existing COVID-19 Face mask detection systems showed that Single Stage detectors like YOLOv2 [12], YOLOv3 [13] and SSD[14] architectures were predominantly used as the favoured solution architecture [15] Another motivation for this research is also the real-time deployment of deep learning on the Internet of Things (IoT) devices [16], [17]. Deploying and leveraging edge computing has been shown to alleviate workload through computational offloading and overcome cloud challenges such as inefficient bandwidth utilisation and privacy concerns stemming from systems such as person and face detection. Extensive research on current FMD system implementations has also revealed gaps in recent research, such as the insufficient emphasis on real-time deployment. In this research, we will be focusing research on state-of-the-art ConvNets and Object detectors such as YOLOv5 [10] and EfficientDet [18] to build an Artificial Intelligence (AI) model for the COVID-19 FMD system. This research will also look at state-of-the-art network optimisation techniques to reduce the model's bandwidth, energy requirement and memory footprint to support embedded systems' real-time needs [19]. This research will also deploy the models to benchmark and evaluate their suitability for real-time edge deployment [20]

1.1 Problem Formulation

1.1.1 Lack of research and implementations for Face Mask Detection (FMD) systems despite high forecasted growth.

The outbreak of the SARS-CoV-2 coronavirus (COVID-19) has brought about an urgent increase in public safety standards as the community travels to and fro work destinations to resume their work obligations. Research done by Allied Market Research under the COVID-19 Impact Analysis report has stated that Face Mask detection markets are expected to grow from US$1.8 Billion in 2021 to US$4.2 Billion globally by the Year 2030 at an 8.1% Compound Annual growth rate (CAGR) [21] Government and relevant authorities require massive deployments of FMD systems to ensure a safe environment for employees to return to their workplaces. Deep learning (DL) has been heavily researched and extensively utilised in areas such as Computer vision (CV) for autonomous vehicles and medical imaging. In comparison, in specific sub-domains such as public health and safety systems such as the Face-Mask system, the existing literature on COVID-19 Face Mask detection systems remains nascent.

Table 1 - Comparison of Model Size, Trainable Parameters across ConvNets
Source: https://keras.io/api/applications/

Model	Model Size (MB)	Percentage increase compared to MobileNet	No.of Traininable parameter
VGG-16	528	+3671%	138.35m
VGG-19	549	+3821%	143.7m
ResNet-50	98	+600%	25.6m
InceptionV3	92	+557%	23.9m
MobileNetV2	14	+ 0 %	3.5m
EfficientNet	29	+ 100%	5.3m

1.1.1 Existing solutions have high processing time and model complexity

After extensively reviewing current literature and existing implementations of COVID-19 Face Mask detection systems, there are two different categories of techniques; traditional machine learning (ML) and Deep learning, which utilises multi-layer networks to perform image classification and detection. Machine learning algorithms such as Decision Trees (DT) and Support Vector Machines (SVM) have been shown to accurately and effectively perform non-linear classifications on unstructured data such as images. However, the model parameters will reach upwards of millions when dealing with images with high resolutions and dimensions. For example, a single 1280 by 720 image would have 2,764,800 input parameters and require colossal processing power to perform model training on billions of parameters. Generally, artificial neural networks are preferred over traditional ML approaches as neural networks such as Convolutional Neural Networks (CNN) can reduce the dimensionality and size of the training images by extracting important features from the image, shrinking the input size, which lowers the number of parameters and therefore, lowers the required processing power. Table 1 compares modern deep CNN networks and their model size and parameters. Mainstream CNN architectures like VGG-16 [4] and Residual Nets (ResNet-50) [6]have 134 million and 23.9 million total trainable parameters, respectively. These models are also heavy in terms of their model sizes as compared to state-of-the-art lightweight models such as MobileNetV2 [22] and EfficientNet [23]

Although CNN models such as ResNet, VGG and Inception networks can achieve high precision and accuracy, deploying such models requires a vast amount of graphical processor (GPU) and memory[24]. Typically, to run the model, the hardware has to have enough memory storage to hold the number of parameters, store its activations, and miscellaneous data such as the current batch of data or any interim preprocessing and augmentation of the data.

1.1.3 Cloud Computing challenges for Deep Learning Internet of Things (DIoT)

The proliferation of the Internet of Things (IoT) and Cyber-Physical systems has caused a massive increase in data produced and transmitted. By 2025, it is forecasted that 30.9 billion IoT devices will be connected to the internet. These devices will contribute up to 79.4 Zettabytes (ZB)[25]. Cloud-centric approaches are no longer sufficient as they fail to meet the real-time demands of IoT applications due to issues such as insufficient network bandwidth bottleneck due to the large stream of data sent from the sensors. Furthermore, it is inadvisable to use cloud computing for video analytics use cases such as an FMD system as it deals with private data such as people’s physical features and, to some extent, even geographic location.

Pushing these sensitive data to the cloud exposes it to potential cybersecurity attacks such as identity theft from man-in-the-middle exploits.

Edge computing presents a new computing paradigm; edge nodes are placed near sensor devices to reduce inefficient network utilisation and the overall communication latency whilst fulfilling privacy preservation requirements. This has made the real-time deployment of deep learning on IoT edge devices increasingly popular.

1.1.4 Lack of neural network optimisation for real-time application

Deep learning networks such as CNNs consist of several components such as convolution layers, pooling layers and fully connected layers that perform classification tasks. For a given n feature map, the convolution operation with kernel k on a given image with height w and width w, the total amount of operations can be estimated to be (n * m * h * w * k * k) [26]The convolution process is computationally expensive, and it is difficult for already resource-constrained devices to have the bandwidth to host and compute all parameters within the network. One characteristic of CNNs is that millions of parameters are connected to perform forward propagation and prediction for image classification or object detection problems. However, literature has suggested that in a neural network, there are over-parameterisation properties, meaning that many neurons that are computed and present in the network do not contribute to the overall performance of the network [19] Regarding current COVID-19 implementations, research has shown insufficient depth in real-time deployment results, and network optimisation strategies such as parameterisation reduction through network pruning or quantisation were also not considered.

1.2 Project Objectives

The proposed FMD system will be trained and built using lightweight state-of-the-art object detection models to automatically perform object detection and localization of face masks to determine whether a person has correctly worn their face mask. The state-of-the-art object detection model will be trained using novel data prepared using open-source datasets, which will be annotated and made public on open platforms for other engineers and researchers for future studies on FMD systems. 2 state-of-the-art models, the YOLOv5s [10], [18]variant from the YOLO family of neural network and Google EfficientDet [18] have been selected as the target architectures. The objective is the benchmark two different state-of-the-art lightweight detectors, to determine if the models are suited for real-time edge deployment.

Another objective is to research the effects of network optimisation strategies such as network pruning and quantization on convolutional neural networks. Sparsify transfer learning will be used to train the optimised model for FMD tasks, and the resultant model will be benchmarked to evaluate the impact on model accuracy and performance.

2. Literature Review

This section will review past works, the current state of deep learning ConvNets, and the 2 main object detection architectures; Single-shot detection and Two-stage detection architectures. Followed by a segment that reviews related works and existing implementations of current COVID-19 Face Mask Detection systems.

2.1 Related Works

This segment reviews the current literature on existing COVID-19 Face mask detection systems. The areas of evaluation include the complexity of the proposed architecture, methodology, datasets used, overall performance, and the suitability of real-time deployment. Existing implementations can be segmented into CNN-based solutions and object detection solutions. Object detection solution is also differentiated based on single-stage and two-stage detectors.

2.1.1 Convolutional Neural Network (CNN) based solutions

In [27], Chowdary et al. proposed using the InceptionV3 [5] architecture, which consists of 48 convolutional layers. Instead of choosing the kernel size for the convolution, the Inception architecture uses all 1 x 1, 3 x 3, 5 x 5 and max-pooling layers. The resulting feature maps are then concatenated. The InceptionV3 model reduces the high computational cost of the 5 x 5 convolutional layers by breaking it down using a smaller filter size and asymmetric convolution technique to reduce overall computation complexity. The dataset used was the Simulated Masked Face Dataset (SMFD), balanced between the mask and non-mask classes. The authors used the weights of the InceptionV3 base model pretrained with 14 million images from the ILSRVC dataset. They applied transfer learning by adding five layers, namely, a 5 x 5 average pooling layer, a flattened fully connected layer, a rectified linear unit (ReLU) layer with 128 neurons, a dropout layer and finally connected it to a softmax classifier of 2 neurons, for mask and no mask classification. However, the model achieved a score of 99.9% and 100% accuracy in both training and testing, respectively, which signifies a possible overfitted model. The computation parameter of the InceptionV3 is more than 24 million parameters that require heavy computational resources to be able to deploy and run classification in real-time applications.

In [28], Mandal et al. proposed using the ResNet-50 architecture, comprising 48 convolutional layers, 1 Max pooling and 1 Average pool layer. The methodology is also utilizing transfer learning on the ResNet-50 trained with the Microsoft Celebrity (MS1M) dataset and VGGFace2. The weights of the first ten layers were frozen, and the rest of the networks were trained with the Real-world Face recognition dataset (RMFRD). However, their approach differed from [27], [28], as they trained the masked and unmasked dataset separately with different hyperparameters, utilising different gradient loss optimisers. There was also a massive disparity in the data distribution, with only 525 masked images, whereas 90,000 unmasked images were used. The unmasked dataset achieved an F1-Score of 89.7%. In contrast, the masked dataset only achieved an F1-Score of 44%, which was highly attributed to the low availability of training and test data. A majority of the network weights were retrained, and the number of images supplied to train the network was insufficient. The ResNet-50[6], similar to InceptionV3 [5], [29]has 23 million parameters, and these deep networks require significant compute resources and have high memory and power consumption.

