



University | School of
of Glasgow | Computing Science

Social Distancing Detector with Deep Learning and Parallel Computing on Android Application

Jinkawin Phongpawarit

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the
Degree of Master of Science at The University of Glasgow

xx September 2020

Abstract

Due to pandemic on this year, social distancing has been recommended by the government, and this can reduce the spread of disease. With the help of technology, social distancing violation can be determined by using computational devices, such as mobile phone. To determine social distancing violation, firstly, humans are detected by using deep neural network. Then, the distance will be calculated among detected humans. This dissertation is developing social distancing detection on Android application with the purpose of performance maximisation by using all tools and techniques. Parallel computing and NEON instruction are used for performance enhancement. To detect humans by using deep neural network, OpenCV, a third-party library, is used in this project. OpenCV is used for network initialisation, and forwarding preprocessed input through the network. To understand performance maximisation, there are comparisons among detection models, technologies, and programming languages. Comparisons and evaluation of the performance are presented in this dissertation. The results show that parallel computing and NEON instruction improve the application's performance. In addition, the performance is better when OpenCV is manually built. Moreover, the performance is improved with speed-up up to 52 per cent.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Jinkawin Phongpawarit Signature:

Acknowledgements

I would like to express my very great appreciation to my supervisor, Dr Jose Cano Reyes, for his mentorship throughout a whole duration of this project. This dissertation would not be completed without the guidance of my supervisor. He always provided me with very invaluable advice, and he encouraged me when I struggled in troubles; these have been very much appreciated.

I would like to thank my family for their love and the opportunity of studying a master degree that I have given for a whole year. It is the most valuable experience in my life. Last but not least, I would like to thank my friends who have supported and encouraged me during the past year.

Contents

1	Introduction	5
1.1	Motivation	5
1.2	Aim and Objectives	6
1.3	Structure	6
2	Background	7
2.1	Threading in Android	7
2.2	Distance Calculation	8
2.3	Specification	9
2.4	Existing Applications	9
2.4.1	Object Detector	9
2.4.2	TensorFlow Object Detection	9
2.4.3	Computer Vision Detection	9
3	Design and Implementation	10
3.1	MoSCoW Statement	10
3.1.1	Must have	10
3.1.2	Should have	10
3.1.3	Could have	10
3.1.4	Won't have	11
3.2	Interface Design	11
3.3	Preparation	11

3.4	Java Native Interface	11
3.5	Social Distancing Detection	13
3.6	Parallelisation	14
3.7	Frame Rendering	15
4	Evaluation and Testing	17
4.1	Controlled Variables	17
4.2	Performance Evaluation	17
4.2.1	Single Frame Comparison	18
4.2.2	Multithreading	18
4.2.3	Android Native Development Kit	20
4.2.4	NEON Instruction	21
5	Conclusion	23
5.1	Limitation	23
5.2	Future Work	23
A	Existing Application	25
B	Application Screenshot	27

Chapter 1

Introduction

1.1 Motivation

Humanity has been faced several pandemics over hundreds of years, and many lives are threatened. For example, there was the black death in 1346, and the flu pandemic in 1918, and millions of people died during each pandemic [2], [8]. This year, humanity is confronting the other pandemic, which is named as coronavirus or COVID-19.

However, a threat from illness or the impact of the outbreak can be reduced by using technology and scientific discovery. For instance, social distancing has been recommended since there were Influenza A (H1N1) outbreaks in 2009 [6]. Social distancing is able to reduce the spread, and slow down and reduce the size of the epidemic peak [4], [5]. Consequently, an infection curve is flattened, and the number of deaths is reduced. Currently, scientific researchers are still working on this pandemic with the aim of reducing the infection rate. In addition, technology can be integrated with scientific theory to gain more advantages.

Due to competitive advantage and business competition, the ability of computable devices have been improved, which is capable of executing very complex tasks. Mobile phone, which is one of a highly competitive market, become a powerful device. People use a mobile phone for many purposes, such as entertainment, education or business. Likewise, in application development, there are advantages to the mobile device. The first advantage is performance. As the aforementioned competition, the hardware of the mobile phone has been improved over the years, including CPU, GPU, and memory. Thus, the capability of the mobile phone is sufficient for performing calculation tasks heavily. The second advantage is portability. Most of the mobile phones provide necessary features, such as a camera, sensor, and GPS. In addition, a mobile phone has a battery, which does not require a charger during being used.

For the advantages above, a mobile application can be used as a tool to help human determine social distancing.

1.2 Aim and Objectives

The aim of this project is determining distances between people, and maximising the performance of the application to achieve the highest frame rate (frame per second or FPS). To accomplish the aim, the objectives of this project are specified by following:

- The application will determine objects from the given image and video by using Deep Neural Network (DNN).
- The application will determine distances between people in real-time from a camera.
- Input will be processed simultaneously by using parallel processing techniques.
- An advanced single-instruction multiple-data (SIMD), NEON instructions, will be implemented to improve the calculation performance.

1.3 Structure

The content of this dissertation is structured as follows:

1. Chapter 2 provides background knowledge of Mulithrading in Android application, social distancing determination, hardware specification, and existing applications.
2. Chapter 3 shows an overview of the system and describes implementation details in various aspects.
3. Chapter 4 evaluates and compares the result of the performances of each implemented technology.
4. Chapter 5 concludes the results of the project, limitation of this project, and future works.

Chapter 2

Background

This chapter aims to introduce background knowledge that relates to this project. Then, this chapter gives examples of related applications.

2.1 Threading in Android

Threads in Android development are categorised into two main categories: the main thread and a worker thread. A main thread or UI thread is a thread that dispatch events of user interface widgets. The events are dispatched regarding Android's activity lifecycle, and all events are managed by only one thread. In other words, threads will not be spawned for handling a single event or component. Thus, if there is a long-running task, which is run by UI thread, events cannot be delivered because long-running task block UI thread. If UI thread is blocked more than 5 seconds, an application will show "Application not Responding" [1].

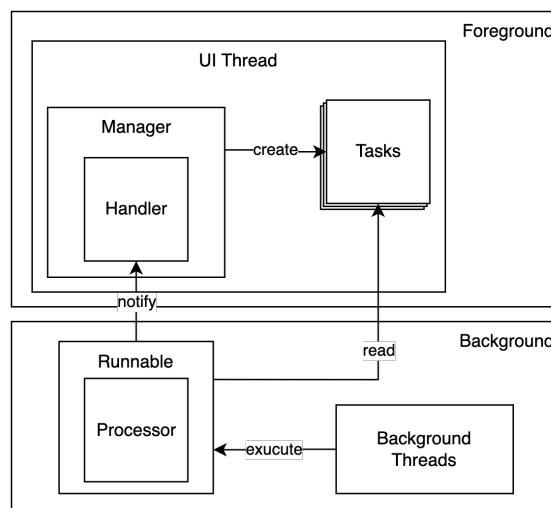


Figure 2.1: Threading Overview

A working thread is separated from the UI thread to avoid freezing application, and it is called

worker thread or background thread. This thread is able to process a long-running task in the background without interrupting the UI thread. In addition, the priority of the thread can be set from -20 to 19—the lowest priority is 19 and the highest priority is -20. The default value of background threads priority is 10, and the default value of foreground threads is -2.

UI thread and background thread are running on different threads; thus, Handler is needed when there is communication between these threads. For example, as shown in figure 2.1, processes are divided into two parts: foreground and background. Tasks are created by a manager which is running on the UI thread. Then, Runnable, a component that can be run by background threads, will read tasked. After that, Runnable will be executed by background threads. Finally, Runnable will notify the manager through Handler.

