YOLO Application in COVID-19 Use Cases

Final Report

Srijak Sengupta
3035437032

Supervisor: Dr. Luo Ping

18th April 2021
ABSTRACT

The sudden outbreak of COVID-19 has made 2020-21 extremely challenging for doctors, researchers and the general public alike. The world is still behind in terms of Coronavirus research and this has had a negative impact on public health and the global economy. The adoption of the use of state-of-the-art object detection models like YOLO is required to improve virus detection speeds and disease spread control. To serve this purpose, this project explores training of a Darknet YOLO model and building detectors capable of being applied in COVID-19 case studies like coronavirus detection from microscopic visual output and face mask detection. We started off the project implementation by creating 2 datasets for the selected use cases – Coronavirus Detection Dataset (made up of 750 images belonging to 3 main classes) and Facemask Detection Dataset (consists of 1400 images of 4 types belonging to 3 main classes). After successful data labelling and initial YOLO architecture setup (with the help of Python based helper scripts), we built and trained the custom models for each use case by leveraging the GPU based runtime provided by Google Colab. In terms of testing and evaluation results, the Coronavirus Detection model and the Facemask Detection model yielded mAP score highs of 98.07% and 91.05% respectively. The saved best weights for each of these models were then used to build detectors capable of handling image, video and real-time video inputs. A future experiment suggested for the Coronavirus Detection model is to feed microscopic video output directly into the model in order to get a better gauge of its’ real-world performance. With regards to the Facemask Detection model, a future trial proposed is to train it on a larger dataset having higher variability between the images and a larger number of specific classes which can definitively pinpoint the mask status at one glance.
ACKNOWLEDGEMENTS

I would first like to thank my supervisor Dr. Luo Ping for giving me the opportunity to take up this project and for his constant support during the project execution.

I would also like to thank Miss Mable Choi for taking out the time to assist me in formulating this report and for enlightening me on the various aspects of technical report writing.
# TABLE OF CONTENTS

**ABSTRACT**

i

**ACKNOWLEDGEMENTS**

ii

**LIST OF FIGURES**

v

**LIST OF TABLES**

v

**ABBREVIATIONS**

vi

1 **INTRODUCTION / BACKGROUND**

1.1 Overview of the COVID-19 crisis ........................................... 1
1.2 Conventional Object Detection models ........................................ 1
1.3 You Only Look Once (YOLO) .................................................. 2
1.4 Project Objectives ....................................................................... 3
1.5 Project Contribution ................................................................. 4
1.6 Outline of the Report ............................................................... 5

2 **METHODOLOGY**

2.1 Introduction .................................................................................. 6
2.2 Overview of YOLO Architecture .................................................... 6
2.3 Dataset Creation ........................................................................... 8
2.3.1 Coronavirus Detection Dataset ............................................... 8
2.3.2 Facemask Detection Dataset .................................................... 11
2.4 Dataset Augmentation and Pre-processing ........................................ 14
2.5 Data Labelling .............................................................................. 15
2.6 Architecture Setup ....................................................................... 16
2.6.1 Helpers .................................................................................. 18
2.7 Training the Custom Models .......................................................... 19
2.8 Summary ..................................................................................... 21

3 **RESULTS & APPLICATIONS**

3.1 Introduction .................................................................................. 23
3.2 Testing and Evaluation .................................................................. 23
3.2.1 Evaluation Metrics .................................................................. 23
3.2.2 Inferences from training the custom models ......................... 24
3.2.3 Evaluating the models ........................................................... 26
3.3 Building Detectors ....................................................................... 28
3.4 Summary ..................................................................................... 31
LIST OF FIGURES

1.1 YOLO concept ................................................................. 2
2.1 Basic CNN architecture ...................................................... 6
2.2 Basic Object Detector architecture ...................................... 7
2.3 Instances of Cell Type 2 ...................................................... 9
2.4 Instances of Cell Type 3 ..................................................... 10
2.5 Instances of Coronavirus ................................................... 11
2.6 OK Mask image ............................................................... 12
2.7 NO Mask image ............................................................... 12
2.8 NOT OK Mask image ......................................................... 13
2.9 Multi class image ............................................................. 13
2.10 Image and it’s labelling output .......................................... 15
3.1 Training chart linked to the Coronavirus Detection model ...... 24
3.2 Training chart linked to the Facemask Detection model ........... 25
3.3 Detector predictions for Coronavirus Detection Dataset images 29
3.4 Detector predictions for Facemask Detection Dataset images .... 30

LIST OF TABLES

3.1 AP values for classes in the Coronavirus Detection Dataset ..... 26
3.2 Evaluation results for the Coronavirus Detection model ........... 27
3.3 AP values for classes in the Facemask Detection Dataset ........ 27
3.4 Evaluation results for the Coronavirus Detection model ........... 28
### ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>CSP</td>
<td>Cross-Stage-Partial-connection</td>
</tr>
<tr>
<td>DPM</td>
<td>Deformable parts model</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>IOU</td>
<td>Intersection over Union</td>
</tr>
<tr>
<td>mAP</td>
<td>Mean Average Precision</td>
</tr>
<tr>
<td>PAN</td>
<td>Path aggregation network</td>
</tr>
<tr>
<td>R-CNN</td>
<td>Region-based Convolutional Neural Network</td>
</tr>
<tr>
<td>ReLu</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>RNA</td>
<td>Ribonucleic acid</td>
</tr>
<tr>
<td>SARS-COV-2</td>
<td>Severe acute respiratory syndrome Coronavirus 2</td>
</tr>
<tr>
<td>SPP</td>
<td>Spatial Pyramid Pooling</td>
</tr>
<tr>
<td>YOLO</td>
<td>You Only Look Once</td>
</tr>
</tbody>
</table>
1 INTRODUCTION / BACKGROUND