2.1.2 Object Detection solutions

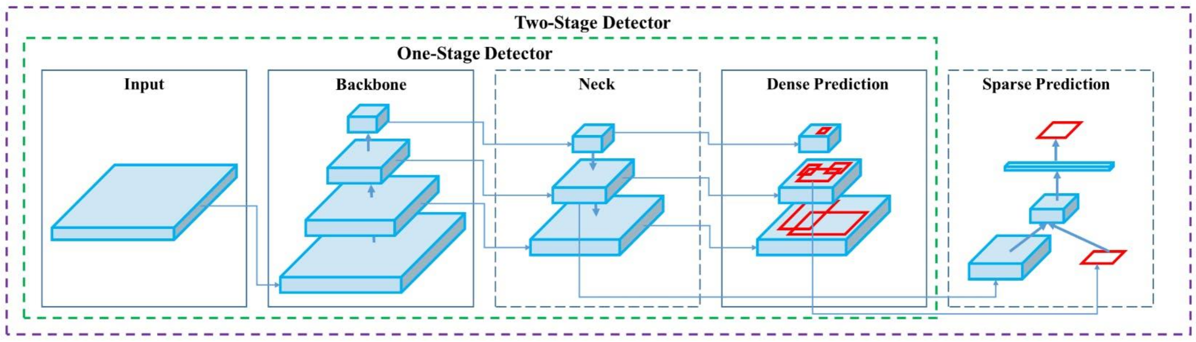

Figure 1- Architecture of Object Detectors

Object detection allows for localisation, detecting multiple objects, and drawing anchor boxes surrounding the target object. Object detection can be categorically divided into 2 categories, One-stage Detectors such as YOLO and SSD and Two-stage Detectors such as R-CNN. Figure 1 highlights the architectural differences.Two-stage detectors use Region Proposal Networks (RPN) to generate candidate anchor boxes called regions of interest (ROI) and run the classification model on these ROIs. [29], Gathani et al. proposed a Region-based network with ResNet-50 backbone architecture to perform feature extraction. The output features were subsequently passed through an RPN to propose ROIs. The ROIs go through a max-pooling layer and are then flattened and passed through densely connected layers and a ReLU activation function. The resultant output is sent to both a classifier to detect if a “mask” or “non-mask” object is present and another regressor to output the bounding box surrounding the object. The network was trained using the 1047 images of masked and non-masked data. The proposed achieved a mAP@ IoU (mean average precision) of 85%. However, similar to [27], [28]the model was not trained with sufficient data, and augmentation techniques were not fully utilized to enrich the dataset. The model also performed significantly poorer on Masked faces, with a mAP@ 0.5 IoU of 64%.

Likewise, in [30], Zhang et al. proposed a Two-stage region proposal network called the Context-Aware R-CNN. For the backbone architecture, the authors used VGG-16 to extract feature maps. They also introduced a Multiple context feature extractor (MCFE) that takes the output feature map and runs 4, 3 x 3 filters with different dilution rates, performs atrous convolution, and concatenates the results into a resultant feature map before running the RPN, followed by the localization and classification branches. The network was trained 4672 was split between 3 different classes, mask, non-mask and incorrect mask. The model achieved an mAP of 84%. However, the accuracy is at the expense of running through 2 different CNNs, which adds to extremely high training parameters. The total configurable parameters would also result in poor FPS performance.

On the other hand, Single-shot detectors do not use RPNs. Instead, the image goes through the backbone CNN and output feature maps are used to directly perform object detection and anchor box prediction in a single forward propagation. In [31], Nagrath et al. developed their face mask detector using SSD architecture to perform face detection; the output is then fed as input to a MobileNetV2 classifier to detect classes such as Mask or Non-Mask. A balanced dataset of 11,042 images was compiled from PyImageSearch and Kaggle. It was processed, augmented and split evenly between the two classes. The object detection model used was the SSD architecture with the ResNet-10 as the backbone. The SSD outputs the detected faces, confidence scores as well as predicted anchor boxes and feeds this data to the MobileNetV2 for classification. The MobileNetV2 used the pre-trained weights obtained from training 14 million images from the ILSRVC dataset, transfer learning was then applied using the custom face mask dataset. The SSDMNV2 model managed to achieve an F1 score of 93%. The authors also considered the real-time requirements of the SSDMNV2 and performed a performance analysis with popular models like VGG-16 and ResNet-50. Their proposed model was able to achieve four times increased Average performance in Frame per Second (FPS).

In [32], Loey et al. built a similar system, instead of using SSD, the YOLO architecture was used. Their system adopted ResNet-50 as the backbone to extract features maps, and the features were used as input to train their YOLOv2 detector model. The proposed system utilized 1415 images from Kagle to train their YOLOv2 model. The model achieved an mAP of 81%. In [33], Vyas et al. used the improved YOLOv4 model to create a Face Mask detector. The YOLOv4 model replaces DarkNet-19 to use CSPDarkNet53, a Cross Stage Partial Network (CSP) that alleviates issues of vanishing gradient and improves feature reusability. The YOLOv4 model also introduced the use of the Spatial Pyramid Pooling (SPP) and Path Aggregation (PANet) which helps the network increase the receptive field and improve the overall performance. To show the effectiveness of the YOLOv4 model, the author trained previous variances such as YOLOv2 and YOLOv3 and evaluated their performance. The dataset consisted of 7959 images collected from WIDER-FACE and MAFA datasets. Given the consistent dataset used, the YOLOv4 had the highest mAP of 84.3% and 63% FPS performance as compared to the YOLOv1 model.

To summarise, there were several different methods of implementation for FMD systems. Most of the implementations prioritized achieving high accuracy of Face Mask Detection rather than a balance of performance and memory consumption. For example, for an input size of 448 by 448, it would require 2 Tera Floating Point Operations per Second (TFLOPs) for a single forward pass for the ResNet-50 and InceptionV3 model, compared to architectures such as MobileNet, which only takes 296 Giga Floating Point Operations per Second (GFLOPs). From the perspective of real-time deployment, it would require a lot of computational resources to deploy models with high model parameters and computation, which will ultimately drive up the cost of deployment. Future research is placing a stronger emphasis on lightweight networks and creative solutions to scale ConvNets to continue to preserve the accuracy while shrinking the model size, such that these detectors can be deployed on the edge for real-time applications.

3. Methodology / Proposed Design

3.1 Proposed Machine Learning (ML) workflow stages

This segment reviews the proposed End-to-End (E2E) machine learning process. Within each workflow stage, it drills down on the engineering aspects, such as the processes and algorithms used and introduces the tools and library utilized.

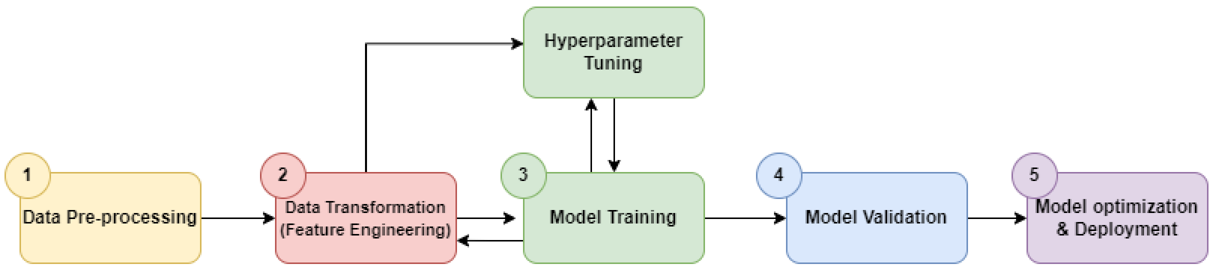

Figure 2 - End to End (E2E) Machine Learning Workflow

As seen in Figure 2, the E2E Machine learning process comprises data engineering, data model engineering and code engineering processes. Firstly, data pre-processing deals with the acquisition and processing of data. The data must be collected, sanitised, and annotated with their respective object classes. As image classification and object detection are supervised learning algorithms, each data has to be labelled and given a bounding box that captures the object's location in the image.

The data transformation stage, also known as feature engineering, will perform data engineering and transformation to the dataset to enrich the data for better model accuracy. Data augmentation techniques will be utilized to enhance overall dataset quality and to improve the model robustness. Dataset will then be split into training, validation, and testing datasets with ration according to industry standards. Once the dataset is prepared, it is exported to its appropriate annotation format, such as COCO JSON so that it can be ingested into the neural network training script.

Training a deep neural network is a challenging task which involves heavy computations such as loss computation of the stochastic gradient descent. It requires a lot of memory to contain the network layers and the training dataset, which will be constantly ingested as input to the neural network in batches. Therefore, the optimal way is to utilise cloud GPUs and perform cross-compilation and training. The output model will only be deployed on edge devices for inferences only. Training a neural network from scratch is undesirable as it requires a lot of computational resources as the network contains a lot of parameters. Backpropagation will have to be computed for every layer, which is undesirable and leads to exceptionally long training time even for lightweight detectors such as YOLOv5 and EfficientDet. Hence, this project will utilize the transfer learning method and use a pre-trained weight that has been trained on the COCO dataset and retrain and update the weights associated with the classification and bounding box regressor. This allows more time to tune the hyperparameters which is more important to ensure high model accuracy. Another critical process of training is evaluation. The specifics of the metrics used for model evaluation methodology are detailed in the model training subsection. From the metrics, the model performance can be interpreted and evaluated.

Next, the model with the best performance will be selected to undergo the network optimisation technique. The techniques chosen are network pruning, which will remove redundant connections within the neural network and thereby shrink the mode, reducing the bandwidth for embedded system deployment. Another compression technique explored is network quantisation, which aims to lower the bit-representation from FP32 to a lower-point representation like INT 8-bit. Sparse transfer learning will also be conducted to retrain the model to prevent accuracy from dropping in the event essential weights in the network are pruned and removed from the network. In the last stage, the model will then be deployed onto the edge device, the NVIDIA Jetson Nano. Once the model has been deployed on the edge device, the relevant real-time application benchmarks will be performed.

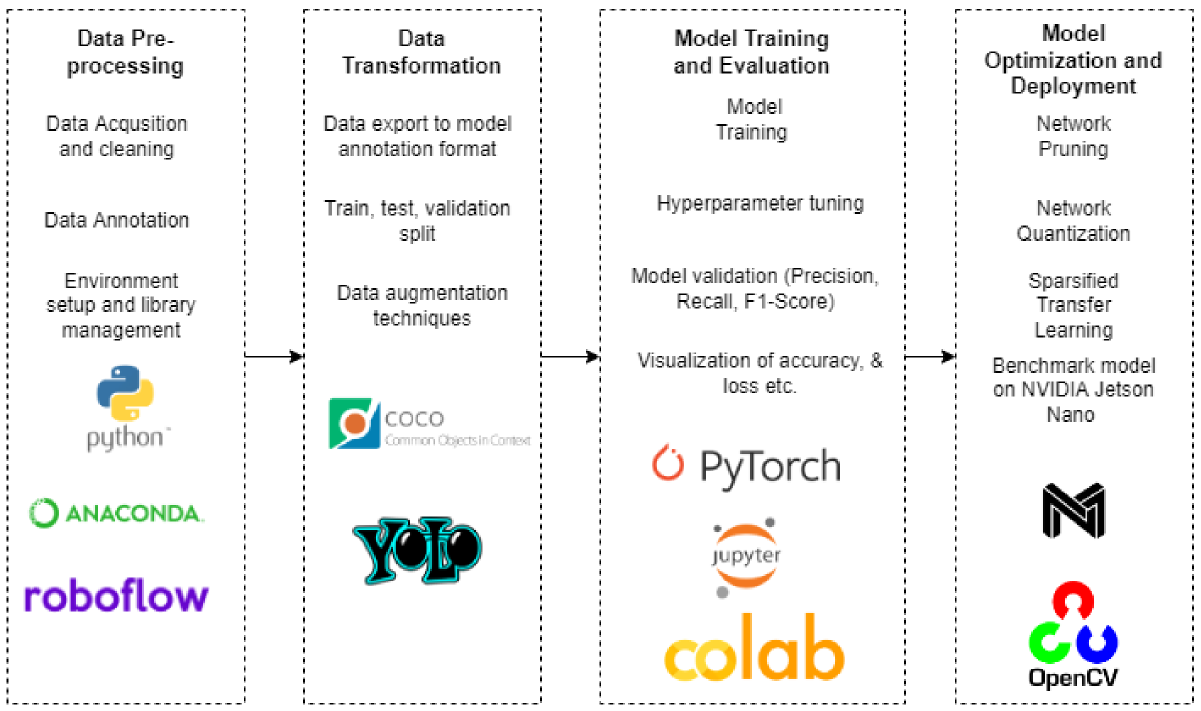
Data Pre-processing
Data Acqusition and cleaning
Data Annotation
Environment setup and library management
python
ANACONDA
roboflow
Data Transformation
Data export to model annotation format
Train, test, validation split
Data augmentation techniques
COCO Common Objects in Context
YOLO
Model Training and Evaluation
Model Training
Hyperparameter tuning
Model validation (Precision, Recall, F1-Score)
Visualization of accuracy, & loss etc.
PyTorch
jupyter
colab
Model Optimization and Deployment
Network Pruning
Network Quantization
Sparsified Transfer Learning
Benchmark model on NVIDIA Jetson Nano
OpenCV

Figure 3 - Processes Within the E2E workflow and the chosen tools

The following section will describe each of the steps in the E2E ML workflow stages. It also goes into the specific configuration and tools used for the different tasks.