2.2 Distance Calculation

According to Gurucharan [7], to measure a distance between 2 people, the reference point of people are used for calculation. The reference point is the coordination of each people, which is the centre of the detection frame. The calculation formula is based on Euclidean distance.

$$d = \sqrt{(a_0 - b_0)^2 + (D/c) \times (a_1 - b_1)^2}$$

$$c = \frac{a_1 + b_1}{2}$$

However, three-dimensional space are captured into a two-dimensional image, so depth and perspective are concerned as can be seen in figure 2.2a. Thus, a couple of variables are added into the formula. The first variable is D , which is the diagonal of the image. The second variable is c , which is a calibration. These two variables will determine the depth of people in the image.

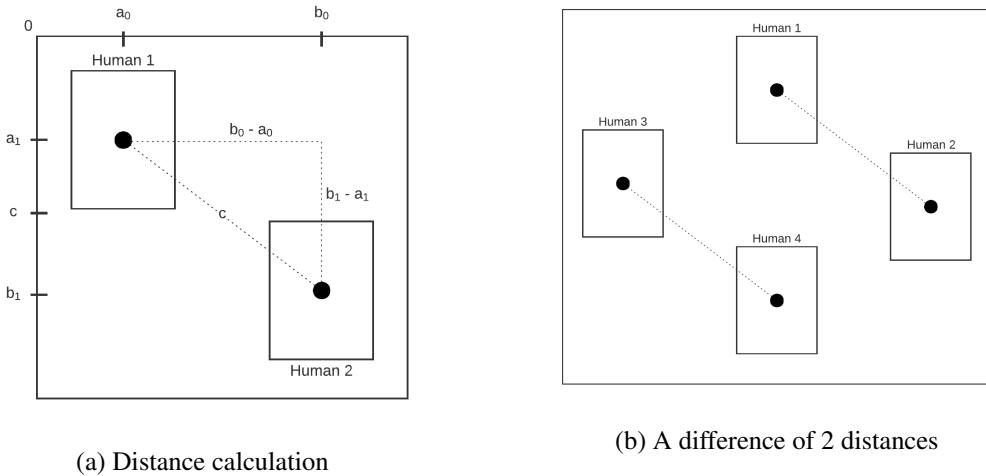


Figure 2.2: Determining Social Distancing

For example, according to figure 2.2b, if the distance is calculated without calibration, the distance between 2 couples will be the same. Naturally, the distance between Human1 and Human2 must be further than the distance between Human3 and Human4.

2.3 Specification

This application is developed and tested on following specification:

Device	Device	Samsung S10+
Operating System	Operating System	Android 10 (Q)
Processor	CPU	Samsung Exynos 9820
	Cores	8
	Architecture	2x ARM Cortex-A75 2.73GHz 4x ARM Cortex-A55 1.95GHz 2x Samsung Exynos M4 1.95 GHz
	GPU	Mali-G76
Memory	RAM	8 GB

Table 2.1: Specification

2.4 Existing Applications

2.4.1 Object Detector

Object Detector is an Android application, which is able to detect objects through a camera. This application is implemented by deep learning library from TensorFlow with MobileNet model. There are two modes in this application, which are detection mode and classification mode. The processing time of each frame is 250 to 300 milliseconds.

2.4.2 TensorFlow Object Detection

TensorFlow Object Detection is an object detection application, which is run on the Android operating system. TensorFlow's library is used in this application, along with the MobileNet model. This application is able to detect objects in real-time from the camera.

2.4.3 Computer Vision Detection

Computer Vision Detection application operates on the Android operating system. This application is able to process real-time video from a live camera, and OpenCV is used as a library. There are 12 options of algorithms, such as colour detector, canny detector, motion detector, and shape detector. Besides, this application is able to detect human faces and smiling faces. This application is able to process around 13 frames per second.

Chapter 3

Design and Implementation

This chapter shows the overview of application design and the application. MoSCoW statement will describe the functional requirement and the implementation details will be explained in many aspects, including Native library, social distancing detection process, parallelisation and processed video rendering.

3.1 MoSCoW Statement

3.1.1 Must have

- The application **must** be able to detect people in the given image or video.
- The application **must** be able to determine distancing between people in the given image or video.
- The application **must** save the processed image of video.
- The application **must** allow user to choose image or video from device's storage.

3.1.2 Should have

- The application **should** be able to stream video from camera.
- The application **should** be able to show detected people on camera.
- The application **should** support parallel computing.

3.1.3 Could have

- The application **could** choose computation options between sequential or parallelism.
- The application **could** support NEON technology.
- The application **could** be able to process the given tasks in background.

3.1.4 Won't have

- The application **won't** other objects which are not human.
- The application **won't** support GPU computation.

3.2 Interface Design

This application consists of three main processing functions: using video, using a picture and live from a camera. When a user opens the application, the user will navigate to the main menu, which can be seen in figure B.1a. The user will be able to choose the processing function by clicking buttons. As can be seen in figure B.1b, video and picture button will navigate the user to the device's directory for choosing a file. The file will be processed in the background when a file is chosen, and the status bar will be updated regarding the progress. When processing is done, as it shows in figure B.2, video or image will be shown at the top of the application. The detected human will be drawn a red rectangle if the detected human is breaching social distancing detection, while The detected human will be drawn a red rectangle if the detected human is retaining social distancing. In addition, the user is able to use the camera to determine social distancing in the real-time. The user will be navigated to the camera activity when user click "camera" button in the application as can be seen in figure B.4.

3.3 Preparation

This application is implemented on the Android Operating System, and the target Software Development Kit (SDK) version is set at level 29, namely Android Q. This application is compiled and built by Android Studio version 3.5.3, and CMake is used for compiling C++ library with C++ version 11. Furthermore, OpenCV version 4.4.0, which is built as a shared library, is used for image processing and object detection. For object detection model, two models are used: You Only Look Once and MobileNet SSD.

3.4 Java Native Interface

Implementation is divided into 3 layers. The first layer is a Java layer, which mainly interacts with a user, checks permissions, handles activity lifecycle, communicates with Java Native Interface (JNI), and loads native libraries. Native libraries are compiled and built into shared libraries by a Native Development Kit (NDK). The second layer is JNI, which is written in C or C++. The task of the JNI layer is an intermediate connection between the Java layer and a C++ layer. The last layer is the C++ layer, which alternatively performs calculation tasks, including Deep Neural Network and distance calculation.

However, Deep Neural Network and distance calculation can be implemented in all layers. According to Android Developer Guide [1], Native Development Kit (NDK) is recommended for compiling C and C++ code into the native library, which is able to achieve higher performance. Thus, executing both tasks in the JNI and C++ layer gains a better performance. Furthermore, there

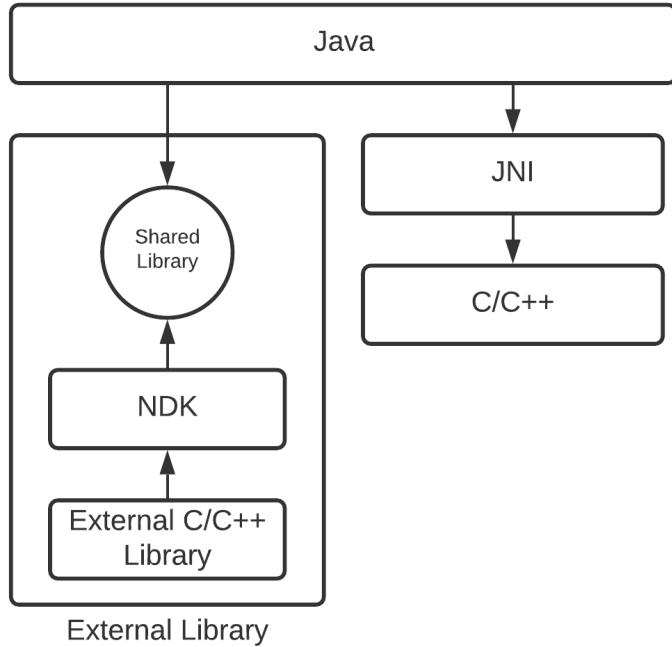


Figure 3.1: Application Layers

are two advantages to implementing JNI. The first advantage is reducing JNI calling. Performing both tasks in an application layer have to call JNI methods many times, and this is expensive and cost overhead. Thus, manually implementing JNI reduces the number of JNI calls. The second advantage is memory management. C++ is able to access values in the memory by using a pointer. Thus, values can be directly accessed without copying.

