Mobile Surveillance Siren Against Moving Object as a Support System for Blind People

David H. Hareva*, Sebastian A., Aditya R. Mitra, Irene A. Lazarusli, and Calandra A. Haryani

Universitas Pelita Harapan, Tangerang, Indonesia; Email: sa70015@alumni.uph.edu (S.A.), aditya.mitra@uph.edu (A.R.M.), irene.lazarusli@uph.edu (I.A.L.), calandra.haryani@uph.edu (C.A.H.)
*Correspondence: david.hareva@uph.edu (D.H.H.)

***Abstract*—Visually impaired people can use smartphone navigation applications to arrive at their destination. However, those applications do not provide the means to detect moving objects. This paper presents an Android application that uses the smartphone's camera to provide real-time object detection. Images captured by the camera are to be processed digitally. The model then predicts objects from the processed image using a Convolutional Neural Network (CNN) stored in mobile devices. The model returns bounding boxes for each of the detected objects. These bounding boxes are used to calculate the distance from the object to the camera. The model used is SSD MobileNet V1, which is pre-trained using the Common Objects in Context (COCO) dataset. System testing is divided into object distance and accuracy testing. Results show that the margin of error for calculating distance is below 5% for distances under 8 meters. The mean average precision is 0.9393, while the mean average recall is 0.4479. It means that the system can recognize moving objects through the embedded model in a smartphone.**

***Keywords*—Convolutional Neural Network (CNN), deep learning, object distance, object detection, support system for blind people**

I. Introduction

According to a report from the World Health Organization, there are about one billion people in this world who are visually impaired. Visual impairment can be caused by various things, such as untreated presbyopia (826 million), cataracts (65.2 million), glaucoma (6.9 million), and other causes [1]. However, just because people are visually impaired does not mean they cannot use smartphones to do daily activities. Those who are visually impaired can use smartphones via the screen reading feature. Apple mobile phones have voiceover, while Android phones have TalkBack [2]. Using smartphones, visually impaired people can use navigation applications such as Google Maps or Waze to reach a destination. Unfortunately, these applications are not able to recognize moving objects. Then one possible solution is to develop a system that can detect moving objects through their smartphone camera. Many mobile applications aim to help blind people, but they only provide directions to blind people via voice and GPS or require ultrasonic sensors to detect objects in front of the user. This system allows users to detect moving objects in front of them to avoid accidents, especially in crowded places. Upon receiving input from the smartphone camera, the system performs image computation. TensorFlow Lite will detect objects on the processed image [3]. As the output, the system will produce a series of frames containing detected moving objects in the form of a video stored in the phone device. When a moving object is close enough to the user, the system will generate an audible warning to the user.

This research aims to create a system that alerts visually impaired persons when an object moves in front of them. Moving object detection is performed soon after the system receives images from the smartphone's back camera. Image processing and object detection will be fully processed on the smartphone. The user will be warned through the text-to-speech feature when any object moves in front of them. In addition, the system will emit a siren sound to alert people around the user if there are moving objects that can hit the user with specific risks. It is hoped that this mobile application provides benefits for visually impaired people by guiding them to reach their destination safely.

II. Related Works

Various applications have been developed for blind people. A few example applications developed for assisting visually impaired people are Blind Square, Be My Eyes, and Aipoly Vision. Blind Square is a navigation application for the visually impaired with additional features such as describing the surrounding road conditions and announcing when the user is at a crossroads [www.blindsquare.com]. Be My Eyes is an application that connects visually impaired people with volunteer helpers via video calls to assist visually impaired people in navigation. [www.bemyeyes.com]. Aipoly Vision is an application that uses the camera to detect any object simply by pointing at the camera and pressing a button on the application [https://www.aipoly.com/].

