



University of Glasgow | School of
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

Date of submission placed here

Abstract

abstract goes here

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: _____ Signature: _____

Acknowledgements

acknowledgements go here

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	13
3.1	MoSCoW Statement	13
3.1.1	Must have	13
3.1.2	Should have	13
3.1.3	Could have	13
3.1.4	Won't have	14
3.2	Preparation	14
3.3	Java Native Interface	14

3.4	Social Distancing Detection	15
3.5	Parallelisation	17
3.5.1	Multithreading with Java	17
3.5.2	Multithreading with C++	18
4	Evaluation and Testing	19
4.1	Controlled Variables	19
4.2	Performace Evaluation	19
4.2.1	Single Frame Comparison	19
4.2.2	Multithreading	20
4.2.3	Android Native Development Kit	22
4.2.4	NEON Instruction	23
4.3	Usability Testing	24
5	Conclusion	26
5.1	Limitation	26
5.2	Future Work	26
A	First appendix	27
A.1	Section of first appendix	27
B	Second appendix	28

Chapter 1

Introduction

1.1 Motivation

Humanity have been faced several pandemics over hundreds years, and many lives are threatened. For example, there were the black death in 1346 and the flu pandemic in 1918, and millions 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 was 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, an ability of computable devices have been improved which is capable of executing very complex tasks. Mobile phone, which is one of a high competitive market, become a powerful device. People use a mobile phone for many purposes, such as entertainment, education or business. Likewise, in an application development, there are advantages of the mobile device. The first advantage is a performance. As aforementioned competition, hardware of mobile phone has been improved over years, including: CPU, GPU, and Memory. Thus, a capability of the mobile phone is sufficient for performing heavily calculation tasks. Then, mobile phone is portability. Most of mobile phones provide necessary features, such as 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, an 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, 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 technique.
- 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 a background knowledge of Android application, object detection, distance calculation, and parallel computing.
2. Chapter 3 shows an overview of the system, class diagram, and design.
3. Chapter 4 describes implementation details and technical development.
4. Chapter 5 analyses development's problems and the performance evaluation.
5. Chapter 6 concludes results of the project, limitation, and future works.

Chapter 2

Background

This chapter aim to explain technologies, tools, and specification of a device that are used in this project. Then, this chapter gives an analysis of related applications.

2.1 Threading in Android

Threads in Android development are categorised into 2 main categories: a main thread and a worker thread. A main thread or UI thread is a thread that dispatch events of user interface widgets. The events is dispatched regarding to Android's activity lifecycle, and all events are managed by only 1 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. In addition, an application will show "Application not Responding" when UI thread is blocked more than 5 seconds [1].

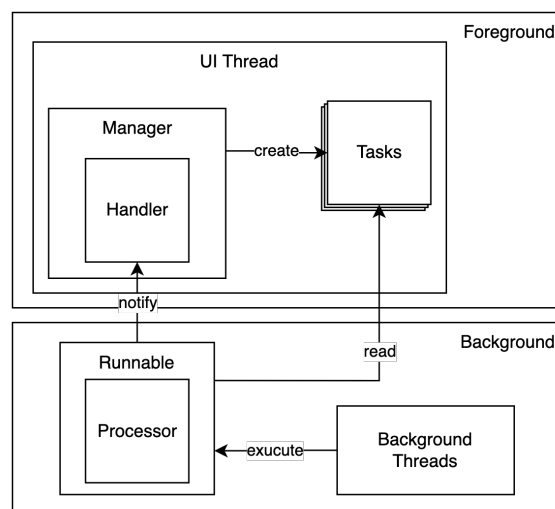


Figure 2.1: Threading Overview

To avoid freezing application, a working thread is separated from the UI thread, 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— -20 is the highest priority, and 19 is the lowest priority. 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 2 parts: foreground and background. Tasks are created by a manager which is running on the UI thread. Then, Runnable, a component that is able to 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.

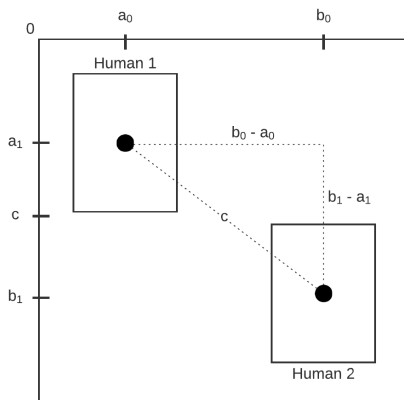
2.2 Distance Calculation

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.