Java and JNI communicate through native function, which is written in the Java layer. Memory addresses of the pre-processed frame in Mat format will be pass as a parameter through native function, and it will be converted from Java type to Native type. Then, the given addresses will be converted back into Mat format.

```

1  public class NativeLib {
2      static {
3          System.loadLibrary("native-lib");
4      }
5
6      public native static void process(long imageAddr);
7  }

```

Listing 3.1: Java Native Function

```

1  Java_com_jinkawin_dissertation_NativeLib_process(jlong matAddr){
2      Mat &frame = *(Mat *) matAddr;
3      ImageProcessor::process(frame);
4  }

```

Listing 3.2: C++ JNI Method

3.5 Social Distancing Detection

There are three main processes that are implemented to determine social distancing violation from the image and video.

The first process is pre-processing the given image, video, or video stream. The given video will be extracted into an array of images. Then, images will be converted into Bitmap and Mat format with RGBA colour model respectively. After that, the colour will be converted, which depends on the detection model. As mentioned in section 4.1, there are two detection models used for detecting humans in the given picture and video: YOLO model and MobileNet SSD model. Colour will be converted to RGB if the detection model is YOLO, while MobileNet SSD requires BGR colour model.

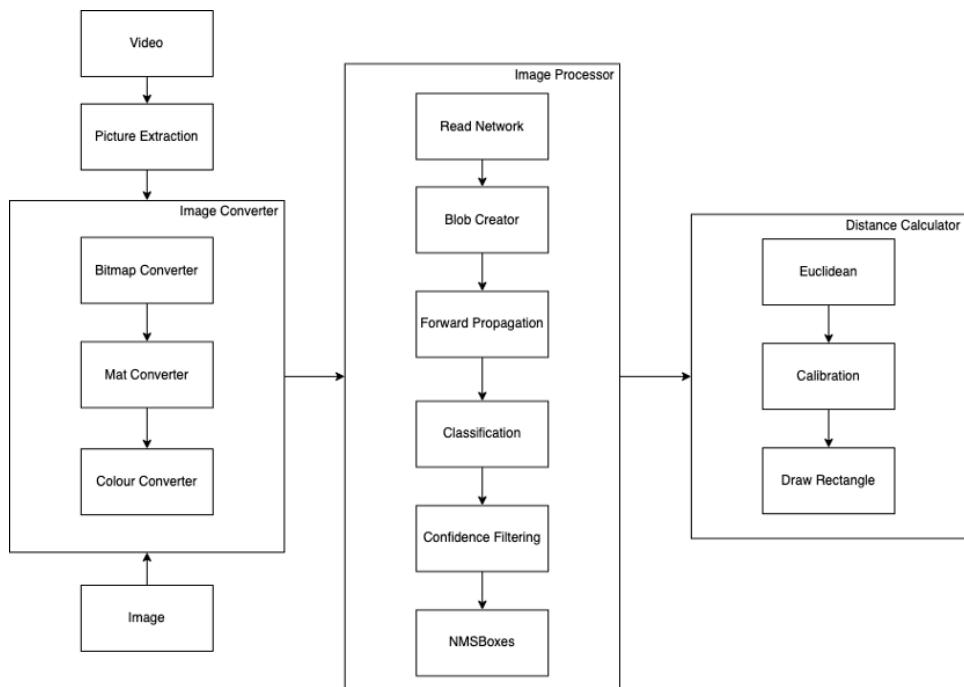


Figure 3.2: Detection System Overview

The second process is object detection. Detection models are set up and configured differently before processing images. First of all, YOLO is used with Darknet, which is an open-source neural network framework, while MobileNet SSD. MobileNet is used with Caffe framework. Then, the threshold will be set for determining the confidence score of the detected object. The confidence score threshold of YOLO model will be set to 0.5 or 50 per cent. In other words, the detected object will be rejected if YOLO model cannot guarantee that a detected object is human, and its confidence score is lower than 50 per cent. In contrast, the confidence score threshold of MobileNet SSD model will be set to 0.3 or 30 per cent. Because of MobileNet SSD model has a lower ability to detect an object, the confidence threshold is set to be lower. After models are set up and configured, the image will be processed in 5 steps. For the first step, the pre-processed image will be converted to input blob bypassing Mat image to `blobFromImage()` function with scale factor. Blob will be used for the forward propagation in the neural network. Secondly, the blob will be passed to the network through `forward()` function, and the network's output is a list of detected boxes with a label and a confidence score. Then, detected objects will be classified. Non-human objects will

be removed by considering the label of the detected object. After that, a confidence score will be filtered by considering the threshold that is set in the configuration step. Finally, *NMSBoxes()* performs non-maximum suppression, which will reduce overlapping detected boxes.

The last process is determining social distancing. After a list of detected object boxes is filtered, the distance between each box will be calculated by using the formula which is based on Euclidean distance¹. If the value of the calculated distance is lower than the threshold, this means that the couple is too close, and they are breaking social distancing rule. The application will change the box's colour to red. In contrast, if the value of the calculation is greater than the threshold, there is no breaching of social distancing rule, and the box's colour will be changed to green.

3.6 Parallelisation

Multithreading is used to reduce the processing time, which can achieve nearly real-time processing. The strategy of multithreading is dividing an input video into frames, and assign frames to threads by considering a number of available cores.

There are two things that will be considered while the application is performing multithreading to avoid overheads. Firstly, Input/Output (I/O) operation must be avoided by threading. Secondly, all variables should be considered due to limited memory. For variables, that consume a large space of heap, will be initiated as static by using static block. In addition, the usage of the short-lived variable will be reduced to avoid garbage collecting. Furthermore, a task will be recycled after thread finished processing the given frame.

To begin with, multithreading in Java, a processor manager is implemented for managing threads efficiently. To ensure that the processor manager will be created at once, the processor manager is designed with singleton pattern and static block. Thread pool and queues are fundamental operators in the processor manager. As can be seen in figure 3.3, frames in Mat format will be assigned as a task and queued in TaskQueue. Then, tasks will be mapped with thread orderly by the thread pool. A working thread will process the given task. After a thread finished the given task, the thread will write a log regarding social distancing detection, and notify the processor manager. Then, a task will be recycled to free up memory. When the processor manager receives a notification from workers, a processed frame will be retrieved and ordered in the priority queue. After the process manager retrieved all processed frames, the process manager will notify the main activity. All of these processes are run in the background to avoid a frozen application.

