COMP1682 FINAL YEAR PROJECT

DETECTION OF FACE MASK PRESENCE USING CONVOLUTIONAL NEURAL NETWORKS AND MACHINE LEARNING

Dominykas Stasiulionis - ds2182y@gre.ac.uk - 01044522

A dissertation submitted in partial fulfilment of the University of Greenwich undergraduate degree programme

BEng (Hons) Software Engineering

Supervisor: **Dr Markus Wolf**

Due: **26th April 2021**

Word count: **39,847 (Excluding Bibliography & Appendices)**

Contents

Abstract

In a world prior to the COVID-19 virus, a face mask was not so commonly worn in public. However, amidst the pandemic, wearing a face mask can reduce the risk of spreading infection and also from contracting the virus yourself. Thus, it is an essential defence that must be utilised where possible. Through scientific research and studies, people globally have been informed of how important wearing a face mask has become to preventing the spread of the virus. As a result of this, government regulations have been instated around the world to ensure people are wearing face masks in commonly-visited places and close-quarter environments such as shops, train stations, buses, etc. With the use of a convolutional neural network and machine learning, this project aims to create a lightweight and accurate face mask detection model capable of real-time detections. The main objective being to enforce the health and safety policies set into place by identifying those who wear masks and those who do not. With a computationally lightweight model, a potential use-case includes integrating the model with surveillance cameras for real-time monitoring.

Acknowledgements

Firstly, I would like to express my appreciation to God for giving me the strength and mental resilience needed to complete the course and the academic year through these difficult times.

I would also like to express my gratitude to Dr Markus Wolf for being an excellent supervisor who provided helpful advice and guidance throughout the development of the project.

List of Abbreviations

WHO – World Health Organisation

HEPA – High Efficiency Particulate Air

PPE – Personal Protective Equipment

ML – Machine Learning

mAP – Mean Average Precision

FPS – Frames Per Second

AI – Artificial Intelligence

NN – Neural Network

ANN – Artificial Neural Network

CNN – Convolutional Neural Network

DNN – Deep Neural Network

DCNN – Deep Convolutional Neural Network

SSD – Single Shot Multi-Box Detector

DL – Deep Learning

mAR – Mean Average Recall

ILSVRC – ImageNet Large Scale Visual Recognition Challenge

CV – Computer Vision

GPU – Graphical Processing Unit

CPU – Central Processing Unit

SVM – Support Vector Machine

RGB – Red Green Blue

IOU – Intersection Over Union

TP – True Positive

FP – False Positive

TN – True Negative

FN – False Negative

AP – Average Precision

AR – Average Recall

FC – Fully-Connected Layer

1

Introduction

1.1 Background

On March 11th 2020, COVID-19 was declared a pandemic as a result of its territorial widespread and infection of a multitude of people across a plethora of countries. According to WHO (2021), there have been 134,308,070 confirmed cases of COVID-19, including 2,907,944 deaths as of April 8th 2021. As the virus continues to uphold its grasp on the world, it also causes major impact to people without COVID-19 as well. Many medical resources had been redirected towards caring for COVID-19 patients, due to this change, many medical patients without COVID-19 were not able to receive treatment (Rosenbaum, 2020). Furthermore, the virus has impacted other areas such as research as travel, social, and funding restrictions also took a serious toll on scientific research worldwide. Research staff and resources have been purposely and purposefully prioritized to COVID-19 activities above all else (Harper et al., 2020). Other areas impacted include students' mental health, as it is greatly affected when faced with a public health emergency, and they need attention, assistance, and support from the community, family, and tertiary institutions. About 24.9% of students have experienced anxiety because of this COVID-19 outbreak (Pragholapati, 2020). The name “COVID-19” was chosen because the virus is genetically related to the coronavirus responsible for the SARS outbreak of 2003. While related, the two viruses are different (World Health Organisation, 2020). Bioinformatic analysis have shown that SARS-CoV-2 belongs to the genus of β-coronavirus and shares at least 70% similarity in genetic sequence with SARS-CoV (Yang, Shang and Rao, 2020). One difference between these two viruses is that COVID-19 spreads faster than its predecessor variant. A handful of genetic and structural analysis have identified a key feature of the virus — a protein on its surface — that might explain why it infects human cells so readily (Mallapaty, 2020). A Severe Acute Respiratory Syndrome coronavirus 2 (SARS-CoV-2), the virus responsible for causing COVID-19, is primarily transmitted from person-to-person through close contact (approximately 6 ft) by respiratory droplets (Chavez et al., 2020).

Due to this, many health and safety measures have been put into place, one of these measures include the use of face masks. Policymakers are being challenged to enforce these safety measures, with some measures such as wearing a face mask becoming law in certain countries (Altmann et al., 2020). Wearing a mask has become a vital asset in defending the health of yourself and others. Masks are a type of PPE used to prevent the spread of respiratory infections. These masks cover the mouth and nose of the wearer and if worn properly, may be effective at helping prevent transmission of respiratory viruses and bacteria (Desai and Mehrotra, 2020). Without the use of a face mask, there is little one can do to contain such a contagious virus in a public setting. As Liu and Zhang (2020) mention in their report, there was a situation involving one male patient with COVID-19 from Chongqing, China, who found himself coughing. Unaware of the fact that he might have been infected with COVID-19 and in a hurry, he did not manage to get a face mask before he

took the coach bus from the city back to his county. Many passengers did not wear face masks on the same coach bus. The duration of this bus was 2 hours and 10 minutes, and there were 39 other passengers on the same coach bus. According to an epidemiological survey, 5 other passengers on the same coach bus were infected. Furthermore, WHO created a list of methods that can be followed to reduce the spread of the virus. Within that list, face masks had also been included: "Wearing a medical mask if you have respiratory symptoms and performing hand hygiene after disposing of the mask." (World Health Organisation, 2020). In the early stages of the outbreak, Japan's panel advice for communities stated that the effectiveness of wearing a face mask to protect yourself from contracting viruses is thought to be limited. If you wear a face mask in confined, badly ventilated spaces, it might help avoid catching droplets emitted from others but if you are in an open-air environment, the use of a face mask is not very efficient (Feng et al., 2020). However, even with a limited protective effect, face masks can reduce total infections and deaths (Worby and Chang, 2020). The masks can be used to block respiratory transmission from human to human and are an effective way to control influenza. Therefore, it is necessary to wear a mask when respiratory infectious diseases are prevalent (Wang et al., 2020). A study that looked at the reduction of secondary transmission of COVID-19 in households, found that face masks were 79% effective at preventing the transmission of the virus. This solidifies the points made by the Japanese panel in the paper by Feng et al., (2020) regarding face masks being effective at preventing spread in close quarter environments. It is important to investigate the impact of face masks in smaller environments as some of these environments include shops, public transport, buildings, train stations, etc. These environments are where the virus can dominate and spread most since social distancing will be reduced and sometimes even broken. Research showed that social distancing measures were most effective when large-scale return to work took place in early April. This reduced the median number of infections by 92% (Qian and Jiang, 2020). Therefore, in smaller commonly-visited areas where social distancing cannot always be maintained, it is even more imperative that face masks are worn to avoid infection and further potential spread. Due to the protection a face mask can provide against the spread of COVID-19, face masks have risen in popularity and demand. In any scenario, an N95 mask shortage was predicted to occur on 24 January 2020 with a daily facemask shortage of 2.2 million (Wu et al., 2020). For health-care workers and administrators, findings suggest that N95 respirators might be more strongly associated with protection from viral transmission than surgical masks. Both N95 and surgical masks have a stronger association with protection compared with single-layer masks (Chu et al., 2020).

The type of face mask that is being worn can also contribute to the risk of being infected. This is dependent on the material and the craft of the face mask as some have a better resistance to particle penetration. A study was conducted that compared the efficiency of cloth masks against medical masks in hospital healthcare workers. The results of this study caution against the use of cloth masks as moisture retention, poor filtration and cloth mask reuse could result in a greater risk of infection. The study concludes that penetration of cloth masks by particles was almost 97% and medical masks 44% (MacIntyre et al., 2015). The projection of secretions from the upper airways or saliva that may contain infectious agents transmissible by 'droplet' or 'air' routes (Lepelletier et al., 2020). Infected hosts can help prevent the propagation of the virus by donning a face mask covering their mouth and nose to disrupt the airflow near the source (Galbadage et al., 2020). Moreover, studies had been reviewed in order to gain insight into what types of masks people commonly wear. Firstly, A study was recently conducted to assess the use of face masks among young adults during the current COVID-19 pandemic in Poland (Matusiak et al.,2020). 2315 people participated in the study; the types of face masks being worn was also recorded. The study concluded that cloth masks (46.2%) appeared to be the most popular ones, followed by surgical masks (39.2%), respirators (N95 and FFP) (13.3%), half-face elastomeric respirators (0.8%) and full-face respirators (0.4%). A study in Germany, surveyed people regarding their views on wearing masks. 20% of responders had said that they would probably not wear a face mask even if it were legally required. This suggests that policymakers should be rather cautious about making it compulsory, as compliance could become a

larger issue (Rieger, 2020). As mentioned previously, smaller/tighter areas such as public transport and shops almost promote the exponential growth of COVID-19 if no protective actions or measures are taken. A study was created that examined how people adhere to face mask and distancing policies while using public transport. The study had found that about 12.6% of the vehicles had fewer than three commuters without face masks, while 21.3% of buses that had fewer than 3 people with face masks. The results suggest public transport remains an area of high risk in the fight against COVID-19 (Dzisi and Dei, 2020). There are people who understand the risks of not following policies put in place as well as understand the positive impact those policies can have on their health and others. However, even with this knowledge, some still actively choose to not wear a face mask in situations when they are supposed to be worn. Despite some studies showing that only a small percentage of people at times behave like this, it only takes one person to infect another and create a chain reaction of infections. For example, on January 14th 2020, WHO had confirmed that there were 41 confirmed cases and warned of a potential wider outbreak. 16 days later, the total number of confirmed cases rises to 7818, with 82 cases being reported in 18 other countries outside China. This is evidence that every situation where policy is not followed must be treated seriously, as inaction can cost lives. Further evidenced by the 134,308,070 people that have been directly impacted by contracting COVID as of April 8th 2021, not to mention the side effects this has on the people around them. Additionally, not wearing a mask has been perceived as socially incorrect during the pandemic. As a study by Betsch et al., (2020) from Germany reported. The study denotes that mask wearing was perceived as a social contract, as those who complied with it socially "rewarded" each other but "punished" others who did not wear a mask. There will always be people who do not wear masks in public settings where masks are required for their own objective or subjective reasons. Especially in close quarter settings such as public transport, as previously discussed in the report by Liu and Zhang (2020). Consequently, the man from China infected multiple people on public transport by not staying protected. Therefore, a system is required to detect such instances and inform the people to follow policies and protect their health and others by wearing a mask where its required.

1.2 Project Aim

Once research had been conducted on the importance of wearing face masks and following health and safety measures to stop the spread of COVID-19, there was a clear underlying need for a system which helped enforce these policies to mitigate the impact of the virus. Therefore, the aim of the project is to create a face mask detector that is capable of detecting, accurately and in real-time, whether people are wearing face masks or not. The project aims to employ the use of a CNN that will be trained using image datasets to detect two object classes: "Face Mask" and "No Mask". Furthermore, the model will also be configured to make detections in real-time via a webcam, with integration with embedded systems such as surveillance cameras also being considered as a possible future use-case for the model.

1.2.1 Project Scope

It is not within the scope of the project to integrate the face mask detector with embedded systems, but to build an accurate face mask detector that would be compatible of integration with these types of systems in the future. The face mask detector must only detect masks that are being worn by people, and not the face mask individually. If the face mask is detected individually, then this would cause some confusion. For example, a person may have removed the mask from their face but the mask is still visible to the camera. As a result of this, a contradicting detection would be made, as "No Mask" would be detected since the person is no longer wearing the mask but a mask would also be detected as it is in view. The topic of mask variation required investigation in section 1.1 for this project as the project did not aim to detect only surgical masks. But instead, to detect all

types of masks that are being worn by people. This includes any type of variations in colour, shape, material, or design patterns, etc. Likewise, when no mask is present, the model must be capable of detecting people of all colour, background, gender, size, and overall general variations in appearance.

1.3 Project Objectives

1.3.1 Model Performance

1.3.2.1 *The final model must be able to detect "Face Mask" and "No Mask" objects correctly when appropriate.* This can be measured by reviewing the test images after evaluation to check whether the model is detecting Mask/No Mask and with how much accuracy. Additionally, this can be measured using real-time input via webcam.

1.3.2.2 *The final model must be able to detect the "Face Mask" object only if a face mask is being worn by a person.* Aside from testing this by providing the model with images of people wearing face masks, this can be measured by providing the trained model images of an individual face mask, not worn by a person to check whether it will make a detection.

1.3.2.3 *The final model must be able to detect all types of face masks being worn, referring back to what was mentioned in section 1.2.1.* This can be measured by providing the trained model images of a multitude of different types of masks being worn and gauging its success by monitoring the accuracy of the predictions.

1.3.2.4 *The final model must be able to detect people who are/are not wearing face masks regardless of their colour, background, gender, size and general variations in appearance.* To ensure this can be measured, a variety of different types of people with varying appearances must be included in the training and testing datasets beforehand. This can then be measured by reviewing those images after training has completed to check whether appropriate detections have been made.

1.3.2.5 *The final model must be able to make detections from various angles*. The training and testing dataset needs to contain images of people with and without masks from various viewpoints and angles. The success can be measured by examining the detection on those specific images.

1.3.2.6 *The final model must be able to detect objects simultaneously. This includes detecting more than one class or object at the same time.* This can be measured by providing the model images with more than 1 person present in the image and examining the result.

1.3.2.7 *The final model must be able to detect objects of no matter the size (big to small).* As the models will be evaluated using the COCO mAP metrics, it will be possible to see the accuracy of detections for small, medium and large areas across 10 different IOU's. This will be indicative of the model's ability to detect different sized objects. Furthermore, to detect different sizes of objects in real-time, simply distancing yourself away from the web cam will be enough to reduce the object size as it will be further away from the camera, making it appear smaller.

1.3.2.8 *The final model should achieve the desired detection accuracy score of 70% or greater.* This will be measured during the evaluation process as the COCO mAP metrics will display a comprehensive analysis of the models performance. This can also be evaluated in

real-time via webcam input. The confidence scores of each detection in real-time can indicate the accuracy. However, it will be better to follow the results of the COCO mAP metrics as they statistically and mathematically evaluate the model, whereas in real-time the evaluation would be more of a self-interpretation of the results.

1.3.2.9 *The chosen model must be computationally lightweight and have a low computational cost. This ensures that the model could potentially be integrated with existing systems such as surveillance cameras.* Aside from conducting research to find the most suitable model, the computation requirements of a model can be indicated by how many frames per second it allows during real-time. Therefore, during real-time detections, paying attention to the quality and FPS of the video can indicate whether the model is lightweight or requires more computational power than this project is willing to accept. Although the performance can depend on the capabilities of the GPU installed as well.

1.4 Project Deliverables

There are two deliverables for this project. An implementation of the knowledge acquired by researching and analysing good-quality literature. That relates to the technical aspects required to create a suitable machine learning model to meet the aim of the project. As well as non-technical background research required to gain insight into the problem domain of the project. Lastly, a detailed final year project thesis based on the research, implementation, results and various discussions of the overall project from start to finish.

- A fully trained neural network object detection model that is capable of detecting real-time presence and non-presence of face masks among people in 3-dimensional space, with high speed and accuracy.
- Final Year Project Thesis

1.5 Project Development Methodology

The methodology followed during the development of this project was an adapted agile methodology. Contemporary software engineering is driven by a number of key themes, such as agile development cycles and the continuous delivery of production software (Jackson, et al. 2019). However, instead of continuous delivery of software, this project aims to produce continuous delivery of project iterations. It is clear that the objectives of this project will not be met with a single iteration. There will be a need for continuous improvement incrementally made with each passing iteration, continuous training and evaluation after each version. It is an inconspicuous goal of the project to make mistakes early-on in order to understand what works and what does not in an educated trial-and-error fashion. This is why optimal results are very unlikely to be achieved with a single attempt. However, they would provide a baseline for the next iteration.

Thus, there is a need to adopt and adapt a software engineering methodology that will allow for this kind of development behaviour, better yet encourage it. Agile is a project management methodology that is characterized for building software/products using short and rapid work cycles that allow for constant production and revision. It involves breaking up the project into multiple phases with continuous improvement being made at every stage. In the beginning stages of the project, specifically the research stage, other methodologies were being investigated to decide whether they are more fit for purpose (See 1.5.1 for discussion). Ultimately, due to the flexibility and

frequent testing that would become routinely available by following the agile approach, it was the clear choice.

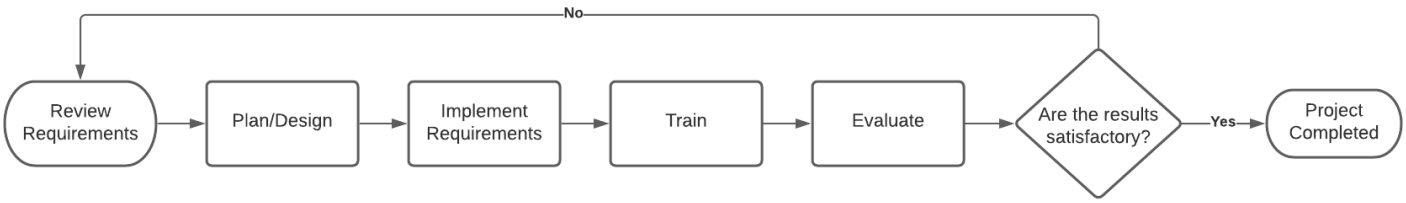

Figure 1: Agile methodology adapted for project development

1.5.1 Other Methodologies

Many software engineering methodologies exist. However, some methodologies are not compatible with certain projects or they are not as optimal as others. With projects in relation to artificial intelligence and machine learning, the two most common methodologies would be the waterfall model and the agile model. Balaji and Sundararajan (2012) compares the strengths and weaknesses of these two development methodologies. The waterfall model was a consideration for this project in the early stages. However, it is a sequential model where the requirements and planning need to be fixed from the beginning. Whereas its counterpart, the Agile method, is an iterative approach with a well-defined scope and requirements with the ability to still be subject to change as the project progresses. Testing is usually done at the end of the project when using the waterfall model while with the agile model, tests are performed throughout the development of the project. The waterfall model does not align with the nature of the development projected for this project. Therefore, it was discarded as a potential development approach to building the face mask detection model.

2

Literature Review

2.1 Introduction

Machine learning and object detection are challenges in the computer vision field that are constantly evolving as well as providing value to different aspects of the world. A modern-day example of a machine learning and object detection use-case would be the Tesla autopilot. Tesla autopilot utilises object detection to enable autonomous driving cars. The front bumpers of Tesla Model S and Model X have radars behind them with a range of several hundred meters that can detect cars and moving objects from a substantial distance (Ingle and Phute, 2016). In the real world, objects never occur in isolation; they co-vary with other objects and particular environments as mentioned by Oliva and Torralba (2007) in their discussion about object detection in context. The context and background cluster has proven to be an object detection challenge over many years and has made it even more difficult to detect objects in certain environments. However, in the past there have been multiple attempts at creating object detection algorithms, with improvements to those algorithms being made incrementally with updated versions being released over the years. The same theories and concepts used to achieve object detection and mitigate its challenges can be scaled accordingly and applied to new problem domains outside its original use-case. Therefore, applying existing knowledge of machine learning and algorithms based on previous endeavours of others can be seen as a viable direction to take when creating a face mask detection model. Ergo, research must be conducted into relevant areas of machine learning and object detection to gain insight into the best routes available to achieve the end goal of creating a fast and accurate face mask detector.

2.2 Machine Learning

Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment (El Naqa and Murphy, 2015). Machine learning is a subset of Artificial Intelligence. The goal of machine learning is to create and/or to leverage algorithms to automate the process of learning independently from contextualised data input for machines. The output of a machine learning model is mainly dependent on the data input, machine learning empowers the production of models used for prediction, pattern recognition, decision-making and emulating other cognitive tasks. Furthermore, computational sustainability is an ever-growing area of technology, Gomes (2009) discusses making the connection between machine learning and the challenges presented in the real-world in relation to the economy, society and environment. Wagstaff (2012) also supports the call for more meaningful machine learning work to be done by discussing evaluation metrics for research topic suggestions. They mention, in addition to traditional measures of performance, we can measure

dollars saved, lives preserved, time conserved, effort reduced, quality of living increased, and so on. Focusing our metrics on impact will help motivate upstream restructuring of research efforts. Relating this information back to the problem domain of the project, the research topic evaluation criteria suggested by Wagstaff (2012) inspired the same approach for the project. Given the current circumstances, there was a need for an automated impactful system to detect face mask presence. There are a myriad of approaches and methods that can be taken to solve a problem using machine learning, with new methodologies and technologies constantly being developed, innovated and optimised. However, such approaches have some disadvantages such as needing large sums of quality data, much computational power and engineering effort (Holzinger, 2018). Amid a growing focus on "Big Data," it offers epidemiologists new tools to tackle problems for which classical methods are not well-suited (Bi et al., 2019). Big Data defines a large volume that is exponentially growing and contains too much information for a human data analyst. However, machine learning has aided in managing big data better over the years. Machine learning is used to train machines by providing them with datasets and constructing algorithms that allow for automated decision making and problem-solving. With that being said, machine learning algorithms are organised into taxonomy, based on the desired outcome of the algorithm (Zhang, 2010).

2.2.1 Supervised Learning

Supervised learning is a machine learning algorithm typically utilised when training object detection models. It begins with prior knowledge of the results desired in the form of datasets that are labelled. Supervised learning entails learning a mapping between a set of input variables x and an output variable y and applying this mapping to predict the outputs for unseen data (Cunningham, et al., 2008). Scientific studies have proven that supervised learning also exists in the biological nervous system (Glaser, et al. 2019). Supervised learning involves providing a labelled dataset as input to a model, such as a CNN model, and trains the model to produce reasonable output predictions when presented with unseen data from the same category(-ies). Relating this back to a real-world scenario, this type of supervision invokes the idea of a teacher who guides the learning process. Typically, this guidance comes in the form of labelled training examples that can be used to build a classification model (Wilson et al., 2008). Supervised learning has a key role in the training of CNN models. The process involves labelled training data that consists of a set of examples to be used by the neural network during training to generalize the relationship between data and labels to unknown examples (Almási et al., 2016). As mentioned by Oquab et al., (2015), successful visual object recognition methods typically rely on training datasets containing lots of richly annotated images. This suggests utilising a supervised learning approach for object detection can achieve success but the degree of that success will be dependent on the dataset used. The dataset must contain a good quantity of quality labelled images otherwise the results may be poor, or not as accurate as they potentially could have been. Durand el al., (2016) proposes a method which is dedicated to automatically selecting relevant image regions from weak annotations with a global label and to perform end-to-end learning of a deep CNN from the selected regions. It was a successful attempt to deal with weak supervision, concluding that the method had outperformed state-of-the-art results on six different datasets. Although this method provides an automated solution to weakly labelled data, the necessity of this method is questionable as it could be argued that if a dataset is poorly labelled, then the labelling can be revised or more data can be added. However, this presents a drawback of supervised learning regarding data scarcity. Having extra data to provide for a supervised learning task is not always guaranteed. The costly work of collecting more labelled data and the tedious work of doing cleverer engineering can go a long way in solving particular problems, but this is ultimately unsatisfying as a machine learning solution (Srivastava, et al., 2015). There are many well-known datasets that are used by researchers to examine various metrics of their work. Some of the most popular datasets include Pascal VOC 2012, Microsoft COCO and ImageNet.

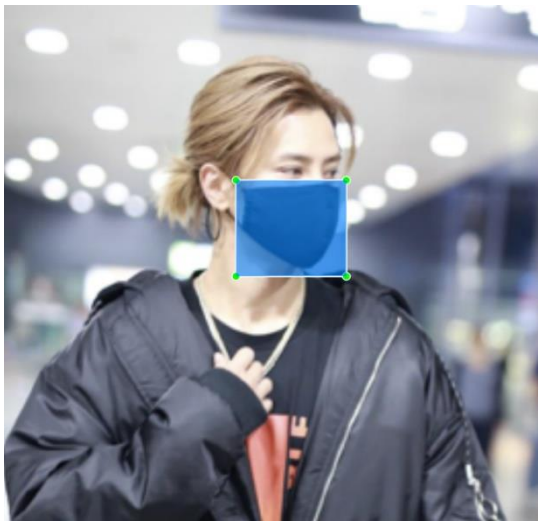

Figure 2: Labelled data / Image of person wearing a mask[1]

Figure 3: Labelled data / Image of person wearing a mask[2]

Figure 2 and *Figure 3 are* demonstrations of what a typical labelled image, ideal for a supervised learning algorithm would look like. These images were labelled using a package called "Labelimg".

2.2.1.1 Pascal VOC 2012 Dataset

The Pascal VOC 2012 dataset presented by Everingham and Winn (2011) has 4,340 images containing 10,363 labelled objects, representing 20 object classes. It is the smallest dataset discussed here but there have been multiple uses of this dataset and it has served as a reliable benchmark over the years. For example, Pascal VOC 2012 was used as the benchmark in an experiment comparing the performance of five different CNN models. The results of this experiment showed that the best CNN model had produced a validation accuracy of 85.6% and a testing accuracy of 85.24% (Shetty, 2016). This demonstrates that with the use of a well-designed dataset, very accurate results can be achieved. However, for this project none of the datasets listed can be used to solve face mask detection as that object does not exist within the dataset. Although, models pretrained on one of these datasets, already provide a better starting point than if the model were to be trained from scratch. The dataset created for face mask detection requires labelling. Pascal VOC has its own data labelling format. When labelling an image using the Pascal VOC format, an XML file corresponding to the image is created and it stores the coordinates of the labels for that specific image. This allows for labelled training/testing/validation datasets to be created for object detection problems.

2.2.1.2 Microsoft COCO: Common Objects in Context Dataset

The COCO dataset is a large-scale dataset that aims to address the three main research problems in scene understanding: detecting non-iconic views (or non-canonical perspectives of objects, contextual reasoning between objects and the precise 2D localization of objects (Lin et al., 2014). This approach seems to attempt to tackle some of object detections' most prominent challenges, those being occlusion and viewpoint changes. In a paper by Hsiao and Hebert (2014), they present a unified occlusion model for object instance detection under arbitrary viewpoints in contrast to COCO, which attempts to learn the structure of occlusions from data. That being said, if a dataset can account for occlusion and viewpoint changes, this could potentially render the approach

1 Image Source

2 Image Source

discussed by Hsiao and Hebert (2014), redundant or less impactful. Furthermore, COCO mAP evaluation metrics are a variation of the Pascal VOC mAP, that can be used for extensive object detection model evaluation.

2.2.1.3 ImageNet Dataset

ImageNet contains an unprecedented amount of image data, it contains over 14 million images, representing 200 object classes. ILSVRC has become a well-known benchmark for large-scale object recognition. ILSVRC follows in the footsteps of the PASCAL VOC challenge, which set the precedent for standardized evaluation of recognition algorithms in the form of yearly competitions (Russakovsky et al., 2015). In a paper by He et al., (2015), a method which made modifications to the ReLu activation function was being tested using ImageNet 2012 classification dataset. It had achieved a top-5 test error, beating out the previous ILSVRC 2014 challenge winner GoogLeNet by 26%. Furthermore, the result was also the first to surpass the reported human-level performance (5.1%) on this dataset. CNNs crave data input, the more they have, the more likely a better output may be produced. Assuming the data is relevant to the problem. Barbedo (2018) supports this by saying that generally the application of CNNs requires large datasets containing a wide variety of conditions to work properly. ImageNet is one of the largest datasets available for object detection to date. Naturally, a larger dataset that provides valuable input will most likely provide valuable output in regard to most CNN models. ImageNet was considered for use for this project as it can be paired with machine learning techniques such as transfer learning, discussed later in 3.4.1.

2.2.2 Other Learning Algorithms

There are a handful of other categories of learning algorithms. An example of a notable one would be unsupervised learning. It is an algorithm that works in contrast to supervised learning. Unlike supervised learning where the dataset is categorised by labels, unsupervised learning utilises machine learning algorithms to cluster and analyse unlabelled datasets. These unsupervised learning algorithms can find hidden patterns or groups of data without human assistance or intervention. Supervised learning begins with prior knowledge of the results desired in the form of datasets that are labelled. Meanwhile, unsupervised learning begins work directly with unlabelled data. In unsupervised learning, “clustering” refers to the process of grouping together similar entities. Unsupervised learning does not have a Y variable to its method; therefore, the algorithm explores the unlabelled data in search of patterns. Commonly, these newly-discovered patterns lead to a number of various groups forming based on the type of data similarities. Unsupervised algorithms attempt to solve any particular task in a harder way than is necessary or, even the wrong problem altogether is an often-made criticism (Khanum, et al., 2015). For example, if this project focused an unsupervised learning approach, unsupervised learning would provide no feedback based on the prediction results. It would be a more reasonable approach to first learn the features, then classify them using unsupervised learning. However, if the classification of features are already known then supervised learning would be used in the next stage. Analysing a potential unsupervised learning approach only supports the point made by Khanum et al. (2015) that sometimes-unsupervised learning makes completion of a task more difficult than it should be. Especially, in object detection, relevant research points to the exclusive use of supervised learning above other learning algorithms for optimal results (Cheng et al., 2013).

2.3 Deep Learning

Deep Learning (DL) is a subset of ML and is based on Deep Neural Networks (DNN). DL has been widely studied over the past decade and due to its rapid evolvement, this has made way to some major advancements in object recognition and other CV problems over the years. The founding concepts of deep learning originate from ANN research (Hong, 2011). Traditional architectures for solving computer vision problems and the degree of success they enjoyed have been heavily reliant on hand-crafted features. However, of late, deep learning techniques have offered a compelling alternative – that of automatically learning problem-specific features (Perez and Wang, 2017). Some of these computer vision problems include object detection and recognition, object tracking, and 3D reconstruction, to high-level CV tasks, such as facial expression recognition and human activity recognition (Bhardwaj et al., 2019).

Deep learning models require a significant amount of data to be used for training. Generally, deep learning is performed using supervised learning and has been proven to perform better than traditional ML methods in a range of different tasks (LeCun, et al. 2015). One of these tasks include object detection, therefore deep learning could prove valuable to the project as the project mainly deals with this computer vision problem. Since deep learning requires the use of a vast amount of data and is typically applied using supervised learning, this can become a problem. As discussed in 3.2.1, having extra data to provide for a supervised learning task is not always guaranteed. It can be hard to come by this data, especially if the specific required data category is not very common. However, as mentioned by (Bi et al., 2019) in 3.2, big data is on the rise and can provide substantial amounts of data that can be utilised for training models. As much as 90% of current data were created in the last couple of years – a trend that will continue for the foreseeable future Al-Jarrah et al., (2015).

It was discovered in the mid 2000's that utilizing a graphical processing unit (GPU) is a cheap and viable alternative in lieu of CPU's and specialist hardware for implementation of machine learning algorithms. Especially since modern-day computers very commonly contain GPUs (Steinkraus et al., 2005). Furthermore, due to advancements made by Google, it is now possible to utilise an NVIDIA Tesla K80 GPU by connecting to their runtime environment in Google Colaboratory. A study investigated the performance of the GPU runtime environment within Google Colab. The study concluded that the performance reached using Google Colab is equivalent to the performance of dedicated testbeds, given similar resources. Thus, this service can be effectively exploited to accelerate not only deep learning but also other classes of GPU-centric applications.

DL methods are generally applied in practice using Deep Neural Networks (DNNs). DNNs refer to any neural network that contains several hidden layers. CNNs are a certain type of DNN that are especially efficient for object recognition. CNNs use convolution and pooling layers as discussed in chapter 3. Therefore, CNNs have the ability to self-teach, meaning they can automatically detect and learn the most important and key features of an image without any human supervision. Image representation for classification tasks often used feature extraction methods which have been proven to be effective for different visual recognition tasks (Jmour et al., 2018). Therefore, it is evident that application of CNNs is necessary in order to achieve optimal results for object detection as demonstrated by the research discussed in this chapter thus far.

2.4 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are at the top of the ANN chain when it comes to object recognition. First introduced in 1980 (Fukushima, 1980), thanks to DL and ANN research advancements, CNNs were able to thrive in performance due to these advancements. The main documented uses of CNNs are related to image classification, processing and segmentation. Although, CNNs have been used in NLP applications for purposes such as text classification and speech recognition as well (Yin et al., 2017). Therefore, it is necessary to investigate the capabilities of this type of neural network to understand whether they are a good option for this project. Li et al. (2015) examines the use of CNNs in face detection, which is a closely-related task to that of the project. They mention that CNNs are relatively computationally expensive. Specifically mentioning that exhaustively scanning the full image in multiple scales with a deep CNN is not a practical solution. However, since this paper was published, there have been many frameworks and advancements introduced for CNNs to help combat the object detection problems. FaceNet was introduced that same year, which utilises a deep convolutional neural network in its approach. It achieved 99.63% accuracy on the Labelled Faces in the Wild (LFW) dataset and a further 95.12% accuracy on the YouTube Faces dataset (Schroff et al., 2015). Additionally, Triantafyllidou and Tefas (2016) proposed a model for fast face detection using a deep convolutional neural network. The model had only utilised 76,544 free parameters, meaning this model is lightweight as it does not utilise too many resources. This approach had achieved a recall rate of 90% on the FDDB dataset (Face Detection Data Set and Benchmark).

Many of the approaches used to create accurate face detection models have also been based on the R-CNN framework, which achieved a mAP of 53.3% on the Pascal VOC 2012 dataset (Girshick et al., 2013). Furthermore, an improved version of R-CNN, known as Faster R-CNN, published in 2015 Girshick, R. 2015) had become the foundational framework that was used in the COCO detection challenge in 2015 by the winning entry3 as stated in a paper by Jiang and Learned-Miller (2017). The paper also discusses the use of the Faster R-CNN algorithm for face detection, stating that the effectiveness of Faster R-CNN comes from the region proposal network module (RPN). Concluding that it is possible to use multiple convolutional layers within an RPN without adding extra computational burden. Relating this back to the statement by Li et al. (2015) regarding CNNs being too computationally expensive for face detection, it seems their observations may only be partially correct or outdated. DCNNs can be expensive computationally on their own, however as discussed in the section, a plethora of advancements and algorithms encapsulating DCNNs have been made and some optimised especially for the task of face detection. After examining the results of these frameworks and advancements in the context of face detection, it is clear to see that these approaches have seen success. Which demonstrates their capability of undertaking face detection tasks, meaning CNNs could prove to be an ideal solution of the very similar task of face mask detection.

2.4.1 Transfer Learning

Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned (Torrey and Shavlik, 2010). This method involves utilising a pretrained CNN model and exploiting the knowledge already present within that model to improve the generalization and training of another task. For example, if the pretrained CNN has been trained to detect cars, it would be possible to utilise transfer learning to train the model to detect trucks. An informative paper by Kornblith (2019) discusses whether models trained using the ImageNet dataset are better for transfer learning. A CNN model pretrained on the ImageNet dataset was the main focus of the experiments, the experiment included retraining this pretrained model to

3 https://cocodataset.org/#detection-2015

detect human faces, cars and animals using transfer learning to see how it would perform. This experiment concluded with the average score of the human faces being the highest out of the three categories with 93% accuracy on the Caltech Faces dataset. Their execution of this method involved modifying the final layer of the pretrained model and retraining it for classification, this keeps all the weights and knowledge from initial training of the model (on the ImageNet dataset) and transfers that information onto the new dataset. Transfer learning is especially useful when the dataset to be trained does not contain many samples. Another instance where transfer learning was used, was in a paper by Deepak and Ameer (2019) which examined the effectiveness of transfer learning in relation to a brain tumour classification problem. A CNN model that was pretrained on the ImageNet dataset known as GoogLeNet was used for training. The dataset used consisted of 3064 images for three classes: glioma (1426 images), pituitary (930 images) and meningioma (708 images). Immediately looking at this data, it is clear that there is a data-imbalance. Mikołajczyk and Grochowski, (2018) mention that the lack of sufficient data or uneven class balance can be dealt with using data augmentation. Despite the imbalance, the model had achieved 97.8±0.2% when classified with SVM while using deep CNN features. However, an experiment was conducted that involves using less input training data. 25% of the data was used and the overall accuracy resulted being 93.3±0.6%. This shows that as transfer learning does not require a vast amount of images to be effective, therefore can prove impactful to solving the problem of labelled data scarcity as discussed previously in this chapter, while still providing optimal results. Not only does transfer learning help solve the problem of limited data, but it also provides high accuracy detections that usually reside in 90%+ accuracy as seen in the papers discussed in this section. This would be especially useful to the project as one of the main objectives is to obtain an accuracy of 70%+ (1.3.2.8).