Nevertheless, currently, no applications can provide real-time detection of objects and distance information through the camera. Aipoly Vision can only detect one

Manuscript received December 3, 2022; revised March 15, 2023; accepted May 1, 2023.

image and requires a continuous button press which may be inconvenient or practical for the user. Be My Eyes requires volunteers who are willing to help. Further, this application is not equipped with a feature that allows object detection [4]. To help in processing object detection by capturing images from a smartphone's camera is TensorFlow Lite. TensorFlow Lite is a TensorFlow application for lightweight use on mobile phones and embedded systems [5]. However, only SSD (Single Shot Multi Box) models are currently supported by TensorFlow Lite. Models such as Faster-RCNN are not supported yet, so the performance of SSD and Faster-RCNN cannot be compared [6]. Currently, there is no other way to detect the moving object solely by images from the camera. The developed system must quickly detect an object from video image pieces. Based on tests conducted by the TensorFlow team, MobileNet's speed for predicting a single frame is 30 milliseconds [7]. If one frame takes 30 milliseconds, MobileNet will perform a maximum of 33 detections in one second. Of course, the actual system performance depends on the mobile hardware used. However, this is one way that can be used to process object detection quickly.

III. METHODOLOGY

The system that will be implemented is an Android application called "Blindness Path Guidance". Through Android, app developers can access the phone's hardware, such as the camera, WiFi connection, cellular network, and data on the phone. Every application developed for Android needs access permissions in three threat levels [8] that are declared in the app's manifest file [9]. First, they are standard permission that does not harm the user; for example, is asked to change the time zone. Signature permission regulates access to the most dangerous permissions, such as deleting other applications. This permission is only granted to pre-installed apps by the phone manufacturer [8]. Last is Malicious permission, which can access users' personal information such as reading the contact list or accessing the camera. The user must grant malicious permissions during runtime to operate [9].

Human visual senses can be modeled using computer vision to automate the recognition of objects that are seen around. For example, text-to-speech, automated car parking, accessing information by reading QR codes or moving object detection [10]. Most computer visions use something called Convolutional Neural Network (CNN). It is a neural network often used for high-dimensional data, such as images and videos. Neocognitron is a sample of the CNN model proposed by Kunihiko Fukushima in 1982. It is inspired by the work of Hubel and Wiesel based on how nerves work in the brain and consist of many layers [11]. A model of CNN used machine learning to develop algorithms by studying specific data patterns. The machine can make predictions from the patterns learned by assuming that the existing data will not differ much from the data at the time of collection [12]. Deep learning is a subset of machine learning that learns data through different representations. Deep learning using images can be applied to recognizing objects or recognizing certain features, such as recognizing humans, bicycles, motorbikes, or others. Typically, deep learning makes use of artificial neural networks [13].

Improving accuracy in recognizing objects in the selected image needs image processing. This can be in the form of noise filtering, converting color images to monochromatic, changing image size, location of objects in the image, and much more [14, 15].

MobileNet is a type of CNN model designed by a team of researchers from Google to apply computer vision on mobile phones and embedded systems. The purpose is for real-world tasks that require efficient models with limited computational resources, such as building robots, augmented reality, and autonomous cars. From the results of tests conducted by Andrew G. Howard and his team of researchers, it can be seen in Table I that MobileNet has similar accuracy to VGG and Inception V2 but with far fewer parameters [16].

There are several methods to get distance from an object to the camera, but the best method to obtain the distance using a single image is by using perspective. The information needed to calculate the distance from the object to the camera is the lens's focus, the sensor's height, the height of the object in the image, the height of the image, and the actual height of the object. The formula for calculating the distance can be seen in Eq. (1). The unit of the object's height will determine the object's distance from the camera [17].

TABLE I. COMPARISON OF MOBILENET TO VGG AND INCEPTION V2

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	MobileNet	19.8%	30.5	6.1

$$D = \frac{h_{real}\ x\ f\ x\ h_{image}}{h_{sensor}\ x\ h_{object}} \tag{1}$$

where, h_{real}: actual object height, f: focal length (mm), h_{image}: image height (*pixel*), h_{sensor}: sensor height (mm), and h_{object}: object height in picture (*pixel*).

However, getting the actual height of the detected object is difficult, so the value of the actual height of the object is the average height of the detected object. For example, the average human height in Asia is 1.58 meters [18], so that number will replace the actual object height.

Precision is the chance that the system predicts the correct prediction, while Recall is the chance the system can predict an object. The formula for Precision can be seen in Eq. (2), while the formula for Recall can be seen in Eq. (3).