1.1 Overview of the COVID-19 crisis

2020-21 has been an extremely testing time due to the widespread outbreak of COVID-19 and it has posed new challenges to doctors all around the world. This disease is caused by the novel Coronavirus, formally termed as SARS-COV-2 (Severe acute respiratory syndrome Coronavirus 2) and it made its first appearance in Wuhan, China at the end of December 2019 [1]. It was known that Coronavirus can affect animals, but the zoonotic nature observed in this new strain of Coronavirus made it transferrable to humans as well [2].

The pandemic has brought the world to a standstill with the economy and public health taking the biggest hit. Due to the sudden spread of the virus, the world is still behind in terms of research and trials of contemporary technology towards curbing its spread. Migrating from the use of more conventional object detection models to state-of-the-art models is needed in order to bring in much needed speed and accuracy in Coronavirus detection, study and spread control.

1.2 Conventional Object Detection models

Conventional object detection models consist of fairly complex architectures. Models like Deformable parts model (DPM) use classifiers which are evaluated at uniformly spread-out positions across the input image in order to perform detection.

Other object detection models like Region-based Convolutional Neural Networks (R-CNN) evaluate a classifier after predicting bounding boxes for the objects in the input image. It also does another estimation of the bounding boxes after classification of the detected objects, making it even more complicated [3].

There is a need for powerful models which are reasonably less complex to maintain in the Covid-19 research domain. Google’s You Only Look Once (YOLO) object
detection model which uses a simpler regression-based methodology can be adopted to achieve this objective.

1.3 You Only Look Once (YOLO)

Google’s You Only Look Once (YOLO) is a state-of-the-art real-time object detection model which applies a single neural network to input images. The network divides the image into multiple regions or grid cells, where each region individually tries to detect a specific object as seen in part (i) of Fig. 1.1 below.

(i)  (ii)  (iii)

![Fig 1.1: YOLO concept](image)

YOLO is a simplified approach to object detection where it considers the problem as a regression problem and predicts bounding boxes (as seen in part (ii) of Fig. 1.1) and linked confidence scores for each region and calculates conditional class probabilities for each object it outlines. Here, the confidence score is related to the bounding box and it measures the bounding box accuracy in terms of the likelihood of it containing a classifiable object. Conditional class probability is the measure of how sure the model is that the detected object within the bounding box belongs to a particular class. YOLO relays a final output of the image with the bounding boxes having the highest confidence scores and assigns a class to each bounding box based on which one has
the highest conditional class probability, as seen in part (iii) of Fig. 1.1. The later versions of YOLO also display the class confidence scores of the detected object within the bounding boxes. Class confidence scores are calculated by multiplying the confidence score of the bounding boxes and the conditional class probability of the object detected within that bounding box [4].

YOLO is built using the Darknet framework and uses a single Convolutional Neural Network (CNN) to view complete images during training and testing unlike DPM and R-CNN. This helps YOLO store integral data regarding classification in the context of entire images, making it fast and better at generalisation [3]. Despite having a higher localisation error (difference between true position and estimated position of an object) than R-CNNs and low success with closely packed objects, YOLO’s relatively simpler architecture and better generalisation ability makes it a good option for application to COVID-19 use cases.

1.4 Project Objectives

The objective of this project is to attempt to train a Darknet YOLO model and build detectors capable of being applied in COVID-19 related case studies. The training of the model is done with the help of custom datasets created for the selected use cases.

The use cases which form the centre of this project are:

- **Coronavirus Detection:** With the sudden outbreak of the COVID-19 pandemic, doctors and researchers have been looking for fast and efficient methods to assist them in detecting and segmenting coronavirus from microscopic visual outputs. To serve this cause, this project looks into designing a custom trained YOLO model and its use in detecting Coronavirus from an electron microscope image and video output.

- **Face Mask Detection:** Coronavirus is generally spread through respiratory droplets when an infected person coughs, sneezes or talks. Wearing a surgical face mask can lower the presence of Coronavirus ribonucleic acid (RNA) in respiratory droplets and therefore help control the spread of COVID-19 to
some extent [5]. Almost all shops and outlets which provide some sort of customer service requires a customer to wear a face mask before entering the location [6]. This requirement only adds to the need for automated face mask detection systems. This project explores the making of a custom trained YOLO model which can detect whether a person is wearing a face mask or not, and whether a person with a mask on is wearing it properly or improperly.

In this project, we mainly use YOLOv4 (the latest version of YOLO released recently in April 2020) [7].

1.5 Project Contribution

The significance of this project lies in the fact that it is an attempt to contribute to the fight against COVID-19 in terms of speeding up detection and controlling its spread.

Through the application of a powerful state-of-the-art object detection model such as YOLO, doctors and researchers all over the globe will have access to rapid Coronavirus detection capabilities from an electron microscope image and video output. This may increase their work efficiency and help them focus on more challenging areas such as identifying new mutations of the virus and developing a cure for the disease.