2.4.2 Image Data Augmentation

Another technique in machine learning that can be used to solve the problem of a limited dataset is known as Image data augmentation. It is a technique which can be used to expand the size of the dataset artificially by creating modified versions of existing images from the dataset. By creating more images using image data augmentation, the amount of images in a dataset will increase. Thus, resulting in more training data. Traditional data augmentation include translation, rotation, flips, mirror, zoom, colour perturbation and shear. In a paper by Han et al. (2018), transfer learning had been used with data augmentation. It is mentioned that data augmentation had strengthened the generalization ability of the fine-tuned CNN network by adding much diversity to the training dataset and alleviated over-fitting. While transfer learning removed the requirement for a vast amount of training data. Data augmentation is another technique frequently used when training deep networks that has been empirically shown to reduce overfitting (Rice et al. 2020). It is a technique that does not require a massive amount of training data to do so either (Cogswell et al., 2016). However, by transforming the images by either flipping or rotating them, this also creates new viewpoints. For example, in an article by Dawar et al. (2019), a CNN and LSTM model are used to create a deep learning-based decision fusion approach for action or gesture recognition via simultaneous utilization of a depth camera and a wearable inertial sensor. Data augmentation was included in this project as training data was limited. Concluding that overall, data augmentation had a positive impact on the recognition accuracies. Additionally, the augmentation of depth images was achieved by mimicking different viewpoints of the depth camera. This supports what had been said previously, new viewpoints are in fact being created by transforming the data and creating a positive impact on the recognition accuracies as mentioned by Dawar et al. (2019). This would be an asset to the performance of the face mask detector as creating new viewpoints will train the model to recognise the face mask and no mask objects under various visual circumstances, as well as simultaneously contributing more data to the overall training set.

Figure 4: Original Image[4]

Figure 5: Augmented Image

Figure 4 is an example of an image that has not been augmented in any way whereas *Figure 4* is that same image but augmented. *Figure 5* had been rotated which caused a change of angle and viewpoint. Also, in relation to face mask and no mask object classes, the rotation has caused the image to become zoomed into the centre. Assuming that the face mask detector were to be used here, it could consider the face in *Figure 5* to be a small object whereas *Figure 5 ,* it would be of medium size due to the size change caused by image becoming zoomed in. COCO mAP metrics measure the precision and recall of large, medium and small areas across various IOUs. Image augmentation can be especially useful here as the original training set may be lacking in a certain area. There may be an imbalance between large and small area objects, image augmentation could potentially aid in creating a balance by transforming images. Not to mention the new viewpoint created from the rotation that can also provide diversity to the dataset as previously mentioned by Han et al. (2018).

4 Source for image (Discovered via Google Images)

3

Theoretical Background

3.1 Artificial Neural Networks (ANNs)

An artificial neural network (ANN), also commonly referred to as a neural network, is a biologically-inspired method of creating computer algorithms that are capable of learning independently while finding and recognising connections in data. The human brain has billions of cells known as neurons. Similarly, an ANN contains hundreds or even thousands of artificial neurons, commonly referred to as processing units. Processing units make up the ANN, they are all interconnected by nodes within the network. ANNs are a collection of algorithms that undertake the task of simulating the functioning of the human brain by attempting to recognise underlying relationships between sets of data. Neural networks are typically paired with supervised learning. The purpose is to learn to map between the input (x) and the output (y). A neural network model is in many cases, a function approximator $y = f(x)$. A neural network may be created by stacking multiple layers and functions together. For example, $y = f^1(f^2(f^3(x)))$, where f is representing a layer in the network.

3.1.1 Layers

Layers are the building blocks of an artificial neural network. Layers are containers that receive weighted input and then transform that input values using a set of functions. These values then get passed through as an output to another layer. A layer only contains one type of activation function, this is so that it can be easily compared to other parts of the neural network. ANN's consist of an input layer of neurons, multiple hidden layers of neurons and an output layer of neurons. The first layer in a neural network is known as the input layer and the last layer is called the output layer. All of the layers in between the input and output layers are known as hidden layers. See *Figure 6* for a visual representation. The output of h_i , neuron i in the hidden layer h can be seen in equation 1.

$$h_i = \sigma\left(\sum_{j=1}^{N} V_{ij}\, x_j + T_i^{hid}\right) \tag{1}$$

σ is representing the activation/transfer function.
N represents the number of input neurons.
V_{ij} represents the weights
x_j represents the input neurons
T_i^{hid} represents the threshold of terms of the hidden neurons

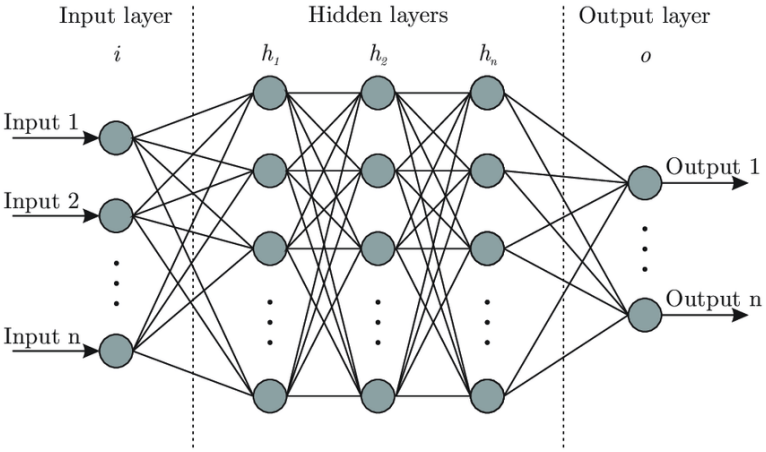

Figure 6: Diagram of common ANN architecture[5]

Figure 6 is a visual representation of the architecture belonging to a typical Artificial Neural Network (ANN).

There are many different variations of Neural Networks, all of them utilising this basic architecture and principles as their foundation.

3.1.2 Neurons

Layers in an ANN have a multitude of artificial neurons – also referred to as perceptron's (usually at least in the hundreds or thousands). These neurons are parameterizable and they are all interconnected with each other. An artificial neuron is a mathematical function that aims to mimic the functioning of a biological neuron. Commonly, a neuron will compute the weighted average of the input it receives, this sum will then be passed through a non-linear function. This function is known as the activation function. Neurons in deep learning are nodes that computations and data will flow through. A neuron will receive input signals. The signals come from the previous neurons in the previous layer in the neural network, or they will come from the raw data set that has been inputted. Overall, a neuron is simply a set of inputs, weights and an activation function. Using these properties and inputs, it will translate them to produce an output. Once an output is produced, it can then be picked up by the next layer.

$$Y = f(w_1.x_1 + w_2.x_2 + b) \quad (2)$$

Equation 2 represents a single neuron, assuming it has two inputs. Inputs x^1 and x^2 have associated weight with them, represented by w^1 and w^2 and Y is representing the output of the neuron. There is also another input with the weight being the bias which has a value of 1, represented by b in the equation. The bias acts like the intercept in a linear equation. By which the constant value would cause the line to be transposed. The bias vector is an additional parameter in the NN that is used to make adjustments to the output as well as the weighted sum of the input(s) to the neuron. It allows the activation function to be shifted by including a constant (bias) to the input. It can cause delay or acceleration to the trigger of the activation function; the bias is often tuned to train models to fit the data better. Overall, the primary function of the bias is to supply a trainable constant value to every node, in addition to other inputs the node may receive.

5 Image source

3.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks, otherwise referred to as CNNs, originally got its name from the mathematical linear operation between matrices known as “convolution”. Convolution is referring to the mathematical combination of two different functions in order to produce a third. The mathematical equation for the convolution formula can be expressed as such:

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \qquad (3)$$

CNNs are typically used for analysing and processing visual imagery. They are trained using images and require a multitude of samples before it becomes capable of generalising input and creating accurate predictions on data it has not seen before. Despite CNNs and neural networks in general being modelled after the function of the human brain, they do not see an image the same way a human brain would.

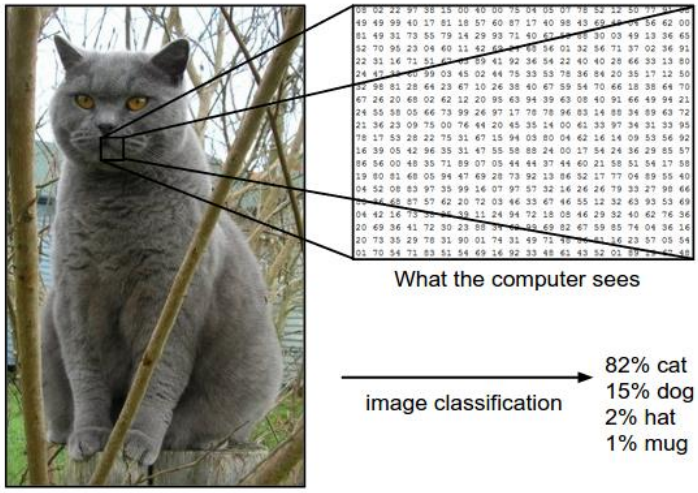

Figure 7: Example of how humans perceive images vs NNs[6]

The image of the cat in *Figure 7* is the human brain would see the image.

A CNN would see an image as nothing more than matrix of numbers.

However, CNNs can still be trained to recognise patterns in an image, despite CNNs seeing the image differently.

The architecture of a CNN model is different to that of a standard neural network. The layers within a CNN model are organised into three dimensions: width, height and depth. Also, the neurons in one layer will not connect to all of the neurons in the next layer, only to a small portion of that next layer. CNNs also have more layers than a traditional neural network. Specifically, more hidden layers. The final layer of a CNN is fully connected, but in an ANN, every neuron is connected to each of the other neurons present in the network. The output of the final layer will also become reduced to a single vector of probability scores in a convolutional neural network. CNN models have two main components: Feature extraction/hidden layers and Classification.

6 Image source

3.2.1 Input Layer

The input layer is the first layer in the convolutional neural network to deal with and process any input images. The input can be a 3D or 2D Image that is in grayscale or a 3-channel image (RGB). CNNs handle input best when the input is of a certain dimension. This is due to the behaviour of the convolution in the convolutional layer. Consequentially, the outputs of an image may become invalid, this is dependent on the stride of the filter and the pooling layers that follow. The output image may have an invalid size, which may even include half-pixels. The input layer, in terms of image processing, will present the pixel matrix of the image. Which would then be passed down to the convolutional layer for further image processing.

3.2.2 Convolutional Layer

The convolutional layer is the first layer in the CNN. It is the core building block of the CNN and performs a lot of the computationally heavy tasks. There may be more than one convolutional layer in a complex CNN. This layer applies the convolution operation to the input image and will then pass the result onto the next layer in the network. The main task of the convolutional operation is to extract the high-level features. However, the first convolutional layer will capture the low-level features of the input such as gradient orientation, colour, edges, etc. If more layers are added then the architecture will be able to adapt to the high-level features later on too. By the end, the network will have a well-rounded understanding of the images (input), in a similar fashion to how humans would, and will then be able to make predictions on data it has not seen before.

Image input into a convolutional layer is convolved using filters (filters and kernels are terms that are often used interchangeably). The filters are a small matrix of numbers that use the backpropagation algorithm to store and convert all the pixels of the input in its receptive field into a single value. The filter will pass by and scan the pixel of the image and will transform it based on values from the filter. A receptive field is a region of space in the input image that the filter is looking at. The filters are defined by its hyperparameters (height x width). Additionally, there are other hyperparameters that belong to the operation of convolution such as: stride and padding

The filter contains certain height x width parameters and is positioned at the top left corner of the input image. The filter slides over the input performing element-wise multiplication on the input pixels using the filter, the results are then summated and stored as a single value on the feature map as shown in *Figure 8* below. The filter is moved one pixel at a time and this is repeated until all of the possible locations are filtered in the image. In *Figure 8*, the horizontal border contains empty values, this is known as (zero) padding. It is the process of adding zeros to the border to widen the pixel so that more of the edge pixels can be covered. Convolution will still work the same, but there will be some zero values. By adding padding, the image will become larger than the original due to the zeros being added, however the image will still shrink down to its original size eventually as the zeros themselves will not impact the results from the filter scan. However, the image is better recorded with padding than without it, leading to better accuracy. *Note:* The example filter in *Figure 8* is set to detect vertical edges only. Stride is the hyperparameter in a CNN that controls the movement of the filter over the image or input data. For example, if the stride in a CNN has been set to 1, then the filter will move 1 pixel or unit at a time.

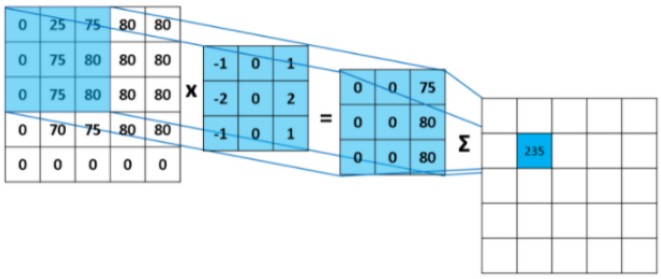

Figure 8: Illustration of how filter slides over input image to create an output[7]

The feature map values are calculated using the following equation where the filter is denoted by h, the input image by f, and the indexes of the columns and rows of the resulted matrix are represented using n and m respectively.

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \tag{4}$$

3.2.2.1 Non-linearity (Activation)

The non-linearity function is part of the convolutional layer and is not considered its own layer. Just like a regular neural network, non-linearity is performed on the output of the neurons from the layer. The data is first transformed into non-linearity before proceeding to the next layer. This is done by different activation functions. Activation functions decide if a neuron should be activated by calculating the weighted sum and then adding the bias with it. There are many different activation functions. Some common activation functions include: Sigmoid, ReLu and Softmax.

Sigmoid function: $$f(x) = \frac{1}{1+e^{-1}}, \tag{5}$$

ReLU function: $$f(x) = \max(0, x), \tag{6}$$

Softmax function: $$\sigma(\underset{z}{\rightarrow})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{7}$$

For softmax, the number of nodes need to be the same as the number of nodes in the output layer. It is typically only used for multi-class classification whereas sigmoid is used for binary classification tasks, as discussed later on in this chapter. ReLU (Rectified Linear Unit), ReLU is a very common non-linear activation function that is used at this stage. ReLU's derivative is very fast to compute and in turn the activation function is faster to compute than other functions such as sigmoid. The main advantages of this is it decreases inference and training time needed for neural networks.

3.2.3 Pooling Layer

The main function of the pooling layer is to reduce the dimension of the input image. This is done by sampling the convolved feature maps produced by the filters in the convolutional layer as discussed in 2.2.2. In the pooling layer, the unnecessary redundant information is discarded while any useful information from the image is kept. In turn, speeding up the computation as well as preventing the issue of overfitting arising. Furthermore, this layer contains feature invariance meaning that the network will become interested in the presence of specific features instead of the certain location of

7 Image source

those features. Thus, it will have a small tolerance for feature displacement. The following equation represents how the pooling layer functions:

$$X_j^i = f\left[\beta_j^i samp\left(X_i^{l-1}\right) + b_j^i\right], \quad (8)$$

Where $f[\,]$ is representing the non-linear activation function, $samp$ is representing the subsampling function and β is the subsampling coefficient. The feature map from layer l is represented X_j^i and X_i^{l-1} is representing the i-th feature map in the layer $l-1$ and finally b_j^i is representing the bias vector.

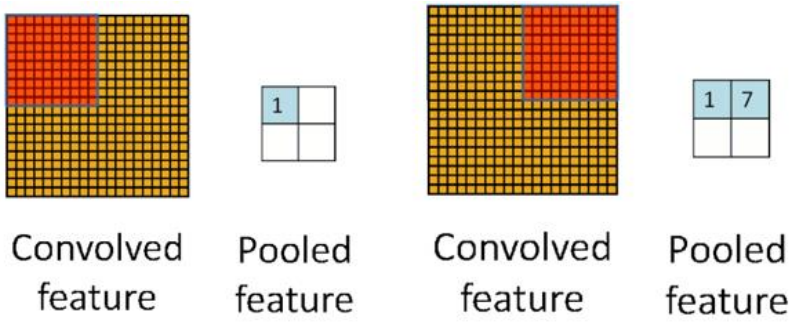

Figure 9: Functionality of pooling layer illustrated[8]

The pooling layer is another core building block of a convolutional neural network. Its main function is to reduce the spatial size of the representation, progressively, in order to reduce the amount of computation and parameters for the network. The filter, unlike the filter in the convolutional layer, will not overlap. This non-overlapping filter is demonstrated in *Figure 9*, the contrast can be seen when comparing the behaviour of the filter in *Figure 8 and Figure 9.* The pooling filter in *Figure 9* does not overlap over previous attributes covered by the filter in *Figure 8.*

3.2.4 Fully Connected & Flattening Layer

The fully (dense) connected layer is among the last layers in the CNN. This layer combines all of the information transferred from the previous layers. The information passed down to the fully connected layer will come from the previous layers in the network, including the flattening layer which comes before the fully connected layer and after the pooling layer. The information provided to the flattening layer will be in a 3-dimensional matrix. Flattening simply refers to converting the data from a 3-dimensional matrix to a 1-dimensional array that will be used as input for the next layer (fully connected layer) for classification. Flattening will create a single feature vector; all of the data will be in a single column which will be connected to the fully connected layer. Flattening is required as the input to the fully connected layer cannot be direct inputs such as cubic or rectangular (2D or 3D) shapes. An activation function such as Sigmoid or softmax are used here to classify the output based on category, number of classes, probability, etc. The sigmoid activation function is typically used for binary classification whereas softmax is used for multi- class classification (more than two classes in the classification problem). The activation functions are implemented in the layer just before the output.

8 Image source

3.2.5 Output Layer

The output layer is the fully connected layer that takes the inputs and information that has been passed through every other layer in the network and transforms this information into a complete model. It is the final layer in the CNN. Lastly, a loss function is used to compute the mean square loss. The loss function conveys information during training in regard to how well the algorithm is modelling the dataset (input). If the predictions are poor, then the loss function will output a greater number. However, if the loss function outputs a lower number then this means that the model is performing well. There are two main loss functions that are used in practice, binary cross-entropy and categorical cross-entropy. This is further discussed in 2.2.6.

3.2.6 Loss Function

There are a multitude of loss functions, but the most common used in image classification practice are known as binary cross-entropy and category cross-entropy, otherwise referred to as multi-class cross-entropy. The "entropy" term generally refers to uncertainty or disorder. Loss functions are used to indicate how well the model is modelling after the dataset provided during training. As mentioned previous, the higher the loss is, the worse the model is performing in learning the data using the data provided. Vice versa, if the loss is low, then this indicates that the model is training well.

3.2.6.1 Cross-Entropy Loss Function

Cross-entropy is a loss function that measures the difference between two probability distributions for a set of events or a given random variable. The cross-entropy equation takes in two distributions. $p(x)$ which is the true distribution (the ground truth) and $q(x)$ which is the estimated distribution, which is defined over the variable x in the equation:

$$H(p,q) = -\sum_{\forall x} p(x)\log\left(q(x)\right) \tag{9}$$

In relation to a CNN, equation 9 may be reformulated whereby y is the ground truth and the estimate value may be represented by $\hat{y}$. For a standalone example, this equation would look like the following:

$$Loss = y \cdot \log\left(\hat{y}\right) \tag{10}$$

There are different variations of the cross-entropy formula, modified to accommodate the purpose for which it is required for. The two most notable variations are the binary cross-entropy and categorical cross-entropy.

3.2.6.2 Binary Cross-Entropy Loss Function

The binary cross-entropy loss function is otherwise known as the sigmoid cross-entropy loss. This is due to the sigmoid activation function being the only activation function that is compatible with the binary cross-entropy loss function. This activation function is almost exclusively only used for binary classification tasks, in other words, it answers a question that only has two choices. For example: yes or no, cat or dog , face mask or no mask, etc. The binary cross-entropy uses the following equation to calculate the loss:

$$Loss = \frac{1}{output\ size} \sum_{i=1}^{output\ size} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i) \tag{11}$$

Where $output\ size$ refers to the amount of scalars in the output of the model, $\hat{y}_i$ being the i-th scalar value in the output of the model and y_i being the corresponding target value. Refer to equation 5 for the sigmoid function.

3.2.6.3 Categorical Cross-Entropy Loss Function

The categorical cross-entropy loss function is often used for solving multi-class classification problems. Multi-class classification refers to a task where there are multiple classes that an example can potentially be classified into. For example, unlike binary classification where there are only two classes that an example could belong to such as cat or dog, with categorical cross-entropy the categories could include: cat, dog, lion, tiger, bird, etc. Softmax is the only activation function that is recommended for use with this loss function. The softmax function is a generalization of the sigmoid function, but for a number amount of classes. The categorical cross-entropy uses the following equation to calculate the loss:

$$Loss = -\sum_{i=1}^{output\ size} y_i \cdot log\ \hat{y}_i \tag{12}$$

Where $output\ size$ refers to the amount of scalars in the output of the model, $\hat{y}_i$ being the i-th scalar value in the output of the model and y_i being the corresponding target value. Refer to equation 7 for the softmax function.

3.3 Evaluation Metrics for Object Detection

In order to evaluate the performance of a CNN model, there are specific methods that can be evoked to assess the various metrics of the model. This section will review methods which are typically used to assess state-of-the-art CNN object detection models. However, first it is important to become familiar with the following terms:

Confidence Score: This refers to how likely the bounding box, predicted by the model, contains an object as well as how accurate the position of the bounding box is over said object. If no object exists, then the confidence score must be 0.

Bounding Box: A bounding box is used to mark out the target location of an object. A bounding box can be determined by the x and y axis coordinates of the lower-right corner of the box and the top-right corner of the box. The x and y axis also represents the height, width and centre of the box.

Ground Truth: This term in relation to object detection refers to the correct localization and measurement of an object in an image. Typically used in supervised learning, ground truths represent the desired output of the model. They are commonly referred to as ground truth labels, an image may contain one or more labels that can be used to teach the model to detect the very object that has been labelled. Furthermore, the labels serve as a point of reference for the evaluation of the model. Ground truth labels define the standard for the model and are used to evaluate a model by comparing its output against the ground truth. Ground truth labels, like bounding boxes, are defined by an x and y axis and centre.

3.3.1 Intersection Over Union (IOU)

IOU can be expressed using equation 13, where the area of the intersection is divided by the area of the union of a ground truth box (B_{gt}) and the predicted bounding box (B_p). The confidence score alongside the IOU are required in order to determine if a detection is a TP, FP, TN or FN. IOU, in relation to object detection, is used to measure the association of the ground truth boxes and the predicted bounding boxes.

$$IOU = \frac{area(B_p \cap B_{gt})}{area(B_P \cup B_{gt})} \tag{13}$$

A detection is considered a true positive **(TP)** when the model correctly categorizes an object in an image. For example, categorizing a mask as a mask.

A detection is considered a false positive **(FP)** when the model categorizes an object in an image when it should not have. For example, categorizing a mask as a hat.

A detection is considered a true negative **(TN)** when the model correctly does not categorize an object in an image. For example, not categorizing a mask as a hat.

A detection is considered a false negative **(FN)** when the model does not categorize an object in an image when it should have. For example, not categorizing a mask as a mask.

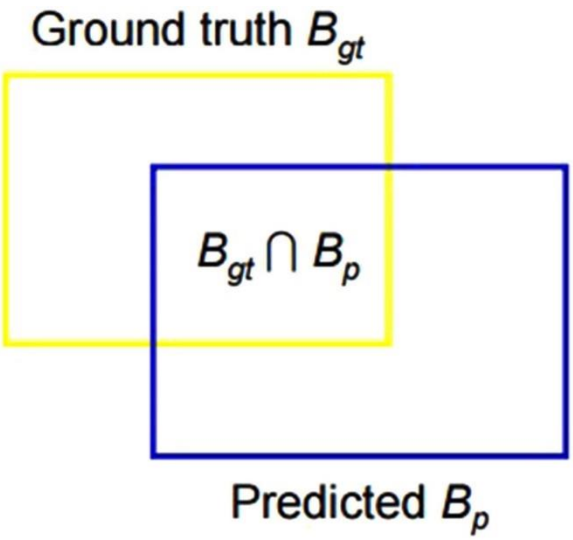

Figure 10: Visual representation of intersection over union (IOU)[9]

3.3.2 Average Precision (AP)

Precision is the amount of relevant instances from all of the retrieved instances. Precision can be defined as the amount of TP's divided by the sum of TP's and FP's. This is expressed mathematically in equation 14.

$$precision = \frac{TP}{TP + FP} \tag{14}$$

Recall is the amount of relevant instances that have been retrieved. Recall can be defined as the amount of TP's divided by the sum of TP's and FN'S. This is expressed mathematically in equation 15.

$$recall = \frac{TP}{TP + FN} \tag{15}$$

Setting a threshold for the confidence score at various IOU thresholds, means that different pairs of precision and recall can be recorded. A graph of recall (x) against precision (y) can be plotted. This graph creates a precision-recall curve and can be used to indicate the association between the two metrics. Furthermore, there is a different type of curve known as the recall-IOU curve. It is the foundation of the average recall metric.

The precision-recall curve may be used as an evaluation metric; however, for evaluation purposes, it would be more useful to have a numeric figure to describe the performance of the object detection algorithm. It would also be difficult to compare the different object detectors based on a precision-recall curve alone. Thus, average precision (AP) is introduced to deal with this task. The average precision metric is essentially the precision that has been averaged throughout all unique levels of recall.

The precision must then be interpolated at various recall levels before the AP can be calculated. See equation 16 for a mathematical representation. Where the interpolated precision (represented by $Pinterp$) at a fixed recall level (represented by r) is established as the highest precision found at any recall level (represented by $r' \geq \mathrm{r}$).

$$Pinterp = \max_{\mathrm{r}' \geq \mathrm{r}} p(r') \tag{16}$$

9 Image source

In equation 16, it can be seen that there are in fact two ways of choosing the recall levels at which the precision may be interpolated, as indicated by r. Choosing 11 equally spaced recall levels has been the traditional way of achieving this, however there is a standard called Pascal VOC challenge, which selects all of the unique recall levels available from the data. *Figure 11* visually represents how the discussed interpolated precision recall curve is obtained using the original curve and the Pascal VOC AP method.

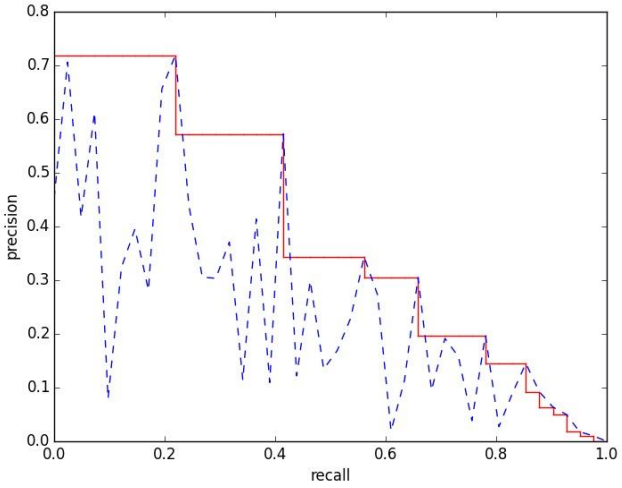

Figure 11: Interpolated curve obtained using precision recall curve[10]

Average Precision, also referred to as mean average precision (mAP), is defined as the area beneath the interpolated recall curve and can be mathematically expressed as:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) Pinterp(r_{i+1}) \tag{17}$$

$r_1, r_2 \ldots, r_n$ represents the recall levels in ascending order at which the precision will be interpolated.

3.3.3 Mean Average Precision (mAP)

AP typically only refers to the calculation of one class, however there are commonly $C > 1$ classes in object detection. Mean average precision can be defined as the mean of the average precision across all C classes. It is the average of the AP. Mean average precision can be mathematically expressed as:

$$mAP = \frac{\sum_{i=1}^{C} AP_i}{C} \tag{18}$$

10 Image source

3.3.4 Average Recall (AR)

Average recall (AR) measures the amount of correct predictions made from all of the predictions overall. Average recall is recall averaged across all IOU $\in [0.5: 1.0]$ and is calculated as two times the area beneath the recall-IOU curve. This can be mathematically defined as:

$$AR = 2\int_{0.5}^{1} recall(o)do \tag{19}$$

o representing the IOU and $recall(o)$ being the corresponding cell. This recall-IOU curve does not differentiate between different classes but there are other challenges such as COCO that do. This challenge computes its average recall on a class per class basis.

3.3.5 Mean Average Recall (mAR)

Similar to mAP, mAR is computed as the mean of AR across all C classes. This is mathematically expressed as:

$$mAR = \frac{\sum_{i=1}^{C} AR_i}{C} \tag{20}$$

3.3.6 COCO Challenge

The Pascal VOC challenge computes the mAP metric using only a single threshold of 0.5. The COCO challenge is a variation of this calculation and works in a similar way. However, instead of computing the mAP using a single threshold, the COCO challenge metrics will use multiple different thresholds.

$mAP^{IOU=.50:.05:.95}$ This is the overall mAP averaged across 10 different IOU thresholds. (Main metric)
$mAP^{IOU=.50}$ This metric is equivalent to the Pascal VOC metric
$mAP^{IOU=.75}$ This is a fixed IOU metric

Furthermore, COCO mAP is also computed across different scales, these variations in scale are also averaged over the 10 IOU thresholds.

mAP^{small} This metric is used for calculating the objects that cover an area of $< 32^2$
mAP^{medium} This metric is used for calculating the objects that cover an area of $> 32^2$ but $< 96^2$
mAP^{large} This metric is used for calculating the objects that cover an area of $> 96^2$

Additionally, mAR also has a multitude of variations in the COCO challenge. The maR metric varies across a different amount of detections per image.

$mAR^{max=1}$ This is the metric calculated for 1 detection per image
$mAR^{max=10}$ This is the metric calculated for 10 detection per image
$mAR^{max=100}$ This is the metric calculated for 100 detection per image

Just like mAP, mAR also varies over different sized object areas.

mAR^{small} This metric is used for calculating the objects that cover an area of $< 32^2$
mAR^{medium} This metric is used for calculating the objects that cover an area of $> 32^2$ but $< 96^2$
mAR^{large} This metric is used for calculating the objects that cover an area of $> 96^2$

3.3.7 K-Fold Cross-Validation

K-Fold cross-validation is a resampling procedure typically used in machine learning to evaluate the performance of a model on unseen data. This procedure contains a parameter known as k, which represents the number of groups, otherwise known as folds, that the sample data will be divided into. When the k value is chosen, such as $k = 5$, it may be referred to instead as a 5-fold cross-validation. If the data is divided into 5 folds, then the dataset will be split into 5 equal parts and the training process will be executed 5 times, each time a new fold will be used for testing and the rest will be used for the training. See *Figure 12* for a visual representation of this procedure.

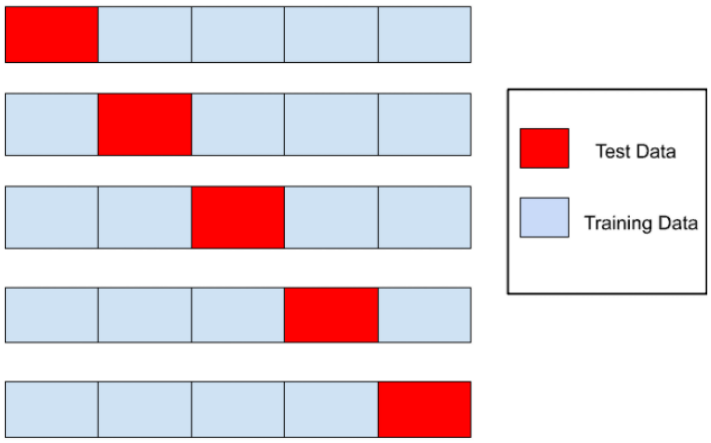

Figure 12: K-Fold Cross-Validation 5-Fold Split[11]

11 Image source

4

Model Selection

4.1 Selection Criteria

Choosing the main model to work with was a crucial part of the project which required reviewing existing models and algorithms that would be of aid to the aim of the project. Objective 1.3.2.9 under model performance in chapter 1 states the model must produce a low computational cost. Furthermore, objective 1.3.2.8 mentions the desired accuracy metric being 70%, this is the minimum score the end iteration should achieve to be considered accurate and reliable. The selection process involves a range of different object detection and deep learning algorithms being reviewed in order to gain insight into which are best suitable for the purpose of this project. The selection process was data and literature driven to help create a comprehensive evaluation of the model/algorithm. Popular object detection algorithms such as SSD, YOLO and Faster R-CNN were reviewed and compared in this chapter.

4.2 Review of Existing Algorithms

In the section, potential algorithms that were considered for this project are discussed, explaining their dismissal or approving them for use within the project. This was an important aspect of model selection as mentioned in section 4.1 since the model used will have a direct impact on the output and the results. The review aims to examine the general outline of each algorithm.

4.2.1 YOLO: You Only Look Once (2015)

The YOLO model was one of the first ever attempts at creating an object detection algorithm capable of real-time detections (Redmon et al., 2015.). It only requires a single propagation pass through the network to make a prediction, hence the name You Only Look Once. It is mentioned that the base version of the algorithm was able to process images in 45 frames per second in real-time utilising a GPU. Additionally, a smaller version of the model known as Fast YOLO had achieved 155 frames per second while achieving double the accuracy scores (mAP) of existing models at the time such as R-CNN and DPM. Fast YOLO also acts as a framework to accelerate a later version of the algorithm known as YOLOv2 to be able to achieve real-time in embedded devices (Shafiee et al., 2017). YOLOv2 did not have any fully-connected layers in its architecture. To improve the training process for the network, k-means clustering had been applied to the training set of bounding boxes in order to select good priors automatically and improve the accuracy of the model. Moreover, the speed of the network was improved by utilising a new CNN architecture known as Darknet-19, which was a contrast to VGG16, which is what most other frameworks were using at the time. Although YOLOv2 can achieve real-time performance on a powerful GPU, it still remains very challenging for

leveraging this approach for real-time object detection in video on embedded computing devices with limited computational power and limited memory. For this very reason, YOLOv2 would not be a suitable network for the project as the model must by default be able to achieve real-time with a low computation cost as the requirements mention. However, later versions of the YOLO model seemed to be demonstrating improvements from the previous iterations of YOLO. YOLOv3 was released in March of 2018, this iteration claimed to be as accurate as an SSD (Single Shot Detector algorithm), but three times faster (Redmon and Farhadi, 2018). The model still utilised k-means clustering to make estimation on the initial height and width of the predicted bounding box. YOLOv3 also contains a stronger feature extractor network, and changes to its loss function, effectively allowing it to detect more targets ranging from big to small sizes. All versions of YOLO also use full images for its training. Later versions such as YOLOv3 and onwards would be suitable for this project as they demonstrate significant superiority of other algorithms in terms of both accuracy and speed. Although, its computational power use makes a questionable choice as the main model since the model should be able to seamlessly interact with embedded devices such as surveillance cameras.

4.2.2 SSD: Single Shot Multi-Box Detector (2016)

SSD is a single-shot detector that makes predictions using bounding boxes and directly classifies them from feature maps in a single pass. Hence the name, single-shot detector. For 512×512 input, SSD achieves 76.9 % mAP, outperforming a comparable state of the art Faster R-CNN model. Compared to other single stage methods, SSD has much better accuracy even with a smaller input image size (Liu et al., 2016). Unlike the YOLO algorithm, choosing a full connection layer for prediction, the SSD algorithm chooses to extract detection results directly by using a convolution layer for feature maps of different sizes. For feature graphs of size m × m × p, small convolution kernels such as 3 × 3 × p can solve their prediction needs and obtain the expected regression quantity, then flatten it into a one-dimensional vector, and finally make regression calculation (Yang et al., 2020). However, as mentioned by Kim, et al. (2020), in a real-time performance comparison for vehicle type recognition, the YOLOv4 model had outperformed both Faster-RCNN and SSD by achieving an accuracy of 93% for this task. Although, a different vehicle detection study was conducted which involved a proposed model known as “Fast SSD”. This model was a combination of SSD and the Slim ResNet-34 algorithms. Results show that the vehicle detection accuracy achieves 99.3% and the classification accuracy is 98.9% (Chen, et al. 2018). Furthermore, another variation of SSD called Tiny SSD was created for real-time embedded object detection which concluded that very small deep neural network architectures can be designed for real-time object detection that are well-suited for embedded scenarios. This suggest that applying an accurate framework such as SSD to a lightweight model can produce exceptional results when used with embedded systems.