3.2 Data Pre-processing

Table 2- Summary of Pros and Cons of Open-source COVID-19 Mask Datasets

Dataset	**Advantages of dataset/repository**	**Disadvantages of dataset/repository**
Kaggle Face Mask Dataset	Dataset has been annotated with bounding boxes of 3 classes **(Correctly, Incorrect and No Mask)**	Huge imbalance between classes (For example, 70% of training samples are for correctly masked and only 30% shared between the other 2 classes)
Real-World Mask Face Dataset (RMFD)	A large volume of data	Dataset is not annotated, and each image contains a lot of background noises.
FaceMask Dataset	A large volume of data that is scraped from the Internet	Dataset masked important features like the eyes which affects model performances.
MaskedFace-Net	A large volume of data, equally balanced between classes. Data is focused on human faces with minimal background noises.	Dataset is not annotated.

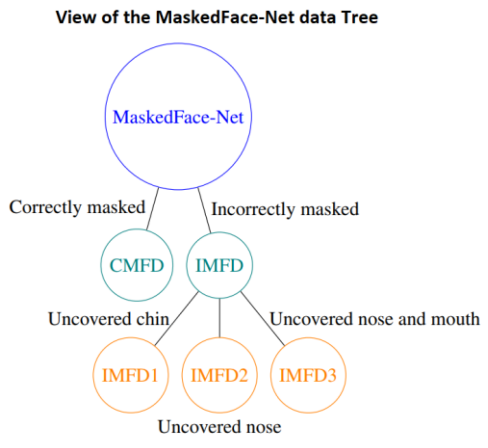

Figure 4- Breakdown of classes within MaskedFace-Net (MFN)

Data pre-processing is one of the most crucial steps in the machine learning workflow as data determines how well the machine learning algorithms learn. Utilizing poorly annotated or imbalanced datasets leads to unwanted behaviours and in the worst case, poor performance. Hence, it is vital to prepare a good quality dataset. Table 2 summarizes the pros and cons of each of the datasets that were considered for the project. Several different datasets have been used to build COVID-19 FMD systems, such as the Real-word Masked Face Dataset (RMFD), Kaggle Face Mask Dataset, and FaceMask Dataset as well as the MaskedFace-Net (MFN).

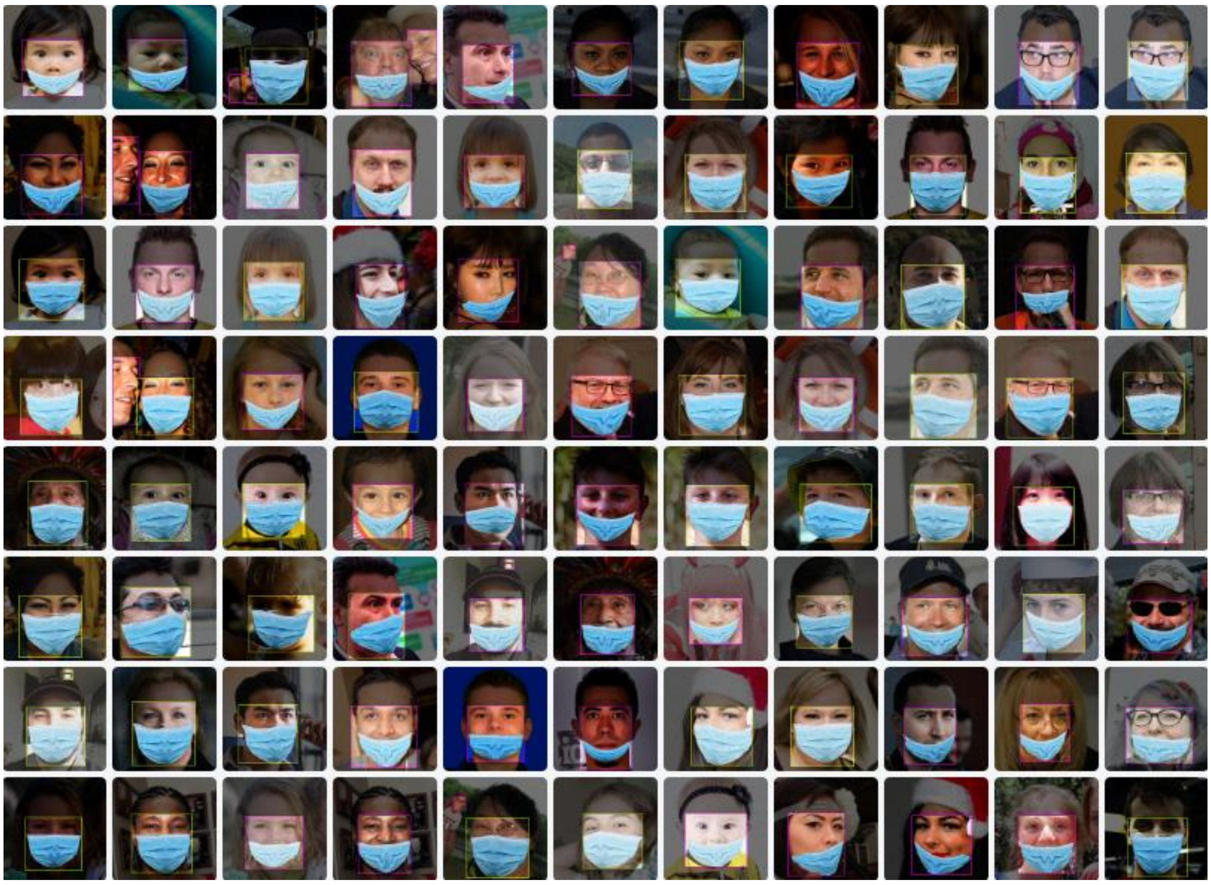

Figure 5- Annotated Data using MaskedFace-Net (MFN)

After evaluating criteria such as class imbalances, ease of usage and integration as well as data quality, the MaskedFace-Net (MFN) was chosen as the dataset as it contains high-quality images with the least amount of background image. Figure 4 shows the breakdown of the MaskedFace-Net dataset. Using this data repository, the project will acquire data for “Correct Mask” from the Correctly masked repository (CMFD) and “Incorrect Mask” from the Incorrectly Masked (IMFD) dataset. Within the “Incorrect Mask” dataset it will contain data which includes 3 different types of incorrectly mask types such as uncovered chin, nose, and mouth. Figure 5 shows the annotated dataset uploaded onto the Roboflow repository; each object within the image has been labelled as ‘correctly_masked’ or ‘incorrectly_masked’. The project also follows the industry standards of the 80-20-10 split into train, validation and test sets ,respectively.

3.3 Data Transformation (Feature engineering)

3.3.1 Data Augmentation

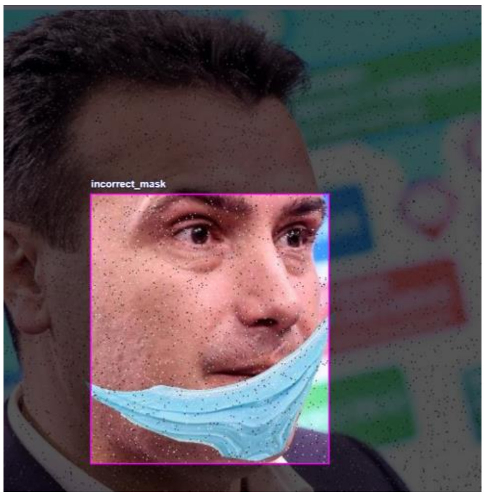

Figure 6- Training dataset example with augmentation (Noise)

Data augmentation was applied to the dataset to increase the amount of data. Augmentation also acts as a regularisation technique and prevents the model from overfitting by exposing more variety of data to the model. As this project deals with image data, visual transformation techniques can be applied to enrich the data. The augmentation technique applied to the dataset includes saturation, exposure and adding noise data onto the image. The following details the rationale behind the augmentation.

- Saturation and Exposure: The vibrancy of the data is adjusted to assist the model in dealing with wild colours affected by things such as lighting.
- Noise: Gaussian random noise is added to the image to prevent over-generalization of the training set and reduce the overfitting of the model.

3.4 Model training and Evaluation

For this project, 2 object detection architecture; the EfficientDet and the YOLOv5 model was chosen as both architectures have shown to be lightweight whilst being able to achieve high accuracy. EfficientDet, specifically the lightest variant, EfficientDet-B0, has shown to be able to outperform the YOLOv3 models with an mAP of 35 compared to 33, with much lesser model parameters and FLOPs. The YOLOv5 model was selected as it has shown promising results, its benchmarked results on the COCO dataset achieved an mAP of 56.8 for the YOLOv5s model outperforming other well-known detectors such as RetinaNet and even other state-of-the-art variants like PP-YOLO.

3.4.1 EfficientDet Architecture

The EfficientDet is a neural network architecture created by the Google AI brains team back in 2020. ConvNets such as Inception, VGG-16 and ResNet have different ways of scaling. For example, ResNet architecture scales using depth, allowing neural networks to traverse deeper into the network without issues such as vanishing and exploding gradients. Other ways of scaling include depth and resolution scaling. Research has shown that the accuracy gain saturates when scaling only by either depth, width, or resolution. It is proven that a neural network's depth, width, and resolution have a linear relationship and that having a compound scaling factor to scale all 3 dimensions of the network will result in better model performance.