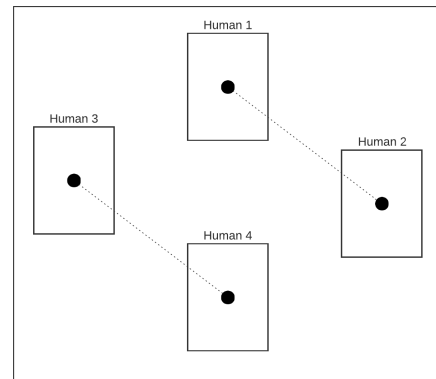
$$d = \sqrt{(a_0 - b_0)^2 + (D/c) \times (a_1 - b_1)^2}$$

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

However, three-dimensional space are captured into two-dimensional image, so depth and perspective are concerned as can be seen in figure 2.2a. Thus, a couple 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 2 variables will determine the depth of people in the image.



(a) Distance calculation



(b) A difference of 2 distances

Figure 2.2: Determining Social Distancing

For example, according to the figure 2.2b, if distance is calculated without calibration, 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 2 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 Android operating system. TensorFlow's library is used in this application along with MobileNet model. This application is able to detect object in real-time from the camera.

2.4.3 Computer Vision Detection

Computer Vision Detection application operates on Android operating system. This application is able to process real-time video from 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. In addition, this application is able to detect human faces and smiling faces. This application is able to process around 13 frames per second.