4.2.3 Faster-RCNN (2015)

A research group at UC Berkeley had created a deep convolutional network known as R-CNN (standing for “Region-Based Convolutional Neural Network”) (Gupta et al., 2014). The first publication of R-CNN proposed an objection system that needed an algorithm like “Selective Search” (Uijlings, J.R.R. et al.2013)or something of equivalence in order to propose candidate bounding boxes that were capable of containing objects. The regions would then be passed into a Convolutional Neural Network (CNN) for classification. R-CNN was one of the first object detectors that utilised deep learning. In a later paper publication from 2015 (Girshick, R. 2015), the Fast R-CNN algorithm introduced many considerable changes and improvements to the original C-RNN algorithm. It primarily increased accuracy whilst also reducing the time needed to perform a forward pass. But this newer improvement version R-CNN still entrusted the use of an external proposal algorithm. A follow-up paper that introduced Faster R-CNN(Ren, S. et al. 2015).This R-CNN had achieved a real end-to-end deep object detection system. It achieved this by removing the use of the Selective Search requirement and instead it relied on a Region Proposal Network (RPN). Faster R-

CNN is completely convolutional and is able to predict the object bounding boxes as well as a confidence score. RPN outputs are passed into the R-CNN for the labelling and final classification. However, in a report discussing the SSD framework, it is mentioned that SSD achieves 59 FPS with mAP 74.3 percent on VOC2007 test dataset, vs Faster R-CN, which achieves 7 FPS with mAP 73.2 percent (Liu et al., 2016). These results and the ones mentioned in 4.2.2 are indicative of the algorithm's capabilities and therefore, rules out Faster R-CNN as a potential model for this project on account of its high computational cost, which results in its poor performance in terms of speed.

4.3 Model

The model chosen is called SSD MobileNetV2. This model contains all benefits of utilising an SSD as discussed in the sections prior in this chapter. Furthermore, in section 3.2.2 it was mentioned that applying an accurate framework such as SSD to a lightweight model can produce exceptional results when used with embedded systems. After conducting research into such models, MobileNetV2 was a clear choice. Arcos-Garcia et al. (2018) explains that SSD MobileNet is the fastest and the lightest model in terms of memory consumption. Therefore, this makes the model an optimal choice for deployment in embedded devices in the future. Moreover, MobileNets had been designed with a "mobile first" approach. This means that they are resourceful and are able to run smoothly on mobile devices making this model lightweight enough for the task of integration (the Google AI Blog. 2017). Hu and Ge (2020) have denoted the capabilities of MobileNetV2 by including it in their research paper which had aimed to create a real-time facial expression detector. It was mentioned that the success of their approach was partly attributed to the characteristics of MobileNetV2 as it provided state-of-the-art accuracy and speed for their proposed framework. Therefore, this indicates that the MobileNetV2 has already seen some success when working with face-related recognition problems.

MobileNetV2 has over 2 million parameters by default and was released as part of the TensorFlow-Slim Image Classification Library. This means that MobileNetV2 is compatible for use in Google Colaboratory. The MobileNetV2 model builds on the ideas and foundations created by MobileNetV1, which used depth-wise convolutional blocks as its main building blocks, whereas MobileNetV2 presents two new features to the architecture. Those features being: linear bottlenecks between the layers and shortcut connections between the bottlenecks. The basic structure can be seen in *Figure 13*. MobileNetV2 contains a 53-layer architecture and is based on an inverted residual structure; in which case the 2 residual connections are in between the bottleneck layers. Lightweight depth-wise convolutions are used in the intermediate expansion layer to filter features as a source of non-linearity. Overall, the architecture of the model contains the initial fully convolutional layer, followed by 19 residual bottleneck layers (Sandler et al., 2019). Overall, MobileNetv2 offers a great trade-off between speed and accuracy as well as little computational consumption.

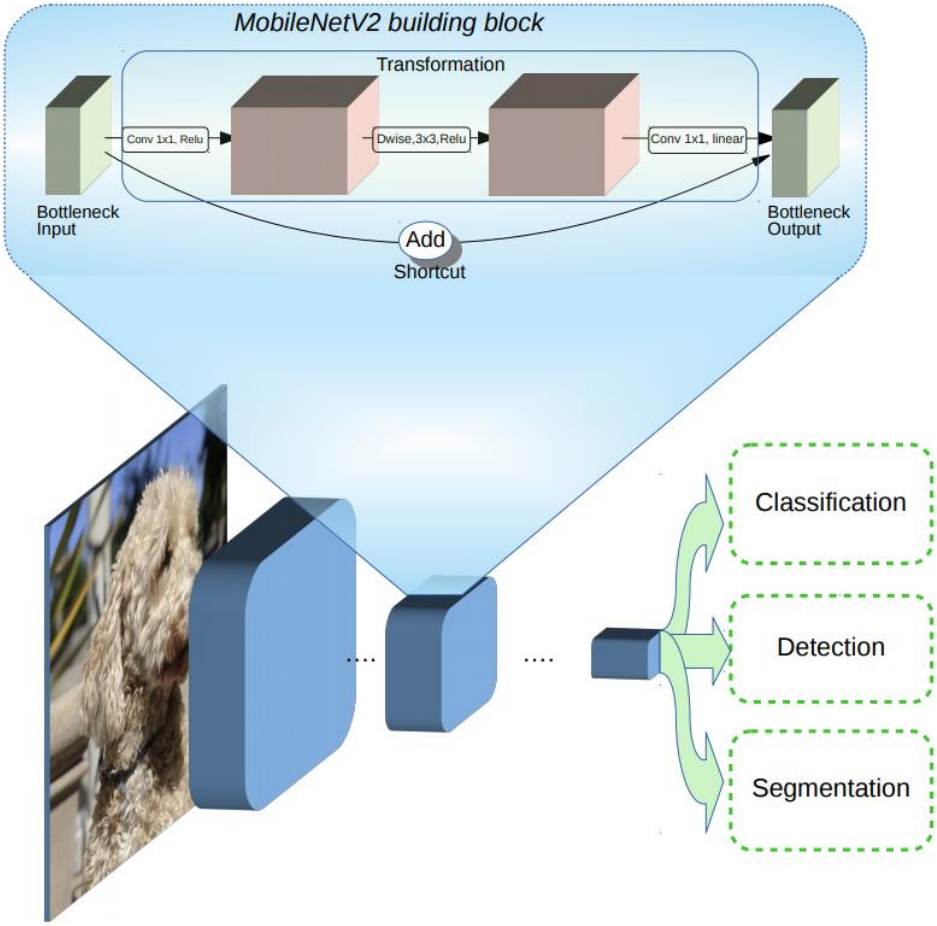

Figure 13: Structure of MobileNetV2, including building blocks[12]

12 Image source

5

Model Implementation

5.1 Chapter Overview

There were two methodologies that were used to create the face mask detection model. This chapter aims to explain and discuss those methodologies and their implementation. Chapter 6 will discuss their results whereas this chapter will focus only on explaining how they were implemented. Both methodologies use the same model discussed in chapter 3, implementing it in various ways to achieve the best results possible.

5.2 TensorFlow API Method

The TensorFlow API methodology is a one-stage method that involves training the chosen CCN model from scratch using supervised learning to eventually achieve accurate results, with each version striving to make improvements. The optimisation of each iteration is based on the performance evaluation and metrics of the previous one. Organised documentation was key to ensure this cycle works continuously. Once an iteration had been trained and evaluated, crucial elements such as the dataset used, the results, previous checkpoints and the model itself are stored. Furthermore, each iteration has a corresponding document which details the approach for that specific iteration and what it was trying to achieve. Examples of how this methodology connects all iterations together is demonstrated in the experiments & results chapter, where each iteration produced is reviewed and discussed. The training environment also holds a vast impact on the output and the overall process of creating an iteration. Outdated, old and/or faulty hardware may intervene with the training process, causing a multitude of errors to occur and disrupt the process. The following sections discuss in detail how this method has been implemented to create an iteration. *Figure 14* outlines the TensorFlow API methodology that has been implemented in this section.

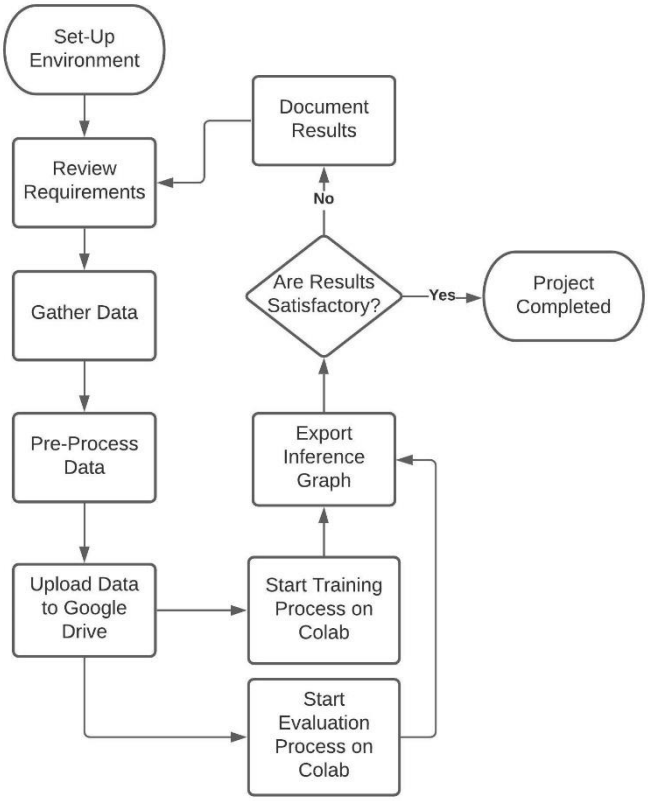

Figure 14: Overview of TensorFlow API Methodology

5.2.1 Programming Language, Libraries & Framework

The TensorFlow Object Detection API[13] was utilised for this method. The TensorFlow Object Detection API is a framework that focuses directly on training a CNN model that can solve problems in relation to object detection. The API was the core of this method as the API provided files for training and evaluation written using the Python programming language, which was the language being used for the entirety of the project. The model was trained from scratch using Python 3.8 and the TensorFlow Object Detection API. Python was the clear choice as the leading programming language for this project and method. The simplicity and conciseness of the language promotes faster application development and testing. Furthermore, it offers an extensive selection of compatible frameworks and libraries such as NumPy, TensorFlow, PyTorch, Sklearn, etc. All catering to development and optimisation of machine learning. Additionally, with all libraries and frameworks being well-documented, this allows for easy implementation and resolving any programming mishaps. A new and more active and stable version of Python was chosen for this project. This excludes any Python version below Python 3. Instead, Python 3.8 was chosen and used for the development. By utilising a newer version of Python, this means that the newer versions of TensorFlow can also be used. TensorFlow 2.x offers a greater choice of refined models in contrast to the dated TensorFlow 1.x models and API. Therefore, TensorFlow 2.x, specifically the latest version 2.4.1, was the more favourable choice.

13 https://github.com/tensorflow/models/tree/master/research/object_detection

5.2.2 Gathering Data

Training a CNN model to detect the presence of face masks among people, involves gathering sets of image samples to represent each class during training, respectively. Secondary datasets of people wearing/not wearing face masks were downloaded from Kaggle. Kaggle is a data science-based website that allows users to publish and find datasets. Many different collections of face mask images were uploaded for other users to use in their experiments. Almost all data used for this project was gathered from this website. However, sometimes datasets contain poor-quality images. This means that the quality was not sufficient enough to be considered a good sample for the CNN model to use during training. This may be due to a multitude of reasons but the common one being image resolution or pixelation as this harms the overall quality of the image most and causes the samples to become unclear, in turn, unusable. To ensure project objective 1.3.2.2 is met, when collecting images for the face mask category, it was essential to ensure that all face masks were being worn by a person in the image. Furthermore, this is also an opportunity to meet project goals: 1.3.2.5, 1.3.2.3 and 1.3.2.4 since the type of images gathered will have a direct impact on whether these goals were met. Therefore, it was important to gather images of people with variations in appearance (1.3.2.4) wearing different types and colours of face masks (1.3.2.3) from various angles (1.3.2.5).

5.2.3 Data Pre-Processing

Before training of the CNN model can begin, there are steps that must be taken in order to prepare the model and the data for training and evaluation. Once data has been collected, it must be converted to a format readable by the model. The model's hyperparameters must also be tailored towards achieving the desired output and the environment must be set-up correctly to accommodate the training and evaluation processes of the CNN model. The upcoming sections will discuss how this was all implemented.

5.2.4 Labelling Dataset

Once the necessary image data had been collected, the ground truth's in each image had to be labelled, the ground truths in this case being the face mask or the lower half of the face. This was done by using an application known as Labelimg[14]. Labelimg was installed from source and required the additional installations of qt5 and lxml. The labelling process involved drawing a ground truth label over the desired object in the image. By labelling the objects in every image, the model would be able to learn the features and attributes of the object during training. Furthermore, these labelled ground truths can be used to evaluate the accuracy of the model, as the detections produced by the model after training can be compared to the ground truth labels to determine how accurate the model is. The ground truths were labelled using the Pascal VOC format. This simply means that once a ground truth label has been drawn for an image in Labelimg, an XML file corresponding to the image is generated. This XML file will store valuable information relating to the label such as: $xmin$. $ymin$, $xmax$ and $ymax$ coordinates of the bounding box, along with the depth (3: recognising the image is RGB) and the label (class) name. *Figure 15* is a screenshot from Labelimg during the ground truth labelling process.

14 https://github.com/tzutalin/labelImg

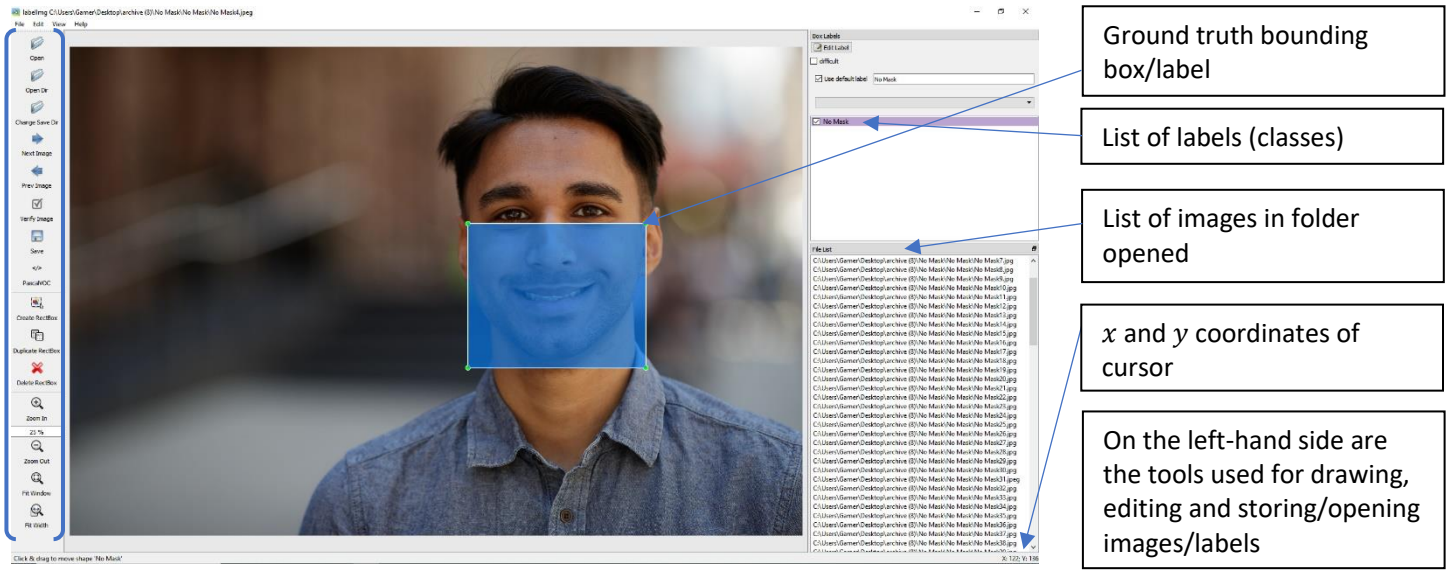

Figure 15: Labelling images in Labelimg

The labelling approach used for this method strays away from traditional face labelling approaches. A traditional face label looks like *Figure 16,* where the entire head or face is covered. Although some very early iterations of the project did follow the traditional style of labelling, eventually the labelling approach had changed. *Figure 17* is an example of the labelling approach used specifically for the No Mask class and *Figure 18* is an example of the labelling approach used for the Mask class. *It should be noted that despite a certain dataset being used, commonly not all images from that dataset are actually used/labelled, especially if the dataset contains a vast amount of samples. Multiple datasets may be referenced however only a few samples may have been used from each, depending on the needs at the time.*

Figure 16: Traditional Face Labelling[15]

Figure 17: No Mask Face Labelling

Figure 18: Mask Face Labelling[16]

It was believed that a different labelling approach would be needed for the face mask / no mask classes since the labelling approach in *Figure 16* was too general and had covered regions of the face that could potentially negatively impact the results. This mainly refers to the features in the upper part of the head such as eyes, eyebrows, forehead, hair, etc. These are areas of the face that are common to both the face mask / no mask classes. By ignoring these areas during the labelling process, it could potentially aid in reducing overlap between the two classes, thus potentially allowing for better predicted detections with minimal confusion or overlapping bounding boxes.

15 Image source

16 Image source

However, when training images using this approach, the surrounding areas of the label are also considered. This means the upper-face features, mentioned earlier, are still being noticed and associated with the face mask / no mask classes during training but they are not directly part of the said class. Therefore, this will result in the CNN model being able to detect the face mask only when it is being worn by a person, as the face mask has always been labelled when it was being worn by a person and not when a face mask is present individually. This approach has helped meet the project goal 1.3.2.2, as the goal stated the final model must be able to detect the "Face Mask" object only if a face mask is being worn by a person.

For the face mask class, only the face mask had been labelled as it was the only relevant area in the image. Whereas, for the no mask class, as seen in *Figure 17,* the lower half of the face where the face mask would typically be covering is labelled. Features unique to the no mask class are labelled such as mouth, lips, teeth, nostrils, cheeks, chin. By labelling these features, an even greater distinction between the two classes is being made as these are features that are usually covered and cannot be seen by the face mask when it is being worn. Meaning that whenever the lower-facial features become visible, this immediately should indicate that no face mask is present, vice versa, if they are being covered – by something that looks like a face mask, then this should indicate a face mask is being worn. Overall, this approach to labelling theoretically provides benefits to bettering the distinction between the face mask and no mask classes, in turn aiding to meet the main project goal of detecting face mask and no mask objects correctly when appropriate (1.3.2.1).

5.2.5 Anaconda Virtual Environment

To utilise the TensorFlow Object Detection API, a virtual environment was created with the help of Anaconda 3. The environment was created mainly to store relevant TensorFlow packages that will be used later for training and evaluation. Within this environment, the latest version of TensorFlow-GPU (2.4.1) had been installed as well as Protobuf (Protocol Buffers). Protobuf is a protocol for data serialization, similar to XML and JSON. Protobufs use a structured format when data is being stored, therefore they can be represented in Python using classes. Installing and compiling protobuf was also essential for generating the label map as that file is saved as a .pbtxt. Other dependencies such as Cython, 2015 visual C++ build tools and imutils were installed. The full installation guide to replicate this environment can be found here. Originally, the training process was going to be executed locally, that is why the creation of the virtual environment was needed. However, due to technical issues, an alternative training approach had to be used. This had resulted in the virtual environment becoming unnecessary for training. Despite no longer being needed for its original purpose, the virtual environment was still used for working with other parts of this method such as creating TF records.

5.2.6 Creating Label Map & TF Records

Once all images had been labelled and their corresponding XML files had been generated, it was time to partition this data into a training and testing set. The training set is provided to the model for it to use and learn from, whereas the testing set is used for evaluation purposes. Generally, the datasets were divided 90% ± 2% for training and 10% ± 2% for testing. When partitioning the data, it was essential to ensure that the classes are divided equally, if a data imbalance is to occur, then one class may perform significantly worse than the other. Next a label map had to be created. TensorFlow requires a label map, which stores every label/class mentioned in the XML files and assigns these labels an ID number. In this instance, there are only two labels stored representing the face mask and no mask classes. Additionally, the label map file is stored as a .pbtxt file, which is readable by the model.

```
item {
   id: 1
   name: 'NoMask'
}

item {
   id: 2
   name: 'Mask'
}
```

This is an example of what a label looks likes. Specifically, this was the label map used during the implementation of this method.

Each class/object is assigned to a unique ID. If there were no unique ID's, then even if whether the model made accurate detections, the model would label the detections incorrectly or using the same label represent both classes.

Once this procedure had been completed, the rest of the data had to be converted to a format which is readable by the CNN model. This involves generating TF Records for both the training and testing datasets. TF records are files that store data as a sequence of binary strings. Therefore, the structure of the data needs to be defined before that data can be written as a TF record. In this instance, the image data had been structured in the XML files that were generated during the labelling process. Therefore, the XML files will be converted to TF records. This is where the TensorFlow Object Detection API comes in, as it provides the necessary documentation and files to help generate the TF record files.

For dataset optimization, it is essential to ensure that the images (before labelling) are not too large in terms of file size and dimensions as the TF records file will reflect this. During implementation, this issue arose, causing issues with creating TF records and with GPU memory consumption when training was attempted locally. Some images were of very high quality but these same images were also very memory-expensive as each image would have a file size of roughly 1MB. 500 of these images would result in 500MBs. To train a CNN model, it is recommended that there are at least 1000 images in the training dataset. Therefore, cutting down on the file size was required to allow for more images to be added. One way this was achieved was to discard the images that would exceed 800KB. Another approach used to reduce file size was to batch resize all images, this would greatly impact the overall file sizes of the images resized. Some sets of images were resized to 640x640 as this was the input size used by the MobileNetV2 model that was chosen.

Additionally, by resizing the images, another error had been resolved. This error would occur when creating the TF records after labelling has finished. The ground truth boxes were too large as indicated by the exception returned when attempting to create TF record files: "Assertion failed: [maximum box coordinate value is larger than 1.100000:]". This error refers to the ground truth label being too large or out of bounds. However, sometimes it was necessary for the label to be a large size to cover the desired regions of the image. Therefore, all images that exceed 1000x1000 had to be resized or discarded. The thought process was that if it is true that the error is alluding to the ground truth labels being too large or out of bounds, then the actual problem resides with the image itself and not the label. This is because the label will inherently be of larger size when working with a larger image. Additionally, Labelimg will stop any labels from going out of bounds. Resizing the images also had a positive impact here since the error had disappeared, thus making the assumption about image size correct.

Many issues had come to light during the pre-processing stages of this method. However, a surprising issue, which had nothing to do with the image dimensions or file size as previously discussed, had appeared when attempting to create TF Record files. A data corruption error was returned when creating the TF record files: "UserWarning: Corrupt EXIF data. Expecting to read 4 bytes but only got 0." The cause was unclear; however, the assumption was that there were one or more corrupt XML files. This may have been caused by a label being created in Labelimg, which would consequently create an XML storing all of the information about the label, and then that label might have been deleted. If the label were to be deleted, the XML file would still exist, however the data stored in it would become invalid. This issue was resolved in an unorthodox fashion, where the

data was divided into multiple folders and a TF record file was created for each folder to examine which folder(s) would return the error and which folder(s) were clean. Once a folder had returned that same error, the data from that folder was then split equally into another two folders and TF records were generated for both of the new folders to determine which folder had the error. This process continued until the data was narrowed down to about 125 images. As there was only a small amount of images remaining, all of the images were deleted. Overall, this issue had presented itself more than once. Referring back to 5.2.6, it is mentioned that the data was split into 90% ± 2% for training and 10% ± 2% for testing. It was not strictly divided 90/10 due to the impact that this corruption error had. There was a very slight data imbalance at times, but not large enough to cause any significant change in results. The division of data between classes was always closely monitored every time more data was introduced.

5.2.7 Training Environments

Originally, the model was supposed to be trained and evaluated locally however this proved impossible without an additional GPU and better technical equipment. There were a vast multitude of errors spanning for weeks which had impacted the training and evaluation of the model when locally trained. The model was going to be trained locally using the GPU installed in the machine, the GPU installed was the GTX NVIDIA 1060 6GB. As mentioned in 2.3, research has enabled GPU's to be used for training neural networks. Therefore, utilising a GPU for training was thought to be the optimal decision when implementing this model. Additionally, using the CPU for training was also a viable possibility. However, this would slow down the training process by roughly 3x and put a lot of strain on a very important component of the machine for a long amount of time. Thus, further supporting the use of a GPU instead. To set-up the local environment for training using a GPU, CUDA 11 and its corresponding cuDNN version (8.0) had to be installed to enable TensorFlow GPU 2.4.1 to be used. However, the GTX NVIDIA 1060 GPU only had 6GB of RAM, which proved insufficient for training the model as the training process often halted and returned a "_dist_train_step -> _dist_train_step" error, indicating a VRAM memory failure. Furthermore, there had been other errors caused by lack of sufficient equipment such as: "CUDA_ERROR_LAUNCH_FAILED" and "cuDNN launch failure".

Therefore, there was a need for specialist equipment in order to execute the training and evaluation process as intended. Alternative methods of training and evaluating the model had to be investigated. Google Colaboratory is a data-science, Jupyter notebook style web application provided by Google. Using this application as a platform for training and evaluating CNN models was the most viable option as Google even provided a GPU runtime environment. Meaning it was possible to connect to a GPU runtime environment and utilise a specialist GPU that was virtually hosted by Google. However, a Google Colab subscription was required to use this service extensively since it was a shared resource and if used for too long continuously would result in a cooldown being issued. The subscription virtually removed this cooldown and offered priority to the shared resources as well. The GPU that is assigned to the user when they are connected to the GPU runtime environment is called the "NVIDIA Tesla K80". This is a GPU that acts as an accelerator for demanding computational tasks and provides 12GB of RAM, double what the local GPU was able to provide.

5.2.8 Configuring Model Pipeline

After the TF record files were created successfully, the model pipeline had to be downloaded and modified appropriately to output the desired result. The pipeline had been downloaded from the TensorFlow 2 model zoo[17]. This GitHub repository contains a selection of TensorFlow 2 model pipelines, the chosen model was called: "SSD MobileNet V2 FPNLite 640x640". The download had

17 https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

contained the pipline.config among other files. Within this file, a few parameters needed modification. The number of classes was set to 2 (mask / no mask) on line 3, The batch size had been changed on line 135. The path points needed to point to the correct folders and files. The fine tune checkpoint parameter needed to point to the last checkpoint of the downloaded model on line 165. The fine tune checkpoint was set to detection on line 171 and the label map paths had to be set on lines 175 and 193. Lastly, the path to the train.record file needed to be set on line 177 and the same needed to be done for test.record on line 189. (Note: the .record files are the TF Records, as discussed in 5.2.6).

5.2.9 Evaluation Criteria

This method uses the COCO mAP metrics to evaluate the model and its predictions. COCO mAP metrics provide a comprehensive evaluation of the model, from multiple IOUs and other variations. Resulting in a detailed summary of the models performance, allowing for better decision-making to be done using the provided information. The COCO mAP metric was also written into the pipeline.config file on line 181, as it was the chosen evaluation metric. Furthermore, in line 185, "num_visualizations: 20" was entered as this will return 20 image samples after evaluation with bounding boxes representing the detections made by the model. This provided a visual representation of how the model performs on the test images.

5.2.10 Model Training & Evaluation

Google Colaboratory uses Google Drive to store, write to and read files. It is possible to connect a Google Colab notebook with Google Drive, allowing access to all the files stored on there. Therefore, the file directory structure that was set-up and intended for local use, had to be replicated and uploaded on Google Drive. After the TF records and label map were created, they all had to be uploaded to Google Drive along with the entire TensorFlow Object Detection API. The TensorFlow API folder had to be uploaded as a .zip file due to the file size, it was then unzip using a line of command, executed from a Google Colab notebook. Once all of the files and folders were allocated to their respective locations, the Google Drive folder set up looked like this:

```
tf2
├─ object_detection/
│   ├─ ...
│   ├─ training/
│   ├─ images/
│   ├─ ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8/
│   ├─ train.record
│   └─ test.record
└─ ...
```

Once the Google Drive folder set up was completed, two Google Colab notebooks had been written; one titled "training" and the other titled "evaluation". For both notebooks, Google drive had first been mounted to allow access to the files in the drive. The code below was used to mount Google drive to the notebook. Once executed, the code will return a link, requesting access to the Google account. Once all the steps are followed and access is granted, the Google Drive from the chosen Google account will be imported. As mentioned previously, TensorFlow is already installed in Google Colab by default, therefore there was no need to install, however some additional dependencies required installation: tf-models-official, tf_slim and lvis.

Tf-models-official is a collection of models that utilise TensorFlow's high-level api. They are tested, updated to match the latest version of the TensorFlow API and generally well-maintained. This import will support the use of a TensorFlow 2 model such as MobileNetV2. Tf-Slim is a high-level library that simplifies the process of training and evaluating neural networks. It allows for the user to

remove boilerplate code and define the models more compactly. Lvis is described as a dataset for large vocabulary instance segmentation[18]. Once the dependencies were installed, it was time to change directory into the object detection folder, then load and launch tensor board in preparation for training/evaluation. Tensor board had been pointed to examine the "training" folder as that is where the model checkpoints will appear when generated. Model checkpoints are generated every 1000 steps.

The difference between the training and the evaluation notebook is the cell that initiates the training or evaluation process. The code below initiates the training process: As depicted by *Figure 14* in 5.2, the training and evaluation had been running simultaneously, in parallel with each other. Even though the training and evaluation code come from the same file, the evaluation code is not embedded in the training loop, therefore when training begins, the only metrics generated are the loss metrics such as classification loss, localization loss, total loss, etc. Although useful, the mAP metrics need to be generated as well to evaluate the model completely. By executing the evaluation and training notebook, graphs for COCO metrics such as mAP can be generated. A checkpoint storing all of the model's information thus far is created every 1000 steps, when this checkpoint is generated, it is stored in the training folder. The evaluation loop will listen for any new checkpoint entries in the training and evaluate them immediately. These checkpoints generated an "events.out.tfevents" file which stores all of the evaluation data of that said checkpoint. Using these event files, a graph can be plotted for every checkpoint (1000 steps), allowing for the progress of the model to be visualised. This graph was monitored using tensor board and allowed for the progress to be monitored in real-time.

5.2.11 Continuing Training from Checkpoint

Once a model is trained, it is possible to continue training that model from where it left off using new data. This is done by using the last checkpoint of the trained model, the path to this checkpoint needs to be established in the pipeline.config file, specifically under the "fine_tune_checkpoint:" option. To introduce new data, the new data needs to be added on top of the existing data that was used for training. Then a new TF records file needs to be created which encapsulates all of that data in the folders, new and old. The model will recognise which data it has seen before and which data it has not due to the information stored in the checkpoint file. Therefore, if training were to be continued from the last checkpoint, the model would not be getting retrained on the old data. This is important to understand as this is a way that new data can be introduced to improve a current model, it removes the need to start training the model from scratch every time new data is added.

5.2.12 Exportation and Visualisation

Lastly, once training and evaluation had been completed, the model needed to be compiled and exported as an inference graph, older versions of TensorFlow (1.x) refer to this as a frozen graph. The inference graph folder contains the last checkpoint of the model, the saved_model.pbtxt file which stores the model in a binary format, a variables folder and a copy of the pipeline config.

In order to visualise the detections in real-time, a python script was written which employed the use of the OpenCV library to visualise the bounding boxes of the detections in real-time via webcam input. The required files were the label map, to map the detections to their respective classes, the saved_model.pbtxt from the inference graph folder was also required as it was the compiled and trained model. The detection threshold had also been set to 0.5 in this area of the code. The live webcam feed was initialised using the cv2 module. Furthermore, the TensorFlow virtual environment was needed here as it provided object_detection.utils, which were necessary for establishing object detection, since this import provided other imports such as visualisation_utils and the label_map_utils. The visualisation_utils was the primary import required to visualise the

18 https://www.tensorflow.org/datasets/catalog/lvis

bounding boxes and labels. Whenever image data is loaded in, it is then converted into a NumPy array with the shape: [y, x, 3]. The y and x represent the height and width respectively, while the 3 represents that the image is RGB as RGB has three channels. A while loop was created that acquires frame dimensions and expands them to have a shape of: [1, none, none. 3], essentially converting it into a single-column array whereby each item in the column has a RGB pixel value. The input needs to be a tensor as all of the outputs are batch tensors. Therefore, the input needs to be converted accordingly. Lastly, once all the data was converted into the required formats, the settings for the bounding boxes had been written, including thickness of the outlines, colours, label size and colour, etc. Additionally, the confidence score is displayed next to the class label. Also, if the loop breaks (by closing down the webcam input window) then the cv2 module will destroy all windows and the program will cease to execute. Making detections via webcam input is nothing more than a series of images being inferenced in real-time one after another; therefore, the approach is mostly the same as if the detections were being visualised for a single image. As this is a more computationally-heavy task, a GPU must be utilised to complete this task reliably. However, Google Colab cannot be utilised as it does not support webcam input. This resulted in using the installed GPU of the local machine being the only choice. It was able to support this task but the initial inference time for the model (loading the model) would range anywhere from 55s to 66s. Which can be considered fairly slow. However, the real-time detections via webcam input were consistent and displayed high frames per second no matter which iteration of the model was being used. Performance in real-time will be discussed in the next chapter where the outputs of the model are reviewed.

5.3 Transfer Learning Using Keras Method

This methodology aims to train the CNN model using transfer learning and the Keras library from TensorFlow. Unlike the TensorFlow object detection API, which was a one-stage implementation, this methodology is completed in two stages. The TensorFlow Object Detection API Method involves the model learning to detect faces as well as face masks. Whereas this methodology only requires the model to learn to classify whether a face contains a mask or not, instead of learning to localise the face as well. This is done by using a pretrained face classifier and applying the trained face mask model to examine the detected face to find out whether the face image contains a mask or not, then the frame is classified accordingly. The first stage of the implementation involves a pretrained MobileNetV2 model being trained to detect people who are wearing/not wearing a face mask. The model is trained using transfer learning and the Keras and TensorFlow libraries. Once the model has completed training, the model is saved as a .h5 file to disk.

The second stage involves the use of another pretrained model, this model has been trained to detect faces. The model used for this task was not trained during the development phase of the project, instead it was trained and uploaded to a GitHub repository[19] by another person. Once the faces have been localised by the pretrained face classifier, the ROI's (Region of Interest) being the facial features of the entire face, are extracted. These extracted features are then passed through the face mask model that was trained in the first stage in order to classify whether the face was wearing as a face mask or not. The results are then displayed accordingly with a confidence score being displayed which refers to how confident the model predicts that the person is/is not wearing a mask. It does not refer to how confidently the face classifier model predicts a face. Figure 19 below illustrates how the methodology works.

IMPORTANT NOTE: It must be mentioned that the idea of this method was not authored during the production of this project. Instead, this method was published by Adrian Rosebrock[20] and available online to all who wish to recreate it. The reason for including this method had been to examine

19 https://github.com/sr6033/face-detection-with-OpenCV-and-DNN

20 https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

different perspectives of training CNN models for the purpose of the project and to see if this method could obtain a higher accuracy. The transfer learning training aspect of the method (stage 1) had already been of knowledge beforehand, however. This method mainly adopted the visualisation part (stage 2) as that was the part most unique to this idea. Lastly, the main focus during the project had been the development of the TensorFlow API method, hence the lack of experiments for this method. This method had mainly been implemented for comparison and to see whether it would obtain higher results than the TensorFlow API method. K-Fold cross validation had not been part of the original method idea and was implemented separately.

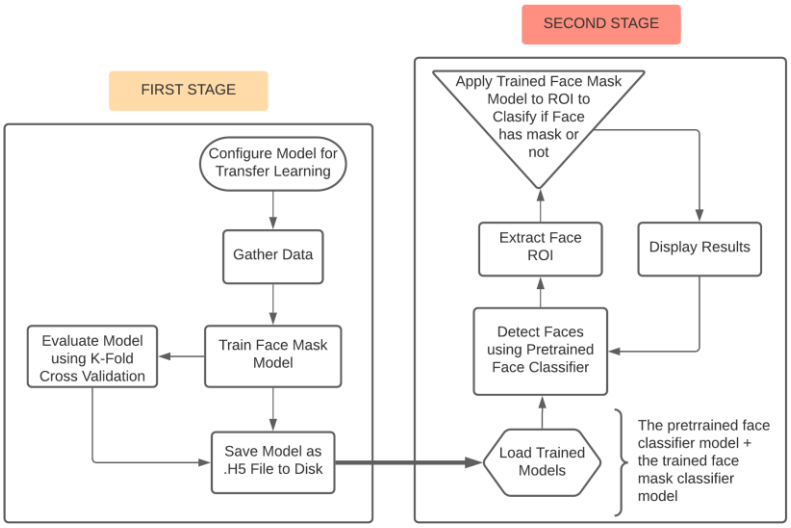

Figure 19: Overview of Transfer Learning with Keras Methodology

5.3.1 Programming Language, Libraries & Framework

As mentioned previously, the Keras and TensorFlow libraries are used in this implementation, therefore Python is the appropriate programming language to be used as it supports these libraries and more. The use of Python for this project has already mostly been covered in 5.2.1. The Keras library comes from TensorFlow and is imported typically as such: “from tensorflow.keras”. By importing the Keras library, this provides the ability to utilise a multitude of useful modules and allows for transfer learning configuration. The Keras library allows to load the MobileNetV2 model as well as modify it by creating a new fully-connected layer. The library also allows for implementation of automatic data augmentation and loading and pre-processing image data. Another useful library was the sk.learn library. Through this library, “K-Fold” had been imported, which allowed for the model to be evaluated using K-Fold cross validation. Additionally, this library had also allowed the dataset to be segmented for K-Fold cross validation and the class labels to be binarized. Lastly, matplotlib was used for plotting the training and validation accuracy/loss and imutils for locating the images in the dataset. Datetime was also imported to record the duration time of the training.