On the other hand, multithreading in C++ is slightly different. The central concept of multithreading is the same as Java, except thread management. Thread pool is implemented in Java to manage and handle threads, while parallelism framework in OpenCV is used in C++. This framework is compiled with Threading Building Blocks (TBB), which is developed by Intel.

```

1  virtual void operator()(const cv::Range& range) const{
2      for (size_t i = range.start; i < size; i += threadNo){
3          cv::Mat &frame = *(cv::Mat *) frames[i];
4          ImageProcessor::process(frame);
5      }
6  }
```

¹an explanation was given in chapter 2

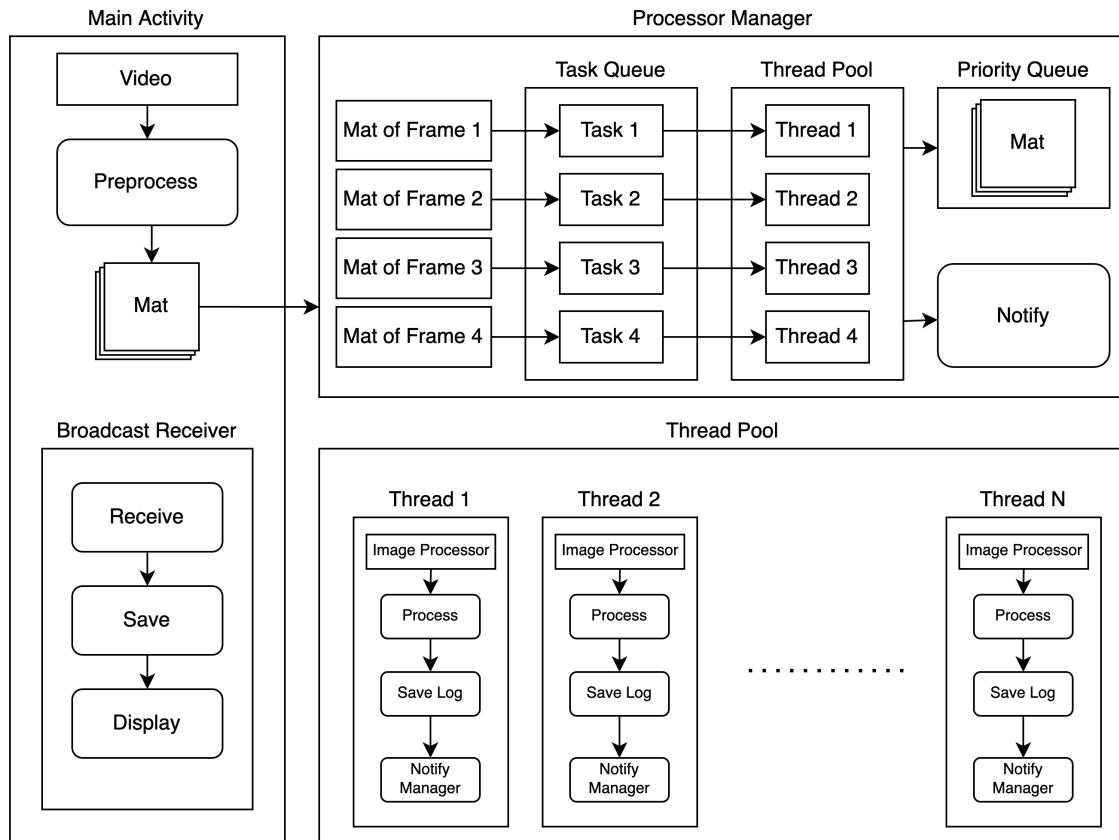


Figure 3.3: Parallel Computing Diagram

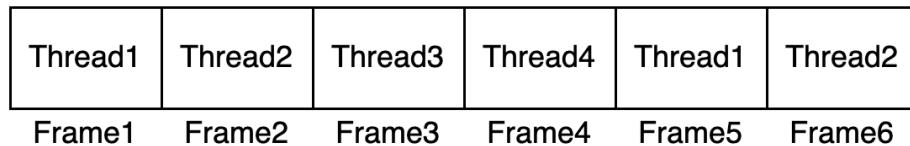


Figure 3.4: Multithreading in C++

To reduce the processing time, the threads will process the frame which has the same index as itself, and the index will be increased by the number of total available cores. For example, as can be seen from figure 3.4, the first thread will process the first frame and the fifth frame. Theoretically, if the frame rate of the video is 30 FPS and there are four threads, the first thread will have 100 milliseconds to finish processing the first frame before moving to the fifth frame.

3.7 Frame Rendering

The maximum processing frame rate of this application is 10 FPS, which will be discussed in chapter 4. At this frame rate, video cannot be displayed smoothly in live camera; thus, there are steps that help video can be displayed in a higher frame rate. First of all, when a frame is streamed

from the camera, the frame will be considered to be processed or not be processed. Processing will be determined from the capability of the device. In other words, the maximum processing frame rate will be used as a threshold. If a number of processed frames in 1 second exceeds the threshold, the frame will not be sent to the processor manager. After that, the main activity will check a processed frame in a priority queue from processer manager. If there is a processed frame in the priority queue, the processed frame will be displayed to the screen. Otherwise, the main activity will read the last log of the processed frame and retrieve all detection rectangles. Then, the main activity will draw those rectangles on the incoming frame and display it on the screen.

Chapter 4

Evaluation and Testing

This chapter presents an evaluation of the application, which is divided into three parts. The first part will show variables that must be controlled for the reliability and stability of the result. The second part is going to evaluate the application's performance, which are comparisons among models, programming languages, and technologies. The last section is going to show a usability testing of this application.

4.1 Controlled Variables

To ensure the result of the performance will not be varied by other factors, some variables must be controlled as following:

1. All testing cases will be run on Samsung Galaxy S10+.
2. All running background applications will be closed, and memory will be freed before testing.
3. To prevent CPU's speed is limited, a power management mode will be set to "Optimised".
4. Total number of frames in the testing video will be set to 31 frames.
5. A testing video resolution will be set to 540x480 pixels.

4.2 Performance Evaluation

In this section, the performance of detection models will be compared, and the result will be discussed. There are three sections of discussion. The first section will show the result of processing a single frame. The second section will compare the performance among a number of threads. The last section will discuss the result of NEON instruction and improvement. Computation time and frame rate are used as a measurement to evaluate the performance of the application. Frame rate is a number of frames that application can process in 1 second. The sample of pre-recorded file is obtained from Ben Benfold and Ian Reid [3].

4.2.1 Single Frame Comparison

In this section, the performance of 2 detection models will be compare, regardless of other tools and techniques. To evaluate the performance, each model will process on a single frame. There are two different resolutions used as inputs for comparing the performance and understanding the variation of calculation time.

Model	YOLO		SSD	
	960x540	540x480	960x540	540x480
Size	960x540	540x480	960x540	540x480
Total Process Time (second)	4.235	3.827	0.337	0.323
Forward Propagation per frame (second)	3.456	3.019	0.284	0.278
Forward Propagation per frame (percentage)	81.61%	78.89%	84.27%	86.07%

Table 4.1: Picture Processing Performance

As can be seen from table 4.1, the processing time of YOLO model significantly increases when the size of the picture is greater. In contrast, MobileNet SSD is able to process the given picture faster than YOLO model. The processing time of MobileNet SSD slightly increases when the size of the picture is increased.

4.2.2 Multithreading

In this section, the performance of both models will be evaluated with multithreading technique, and the evaluation will be divided into three parts: sequential computing, YOLO with multithreading, and MobileNet SSD with multithreading. As mentions previous section, in this evaluation, controlled variables are set. The number of frames in the testing video will be set to 31 frames.

For the first part, an application will process the testing video sequentially and measure the performance. This measurement will be compared to multithreading and evaluate the improvement of the performance. As a result in table 4.2 and 4.3, YOLO model took 102.972 seconds to processed 31 frames video, while MobileNet SSD model took 7.132 seconds.