Figure 2.3: Object Detector

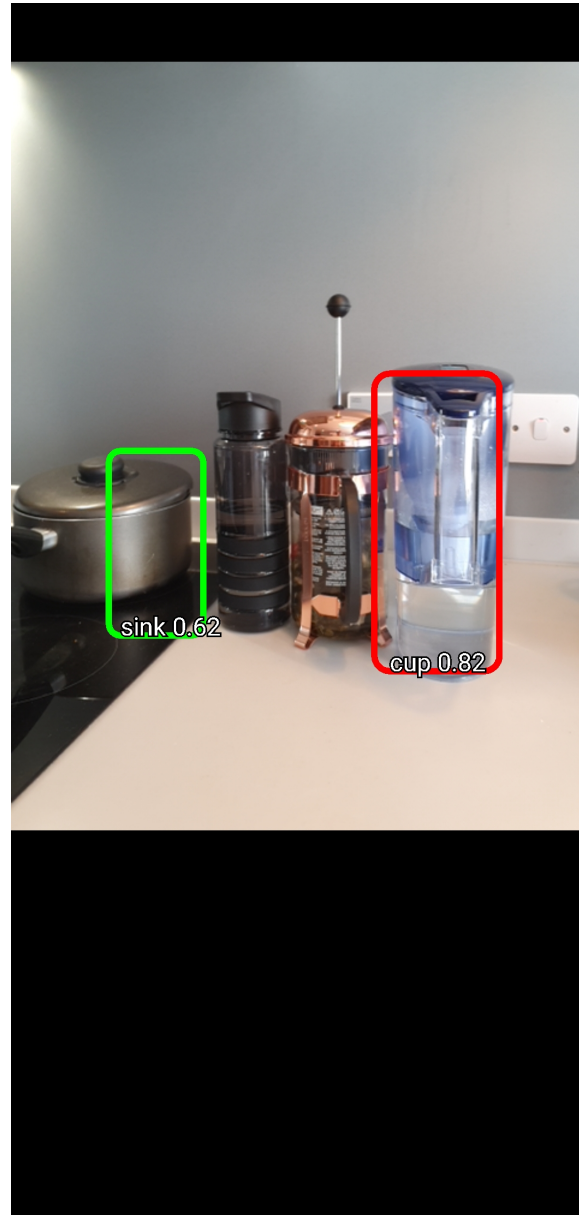


Figure 2.4: TensorFlow Object Detection

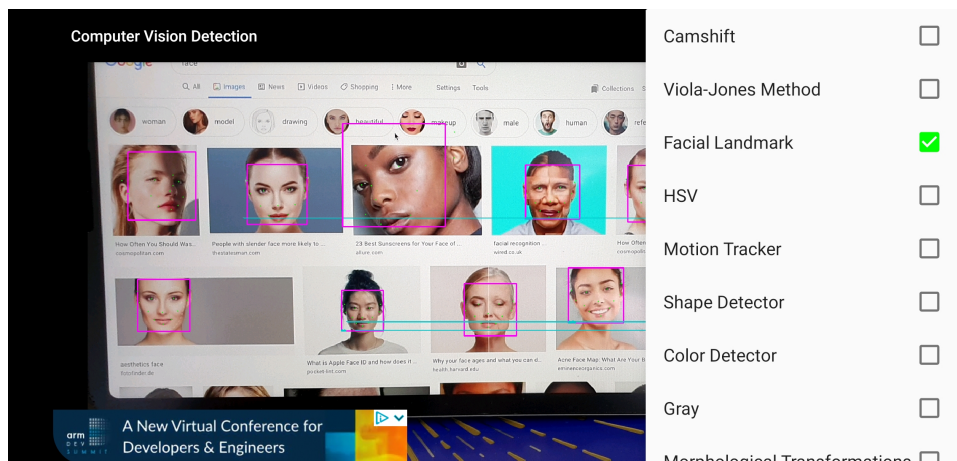


Figure 2.5: Computer Vision Detection

Chapter 3

Design and Implementation

In this chapter, the implementation details will be explained in many aspects, including Java Native Interface, human detection, and distance calculation.

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 determind 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 techonology.
- 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 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 shared library, is used for image processing and object detection. For object detection model, there are 2 models are used: You Only Look Once and MobileNet SSD.

3.3 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 JNI layer is being 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 native library, which is able to achieve a higher performance. Thus, executing both tasks in the JNI and C++ layer gains a better performance. Furthermore, there are 2 advantages of 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 an overhead. Thus, implementing JNI manually 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 Java layer. Memory addresses of pre-processed frame in Mat format will be pass as 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