5.3.2 Gathering Data (Stage 1)

Image data needs to be gathered to represent the two object classes: face mask and no mask. However, unlike the TensorFlow API method, there is no pre-processing involved for the data other than organisation. The data does not need to be labelled using Labelimg or converted to XML files or TF Records. Instead, the data is stored into their folders and the folder names act as the labels for these images. For example, images of face masks are all stored into a folder named “Face_Mask”. This is how the classes are categorized without using ground truth labels. All image data used for this implementation had been collected from the website known as Kaggle, as previously discussed in 5.2.2.

5.3.3 Fine-Tuning Model for Transfer Learning

It is important to reiterate what was said in 2.4.1, transfer learning involves utilising a pretrained model to exploit the knowledge that it currently has to improve the training and generalization of another task. In this implementation, the pretrained CNN model is MobileNetV2 and it has been pretrained using the ImageNet dataset which had been discussed in 2.2.1.3. The ImageNet dataset has been proven to stimulate better performance when used with transfer learning. This dataset is one of the largest object detection datasets available with over 14 million images. Using the Keras library, the MobileNetV2 model had been downloaded, pretrained on the ImageNet dataset.

5.3.3.1 Settings & Hyperparameters

One of the first things implemented was the path to the dataset, as well as the names given to the exported model and graphs. The number of folds that will be used for K-Fold cross validation was also stated in this section, this number was set to 5. Next, the hyperparameters for training were set. The number of epochs set was 10. The number of epochs refer to how many cycles the full training dataset will go through during training. The batch size was set to 18. Previously, the local GPU was not able to handle this high of a batch size. However, in those instances the model was being trained from scratch which required more memory and computational power. With this implementation the training time for the model is cut significantly, this allows for a higher batch size to be used without clashing with the upper limit of the RAM. Lastly, the learning rate defined as LR, is set to 1e-4.

5.3.3.2 Data Input Pre-Processing

The images from the dataset and folders are initialised as a list. The data and the label arrays are initialised. A loop is created where the input image data is converted to 224x224 to ensure that all images are of the same size before training. These arrays are then appended to either the label or data array accordingly and then ensuring that the training data is stored as a NumPy array. Once this was completed, the class labels had been one-hot encoded, which is a pre-processing step that represents the labels from the NumPy array in a binary column format in accordance with their class.

5.3.3.3 Creating New Fully-Connected Layer

The layers that contain the weights from the ImageNet data are frozen so they will not be affected during the training of the face mask classifier. New trainable parameters must therefore be created that will store the new weights when the face mask classifier is trained. The architecture of a CNN model is discussed in Chapter 3. Specifically, fully-connected layers are explained in sections 3.2.4 and 3.2.5. As mentioned in 4.3, the MobileNetV2 has 53 layers, however the layers containing the pretrained weights at this point are frozen and their parameters have become non-trainable. The newly created fully-connected layer will become the last fully-connected layer, meaning it will become the output layer, more information about this layer can be found by referring back to 3.2.5.

The last fully-connected layer now has to be modified. Within the layer, an AveragePooling2D layer of size 7x7 is added, this layer is connected to the output activation: “out_reLU”, which in turn is connected to the last convolutional layer in the block, meaning a connection is established to the rest of the model. A flatten layer is added afterwards which is connected to the averagepooling2d layer. This is then followed by a dense layer with the reLU activation function, which provides 163968 trainable parameters. Next, a dropout layer with a probability of 0.5 is introduced. Chapter 3 does not discuss the dropout layer as chapter 3 only focuses on establishing the core fundamentals, to allow for the technical documentation of this project to be understood more clearly. A dropout layer is not always featured in CNN models, it was not part of the MobileNetV2 original architecture. A dropout layer is important as it can prevent

CNNs from overfitting. A FC typically uses most of the available parameters, meaning that the neurons will develop a co-dependency among one another during the training process. This restrains the power that an individual neuron has, which will eventually lead to the training data overfitting. A dropout layer drops the connections between layers randomly. When a connection is dropped then the neurons will no longer have this co-dependency. Lastly, a final dense layer is added which is connected the dropout layer and provides 258 trainable parameters and integrates the sigmoid activation function, which is appropriate for binary classification.

Once these modifications to the last FC layer are made, the new model then needs to be compiled with all of the changes included. The loss function, optimizer, learning rate and metrics are stated here. The loss function used is binary_crossentropy, as discussed in chapter 3 section 3.2.6.2, it is most appropriate to use this loss function when dealing with a binary classification problem such as face mask or no mask. The chosen optimizer was “Adam”, it is an adaptive learning optimization algorithm and it has been created for training DNNs. It is an algorithm which serves as a replacement for the stochastic gradient descent method when training DNNs. As this is a hyperparameter, it can be changed if necessary. The learning rate is the same as defined by the variable LR, mentioned in 5.3.3.1. The metric being assessed is accuracy.

5.3.4 Image Data Augmentation

Image data augmentation had been used in this implementation to create more images for training. The impacts of this technique are discussed in chapter 2 section 2.4.2. As mentioned in that section, image augmentation can include translation, rotation, flips, mirror, zoom, colour perturbation and shear. For this implementation, the zoom range was set to 0.12, the rotation range to 15, the shear range to 0.15 and the horizontal was = true. The width and height shift range had been set to 0.2 and fill mode set to “nearest”. This will create augmented images which are not so disruptive visually to the point where they become bad samples. The use of image augmentation here will also expand the dataset, in turn providing more training data for the model.

5.3.5 Training & K-Fold Cross Validation

The data was originally split into trainX, testX, trainY, testY before K-Fold cross validation was going to be implemented. However, now the trainX and testX had been concatenated and assigned to the “inputs” variable using the “np.concatenate” function. Likewise, the same was done for the trainY and testY but they had been stored in a variable named “targets”. The K-Fold cross validation had been defined using the KFold function, the number of splits = the number of folds, shuffle had also been set to true. A for loop had been written starting from where the FC was being modified, it encapsulated everything from there to where the code for training ends. (lines 99 to 157 in the training.py file). Text in the format of strings had been written to inform the user which fold was being trained and each fold was being separated using multiple entries of the “-” character. The metric measured was accuracy and each time a fold had completed training, the score per fold was printed. Then the next fold would begin training and the fold number would be incremented each time a new fold would start, until it reaches the maximum number of folds set in the “num_folds” hyperparameter.

Once the model had finished compiling, it was ready to be trained. The code written to initiate the training process included fitting the newly compiled model which included stating the batch size, the epochs, training data, augmented data and numbers of steps. Additionally, when the model started training, a timer had been initiated to measure the duration of training. All of this was stored in a variable named “mh” which stands for model history. Once the model had completed training, the graph of the final fold was created using the matplotlib and the .history function was used with the mh variable to visualise the training and validation data. Admittedly, it would have been better to draw a graph for every fold but there was difficulty implementing this due to time

constraints. Once the model completed training, it was saved as a .H5 file to the specified location and that marks the end of stage 1.

5.3.6 Visualization (Stage 2)

This is the second and final stage of the implementation for this method. In this stage the trained face mask model is deployed and used in combination with the pretrained face classifier. The pretrained face classifier model used is called "res10_300x300_ssd_iter_140000.caffemodel"[21] but it will often be referred to as the face classifier.

5.3.7 Input Data Pre-Processing

The input images have to be pre-processed. A function called mask_detection was written which takes in three parameters. These parameters being the frame/image from the live webcam stream, faceDet: the model that is used to detect the faces in an image and maskDet, which is the model trained in stage 1. The images are pre-processed using the blobFromImage function from OpenCV. Beforehand though, the frame dimensions are recorded for future display and scaling purposes. The images are resized to 300 x 300 pixels as this is the required input for the face classifier model, then mean subtraction is performed. A list of faces, their locations for the face classifier model are initialised as well as the predictions for the face mask/no mask model. Whenever a face is predicted, it is essential to ensure that the detection meets the minimum confidence threshold. The threshold was set to 0.5, similar to the TensorFlow API method. To exclude any weak detections, the confidence score must be greater than the threshold stated. A loop is created that constantly monitors detections to ensure any weak ones are discarded. Furthermore, it was essential to ensure that the predicted bounding boxes fall inside the image boundaries to avoid any errors when visualising the detections.

5.3.7.1 Face ROI Extraction (Region of Interest Feature Extraction)

A face ROI refers to facial landmarks such as the mouth, eyes, chin, eyebrows, nose, etc. These features can be extracted to localise different areas of the face. Therefore, using this feature extraction method, it would be possible to determine whether a face mask is being worn or not. This approach is very similar, but not exact to the approach used in the TensorFlow API, described in 5.2.4. Where if the lower half of the face is covered, it indicates a mask is being worn. However, the face mask model is not applied until a detection is made by the face classifier.

Once a strong detection which exceeds the minimum threshold of 0.5 is made, the ROI for that face which had been detected needs to be extracted. The ROIs are extracted using NumPy slicing and are pre-processed in a similar fashion to how the input images were pre-processed during the training stage. The image was resized to 224 x 224 and converted to an array where it becomes pre-processed and dimensions become expanded. (See lines 48 to 53 in the Webcam_Mask_Detection.py file). Once the ROIs of face are extracted and pre-processed, the bounding boxes and the ROIs will be appended to their lists, respectively. The face ROIs are appended to the ROI list, while the bounding boxes are appended to the loc list. Lastly, a while true loop is created, which loops over the frames in the video stream. For every frame captured, that frame will become resized to ensure it has a maximum width of 350 and the extracted and pre-processed face ROIs of that frame are passed through the face mask classifier model to determine whether a face mask is present or not.

21 https://github.com/sr6033/face-detection-with-OpenCV-and-DNN

5.3.8 Visualising Predictions

After a face is detected and its features are extracted and processed, the face mask classifier model is ready to be initiated. However, firstly the models need to be imported and the webcam stream needs to be initialised. The webcam is initialised using the cv2 module (OpenCV) and each frame is displayed at 1280 x 720 (720p – matching the webcam quality). The models are imported with variable names assigned with respect to the parameters accepted by the mask_detection function mentioned in 5.3.7. Within the while true loop, a for loop is created which loops over the predicted results to unpack the bounding boxes of the face mask classifier and the face classifier as well as establish the aesthetics of the bounding boxes, text and labels, etc. An if statement was written to determine what the appearance of the output will be. Which essentially states that If text is equal to face mask, set the colour to green (0, 255, 0). Else, set the colour to red (0, 0, 255). The confidence score of each frame is paired with the label of the class and is multiplied by 100, meaning the maximum confidence can be out of 100%. Lastly, some post-processing is done by assigning the window title and assigning a key that when pressed, will shut down the window and end the webcam stream. The key assigned was “x”, once this key is pressed, all cv2 windows will be destroyed and the capture will be stopped.

6

Experiments & Results

6.1 Chapter Overview

The experiments & results chapter aims to discuss all of the documented iterations of the project from both the TensorFlow and the Keras methodologies. Routinely, once an iteration was completed, files such as datasets and results were stored and documented. This allowed for all aspects of said iteration to be reviewed, including the result metrics. The next iteration that followed customarily was optimised for more favourable results based on the output metrics and interpretation of its predecessor version. The iterations belonging to the TensorFlow methodology generally adhered to the agile methodology more than the Keras methodology did. As 8 out of the 9 iterations belong to the TensorFlow methodology. The two methodologies had also used different evaluation techniques. Nevertheless, the same model was used for both, thus creating grounds to make a fair comparison and discussion between the efficiency of each methodology as they both held a major common denominator.

6.2 TensorFlow API Method

In this section, the experiments conducted and their corresponding results following the TensorFlow API method will be presented and discussed. Further analysis of the results, approaches and method can be found in chapter 7. The results from each experiment were analysed in order to find ways to improve the performance for the next iteration. The main metrics analysed were accuracy and training/validation loss. This was done by using the COCO mAP metrics to evaluate the performance of the model on test images. Additionally, every model produced also had its performance examined when dealing with live webcam input. The accuracy of the predictions produced via webcam input, indicated by the confidence scores and positioning of the bounding box, was the main metric reviewed. By testing the model using live webcam input, it offers a practical perspective on the performance of the model, in contrast to the more mathematical results produced by the COCO mAP metrics. Additionally, the test images are returned after training/evaluation, containing predictions made by the model. These test images are available to be accessed and viewed using tensor board and can provide a deeper understanding of the models performance. It should be noted that the x axis in the graphs shown in this method represent the amount of *steps* the model has been trained for. Whereas the y axis represents the *metric* that is being assessed and displayed, this metric is indicated by the title of the figure.

6.2.1 "INITIAL BASELINE" EXPERIMENT

This was the first attempt at creating the face mask detector, it also serves as a starting baseline for other versions to come. Clearly, during this experiment, the model was undertrained and did not have enough data to produce a good output. However, there is a possibility that the amount of data used was indeed sufficient as there were 1000 images used collectively for both the face mask and no mask classes. It could be argued that the poor results produced were due to the lack of training. However, the accuracy of this assumption is uncertain as the model perhaps needed more data too.

6.2.1.1 Experiment Settings

The model was trained for 5000 steps on a batch size of 6. Which was deemed a good starting point for the model. In hindsight, the number of steps should have been increased as 5000 would not be enough for the model to analyse 1000 images well. However, the batch size was set to a smaller size of 6 as smaller batch sizes are known to foster faster learning. The sigmoid activation function was used here as two classes were being trained at this stage and it was stated in chapter 3, the sigmoid activation function is best suited for binary classification. *Note: the sigmoid function was used for all of the experiments unless stated otherwise. There was no need to alter this hyperparameter as long as the goal of the experiment was binary classification and not categorical classification.*

6.2.1.2 Dataset

1000 images had been used for this experiment, with 500 images belonging to the face mask class and the other 500 being a part of the no mask class. 100 Images from each class were put into the testing set and the other 800 were used for training, splitting the data 80/20. However, in 5.2.6, it was mentioned that the data is typically split 90/10, this was a general statement, which can vary from experiment to experiment. This is especially true with the earlier versions as parameters and datasets were being experimented with until satisfactory results were found. Three datasets were used for this experiment. The dataset used for the no mask class by prasoonkottarathil[22] had featured high-quality close-ups of people who were not wearing a mask. The dataset unfortunately did not place these people into much context, the people in the dataset were all relatively in the same environment with no variation in scale. However, the dataset did display the features of the no mask class very clearly. Two datasets were mixed and used for the face mask class. The first dataset by dhruvmak[23] (220 images) was a well-put-together dataset which contained many examples of people wearing face masks. There were many examples of different face masks being worn at various angles in different environments. Whilst it provided good samples for training, there were only 220 images in this dataset. Therefore, a second dataset by saurah403[24] was introduced to fill in the remaining 280 images to restore data balance between the classes. Much like the other dataset, it provided thorough samples in context for the model to use, with various angle and face mask types featured.

6.2.1.3 Results

The model had completed training in 1hr 31mins 5secs. Unfortunately, the results of this experiment had been poor and unsuccessful as the mAP scored below the 50% threshold, in turn meaning that the results could not even be visualised using OpenCV as discussed in 5.2.12. This is due to the threshold being set to 0.5 to filter out any weak detections. This model had only achieved an

22 https://www.kaggle.com/prasoonkottarathil/face-mask-lite-dataset

23 https://www.kaggle.com/dhruvmak/face-mask-detection

24 https://www.kaggle.com/saurah403/face-mask-detectionimages-with-yolo-format

accuracy of 0.4584 mAP (46%), similarly the mAR achieved per single detection was also low, achieving only 0.4771 (48%). However, it is evident by reviewing the graphs in *Figure 20* and *Figure 21*, that the scores for mAP and AR were still rising. Typically, when a model reaches its true score, it will become more stabilized and the scores will no longer fluctuate too greatly. However, the graphs clearly show no sign of stabilizing. Therefore, this is a clear indication that the model is undertrained. More training time could increase the scores of the mAP and mAR. Additionally, the training loss (0.418) seen in orange and validation loss seen in blue (0.618) was still high as shown in *Figure 22*, further advocating the need for longer training. One of the impacts of undertraining and possibly the dataset being used, is displayed in Figure 23. The detection of small objects (objects with an area of $< 32^2$) is non-existent. See Table 1 in appendix 1 for full results.

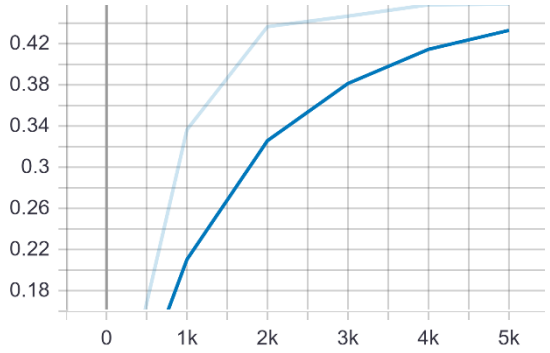

Figure 20: mAP Graph for "Initial Baseline"

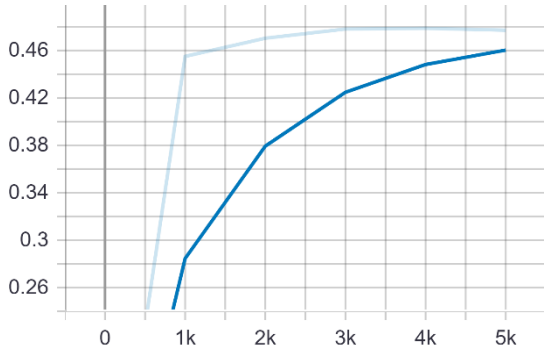

Figure 21: mAR Graph for "Initial Baseline"

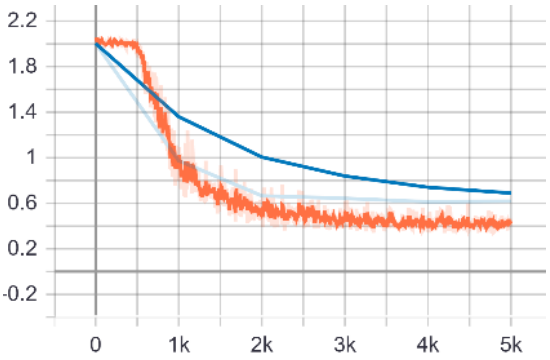

Figure 22: Training/ validation loss for "Initial Baseline"

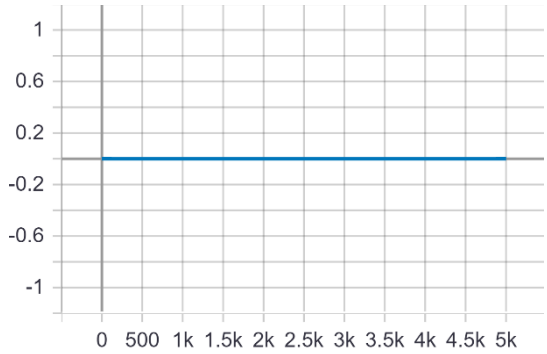

Figure 23: mAP for "Small" Area Objects for "Initial Baseline"

6.2.1.4 Live Webcam Input

No detections were made when using the live webcam feed as input.

6.2.1.5 Test Images Evaluation

Since there were no detections made via webcam input, the attention was turned to the test images to provide visualisation of the models predictions. Upon reviewing the images, it is clear that there were some images that exceed the 0.5 threshold. However, there were no detections being displayed during the live webcam input test most likely due to the detections being too weak to be maintained. If the threshold were to be lowered, perhaps some detections may have appeared. Furthermore, there seems to be a lot of overlap in the test images, especially in example *Figure A2.4.* This is likely due to the lack of training, not allowing the model enough time to generalise the training data as the process was still in its early stages. Additionally, in *Figure A2.3,* an incorrect detection entirely is made as a hood is detected to be a mask. Longer training time should help the model process more data, in turn allowing it to realise and rectify the incorrect detections. See appendix 2 for some examples of predictions performed on test images.

6.2.2 "INITIAL BASELINE PART 2" EXPERIMENT

Following the poor results of the first experiment, this experiment had aimed to take into consideration the possible improvements that could be made as discussed in 6.2.1.3. As indicated by *Figures 20, 21* and *22*, the model would have greatly benefited from longer training as the graphs indicate that the model had not yet converged on its peak score using this dataset. In this experiment, the model is trained from scratch, not continuing from the first experiment, but in this instance it is trained for 6x longer.

6.2.2.1 Experiment Settings

The model for this experiment was trained for 30,000 steps on a batch size of 6. By increasing the amount of training steps, it will result in the model training for longer.

6.2.2.2 Dataset

The same datasets were used from the first experiment. The dataset is described in 6.2.1.2.

6.2.1.3 Results

After reviewing the results of this experiment, it appears there is a clear correlation between longer training and the improvement in performance. The previous experiment trained the model for 5000 steps whereas in this experiment, the model was trained for 30,000 steps using the same dataset. The model for this experiment had completed training in 2hrs 36mins 33secs. The mAP had increased from 0.4584 mAP (46%) to 0.524 (52%), showing a 6% increase due to longer training. However, the mAR has not seen the same improvement, going from 0.4771 (48%) to 0.479 (48%). The mAR saw almost no improvement and neither did the detection of small objects as seen in *Figure 27.* Unlike the first experiment, the mAP and mAR graphs appear to have stabilized as they are no longer sloping upwards but more so, horizontally parallel to the x axis. This indicates that the results have most likely peaked and will not rise much further beyond the point they are at now. Even though longer training has shown improvements to the overall results including the mAP and decreased the training loss (0.3384) and validation loss (0.5154), the mAP metric appears to have settled. Therefore, it is capped at the score of 52 ± 1 %. Which is not a satisfactory result. Longer training will not benefit the model in this instance and score for mAP and mAR will continue to hover around the same numbers. This assumption is based on looking at when the scores had peaked. For example, for the mAP score in *Figure 24,* the score appears to have generally peaked and stabilized at around 15k steps, meaning for the remainder 15k steps, the score did not vary much. This is exactly what would happen with longer training in this instance, the results will not change by much. Furthermore, this model displayed no detections when testing using live webcam input. See Table 2 in appendix 1 for full results.

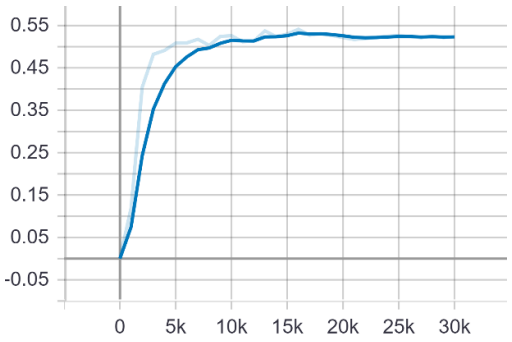

Figure 24: mAP Graph for "Initial Baseline Part 2"

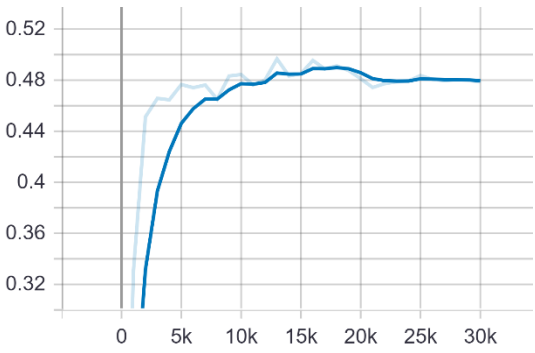

Figure 25: mAR Graph for "Initial Baseline Part 2"

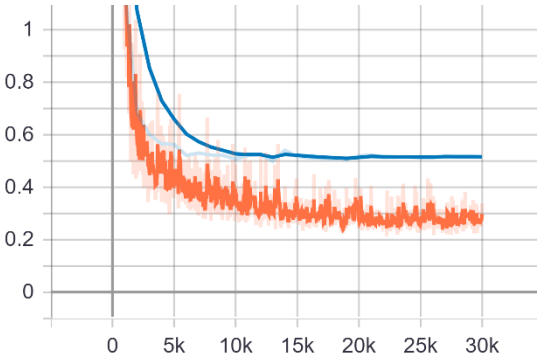

Figure 26: Training/ validation loss for "Initial Baseline Part 2"

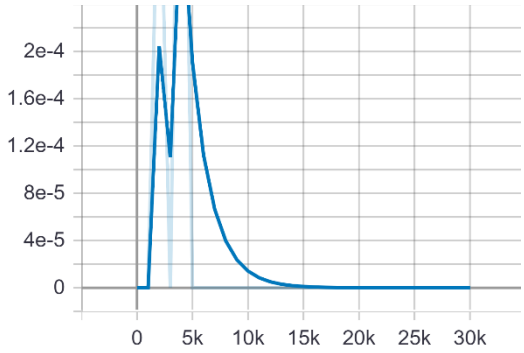

Figure 27: mAP for "Small" Area Objects for "Initial Baseline Part 2"

6.2.2.4 Live Webcam Input

No detections were made when using the live webcam feed as input.

6.2.2.5 Test Images Evaluation

The test images still indicate that the model is struggling to create accurate predictions as there are a lot of overlapping bounding boxes in the images. However, generally, the masks are being localized well, as can be seen by the lime-coloured bounding boxes in appendix 2. The model is performing slightly better at localising the objects than the previous model. However, in Figure A2.7, the people who are wearing a mask in that picture are doing so at an angle which may not have been accounted for in the dataset. Therefore, the inclusion of more samples to account for different angles, and longer training, to help improve the models accuracy and remove the overlapping errors, would benefit the model. The model was still unable to visualise detections despite the mAP being over 50%, this is because the majority of its detections still reside under 50% (0.5 threshold). If there were any detections via webcam, then they were very brief. In theory, it is possible for this model to make detections via webcam input since its mAP was over 50%, however.

6.2.3 "NO MASK ONLY" EXPERIMENT

As mentioned in 6.2.1.3, longer training would not serve any benefit to the model. Therefore, the attention turns to the other major variable, that being the dataset. In this experiment, a new dataset is used but this dataset only contains images representing the no mask class. This is a two-stage experiment that aimed to find out what would occur if the classes were trained individually instead of together at the same time. In this experiment, the no mask class was trained and in the next experiment the face mask class is trained on top of the no mask class to create the final model. The hypothesis for this specific experiment is that the no mask class will do exceptionally well as there is more data introduced, the training time is even longer than the previous experiment and its counterpart class is not present. This removes any potential overlap between the two classes that may impact results and overall performance. *Note: this experiment was not a continuation of the previous one, the model was trained from scratch here just like in the other previous experiments.*

6.2.3.1 Experiment Settings

This model had been trained for 50,000 steps, on a batch size of 6. Since there was significantly more data being used for this experiment than the previous ones, logically it would make sense to increase the training time to allow the model to process all of this data as a shorter training time could replicate the results from the "Initial Baseline" experiment.

6.2.3.2 Dataset

There were 5000 images used in this experiment partitioned 90/10 %, meaning 4500 images went to the training set and the remaining 500 to the test set. A well-known dataset uploaded by Kaggle user greatgamedota called the FFHQ Face Data Set had been used for this experiment[25], which contained images that focused on people's faces. These images were high-quality images with a low file size per image. The dataset contains 70k images which take up only 2GB. This means many of these images could be used without concern regarding memory issues or such. 5000 was the number chosen as each image had to be hand-labelled, a very tedious task which took roughly 9 straight hours. The images in the dataset show people's faces from different angles, which would therefore improve the models ability to make detections at different angle variations.

6.2.3.3 Results

The model completed training in 5hrs 14mins 50secs and produced a significant increase in results across all categories. Firstly, this model was able to visualise predictions unlike the previous iterations as they exceeded the 0.5 threshold. The mAP had increased to 0.608 (60%) which was a major leap from 0.524 (52%). So far, since the first experiment the mAP has increased by 14% by manipulating the hyperparameters and acquiring more and better-quality data. The mAR had increased from 0.479 (48%) to 0.538 (54%), although an increase, 54% is still considered to be on the lower end of the spectrum in terms of satisfactory results. Another major change seen with this experiment was that the mAP metric for small area objects was finally recording detections. In previous versions, the results for this metric had been almost completely absent but after this experiment, despite only achieving a mAP of 0.213 (21%), It was an indication that the model was on the right path. The change/increase in data is most likely responsible for this as better-quality images that represented small area objects were provided to the model for training. Additionally, this was the first time that the mAR for small area objects had also risen above 0. mAP is the main focus of the experiments as it directly refers to the overall accuracy of the model, this is why the mAR, although a very important metric, is not prioritized in discussion as much as mAP. The training loss (0.257) and validation loss (0.4179) also reached a new-low, indicating the model is fitting better to the data as seen in *Figure 30.* See Table 3 in appendix 1 for full results.

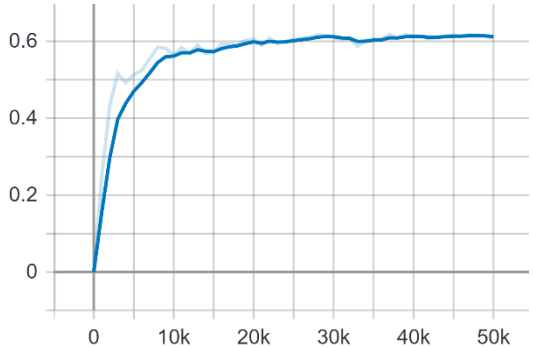

Figure 28: mAP Graph for "No Mask Only"

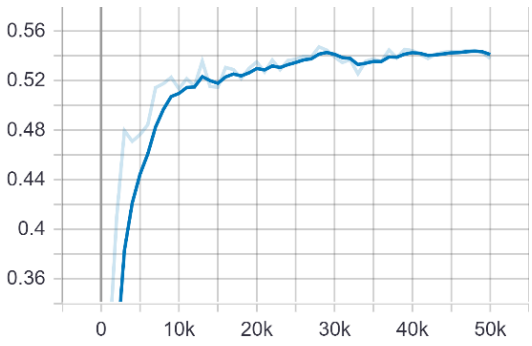

Figure 29: mAR Graph for "No Mask Only"

25 https://www.kaggle.com/greatgamedota/ffhq-face-data-set

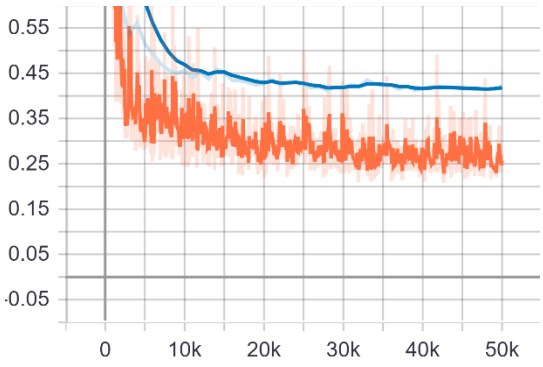

Figure 30: Training/ validation loss for "No Mask Only"

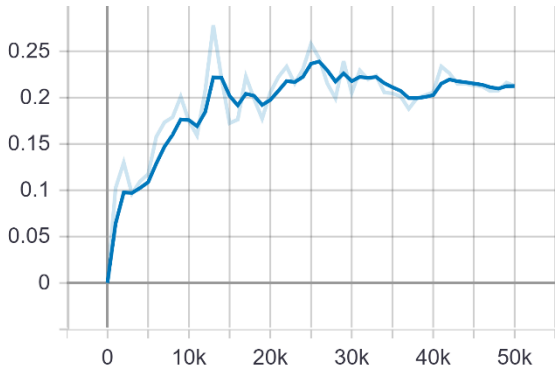

Figure 30: mAP for "Small" Area Objects for "No Mask Only"

6.2.3.4 Live Webcam Input

The model produced from this experiment was the first model able to make and visualise detections in real-time as the accuracy was sufficient enough to pass the 0.5 threshold. *Figures 31, 32* and *33* demonstrate how the model reacts to certain inputs. In *Figure 31*, the model is able to accurately predict that the person is not wearing a mask. This accuracy is maintained over various distances. As displayed in *Figure 32*, where the person is further away from the camera. Thus, making the object to detect, smaller. Despite this, the model is still able to maintain good accuracy (above 80%) similar to *Figure 31* where the person was in a closer position. Furthermore, the labelling approach is being reflected in these predictions as the bounding box is being drawn around the lower facial features such as the nose, lips, chin, etc. However, when the lower half of the face is covered as seen in *Figure 33*, including the facial features previously mentioned, there are no predictions being made. This is due to all the fundamental features of the no mask class no longer being visible. The face mask class is not yet implemented therefore the face mask is not being detected. This means that the model is performing as intended at this stage.

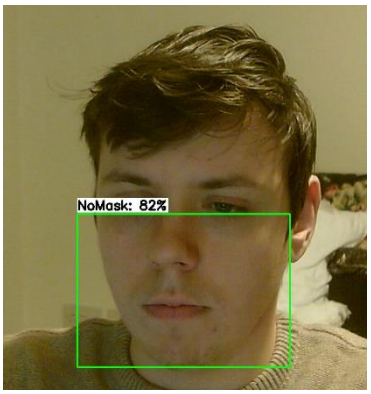

Figure 31: No Mask Close-up

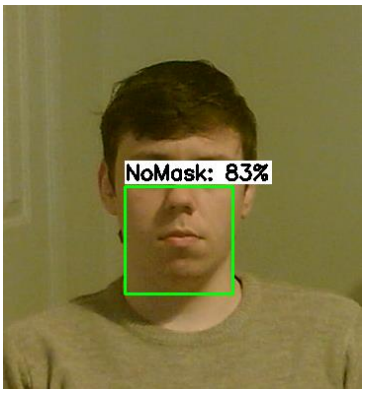

Figure 32: No Mask at Medium Distance

Figure 33: Wearing Mask

6.2.3.5 Test Images Evaluation

The detections made by this model had been very accurate and consistent. This was expected as there the model only had one task and that was to detect whether the no mask class was present in the image. According to the test images, specifically *Figure A2.15*, this model is even capable of detecting different facial expressions as the person in that image is smiling and the person in the *Figures 31 and 32* is not. The prediction images also correspond to their respective ground truth labels very accurately, indicating this model is trained for a significant amount of time. Figure A2.13 also demonstrates the models ability to detect smaller sized objects.

6.2.4 "FACE MASK ADDITION" EXPERIMENT

This experiment aims to build upon the model created in the previous experiment. The previous experiment was the first stage where the aim was to implement the no mask class only. Whereas this experiment is the second and final stage where the face mask class is implemented on top of the already implemented no mask class. Only images representing the face mask class are used in this experiment. After completion of this experiment, a complete model was formed that was able to detect people who are wearing/not wearing a face mask and visualise the detections.

6.2.4.1 Experiment Settings

The model started training from the final checkpoint created by the no mask only model in order to keep the progress made and add more data to the already trained model. The model for the experiment had been trained using exactly the same hyperparameters used in the "No Mask Only" experiment. 50,000 steps and a batch size of 6 was used for training this experiment. It was important to ensure that both classes were trained using the same settings to make the experiments fair and to avoid any imbalances during training, which could potentially encourage a class to perform worse.

6.2.4.2 Dataset

It was important to match the properties of the "No Mask Only" experiment to ensure the two classes are both being represented fairly. Therefore, 5000 images were also used and also divided 90/10, training and testing, respectively. Multiple datasets had been combined to create the dataset used for this experiment. The first dataset by dhruvmak[26] had already been featured in the initial baseline experiment. The second dataset by andrewmvd[27] was a dataset of face mask wearers, full of variation in terms of angles, face mask types and environments. It had received a usability rating of 8.8 and contained 853 image files. However, some of these images were representing the no mask class and also another class known as incorrect mask. These images had been filtered out from the dataset and set aside for possible future use. The third dataset by harshkhandewal[28] contained 1630 images of face mask wearers. However, some of the face masks in certain images were not being worn by a person, therefore these face masks were ignored during the labelling process. Overall, the dataset had a good variation of different types of face masks but some images had to be discarded after review, due to their quality not being satisfactory for training. Some images had contained too much pixelation. The fourth dataset by saurah403[29] used for this experiment had also previously been featured in the initial baseline experiment and contributed a larger quantity of images for this experiment. Any weak samples had also been filtered out. The fifth and final dataset by aditya276[30] contained 1844 images of face mask wearers, the images were pre-labelled but using the YOLO format. Unfortunately, this format was incompatible with this experiment and model, therefore the labels had to be discarded and relabelled using the Pascal VOC format. This dataset offered images with multiple face mask wearers in a single image. Specifically, the people in these images were far away from the camera. This means the face masks they are wearing would inherently also be further away or in other words, smaller. Meaning that they would qualify as the medium or small area objects. On this account, the hypothesis for the outcome of this experiment is that the mAP for small area objects would see an increase in score, as the model would have more examples to learn from. Lastly, as mentioned in 5.2.6, there had been issues with some XML files becoming corrupted, consequently causing the loss of several samples. This was especially

26 https://www.kaggle.com/dhruvmak/face-mask-detection

27 https://www.kaggle.com/andrewmvd/face-mask-detection

28 https://www.kaggle.com/harshkhandewal/faces-with-masks

29 https://www.kaggle.com/saurah403/face-mask-detectionimages-with-yolo-format

30 https://www.kaggle.com/aditya276/face-mask-dataset-yolo-format

prominent at this stage. 125 images overall had been lost from the face mask category. The response to the assumption made in 5.2.6 seemed to have rectified this issue but whether this was the root cause was still unclear. Despite losing these images, new images had been acquired and labelled to replace the lost ones. Then the newly labelled images had been inserted into the dataset to ensure that the dataset contained 5000 images again.