Model	YOLO			
	Sequential Computing	Parallel Computing		
		1 Thread	2 Threads	4 Threads
Total Process Time (second)	102.972	117.805	96.415	92.242
Garbage Collector (second)	-	0.102	0.280	2.024
Process Time without GC	-	117.703	96.136	90.218
Forward Propagation (Total)	79.097	-	-	-
Forward Propagation (Average)	2.553	2.872	4.840	9.231
Forward Propagation (Min)	2.213	2.564	4.003	5.478
Forward Propagation (Max)	2.693	3.092	6.436	12.566
Number of frame	31	31	31	31
Process per frame (second)	3.322	3.800	3.110	2.976
Improvement			18.16%	21.70%
				15.59%

Table 4.2: Video Processing with YOLO Model with official build

Then, a multithreading technique is implemented to increase performance and achieve real-time processing. To evaluate the improvement of multithreading, a number of processors will be doubled as follows: 1, 2, 4, and 8. In the testing device, there are 8 physical cores, so it can effectively process up to 8 threads.

Model	MobileNet SSD				
	Sequential Computing	Multithreading			
		1 Thread	2 Threads	4 Threads	8 Threads
Total Process Time (second)	7.132	8.237	6.873	6.270	5.064
Garbage Collector (second)	-	-	-	-	-
Process Time without GC	-	-	-	-	-
Forward Propagation (Total)	7.019	-	-	-	-
Forward Propagation (Average)	0.226	0.235	0.401	0.738	1.133
Forward Propagation (Min)	0.218	0.212	0.353	0.406	0.466
Forward Propagation (Max)	0.243	0.320	0.456	1.477	2.582
Number of frame	31	31	31	31	31
Process per frame (second)	0.230	0.266	0.222	0.202	0.163
Improvement			16.56%	23.88%	38.52%

Table 4.3: MobileNet SSD Model with OpenCV official build in Java

As shown in table 4.2, the performance of 2 threads is improved only 18.16 per cent, and it reaches the best performance at 21.7 per cent by using 4 threads. However, the performance of 8 threads is worse than 2 threads. One of the factors is the Garbage Collection (GC). The application was frozen while GC is collecting garbage. GC is not collecting only short-lived objects but long-lived objects as well, and GC is more often collect garbage when the number of threads is increased.

As shown in figure 4.2, memory was allocated by double and array of double, and GC was freeing these allocation 10 times within 5 seconds. Consequently, CPU usage is dropped when GC is working. This problem can be seen in figure 4.1. The progress status will be green when a thread is working. It will be yellow when a thread is interrupted by GC, and it will be grey when a thread has no activity.

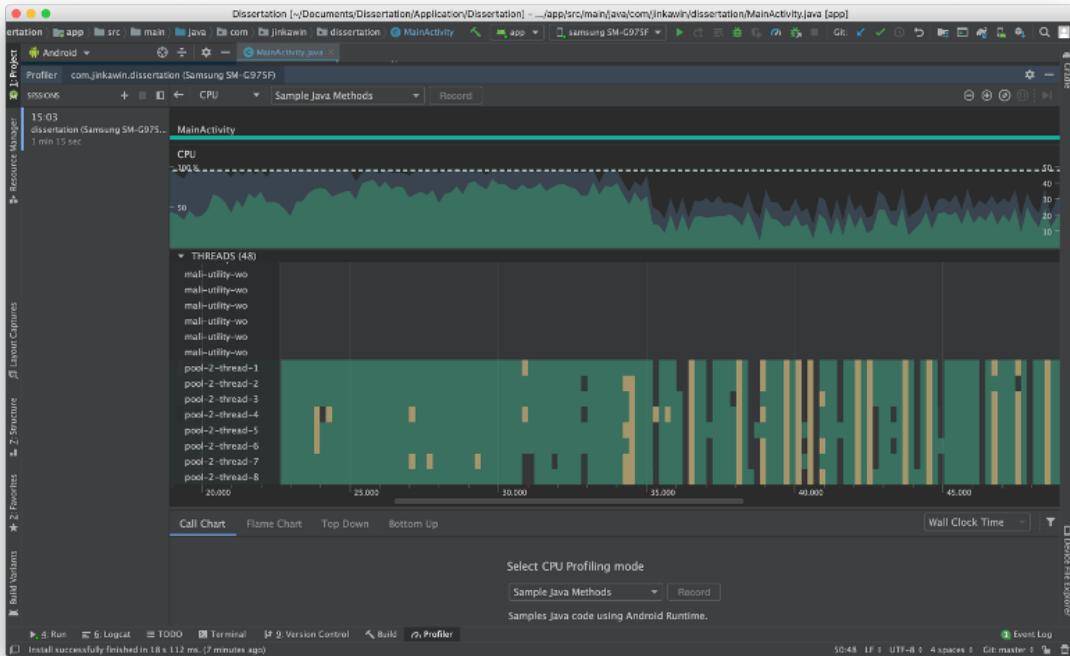


Figure 4.1: CPU Usage of YOLO Model with 8 threads

For the MobileNet SSD model, the calculation time of 2 threads is improved only 16.56 per cent,

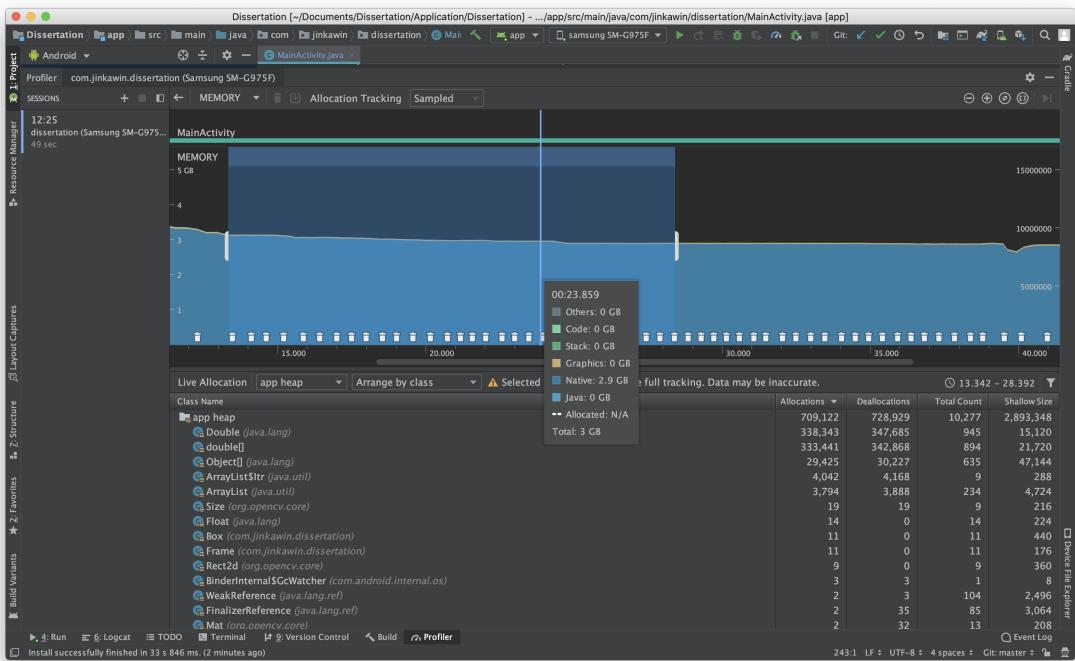


Figure 4.2: Memory Usage of YOLO Model with 8 threads

and 8 threads give the best performance with 38.53 per cent as can be seen in table 4.3. Objects are not collected by GC in this model, but memory will be freed after processing is finished. The improvement computation time of 8 threads by using MobileNet SSD model is better than YOLO model, but the speed-up time of both models still lower than an ideal time.

In summary, the overall performance of YOLO model with multithreading is slightly improved, and it becomes worse when using 8 threads. Regarding this examination, one of the obvious factors of these problems is GC. In contrast, the speed-up scale of MobileNet SSD is better than YOLO model because there is no GC during processing. YOLO model uses more memory than MobileNet SSD in term of variables in double type.