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

where, *TP*: True Positive, *FP*: False Positive, and *FN*: False Negative

Precision and Recall will result in a number between zero and one. Ideally, Precision and Recall should be high. If the Precision is high, but Recall is low, many predicted objects are correct, but many objects are not detected. If the Recall is high, but the Precision is low, it means many objects were predicted correctly, but there are many wrong predictions. After calculating the Precision and Recall for each type of object detected, the Precision and Recall values can be added and divided by the number of object types to obtain the mean average Precision and mean average recall [19].

IV. System Design

The application begins by loading the model embedded on a mobile phone. After the model is successfully loaded, the mobile phone camera records a video, then the application will capture image frames from the video, and the image processing stage begins. This stage produces images with a size of 300 × 300 pixels. The results are then used as input for the object recognition stage. The object recognition stage is carried out using a model loaded in the system's early stages. The model will produce output in the form of a bounding box that stores information on the location and type of objects. The system then calculates the object's distance from the camera using the perspective of a single image in Eq. (1). If any objects are detected, the system will then give a warning to the surrounding by sirens and information to the user.

A. Model Training

Before creating the system, the model that will be used to detect objects needs to be trained using the data that has been prepared. The base model that is going to be used for the system is ssd_mobilenet_v1_coco. After the computer has finished installing TensorFlow for training, the next step is to label the training data. The *labelImg* program will be used to label images. *LabelImg* is a tool that make it very easy to annotate image (https://pypi.org/project/labelImg). After labeling all training data, *labelImg* will generate an xml file per image. Then, the data for training and testing will be separated. Then, the training and testing data need to be converted into a record file. The conversion is done using a script provided by TensorFlow. After the conversion is complete, model training can be started.

The number of training epochs can be calculated by the batch size multiplied by the number of steps divided by the number of images used for training. The batch size used is two, the number of images is 300, and the number of steps is 18,000. If counted, this means the model trained for 120 epochs.

B. Testing

Testing will be done through distance testing and accuracy testing. Distance testing will be done by placing the smartphone at a height of one meter above the ground, then someone will walk towards the cell phone. A measuring tape will be used to measure the original distance. Distance measurement starts at 8 meters and ends at two meters. To test the performance of the system for measuring distance, error percentage will be used. The smaller the error percentage, the more accurate the system is in calculating distance. The error is calculated by taking the difference between the calculated distance and the original distance and dividing it by the original distance. Multiply the number by 100 to get the error as a percentage. The accuracy test will be done by taking a recording of the system using a smartphone while walking in a crowd, for two minutes. The recording will be checked every second for true positive, false positive, false negative.

- True Positive (TP) is when an object that was detected correctly by the system.
- False Positive (FP) is when the system to detect an object, even though there are no objects in the detected area.
- False Negative (FN) is the system does not detect an object, even though there should be an object detected.

The sum of the TP, FP, and FN collected within the two-minute period will be used in calculating system performance. The units used to evaluate system performance are Precision and Recall. The method of calculating precision and recall can be seen at Eq. (2) and (3).

V. Implementation and Testing

The system will be created for Android using the Kotlin programming language. The system is divided into several stages, namely accessing the camera, Android image processing, object recognition, calculating object distances, sirens, and text-to-speech as illustrated on Fig. 1.

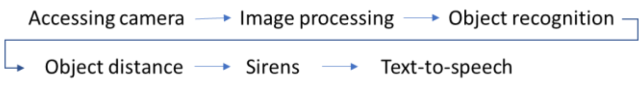

Figure 1. Model of blindness path guidance.

A. Accessing the Camera

The system requires camera access to capture images. To gain camera access, the system will first ask the user for permission to use the camera. Helper code for managing camera permissions is divided into four functions. Each function has its own role that can be called on the main logic of the system, namely:

- `hasCameraPermission`, whose role is to check whether the permission for camera access has been granted or not. Returns true if given and returns false if not already provided.
- `requestCameraPermission`, whose role is to request camera access permission to the user. This function is called when the user has never or refused to provide camera access to the system. The system can request access permission for the camera by writing Manifest.permission.CAMERA in manifest.xml.
- `shouldShowRequestPermissionRationale`, whose role is to check whether a permission has been granted or not. This function returns true if

the system never asked for permission or asked for permission but was denied, and false when the user refuses with the option not to ask again. This function is needed to check when the user presses the button not to ask again, so that the system can show an alternative screen if needed. `launchPermissionSettings`, which moves the active screen to the phone settings screen. This function is required when the user presses the button to deny and never ask again when requesting camera access. The system moves the user to the settings screen so that the user can grant access manually.