6.2.4.3 Results

The model had completed training in 6hrs 15mins 2secs. The mAP had remained at a similar level as it was when the first stage completed training, rising to 0.628 (63%). However, before conducting this experiment, the hypothesis for the outcome was that the overall mAP would decrease. This was assumed since a new class would be introduced which would result in the model needing to perform binary-class object detection as opposed to singular-class object detection, which was a lot simpler. The increase in mAP for small area objects was also another incorrect hypothesis, as the mAP for small area objects actually dropped down to 0.161 (16%) from 0.213 (21%) since the last experiment. This may have been due to too many variations in categories such as angles, appearance and areas etc. with not enough examples to effectively represent all these different categories during training, hence the lack of predictions for this metric. Additionally, the mAR increased by 1% to 5%. However, the graphs for mAP (*Figure 34*) and mAR (*Figure 35*) appear to still be moving in a slight positive velocity while decelerating. Similarly, to what was seen in the initial baseline experiment, the model may have benefited from more training. However, unlike the initial baseline experiment, this was not required as urgently and other areas such as the data could be investigated also. Lastly, the loss scores for training (0.368) and validation (0.406) as depicted in Figure 36, had not converged very low. Indicating the data is not fitting as well as it could be. The assumption here is that the diversity of data is too large, while containing insufficient amounts of data to represent each variation confidently. See Table 4 in appendix 1 for the full table of results.

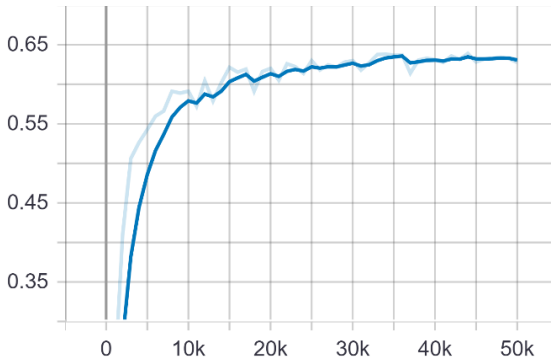

Figure 34: mAP Graph for "Face Mask Addition"

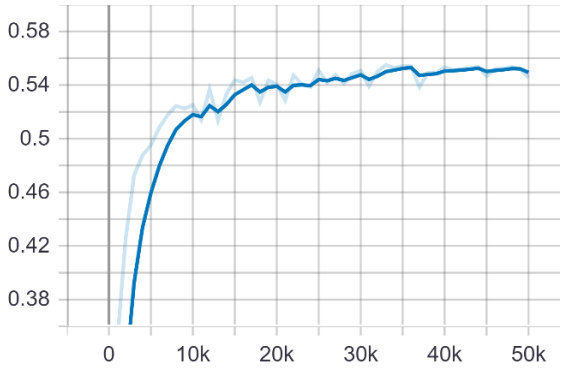

Figure 35: mAR Graph for "Face Mask Addition"

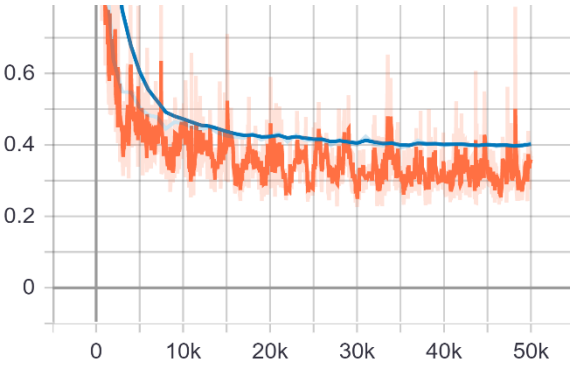

Figure 36: Training/ validation loss for "Face Mask Addition"

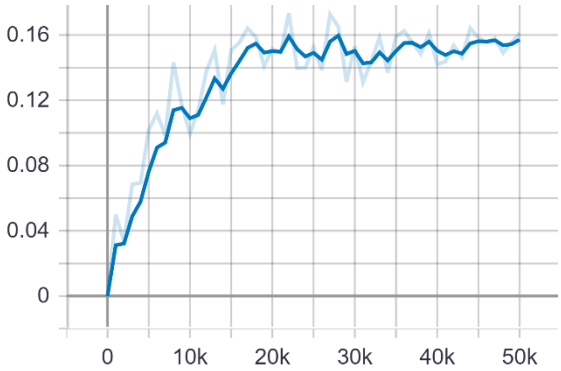

Figure 37: mAP for "Small" Area Objects for "Face Mask Addition"

6 2.4.4 Live Webcam Input

The model received input via live webcam feed to examine how it would deal with real-time detections when there is an additional class present. A key area of interest to examine was whether the bounding boxes generated for each class would overlap at any point. This would be problematic as it would indicate that the model needs more training with additional data to assist in further realising the difference between the two classes. Specifically in various environments or particular instances where it may have struggled to make this distinction. On that note, *Figure 41* shows that the model often predicts a mask is being worn if a person is wearing glasses. This may be only a one-off mishap. Nevertheless, it is something that has happened once meaning it is possible it can occur again in another situation and therefore is an issue that needs to be addressed. Whereas the predictions made when the face mask being worn are very accurate. *Figure 38* demonstrates a standard straight-forward angle while *Figure 40* demonstrates a different, yet common angle that a face mask can be seen from. *Figure 39* demonstrates that the model is still able to make predictions despite the face mask being occluded by the hand. Overall, a mostly satisfactory result, the quality of the no mask class remains just as accurate as described in 6.2.3, this experiment mainly focused on the performance of the face mask class when merged into a model that had already been trained to detect a similar object.

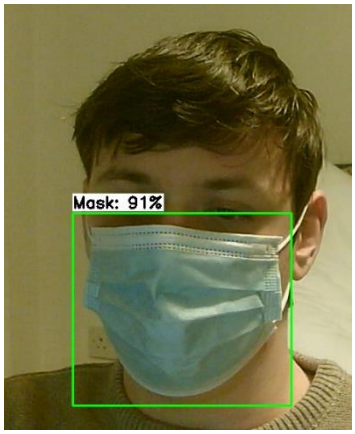

Figure 38: Mask on

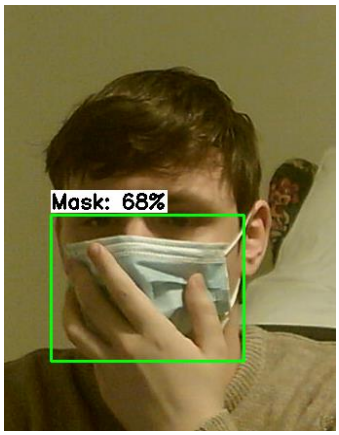

Figure 39: Mask Covered

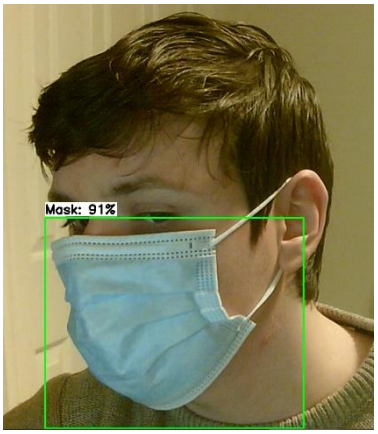

Figure 40: Different Angle Mask

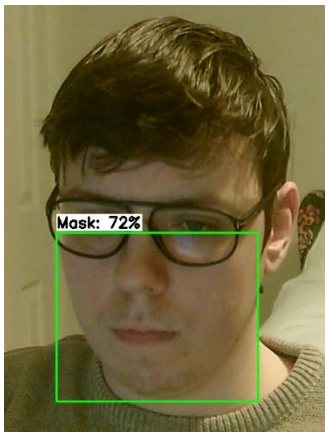

Figure 41: Glasses Mistaken for Mask

6.2.4.5 Test Images Evaluation

With the addition of the face mask class in conjunction with the previous model, there had been some issues that have come to light in relation to the no mask class as described in 6 2.4.4. However, in terms of the face mask class, the test images indicate the model is accurate and versatile as it is able to detect a blue mask and a black mask in *Figures A2.19* and *A2.23*, respectively. However, Figure A2.21 indicates there is still room for improvement as no detections were made despite two ground truth labels being registered in its corresponding ground truth image (Figure A2.22). Overall, this model requires more data for both classes as there seems to be an issue with people wearing glasses when not wearing a mask, as well as missed face mask detections.

6.2.5 "GLASSES" EXPERIMENT

Overall, the implementation of the face mask addition experiment was successful for the most part. However, as mentioned 6.2.4.4, an issue had been discovered when testing the completed model using a live webcam feed as input. This issue can be seen in *Figure 41* where the person is not wearing a face mask but is wearing glasses. This issue arises after the face mask class had been implemented, the issue was not present before in the no mask only experiment and could not have been present since the face mask class was not implemented then. This issue would not have been recognised if the model were not tested using a webcam. Although mAP provides a comprehensive outlook on the statistical performance of a model, testing the model "in-field", using a more practical approach can reveal hidden errors that mAP could not. In order to rectify this issue, we must first begin to understand why it had happened. The likely cause of this issue resides in the data which represents the no mask class. The assumption is that there was a lack of data representing the people who wore glasses but not a face mask. It is a variation that may not have had many examples representing it. Therefore, including more data of people who are not wearing face masks but are wearing glasses would be enough to resolve this issue. The hypothesis was that the issue would be resolved and the mAP would increase slightly as more data is being added and the model is being trained for longer. Especially since this was something that was identified in 6.2.4.3 to possibly better the results of the model.

6.2.5.1 Experiment Settings

The model had continued training from the final checkpoint generated during the face mask addition experiment. More data had been added and the model was trained for 50,000 steps on a batch size of 6.

6.2.5.2 Dataset

2000 new images were introduced, 1000 for each class. This new data had been split 90/10, with 1800 going to training and 200 going to testing. The dataset used to represent the no mask class was focused mainly on individuals who wore glasses, for the reasons discussed in 6.2.5. This dataset by the Kaggle user coffeeshop[31], contained a vast dataset with a plethora of images some which were not relevant to the objective of this experiment and were therefore ignored. Only the images featuring people who were wearing glasses, without a face mask, were used. Secondly, there was an equal amount of images used to represent the face mask wearers in this experiment to avoid any data imbalance as this could create even more issues down the line. The dataset by sumansid[32] contributed 323 images to the face mask class, while the dataset by Kaggle user wobotintelligence[33] was a vast dataset which filled in the remaining amount of images needed (677). This dataset contained 10.4k images, after review of the dataset, it was noticeable that there were some images present in this dataset that had already been used in the face mask addition experiment. These duplicate images were hand-removed from the dataset during the labelling process, although a couple of duplicates may have gotten through as recalling similar images from memory was not a reliable strategy here. Nevertheless, a couple of duplicates should not cause any harm to the models performance.

6.2.5.3 Results

The main objective of this experiment was to provide the model data of people who were wearing glasses and representing the no mask class, in hopes of solving the issue that had come to light when testing the output of the 6.2.4 "Face Mask Addition" experiment. This issue is discussed in 6.2.5.4,

31 https://www.kaggle.com/coffeeshop/people-in-dioptric-glasses

32 https://www.kaggle.com/sumansid/facemask-dataset

33 https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset

where the model is once again tested using live input via webcam stream. In this section, the results of the model are interpreted using COCO metrics. The model had completed training in 6hrs 10mins 56secs and the mAP decreased slightly from 0.628 (63%) to 0.618 (62%). This is not a major loss; however, it is not progress either. The mAR had also decreased by 1%, back to 54%. However, a metric that did see some progress after this experiment was the mAP for small area objects, rising from 0.161 (16%) to 0.197 (20%). The validation loss (0.405) stayed roughly the same, whereas the training loss had risen to 0.413. This was surprising as more data and longer training had worked in the past to increase the performance of the model. However, after this experiment, that was no longer the case. An assumption had been made after this experiment as to why this happened. The overall model at this stage had been trained using 10800 images and tested using 1200. The mAP score had still not achieved the 70% accuracy mentioned in project goal 1.3.2.8. The quality of the data came into question as after review of the dataset, there had been some samples that may not have been suitable after all. This refers to the quality and resolution of some images not meeting the expectations of what a good sample is. Additionally, adding more data and expecting it to improve the model may not work since new data may be representing a variant that already has a substantial amount of samples. Therefore, adding more samples would not make the highly impactful difference that was originally anticipated since the samples for that variant would just stack and not provide any further value. However, if poor data was in fact used, then there foundational learning has already been done by the model. Meaning this could not be reversed, which would inherently provide poorer results as the good data could be mixing with bad quality data. Therefore, the quality of data being inputted needs to be reviewed more closely and furthermore, it may be best to avoid inputting mass batches of data at once. It may be a better approach to distribute the data in smaller sizes, for example 2000 per iteration when training the model. Instead of transitioning from 500 images to 5000 for a certain class, like the initial baseline 2 and no mask only experiments did. However, this is simply an approach which is being considered to more closely monitor how the model reacts to data and not a guarantee of increasing performance. See Table 5 in appendix 1 for full results.

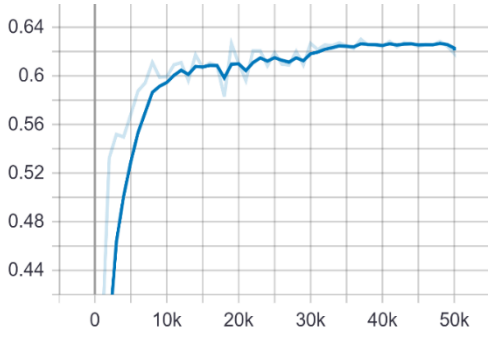

Figure 42: mAP Graph for "Glasses"

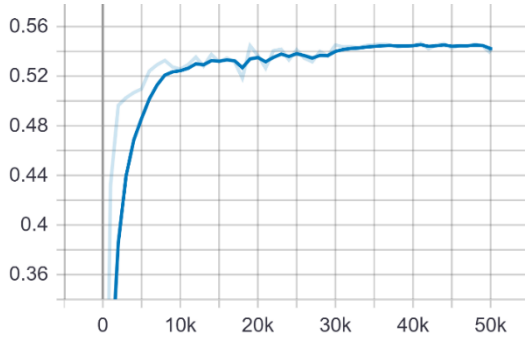

Figure 43: mAR Graph for "Glasses"

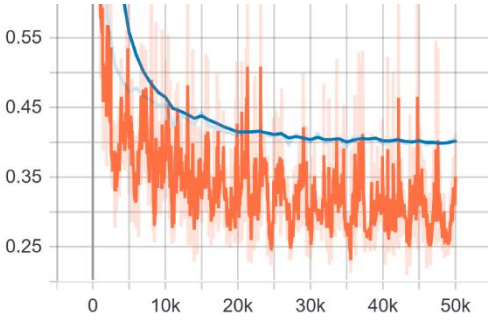

Figure 44: Training/ validation loss for "Glasses"

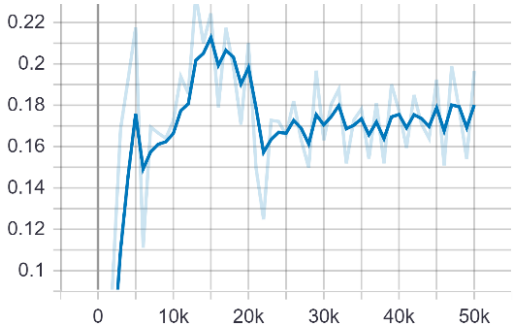

Figure 45: mAP for "Small" Area Objects for "Glasses"

6.2.5.4 Live Webcam Input

As previously mentioned, the main objective of this experiment had been to solve the issue of a person who is not wearing a face mask being identified as a face mask wearer if the person had been wearing glasses, as seen in *Figure 41*. This issue had originally been found when testing the model using live input via webcam. The approach taken to resolve this issue included using images of people who were wearing glasses, but not a mask, and feeding it to the model. After the experiment was conducted, the model was once again tested using live input via webcam and it appears the problem had ceased to exist. Whenever a person wears glasses without a mask, the model does not miscategorise this person into the face mask class anymore, as demonstrated in *Figure 46*. This means that the approach used to solve this issue had been successful, it was mainly due to the model being provided relevant samples that target this specific instance and allowed the model to learn to better distinguish the two classes when glasses are being worn. In addition to this, glasses were also worn with a face mask as seen in *Figure 47* and there had been no issues found with this. Lastly, *Figure 48* demonstrates the behaviour of the model when presented with a face mask that is not being worn by a person. Correctly, this model did not detect the face mask at all but instead the features of the no mask class and categorized them as no mask. This can be seen in *Figure 48,* as the person is holding the face mask and the model does not detect the face mask but instead detects the person is not wearing one. Due to this, project goal 1.3.2.2 is met as it states: the final model must be able to detect the "Face Mask" object only if a face mask is being worn by a person.

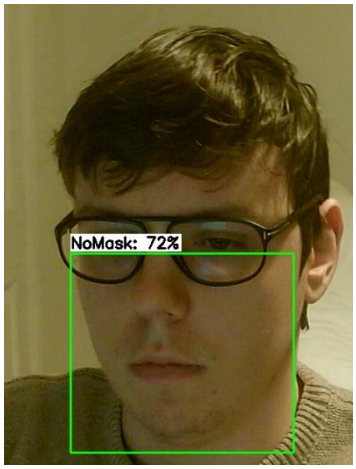

Figure 46: Glasses + No Mask Solved

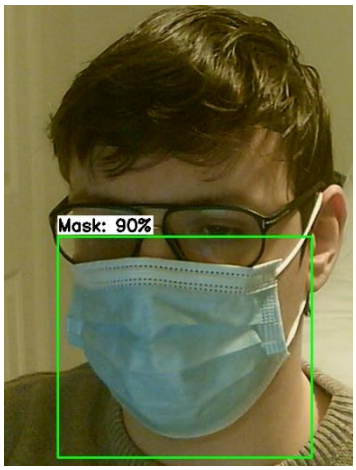

Figure 47: Glasses + Face Mask

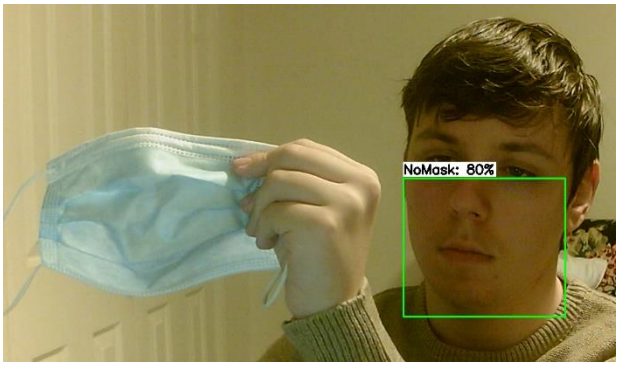

Figure 48: No Mask + Face Mask NOT Being Worn

6.2.5.5 Test Images Evaluation

This model shows clear improvement, not only via webcam input, but also when reviewing the test images. *Figure A2.29* was the same image used in the initial baseline experiment (*Figure A2.4*). The detections on that image were not accurate and there had been multiple overlapping bounding boxes. However, this model was able to correctly categorise that image without overlapping and make an accurate prediction. *Figure A2.27* contained a person wearing a mask with an unorthodox appearance; however, the model was still able to detect and categorise the fae mask in this image correctly with great correspondence to its ground truth label in *Figure A2.28*. Lastly, Figure A2.25 contains a person wearing a mask at a side-angle. The model is able to correctly detect and categorise this too. Not to mention, all three people in all three test image examples discussed here and presented in appendix 2, are wearing different coloured masks. This is evidence that the model is able to detect the presence of the face mask no matter its appearance, or the appearance of the person wearing it.

6.2.6 "INCORRECT MASK" EXPERIMENT

Due to the previous experiments, a stable and functional model with minimal errors had been created. The purpose of this experiment was to experiment with the current model instead of attempting to improve it. A new class had been introduced called "Incorrect Mask" and this class was supposed to represent the notion of a person wearing the face mask incorrectly or "half wearing" it. This typically means the parts of the lower face are exposed that should not be, such as nose, mouth, teeth, etc. See *Figures 49, 50 & 51* for an example. There was uncertainty about implementing this class as there will most certainly be overlap between the three classes. The features of the no mask class (lower part of the face) will become partially exposed, additionally the face mask will still be worn by the person. The incorrect mask class would become a middle ground between these two classes. The accuracy of the incorrect mask would most likely be the lowest out of the three classes due to overlap and the class being so similar to the other classes already present.

Figure 49: Incorrect Mask; Below Nose

Figure 50: Incorrect Mask; Mouth & Nose Exposed

Figure 51: Mask on Chin

6.2.6.1 Experiment Settings

This model had been further trained using 50,000 steps and a batch size of 8. Additionally, the activation function had been changed from sigmoid to softmax and the loss function changed to the categorical cross-entropy loss function. As stated in chapter 3, the categorical cross-entropy loss function and the softmax activation function are both best used for multi-class classification. As the model was aiming to detect more than two classes, it was appropriate to make these changes for this experiment. The model had continued training from the final checkpoint generated during the previous experiment.

6.2.6.2 Dataset

1600 images of people wearing face masks incorrectly were added to the dataset for this experiment, with a split of 90/10 between training and testing, respectively. Two datasets had been mixed together to represent the incorrect mask class during training. The first dataset used had contained roughly 400 images. These images were collected from various other datasets, for example as previously mentioned in 6.2.4.2, where it was said that images representing the incorrect mask class were set aside for future use. Most of these samples came from the dataset by andrewmvd[34], however some images were found via Google. These remaining images came from a stock photo website, where certain images had been hand-picked to be used in this experiment[35].

34 https://www.kaggle.com/andrewmvd/face-mask-detection

35 https://www.istockphoto.com/photos/wearing-mask-wrong

The second and main dataset used [36] was introduced in a research paper by Cabani et al. (2021) called MaskedFace-Net[37]. The authors had taken an existing dataset known as the FFHQ Face Data Set, which was previously used in the no mask only experiment, and they artificially applied face masks to this dataset. The original FFHQ Face Data Set was a collection of people's faces, or in terms of the project, people not wearing face masks. *Figure 52* shows an image from the FFHQ Face Data Set and *Figure 53* shows how that same image appears when the face mask is artificially applied. Additionally, *Figures 49, 50 & 51* are also from this dataset.

Figure 52: FFHQ Dataset Image – No Mask

Figure 53: MaskedFace-Net Image – Mask

6.2.6.3 Results

The model had completed training in 9hrs 51mins 18secs. Despite this experiment being outside the original scope of the project, it was still important to understand how the model would react to a new class being implemented in case it was to occur in the future. Furthermore, the addition of the "Incorrect Mask" class is a viable and acceptable addition as it also directly correlates to the problem of people not wearing face masks and spreading the virus. Wearing a face mask incorrectly severely limits the protection a face mask can provide. The mAP of the model produced had roughly stayed around the same score it has been at for the past 2 experiments, achieving 0.627 (63%). It was expected for the mAP to slightly increase due to another class being introduced which, in theory, makes more room for growth statistically. On the other hand, there was a chance that the mAP would also decrease as the previous knowledge of the model could interfere with the incorrect mask class since all three classes are so closely related. The mAR had risen to 0.612 (61%), this is due to the new class being introduced as there are more objects that the model is able to detect now. According to the link between the mAP and mAR here, it seems that the model is making more predictions but they are not very accurate since the mAP did not rise accordingly with the mAR. This is an interpretation that will require further analysis to confirm, however.

The mAP for small area objects has been heavily focused on as it can provide an understanding of the range the model has when compared with other metrics such as mAP or mAP (Large and/or Medium). These metrics also indicate that the model(s) are able to detect objects of different sizes in terms of area. Due to this, project goal 1.3.2.7 is met as it states that the final model must be able to detect objects no matter the size (big to small). The mAP for small area objects for this model however is very poor, only achieving 0.124 (12%). This may possibly be due to the lack of samples that are small in area which represent the incorrect mask class. However, the

36 https://github.com/cabani/MaskedFace-Net

37 https://github.com/cabani/MaskedFace-Net

solution to increasing the mAP of small area objects may not be as simple as adding more data to represent that category. This was explored in the face mask addition experiment, where it stated in 6.2.4.2 that including more data to represent the small area objects may work, but it did not, as discussed in the 6.2.4.3 results section.

The validation loss (0.429) and the training loss (0.303) continue to maintain approximately the same values as achieved in the previous two experiments. With the validation loss rising slightly, as expected since a new class has been added which may cause difficulty for the model to distinguish the new class from the others due to their similarity. Furthermore, there are over 5000 images for both the face mask and no mask classes at this stage, meanwhile the incorrect class only has 1600. This is a vast data imbalance which most certainly will be held responsible if the class performs poorly during other phases of testing such as webcam and test images. More data must be added in the future to remove this imbalance as it most certainly has a large impact on the results. See Table 6 in appendix 1 for the full table of results.

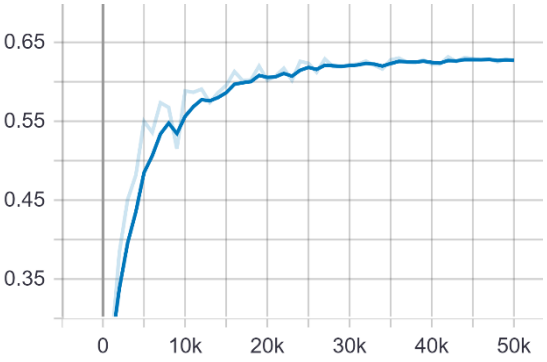

Figure 54: mAP Graph for "Incorrect Mask"

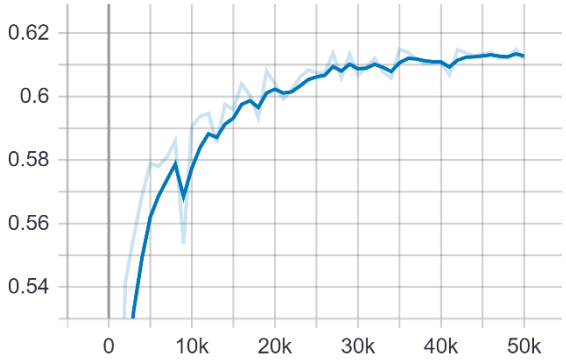

Figure 55: mAR Graph for "Incorrect Mask"

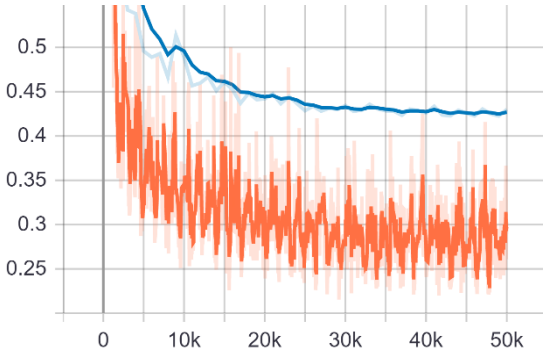

Figure 56: Training/Validation Loss for "Incorrect Mask"

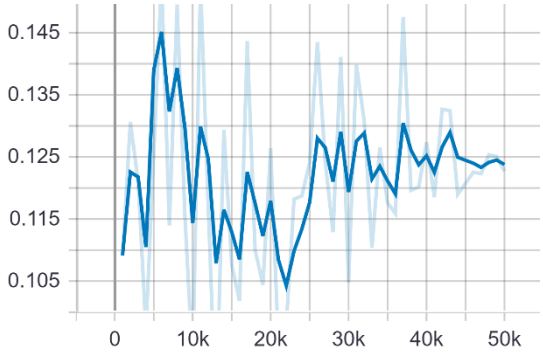

Figure 57: mAP for "Small" Area Objects for "Incorrect Mask"

6.2.6.4 Live Webcam Input

The incorrect mask class has seen partial success when testing the model using live input via webcam. *Figure 58* demonstrates that the incorrect mask class can indeed make accurate predictions in its current state. However, these predictions are unreliable at the moment as seen in *Figure 59* and *Figure 60. Figure 59* shows how the incorrect mask class overlaps with another class. *Figure 60* shows how the incorrect mask is disregarded when the face mask is on the neck of the person, revealing the features of the no mask class. This indicates that the model needs more samples to aid it in detecting the incorrect mask in different positions. Overall, the incorrect mask class is able to make detections, typically only when the person is closer to the camera rather than further away. In a lot of instances, the incorrect mask will overlap with other classes and not make detections when it should. More training data is needed to improve this class as there was a

significant data imbalance which obviously has impacted negatively on the performance of this class in real-time.

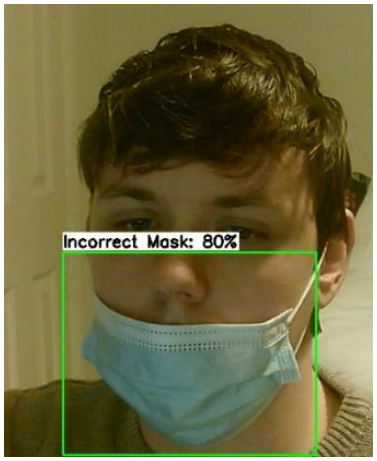

Figure 58: Incorrect Mask; Accurate Prediction

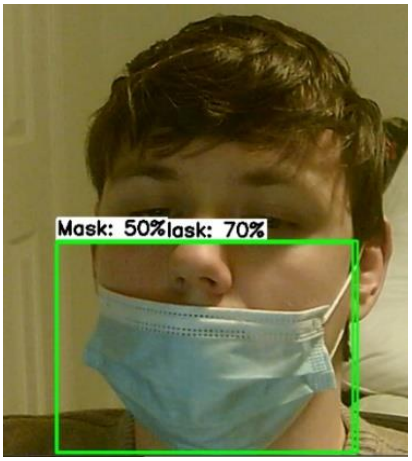

Figure 59: Incorrect Mask; Overlap

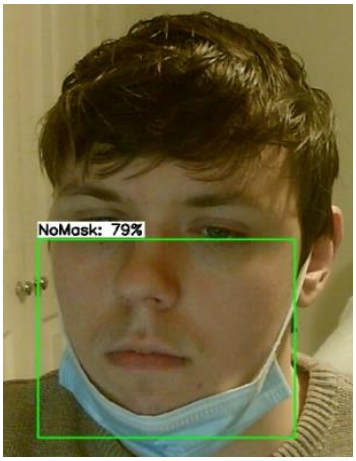

Figure 60: Incorrect Mask; Incorrect Prediction

6.2.6.5 Test Images Evaluation

Despite the vast imbalance of data samples between the three classes, the incorrect mask model had still been able to make some accurate predictions. It was a lot more difficult for the model to predict the presence of the incorrect mask class in an image as firstly, it is the middle ground between two other classes already implemented and may overlap. Secondly, unlike the face mask and no mask class, the incorrect mask class has more than one position where the detection would be valid. For example, when the nose is showing while the face mask is on; that should be detected as an incorrect mask. Also, when the mouth and teeth are showing when the face mask is on; this should also be detected as an incorrect mask.

In terms of the test images, the incorrect mask had been able to detect three different positions correctly. These positions included the nose and lips showing while the face mask was worn (*Figure A2.31*), only the nose showing while the face mask was worn (*Figure A2.33*), and finally the nose, lips and teeth showing while the face mask was worn (*Figure A2.35*). In other examples not included in appendix 2, there had been overlap between the face mask and the incorrect mask class as the model would still detect the mask when it was being half-worn. Overall, these are promising results and infer that with more image data and training, the model would begin making predictions belonging to this class a lot more sharply.

6.2.7 "DATASET REWORK" EXPERIMENT

As mentioned previously in 6.2.5.3, the quality of the data as well as how it was fed to the model during training, could have been a factor responsible for its overall performance and lack of progress when new data was introduced. For this experiment, the old model had been set aside and a new and smaller dataset had been put together that contained better quality images. The model had been trained from scratch using this dataset. The aim with this experiment was to closely monitor how the model will react to receiving less data which is high quality. Essentially the model is trained from scratch, with each iteration to come aiming to improve the dataset overall which in turn would improve the performance of the model.

6.2.7.1 Experiment Settings

This model was trained from scratch for 50,000 steps on a batch size of 6. As this experiment is only training the face mask and no mask classes and not the incorrect mask class, the activation function used is set to sigmoid and the loss function is set to binary cross-entropy.

6.2.7.2 Dataset

1500 images were used to represent the no mask class and another 1500 were also used to represent the face mask class with a 90/10 split. All 1500 images for the no mask class came from the FFHQ Face Data Set[38] and likewise the MaskedFace-Net Dataset[39] was used for the face mask class. Both of these datasets have been used and discussed previously in previous experiments. These datasets provide quality examples for the model to use during training. The idea was to label 1500 images from the FFHQ Face Data Set and then label the exact 1500 images from the MaskedFace-Net dataset that correspond to the 1500 labelled from the FFHQ Face Data Set. An example of this can be seen earlier in *Figures 52 & 53.*

6.2.7.3 Results

The model completed training in 5hrs 43mins 23secs. This model had produced the highest mAP result from all of the models discussed thus far. The overall mAP was 0.830 (83%), however all was not as it seemed. After closer examination of the other mAP metrics, it appears that the medium/small metrics for both the mAP and the mAR are shown as -1.000. This indicates an error, meaning no medium or small area objects had been registered at all during training. *Figure 64* represents these metrics in a single as they all had produced the same graph. In the absence of the medium and small area objects, this means that only the large area objects are represented by the primary metric which is the mAP as seen in *Figure 61*. Additionally, since mAR is missing these metrics too, the mAR also only represents the large area, depicted in *Figure 62.* Unfortunately, the dataset which was supposed to set the model in the right direction, is the variable in the experiment that is causing the failure. The dataset had only contained large area objects, hence why no small area objects were recorded. This does not mean that the datasets used in the experiments should be discarded, it simply means more data needs to be added that target medium/small areas. The validation loss (0.232) and the training loss (0.201) are also very low, taking this into consideration along with the score of the mAP @.75IOU being 0.995, there are signs of overfitting. This could be possible as the data used for both classes and in general are very similar to each other. A possible fix would be to diversify the data, including more data from other different datasets. Doing this would also account for the medium/small objects too. See Table 7 in appendix 1 for full results.

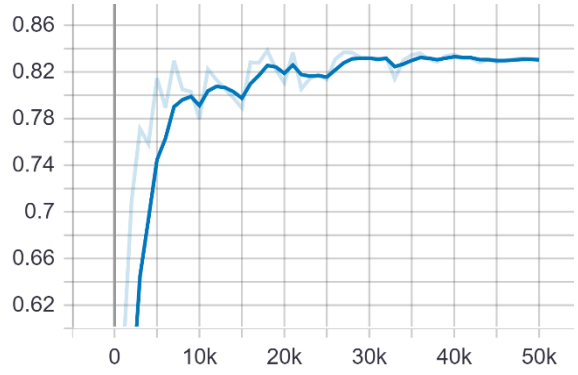

Figure 61: mAP Graph for "Dataset Rework"

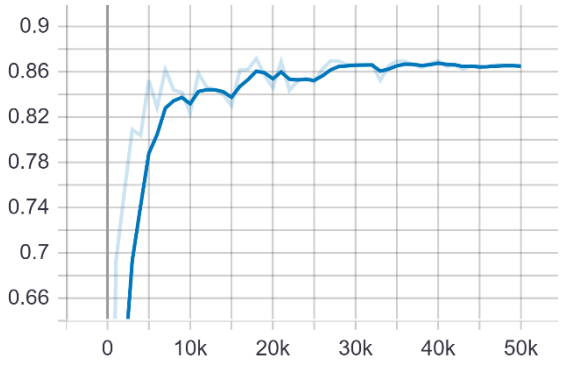

Figure 62: mAR Graph for "Dataset Rework"

38 https://www.kaggle.com/greatgamedota/ffhq-face-data-set

39 https://github.com/cabani/MaskedFace-Net

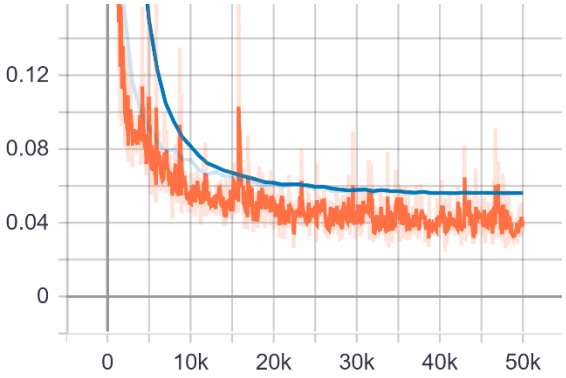

Figure 63: : Training/ Validation Loss for "Dataset Rework"

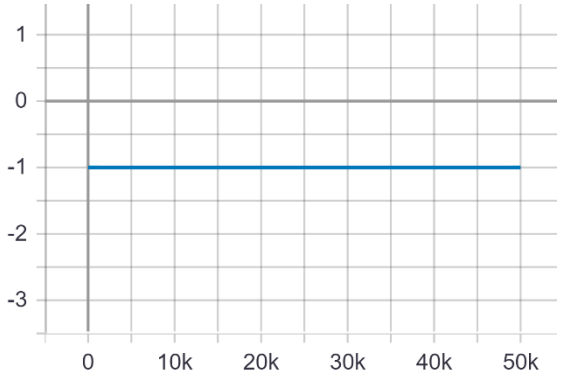

Figure 64: mAP/mAR - Small/Medium Area Objects for "Dataset Rework"

6.2.7.4 Live Webcam Input

When the model had been tested using live webcam input, it was able to make detections only if the person had been positioned very close to the camera, roughly 10 to 20cm away from the lens as seen in *Figure 65*. Otherwise, at greater distances the model was not able to make any predictions at all. This equally applies to both the face mask and no mask classes, *Figures 65* and *66* represent the models performance for both classes. This was due to the model not being trained to detect objects that have a medium/small surface area. As the person gets further away from the camera the object will eventually fall into the medium and small area categories thus, undetectable by this model.