4.2.3 Android Native Development Kit

As mentioned in the chapter 3, detection process and distance measurement can be written in C++ to achieve higher performance. As it can be seen in table 4.4, the performance of using 2 threads is improved 44.79 per cent when compared to using 1 thread, and the performance is fastened up to 58.54 per cent when using 8 threads. However, there are 2 issues of this implementation. The first issues is the forward propagation time. Theoretically, the processing time of sequential computing and using a single thread should be the same. In contrast, the forward propagation time of single thread was doubled, which causes total process time increase. Due to the thread management system in C++ is different when compared to Java. In C++, the thread is managed by OpenCV's parallelism framework, while the thread pool is used in Java which is recommended by Android's document [1]. The second issue is the total process time. Although writing in C++ is able to achieve theoretical speed-up, the overall performance is slightly better when compared to Java.

Model	MobileNet SSD				
	Sequential Computing	Multithreading			
		1 Thread	2 Threads	4 Threads	8 Threads
Total Process Time (second)	6.773	11.949	6.597	6.150	4.954
Garbage Collector (second)	-	-	-	-	-
Process Time without GC	-	-	-	-	-
Forward Propagation (Total)	6.659	-	-	-	-
Forward Propagation (Average)	0.215	0.382	0.408	0.613	0.970
Forward Propagation (Min)	0.198	0.363	0.394	0.405	0.407
Forward Propagation (Max)	0.236	0.401	0.421	1.043	2.691
Number of frame	31	31	31	31	31
Process per frame (second)	0.218	0.385	0.213	0.198	0.160
Improvement			44.79%	48.53%	58.54%

Table 4.4: MobileNet SSD Model with OpenCV official build in C++

4.2.4 NEON Instruction

OpenCV library provides a shared library, which is officially built by OpenCV. However, the library is built to support all CPU chipsets, and many features and conditions were flagged during building. The library is needed to be manually built from scratch to evaluate the performance of NEON. OpenCV shared library was manually built into two versions: version with NEON and version without NEON. To maximise the performance of the NEON version, OpenCV is forced to be built with NEON instruction regardless of any condition, and it will support only ARMv8-A 64-bit architecture.

Model	MobileNet SSD without NEON				
	Sequential Computing	Parallel Computing			
		1 Thread	2 Threads	4 Threads	8 Threads
Total Process Time (second)	17.308	19.030	15.172	11.797	10.624
Garbage Collector (second)	-	-	-	-	-
Process Time without GC	-	-	-	-	-
Forward Propagation (Total)	17.193	17.848	-	-	-
Forward Propagation (Average)	0.555	0.576	0.926	1.416	2.462
Forward Propagation (Min)	0.519	0.545	0.582	0.756	1.310
Forward Propagation (Max)	0.586	0.654	1.412	2.593	5.308
Number of frame	31	31	31	31	31
Process per frame (second)	0.558	0.614	0.489	0.381	0.343
Improvement			20.27%	38.01%	44.17%

Table 4.5: MobileNet SSD Model with OpenCV manully build (**without** NEON)

After OpenCV library is manually built and forced to be compiled with NEON instruction, the performance of human detection is significantly improved. As can be seen in table 4.5 and 4.6, the sequential computing took 4 second for processing human detection, which is faster than OpenCV without NEON 76.85 percent. In addition, multithreading can process 31 frames of the video up to 2.890 seconds, which is faster than OpenCV without NEON 72.79 per cent and OpenCV official build 42.93 per cent. For this performance, video can be processed 10.75 frames per second. In addition, according to rendering technique in chapter 3, a real-time video from the camera can display up to 25 frames per second.

Thus far, these results were not achieved theoretical, and this application did not achieve processing and displaying 30 FPS even if multithreading and NEON are implemented. This underachieved performance could be caused by mulithreading itself. A forward propagation time, which took

Model	MobileNet SSD with NEON				
	Sequential Computing	Multithreading			
		1 Thread	2 Threads	4 Threads	8 Threads
Total Process Time (second)	4.006	6.079	4.208	3.127	2.890
Garbage Collector (second)	-	-	-	-	-
Process Time without GC	-	-	-	-	-
Forward Propagation (Total)	3.927	4.950	-	-	-
Forward Propagation (Average)	0.126	0.159	0.225	0.339	0.645
Forward Propagation (Min)	0.117	0.128	0.131	0.190	0.266
Forward Propagation (Max)	0.135	0.250	0.295	0.596	1.798
Number of frame	31	31	31	31	31
Process per frame (second)	0.129	0.196	0.136	0.101	0.093
Improvement			30.78%	48.56%	52.46%

Table 4.6: Video Processing with MobileNet SSD Model (**with NEON**)

most of the computation time, is doubled when the number of threads is increased. One of the issues that emerge from this finding is parallelisation in the OpenCV library. Some function underneath the forward propagation function is parallelised by using multithreading. Then, when a frame processing thread is created, a number of threads may exceed the number of physical cores. Thus, threads are interrupted, and it caused a context switching overhead. Besides, the calculation time of using a single thread is always greater than the calculation time un sequential computing and it could be caused by this problem.

Chapter 5

Conclusion

This project was undertaken to build a social distancing detection application on the Android operating system, and maximising detection performance by using various technologies and techniques.

The results of this implementation show that the combination of MobileNet SSD model, manually built OpenCV with NEON, and using 8 threads gave the highest performance. The computation time of a pre-recorded single frame took 93 milliseconds or 10.752 frames per second, and this application can display 25 frames of video in 1 second on a live camera with the help of rendering technique. The accuracy of YOLO model was higher than MobileNet SSD model. However, computational time and memory usage of YOLO model was worse than MobileNet SSD. The YOLO model computation time of 31 frames by using 8 threads was 99.441 seconds, while MobileNet SSD model took 5.064 seconds. In addition, if NEON is implemented MobileNet SSD mobile is able to process 31 frames in 2.890 seconds.

5.1 Limitation

Limitations of this application have been found during development. The first limitation is the resolution of the video. Regarding performance, this application has to process video in low resolution, which is 540x480 pixels. Processing high-resolution video needs more computing power and resources to process in real-time, which is limited in mobile phone. Secondly, there is trade-off between computation time and accuracy. YOLO model is able to detect human more efficiently than MobileNet SSD. However, YOLO model has more computation time and resource consumption. Finally, the main computation part cannot be processed by using GPU because OpenCV does not support Samsung Galaxy S10+'s GPU. The GPU of Samsung Galaxy S10+ is ARM Mali. Currently, OpenCV supports 3 GPUs: Intel, Nvidia, and AMD.

5.2 Future Work

These findings show that the performance cannot be enhanced further by using CPU with OpenCV library. Increasing accuracy and performance, computing power is needed more. Using GPU to

forward computing propagation may significantly reduce the processing time. Regarding GPU supporting an issue, the library must be changed from OpenCV to others that support GPU implementation, such as TensorFlow. In addition, OpenCL can be used with TensorFlow to do parallelisation. These can be implemented in Native code to maximise the performance and may achieve higher accuracy, FPS, and resolution.