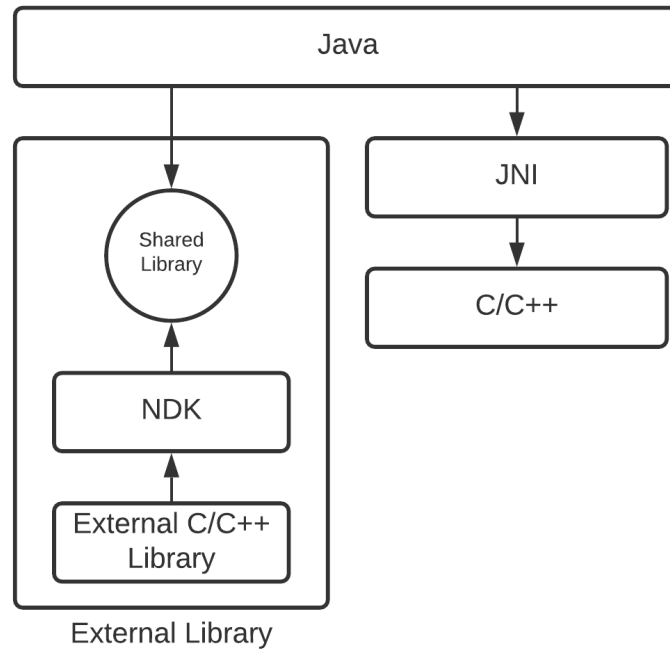


Figure 3.1: Application Layers

```

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

3.4 Social Distancing Detection

There are 3 main processes is implemented to determine social distancing violation from the image and video.

The first process is pre-processing the given image, video, or video stram. 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, colour will be converted, which depends on a detection model. As mentioned in section 4.1, there are 2 detection models are 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.

The second process is object detection. Detection models are setup 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, threshold will be set for determining the confidence score of the detected object. The confidence score

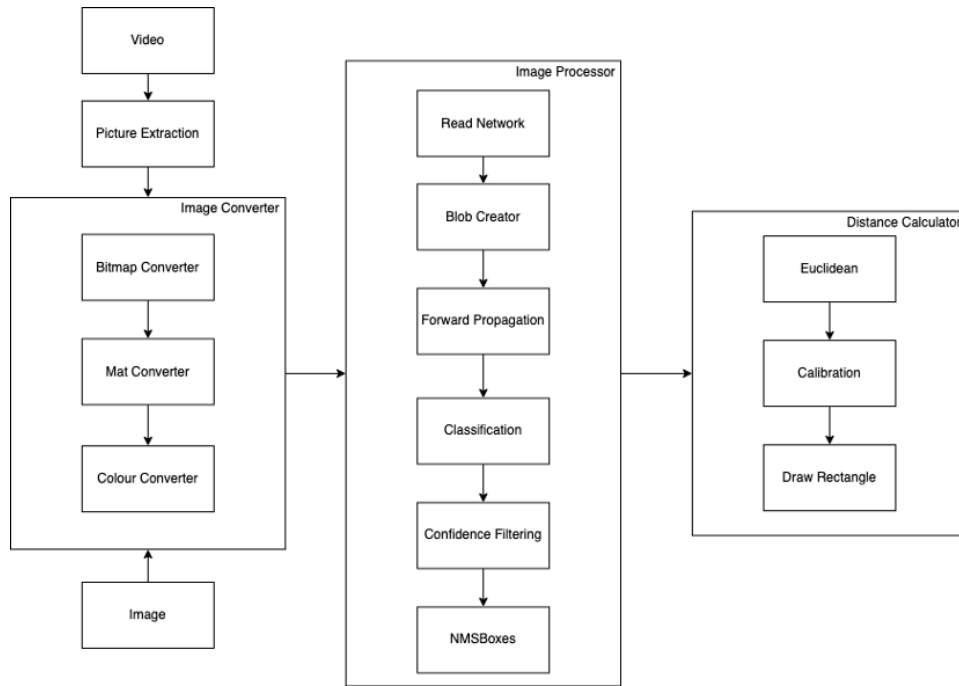


Figure 3.2: Detection System Overview

threshold of YOLO model will be set to 0.5 or 50 percent. 