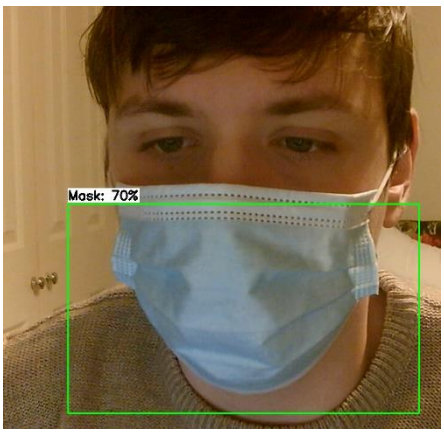

Figure 65: 12cm Away from Camera

Figure 66: 30cm Away from Camera

6.2.7.5 Test Images Evaluation

The performance of the face mask class needed close examination as the images representing this class for this experiment, had been using artificial face masks. According to the predictions visualised on the test images, all artificial face masks from all images had been detected with very high accuracy. However, this is most likely due to overfitting as mentioned earlier, as well as the major similarities each sample image has with one another. While the detections on sample images are very accurate as seen in appendix 2, the next model needs to contain more diversity within the dataset to allow for other types of face masks, object sizes, etc. to be measured and evaluated.

6.2.8 "TRAINING MEDIUM AND SMALL AREA OBJECTS" EXPERIMENT

This experiment aims to build off the model produced by the previous experiment. The previous model had unintentionally only been trained to detect large area objects, leaving medium and small area objects completely unaccounted for. This experiment aims to improve on that model by adding data which represents both the medium and small areas of both the face mask and no mask classes. The expected outcome is that this model will be able to make detections no matter how large or small the object is.

6.2.8.1 Experiment Settings

The model had been trained for 50,000 steps on a batch size of 6.

6.2.8.2 Dataset

An additional 2000 images had been added, all focused more on objects which have a medium and/or small surface area. With the addition of these images, it was expected that they would provide the necessary samples to the model during training to give it the ability to output detections of objects that contain a medium and/or small surface area. 1000 images represent the face mask class and the other 1000 represent the no mask class. The images were divided 90/10 as usual. Image augmentation had also been introduced in this experiment to improve the dataset and provide a bigger diversity of samples for the model to use during the training process. Image augmentation is discussed in chapter 2 section 2.4.2, where it states that image augmentation has been proven to help reduce overfitting, which was a potential issue that was becoming prominent in the dataset rework experiment as mentioned in 6.2.7.3 . It is also known to create diversity in terms of viewpoints and angles. Although, the same section in chapter 2 also states that by rotating an image, this could potentially turn an image with a medium sized object into a large sized one, this would only be the case if the original image wishes to stay in-bounds (not have black spaces around it). The image augmentation implemented for this experiment was done 100% manually by-hand. This was mainly to ensure that the objects (face masks or no masks) in all images being augmented did not become large area objects as this would defeat the purpose of this experiment.

250 out of the 1000 images for each class, had been collected via Google Images. These same 250 images were the images that were augmented. Therefore, creating an extra 250 images. This way, each class already had 500 images and the other 500 for each class were taken from previous datasets. The remaining 500 for the face mask class was taken from the dataset by andrewmvd[40] which had been used many times during development and was a reliable dataset to use. The remaining 500 for the no mask class was taken from the dataset by coffeeshop[41]. This dataset was the same dataset that was used to resolve the glasses issue previously in experiment 6.2.5. Assuming this experiment is successful in restoring the medium and small area objects, this dataset would theoretically prevent the glasses issue from arising again. Additionally, it had also contained many samples of people without masks who were distanced away from the camera. Lastly, *Figures 67* and *68* show an example of what an augmented image looks like from the ones that have been used in this experiment. *Figure 68* was augmented using Windows Photo Editor.

40 https://www.kaggle.com/andrewmvd/face-mask-detection

41 https://www.kaggle.com/coffeeshop/people-in-dioptric-glasses

Figure 67: Original Image – No Mask

Figure 68: Augmented Image – No Mask

6.2.8.3 Results

The model had completed training in 10hr 5mins 35secs. The mAP had dropped to 0.658 (66%) as seen in *Figure 69*, which seems like a more appropriate value. The metrics for medium and small area objects had risen above 0 for both mAP and mAR metrics, therefore this time they are being detected. This means the aim of the experiment has been met. The mAP had dropped significantly now that the medium and small areas are being factored into the equation. This is the highest mAP achieved by a fully functional model but examining the detections via webcam and using test images will reveal whether the model lives up to the results produced using COCO metrics. The mAP for medium areas was 0.467 (47%), rising by 48% since the last experiment and for small areas it was 0.301 (30%), rising by 31%. This indicates that the newly added data had caused significant improvements to the model. Although both medium and small area metrics are still considered to be low, the model should still be able to make detections no matter the object size as indicated by the mAP metrics. *Figure 72* indicates that the small area graph has been stabilizing over time whereas the medium areas graph in *Figure 71* seems to be fluctuating a lot more, possibly indicating that more training + more data is needed. It was unclear of the impact that image augmentation had here but it was more likely that the impact was positive rather than negative as image augmentation had provided more diversity within the dataset, allowing the model to learn about new angles and variations. The training loss (0.209) and the validation loss (0.326) are fairly low, suggesting most of the data is fitting well during training and validation. Future work after this experiment would include adding even more images to represent medium and small areas. This could further improve the results. See Table 8 in appendix 1 for full results.

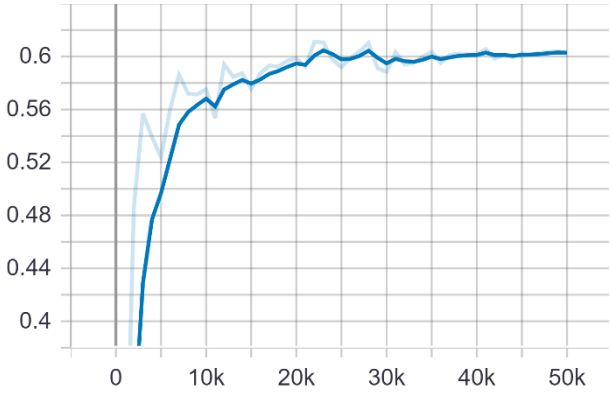

Figure 69: mAP Graph for "Training Medium and Small Area Objects"

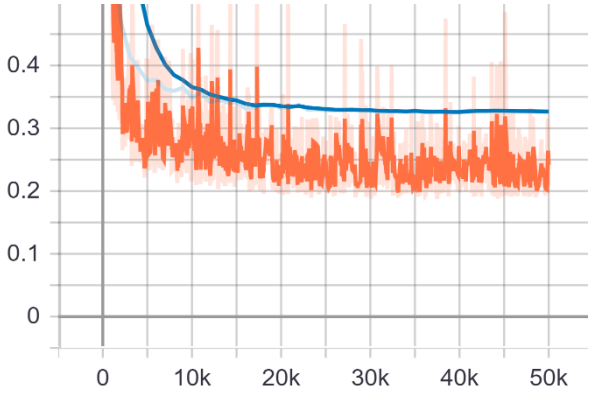

Figure 70: Training/Validation Loss for "Training Medium and Small Area Objects"

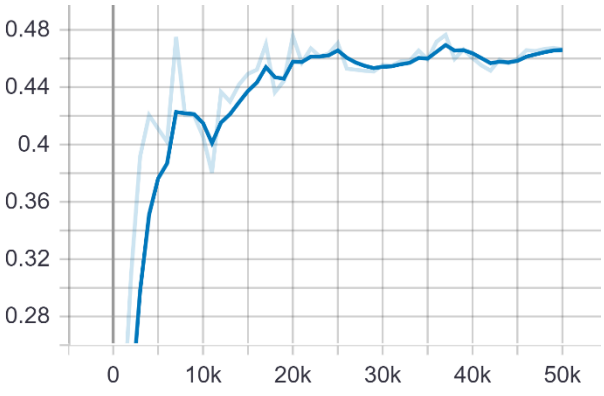

*Figure 71: mAP **medium** area for ""Training Medium and Small Area Objects"*

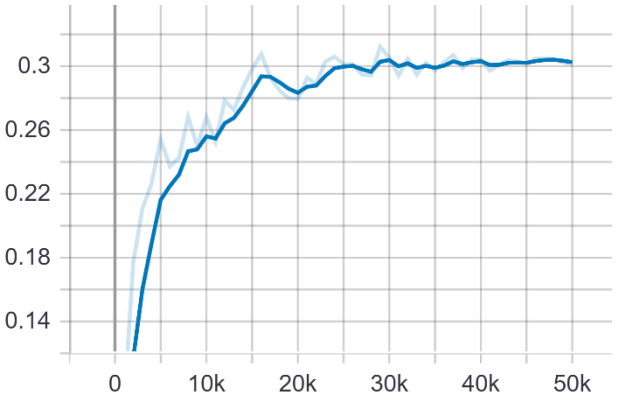

*Figure 72: mAP **small** area for ""Training Medium and Small Area Objects"*

6.2.8.4 Live Webcam Input

Despite its good mAP results and being only 4% away from the project goal 1.3.2.8 which states the final model should achieve a desired detection accuracy score of 70% or greater, the webcam results were disappointing. This instance the accuracy of the detections was roughly 60% to 76%, but the issue resides with the recall. The detections in real-time for this model are very rapid but not consistent. The bounding boxes flicker very often in the right place, but on most occasions the bounding boxes will disappear after a second. Previous models did not have this issue, the main issue with this model is not its accuracy but its ability to reliably detect the actual object itself and maintain that detection over time. On the positive side, the model can now (barely) make predictions for medium and small areas. Which is still an improvement from the previous model. This model, on paper, has achieved its objective but when testing via webcam input has yet again revealed flaws in performance. *Figures 73* and *74* demonstrate the model's performance when the person is sitting further away from the camera. The previous model was not able to make any detections when the person was sitting this far away from the camera. Figures 75 and 76 demonstrate the performance the person is sitting closer to the camera. In Figure 76, it can be seen that there are two bounding boxes overlapping with each other. Overall, despite this model having the best mAP out of all the other complete models, it does not perform as well as expected. This is why a practical approach was needed when testing as it reveals some flaws that the mAP results do not.

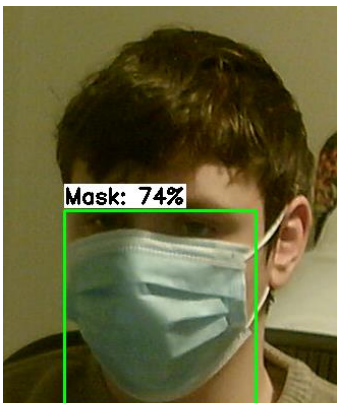

Figure 73: Wearing Face Mask – 30 cm Away from Camera

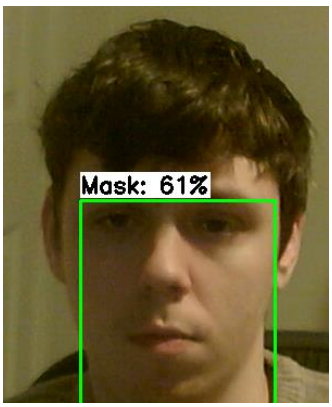

Figure 74: No Face Mask – 30 cm Away from Camera

Figure 75: No Face Mask – 12 cm Away from Camera

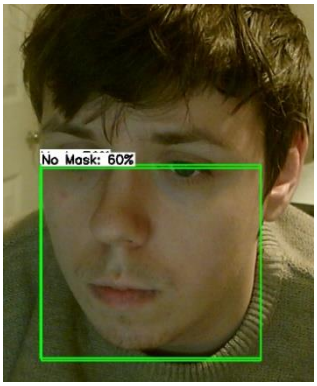

Figure 76: No Face Mask – 12 cm Away from Camera

6.2.8.5 Test Images Evaluation

The predictions made on small and medium sized objects in *Figures A2.45* and *A2.47*, found in appendix 2, demonstrate that with the introduction of new data, these types of objects are now able to be detected by the model. However, both images also demonstrate that the model lacks the ability to detect these objects accurately and consistently. After reviewing *Figure A2.45* in closer detail, it can be seen that the model has not detected the person in the foreground without the mask. However, the model did detect the person in the background who is wearing a face mask but during the labelling process, the person in the background who is wearing the mask had been missed by mistake. As seen in *Figure A2.46*, which represents the ground truth labels in the image. *Figure A2.47* also contains missing detections; three objects were labelled as seen in *Figure A2.48,* but only one was detected in *Figure A2.47*. Results such as this call for more data to be introduced, as the model is struggling to detect the face mask and no mask class at different variations, such as angles and size. Lastly, the detections made on the images containing the artificial face masks remain unchanged. These images are still being detected with high accuracy as seen in *Figure A2.43*. See appendix 2 for test images.

6.3 Transfer Learning Using Keras Method

6.3.1 "FACE CLASSIFIER + FACE MASK CLASSIFIER" EXPERIMENT

In this section, the results of the Keras transfer learning implementation are discussed. To avoid unnecessary overlap, this section will not mention the implementation again as that can be found in 5.3. Since this method only has one experiment, this section will aim to present and discuss the outcome of that experiment. The face mask classifier model had undergone 5 folds of training. The full results of all 5 folds can be found in appendix 3. The method had been adopted and implemented as the experiments from the TensorFlow API method had been failing to reach the 70% accuracy mark. Transfer learning has been known to produce very high accuracy results with minimal data. The same CNN model, MobileNetV2, had been used in this method. The model had been pretrained on the ImageNet dataset as mentioned previously. Some datasets used in the TensorFlow API method had been carried over to this methodology to form the dataset used for the training.

6.3.1.1 Experiment Settings

The model had been trained for 10 Epochs with a batch size of 18 using the GPU installed in the local machine instead of Google Colab this time. The GPU installed was able to handle this task as it was less computationally expensive in comparison to the TensorFlow API method. The learning rate was set to 1e-4. As mentioned before, the model was evaluated using K-Fold cross-validation, using 5 folds. Chapter 3 section 3.3.7 explains and illustrates how K-Fold cross-validation works when using 5 folds.

6.3.1.2 Dataset

There were 1000 images used to represent the face mask class and 1000 images were also used to represent the no mask class. However, as discussed in the implementation of the methods (5.3), image data augmentation was used. By using image data augmentation, each image would become augmented. Therefore, there are in fact 2000 images representing each class during training. Multiple datasets had been used for both classes to create a diversity in data. For the face mask

class, the MaskedFace-Net dataset[42], the face mask dataset by andrewmvd[43] and a dataset by omkargurav[44] were used. The dataset by omkargurav is another instance where the face masks had been applied to the faces of people artificially, similar to the MaskedFace-Net dataset. Additionally, the dataset mentioned in 6.2.8.2, made by collecting as well as manually augmenting images from google images was used to represent both classes. The no mask class was represented by the FFHQ Face Data Set[45], the face mask detection dataset by prithwirajmitra[46] and the dataset[47] used in the glasses experiment was partially included too. **NOTE:** The dataset for both face mask and no mask used *must* contain one sample per image, as the samples cannot be localised using labels like in the TensorFlow API method.

6.3.1.3 Results

This model was the fastest to train, completing training in 42mins 6secs. Transfer learning significantly reduced the time needed to train the model and the computational power required, therefore this model was able to be trained locally. Upon first glance, the results are the highest seen from any experiment performed so far. However, a closer inspection reveals that overfitting may be the cause of this as indicated by the score per fold, which can be seen in appendix 3 along with the full set of results for all 5 folds. Fold 1 had achieved the highest accuracy, reaching 100.0% accuracy and loss of 0.00996. Meanwhile fold 4 achieved an accuracy of 98.5%, which was the lowest accuracy out of all the folds, it also achieved a loss of 0.02849. Overall, the average score of all folds was 99.3500006198883 (+- 0.5385159671441138). *Figure 77* demonstrates the progress of fold 5 during its training. Sadly, due to time constraints during implementation, the code was written to only plot and output the graph of the final fold. However, as the score for all folds indicated, there is only a +- 0.54 give or take difference between the folds. Meaning all of the folds had achieved similar scores, making *Figure 77* a good representation of the behaviour of each fold during training. This model had very obviously been overfitting; this is likely due to the samples in the dataset being too similar to each other. A dropout layer was added during implementation to prevent overfitting, Image augmentation had been proven to help prevent overfitting and was also implemented. However, the model still overfitted vastly. Therefore, the attention turns to the dataset. No model can truly be 100% accurate, 100% of the time. Especially not on the first attempt. Overfitting is not only indicated by a very high validation score but a common way of finding out whether a model has overfitted or not is to provide it with unseen data. This data was provided to the model via webcam, discussed in 6.3.1.4.

42 https://github.com/cabani/MaskedFace-Net

43 https://www.kaggle.com/andrewmvd/face-mask-detection

44 https://www.kaggle.com/omkargurav/face-mask-dataset

45 https://www.kaggle.com/greatgamedota/ffhq-face-data-set

46 https://www.kaggle.com/prithwirajmitra/covid-face-mask-detection-dataset

47 https://www.kaggle.com/coffeeshop/people-in-dioptric-glasses

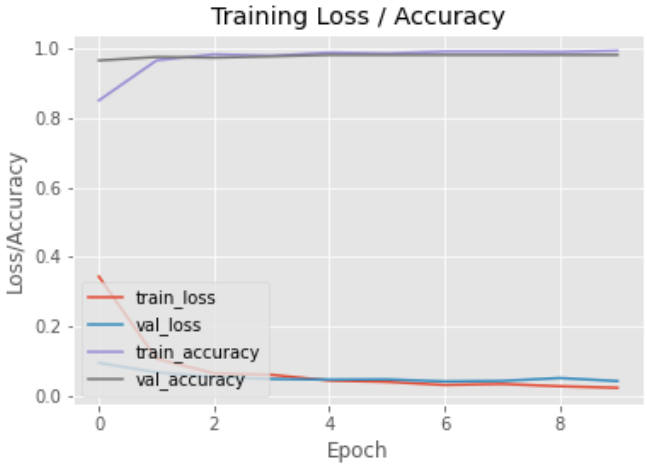

Figure 77: Performance During Training

6.3.1.4 Live Webcam Input

The model had been fed input via live webcam stream, this will help uncover the models true performance, as overfitted models typically cannot make accurate predictions on unseen data. However, this model had performed very well and consistently produced high accuracy results as seen in *Figures 78, 79 and 80.* Predicting each class correctly with a confidence score of 100% each time. However, the model seemed to have had more trouble detecting the dark face mask as opposed to the light blue one, as seen in *Figures 79 and 80*. This may be due to an insufficient amount of data representing different coloured face masks. The bounding box for the dark coloured face mask had been slightly smaller than the bounding box for the blue mask, leaving parts of the face mask outside the bounding box. Similarly, the model had also occasionally struggled with various angles, sometimes rapidly switching between classes. This can be improved by providing the model with more data to represent these weaknesses better, so that the model can improve its predictions for various angles, as well as face mask colours/types.

Lastly, even though the MobileNetV2 model is widely considered a lightweight model, this webcam test could suggest otherwise. Technically, there are two models being executed at the same time, the face classifier and the face mask classifier. This tremendously increases the computational cost of this approach, resulting in generally lower FPS than the TensorFlow API models when they have been tested via webcam stream. Occasionally this approach also causes the webcam to freeze/crash if too much rapid activity is happening. i.e., moving really fast, or purposefully distorting the view of the camera. Overall, this is a highly accurate model but it contains many disadvantages. The model can be improved by providing it with a larger dataset filled with a bigger diversity in samples, especially in relation to face mask types and face mask angles.

Figure 78: No Mask

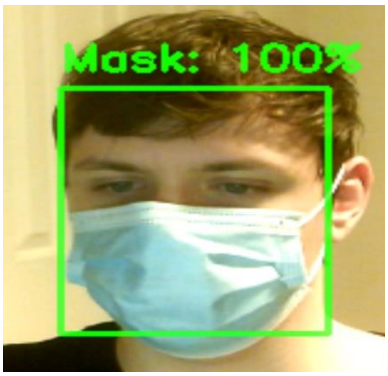

Figure 79: Blue Face Mask

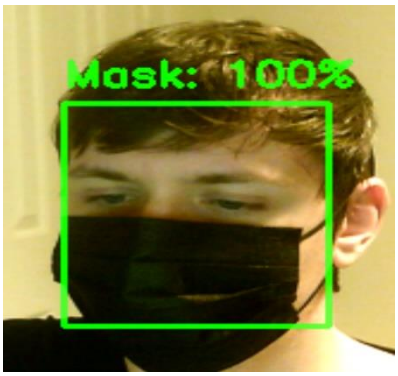

Figure 80: Dark Face Mask

Figure 81: No Mask + Face Mask NOT Being Worn

Additionally, *Figure 81* demonstrates that the model will only detect the face mask if the face mask is being worn by the person.

Figure 83: Face Mask 30cm+ From Camera

Figure 82: No Mask 30cm+ From Camera

7

Evaluation

7.1 Chapter Overview

This chapter aims to evaluate and compare the performance of the approaches and methods used as well as further analyse the results and see what could have been improved. This chapter will also discuss the concerns and challenges faced before or during the development phase of the project. Lastly, a summary and brief discussion will be made as to how/if the project objectives stated in chapter 1 section 1.3 were met.

7.2 The Final Model

A multitude of experiments had been conducted using various approaches and methods. The performance of each experiment had primarily been compared against the previous version. However, in the section the models produced in the experiments chapter are compared against each other to find out which model, overall, is the best performer.

7.2.1 Evaluation of Models

The models marked in blue in the *Figure 84* had been evaluated using COCO Mean Average Precision (mAP). Meanwhile the model marked in orange had been evaluated using the accuracy from K-Fold cross validation. However, the legitimacy of this score is debatable as discussed in 6.3.1.3. However, it was featured nonetheless in order to be involved in the discussion.

Despite the “dataset rework” experiment producing the model with the highest mAP, it had recorded no mAP for detections of medium and small area objects. Making it incapable of detecting said areas. The reason for the mAP being so high is due to the large area objects being the only category being evaluated and the only category of mAP to contribute to the final mAP score. This model cannot be the final model due to its inability to detect medium and small area objects.

The next model with the highest mAP score is the model produced from the “Training medium and small area objects” experiment, which was a direct follow-up to the “dataset rework” experiment. This experiment attempted to rectify the issues faced with the model produced by the “dataset rework” experiment by introducing more data to represent the medium and small area objects. However, it was only partially successful, as discussed in 6.2.8, the detection of medium and small objects had seen some success, however the accuracy and recall via webcam input was very poor. The “dataset rework” experiment is responsible for the high mAP of the model produced by the “training medium and small areas object” experiment, as the mAP for large areas still contributes greatly to the final mAP score. This model still requires further development to improve the detection of medium and small area objects therefore it cannot be the final model.

The models produced from the “Incorrect Mask” and the “Face Mask Addition” experiments both have the next highest mAP, tied at 63%. However, the model produced by the “Incorrect Mask” experiment cannot be considered as the final model as it was an experimental model, aiming to implement a complex class to discover the behaviour of the model when this class was introduced. The class also had a very large data imbalance as mentioned in 6.2.6.3. Meaning this class is not performing at optimal level yet. Additionally, its presence will also disrupt the detections of the face mask and no mask class, therefore it cannot be considered as the best model.

The “Face Mask Addition” experiment had produced a complete model which was able to detect the face mask and no mask class across all area sizes. However, an issue had been found in relation to the no mask class being miscategorized if the people in the image wore glasses. This issue had been discussed in 6 2.4.4 during the live webcam input test. This issue was significant enough to birth the idea of the “Glasses” experiment, which had completely fixed this issue by providing relevant input data. As the model produced by the “Face Mask Addition” experiment still had issues like the one just discussed and the model from the “Glasses” experiment being considered its successor, this model most likely is not the final model. Although, this model is still regarded as a good model, just not good enough as it still needed additional development.

The model produced by the “Glasses” experiment had been the successor to the “face mask addition” model as it had rectified a major issue that the previous model had. While simultaneously providing more data and training time to the model, creating a large benefit overall. The “glasses” model had scored a mAP of 62%, 1% lower than its previous iteration. Overall, this model was a complete model that did not face any major issues when testing on unseen data, when using the live webcam stream. This model was a strong entry that achieved almost all of the project goals and was one of the best models produced on account of its overall performance and reliability. However, additional investigation into the performance is needed to boost the mAP score.

The model produced by the “No Mask Only” experiment had scored 1% lower than the “Glasses” model. However, despite the models excellent performance when tested via webcam stream, it cannot be the final model as the model was part of a two-stage experiment which concluded with the “Face Mask Addition” model being created. The “No Mask Only” model only had the ability to detect the no mask class. Whereas the requirements state that the final model should be able to detect both the face mask and no mask class when appropriate (1.3.2.1). Therefore, the model definitely cannot be considered as the final model as it is only half complete.

Lastly from the TensorFlow API method, the “Initial Baseline” and “Initial Baseline Part 2” models are the early iterations that were used to gauge which direction to take with the development. These experiments had been testing the waters, so to say. Neither of the models produced by these two experiments can be in contention for the final model on account of their poor performance, statistically and practically.

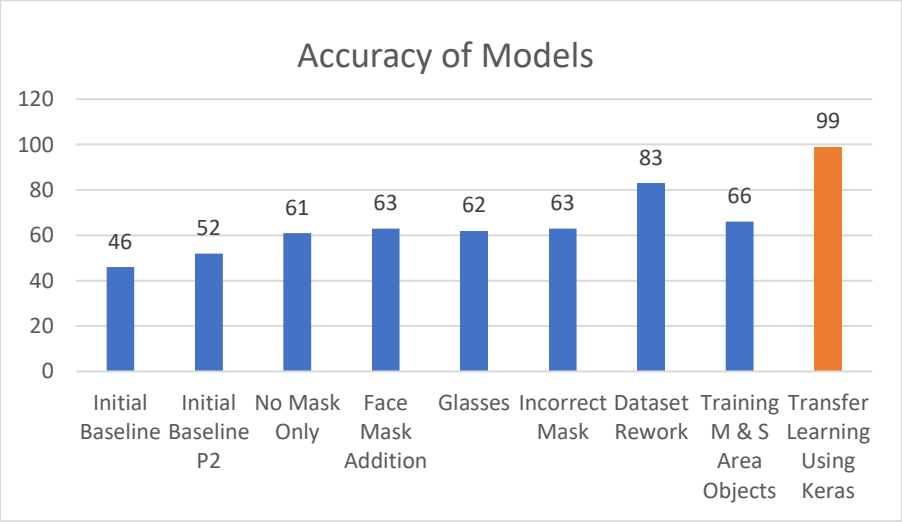

Figure 84: Accuracy Comparison of All Models

7.2.2 TensorFlow API or Transfer Learning Using Keras

Based on the evaluation of the models from the previous section, it is clear that the "Glasses" model had been the best performing and well-rounded model overall created using the TensorFlow API methodology. Since the transfer learning using Keras method only produced one model, that model will be compared against the "Glasses" model to decide which model and method had best met the objectives of this project. Table 1 below outlines the project objectives and whether they had been met by the models/methods. The Transfer learning with Keras model had used many of the same datasets as the "Glasses" model from the TensorFlow API did. The two methods had also shared the same CNN model, that being MobileNetV2. It can be assumed that many of the abilities that the "Glasses" model had, the transfer learning model has too but this still required investigation.

Project Goals	TensorFlow API ("Glasses" Model)	Transfer Learning Using Keras
1.3.2.1 *The final model must be able to detect "Face Mask" and "No Mask" objects correctly when appropriate.*	Completed	Completed
1.3.2.2 *The final model must be able to detect the "Face Mask" object only if a face mask is being worn by a person.*	Completed	Completed
1.3.2.3 *The final model must be able to detect all types of face masks being worn.*	Completed	Completed
1.3.2.4 *The final model must be able to detect people who are/are not wearing face masks regardless of their colour, background, gender, size and general variations in appearance.*	Completed	Completed
1.3.2.5 *The final model must be able to make detections from various angles*.	Completed	Completed
1.3.2.6 *The final model must be able to detect objects simultaneously. This includes*	Completed	Completed

detecting more than one class or object at the same time.		
1.3.2.7 *The final model must be able to detect objects no matter the size (big to small).*	Completed	Completed
1.3.2.8 *The final model should achieve the desired detection accuracy score of 70% or greater.*	Not Met	Completed
1.3.2.9 *The chosen model must be computationally lightweight and have a low computational cost.*	Partially	Not Met

Table 1: Project Goals Met by Models

To understand whether the transfer learning using Keras model has met the objectives, we must consider the capabilities of both the face mask classifier, trained using transfer learning, as well as the pretrained face classifier model known as: "res10_300x300_ssd_iter_140000.caffemodel". In 6.3, its shown that the face mask classifier model is capable of detecting face masks and no masks as well as various types of masks at more than one angle. Additionally, *Figure 81* demonstrates that the model is able to disregard face masks that are not being worn by a person. The face classifier model has been proven to have the ability to detect multiple faces at once, no matter their appearance variations. The face mask classifier simply has to classify whether the person is wearing a face mask or not. To find out whether this model is able to detect faces at various distances (big to small), the model was tested for both classes via webcam stream by standing far away from the camera as seen in *Figures 82 and 83.* It appears that the face mask class performed slightly worse at distance, in that instance it achieved 12% less in terms of confidence score than the no mask class. However, both classes can be detected at range. The face mask classifier model had achieved a score of 99% across all folds. However, despite the data overfitting during training, the score is partially reflected when providing input via webcam stream, as the model is capable of making very accurate detections at times, exceeding the 70% mark usually. The downfall of this methodology is that, despite MobileNetV2 being a computationally lightweight model, when it is paired with the caffemodel, this increases the computational cost very noticeably. Overall, according to the task manager performance tab, the two-stage transfer learning method had utilised 38% of the GPU when executed, whereas the TensorFlow API method had only utilised 3%. However, the TensorFlow API method had used more RAM.

In terms of the "Glasses" model from the TensorFlow API method, the model has met all project objectives aside from the accuracy goal. The model had achieved a mAP accuracy of 62% but the objective had been to meet an accuracy of 70% or greater. The model was able to detect both the face mask and no mask classes, no matter the angle, the person, the face mask type/colour or how large or small the object was. Importantly, the model was able to also disregard the face mask if it was not being worn by a person as seen in *Figure 48*. This can all be seen in section 6.2.5, where the "Glasses" experiment is discussed, as well as in appendix 2, where some examples of the test images are stored. Lastly, the computational cost goal has only been partially met for this model/method. Despite only utilising 3% of the GPU while being executed, it had utilised 4.5GB of the GPU RAM. The GPU used is the NVIDIA GTX 1060 6GB. Therefore, this could be considered as

somewhat high. However, this has no negative impact on the model or performance. Whether a program is interpreted as computationally expensive can also be dependent on the hardware being used. Aside from the computational complexity calculations that could be done, the perspective can also matter in this situation. For example, if an NIVIDA Tesla K80 12GB GPU was used instead, then 4.5GB of GPU RAM usage would appear to be relatively inexpensive. Despite the TensorFlow API method being somewhat computationally expensive, it only utilises 3% of the GPU and provides seamless execution when visualising the models detections in real-time via webcam stream.

On that note, the more preferable model and method in this instance is the "Glasses" model from the TensorFlow API method. This is because reducing computational cost is a far greater challenge than simply increasing the accuracy of the model. Computational cost can be a stubborn issue with a lot of technical background, making it a more difficult problem to deal with. Whereas continuing to experiment with the model and making appropriate results-based decisions can improve the accuracy and allow for the model to eventually meet that final objective of having an accuracy of 70% or greater. On the basis of this and other points made in the section, the "Glasses" model is best suited to be considered the final model.

In terms of training, transfer learning had significantly reduced the training time required. The model trained in 42mins 6secs, whereas the final model had been trained in 6hrs 10mins 56secs (370mins 56secs). Furthermore, a pretrained model is always utilised during transfer learning, the use of the ImageNet dataset provided most of this models accuracy as the knowledge was transferred from the weights created by training using that dataset. Whereas with the TensorFlow API method, there was no prior starting knowledge when training the model. Additionally, transfer learning had removed the efforts required to label the dataset by hand. It had taken over 24hrs in total to label the dataset used in the "Glasses" experiment but with transfer learning, after selecting the appropriate images as mentioned in 6.3.1.2, there was no need to spend this much time. Overall, the transfer learning using Keras method could be improved by finding another way to visualise the detections via webcam, possibly by removing the face classifier model and using the face mask classifier standalone.

7.3 Final Model & Baseline Models

ResNet50V1 and EfficientDet D1 were two additional models that had been trained using the same hyperparameters and dataset that the final model had used. This was mainly done for comparison reasons, to see whether other TensorFlow 2 models could possibly perform better in terms of accuracy. There had not been a strict selection criteria when selecting these models as they were only being trained for comparison and not for the same purpose MobileNetV2 was being trained for. However, all three models take in an input size of 640x640. The current model, MobileNetV2, had an accuracy of 0.618 (62%). Whereas ResNet50V1 obtained an accuracy of 0.607 (61%) and EfficientDet D1: 0.697 (70%). The EfficientDet D1 model had reached the accuracy goal of this project, beating MobileNetV2 by 8% and ResNet50 by 9%. EfficientDet D1 had also obtained the lowest validation loss of 0.191 compared to MobileNetV2's 0.405 and ResNet50V1's 0.469. It is evident by the evaluation and *Figures 85* and *86* that ResNet50V1 had performed the worst out of the models. In *Figure 85*, it can be seen that the ResNet50V1 had taken the longest amount of steps to reach its peak accuracy and shortly after doing so, it had stabilized. Additionally, *in Figure 86*, it had seen a large spike in loss from the beginning. Eventually, it had converged on a smaller loss value however it still remained higher than the two models in terms of loss by step 50,000.

EfficientDet D1 had outperformed MobileNetV2 in terms of accuracy and validation, even reaching the accuracy goal of the project. EfficientDet models are also single shot detectors that utilise the EfficientNet backbone. The EfficientDet models range from D0 (lightweight) to D7 (heavyweight). According to the TensorFlow 2 model zoo[48], EfficientDet D1 has a speed of 54ms. All

48 https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

of the statistics displayed on this page regarding the models performance is collected after the models have been trained on the COCO dataset. However, MobileNetV2 is said to have a speed of 39msm which is faster than the EfficientDet D1 model. Additionally, the (COCO) mAP of EfficientDet D1 is said to have an accuracy of 38.4, while MobileNetV2 has 28.2. This superiority of accuracy from EfficientDet D1 is also reflected in the evaluation discussed earlier in this section. On this basis, the EfficientDet models should be involved in future experiments to see how they perform. The accuracy of the EfficientDet models increase with each incremental version but so does the computational cost. Therefore, the model with the appropriate trade-off between computational cost and accuracy must be realised if these models are to be included in future experiments.

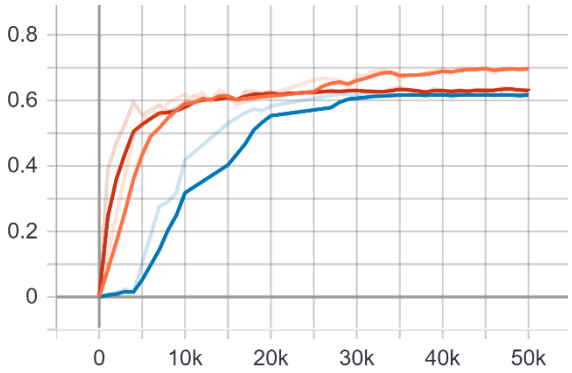

Figure 85: mAP of All Models

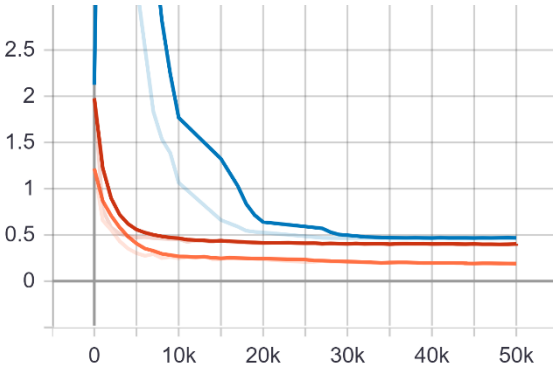

Figure 86: Validation Loss of All Models

Keys:

MobileNetV2: Red

ResNet50V1: Blue

EfficientDet D1: Orange

7.4 Preliminary Concerns & Challenges

Before and during development, there had been some concerns as to whether certain aspects or issues may arise and impact the performance and results of the models. This section aims to discuss these concerns and whether they did in fact hold any weight over the results and overall performance of the final model.