This project also attempts to increase the efficiency of establishments by reducing human involvement at entrances for checking whether inbound people are wearing a mask (properly/improperly) or not.
1.6 Outline of the Report

The report is organised into four chapters. The first chapter discusses the COVID-19 crisis in terms of its effect on public health and global economy. It provides an overview of the more conventional object detection models and then moves on to Google’s state-of-the-art object detection model – YOLO, where it outlines the need for leveraging its benefits in Coronavirus research. This chapter also summarizes the objectives of this research and the significance of this project.

The second chapter includes a breakdown of the methodology in terms of the technology and platforms leveraged as well as the implementation details. It comprises of an overview of the latest YOLO architecture, and a discussion on the phases followed (dataset creation, data augmentation and pre-processing, target labelling, architecture setup, and model building and training) in order to obtain finetuned and trained custom models for the selected use cases.

The third chapter provides information on the how the trained custom models are tested and evaluated and how they are used to build detectors capable of handling image, video and real-time video camera inputs.

The fourth chapter is a short chapter which summarizes the considerations being made in order to plan future experiments and to work around challenges encountered.

Lastly, the fifth chapter gives a refresher on the major takeaways from this Final report and concludes the report.
2 METHODOLOGY

2.1 Introduction

This chapter starts off by providing an overview of the latest YOLO architecture in Section 2.2 to gain a better understand of the intricacies of this contemporary object detection model. It then gives an outline of the technology and platforms used for executing this project and the various implementation steps undertaken in order to obtain a finetuned and trained model for each use case. Section 2.3 and 2.4 contain information on dataset creation and data augmentation / pre-processing, whereas Section 2.5 includes details of target labelling within the dataset images. This chapter ends with Sections 2.6 and 2.7 which will detail the architecture setup procedure of the custom models and how these models have been built and trained.

2.2 Overview of YOLO Architecture

This project will use YOLOv4 which is built using the Darknet open-source framework for neural networks (written in C and CUDA) [8].

YOLO uses a single Convolutional Neural Network (CNN) to perform detection. Therefore, let us first look into the basic architecture of a CNN. In simple neural networks, each input neuron is connected to the next hidden layer. In CNN, only a small region of input layer neurons (called Local Receptive Fields) connect to the next hidden layers.

![Fig 2.1: Basic CNN architecture](image)

Fig 2.1: Basic CNN architecture [9]
Consider an input image, in the form of a pixel matrix, is passed into a CNN as shown in Fig. 2.1. The Local Receptive Fields are scanned by filters across the entire input pixel matrix to create a feature map. After a feature map is computed, it is passed through a convolutional layer where an activation function (such as Rectified Linear Unit, also termed as ReLu) is applied in order to limit or put constraints on the output values. This is followed by pooling which reduces the dimensionality by downsizing the parameters of the matrix fed from the activation layer to make it suitable for processing. There are multiple such convolutional and pooling layers in a CNN which follow the same process. The final downsized matrix is then fed to fully connected layers where the actual classification takes place [9].

Expanding on the idea of CNN, let us consider the standard object detector architecture shown below in Fig. 2.2.

![Fig 2.2: Basic Object Detector architecture](image)

A one-stage object detector like YOLO (YOLOv4 in this case) consists of 4 major components (Fig. 2.2). The first component is the Input where we input the image into the CNN. The backbone component is mainly for feature extraction and refers to the network that converts the input image into a feature map. YOLOv4 has CSPDarknet53 as its backbone. Cross-Stage-Partial-connections (CSPs) partition the feature map generated from the input matrix into 2 sections and thus reduce overall complexity of the model. Therefore, YOLOv4 leverages these benefits of CSP and combines it with the Darknet53 neural network framework to form the backbone.
component. The Spatial Pyramid Pooling (SPP) block on the CSPDarknet53 is used because it is able to separate out the most relevant context features with almost no effect on network functioning speeds. The neck and head can be considered subsets of the backbone. The neck component is for improving discriminability and robustness of information. YOLOv4 uses Path aggregation network (PAN) as its neck. PAN utilizes the element-wise max operation and combines the information rich feature maps incoming from all layers. It then feeds the concatenated information to the head component. This allows for detection of objects at various scales. The head component is the network used for object detection and YOLOv4 uses the YOLOv3 model as its head. YOLOv3 uses a variant of the Darknet19 framework which makes use of 53 base convolutional layers. For object detection, YOLOv3 requires another 53 layers which are stacked onto the base resulting in a total of 106 layers [10]. This is a general idea of the latest YOLO (YOLOv4) architecture.

2.3 Dataset Creation

This is the most important part of the project as it forms the base behind training the Darknet YOLO model. The quality and quantity of quality data collected will directly influence the performance of the custom model which will be built specifically for the use case linked to this data. Since YOLO is a supervised learning-based object detection model, it trains on labelled data, which is data for which the output is already known. This training process is crucial in defining the model’s performance on new or unseen data.

Due to COVID-19 being such a novel disease, data for research purposes is scarce and for the most part even restricted. All things taken into consideration, we have tried our best to put together relatively balanced datasets for the selected COVID-19 use cases (microscopic coronavirus detection and face mask detection).