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 percent. In contrast, the confidence score threshold of MobileNet SSD model will be set to 0.3 or 30 percent. Because of MobileNet SSD model has a lower ability to detect an object, the confidence threshold is set to be lower. After models are setup and configured, the image will be processed in 5 steps. For the first step, pre-processed image will be convert to input blob by passing Mat image to *blobFromImage()* function with scale factor. Blob will be used for the forward propagation in the neural network. Secondly, 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 distance ¹. If the value of the calculated distance is lower than the threshold, this mean 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 grater than threshold, there is no breaching of social distancing rule, and the box's colour will be changed to green.

¹an explanation was given in chapter 2

3.5 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.

3.5.1 Multithreading with Java

To begin with, 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 receive 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.

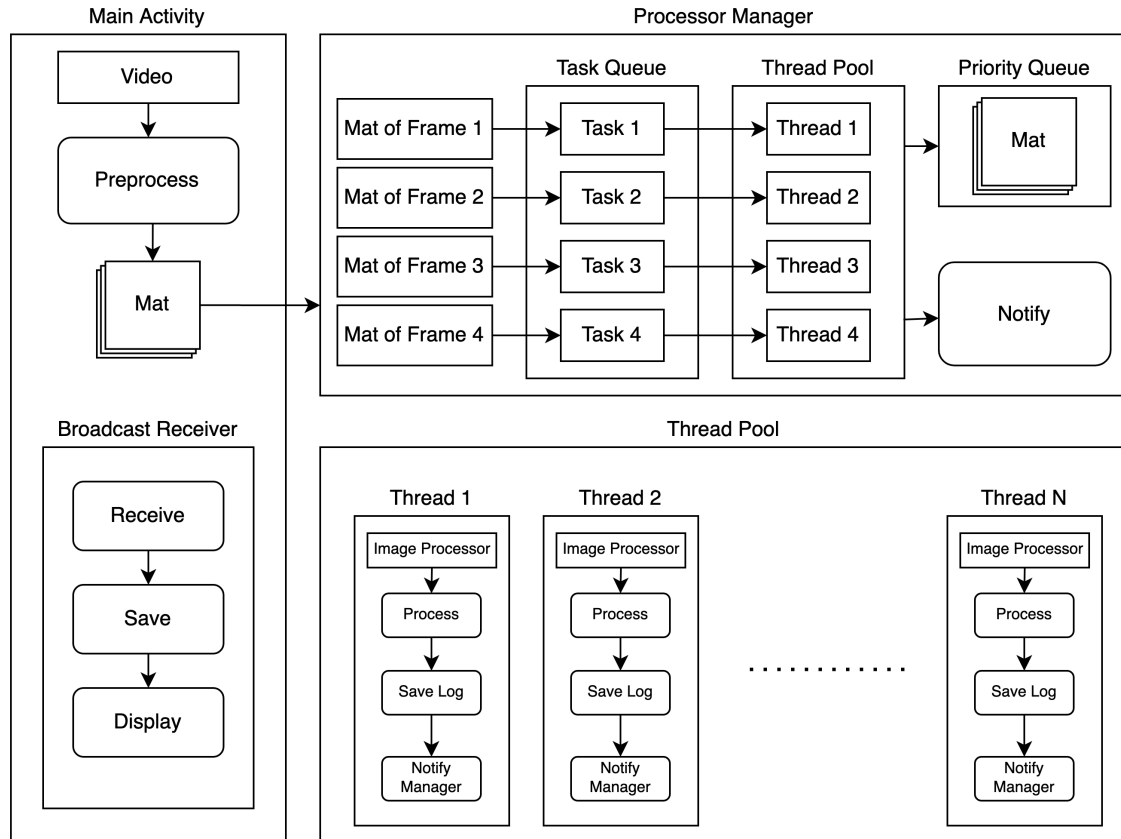


Figure 3.3: Parallel Computing Diagram

3.5.2 Multithreading with C++