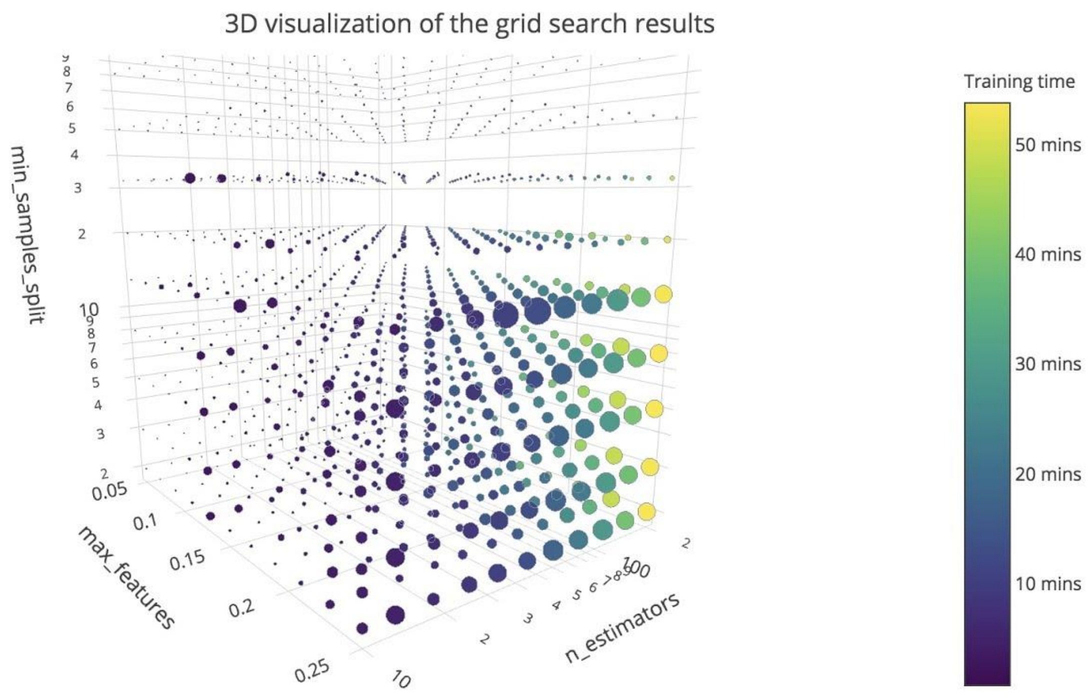

Figure 7 - Gridsearch visualization

3.4.1.1 Backbone ConvNet – EfficientNet

Before moving onto the detector, it is essential first to understand the backbone ConvNet used in the neural network. The EfficientNet architecture introduces the idea of compound scaling. 4 hyperparameters, Φ, α, β, and γ are introduced into the model. Φ represents the constant compound scaling factor and α, β, and γ represent the relative ratio for depth, width and resolution respectively. As shown in Figure 7, the 4 hyperparameters are then selected using the grid search algorithm and after performing cross-validation, the model will select the 4 best parameters that will get the most training accuracy.

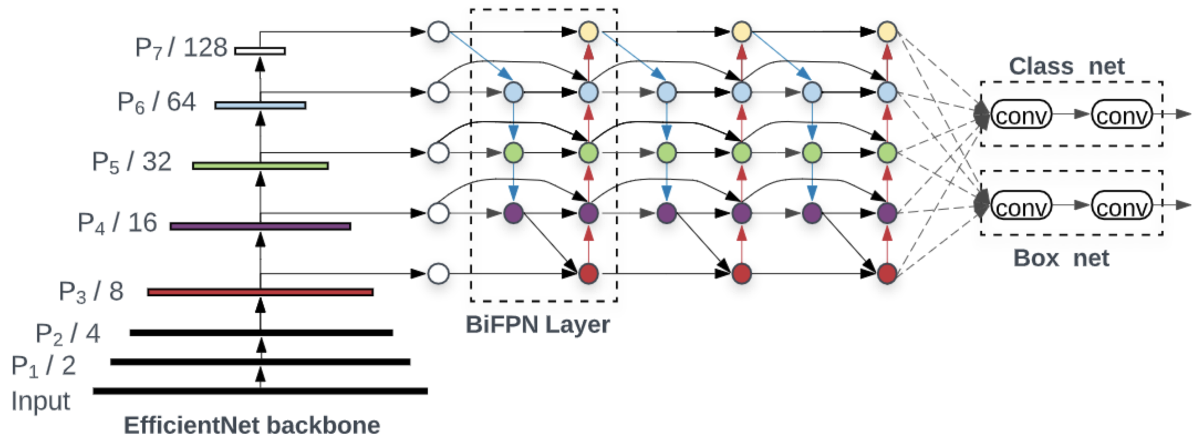

Figure 8 - EfficientDet Architecture

As shown in Figure 8, the EfficientDet detector utilizes EfficientNet as its backbone feature extractor. For the model neck, as opposed to a regular feature pyramid network (FPN), the Bi-directional FPN from EfficientDet introduce novel ways to extract features from the Convnet. Firstly, their research concluded that nodes with just a single edge with no feature fusion will have a lesser impact on the overall feature network. Secondly, an extra connection is made between the original input node and the output feature node, allowing feature fusion without sacrificing much on performance. Lastly, as seen from the figure, this BiFPN layer is then repeated multiple times before feeding the feature maps to the classification and bounding box neural network.

3.4.2 YOLOv5 Architecture

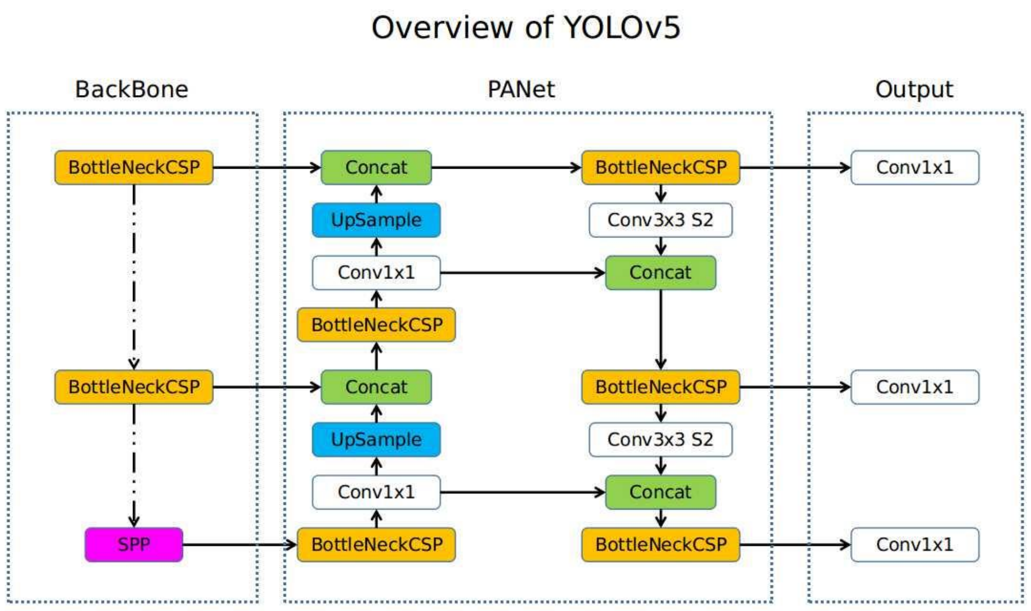

Figure 9 - YOLOv5 architecture

3.4.2.1 Backbone Convnet – Cross spatial partition (CSP) network

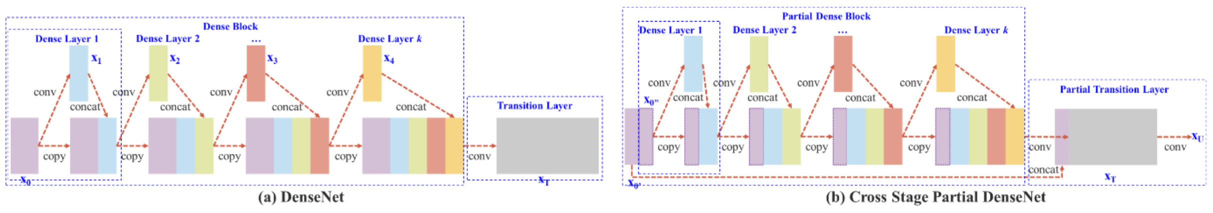

Figure 10- CSPNet - Backbone ConvNet of YOLOv5

Figure 9 shows the YOLOv5 architecture in detail, it first uses the Cross Spatial Partition (CSP) network as its backbone convnet. Similarly to EfficientDet, the output feature maps are then processed by the model neck, in this case, feature maps are passed to the PANet for feature fusion. Finally, the data is then fed to the classification and bounding box regressor to output the prediction and the coordinates of the bounding box.

The underlying ConvNet used in the YOLOv5 architecture is the CSPNet which is an improvement over the DenseNet. Figure 10, shows the difference between the two architecture. The improvement made here is that by partitioning the input features into 2 different portions. The first portion will go through the dense layer whereas the second portion will only be merged in the transition layer. This proposed architecture allows the network to preserve the advantages of DenseNet by reusing gradient but at the same time reduce the amount of duplicated gradient information.

3.4.2.2 Model Neck–Path Aggregation Network (PANet)

Similar to the EfficientDet architecture, the model neck uses an FPN to combine useful features before passing it to the classification layer. The Path Aggregation Network (PANet) is an upgrade to the FPN used in YOLOv2 and YOLOv3. In YOLO architecture, as the complexity of features increases, the spatial resolution of the image decreases, so in order to combine features (which exist in the top layers) and their precision positioning (which exist in the bottom layer), the FPN has to traverse down numerous layers which can be computationally inefficient. Hence, the PANet introduces another bottom-up approach linkage such that the network is able to take a “shortcut”, and it is only around 10 layers

3.5 Evaluating the model performance

After reviewing the model architecture, this segment will list the metrics that will be used to generate the statistics that are required to evaluate the object detection model’s accuracy and performance. This section will also include the benchmark metrics as the model will be deployed onto the NVIDIA Jetson Nano.

3.5.1 Precision

$$Precision = \frac{TP}{(TP + FP)}$$

The precision of the model will calculate the probability that the predicted bounding box outputted by the model matches the actual ground truth bounding box. It refers to the positive predictive value.

3.5.2 Model Recall

$$Recall = \frac{TP}{(TP + FN)}$$

The recall metric will assess the sensitivity or truth positive rates of the model, the higher the recall means that the model manages to successfully predict correct values.

3.5.3 F1-Harmonic Mean

$$F1\ Harmoic\ Mean = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

3.5.4 Mean Average Precision (mAP)

$\text{Mean Average Precision (mAP)} =$

$$\sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

AP metric allows us to summarize the precision and recall curve into one singular value representing the average precision of all precisions.