2.3.1 Coronavirus Detection Dataset

Due to the scarcity of data discussed above, it was not feasible to build a dataset of acceptable size with just microscopic images of viruses. The workaround we used was
that we combined microscopic images of Coronavirus with microscopic images of other similar looking cells.

The Coronavirus Detection Dataset we developed consists of 750 images in total (250 images per class) belonging to the following 3 classes:

- **Cell Type 2:** This class consists of 250 nuclear microscopic images of certain cells stained with Hoechst 33342. These images were derived from the dataset proposed in the research paper titled ‘Nuclei Segmentation in Microscope Cell Images: A Hand-Segmented Dataset and Comparison of Algorithms’ [11].

![Fig 2.3: Instances of Cell Type 2](image)

- **Cell Type 3:** This class consists of 250 microscopic images of the protist genus Volvox. Protists are microscopic organisms which are generally unicellular and have a well-defined nuclear structure. Volvox is a protist which can be widely described as part of the green algae family. These images were derived from the ‘SinfNet’ GitHub repository which contains datasets and neural networks for micro-organism classification [12].
- **Coronavirus**: This class consists of 250 microscopic images of Coronavirus obtained from various sources.

  A section of these electron microscopic images were sourced by manual web scraping from reliable resources like research databases and related specialized pages.

  Another section of images belonging to this class were taken from a dataset titled ‘SARS-CoV-2 Microscopic Image Dataset with Ground Truth Images and Visual Features’ [13]. This dataset was formulated based on a research study conducted by the North-eastern University in Shenyang, China and the University of Electronic Science and Technology of China in Chengdu, which involved the use of 48 electron microscopic images of Coronavirus and creation of their ground truth equivalents. Upon correspondence with the team’s data manager Jiawei Zhang, the licence to use this data towards creating a comprehensive Coronavirus dataset was successfully obtained. To achieve our purpose, we only used the microscopic images to form part of our dataset.
The last section of these microscopic images of Coronavirus were sourced from a dataset titled ‘Electron microscopy of SARS-CoV particles – Dataset 01’ which consisted of 126 electron microscopy images of extracellular Coronavirus instances [14]. Only a small number out of these could be converted from their original 16-bit TIF format to YOLO readable 8-bit format, and these translated images were used as part of the dataset.

![Image of Coronavirus](image.png)

**Fig 2.5: Instances of Coronavirus**

Although the dataset size of the Coronavirus Detection Dataset is rather small for training a Deep Learning model, it is adequate enough to achieve the aim of this project which is to serve as a proof of concept with room for extension.

### 2.3.2 Facemask Detection Dataset

The Facemask Detection Dataset we developed consists of 1400 images in total, of 4 major types and belonging to 3 classes.

The division of the images based on type is as follow:
- **OK Mask**: This image type is directly linked with one of the classes in this dataset – ‘OK Mask’. These include images where people are wearing their mask properly. There are 514 images of this type with varied face structures and mask designs to enhance the training process.

![Fig 2.6: OK Mask image](image)

- **NO Mask**: This image type is also directly linked with one of the classes in this dataset – ‘NO Mask’. These include images where people are not wearing masks. There are 344 images of this type.

![Fig 2.7: NO Mask image](image)
- **NOT OK Mask:** This image type is also directly linked with one of the classes in this dataset – ‘NOT OK Mask’. These include images where people are wearing their masks improperly, i.e., the mask is not covering the nose, mouth or chin to an adequate level of safety. There are 285 images of this type.

![Fig 2.8: NOT OK Mask image](image)

- **Multi-class:** This image type is linked with all 3 of the classes in this dataset. These include images which have instances of multiple classes within them and not just instances of one class. There are 257 images of this type.

![Fig 2.9: Multi class image](image)
These images have been sourced mainly from 3 main datasets -

1. Face Mask Detection Dataset [15]
2. Mask Detection at YOLO format [16]
3. MaskedFace-Net [17]

### 2.4 Data Augmentation and Pre-processing

We leveraged the following data augmentation techniques:

- **Random Crop:** In order to increase instances of the classes ‘Cell Type 2’ and ‘Cell Type 3’ in the Coronavirus Detection Dataset, we augmented the existing images by adding a random crop with zoom range between 45-65% [18].

  We created 3 and 2 augmented versions of each image for the ‘Cell Type 2’ and ‘Cell Type 3’ classes respectively.

- **Flip:** While augmenting the images of classes ‘Cell Type 2’ and ‘Cell Type 3’ in the Coronavirus Detection Dataset using Random Crop, we added in random vertical and horizontal flips to add in more variability between augmented versions.

  Flips also make the model more sensitive to orientation of the target objects [18].

- **Brightness:** While augmenting the images of class ‘Cell Type 3’ in the Coronavirus Detection Dataset using Random Crop, we add in variability to the image brightness by randomly making the image brighter/darker by 25%.

  This data augmentation technique helps the model adapt better to change in lighting [18].
- **Noise:** While augmenting the images of class ‘Cell Type 3’ in the Coronavirus Detection Dataset using Random Crop, we also add in some noise between 0-2% which improves the generalization capability of the model [18].

We leveraged the following data pre-processing techniques:

- **Auto Orient:** This pre-processing technique is for standardizing the ordering of pixels within images [18]. We have used this pre-processing technique for all images in both the datasets we have created.