The application interface when requesting camera access to the user can be seen in Fig. 2.

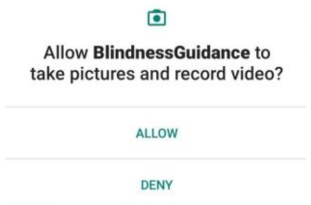
Allow **BlindnessGuidance** to take pictures and record video?

Figure 2. Application interface when asking for permission.

B. *Android Image Processing*

The code to receive an image from a mobile phone camera is a callback that is called by using `setOnImageAvailable Listener`. When the image is ready, the code inside the callback will be executed. By calling `acquireLatestImage`, the latest image will be received by the system. The received image is an object with the class `android.media.Image`, with YUV color formatting.

The input from TensorFlow Lite only accepts information in RGB color format, so every pixel in the received image needs to be converted to RGB. The conversion process to RGB is carried out by separating the YUV information into three parts, namely Y (brightness), U (blue projection), and V (red projection). For each pixel in the received image, the Y, U, and V information are separated, then converted to RGB color format via the `YUV2RGB` function.

Android does not provide a function for converting from YUV to RGB. An additional function is required to convert color format from YUV to RGB. The `YUV2RGB` function accepts the Y, U, and V values of a single pixel and then returns the pixel values in RGB color format. In general, conversion from YUV to RGB requires multiplying by non-integer numbers. However, the input that TensorFlow Lite wants is an 8-bit integer. To solve this, the YUV2RGB function multiplies by a large number, then clamps the pixel value between 0 and 128.

TensorFlow Lite accepts an 8-bit number for each R, G, and B value of the pixels. In hexadecimal form, the RGB color format has the highest value of 0xff or 255 for each color. The red, green, and blue values can then be mixed to get the desired color. For example, the representation of yellow (red mixed with green) is 0Xffff00. The first ff value is the maximum red color, the second ff is the maximum green color, and the 00 value at the end is no blue mixture at all. Through the representation in hexadecimal, R, G, and B information can be mixed and separated easily.

C. *Object Detection*

After processing the image digitally, the application needs to extract the pixels from the image as input. To simplify the application, object detection is performed on a single function that accepts a 300 × 300 pixels bitmap and returns a list of objects in the form of a `Recognition` class. `Recognition` is a self-made class that stores three variables, namely `title`, `confidence`, and `location` of the object.

- `Title` stores the detected object's name (human, motorcycle, or bicycle).
- `Confidence` stores the percentage of confidence that an object is true, with 100% being very sure and 0% being unsure.
- `Location` stores the location of the detected object in rectangular coordinates.

The contents of the function can be broken down into three important parts, namely:

1. The process of extracting pixel values from the bitmap by calling the built-in function `getPixels`. `getPixels` is a function that fills the array in the first argument. In this case, `getPixels` will fill the array used as input for TensorFlow Lite. The pixel information is separated into red, green, and blue via conversion to hexadecimal. The result will be a byte buffer with size 300 × 300 × 3.
2. The process of preparing the input and output, which are included with object detection by calling the `runForMultipleInputsOutputs` function. The function accepts one input array and four output arrays. The input array contains the images that will be detected, but in this case the input array only contains the most recent image. The four output arrays consist of location, object type, confidence percentage, and number of detected objects.
3. The process of converting the results of object detection into the `Recognition` class so that it is more easily recognized by the system. The model used has limitations in recognizing a maximum of 10 objects in one image. However, the number of detected objects is not always certain. If less than 10 objects are detected, the contents of the array for the remaining indexes will be empty. The next step will be converting the detected objects into a `Recognition` class and returning a list with no empty objects. This makes it easier for the application because it does not have to check whether an element in the original output array has a value or not. By using the `Recognition` class, calling the object location or object type can be done by accessing the member of the object's class.

D. Calculating Object Distance

First, for each detected object, it is necessary to see the percentage of confidence to prevent wrong detection in the image. Objects with a low percentage of confidence are ignored. The minimum recommended confidence to reduce false positives is 60% [1]. For each detected object, height information in pixels is required to calculate the object's distance to the camera. Other information is also required, such as the image size, the focal length of the camera, the size of the camera sensor, and information regarding the average height of the detected object.