7.4.1 Beards

Making accurate predictions on people with large beards was a concern from the very beginning as these beards could potentially cover the lower region of the face, which was typically used to determine whether a person was wearing a face mask or not. Even beards that are not very large may aid the model in miscategorising whether a face mask was being worn or not, as these beards are still positioned where the mask would typically be. Another concern was that the beard would be interpreted as a differently coloured mask as well, depending on the colour of the beard. If beards were to be detected as masks, there would be constant overlap as the features of the no mask class would still be present in the image. To help ensure this issue would not arise, images of people who had beards were included in the data that represented both classes. Although, this was more targeted towards the no mask class as this was the class where this issue would most likely arise. As a result of this, no issues regarding beards were recorded during the development of the project. Additionally, it is most likely that beards were not categorised as face masks because the lower facial features were still visible; this was enough information for the model to make the correct prediction.

7.4.2 Face Mask Variation

Ensuring that the model predicted face masks correctly, regardless of the colour or design etc, was not only a challenge that needed to be addressed during development, but also a project goal. All the more reason to plan ahead to accomplish this project goal and ensure all types of face masks can be detected. Face mask variation not only includes different coloured face masks but also differently

shaped face masks too. When gathering data, data had been examined to see whether there was a vast enough diversity in terms of face mask appearance. However, generally speaking, any type of face mask which covers the lower facial region could be enough to incentivise the model to categorise it as a face mask. The main concern in terms of face mask variation was that some masks may have had a very eccentric appearance, so much different from the others that the knowledge gained by the model from examining other face mask images, would not be transferable to the face masks with an unorthodox appearance. However, no issues of this nature had been found during evaluation. Additionally, *Figures A2.27* and *A2.29*, found in appendix 2, are examples of face masks with an uncommon appearance. However, these face masks were still detected accurately by the final model.

7.4.3 Facial Expressions

Another concern and challenge was ensuring that the model was able to detect when people were not wearing a mask, regardless of what their facial expressions looked like. This includes but is not limited to smiling, anger, happiness, open mouth, shouting, mid-sentence, no expression, etc. Similarly, to how the face mask variation was approached, data had been examined and purposefully included to represent certain common facial expressions such as smiling, shouting, no expression. The thought process had been that if these main facial expressions had been accounted for when collecting data, then eventually the model would be able to adapt to other similar expressions. When examining the test data after training and evaluation, no issues regarding facial expressions had come to light. Although, some models had strangely produced higher confidence scores if a certain facial expression such as shouting was seen in the image.

7.5 The Impact of Poor Labelling

Labelling images during the pre-processing stage can be extremely time consuming. It is necessary in order to create the ground truth labels so that the model can use them as a point of reference during training to learn the features and appearance of an object. Additionally, they are used during evaluation to gauge how well the model is performing. Section 5.2.4 in the model implementation chapter describes the labelling approach used during the data pre-processing stage. Although, the "Initial Baseline" and "Initial Baseline Part 2" experiments had first used the traditional labelling approach. This changed after the "Initial Baseline Part 2" experiment and the rest of the experiments had used the new proposed labelling approach.

During the labelling process for the images which represented the no mask class, the labelling approach used had directly targeted the lower facial features that would typically be covered by the face mask such as, mouth, nostrils, teeth, lips, chin, etc. If these features are visible, it indicates that a face mask is not being worn or it is being worn incorrectly. After training and evaluation concludes, test images are returned with predictions made by the model, visualised by bounding boxes. These images can be found in appendix 2, however *Figures 87* and *88* provide an interesting example of "poor labelling". *Figure 88* is the ground truth which was labelled during pre-processing while *Figure 87* is the prediction made by the model. Specifically, this model was produced by the "No Mask Only" experiment.

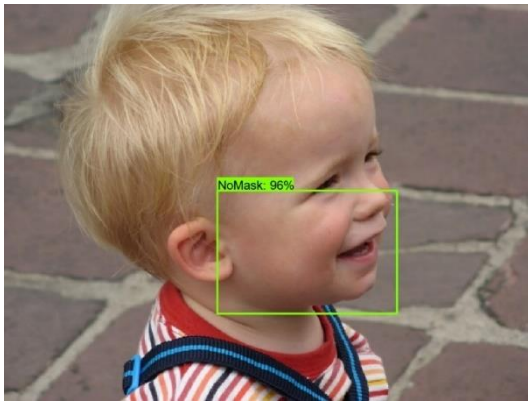

Figure 87: Predicted Bounding Box

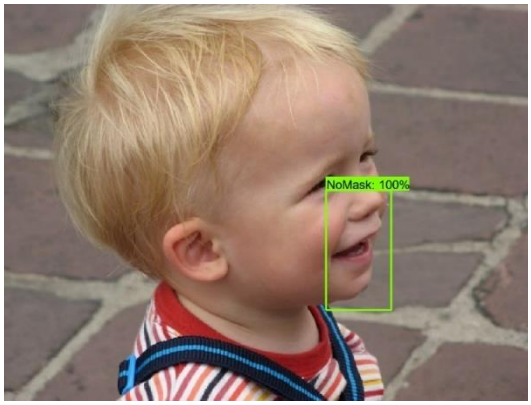

Figure 88: Ground Truth Bounding Box

Despite the predicted bounding box not converging on the ground truth bounding box, the predicted bounding box is actually still correct. To demonstrate how this can impact the results, the IOU needs to be calculated. In simple terms, the IOU is expressed as:

$$IOU = \frac{Intersection}{Union} \quad (21$$

Examine *Figure 89,* It is the product of mixing *Figures 87* and *88* together using Adobe Photoshop CC in order to visually represent the intersection and union in the images. The predicted bounding box does not fit to the ground truth box well. Whether this predicted bounding box will be a TP or FP now depends on the result of the IOU and threshold it is being measured at. Let us hypothetically assume that the IOU result of *Figure 87* is 52%, we must realise that the COCO mAP measures across 10 different thresholds *(IOU=0.50:0.95).* These different IOU values act as cut-off points. For example, at threshold =0.50, this detection will barely pass as a TP, which contributes positively to the final mAP score. However, as the IOU threshold gets stricter, such as threshold=0.75, this detection will no longer be considered as a TP, but instead a FP since 52 is less than 75, this would impact the final mAP negatively.

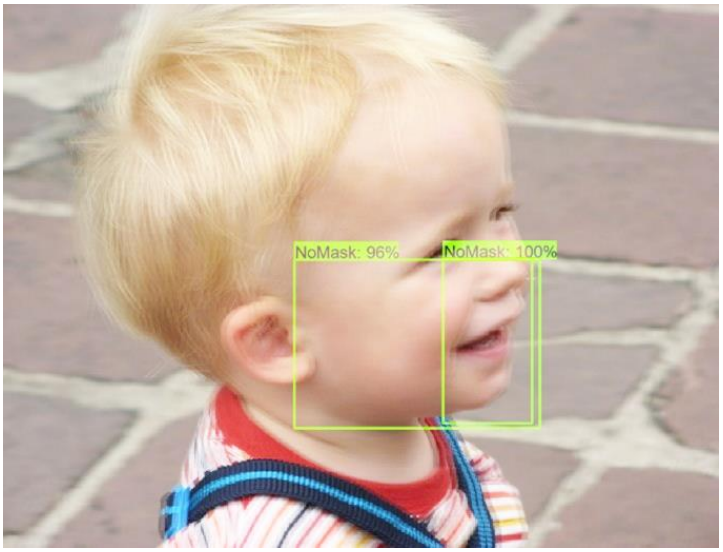

Figure 89: Predicted & Ground Truth Overlayed

However, this could all have been avoided if the image had been labelled more appropriately or if the model were able to converge more accurately to the ground truth label. While the fault still

mainly belongs to the model for not converging correctly, human error or perfectionism can still be blamed here. The prediction made by the model as seen in *Figure 87* is by all means a correct one as the prediction covers the lower region of the face where the face mask would typically reside if it were being worn. This was the labelling approach used as mentioned many times before but in this particular instance, the ground truth label had been labelled poorly but not incorrectly. This may seem like a very minor detail as one image may not hold much weight over the final mAP result. However, if this labelling situation were to be continuously repeated, it would begin to generate more and more FP's, which will impact the mAP negatively. If the ground truth label had been labelled more appropriately, then the IOU result could have been a higher one, such as 80%. This would now be considered as a TP at both 0.50 and 0.75 thresholds, generating an extra TP instead of an unnecessary FP. Additionally, this issue could lead to a mAP score that does not represent the models performance very accurately if there were to be many instances of this issue.

7.6 Legal, Social, Ethical and Professional Issues

A plethora of issues may arise if this model were to be implemented into embedded systems such as surveillance cameras in order to monitor close-quarter environments such as buses, stores, schools, etc. Constant tracking of people may raise a number of concerns and ethical issues as people typically value their privacy. If one of these systems were to be infiltrated, the model would also be highlighting how many people are present in a room or a certain location, no matter if they are wearing a face mask or not. The infiltrators may find this information useful but generally any security issue that may arise from the use of surveillance cameras will inherently also arise if this model were to be implemented into one of them. Those who would use this model as part of their surveillance systems and stored collected data, would need to adhere to the data protection principles[49] to ensure no data laws are broken and that no data is distributed illegally.

7.7 Potential Improvements

- The model could potentially be improved by creating more experiments where new results-based or data-driven hypotheses are tested. Additionally, different hyperparameters should be tested in these experiments, with changes made to other hyperparameters outside of the usual ones, such as the learning rate. The learning rate for TensorFlow API models typically used 1000 warmup steps followed by a gradual cosine decay as seen in *Figure 90.* The learning rate controls the speed at which the model is learning. It is important to set a suitable learning rate. If the learning is too high then this may cause divergent behaviour in the loss function which is undesirable. However, if the learning rate is set too low then the training would progress much more slowly as it would only be making small updates to the weights of the network. Lower learning rates are also known to increase the risk of the training data overfitting. Possible experiments for the transfer learning with Keras would be to experiment with the learning rate to see whether this has any impact on the overfitting model.

- The datasets used for the experiment need to have a stricter selection criteria as many of the issues during development, with both methodologies, had involved datasets influencing an undesired result. The results are typically only as good as the data is. Therefore, it is essential to learn from previous mistakes when working with data. Developing an understanding of what good and bad data is, could be the deciding factor in whether the model performs well or not.

49 https://www.gov.uk/data-protection

- Lastly, different models such as EfficientDet's, as previously discussed, can be experimented with to examine whether they provide any benefit to the project goals. There may potentially be models which perform better in certain areas, which may positively impact the project. More research would need to be conducted in order to find out about such models and whether they are a good fit for the project.

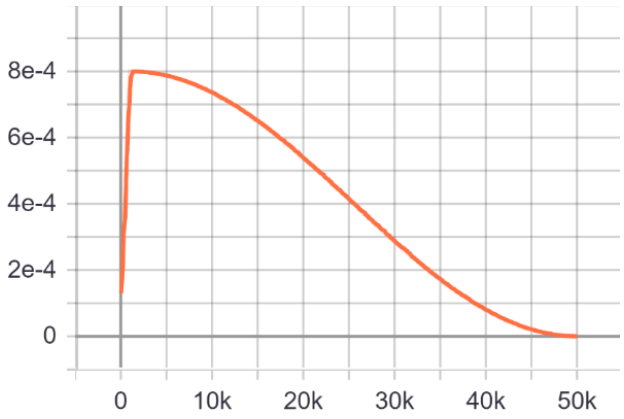

Figure 90: Learning Rate (TensorFlow API Models)

8

Conclusion & Future Work

8.1 Conclusion

The seriousness of the current COVID-19 global pandemic as well as the importance of correct face mask usage had inspired the idea for this project. The project aim had been to create a face mask detection model, this included detecting people who are/are not wearing a face mask, in real-time. This was believed to be of benefit given the current pandemic situation across the world, as the model could potentially be integrated with embedded systems in the future. This would allow for monitoring and detection of those who are not wearing a face mask in areas where it is necessary. This would help prevent the spread as wearing a face mask is proven to mitigate the travel of the virus. The project was made with this in mind, purposefully selecting a lightweight model for training as well as regularly examining the performance of the trained model(s) in real-time to gauge how well they may perform if deployed.

The experiments and results were crucial to the progression and the continuous development of the project, as the results of these experiments provided vital information about the status of the model. This information was examined and used to construct a suitable follow-up experiment. As the results were so important, appropriate evaluation techniques had been researched before development began to plan ahead and implement these techniques into the development cycle. Out of all available evaluation methods, COCO mAP seemed the most comprehensive and thorough as it had many layers of evaluation, such as multiple IOU thresholds, mAP, mAR, measurement of performance in regard to different area sizes, IOU, etc. Furthermore, tensor board and the live webcam input had provided visual representations of the detections, this proved useful in the decision-making process as the model could be evaluated from three different perspectives. mAP had provided statistical and mathematical evaluation, the test images returned with bounding boxes after evaluation and had provided a visual representation of the models performance in various settings. Finally, the webcam input tested the model on unseen data, in real-time. All of these different evaluation perspectives assisted in creating the best possible model and experiments.

The development of the project had come to a halt numerous times due to technical difficulties. The hardware and the machine originally used to train the models had not been very capable of doing so. Training and evaluation was impossible with the use of a single GPU as these processes had to run simultaneously. Additionally, it was also draining and damaging an already faulty GPU, therefore an alternative way to perform these tasks had to be found. Research regarding computational cost of training and evaluation should have been done in more depth beforehand to prepare for such problems. These problems had genuinely come unexpectedly and caused major disturbance to the development until a solution had been found.

The notion of a transfer learning methodology was always planned, for the purpose of comparison and in hopes of obtaining higher accuracy results. It was only introduced after the main methodology, the TensorFlow API, had stopped showing promising results. Which had occurred after the "Training medium and small area objects" experiment, as at that point, the progress felt like it was going backwards. The visualisation of the transfer learning methodology, adapted from the post by Adrian Rosebrock[50], may have been the downfall of the methodology, however. If the face mask classifier model had been visualised in a more traditional way, like the models from the TensorFlow API methodology were, then this may have reduced the computational cost greatly. This is because there would only be one model involved, whereas with the current implementation, there are two. This should have been foreseen and is a mistake that resulted in the method becoming so computationally expensive, that it could hardly execute reliably on a machine which was utilising a GPU. Thus, it will most likely not be able to execute reliably when integrated with embedded systems.

8.2 Future Work

There are many opportunities for the models and methodologies to see improvement in the future, some have already been mentioned in previous sections and chapters. However, there are still some suggestions which have not been mentioned or only partially discussed. Future work can include but is not limited to:

- **Further development of the "Incorrect Mask" class**

If the face mask is being worn incorrectly then this will greatly reduce the impact that the face mask has at preventing the spread of the virus. Therefore, this is a relevant area to investigate. A start has already been made at implementing this class as discussed in 6.2.6. However, a vast data imbalance prevents this class from performing well and becoming useful. Future work includes providing more data to the model to resolve this imbalance or even taking a whole new implementation approach entirely to implement this class.

- **Further development of Illumination Changes & Angles Coverage**

The model will react differently when presented with an object at a certain angle. Additionally, different lighting may also cause difficulty for the model. For example, standing directly under a light will alter the appearance of the face mask or face, due to the light pointing from a different direction or creating shade in different areas of the face. Image augmentation could assist in providing relevant training data to target this issue, as it could boost the brightness of images. The model could also benefit from further training using images that represent various new angles for each class. This may even boost the mAP score as the model would be able to make more accurate detections overall as well as from new angles more confidently.

- **Feature Extraction to Detect Masked Criminals via CCTV**

When a person is arrested, the police will photograph the person after the arrest to create a photographic record of the individual, informally known as a "mugshot". This can aid investigators in identifying the criminal if they are ever to be released or set on bail bond. These days, criminals are able to use face masks to simultaneously shield themselves from facial recognition as well as blend with the crowd without looking out of place. However, when wearing a face mask, the eyes and the upper face region are typically visible. Using the photographs taken during their arrest, the upper facial features can be extracted in a similar way to how the face ROI's had been extracted in the transfer learning using Keras implementation. A CNN model might then be trained to detect these features and the model could be integrated with embedded systems such as surveillance cameras.

50 https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

The model could be integrated with cameras that are strategically placed to get a close-up view of the person when in frame. These cameras could include ATM cameras or cameras positioned at the entrance/exit of stores or public transport. This way, even if the criminal is wearing a face mask, features such as eyes and upper facial structure, excluding general variables such as hair style and colour, may potentially be examined.

However, there would be a need for a feasibility study to be conducted before development could begin. CCTV cameras are not very clear or fast, they may not provide the quality input required to make such detections. Secondly, extracting such critical details and predicting them accurately will require thousands of samples. All of these factors would need to be taken into account and a small-scale model would need to be created to examine the theory. Most likely there will be a need for revisions to be made to the methodology in order to achieve the desired result.

- **Increasing Accuracy of Model**

There is still much room for improvement in terms of accuracy. The final model did not meet the 70% accuracy score, therefore improving accuracy and evaluating correctly should be prioritized. The experiments and results chapter contains a lot of knowledge in relation to what has already been attempted, this could be utilised in planning future experiments aimed at increasing the accuracy and overall performance of the model.

- **Low-Light & Dark Settings**

The data provided to the models during training had mainly contained samples with a daytime setting. However, whenever the light is dimmed or the setting gets darker, the model will have trouble making detections. Training data that primarily targets low-light environments for both the face mask and no mask classes could be provided to aid in solving this issue. This could increase the performance in the model in darker settings. This is especially important as the models in their current state would not be able to make detections at night-time or in darker environments. Therefore, it is important to implement this ability.

8.3 Critical Appraisal

In conclusion, this project had not one but multiple functional models and this thesis contains all the necessary documentation for a future developer to continue the work done here. The report itself turned out to be much larger than originally expected as during development, a lot of content had been created. During the writing stage of the report, the aim had been to only include the topics and information that was relevant to the topic, development, problem domain and technical implementation. The report is very extensive but it covers all of the relevant information obtained since the beginning of the project in mid-January up to the end of development in early April. The project itself can be seen as successful despite the main model not completing an objective. The documentation provided in this report will be enough to influence appropriate decision-making in regard to future development.

Overall, the project had been very enjoyable once all of the technical issues had been solved. Experimenting and training had been interesting as it involved a lot of critical thinking and problem solving to generate suitable experiments to solve the issues of the previous one. This is where most of the time during development had been spent and can be seen as a core aspect of this thesis. As machine learning and AI are fields of study that are at large right now, the understanding and knowledge gained while developing this project will be of real use in the world today.

BIBLIOGRAPHY

Almási, A.-D., Woźniak, S., Cristea, V., Leblebici, Y. and Engbersen, T. (2016). Review of advances in neural networks: Neural design technology stack. Neurocomputing, 174, pp.31–41.

Al-Jarrah, O.Y., Yoo, P.D., Muhaidat, S., Karagiannidis, G.K. and Taha, K. (2015). Efficient Machine Learning for Big Data: A Review. Big Data Research, 2(3), pp.87–93.

Arcos-Garcia, A., Alvarez-Garcia, J.A. and Soria-Morillo, L.M. (2018). Evaluation of deep neural networks for traffic sign detection systems. Neurocomputing, [online] 316, pp.332–344.Available at: https://www.sciencedirect.com/science/article/pii/S092523121830924X

Altmann, D.M., Douek, D.C. and Boyton, R.J. (2020). What policy makers need to know about COVID-19 protective immunity. The Lancet, [online] 0(0). Available at: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(20)30985-5/fulltext.

Balaji, S. and Sundararajan, M. (2012). International Journal of Information Technology and Business Management WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. International Journal of Information Technology and Business Management, [online] 2(1). Available at: https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf.

Barbedo, J.G.A. (2018). Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification. Computers and Electronics in Agriculture, 153, pp.46–53.

Betsch, C., Korn, L., Sprengholz, P., Felgendreff, L., Eitze, S., Schmid, P. and Böhm, R. (2020). Social and behavioral consequences of mask policies during the COVID-19 pandemic. Proceedings of the National Academy of Sciences, 117(36), p.202011674.

Bhardwaj, K.K., Banyal, S. and Sharma, D.K. (2019). Artificial Intelligence Based Diagnostics, Therapeutics and Applications in Biomedical Engineering and Bioinformatics. [online] Available at: https://www.sciencedirect.com/science/article/pii/B9780128173565000097

Bi, Q., Goodman, K.E., Kaminsky, J. and Lessler, J. (2019). What is Machine Learning? A Primer for the Epidemiologist. American Journal of Epidemiology, 188(12).

Cabani, A., Hammoudi, K., Benhabiles, H. and Melkemi, M. (2021). MaskedFace-Net – A dataset of correctly/incorrectly masked face images in the context of COVID-19. Smart Health, [online] 19, p.100144. Available at: https://arxiv.org/pdf/2008.08016.pdf.

Chavez, S., Long, B., Koyfman, A. and Liang, S.Y. (2020). Coronavirus Disease (COVID-19): A primer for emergency physicians. The American Journal of Emergency Medicine.

Cheng, G., Han, J., Guo, L., Qian, X., Zhou, P., Yao, X. and Hu, X. (2013). Object detection in remote sensing imagery using a discriminatively trained mixture model. ISPRS Journal of Photogrammetry and Remote Sensing, 85, pp.32–43.

Chen, L., Zhang, Z. and Peng, L. (2018). Fast single shot multibox detector and its application on vehicle counting system. IET Intelligent Transport Systems, 12(10), pp.1406–1413.

Chu, D.K., Akl, E.A., Duda, S., Solo, K., Yaacoub, S., Schünemann, H.J., Chu, D.K., Akl, E.A., El-harakeh, A., Bognanni, A., Lotfi, T., Loeb, M., Hajizadeh, A., Bak, A., Izcovich, A., Cuello-Garcia, C.A., Chen, C., Harris, D.J., Borowiack, E. and Chamseddine, F. (2020). Physical distancing, face masks, and eye protection to prevent person-to-person transmission of SARS-CoV-2 and COVID-19: a systematic review and meta-analysis. The Lancet, [online] 395(10242). Available at: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(20)31142-9/fulltext.

Cogswell, M., Ahmed, F., Girshick, R., Zitnick, L. and Batra, D. (2016). Reducing Overfitting in Deep Networks by Decorrelating Representations. arXiv:1511.06068 [cs, stat]. [online] Available at: https://arxiv.org/abs/1511.06068

Cunningham, P., Cord, M. and Delany, S.J. (2008). Supervised Learning. Machine Learning Techniques for Multimedia, pp.21–49.

Dawar, N., Ostadabbas, S. and Kehtarnavaz, N. (2019). Data Augmentation in Deep Learning-Based Fusion of Depth and Inertial Sensing for Action Recognition. IEEE Sensors Letters, 3(1), pp.1–4.

Deepak, S. and Ameer, P.M. (2019). Brain tumor classification using deep CNN features via transfer learning. Computers in Biology and Medicine, 111, p.103345.

Desai, A.N. and Mehrotra, P. (2020). Medical Masks. JAMA. [online] Available at: https://jamanetwork.com/journals/jama/fullarticle/2762694.

Durand, T., Thome, N. and Cord, M. (2016). WELDON: Weakly Supervised Learning of Deep Convolutional Neural Networks. [online] openaccess.thecvf.com. Available at: https://openaccess.thecvf.com/content_cvpr_2016/html/Durand_WELDON_Weakly_Supervised_CVPR_2016_paper.html

Dzisi, E.K.J. and Dei, O.A. (2020). Adherence to social distancing and wearing of masks within public transportation during the COVID 19 pandemic. Transportation Research Interdisciplinary Perspectives, 7, p.100191.

El Naqa, I. and Murphy, M.J. (2015). What Is Machine Learning? Machine Learning in Radiation Oncology, pp.3–11.

Glaser, J.I., Benjamin, A.S., Farhoodi, R. and Kording, K.P., (2019). The roles of supervised machine learning in systems neuroscience. Progress in neurobiology, 175, pp.126-137.

Google AI Blog. (2017). MobileNets: Open-Source Models for Efficient On-Device Vision. [on-line] Available at: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html

Everingham, M. and Winn, J., 2011. The pascal visual object classes challenge 2012 (voc2012) development kit. Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep, 8.

Feng, S., Shen, C., Xia, N., Song, W., Fan, M. and Cowling, B.J. (2020). Rational use of face masks in the COVID-19 pandemic. The Lancet Respiratory Medicine, 8(5).

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4), pp.193–202.

Galbadage, T., Peterson, B.M. and Gunasekera, R.S. (2020). Does COVID-19 Spread Through Droplets Alone? [online] Available at: https://www.frontiersin.org/articles/10.3389/fpubh.2020.00163/full.

Girshick, R. (2015). Fast R-CNN.[online] arXiv.org. Available at: https://arxiv.org/abs/1504.08083

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2013). *Rich feature hierarchies for accurate object detection and semantic segmentation*. [online] arXiv.org. Available at: https://arxiv.org/abs/1311.2524.

Glaser, J.I., Benjamin, A.S., Farhoodi, R. and Kording, K.P., (2019). The roles of supervised machine learning in systems neuroscience. Progress in neurobiology, 175, pp.126-137.

Gomes, C.P. (2009). Computational Sustainability: Computational Methods for a Sustainable Environment, Economy, and Society. [online] Available at: chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://computational-sustainability.cis.cornell.edu/pdfs/Gomes-NAE-Bridge-Winter09.pdf.

Gupta, S., Arbeláez, P., Girshick, R. and Malik, J. (2014). Indoor Scene Understanding with RGB-D Images: Bottom-up Segmentation, Object Detection and Semantic Segmentation. International Journal of Computer Vision, [online] 112(2), pp.133–149. Available at: https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.pdf

Harper, L., Kalfa, N., Beckers, G.M.A., Kaefer, M., Nieuwhof-Leppink, A.J., Fossum, M., Herbst, K.W. and Bagli, D. (2020). The impact of COVID-19 on research. Journal of Pediatric Urology, [online] 16(5), pp.715–716. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7343645/

He, K., Zhang, X., Ren, S. and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. [online] openaccess.thecvf.com. Available at: https://openaccess.thecvf.com/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html

Holzinger, A. (2018). From Machine Learning to Explainable AI. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/8490530.

Hong, Z. (2011). A preliminary study on artificial neural network. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/6030344

Hsiao, E. and Hebert, M. (2014). Occlusion Reasoning for Object Detection under Arbitrary Viewpoint. IEEE Transactions on Pattern Analysis and Machine Intelligence, [online] 36(9), pp.1803–1815. Available at: https://ieeexplore.ieee.org/abstract/document/6727481

Hu, L. and Ge, Q. (2020). Automatic facial expression recognition based on MobileNetV2 in Real-time. Phys. Conf. Ser. 1549 022136. [online] Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1549/2/022136/meta.

Ingle, S. and Phute, M., (2016). Tesla autopilot: semi-autonomous driving, an uptick for future autonomy. International Research Journal of Engineering and Technology, 3(9), pp.369-372.

Jackson, S., Yaqub, M. and Li, C.X., (2019). The agile deployment of machine learning models in healthcare. Frontiers in Big Data, 1, p.7.

Jiang, H. and Learned-Miller, E. (2017). Face Detection with the Faster R-CNN. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/7961803

Jmour, N., Zayen, S. and Abdelkrim, A. (2018). Convolutional neural networks for image classification. 2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET).

Khanum, M., Mahboob, T., Imtiaz, W., Ghafoor, H.A. and Sehar, R., (2015). A survey on unsupervised machine learning algorithms for automation, classification and maintenance. International Journal of Computer Applications, 119(13).

Kim, J., Sung, J.-Y. and Park, S. (2020). Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition. 2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia).

Kornblith, S., Shlens, J. and Le, Q.V. (2019). Do Better ImageNet Models Transfer Better? [online] openaccess.thecvf.com. Available at: https://openaccess.thecvf.com/content_CVPR_2019/html/Kornblith_Do_Better_ImageNet_Models_Transfer_Better_CVPR_2019_paper.html

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. Nature, 521(7553), pp.436–444.

Lepelletier, D., Grandbastien, B., Romano-Bertrand, S., Aho, S., Chidiac, C., Géhanno, J.-F. and Chauvin, F. (2020). What face mask for what use in the context of COVID-19 pandemic? The French guidelines. Journal of Hospital Infection.

Li, H., Lin, Z., Shen, X., Brandt, J. and Hua, G. (2015). *A Convolutional Neural Network Cascade for Face Detection*. [online] openaccess.thecvf.com. Available at: https://openaccess.thecvf.com/content_cvpr_2015/html/Li_A_Convolutional_Neural_2015_CVPR_paper.html

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L. (2014). Microsoft COCO: Common Objects in Context. Computer Vision – ECCV 2014, [online] pp.740–755. Available at: https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A.C. (2016). SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016, [online] pp.21–37. Available at: https://arxiv.org/abs/1512.02325.
Liu, X. and Zhang, S. (2020). COVID-19 : Face Masks and Human-to-human Transmission. Influenza and Other Respiratory Viruses.

MacIntyre, C.R., Seale, H., Dung, T.C., Hien, N.T., Nga, P.T., Chughtai, A.A., Rahman, B., Dwyer, D.E. and Wang, Q. (2015). A cluster randomised trial of cloth masks compared with medical masks in

Han, D., Liu, Q. and Fan, W. (2018). A new image classification method using CNN transfer learning and web data augmentation. Expert Systems with Applications, 95, pp.43–56.

Healthcare workers. BMJ Open, [online] 5(4), p.e006577. Available at: https://bmjopen.bmj.com/content/5/4/e006577.

Mallapaty, S. (2020). Why does the coronavirus spread so easily between people? search.bvsalud.org. [online] Available at: https://search.bvsalud.org/global-literature-on-novel-coronavirus-2019-ncov/resource/en/covidwho-4897.

Matusiak, Ł., Szepietowska, M., Krajewski, P., Białynicki-Birula, R. and Szepietowski, J.C. (2020). The use of face masks during the COVID -19 pandemic in Poland: a survey study of 2315 young adults. Dermatologic Therapy.

Mikołajczyk, A. and Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/8388338

Oliva, A. and Torralba, A. (2007). The role of context in object recognition. Trends in Cognitive Sciences, [online] 11(12), pp.520–527. Available at: http://people.csail.mit.edu/torralba/publications/reviewPapers/TrendsInCog.pdf

Oquab, M., Bottou, L., Laptev, I. and Sivic, J. (2015). Is Object Localization for Free? - Weakly-Supervised Learning with Convolutional Neural Networks. [online] openaccess.thecvf.com. Available at:https://openaccess.thecvf.com/content_cvpr_2015/html/Oquab_Is_Object_Localization_2015_CVPR_paper.html

Perez, L. and Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. arXiv:1712.04621 [cs]. [online] Available at: https://arxiv.org/abs/1712.04621.

Pragholapati, A. (2020). COVID-19 IMPACT ON STUDENTS. [online] . Available at: https://edarxiv.org/895ed/.

Qian, M. and Jiang, J., 2020. COVID-19 and social distancing. Journal of Public Health, pp.1-3.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. [online] . Available at: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

Redmon, J. and Farhadi, A. (2018). YOLOv3: An Incremental Improvement. [online]. Available at: https://arxiv.org/pdf/1804.02767.pdf.

Ren, S., He, K., Girshick, R. and Sun, J. (2015).Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. [online] arXiv.org. Available at: https://arxiv.org/abs/1506.01497

Rice, L., Wong, E. and Kolter, Z. (2020). Overfitting in adversarially robust deep learning. [online] proceedings.mlr.press. Available at: http://proceedings.mlr.press/v119/rice20a.html

Rieger, M. (2020). To wear or not to wear? Factors influencing wearing face masks in Germany during the COVID-19 pandemic. Social Health and Behavior, 3(2), p.50.

Rosenbaum, L. (2020). The Untold Toll — The Pandemic's Effects on Patients without Covid-19. New England Journal of Medicine.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition

Challenge. International Journal of Computer Vision, [online] 115(3), pp.211–252. Available at: https://hci.stanford.edu/publications/2015/scenegraphs/imagenet-challenge.pdf

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.-C. (2019). MobileNetV2:Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs]

Schroff, F., Kalenichenko, D. and Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [online] Available at: https://ieeexplore.ieee.org/document/7298682

Shafiee, M., Chywl, C., Li, F. and Wong, C. (2017). *Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video*. [online] . Available at: https://arxiv.org/pdf/1709.05943.pdf.

Shetty, S. (2016). Application of Convolutional Neural Network for Image Classification on Pascal VOC Challenge 2012 dataset. *arXiv:1607.03785 [cs]*. [online] Available at: https://arxiv.org/abs/1607.03785

Srivastava, N., Mansimov, E. and Salakhudinov, R. (2015). Unsupervised Learning of Video Representations using LSTMs. [online] proceedings.mlr.press. Available at: http://proceedings.mlr.press/v37/srivastava15.html

Steinkraus, D., Buck, I. and Simard, P.Y. (2005). Using GPUs for machine learning algorithms. Eighth International Conference on Document Analysis and Recognition (ICDAR'05).

Torrey, L. and Shavlik, J. (2010). *Transfer Learning*. [online] Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. Available at: https://www.igi-global.com/chapter/transfer-learning/36988

Triantafyllidou, D. and Tefas, A. (2016). A Fast Deep Convolutional Neural Network for Face Detection in Big Visual Data. Advances in Big Data, pp.61–70.

Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. and Smeulders, A.W.M. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2), pp.154–171.

Wagstaff, K. (2012). Machine Learning that Matters. [online] arXiv.org. Available at: https://arxiv.org/abs/1206.4656.

Wang, M.W., Zhou, M.Y., Ji, G.H., Ye, L., Cheng, Y.R., Feng, Z.H. and Chen, J., 2020. Mask crisis during the COVID-19 outbreak. Eur Rev Med Pharmacol Sci, 24(6), pp.3397-3399.

Wilson, S.P., Dahyot, R., Cunningham, P. and Cord, M. (2008). Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval. [online] . Available at: https://b-ok.cc/book/2199293/04ffd9?id=2199293&secret=04ffd9

Womg, A., Shafiee, M.J., Li, F. and Chwyl, B. (2018). *Tiny SSD: A Tiny Single-Shot Detection Deep Convolutional Neural Network for Real-Time Embedded Object Detection*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/8575741.

Worby, C.J. and Chang, H.-H. (2020). Face mask use in the general population and optimal resource allocation during the COVID-19 pandemic. *Nature Communications*, [online] 11(1), p.4049. Available at: https://www.nature.com/articles/s41467-020-17922-x.

WHO (2021). *WHO COVID-19 dashboard*. [online] covid19.who.int. Available at: https://covid19.who.int/.

World Health Organisation (2020)

World Health Organisation (2020). *Rational use of personal protective equipment (PPE) for coronavirus disease (COVID-19)*. [online] . Available at: https://apps.who.int/iris/bitstream/handle/10665/331498/WHO-2019-nCoV-IPCPPE_use-2020.2-eng.pdf.

Wu, H., Huang, J., Zhang, C.J.P., He, Z. and Ming, W.-K. (2020). Facemask shortage and the novel coronavirus disease (COVID-19) outbreak: Reflections on public health measures. *EClinicalMedicine*, [online] 0(0). Available at: https://www.thelancet.com/journals/eclinm/article/PIIS2589-5370(20)30073-0/fulltext

Yang, J., He, W.Y., Zhang, T.L., Zhang, C.L., Zeng, L. and Nan, B.F. (2020). Research on subway pedestrian detection algorithms based on SSD model. IET Intelligent Transport Systems, 14(11), pp.1491–1496.

Yang, Y., Shang, W. and Rao, X. (2020). Facing the COVID-19 outbreak: What should we know and what could we do? Journal of Medical Virology.

Yin, W., Kann, K., Yu, M. and Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. [online] arXiv.org. Available at: https://arxiv.org/abs/1702.01923.

Zhang, Y. (2010). New Advances in Machine Learning. [online] Google Books. BoD – Books on Demand. Available at: https://books.google.co.uk/books?hl=en&lr=&id=XAqhDwAAQBAJ&oi=fnd&pg=PA19&dq=types+of+machine+learning&ots=r2GqdVCmLp&sig=yVdIJhFldCS3_dqckTx10_Mnc2Y&redir_esc=y#v=onepage&q=types%20of%20machine%20learning&f=false

APPENDIX

Appendix 1 (TensorFlow API)

Note: "AR" refers to "mAR", Mean Average Recall.