```
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  }
```

- How to show processed frame from live stream?

Chapter 4

Evaluation and Testing

This chapter presents an evaluation of the application, which is divided into 3 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 "Optimized".
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 3 sections of discussion. The first section will show the result of processing single frame. The second section will compare the performance among number of threads. The last section will discuss the result of NEON instruction and improvement.

4.2.1 Single Frame Comparison

In this section, the performance of 2 detection models will be compared, regardless of other tools and techniques. To evaluate the performance, each model will process on a single frame. There are

2 different resolutions were used as inputs for comparing the performance and understanding the variation of calculation time.

Model	YOLO		SSD	
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 3 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 4.1 and 4.2, YOLO model took 102.972 seconds to processed 31 frames video, while MobileNet SSD model took 7.132 seconds. However, even if MobileNet SSD model can achieve better performance than YOLO model, the application is not able to process a video in real-time.

Model	YOLO				
	Sequential Computing	Parallel Computing			
		1 Thread	2 Threads	4 Threads	8 Threads
Total Process Time (second)	102.972	117.805	96.415	92.242	99.441
Garbage Collector (second)	-	0.102	0.280	2.024	11.333
Process Time without GC	-	117.703	96.136	90.218	88.108
Forward Propagation (Total)	79.097	-	-	-	-
Forward Propagation (Average)	2.553	2.872	4.840	9.231	19.713
Forward Propagation (Min)	2.213	2.564	4.003	5.478	14.733
Forward Propagation (Max)	2.693	3.092	6.436	12.566	21.815
Number of frame	31	31	31	31	31
Process per frame (second)	3.322	3.800	3.110	2.976	3.208
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, 8, and 16. In the testing device, there are 8 physical cores, so it can effectively process up to 8 threads.

The overall performace of YOLO model with multithreading is slightly improved. There are 2

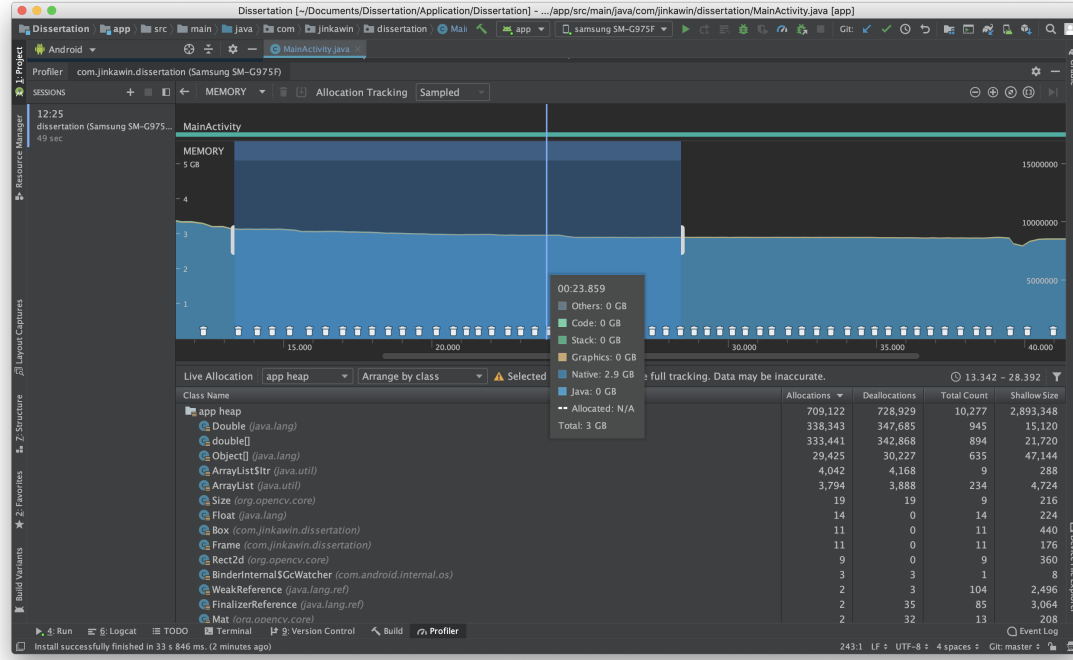


Figure 4.1: Memory Usage of YOLO Model with 8 threads

problems of YOLO model with multithreading as shown in table 4.2. The first problem is the improvement of the performance. The performance of 2 threads is improved only 18.16 percent, and it reach the best performance at 21.7 percent by using 4 threads. However, the performance of 8 threads is worse than 2 threads, which is caused by Garbage Collection. The second problem is 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 4.1, 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 the figure 4.2. The progress status will be green when a thread is working. It will be yellow when it is interrupted by GC, and it will be gray when a thread has no activity.