- The size of the captured image can be obtained simply
- The camera sensor size can be obtained by retrieving the `SENSOR_INFO_PHYSICAL_SIZE` info from the `CameraCharacteristics` class.
- The object's average height information is hard code declared at the start of the system. The average height value per object is obtained by searching publicly available information and can be replaced when there is newer information.

When all the information has been collected, the object's distance from the camera can be calculated using Eq. (1). Apart from calculating the distance, the system also checks whether there are objects that are close to the user. The minimum distance can be changed to adjust it to be further or closer.

E. Siren and Text-to-Speech

For each object detected by TensorFlow, the object name is translated into Indonesian, then the distance information is stored. After saving the object's location information, the system will shorten the spoken text by saying the object type at the beginning, then combine all the detected distances for that object type. An example is the system would shorten "Person 5 meters Person 7 meters" to "Person 5 meters 7 meters". Siren and Text-to-Speech cannot be played at the same time. The system needs to decide whether to sound the siren or turn on Text-to-Speech. Based on the closest distance that has been determined when calculating the distance of all objects, the system will determine the source of sound output, namely from connected headphones or from the speaker of a cellphone. To control the sound output, an object with access to Android's audio system is needed. The `audioManager` variable is an object with the class `android.media.AudioManager`, which controls the sound source. To turn on the speaker on a mobile phone, the value of `isSpeakerphoneOn` must be `true`, and the mode must be `MODE_IN_COMMUNICATION`.

F. Results of System Testing

System testing is divided into distance testing and accuracy testing. Distance testing is done by using a measuring tape to get the real distance. The distance calculated by the system will be compared to the real distance.

The system detects different results for the same distance. This is caused by the fact that uses pixel information for the object height. If the detected object height differs by a few pixels, the resulting calculation will be different. Various conditions affect the object detection process, such as lighting, focus, or is the object being covered by other objects. At distances closer than 3 meters, the camera most likely picks up the whole object, resulting in distances close to the average height information hard coded into the system. Table II shows the maximum error for each detected distance by doing 30 experiments.

Accuracy testing is separated into three classes, which are people, motorcycle, and bicycle. Testing for each class is done by analyzing a 2-minute recording. For detecting people in Table III(a), the criteria are as follows:

- True positive (TP) is when people are detected correctly by the system.
- False positive (FP) is the system detects the presence of people, even though there are no people in the detected area, or when the system detects the object type incorrectly. For example, the system detects a person, even though the real object is a motorcycle.

False negative (FN) is the system does not detect an object, even though there should be an object detected. For example, the system does not detect people, even though there are people on the recording.

TABLE II. RESULTS OF DISTANCE TESTING

	Distance (m)						
	8	7	6	5	4	3	2
1	8.26	6.73	5.97	4.86	3.81	3.12	1.91
2	7.82	7.11	5.89	5.11	3.90	3.02	1.88
3	8.15	7.01	6.04	4.92	4.00	3.12	1.94
4	7.93	6.92	5.87	4.80	4.19	2.95	1.87
5	8.04	7.02	5.82	5.18	3.95	3.18	2.04
6	7.93	7.01	6.21	4.86	3.84	3.15	1.92
7	8.01	6.92	6.17	4.92	3.92	3.02	1.90
8	8.11	7.10	6.13	4.98	3.90	3.07	2.10
9	8.07	6.93	6.12	5.1	3.91	3.02	1.94
10	8.09	7.03	5.90	5.07	4.04	3.02	2.05
11	8.13	7.14	5.99	5.08	4.14	2.96	2.00
12	7.96	6.98	6.03	5.00	4.02	2.97	2.05
13	8.00	6.91	6.04	4.95	3.92	3.14	2.13
14	8.09	7.03	6.14	4.97	3.97	3.01	2.10
15	8.08	6.97	6.12	4.96	4.00	2.92	1.96
16	7.93	7.14	6.06	5.02	4.09	3.02	1.91
17	8.00	7.02	6.1	5.09	4.01	3.11	1.97
18	7.92	7.08	6.13	4.95	4.02	3.02	2.03
19	8.02	6.94	5.91	4.99	3.94	3.07	2.07
20	8.07	6.93	5.94	5.08	4.13	3.05	2.01
21	8.03	6.93	5.99	5.07	4.12	3.12	2.12
22	8.14	7.05	6.08	5.08	4.13	3.04	1.90
23	8.14	6.99	6.05	4.95	3.96	3.12	2.11
24	7.91	7.11	5.98	5.05	4.10	2.99	2.10
25	8.11	7.05	6.10	5.03	3.98	2.94	1.92
26	8.00	6.96	5.91	4.94	3.94	3.11	2.08
27	8.08	6.93	5.96	5.10	4.01	2.98	2.12
28	8.03	7.12	6.10	5.07	4.02	3.07	2.14
29	7.92	7.11	6.08	4.97	4.02	2.97	2.01
30	7.94	6.94	6.10	4.91	4.00	2.92	2.02
Maximum error	3.25%	3.86%	3.50%	4%	4.75%	6%	7%