A1 "Initial Baseline" Experiment Results
Table 1

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.458
mAP (Large) (IOU=0.50:0.95)	0.474
mAP (Medium) (IOU=0.50:0.95)	0.333
mAP (Small) (IOU=0.50:0.95)	0.000
mAP @.50IOU	0.870
mAP @.75IOU	0.438
AR@1 (IOU=0.50:0.95)	0.477
AR@10 (IOU=0.50:0.95)	0.599
AR@100 (IOU=0.50:0.95)	0.626
AR@100 (Large) (IOU=0.50:0.95)	0.634
AR@100 (Medium) (IOU=0.50:0.95)	0.522
AR@100 (Small)	0.000
Validation Loss	0.618
Total Training Loss	0.418

A1 "Initial Baseline Part 2" Experiment Results
Table 2

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.524
mAP (Large) (IOU=0.50:0.95)	0.530
mAP (Medium) (IOU=0.50:0.95)	0.452
mAP (Small) (IOU=0.50:0.95)	0.000
mAP @.50IOU	0.951
mAP @.75IOU	0.499
AR@1 (IOU=0.50:0.95)	0.479
AR@10 (IOU=0.50:0.95)	0.611
AR@100 (IOU=0.50:0.95)	0.626
AR@100 (Large) (IOU=0.50:0.95)	0.631
AR@100 (Medium) (IOU=0.50:0.95)	0.571
AR@100 (Small)	0.000
Validation Loss	0.515
Total Training Loss	0.297

A1 “No Mask Only” Experiment Results
Table 3

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.608
mAP (Large) (IOU=0.50:0.95)	0.668
mAP (Medium) (IOU=0.50:0.95)	0.510
mAP (Small) (IOU=0.50:0.95)	0.213
mAP @.50IOU	0.968
mAP @.75IOU	0.701
AR@1 (IOU=0.50:0.95)	0.538
AR@10 (IOU=0.50:0.95)	0.608
AR@100 (IOU=0.50:0.95)	0.693
AR@100 (Large) (IOU=0.50:0.95)	0.748
AR@100 (Medium) (IOU=0.50:0.95)	0.613
AR@100 (Small)	0.416
Validation Loss	0.271
Total Training Loss	0.421

A1 “Face Mask Addition” Experiment Results
Table 4

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.628
mAP (Large) (IOU=0.50:0.95)	0.686
mAP (Medium) (IOU=0.50:0.95)	0.524
mAP (Small) (IOU=0.50:0.95)	0.161
mAP @.50IOU	0.967
mAP @.75IOU	0.736
AR@1 (IOU=0.50:0.95)	0.546
AR@10 (IOU=0.50:0.95)	0.700
AR@100 (IOU=0.50:0.95)	0.713
AR@100 (Large) (IOU=0.50:0.95)	0.763
AR@100 (Medium) (IOU=0.50:0.95)	0.630
AR@100 (Small)	0.447
Validation Loss	0.406
Total Training Loss	0.368

A1 “Glasses” Experiment Results
Table 5

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.618
mAP (Large) (IOU=0.50:0.95)	0.677
mAP (Medium) (IOU=0.50:0.95)	0.626
mAP (Small) (IOU=0.50:0.95)	0.197
mAP @.50IOU	0.962
mAP @.75IOU	0.735
AR@1 (IOU=0.50:0.95)	0.539
AR@10 (IOU=0.50:0.95)	0.691
AR@100 (IOU=0.50:0.95)	0.705
AR@100 (Large) (IOU=0.50:0.95)	0.753
AR@100 (Medium) (IOU=0.50:0.95)	0.626
AR@100 (Small)	0.447
Validation Loss	0.405
Total Training Loss	0.413

A1 “Incorrect Mask” Experiment Results
Table 6

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.627
mAP (Large) (IOU=0.50:0.95)	0.681
mAP (Medium) (IOU=0.50:0.95)	0.519
mAP (Small) (IOU=0.50:0.95)	0.124
mAP @.50IOU	0.961
mAP @.75IOU	0.771
AR@1 (IOU=0.50:0.95)	0.681
AR@10 (IOU=0.50:0.95)	0.723
AR@100 (IOU=0.50:0.95)	0.730
AR@100 (Large) (IOU=0.50:0.95)	0.765
AR@100 (Medium) (IOU=0.50:0.95)	0.620
AR@100 (Small)	0.432
Validation Loss	0.429
Total Training Loss	0.303

A1 “Dataset Rework” Experiment Results
Table 7

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.830
mAP (Large) (IOU=0.50:0.95)	0.830
mAP (Medium) (IOU=0.50:0.95)	-1.000
mAP (Small) (IOU=0.50:0.95)	-1.000
mAP @.50IOU	1.000
mAP @.75IOU	0.995
AR@1 (IOU=0.50:0.95)	0.865
AR@10 (IOU=0.50:0.95)	0.866
AR@100 (IOU=0.50:0.95)	0.866
AR@100 (Large) (IOU=0.50:0.95)	0.866
AR@100 (Medium) (IOU=0.50:0.95)	-1.000
AR@100 (Small)	-1.000
Validation Loss	0.232
Total Training Loss	0.201

A1 “Training Medium and Small Area Objects” Experiment Results
Table 8

COCO Metrics	Model Performance
mAP (IOU=0.50:0.95)	0.658
mAP (Large) (IOU=0.50:0.95)	0.819
mAP (Medium) (IOU=0.50:0.95)	0.467
mAP (Small) (IOU=0.50:0.95)	0.483
mAP @.50IOU	0.936
mAP @.75IOU	0.724
AR@1 (IOU=0.50:0.95)	0.556
AR@10 (IOU=0.50:0.95)	0.710
AR@100 (IOU=0.50:0.95)	0.730
AR@100 (Large) (IOU=0.50:0.95)	0.867
AR@100 (Medium) (IOU=0.50:0.95)	0.584
AR@100 (Small)	0.483
Validation Loss	0.326
Total Training Loss	0.209

Appendix 2 (TensorFlow API – Test Images)

Note: The images in this appendix are only a few samples that belong to the respective experiment. The total amount of test images returned depends on how much data was used for the experiment and how that data had been partitioned during the pre-processing stage.

A2 "Initial Baseline" Test Image Results

Figure A2.1: Image1 Predicted Label(s)

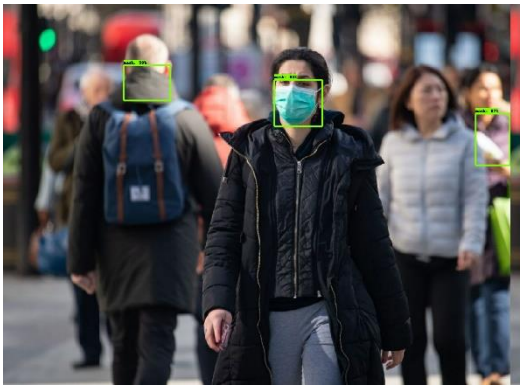

Figure A2.3: Image2 Predicted Label(s)

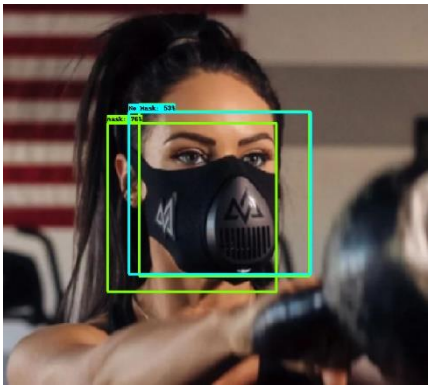

Figure A2.4: Image3 Predicted Label(s)

Figure A2.2: Image1 Ground Truth Label(s)

Figure A2.3 Image2: Ground Truth Label(s)

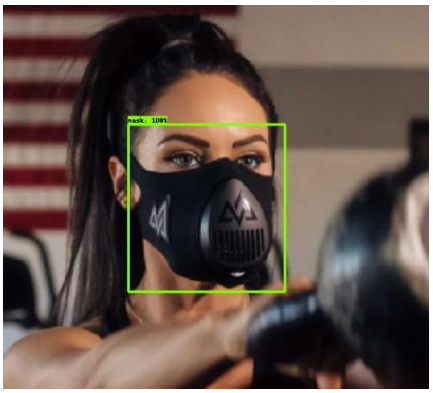

Figure A2.6: Image3 Ground Truth

A2 “Initial Baseline 2” Test Image Results

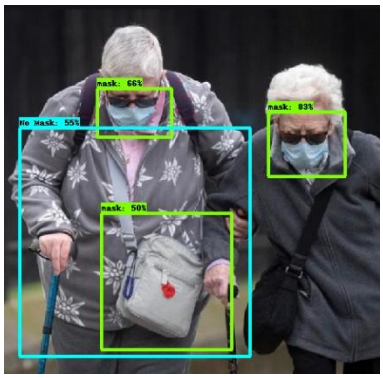

Figure A2.7: Image1 Predicted Label(s)

Figure A2.9: Image2 Predicted Label(s)

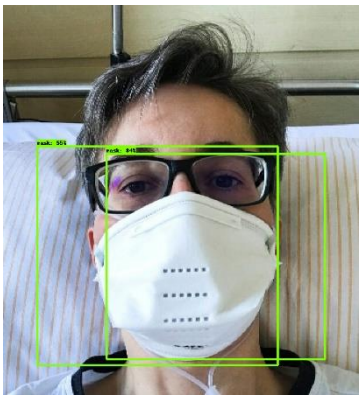

Figure A2.11: Image3 Predicted Label(s)

Figure A2.8: Image1 Ground Truth Label(s)

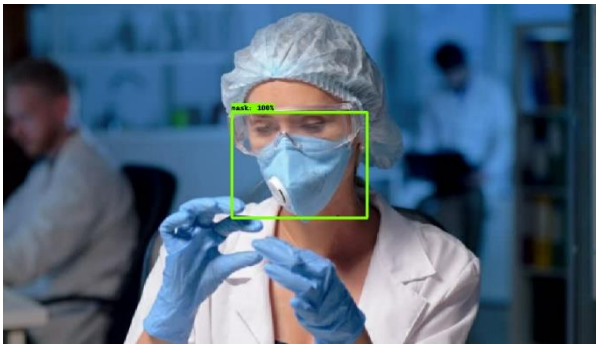

Figure A2.10: Image2 Ground Truth

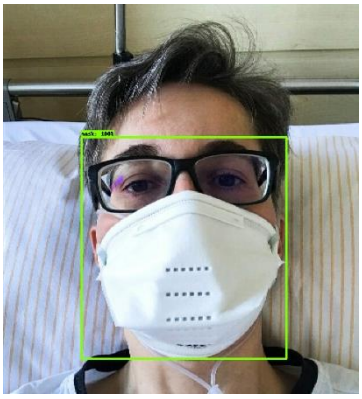

Figure A2.12: Image3 Ground Truth Label(s)

A2 “No Mask Only” Test Image Results

Figure A2.13: Image1 Predicted Label(s)

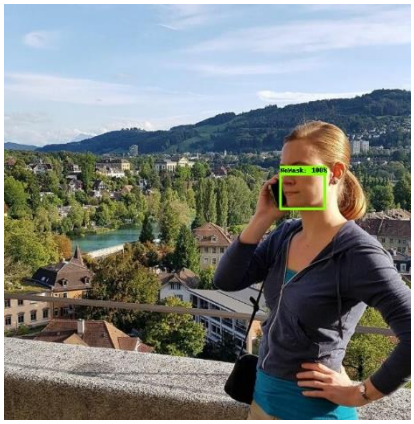

Figure A2.14: Image1 Ground Truth Label(s)

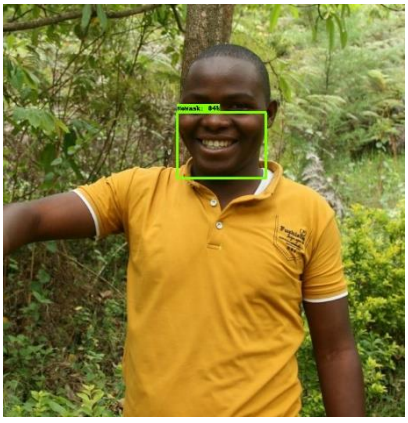

Figure A2.15: Image2 Predicted Label(s)

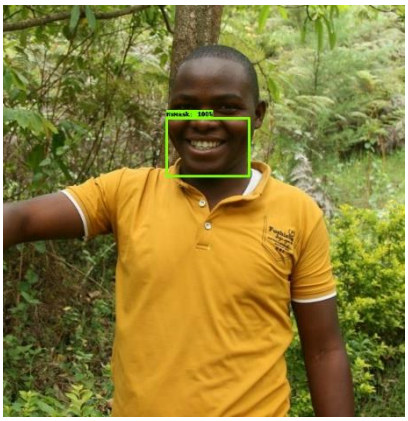

Figure A2.16: Image2 Ground Truth Label(s)

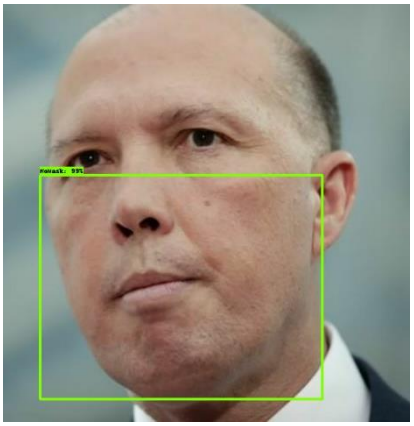

Figure A2.17: Image3 Predicted Label(s)

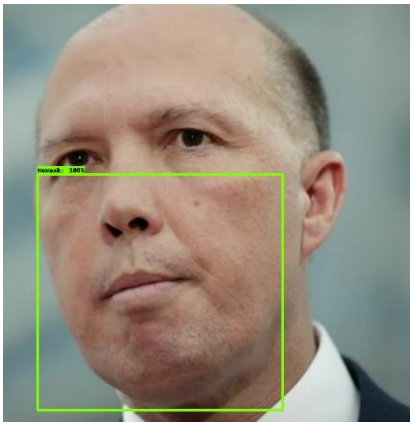

Figure A2.18: Image3 Ground Truth Label(s)

A2 “Face Mask Addition” Test Image Results

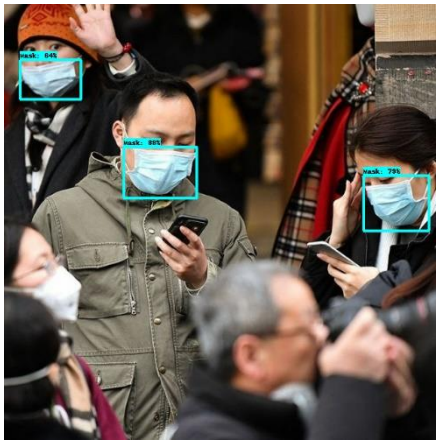

Figure A2.19: Image1 Predicted Label(s)

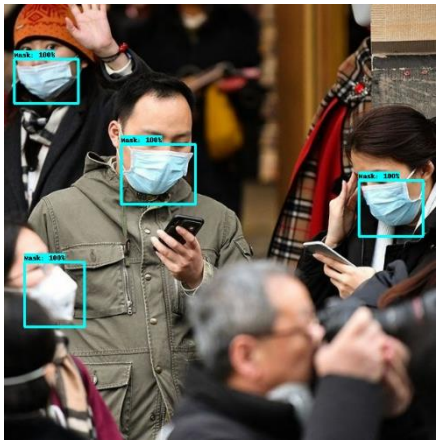

Figure A2.20: Image1 Ground Truth Label(s)

Figure A2.21: Imag21 Predicted Label(s)

Figure A2.22: Image2 Ground Truth Label(s)

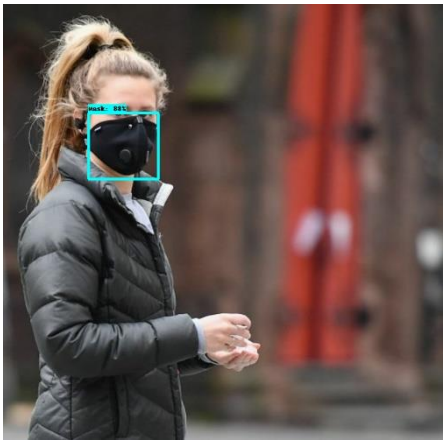

Figure A2.23: Image3 Predicted Label(s)

Figure A2.24: Image3 Ground Truth Label(s)

A2 “Glasses” Test Image Results

Figure A2.25: Image1 Predicted Label(s)

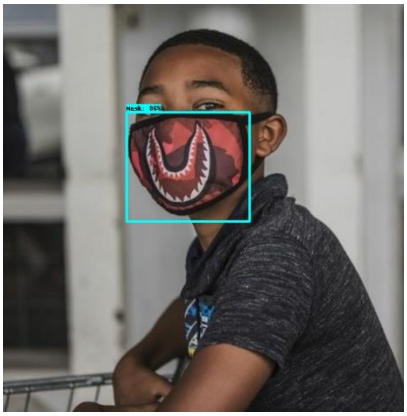

Figure A2.27: Image2 Predicted Label(s)

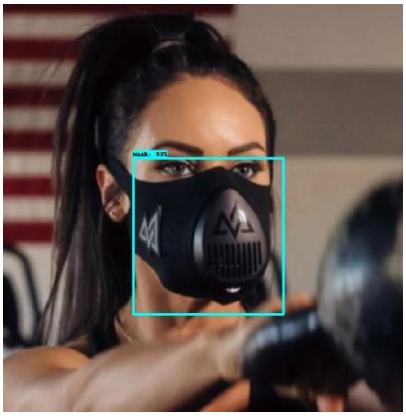

Figure A2.29: Image3 Predicted Label(s)

Figure A2.26: Image1 Ground Truth Label(s)

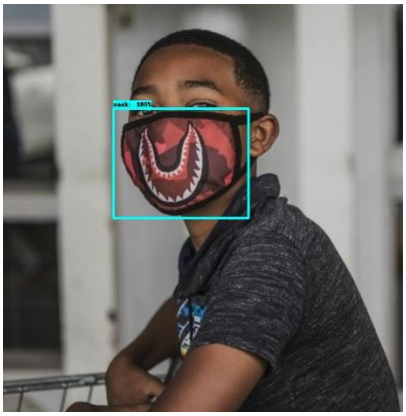

Figure A2.28: Image2 Ground Truth Label(s)

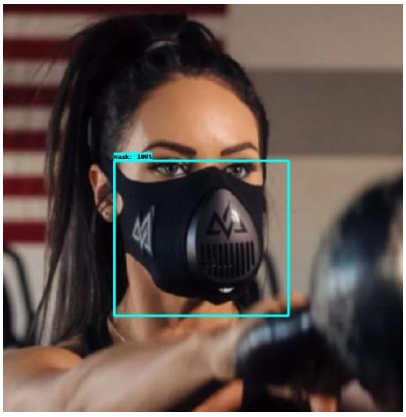

Figure A2.30: Image3 Ground Truth Label(s)

A2 “Incorrect Mask” Test Image Results

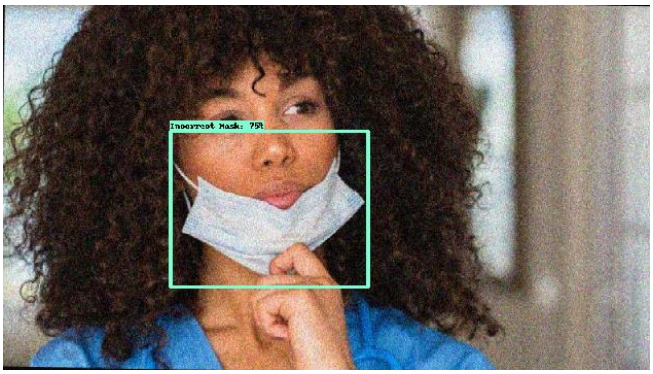

Figure A2.31: Image1 Predicted Label(s)

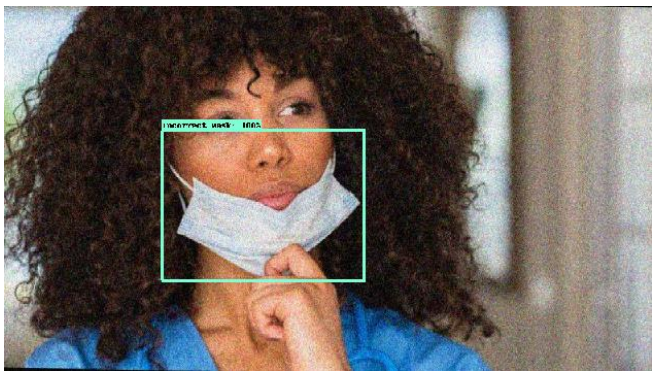

Figure A2.32: Image1 Ground Truth Label(s)

Figure A2.33: Image2 Predicted Label(s)

Figure A2.34: Image2 Ground Truth Label(s)

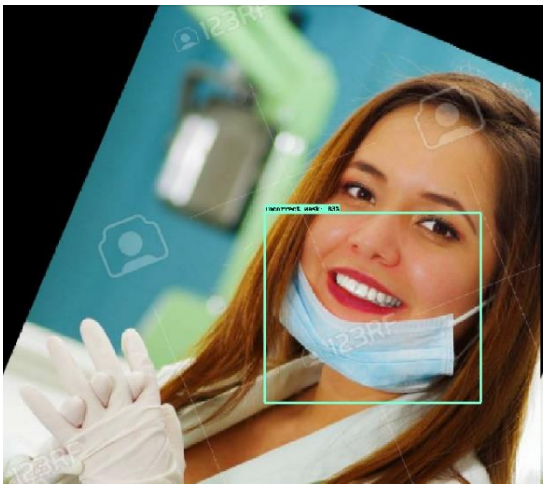

Figure A2.35: Image3 Predicted Label(s)

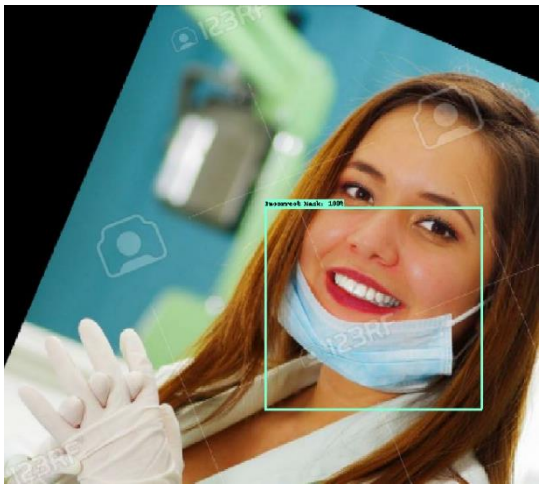

Figure A2.36: Image3 Ground Truth Label(s)

A2 “Dataset Rework” Test Image Results

Figure A2.37: Image1 Predicted Label(s)

Figure A2.39: Image2 Predicted Label(s)

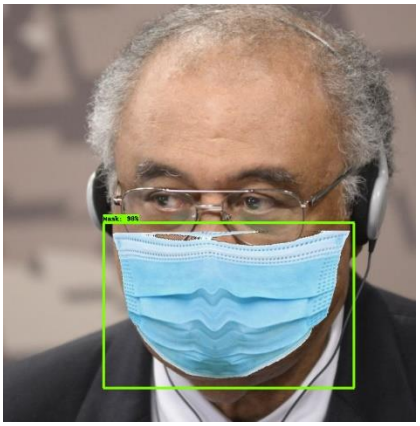

Figure A2.41: Image3 Predicted Label(s)

Figure A2.38: Image1 Ground Truth Label(s)

Figure A2.40: Image2 Ground Truth Label(s)

Figure A2.42: Image3 Ground Truth Label(s)

A2 “Training Medium and Small Area Objects” Test Image Results

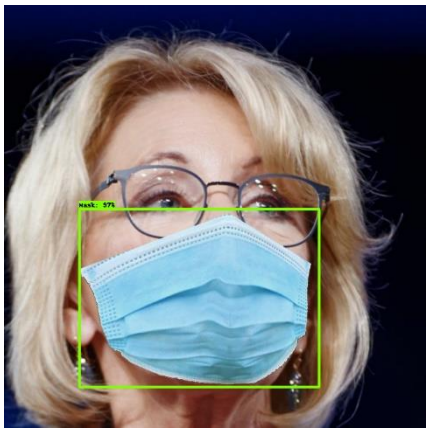

Figure A2.43: Image1 Predicted Label(s)

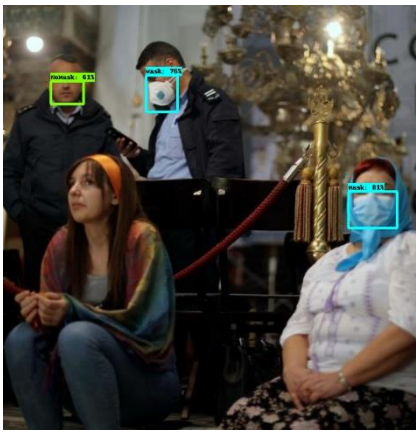

Figure A2.45: Image2 Predicted Label(s)

Figure A2.47: Image3 Predicted Label(s)

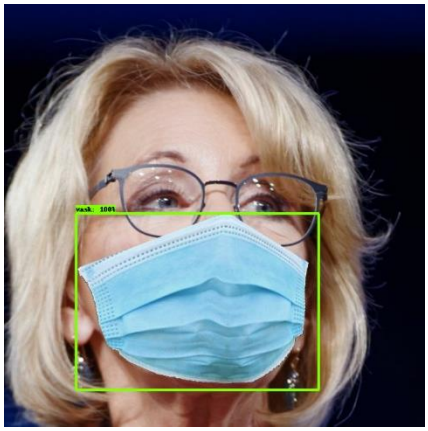

Figure A2.44: Image1 Ground Truth Label(s)

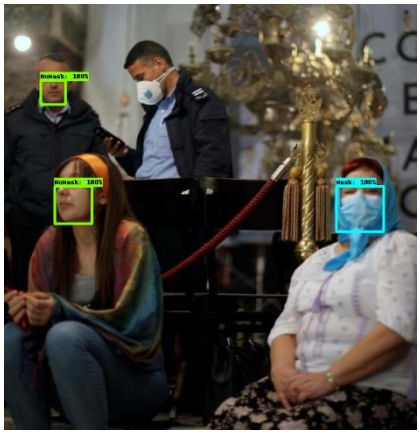

Figure A2.46: Image2 Ground Truth Label(s)

Figure A2.48: Image3 Ground Truth Label(s)

Appendix 3 (Transfer Learning Using Keras)

“Face Classifier + Face Mask Classifier” Experiment Results

Fold 1

```
Training for fold: 1 ...
Epoch 1/10
83/83 [==============================] - 56s 679ms/step - loss: 0.4379 -
accuracy: 0.8104 - val_loss: 0.1239 - val_accuracy: 0.9640
Epoch 2/10
83/83 [==============================] - 49s 590ms/step - loss: 0.1263 -
accuracy: 0.9602 - val_loss: 0.0655 - val_accuracy: 0.9760
Epoch 3/10
83/83 [==============================] - 49s 589ms/step - loss: 0.0763 -
accuracy: 0.9764 - val_loss: 0.0574 - val_accuracy: 0.9800
Epoch 4/10
83/83 [==============================] - 49s 587ms/step - loss: 0.0572 -
accuracy: 0.9818 - val_loss: 0.0439 - val_accuracy: 0.9860
Epoch 5/10
83/83 [==============================] - 49s 589ms/step - loss: 0.0536 -
accuracy: 0.9818 - val_loss: 0.0406 - val_accuracy: 0.9880
Epoch 6/10
83/83 [==============================] - 51s 610ms/step - loss: 0.0394 -
accuracy: 0.9858 - val_loss: 0.0390 - val_accuracy: 0.9880
Epoch 7/10
83/83 [==============================] - 51s 609ms/step - loss: 0.0313 -
accuracy: 0.9879 - val_loss: 0.0363 - val_accuracy: 0.9860
Epoch 8/10
83/83 [==============================] - 48s 572ms/step - loss: 0.0324 -
accuracy: 0.9906 - val_loss: 0.0413 - val_accuracy: 0.9840
Epoch 9/10
83/83 [==============================] - 47s 571ms/step - loss: 0.0252 -
accuracy: 0.9946 - val_loss: 0.0350 - val_accuracy: 0.9880
Epoch 10/10
83/83 [==============================] - 48s 572ms/step - loss: 0.0283 -
accuracy: 0.9933 - val_loss: 0.0351 - val_accuracy: 0.9880
Score for fold 1: loss of 0.009963836520910263; accuracy of 100.0%
------------------------------------------------------------------------
```

Fold 2

```
Training for fold: 2 ...
Epoch 1/10
83/83 [==============================] - 48s 584ms/step - loss: 0.3501 -
accuracy: 0.8387 - val_loss: 0.1070 - val_accuracy: 0.9740
Epoch 2/10
83/83 [==============================] - 48s 579ms/step - loss: 0.0998 -
accuracy: 0.9690 - val_loss: 0.0692 - val_accuracy: 0.9780
Epoch 3/10
83/83 [==============================] - 48s 575ms/step - loss: 0.0652 -
accuracy: 0.9811 - val_loss: 0.0593 - val_accuracy: 0.9760
Epoch 4/10
83/83 [==============================] - 48s 576ms/step - loss: 0.0543 -
accuracy: 0.9811 - val_loss: 0.0511 - val_accuracy: 0.9780
Epoch 5/10
83/83 [==============================] - 48s 574ms/step - loss: 0.0422 -
accuracy: 0.9852 - val_loss: 0.0515 - val_accuracy: 0.9800
Epoch 6/10
83/83 [==============================] - 48s 574ms/step - loss: 0.0347 -
accuracy: 0.9892 - val_loss: 0.0437 - val_accuracy: 0.9780
Epoch 7/10
83/83 [==============================] - 48s 575ms/step - loss: 0.0386 -
accuracy: 0.9906 - val_loss: 0.0429 - val_accuracy: 0.9800
Epoch 8/10
83/83 [==============================] - 48s 573ms/step - loss: 0.0336 -
accuracy: 0.9906 - val_loss: 0.0421 - val_accuracy: 0.9820
Epoch 9/10
83/83 [==============================] - 49s 586ms/step - loss: 0.0255 -
accuracy: 0.9933 - val_loss: 0.0422 - val_accuracy: 0.9840
Epoch 10/10
83/83 [==============================] - 49s 589ms/step - loss: 0.0285 -
accuracy: 0.9906 - val_loss: 0.0409 - val_accuracy: 0.9840
Score for fold 2: loss of 0.009603191167116165; accuracy of
99.75000023841858%
------------------------------------------------------------------------
```

Fold 3

```
Training for fold: 3 ...
Epoch 1/10
83/83 [==============================] - 49s 591ms/step - loss: 0.3770 -
accuracy: 0.8367 - val_loss: 0.1133 - val_accuracy: 0.9700
Epoch 2/10
83/83 [==============================] - 48s 577ms/step - loss: 0.1373 -
accuracy: 0.9521 - val_loss: 0.0686 - val_accuracy: 0.9760
Epoch 3/10
83/83 [==============================] - 48s 577ms/step - loss: 0.0813 -
accuracy: 0.9744 - val_loss: 0.0530 - val_accuracy: 0.9800
Epoch 4/10
83/83 [==============================] - 48s 577ms/step - loss: 0.0483 -
accuracy: 0.9872 - val_loss: 0.0473 - val_accuracy: 0.9780
Epoch 5/10
83/83 [==============================] - 48s 582ms/step - loss: 0.0557 -
accuracy: 0.9811 - val_loss: 0.0417 - val_accuracy: 0.9840
Epoch 6/10
83/83 [==============================] - 48s 584ms/step - loss: 0.0383 -
accuracy: 0.9885 - val_loss: 0.0387 - val_accuracy: 0.9860
Epoch 7/10
83/83 [==============================] - 48s 578ms/step - loss: 0.0349 -
accuracy: 0.9872 - val_loss: 0.0381 - val_accuracy: 0.9860
Epoch 8/10
83/83 [==============================] - 48s 576ms/step - loss: 0.0282 -
accuracy: 0.9912 - val_loss: 0.0370 - val_accuracy: 0.9860
Epoch 9/10
83/83 [==============================] - 48s 578ms/step - loss: 0.0272 -
accuracy: 0.9906 - val_loss: 0.0450 - val_accuracy: 0.9780
Epoch 10/10
83/83 [==============================] - 48s 582ms/step - loss: 0.0272 -
accuracy: 0.9906 - val_loss: 0.0413 - val_accuracy: 0.9820
Score for fold 3: loss of 0.020815784111618996; accuracy of
99.00000095367432%
------------------------------------------------------------------------
```

Fold 4

```
Training for fold: 4 ...
Epoch 1/10
83/83 [==============================] - 50s 603ms/step - loss: 0.3895 -
accuracy: 0.8225 - val_loss: 0.1268 - val_accuracy: 0.9680
Epoch 2/10
83/83 [==============================] - 48s 581ms/step - loss: 0.1135 -
accuracy: 0.9676 - val_loss: 0.0743 - val_accuracy: 0.9800
Epoch 3/10
83/83 [==============================] - 48s 582ms/step - loss: 0.0819 -
accuracy: 0.9750 - val_loss: 0.0563 - val_accuracy: 0.9840
Epoch 4/10
83/83 [==============================] - 48s 580ms/step - loss: 0.0674 -
accuracy: 0.9784 - val_loss: 0.0567 - val_accuracy: 0.9800
Epoch 5/10
83/83 [==============================] - 48s 580ms/step - loss: 0.0578 -
accuracy: 0.9777 - val_loss: 0.0466 - val_accuracy: 0.9800
Epoch 6/10
83/83 [==============================] - 48s 579ms/step - loss: 0.0358 -
accuracy: 0.9885 - val_loss: 0.0422 - val_accuracy: 0.9820
Epoch 7/10
83/83 [==============================] - 49s 593ms/step - loss: 0.0347 -
accuracy: 0.9899 - val_loss: 0.0400 - val_accuracy: 0.9820
Epoch 8/10
83/83 [==============================] - 49s 588ms/step - loss: 0.0293 -
accuracy: 0.9906 - val_loss: 0.0387 - val_accuracy: 0.9820
Epoch 9/10
83/83 [==============================] - 49s 590ms/step - loss: 0.0238 -
accuracy: 0.9933 - val_loss: 0.0399 - val_accuracy: 0.9820
Epoch 10/10
83/83 [==============================] - 49s 589ms/step - loss: 0.0276 -
accuracy: 0.9919 - val_loss: 0.0382 - val_accuracy: 0.9840
Score for fold 4: loss of 0.028497526422142982; accuracy of
98.50000143051147%
------------------------------------------------------------------------
```

Fold 5

```
Training for fold: 5 ...
Epoch 1/10
83/83 [==============================] - 50s 600ms/step - loss: 0.3441 -
accuracy: 0.8495 - val_loss: 0.0968 - val_accuracy: 0.9640
Epoch 2/10
83/83 [==============================] - 50s 600ms/step - loss: 0.1083 -
accuracy: 0.9642 - val_loss: 0.0697 - val_accuracy: 0.9740
Epoch 3/10
83/83 [==============================] - 49s 591ms/step - loss: 0.0668 -
accuracy: 0.9818 - val_loss: 0.0564 - val_accuracy: 0.9720
Epoch 4/10
83/83 [==============================] - 49s 587ms/step - loss: 0.0624 -
accuracy: 0.9777 - val_loss: 0.0504 - val_accuracy: 0.9760
Epoch 5/10
83/83 [==============================] - 49s 588ms/step - loss: 0.0461 -
accuracy: 0.9865 - val_loss: 0.0488 - val_accuracy: 0.9800
Epoch 6/10
83/83 [==============================] - 49s 588ms/step - loss: 0.0424 -
accuracy: 0.9838 - val_loss: 0.0494 - val_accuracy: 0.9800
Epoch 7/10
83/83 [==============================] - 49s 593ms/step - loss: 0.0334 -
accuracy: 0.9899 - val_loss: 0.0435 - val_accuracy: 0.9800
Epoch 8/10
83/83 [==============================] - 49s 589ms/step - loss: 0.0363 -
accuracy: 0.9899 - val_loss: 0.0450 - val_accuracy: 0.9800
Epoch 9/10
83/83 [==============================] - 49s 587ms/step - loss: 0.0297 -
accuracy: 0.9885 - val_loss: 0.0532 - val_accuracy: 0.9800
Epoch 10/10
83/83 [==============================] - 49s 585ms/step - loss: 0.0254 -
accuracy: 0.9919 - val_loss: 0.0446 - val_accuracy: 0.9800
Score for fold 5: loss of 0.020945753902196884; accuracy of
99.50000047683716%
------------------------------------------------------------------------
```

Score Per Fold (Final Scores)

```
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 0.009963836520910263 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.009603191167116165 - Accuracy: 99.75000023841858%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.020815784111618996 - Accuracy: 99.00000095367432%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.028497526422142982 - Accuracy: 98.50000143051147%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.020945753902196884 - Accuracy: 99.50000047683716%
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 99.3500006198883 (+- 0.5385159671441138)
> Loss: 0.017965218424797057
------------------------------------------------------------------------
```