- **Greyscale:** In the Coronavirus Detection Dataset we created, we are only dealing with microscopic visual inputs. To improve model performance in such a case, we can combine the colour channels in order to speed up the model and make it better at detecting objects without taking into account the colour aspect [18].

### 2.5 Data Labelling

The images in the datasets must be annotated or labelled for the models to be able to train on them and perform object detection. For both the created datasets, we have labelled the target objects within the images by using an open-source graphical image annotation tool called LabelImg which supports the YOLO annotation format [19]. Let us consider the image shown below in **Fig 2.10** and its labelling output.

![Fig 2.10: Image (left) and it's labelling output (right)](image)
As seen in Fig 2.10, on labelling an image using the LabelImg tool, the labelling output is stored in a text file. Each line in the text file gives information about a separate label and thus, the number of lines in the text file correspond to the number of objects labelled in the associated image. Considering the image in Fig 2.10, there are 2 instances of class ‘NO Mask’ and 1 instance of class ‘OK Mask’ observed. Therefore, there are 3 lines in the linked text file, each containing information about a separate label.

Each line in the text file containing the labelling output is in the format:

```
Label Centre_X Centre_Y Width Height
```

where the last 4 parameters are normalised to have values between 0 and 1.

The ‘Label’ parameter in the above format refers to the numeric value of the labelled class with respect to the total classes we have in our model. Our Facemask Detection Dataset has 3 classes ‘NOT OK Mask’, ‘OK Mask’ and ‘NO Mask” corresponding to numeric values 0, 1 and 2 respectively. Thus, the labels in Fig 2.10 indicate that 2 people are not wearing a mask and 1 person is properly wearing a mask.

The other 4 parameters are related to the position and size of the labelled boxes, expressed in terms of the image dimensions.

### 2.6 Architecture Setup

We started off the architecture setup by downloading the Darknet source required for YOLOv4 from ALexeyAB’s GitHub repository [20]. Although we maintained separate Darknet directories for each use case (coronavirus detection and facemask detection), the majority of the changes involving architecture setup were the same for both.

Since we decided to leverage the GPU based runtime provided in Google Colaboratory notebook, we made the following changes to the Makefile:

```
GPU = 1
CUDNN = 1
OPENCV = 1
```
We then downloaded the pretrained darknet weights file titled ‘yolov4.conv.137’ from the Google drive linked to the above GitHub repository to serve as the base for our training. We did this to perform model training on top of these weights in order to facilitate transfer learning so that previous knowledge / method of learning can be included. We also downloaded the pretrained YOLOv4 weights file titled ‘yolov4.weights’ for testing darknet compilation.

Finally, the last step in our architecture setup for the custom model involved editing the configurations file for YOLOv4. The changes we made are as follows:

- In line 3 and 4, we set the batch size as well as the subdivisions each batch is broken into to 64. We chose a large number for the subdivisions since we are leveraging the GPU based runtime provided by Google and not an actual GPU farm setup. Thus, we did this to avoid exhausting GPU memory.

  \[
  \text{batches}=64 \\
  \text{subdivisions}=64
  \]

- In line 8 and 9, we set the pixels that are fed into the first layer of the YOLOv4 CNN to 416 in terms of width and height. 416x416 is considered to be the pixel layout of a medium-sized image.

  \[
  \text{width}=416 \\
  \text{height}=416
  \]

- In line 20, we set the maximum batches to 6000 based on the formula \text{max\_batches} = \text{number of classes} \times 2000 stated in AlexeyAB’s GitHub documentation (since we will be using 3-class datasets for training models for both use cases) [20]. This specifies the maximum number of iterations our model should be trained for.

  \[
  \text{max\_batches}=6000
  \]

- In line 22, we set the value of steps to 90% of maximum batches

  \[
  \text{steps}=5400
  \]
- In lines 962, 1050 and 1138, we set the filters to 24 based on the formula in AlexeyAB’s GitHub documentation: \( \text{filters} = (\text{classes} + 5) \times 3 \) [20]. Since we will be using 3-class datasets for training models for both use cases, we get value of filters to be 24.

\[ \text{filters} = 24 \]

- In lines 969, 1057 and 1145, we set the classes to 3 since the datasets we created for both use cases consists of 3 classes.

\[ \text{classes} = 3 \]

### 2.6.1 Helpers

In order to prepare the YOLO architecture of the custom model for training, we may need to perform certain preparatory tasks. For this purpose, a number of Python based helper scripts have been written.

These helpers serve the following purposes:

- **Data Segregation**: The first helper is written specifically for datasets like the Facemask Detection Dataset which includes images containing instances of just one class as well as images with instances belonging to multiple classes. In order to roughly maintain the same ratio of images per class within the train and test sets and to provide for even stratification, we might need to perform a class wise segregation on the original dataset before implementing a train-test split. This helper is designed to create a directory titled ‘segregated_dataset’ (out of the original dataset) which contains separate directories comprising of images belonging to different classes. This helper is written such that the multi-class type of images are also segregated into their own directory.

- **Train-Test Split**: The second helper is written to assist in splitting the custom dataset into training and test sets based on a 90/10 split while roughly maintaining the same ratio of images per class within these 2 sets. After splitting the dataset, this helper is programmed to create 2 text files linked to the training
and test sets, and subsequently write paths of associated images to the respective text files in order to make it accessible to the YOLO Darknet architecture.