Appendix A

Existing Application



Figure A.1: Object Detector



Figure A.2: TensorFlow Object Detection

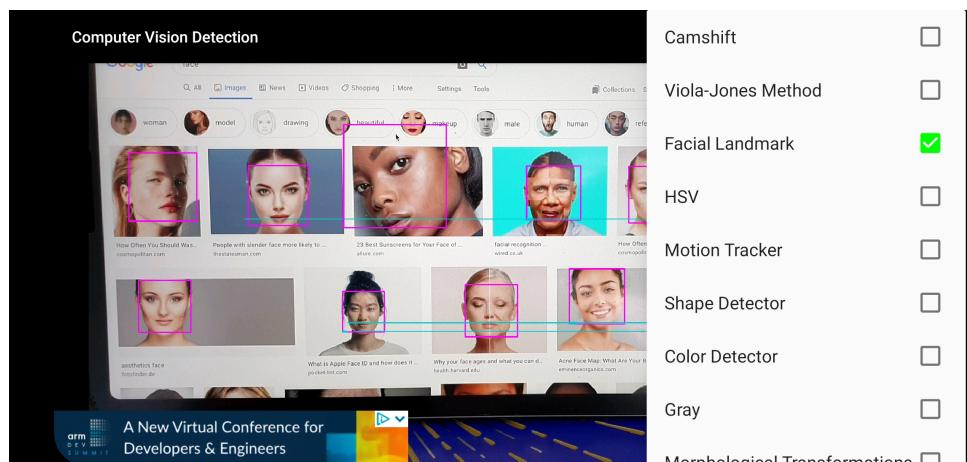
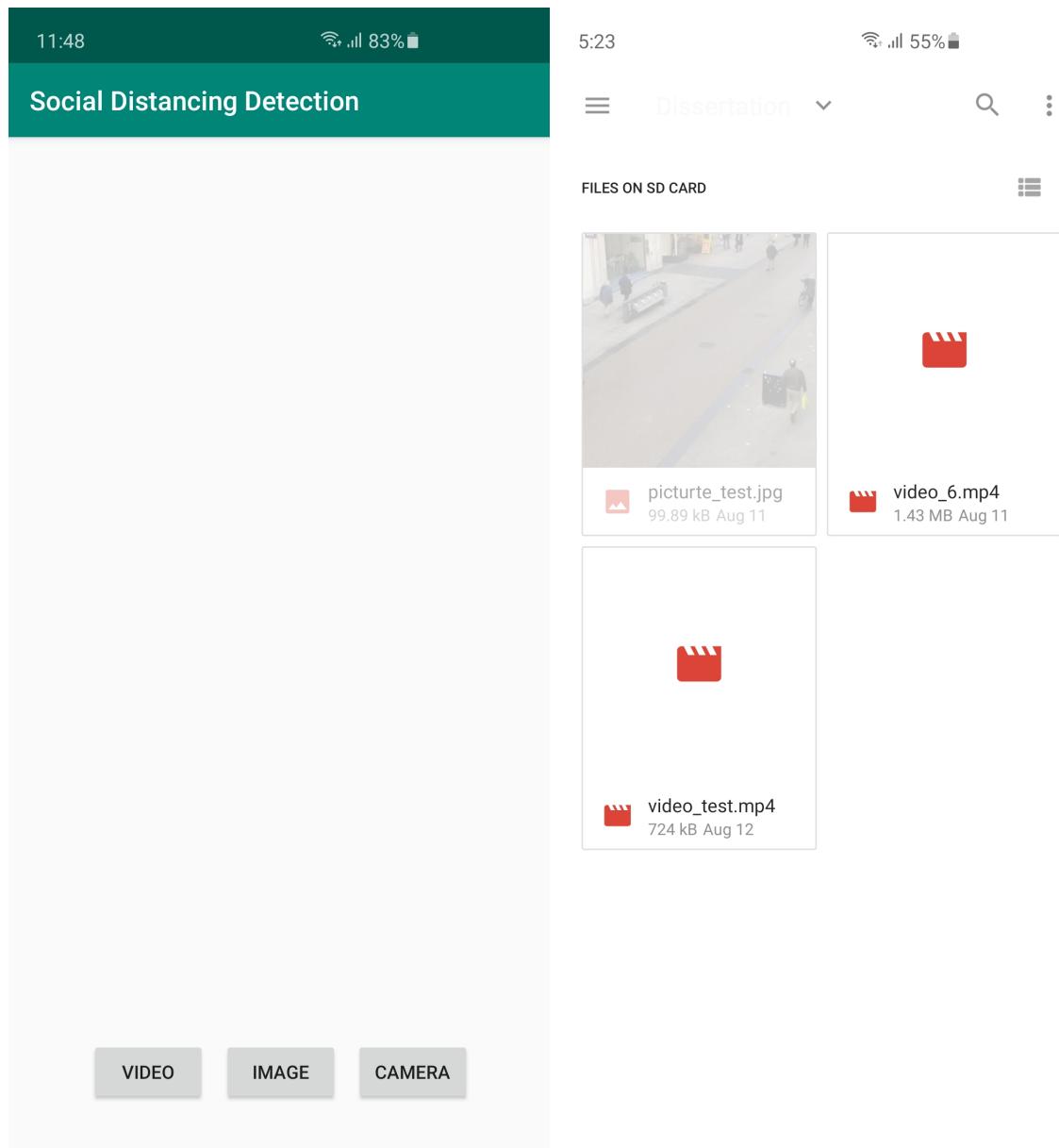