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: Video Processing with MobileNet SSD Model with official build

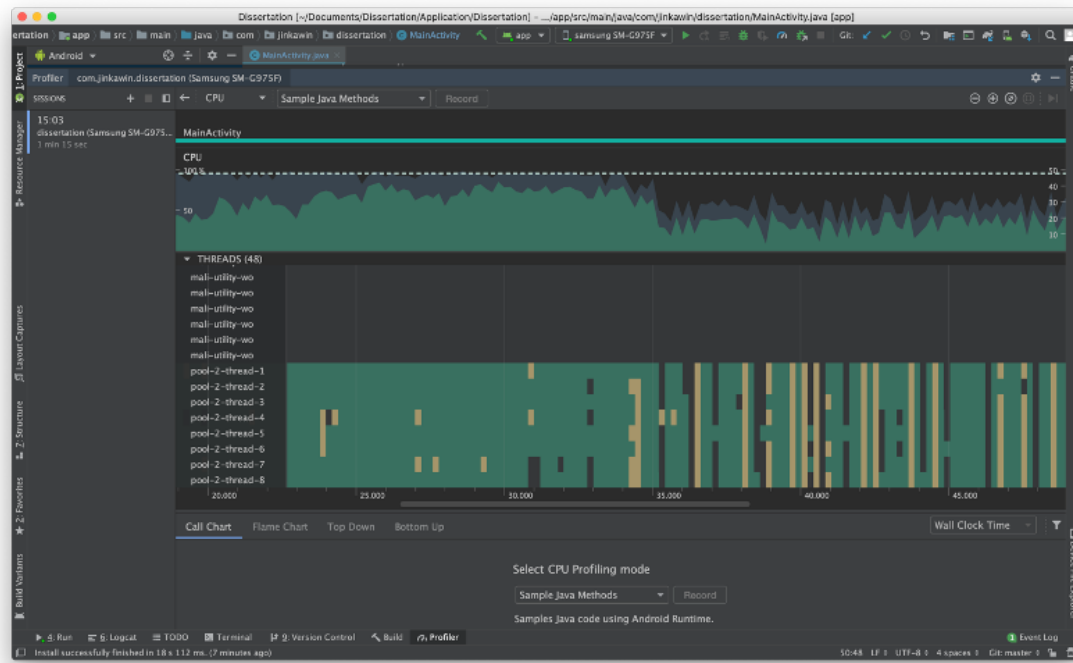


Figure 4.2: CPU Usage of YOLO Model with 8 threads

For the MobileNet SSD model, the calculation time of 2 threads is improved only 16.56 percent, and 8 threads give the best performance with 38.53 percent as can be seen in table 4.3. Objects is not collected by GC in this model, but memory will be freed after processing is finished. The improvement time of 8 threads by using MobileNet SSD model is not reduces likes YOLO model, but speed-up time of both models still lower than an ideal time.

Although MobileNet SSD model use less memeory and has a better performance than YOLO model, it is not able to achieve theoretical speed-up of Amdahl's law by using multithreading.

4.2.3 Android Native Development Kit

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: Video Processing with MobileNet SSD Model with official build

As mentioned in the chapter 4, detection process and distance measurement can be writting in C++ to achieve a higher performance. As it can be seen in table 4.4, the performance of using 2 threads is improved 44.79 percent when compared to using 1 thread, and the performance is fasten up to 58.54 percent when using 8 threads. However, there are 2 issues of this implementation. The first issues is the forward propagation time. Theoretically, the process time of sequential computing and using single thread should be the same. In contrast, the forward propagation time of single thread was doubled, which causes total process time increase. The second issue is totoal process time. Although writting in C++ is able to achieve theoretical speed-up, the overall performance is slightly better when compared to Java.

4.2.4 NEON Instruction

OpenCV library provide an shared library, which is officially built by OpenCV. However, library is built to support all CPU chipsets, and many features and conditions was flaged during building. To evaluate the performance of NEON, library is needed to be manully built from the scratch. OpenCV shared library was manually built into 2 versions: version without NEON and version with NEON. To maximise the performance of NEON version, OpenCV is forced to built with NEON instruction regardless 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: Video Processing with MobileNet SSD Model without NEON

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