- **Data and names file creation:** The third helper is written to assist in creating the names and data files which the YOLO Darknet architecture requires for training. The names file contains the names of all the classes or categories the custom model will be dealing with. The data file contains 5 pieces of information linked to the associated custom dataset – number of unique classes, location of the text file linked to the training set, location of the text file linked to the test set, location of names file and the location of the backup folder.

- **Image format conversion:** The fourth helper deals with the problem of image incompatibility. It is programmed to convert all the images in the custom dataset to ‘jpg’ format. At the moment, it can only handle 3 formats of input images – ‘jpg’, ‘jpeg’ and ‘png’; but this script can easily be extended to deal with other types of images.

- **Labels folder creation:** The fifth helper is rather simple and is written in order to create a directory with all the image labels linked to the associated custom dataset.

### 2.7 Training the Custom Models

After performing the Darknet architecture setup and running the helper scripts to prepare files required for training the custom models, we zipped the darknet source directories for each use case along with the associated datasets and uploaded it to respective directories on Google drive.

Since the computational resources required for training the custom YOLO models is rather high, we made use of Google Colaboratory notebook and the free powerful Graphics Processing Unit (GPU) provided within it by Google. We created a notebook titled ‘coronavirus-detection-model.ipynb’ for the Coronavirus Detection use case and a notebook titled ‘facemask-detection-model.ipynb’ for the Face Mask
Detection use case in the respective directories and wrote scripts for leveraging the allocated GPU based runtime for training the custom models.

Below is a short overview of the basic functionality of the script blocks written.

1. **Mounting Google drive:**
   This script block was written to mount the Google drive containing the darknet zip file which has the code and data we required for training.

2. **Exploring the Ubuntu machine allocated:**
   We then proceeded to write this block for retrieving the runtime details of the GPU based Ubuntu machine allocated by Google Colab and for updating its repository list.

3. **Retrieving the darknet folder:**
   In order to achieve this end, we wrote scripts to perform the following 5 actions:
   a. → Unzip the darknet zip file code and data we required
   b. → cd into the darknet directory created in the Ubuntu system
   c. → Install dos2unix
   d. → Use dos2unix in order to convert Max/DOS files to Unix files
   e. → Make the darknet folder in the Ubuntu system executable

4. **Compiling the darknet framework:**
   This block makes use of the Makefile to compile the darknet framework. After compilation, we could optionally use the pretrained YOLOv4 weights in the file titled ‘yolov4.weights’ to test for successful compilation.

5. **Redirecting the backup folder to drive backup:**
   In this block, we deleted the default backup directory specified within the unzipped darknet folder and created a link from the darknet directory to a google drive backup location (the ‘backup’ directory within the ‘coronavirus_detection_model_weights’ directory for the Coronavirus Detection use case or within the ‘facemask_detection_model_weights’ directory for the Face Mask Detection). During training, our model will save the milestone weights (every 1000 iterations along with the last and the best weights
recorded) periodically in this linked drive location for future procedures like model evaluation.

6. **Training the model:**
   a. This block starts training the custom model for the particular use case by specifying the paths to the associated data file, the custom YOLOv4 configurations file and the pretrained convolutional weights file for training as arguments.
   b. We specify -map as an argument in order to generate a chart to get an idea about the loss and mean average precision (mAP) score with respect to increasing iterations.
   c. We also specify -dont_show in order to prevent inline display of map charts which might hamper with the training process.

7. **Resuming Training (Optional):**
   a. This block deals with the scenario where the Google Colab runtime for training the model stops or fails for some reason. In such a situation, we can resume training from the weights stored frequently in the associated backup folder linked to the mounted Google drive.
   b. The recommended option when resuming training is to use the best weights which are most recently recorded during training, with the other option being the use of the last weights which were backed up.

Upon successfully running the script blocks 1 to 6 described above in both notebooks using the GPU based runtime environment provided by Google Colab, we were able to train each of the custom models in just under 16 hours.

2.8 **Summary**

This chapter provided a general illustration of the architectures of CNN and the latest YOLO (v4) object detection model with an insight into the function of each component. It also summarized the methodology linked with the different development stages (up to training of the custom models linked to the created datasets)
and outlined the means to leverage the technology and platform requirements for this project.
3 RESULTS & APPLICATIONS

3.1 Introduction

This chapter starts off in Section 3.2 by detailing the evaluation metrics we used to assess the created custom models, the inferences we drew from the training charts and the evaluation results we obtained for each model. It ends with Section 3.3 which contains information on how detectors capable of detecting custom objects linked to each created dataset from image, video and real-time data input were built and applied.

3.2 Testing and Evaluation

In order to perform testing and evaluation, we make use of the data from the test set. Before discussing the training inferences and the test set performance of our models, we start by defining the evaluation metrics used.

3.2.1 Evaluation Metrics

Recall is the ability of a model to find relevant targets in a dataset and it is calculated by dividing the number of true positives by the sum of the number of true positives and false negatives. Precision is the proportion of targets our model identifies as relevant that are actually relevant and it is calculated by dividing the number of true positives by the sum of the number of true positives and false positives. F1-score is defined as the harmonic mean of the model’s precision and recall and is calculated by the formula: $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$.