3.5.5 Intersection over Union (IoU)

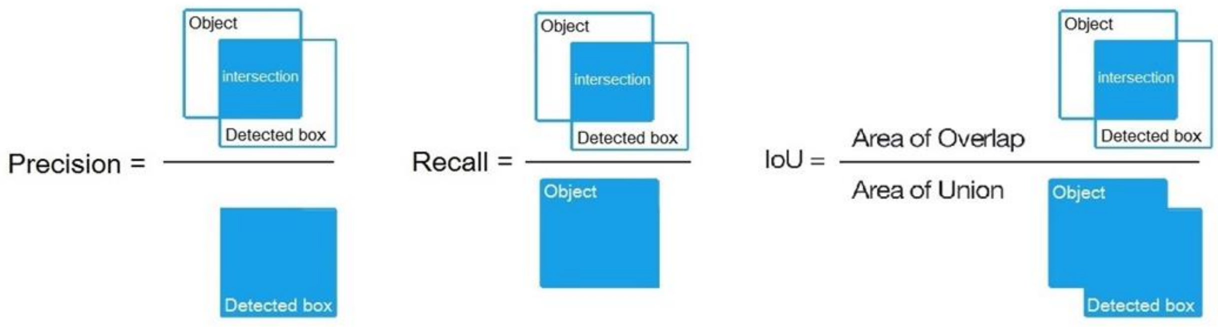

Figure 11- Illustration of Intersection over Union (IoU)

Intersection over union (IoU) calculates how well the predicted bounding box matches the ground truth bounding box. Typically, the IoU, is done by calculating the intersect sections over the total union of both bounding boxes. For the study, the IoU threshold will be evaluated across different IoU values such as 0.5 and 0.5:0.95.

3.5.6 Binary Cross Entropy Loss

Both object detection model uses Binary Cross-Entropy Loss for loss calculation

$$BCELoss \ = -wn[yn \cdot \log\sigma(xn) + (1 - yn) \cdot \log(1 - \sigma(xn))],$$

The loss optimization function used in the YOLOv5 and EfficientDet architecture uses the Stochastic gradient descent (SGD) optimizer from the PyTorch library, and uses the value of the BCELoss to updates the values of the model weights.

3.6 Model Optimization

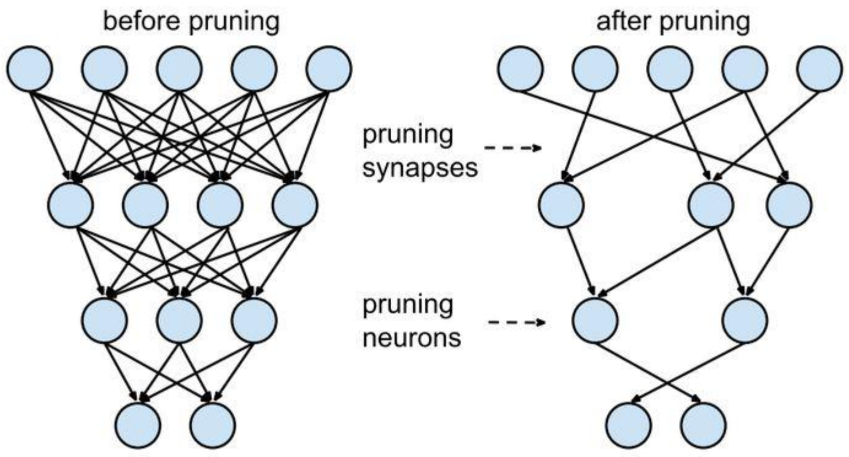

Figure 12 - Model Pruning effects on Neural Networks

The model optimization process comprises several stages. First, pruning and obtaining a base neural network architecture. The neural network is then finetuned on a generic large dataset such as the common objects in context (COCO), which comprises 1.5 million types and 80 classes of objects. Training on a large dataset allows the neural network to learn common patterns and important features. Sparse transfer learning is then applied to the prune and quantized model to transfer important weight to downstream tasks such as face mask detection. After the model training, the sparse model will be used to perform benchmarking to determine the usefulness of the model optimization process. Benchmarking will test and compare the base model against the optimized model against metrics such as:

- Object detection speed – Measured in milliseconds (ms), how fast does the model perform pre-processing, prediction and inference.
- Memory footprint – What is the reduction in terms of the model size in megabytes (MB).

3.7 Deployment

NVIDIA Jetson Nano will be used as the edge device to facilitate the real-time deployment of the model. The image will be streamed using the JVCU100 HD USB Webcam, which will then be processed by the OpenCV library and supplied to the model for object detection and displayed on the front-end.Table 6 summarizes the hardware specification of the edge device as well as the camera.

4. Results and Analysis

4.1 YOLOv5s and EfficientDet training results

Both YOLOv5 and EfficientDet models were trained using a runtime with Google Colab with an NVIDIA Tesla P100 PCIe 16GB GPU with 3584 cores. For the YOLOv5 model, the YOLOv5s variant was chosen. It has a total of 7M trainable parameters and is the second lightest model among the other variants. The model was trained several times and the best hyperparameters that worked well for the dataset to converge were as follows, learning rate 0.01, batch size of 32 and 50 epochs. The model was trained using the SGD optimizer with momentum and weight decay. Similarly, for the EfficientDet Architecture, there exist different variants of the model, from EfficientDet B0 to the EfficientDet B7 model, with B7 having the highest memory, FLOPS, and model size. EfficientDet-B0 was selected as the base model, with a total of 3.9M total trainable parameters, but with a large input size of 512 by 512 as compared to YOLOv5s with an input size of 416 by 416.

Table 3- Model Training results

Model	**mAP @ IoU (0.5)**	**mAP @ IoU (0.5:.95)**	**Precision**	**Recall**	**F1-Score**	**Frames**
YOLOv5s	0.94	0.745	0.966	0.918	0.941	10 FPS
EfficientDet-B0	0.932	0.723	0.932	0.793	0.856	1-2 FPS

Table 3 summarizes the model results of both the YOLOv5s and the EfficientDet-B0, it shows the mean average precision of the intersection over the union of both the correct and incorrect mask class at 0.5, 0.5 to 0.95 IoU threshold, meaning that only bounding box that is above the specific threshold with respect to the ground truth, would be accepted as a correct prediction. Using the details, the F1 score was also computed for both models.

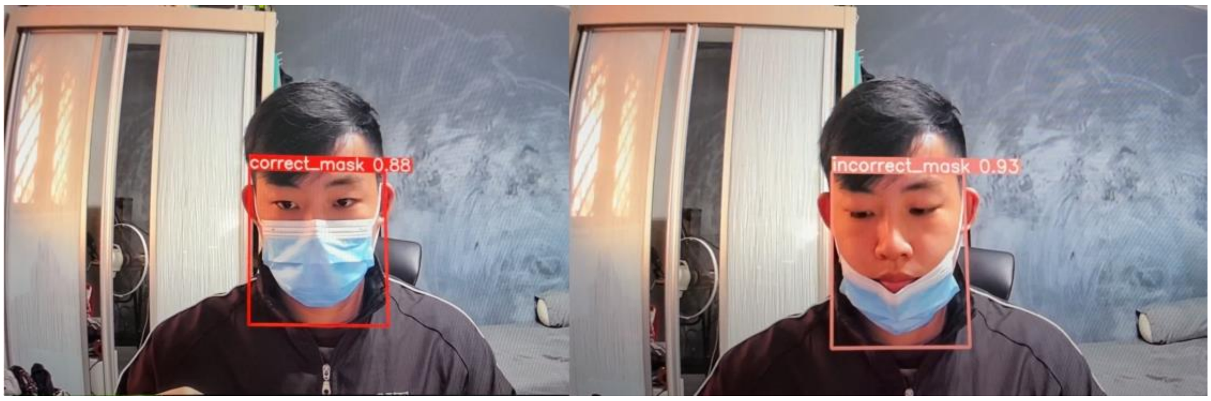

Figure 13- Real Time YOLOv5 detection on NVIDIA Jetson Nano

The YOLOv5s model performed well across all categories with a strong precision and recall score. Figure 18 shows the mAP, precision and recall graph, the model was able to achieve up to 0.94 for mAP @ 0.5 and 0.745 for mAP @ 0.5:0.95 which outperformed the EfficientDet-B0 model. After several iterations of training to get the optimal hyperparameters, both managed to converge their loss and there were no signs of model overfitting. Overall the YOLOv5s model managed to converge to 0.004 (4e-3), while the EfficientDet-b0 model managed to converge to 0.19 for classification loss. In terms of the F1 score, it calculates the harmonic mean of the precision and recall of the model, which considers the robustness of the error against false positive and false negative readings. The YOLOv5 model was able to achieve a higher score of 0.941 as opposed to 0.856 for the EfficientDet-b0 model. Figure 19 and Figure 20 show the sample output of the YOLOv5s and EfficientDet-b0 model respectively, using the test images from the dataset. Using OpenCV and Python, the FPS was calculated based on the processing time required to process every single frame from the camera. YOLOv5s was able to obtain a much higher FPS performance as compared to the EfficientDet-b0 model, Figure 13 shows sample predictions and the model is extremely robust against background noises and is also able to constantly accurately predict and track both correct and incorrectly worn mask with an IoU score above 0.5.