Based on the results, the average precision for people is 0.9452, while the average recall for people is 0.5840. The number of false negatives during testing is high since the system cannot recognize objects at high distances, while a normal human eye still can recognize the object. An example of the system being unable to recognize objects at great distances can be seen at Fig. 3.

The number of false positives is low, but not zero. There are times when the system recognized a motorcycle as a person. An example of this happening can be seen at Fig. 4.

Testing for motorcycle in Table III(b) follows these criteria:

- True positive (TP) is when the system identifies a motorcycle correctly.
- False positive (FP) is when the system detects a motorcycle when it is actually not there, or another object (people or bicycle) is detected as a motorcycle.
- False negative (FN) is when the system does not detect a motorcycle, even though there is a motorcycle.

Based on the results, the average precision for motorcycle is 0.9760, while the average recall for motorcycle is 0.4150. A high recall and low precision mean that the system can identify motorcycles correctly but missed other motorcycles. The motorcycles that are not detected are usually because they are too far, so the system does not have enough information to decide if there is a motorcycle or not. There are also times when a person is riding the motorcycle, the person is detected, but the motorcycle is not detected. An example of this happening can be seen at Fig. 5.

Testing for bicycle in Table III(c) follows these criteria:

- True positive (TP) is when the bicycle is detected correctly by the system.

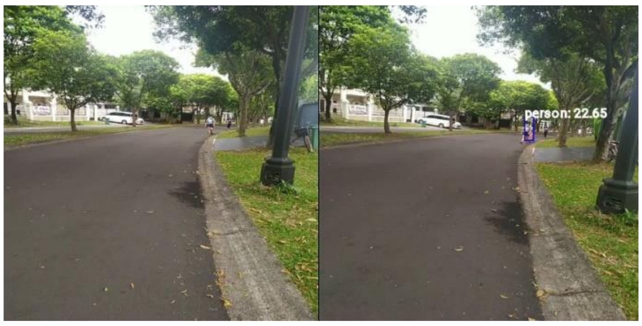

Figure 3. Screenshot of system during testing for people.

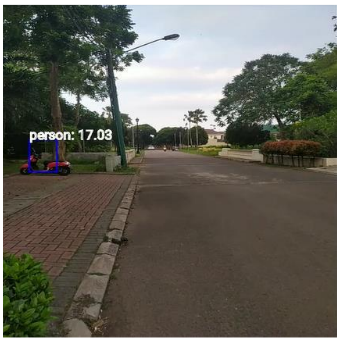

Figure 4. False positive during testing for people.