Intersection over Union (IOU) is a gauge of the overlap between boundaries from our prediction and the ground truth [21]. Finally, the last metric we used to evaluate the custom models is Mean Average Precision (mAP) score which is the mean of the Average Precision (area below the precision-recall curve) across all IOU limits [21].
3.2.2 Inferences from training the custom models

After successfully training the custom models built for each use case, we draw inferences and analyse the training chart we received as output.

→ Coronavirus Detection model

We obtained the training chart shown below in Fig 3.1 as a result of training the Coronavirus Detection model. We observe from the chart that the training process ran for 6000 iterations and the mAP score (calculated on the test set) was calculated starting from the 1000th training, as was set in our configurations file.

![Training chart linked to the Coronavirus Detection model](image)

**Fig 3.1:** Training chart linked to the Coronavirus Detection model
We see that the training loss continually decreases with increasing iterations, which suggests improvement in our custom Coronavirus Detection model. The training process yielded the best result at roughly the 3600th iteration where the mAP score calculated using the images in the test set came out to be 98.07%. This is also the point which corresponds to the best weights saved in the Google drive backup location associated with this use case.

→ Facemask Detection model

We obtained the training chart shown below in Fig 3.2 as a result of training the Facemask Detection model.

Fig 3.2: Training chart linked to the Facemask Detection model
We observe from the chart that the training process ran for 6000 iterations as set in our configurations file and the mAP score (calculated on the test set) was calculated starting from the 1000th training, as was set in our configurations file.

In this case as well, we see that the training loss continually decreases with increasing iterations, which suggests improvement in our custom Coronavirus Detection model. The training process yielded the best result at roughly the 3000th iteration where the mAP score calculated using the images in the test set came out to be 91.2%. This is also the point which corresponds to the best weights saved in the Google drive backup location associated with this use case.

### 3.2.3 Evaluating the models

In the Google Colab notebooks we created for training our custom models, we added in another script block for evaluating the models after training. The darknet script contains a map flag which allows us to obtain many useful evaluation metrics like mAP, precision, recall, F1 score, etc. for the trained YOLOv4 model and average precision with regards to each class in the linked dataset. In the block command, we make use of this flag and specify the paths to the associated data file, custom YOLOv4 configurations file and best weights file recorded during training as arguments.

The following are the results of running the evaluation script block for each custom model.

**Coronavirus Detection model**

We get the following average precision (AP) values for the 3 classes in the Coronavirus Detection Dataset (shown below in Table 3.1):

<table>
<thead>
<tr>
<th>CLASS</th>
<th>AP VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Type 2</td>
<td>96.99 %</td>
</tr>
<tr>
<td>Cell Type 3</td>
<td>99.77 %</td>
</tr>
<tr>
<td>Coronavirus</td>
<td>97.44 %</td>
</tr>
</tbody>
</table>
The following table (Table 3.2) contains precision, recall, F1-score, average IOU and mAP values obtained after evaluating the trained Coronavirus Detection model on the test set:

**Table 3.2: Evaluation results for the Coronavirus Detection model**

<table>
<thead>
<tr>
<th>EVALUATION METRIC</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRECISION</td>
<td>0.85</td>
</tr>
<tr>
<td>RECALL</td>
<td>0.98</td>
</tr>
<tr>
<td>F1-SCORE</td>
<td>0.91</td>
</tr>
<tr>
<td>AVERAGE IOU</td>
<td>74.43 %</td>
</tr>
<tr>
<td>mAP SCORE</td>
<td>98.07 %</td>
</tr>
</tbody>
</table>

**Facemask Detection model**

We get the following average precision (AP) values for the 3 classes in the Facemask Detection Dataset (shown below in Table 3.3):

**Table 3.3: AP values for classes in the Facemask Detection Dataset**

<table>
<thead>
<tr>
<th>CLASS</th>
<th>AP VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT OK Mask</td>
<td>87.70 %</td>
</tr>
<tr>
<td>OK Mask</td>
<td>94.93 %</td>
</tr>
<tr>
<td>NO Mask</td>
<td>90.53 %</td>
</tr>
</tbody>
</table>

The following table (Table 3.4) contains precision, recall, F1-score, average IOU and mAP values obtained after evaluating the trained Facemask Detection model on the test set:
Table 3.4: Evaluation results for the Facemask Detection model

<table>
<thead>
<tr>
<th>EVALUATION METRIC</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRECISION</td>
<td>0.87</td>
</tr>
<tr>
<td>RECALL</td>
<td>0.91</td>
</tr>
<tr>
<td>F1-SCORE</td>
<td>0.89</td>
</tr>
<tr>
<td>AVERAGE IOU</td>
<td>70.18 %</td>
</tr>
<tr>
<td>mAP SCORE</td>
<td>91.05 %</td>
</tr>
</tbody>
</table>

3.3 Building detectors

After training, testing and evaluating the custom models for the selected use cases, we downloaded the best weights recorded for each model from the associated Google drive backup location. We then wrote Python based scripts using OpenCV to build detectors (utilizing these model weights) capable of detecting custom objects linked to each created dataset from image, video and real-time (only for the Facemask Detection Dataset) data input.