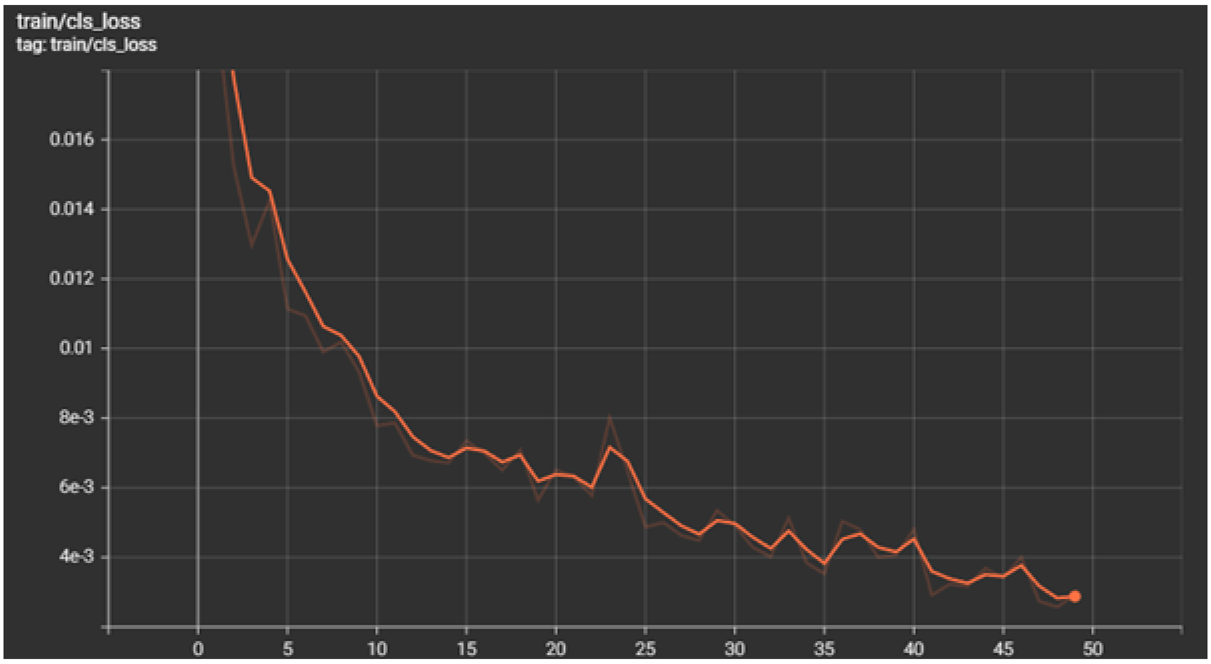

Figure 14 - YOLOv5 Classification Loss

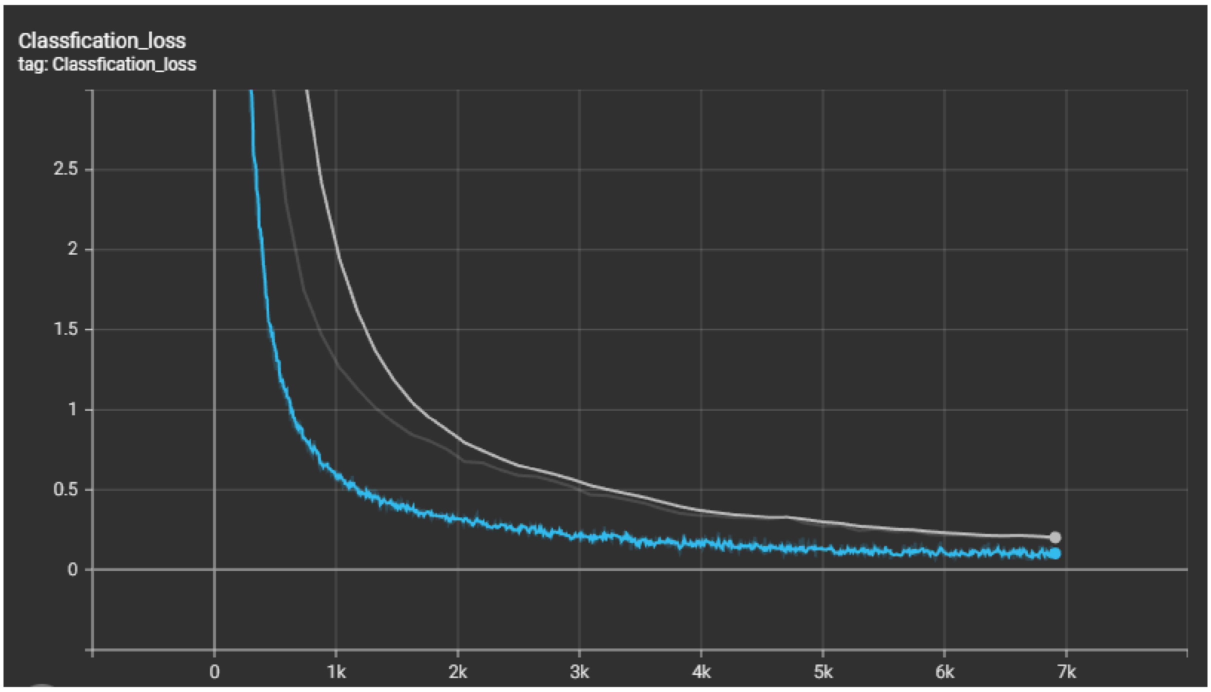

Figure 15- EfficientDet Cross Entropy Loss

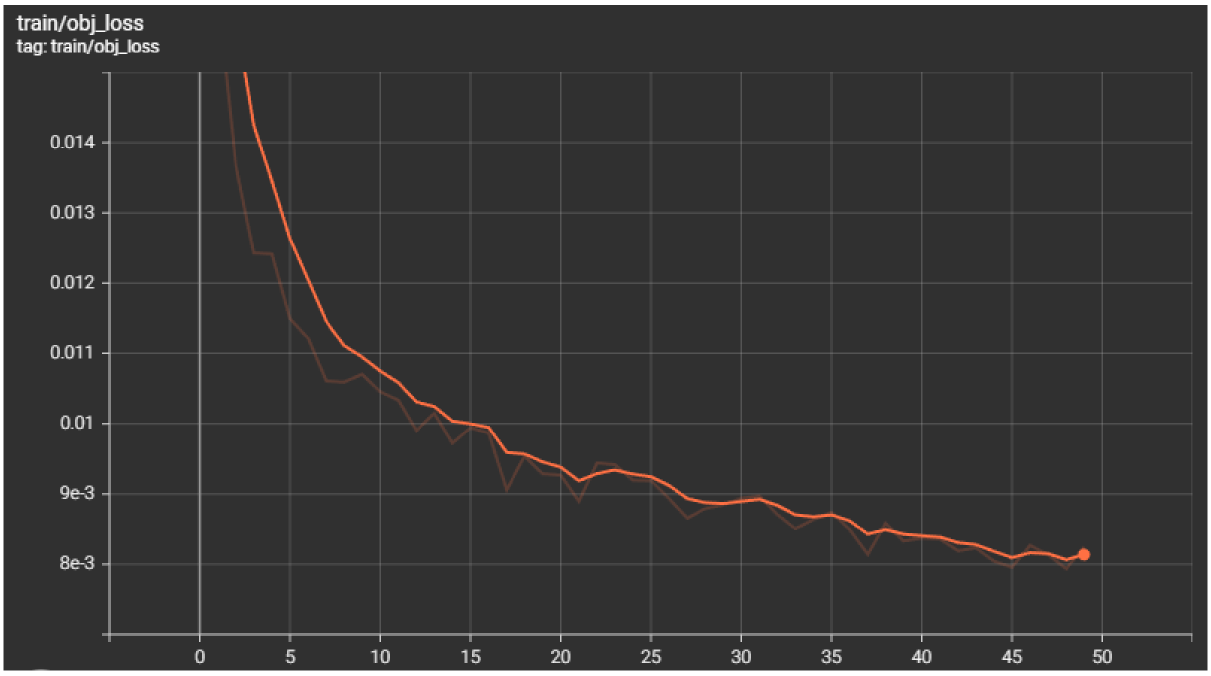

Figure 16 - YOLOv5 Binary Cross Entropy loss

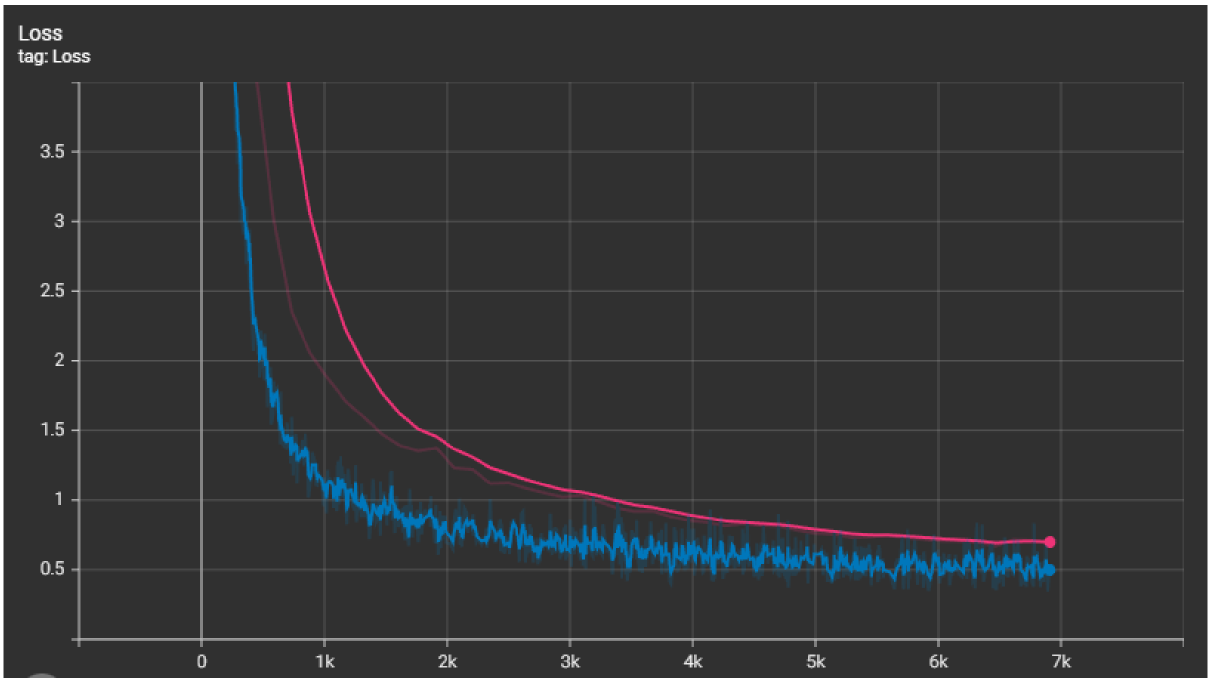

Figure 17 - EfficientDet Binary Cross Entropy loss

Figure 15 and 16 show the classification loss computed using Cross Entropy while Figure 16 and 17 show the object loss using Binary Cross Entropy of the YOLOv5 and EfficientDet model respectively. Overall, both model manages to stabilize and converge, with YOLOv5s converging at a loss value of 0.008 and EfficientDet at a value of 0.5.

4.2 Model optimization results

In order to evaluate the effectiveness of the network quantization, an experiment was conducted on different YOLOv5s models, namely the base model, the network pruned and the network pruned and quantized model. The sparse models have been trained on the COCO dataset to recognize important features. Sparse transfer learning was then applied to transfer weights to perform COVID-19 Face Mask Detection. All the 3 models were then converted from PyTorch to ONNX format and benchmarked on a Windows 10 machine with an 11th Gen Intel ® Core i7-11635G7.

Table 4- Results of Speedup after Network Optimisation

Model	Inference speed (ms)	Model Size (KB)	Speedup
YOLOv5s Base model	Items inferred per second: 14.6928256373676 ms_per_batch: 68.06042790412903 batch_times_mean: 0.06806042790412903 batch_times_median:0.06692051887512207 batch_times_std: 0.007747013819069028 End-to-end per image time: **68.06042790412903**	28,438	0%
YOLOv5s (Pruned Model)	Items inferred per second: 26.5019482085272 ms_per_batch: 37.73307502269745 batch_times_mean: 0.03773307502269746 batch_times_median: 0.01947343349456787 batch_times_std: 0.08469350987664258 End-to-end per image time: **37.73307502269745**	7,028 (compressed **75%**)	44.5595%
YOLOv5s (Pruned and Quantized model)	Items inferred per second: 47.9594443899721 ms_per_batch: 20.850950479507446 batch_times_mean: 0.020850950479507448 batch_times_median: 0.020953893661499023 batch_times_std: 0.0028645656390089646 End-to-end per image time: **20.850950479507446**	7,028 (compressed **75%**)	69.3641%

The network quantization experimentation proved that both network pruning and quantization have a huge impact on the overall performance of the system. Table 4 shows the benchmarked results, the number of inferences per second increased from 14.69ms to 26.50ms by 80.3% and the end-to-end image processing time decreased from 68.06ms to 37.73ms by 44%

The pruned model was further quantized from a Floating-point 32 (FP32) precision down to Integer 8-bit (INT-8) precision and benchmarked against the YOLOv5s base model. The number of inferences per second increased by a large margin from 14.69ms to 47.95ms by 226%. The end-to-end image processing also decreased from 68.06ms to 20.85ms by almost 70%. The model also saw great results in the reduction of the model memory, it managed to get compressed by 75% from 28.4 MB down to just 7 MB.