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 4.6 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 video up to 2.890 seconds, which is faster than OpenCV without NEON 72.79 percent and OpenCV official build 42.93 percent. For this performance, video can be processed 10 frames per second.

- Summarise the result of performance

4.3 Usability Testing

Moving on now to consider the usability of this application. Usability testing was divided into 2 parts. The first part is determining social distancing from pre-recorded files which is obtained from Ben Benfold and Ian Reid [3]. The second part is determining social distancing from camera.

Figure 1 shows detected people with social distancing determination.

- FPS on video

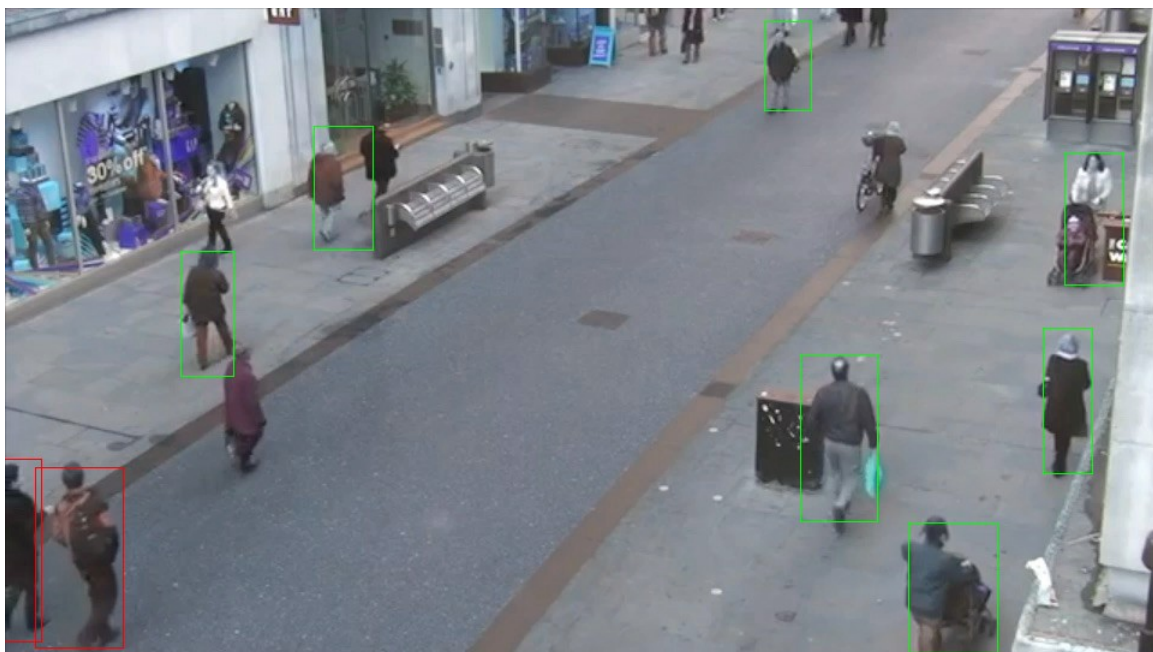


Figure 4.3: Social Distancing Detection from Picture

With multithread processing, camera is able to capture the frame around 25 frames per second.

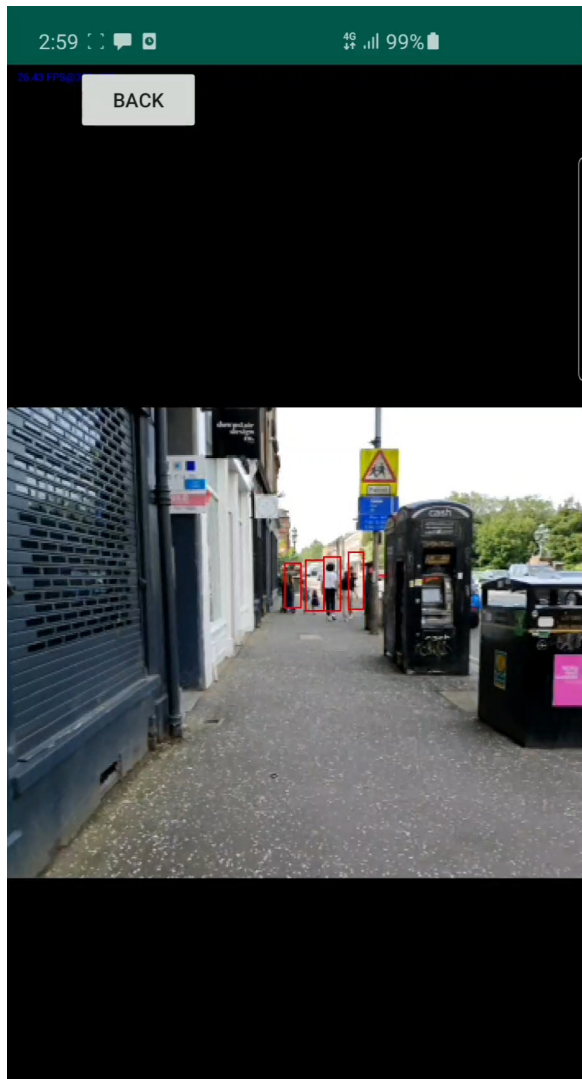


Figure 4.4: Social Distancing Detection from Camera

Chapter 5

Conclusion

5.1 Limitation

- What can be achieved

- Limitation / Problem - Limited Resource - CPU clock speed - RAM - Power resources

- CPU - ARM architecture limitation on floating-point [<http://tinyurl.com/y85ykaqa>] - When the number of threads is increasing, image processing task is not consistently processed by core. - Because the given task has to wait while core switching and doing another task (context switching) - Thus, the given task requires more time to be finished - There are stages of CPU's clock frequency - JNI - Calling JNI is expensive [ref]

- Multithread Performance Analysis - I/O in thread - If there is I/O operation in thread or loop, it will cost a lot of overhead - Out of memory - If let each thread hold the large variable, it will cost memory overhead. - We have to free the variables after used. Otherwise, the x+1 th thread will allocate another xx MB. - Young generation - If there are lot of variables that are initialised in loop, there will be a lot young generation in the heap. So, when the number of young generations is reaching the threshold, GC will correct the young generation (freeing garbage in young generation heap) which affect the performance. - GC - Caused by Native [<https://developer.android.com/studio/profile/memory-profiler>] - Bin is GC - 8 GB (available only 3.8 GB) - CPU hits 100- CPU drops when GC is started - Because threads are paused (Yellow) when GC is collecting

- Thread Pool Problem - Thread Pool only improve when there are large number of asynchronous tasks. [<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html>]

5.2 Future Work

Appendix A

First appendix

A.1 Section of first appendix

Appendix B

Second appendix

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.