In the case of the detector created to handle image inputs, the image path passed as an argument is used to first load the linked image. The loaded image is then processed and converted into a binary object called blob which is easily readable by our custom model. Next, we load our trained custom YOLOv4 model for the use case we are applying and obtain the model predictions by forward propagating the image blob until the output layer. We then loop over the predictions returned by our custom model and convert the bounding box values from the blob dimensions to the original image dimensions by upscaling and converting to co-ordinate format. In order to avoid the issue of overlapping bounding boxes for same object instances in an image, we implement a function known as non-maximum suppression. This function returns only the bounding box with the highest confidence score for each object instance while
suppressing the non-maximum overlapping bounding boxes. Lastly, we iterate through the model predictions returned after non-maximum suppression in order to extract the bounding box co-ordinates \((x_{\text{start}}, y_{\text{start}}, x_{\text{end}}, y_{\text{end}})\), class label and class confidence score. We use this information for drawing the bounding boxes along with the text labelling within the output image shown to the user in a dedicated window.

\textbf{Coronavirus Detection model}

In Fig 3.3, we can see the output produced by the detector created to detect objects in the Coronavirus Detection Dataset from image inputs.

\textbf{Fig 3.3: Detector predictions for Coronavirus Detection Dataset images}
→ Facemask Detection model

In Fig 3.4, we can see the output produced by the detector created to detect objects in the Facemask Detection Dataset from image inputs.

![Image of detector predictions for Facemask Detection Dataset images](image1)

![Image of detector predictions for Facemask Detection Dataset images](image2)

**Fig 3.4:** Detector predictions for Facemask Detection Dataset images
The detectors created to handle video and real-time video inputs have a very similar logic to the detectors created to handle image inputs. The only difference is that since both video and real-time video are basically a stream of image frames, we capture the stream into a stream object and extract each image frame to get predictions, by forward propagating them through the trained custom model for the use case being applied. In the case of the detectors created to handle video inputs, we keep extracting image frames as long as the video is playing or until there is a pre-defined key press, whereas, in the case of the detectors created to handle real-time video inputs, we keep extracting image frames until there is a pre-defined key press.

3.4 Summary

This chapter provided a detailed explanation of the inferences and considerations made in order to test and evaluate the custom models built for the 2 selected use cases. It also gave insight into the results of this project and summarized the process by which we used the custom models we trained in order to build detectors.
4 CHALLENGES & FUTURE PLANS

As discussed in Section 2.3, data and information linked with COVID-19 case studies is scarce and for the most part even restricted. Therefore, this poses a challenge in terms of building a custom model with good statistical performance. The obstacle in this is enhancing the real-world performance of models which train on small datasets like the Coronavirus Detection Dataset (which currently consists of 750 images belonging to 3 classes) and are linked to a use case with a very wide domain (large number of virus/micro-organism classes in reality). To tackle this, we can explore expanding our dataset by adding in more classes of viruses and micro-organisms through collaborations with research institutes and medical imaging centres. In terms of the Coronavirus Detection use case, currently detectors have been built to handle image and video inputs. One future step in this regard could be to feed microscopic video output as input straight into the custom model to perform detections. This would be best way to evaluate real-world performance of the Coronavirus detection model.

The Facemask Detection model does a good job in detecting and differentiating between the 3 classes of the associated dataset. But in some rare scenarios among the cases in which subjects use their hands or other foreign objects to cover their nose and mouth, our custom model outputs a detection of the class – ‘OK Mask’ even though a mask is not worn. Although this case is infrequent, it does affect the overall performance of the model in terms of real-world application. A future step which would serve as a workaround to this could be to add more diverse training images of the case described above and perhaps even create a dataset with more specific classes which can definitively pinpoint the mask status at one glance.
5 CONCLUSION

The widespread outbreak of COVID-19 has had an adverse effect on the global economy and public health. The world is still lagging behind in terms of Coronavirus research and hasn’t been successful in finding a cure for the disease. Therefore, our project is an attempt to apply the state-of-the-art object detection model ‘YOLO’ to COVID-19 use cases in order to allow for faster and more efficient virus detection capabilities and improved spread control. The significance of this project is that it may increase the work productivity of researchers who may be able to focus on more testing issues. The project also attempts to replace manpower at establishment entrances to serve the purpose of social distancing and disease control.

We created 2 datasets for the selected use cases – Coronavirus Detection and Facemask Detection. The Coronavirus Detection Dataset we created is made up of 750 images belonging to 3 main classes and was put together by leveraging data augmentation techniques discussed in Section 2.4. The Facemask Detection Dataset we created consists of 1400 images of 4 types belonging to 3 main classes. After successfully labelling the object instances within the images constituting these datasets and performing initial YOLO architecture setup with the help of some Python based helper scripts, we were successful in building and training the custom models for each use case. In terms of testing and evaluation results, the trained Coronavirus Detection model yielded a mAP score high of 98.07%, whereas, the trained Facemask Detection model yielded a mAP score high of 91.05%. The saved best weights for each of these models were then used to build detectors capable of handling image, video and real-time video inputs. The main challenge we faced was the scarcity of imaging data available due to COVID-19 being such a novel disease. A future experiment could be to feed microscopic video output as input straight into the Coronavirus Detection custom model in order to get a better gauge of its’ real-world performance. With regards to the Facemask Detection model, a future plan could be to train it on a larger dataset having higher variability between the images and a larger number of specific classes which can definitively pinpoint the mask status at one glance.
REFERENCES