4.2.1 Analysis of Sparsification on Model Performance

Table 5- Impact of Sparse training methods

Model	**Precision**	**Recall**	**mAP 0.5**	**mAP 0.5:0.9**
YOLOv5s (Base model)	0.966	0.918	0.94	0.745
YOLOv5s (Post Network-Prune)	0.97782	0.9225	0.93788	0.72493
YOLOv5s (Post Network-Prune and Quantization)	0.97217	0.925	0.94125	0.73342

Another important finding is the impact of sparse transfer learning on the accuracy of the model. Table 5 shows the results of the sparse transfer learning of all 3 models. It can be seen that there was an insignificant impact on the model precision and recall, and there was only a slight loss in terms of the mAP 0.5:0.95 value. It is concluded from the results that introducing sparsity into the network using pruning and quantization, has an insignificant impact on the model performance whilst greatly compressing the model size and increasing its inference speed.

5. Conclusion

Both object detection models showed promising results based on the evaluation metrics, achieving significant accuracy against test datasets with low figures in losses. Both lightweight models were also able to be deployed on resource-constrained edge devices such as the Jetson Nano. The YOLOv5s model managed to perform real-time COVID-19 Face mask detection on the NVIDIA Jetson Nano with live footage coming from the JVCU100 camera with a high 10 FPS with relatively high accuracy and confidence. The lightweight detectors that was trained in this research managed to outperform majority of the existing COVID-19 FMD systems reviewed related works in terms of metrics such as the average precision (AP), mAP and also a higher F1-score. Most importantly, the detectors were able to achieve the results which much lower trainable parameters and memory consumption.

In this paper, we have shown and proven tremendous improvements in the Single-shot detector model performance by using network pruning and quantization techniques and performing sparse transfer learning on the optimized model. By significantly reducing redundant network parameters and quantizing the precision of the model, it has been shown to reduce and compress the model drastically whilst accelerating the end to end inference speed. The sparsification process reduces the overall memory footprint, and the reduced parameters and compution conserves energy and bandwidth needed to deploy the deep learning model. This is hugely desirable for resource constrained edge devices such as Raspberry Pi 3B+ and NVIDIA Jetson Nano. This process of network optimization will definitely be an important key to real-time deployment of deep learning of things.

Overall, two-stage detectors are generally accurate but they are much slower, and hence they are not considered capable of real-time inference and object detection. From recent years, single-shot detectors such as SSD and YOLO have proven to be able to match the accuracy of the former whilst being able to perform real-time inference. As edge computing trends continues to increase, higher demands for on-devices machine learning, and deep learning will result in greater need in lightweight, efficient and equally accurate models.

6. References

[1] World Health Organization, “WHO Coronavirus (COVID-19) Dashboard,” 2022.

[2] R. v. Tso and B. J. Cowling, “Importance of Face Masks for COVID-19: A Call for Effective Public Education,” *Clinical Infectious Diseases*, vol. 71, no. 16. Oxford University Press, pp. 2195–2198, Oct. 15, 2020. doi: 10.1093/cid/ciaa593.

[3] A. Nowrin, S. Afroz, M. S. Rahman, I. Mahmud, and Y. Z. Cho, “Comprehensive Review on Facemask Detection Techniques in the Context of Covid-19,” *IEEE Access*, vol. 9. Institute of Electrical and Electronics Engineers Inc., pp. 106839–106864, 2021. doi: 10.1109/ACCESS.2021.3100070.

[4] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Sep. 2014, [Online]. Available: http://arxiv.org/abs/1409.1556

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, “Rethinking the Inception Architecture for Computer Vision.”

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015, [Online]. Available: http://arxiv.org/abs/1512.03385

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” [Online]. Available: http://pjreddie.com/yolo/

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” Nov. 2013, [Online]. Available: http://arxiv.org/abs/1311.2524

[9] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.01497

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection.” arXiv, 2020. doi: 10.48550/ARXIV.2004.10934.

[11] J. A. Kim, J. Y. Sung, and S. H. Park, “Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition,” Nov. 2020. doi: 10.1109/ICCE-Asia49877.2020.9277040.

[12] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” Dec. 2016, [Online]. Available: http://arxiv.org/abs/1612.08242

[13] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.02767

[14] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.

[15] E. Mbunge, S. Simelane, S. G. Fashoto, B. Akinnuwesi, and A. S. Metfula, “Application of deep learning and machine learning models to detect COVID-19 face masks - A review,” *Sustainable Operations and Computers*, vol. 2, pp. 235–245, 2021, doi: 10.1016/j.susoc.2021.08.001.

[16] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proceedings of the IEEE*, 2019, doi: 10.1109/JPROC.2019.2921977.

[17] H. Li, K. Ota, and M. Dong, “Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing,” *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan. 2018, doi: 10.1109/MNET.2018.1700202.

[18] M. Tan, R. Pang, and Q. v. Le, “EfficientDet: Scalable and Efficient Object Detection,” Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.09070

[19] K. Paupamah, S. James, and R. Klein, “Quantisation and pruning for neural network compression and regularisation,” Jan. 2020. doi: 10.1109/SAUPEC/RobMech/PRASA48453.2020.9041096.

[20] X. Zhou and S. L. Keoh, “Deployment of facial recognition models at the edge: A feasibility study,” in *APNOMS 2020 - 2020 21st Asia-Pacific Network Operations and Management Symposium: Towards Service and Networking Intelligence for Humanity*, Sep. 2020, pp. 214–219. doi: 10.23919/APNOMS50412.2020.9236972.

[21] David Correa, “Face mask detection market to Reach $4.12 Bn, Globally, by 2030 at 8.1% CAGR: Allied Market Research,” *Allied Market Research*.

[22] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 2017, [Online]. Available: http://arxiv.org/abs/1704.04861

[23] M. Tan and Q. v. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” May 2019, [Online]. Available: http://arxiv.org/abs/1905.11946

[24] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A Survey of Modern Deep Learning based Object Detection Models,” Apr. 2021, [Online]. Available: http://arxiv.org/abs/2104.11892

[25] Lionel Sujay Vailshery, “Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030,” *Statista*, 2022.

[26] S. Anwar, K. Hwang, and W. Sung, “Structured pruning of deep convolutional neural networks,” *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 3, Feb. 2017, doi: 10.1145/3005348.

[27] G. J. Chowdary, N. S. Punn, S. K. Sonbhadra, and S. Agarwal, “Face Mask Detection using Transfer Learning of InceptionV3,” Sep. 2020, doi: 10.1007/978-3-030-66665-1_6.

[28] B. Mandal, A. Okeukwu, and Y. Theis, “Masked Face Recognition using ResNet-50,” Apr. 2021, [Online]. Available: http://arxiv.org/abs/2104.08997

[29] J. Gathani and K. Shah, “Detecting Masked Faces using Region-based Convolutional Neural Network,” in *2020 IEEE 15th International Conference on Industrial and Information Systems, ICIIS 2020 - Proceedings*, Nov. 2020, pp. 156–161. doi: 10.1109/ICIIS51140.2020.9342737.

[30] J. Zhang, F. Han, Y. Chun, and W. Chen, “A Novel Detection Framework about Conditions of Wearing Face Mask for Helping Control the Spread of COVID-19,” *IEEE Access*, vol. 9, pp. 42975–42984, 2021, doi: 10.1109/ACCESS.2021.3066538.

[31] P. Nagrath, R. Jain, A. Madan, R. Arora, P. Kataria, and J. Hemanth, “SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2,” *Sustainable Cities and Society*, vol. 66, Mar. 2021, doi: 10.1016/j.scs.2020.102692.

[32] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, “Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection,” *Sustainable Cities and Society*, vol. 65, Feb. 2021, doi: 10.1016/j.scs.2020.102600.

[33] S. Degadwala, D. Vyas, U. Chakraborty, A. R. Dider, and H. Biswas, “Yolo-v4 Deep Learning Model for Medical Face Mask Detection,” in *Proceedings - International Conference on Artificial Intelligence and Smart Systems, ICAIS 2021*, Mar. 2021, pp. 209–213. doi: 10.1109/ICAIS50930.2021.9395857.

7. Knowledge and Training Requirements

7.1 Applicable Knowledge from the Degree Programme

The prerequisite knowledge and skillsets from the degree programme that was necessary for the capstone projects are as follows:

No.	Module(s)	Knowledge(s) Applied
1	CSC1007 – Operating Systems	Interfacing with different operating systems throughout the projects such as Windows, Windows Sub-system for Linux (WSL2), and Ubuntu 18.04. The module taught important concepts of the Linux file system and it was helpful when I was configuring and developing the Linux-based systems. I also had to employ swap files on the NVIDIA Jetson Nano in order to successfully build the required libraries such as OpenCV, PyTorch, and TorchVision as 4GB RAM is insufficient.
2	CSC1009 – Object-oriented programming	The module introduced and taught Python, which is an extremely popular and widely used language for backend and data science projects. This capstone project was predominantly built using Python. I was able to able concepts such as OOP, classes, objects and inheritance in this project.
3	CSC2003 – Embedded Systems	The embedded systems project gave me a lot of first-hand experience in micro-controller programming in Raspberry Pi and also the NVIDIA Jetson Nano. I was able to apply the concept of quantisation when doing the experimentation on network optimisation.
4	CSC2006 – Internet of Things: Protocols and Networks	This module introduced important computing paradigms such as Edge and fog computing, I was able to leverage the knowledge learn and expand it to deep learning in the Internet of Things (IoT) devices.
5	CSC3005 – Data Analytics	The module covers important Machine Learning algorithms ranging from supervised to unsupervised learning. It also taught us the basics of using Python, Numpy and data science packages. During the project I was able to able the end-to-end machine learning workflow similar to our CSC3005 project, from data pre-processing to model training and so on. The knowledge of data science package could also be utilized when using higher level deep learning libraries such as PyTorch, which was extensively used in this data science project.
6	CSC3009- Machine Learning	The module covered a wide range of deep learning algorithms such as the basic multi-layer perceptron to complex algorithms such as convolutional neural networks (CNN). I was able to higher mathematical understanding and appreciation of deep learning algorithms. Some areas that I was able to apply knowledge were loss computation using stochastic gradient descent, hyperparameter tuning as well as techniques on how to compute model accuracy like Intersection over Union (IoU)

7.2 Additional Knowledge, Skillsets, or Certifications Required

The following are the additional requirements and knowledge that were required for the capstone projects:

No.	Additional Requirement(s)	Knowledge(s) Applied
1	Linux (Aarch64 architecture)	As the project is using deep learning libraries such as Pytorch, Torchvision, and OpenCV. It has to be set up on the NVIDIA Jetson Nano. The official pip wheel files from NVIDIA or online repositories are either outdated or have compatibility issues. I have to learn how to use swap files and CMake to build these libraries on the edge device itself to ensure that it was working.
2	Computer Vision	Understanding of OpenCV is required to build the deployment scripts for the EfficientDet model, also implemented a multithreading approach for the deployment script.