TABLE III. RESULTS OF ACCURACY TESTING

No	(a)People					(b)Motorcycle					(c)Bicycle				
	TP	**FP**	**FN**	**Precision**	**Recall**	**TP**	**FP**	**FN**	**Precision**	**Recall**	**TP**	**FP**	**FN**	**Precision**	**Recall**
1	19	0	14	1.000	0.576	8	0	14	1.000	0.364	2	0	7	1.000	0.222
2	33	2	16	0.943	0.673	21	0	20	1.000	0.512	4	0	6	1.000	0.400
3	18	0	27	1.000	0.400	13	0	16	1.000	0.448	3	0	8	1.000	0.273
4	43	1	35	0.977	0.551	7	0	3	1.000	0.700	3	0	6	1.000	0.333
5	44	6	31	0.880	0.587	5	0	13	1.000	0.278	1	0	3	1.000	0.250
6	15	0	22	1.000	0.405	17	0	22	1.000	0.436	4	1	3	0.800	0.571
7	11	3	20	0.786	0.335	7	0	15	1.000	0.318	2	0	4	1.000	0.333
8	26	0	30	1.000	0.464	5	2	10	0.714	0.333	4	0	9	1.000	0.308
9	20	0	27	1.000	0.426	12	0	19	1.000	0.387	5	1	8	0.833	0.385
10	14	0	12	1.000	0.538	8	0	12	1.000	0.400	5	1	7	0.833	0.417
11	24	1	11	0.960	0.686	11	0	21	1.000	0.344	5	1	7	0.833	0.417
12	18	3	10	0.857	0.643	6	0	17	1.000	0.261	3	0	4	1.000	0.429
13	28	0	11	1.000	0.718	12	0	14	1.000	0.462	2	1	8	0.667	0.200
14	23	0	11	1.000	0.676	6	0	9	1.000	0.400	3	0	9	1.000	0.250
15	16	3	10	0.842	0.615	9	0	13	1.000	0.409	2	0	7	1.000	0.222
16	25	0	14	1.000	0.641	5	0	5	1.000	0.500	2	1	5	0.667	0.286
17	18	1	11	0.947	0.621	6	0	11	1.000	0.353	3	0	7	1.000	0.300
18	45	0	31	1.000	0.592	7	0	20	1.000	0.259	4	0	4	1.000	0.500
19	21	0	28	1.000	0.429	10	0	13	1.000	0.435	5	1	9	0.667	0.357
20	19	5	9	0.792	0.679	12	1	14	0.923	0.462	2	1	4	1.000	0.333
21	36	3	21	0.923	0.632	13	0	13	1.000	0.500	2	0	5	1.000	0.286
22	23	1	13	0.958	0.639	10	0	12	1.000	0.455	3	0	6	0.833	0.333
23	24	3	10	0.889	0.706	11	2	11	0.846	0.500	4	1	6	0.667	0.400
24	20	0	19	1.000	0.513	9	0	16	1.000	0.360	4	1	6	1.000	0.400
25	27	1	16	0.964	0.628	14	0	15	1.000	0.483	4	0	5	1.000	0.444
26	28	2	9	0.933	0.757	10	1	14	0.909	0.417	2	1	6	0.667	0.250
27	22	2	19	0.917	0.537	10	0	11	1.000	0.476	2	1	5	0.667	0.286
28	17	1	14	0.944	0.548	10	0	11	1.000	0.476	5	1	5	0.883	0.500
29	24	3	17	0.889	0.585	9	0	15	1.000	0.375	5	0	9	1.000	0.357
30	21	1	9	0.955	0.700	8	1	15	0.889	0.348	3	0	7	1.000	0.300
	Average			**0.945**	**0.584**	**Average**			**0.976**	**0.415**	**Average**			**0.897**	**0.345**

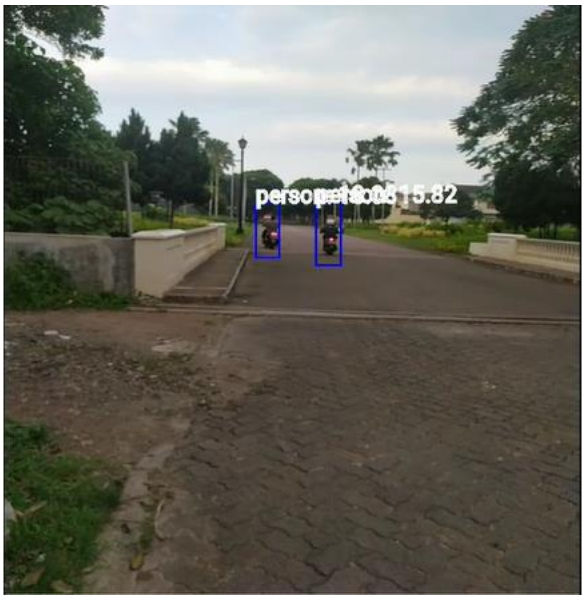

Figure 5. System detects person but not motorcycle

- False positive (FP) is when the system detects a bicycle when it should not be there, or another object (human or motorcycle) is detected as a bicycle.
- False negative (FN) is when the system does not detect a bicycle, even though it should be.