Figure A.3: Computer Vision Detection

Appendix B

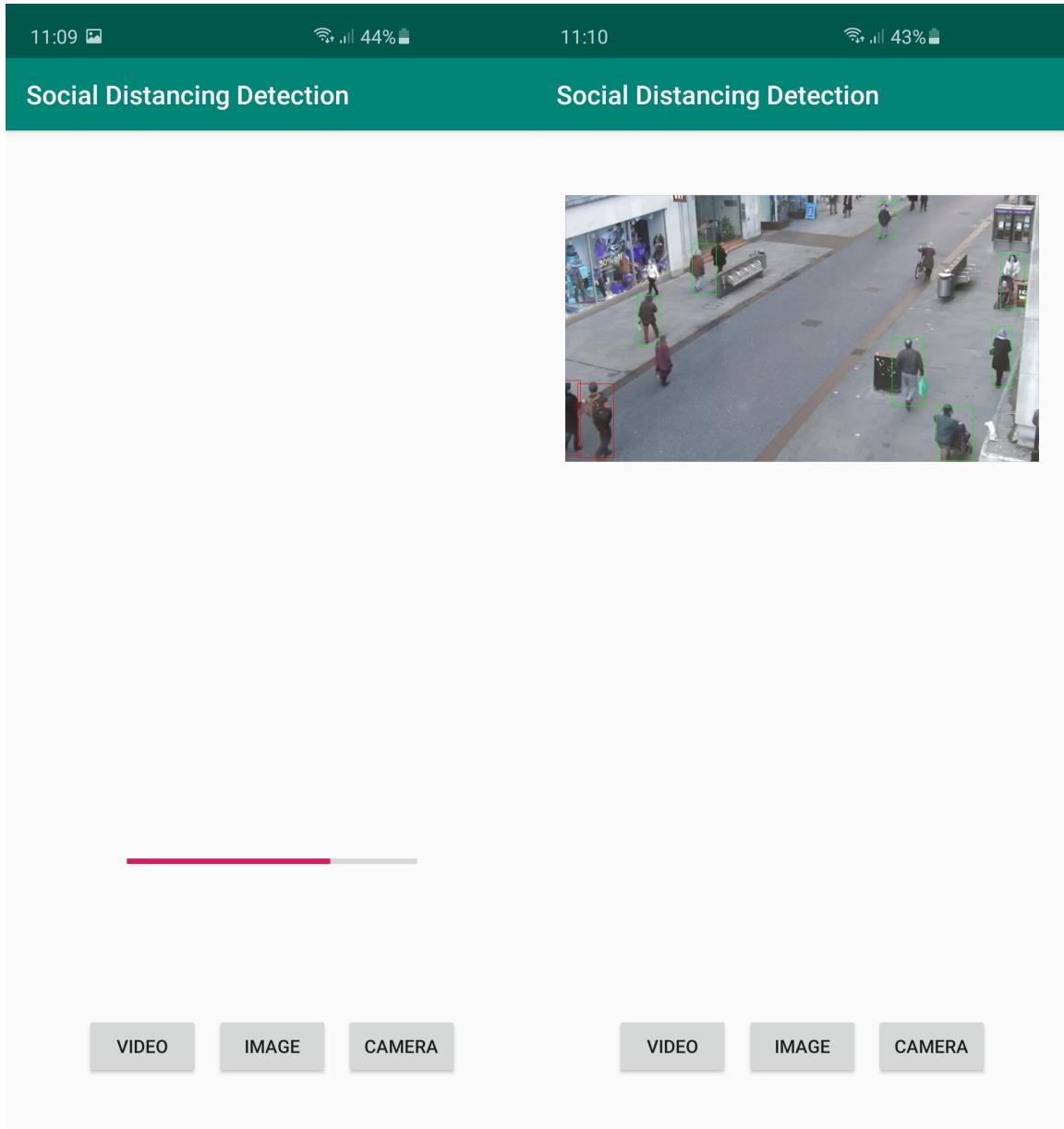
Application Screenshot



(a) Main Menu

(b) Choosing Existing File from Gallery

Figure B.1: Application Interface



(a) Status Bar While Processing

(b) Display a processed image or video

Figure B.2: Processing pre-recorded file

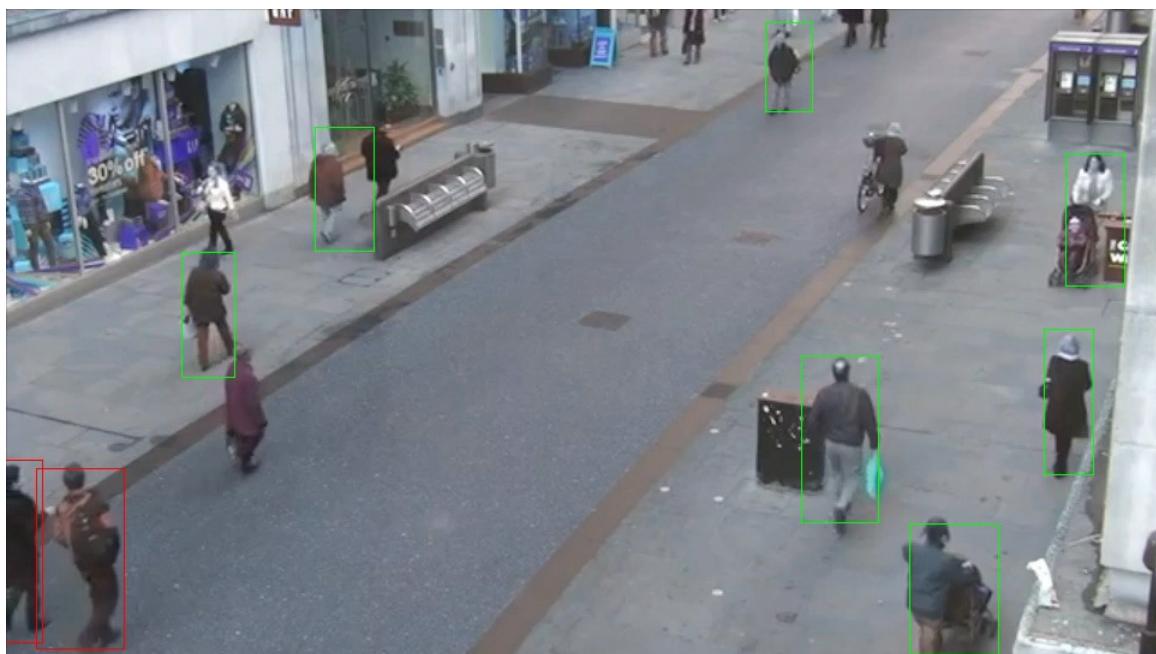


Figure B.3: Social Distancing Detection from Picture

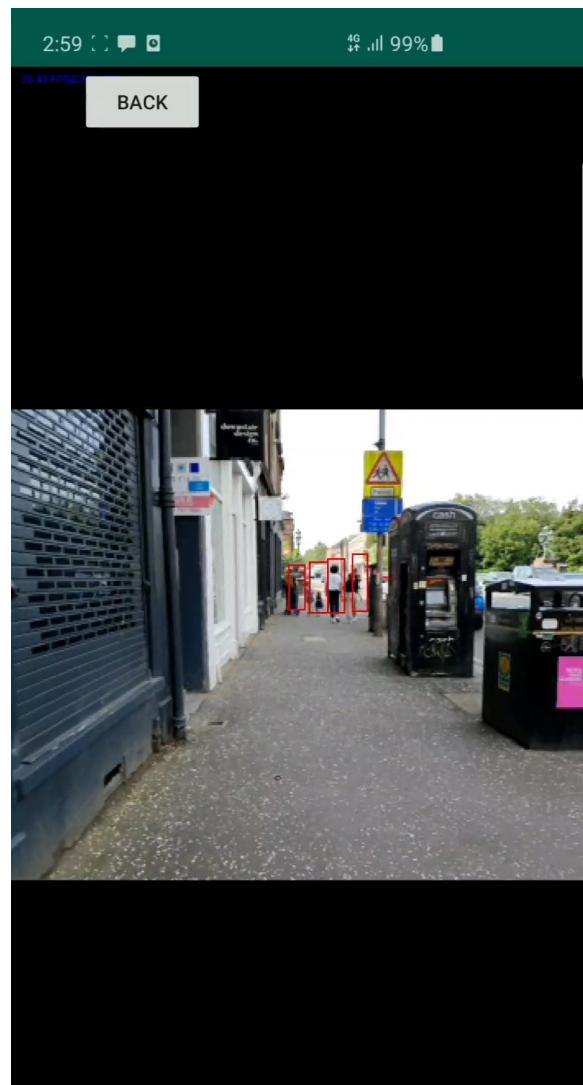


Figure B.4: Social Distancing Detection by Using Camera

Bibliography

- [1] Get started with the ndk [online], 2020. [Accessed 2020-08-18]. Available from: <https://developer.android.com/ndk/guides>.
- [2] Ole J. Benedictow. *The Black Death, 1346-1353: the complete history*. Boydell Press, Woodbridge, Suffolk, 2004.
- [3] Ben Benfold and Ian Reid. Guiding visual surveillance by tracking human attention. In *Proceedings of the 20th British Machine Vision Conference*, September 2009.
- [4] Min W. Fong, Huizhi Gao, Jessica Y. Wong, Jingyi Xiao, Eunice Y. C. Shiu, Sukhyun Ryu, and Benjamin J. Cowling. Nonpharmaceutical measures for pandemic influenza in nonhealthcare settings—social distancing measures. *Emerging infectious diseases*, 26(5):976–984, 2020.
- [5] Centers for Disease Control and Prevention. Social distancing, 2020.
- [6] Jennifer A. Horney, Zack Moore, Meredith Davis, and Pia D. M. MacDonald. Intent to receive pandemic influenza a (h1n1) vaccine, compliance with social distancing and sources of information in nc, 2009. *PloS one*, 5(6):e11226, 2010.
- [7] Gurucharan M. K. Covid-19: Ai-enabled social distancing detector using opencv [online], 2020. [Accessed 2020-08-18]. Available from: <https://towardsdatascience.com/covid-19-ai-enabled-social-distancing-detector-using-opencv-ea2abd827d34>.
- [8] Ann H. Reid, Raina M. Lourens, Ruixue Wang, Guozhong Jin, Jeffery K. Taubenberger, and Thomas G. Fanning. Molecular virology was the 1918 pandemic caused by a bird flu? was the 1918 flu avian in origin? (reply). *Nature*, 440(7088):E9–E10, 2006.