Appendices

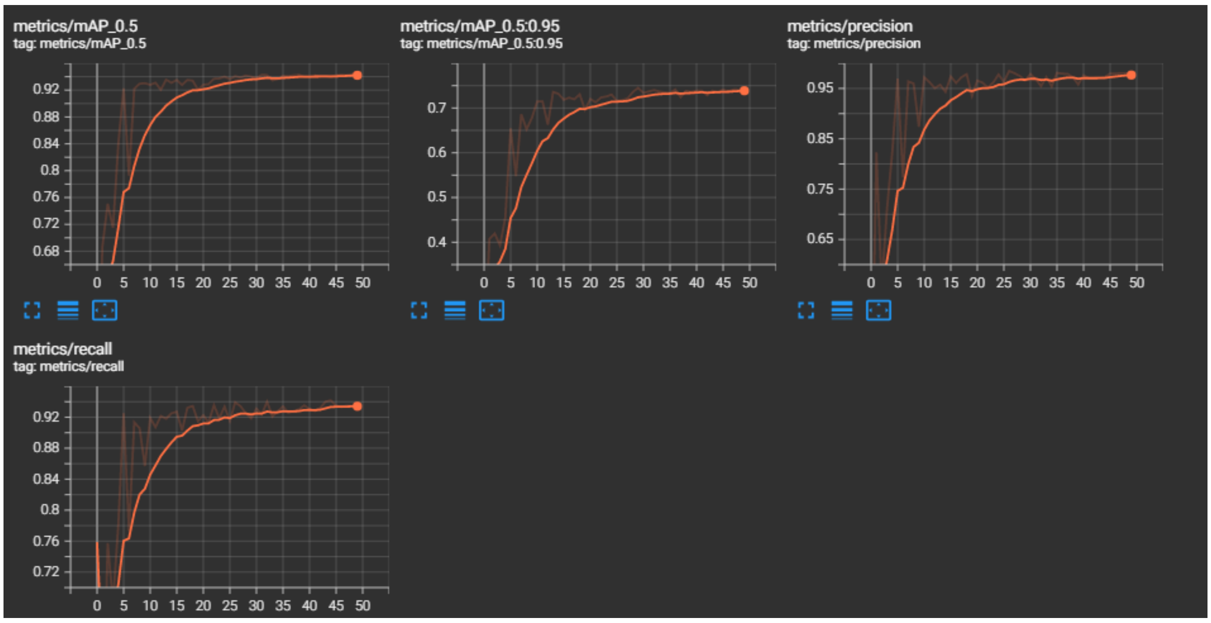

Figure 18 - YOLOv5, Precision, Recall and mAP

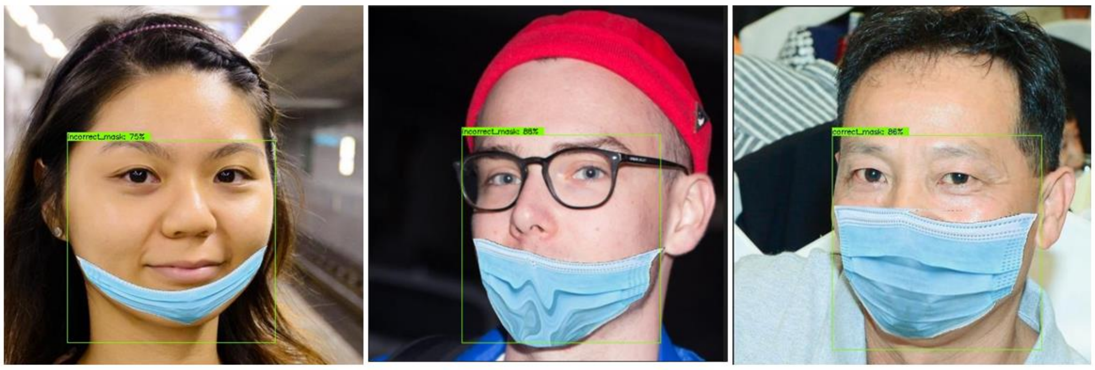

Figure 19 - EfficientDet detection results sample

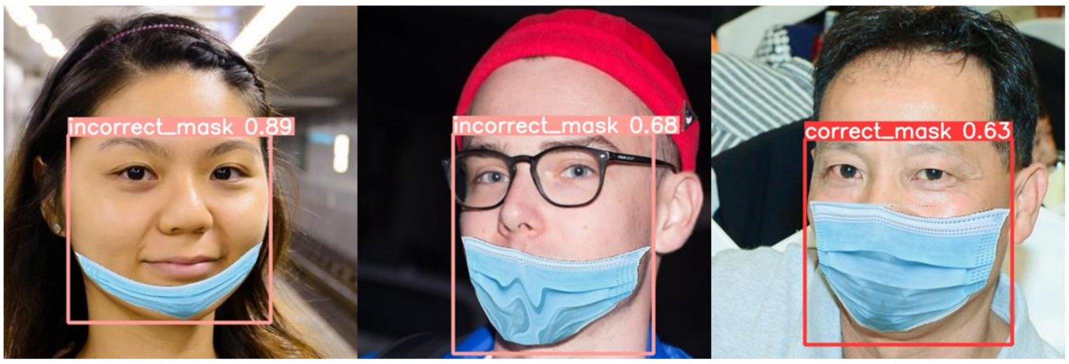

Figure 20- YOLOv5 detection example

Table 6 - Equipment specifications

Devices	**Specification**
	• USB 2.0 Type A • 1/2.9" CMOS Sensor • 1080P / 720P / 640P
	• CPU: Quad-core ARM Cortex-A57 MPCore processor • GPU: NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores • Memory: 4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s • 4x USB 3.0, USB 2.0 Micro-B

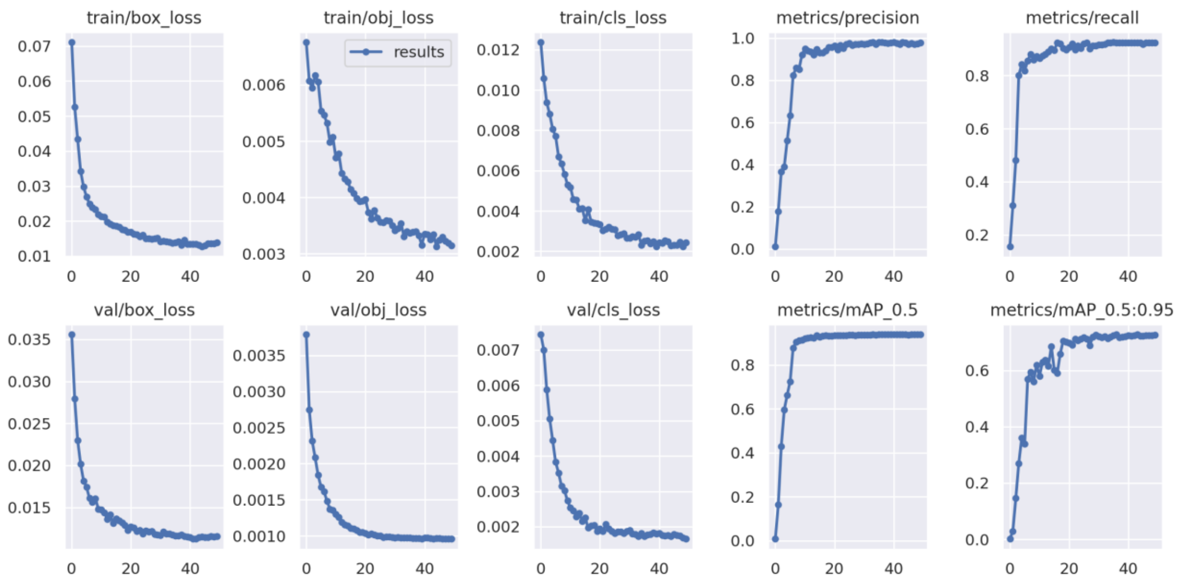

Figure 21 - YOLOv5 Sparse transfer results on Pruned Model

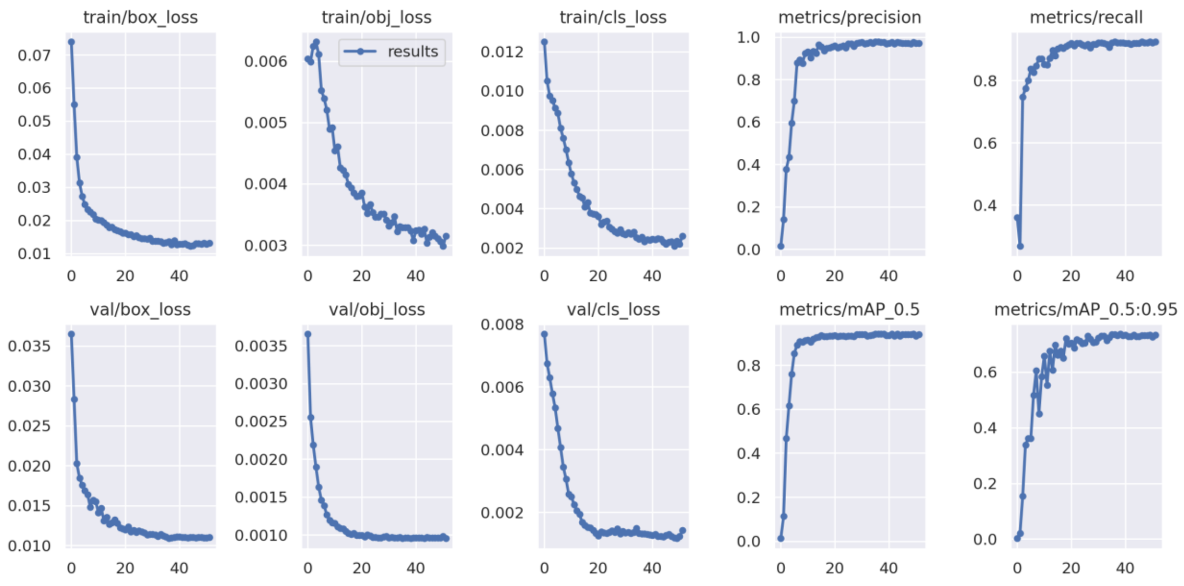

Figure 22- YOLOv5 Sparse transfer results on Pruned and Quantised model

END OF REPORT