Based on the results, the average precision for bicycle is 0.8967, while the average recall for bicycle is 0.3447. Just like during motorcycle testing, people are more likely to be detected while riding a bicycle. The low number of bicycles during testing could also be the reason for a lower precision compared to the other classes. The results for each class and the mean average can be seen at Table IV.

The results show a mean average precision of 0.9393, and a mean average recall of 0.4479. This means that the system can detect most objects correctly but is unable to detect very far objects compared to a normal human eye.

TABLE IV. RESULTS OF OVERALL ACCURACY TESTING

Object type	Average precision	Average recall
Person	0.9452	0.584
Motorcycle	0.976	0.415
Bicycle	0.8967	0.3447
Mean Average	0.9393	0.4479

VI. CONCLUSION AND FUTURE WORKS

Based on the research that has been carried out, it can be concluded that the system has succeeded in recognizing moving objects through a smartphone camera by involving digital image processing and using a trained MobileNet v1 model with the COCO dataset. Digital distance calculation successfully recognizes distances with an error percentage of under 5% for distances of four meters and above. The average precision value reaches 0.9393, while the average recall value reaches 0.4479. Siren and Text-to-Speech also work as intended.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

David H. Hareva conducted a research methodology to recognize multiple moving objects, Sebastian Aldi and Aditya R. Mitra designed and developed a mobile application support system for the visually impaired, Irene A. Lazarusli and Calandra A. Haryani analyzed the application and its output data; and all authors have approved the final version.

FUNDING

Reporting research in this publication supported by Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi (Ministry of Education, Culture, Research, and Technology) of Indonesia under award number 069/E5/PG.02.00.PT/2022, 466/LL3/AK.04/2022, and Universitas Pelita Harapan, Indonesia under award number 121/LPPM-UPH/VI/2022.

REFERENCES

[1] R. R. A. Bourne, *et al.*, "Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: A systematic review and meta-analysis," *Lancet Glob. Heal.*, vol. 5, no. 9, pp. e888–e897, Sep. 2017.
[2] C. Praderio. (2017). Here's how blind people use smartphones. *INSIDER.* [Online]. Available: https://www.insider.com/how-blind-people-use-smartphones-2017-2
[3] M. Abadi, *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, November 2–4, 2016, Savannah, GA, USA, pp. 265–283.
[4] Everyday Sight. (2020). 25 best apps for the visually impaired-everyday sight. [Online]. Available: https://www.everydaysight.com/best-apps-for-visually-impaired/
[5] O. Alsing, "Mobile object detection using TensorFlow lite and transfer learning," Dissertation, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 2018.
[6] TensorFlow. (2020). Running on mobile with TensorFlow lite. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tensorflowlite.md
[7] R. Wagner, M. Thom, R. Schweiger, G. Palm, and A. Rothermel, "Learning convolutional neural networks from few samples," in *Proc. the 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, doi: 10.1109/IJCNN.2013.6706969
[8] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. of the ACM Conference on Computer and Communications Security*, 2011, pp. 627–636.
[9] Android Developers. (2018). Permissions overview | Android Developers. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview#normal-dangerous
[10] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Realtime computer vision with OpenCV," *Queue*, vol. 10, no. 4, p. 40, 2012.
[11] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," vol. 8, no. 1, *Morgan & Claypool Publishers LLC*, 2018.
[12] E. Alpadyn, *Introduction to Machine Learning Second Edition Adaptive Computation and Machine Learning*, 2nd ed. Cambridge: The MIT Press, 2010.
[13] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3–4, p. 202, 2014.
[14] University of Tartu. (2018). Introduction to image processing. [Online]. Available: https://sisu.ut.ee/imageprocessing/book/

[15] R. C. Gonzales and Z. Faisal, *Digital Image Processing*, 2nd Ed. 2019.

[16] A. G. Howard, *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861v1, 2017.

[17] W. Fulton. (2015). Calculate distance or size of an object in a photo image. [Online]. Available: https://www.scantips.com/lights/subjectdistance.html

[18] O. Smith, "Mapped: The world's tallest (and shortest) countries," *Telegraph*, 2019.

[19] M. El Aidouni. (2019). Evaluating object detection models: Guide to performance metrics. [Online]. Available: https://manalelaidouni.github.io/manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html