REAL-TIME MONEY COUNTING APP FOR VISUALLY IMPAIRED

BY

LIM BOON CHONG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEB 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ts. Dr. Saw Seow Hui who has given me this bright opportunity to engage in a Real-time money counting app for visually impaired project. It is my first step to develop an assistive technology for the visually impaired community. A million thanks to you, you guided me on the way to complete this project.

I would also like to extend my sincere thanks to my friends and family for their positive feedback. Your encouragement, love, and belief in me have been a source of strength, especially during challenging times when I felt like giving up. Your kindness and always standing by my side have been invaluable, without your support, this project would not have been possible. Once again, thank you for always being there for me.

ABSTRACT

Visually impaired individuals usually need assistance from others to perform daily activities such as grocery shopping, reading documents, and recognizing banknotes due to their limited vision. Low vision or blindness causes their daily life activities to become time-consuming, especially dealing with financial transactions because they are not able to determine the amount of money that they are handling especially if the banknote is old, worn, or even torn. Even though efforts are made to print Braille on the banknotes, the tactile loses its ability quickly after circulation. Moreover, the existing mobile application specifically designed to help the visually impaired is usually unable to recognize Malaysian currency, especially coins. Additionally, most mobile applications require user subscriptions to enable full functionality, which limits the visually impaired users' ability to use them in daily life. Therefore, this project aims to develop a mobile application using Flutter for both Android and iOS platforms, with the transfer learning on a pre-trained YOLOv8 model to recognize and count new Malaysian banknotes and coins one by one in real-time. The app also utilized TensorFlow Lite to convert the model into a mobile-compatible format to run on mobile devices. Furthermore, the app designed with a user-friendly interface and accessibility features such as high-contrast text, audio feedback, and vibration notification. The mobile application can help visually impaired users recognize and count the banknotes and coins they are handling in a more accessible with mAP50 (mean average precision calculated at an intersection over union threshold of 50) up to 98.7%. A custom counting technique is also implemented, utilizing Euclidean distance calculations and a timeout mechanism for accurate object tracking.

Area of Study: Artificial Intelligence, Mobile App Development

Keywords: Transfer Learning, YOLOv8, Real-time detection, Malaysian currency, Visually Impaired

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

LIST OF ABBREVIATIONS

WHO	World Health Organization
MAHAL	Malaysian Assistive Hub for Advanced Livelihood
MAB	Malaysian Association of the Blind
TTS	Text-To-Speech
STT	Speech-To-Text
IDE	Integrated Development Environment
UI	User Interface
CNN	Convolutional Neural Network
RM	Ringgit Malaysia
YOLOv8	You Only Look Once v8
SiLU	Sigmoid Linear Unit
SPPF	Spartial Pyramid Pooling Fast
SDK	Software Development Kit
HEIC	High Efficiency Image Container
YAML	Yet Another Markup Language
IOU	Intersection Over Union
KNN	K Nearest Neighbor

Chapter 1

In this chapter, we present the background, introduction, problem statement, contribution and report organization to the real-time money counting app for visually impaired.

Introduction

Visual Impairment is a term used to describe a condition where an individual has lost the ability to see clearly or is totally blind. Various factors can cause visual impairment such as genetic disorders, age-related macular degeneration, eye diseases, injuries, or conditions like diabetes. World Health Organization (WHO) stated that there are at least 2.2 billion people worldwide suffering from some form of visual impairment, and in at least 1 billion of these cases, the impairment could have been avoided or has not yet been treated [1]. Visual impairment can come in many forms such as cataracts, blurred vision, or complete blindness, each of those forms has its own set of challenges. The impact of visual impairment can cause daily activities such as reading, navigating their environment, or handling money to become a difficult challenge for visually impaired individuals.

One of the critical challenges faced by visually impaired individuals is the ability to identify and differentiate Malaysian coins and banknotes. For a person with a normal vision, money transaction is a straightforward process of taking cash out of the wallet to make payment. However, the situation becomes very different and more complex when it comes to visually impaired individuals, because they are not able to determine the amount of money that they are handling, especially if the banknote is old, worn, or torn, making it even harder to identify by only touching. As a result, visually impaired individuals usually rely on others to inform them of the value of money they are using. This dependence raised a concern which the increased risk of being scammed, as unethical individuals may provide false information about the amount of money. There are efforts that have been made globally to address this issue, which is printing Braille on currency to aid in tactile identification. However, it ages very fast, after a very short period in circulation, the money marked with Braille loses its ability to be used in its tactile form [2].

The appearance of technology has created an opportunity to overcome some of these challenges. With the rise of smartphone technology in image recognition, it is now possible to develop applications that can help visually impaired individuals in identifying and counting money accurately. These applications can use smartphone cameras to get images of coins and banknotes then use algorithms to identify their value. After processed, the information will be passed to the user with auditory feedback or vibrations to let them know the amount of money.

This project aims to build a mobile application that is for both Android and iOS platforms by using Flutter to process the Malaysia banknotes recognition and counting in real-time. The app will be using computer vision techniques with the use of library from Python such as TensorFlow Lite to recognize banknotes and coins. In terms of accessibility, the recognized value of the currency will be informed to the user through audio feedback and haptic feedback to deliver a response as soon as possible without the user need to look at the screen.

1.1 Problem Statement and Motivation

Difficulty in recognizing banknotes and coins

Recognizing and differentiating banknotes and coins is a challenge for visually impaired individuals. The tactile between coins and banknotes are not so easy to feel which makes it difficult to identify just only touching. This problem is further complex when the banknote is old, worn, or partially torn, which can blank out the key features used for identification. As a result, visually impaired individuals often need to rely on the assistance of other people to make sure the amount of money is sufficient for transactions. This will lead to a loss of independence and an increased risk of being scammed by unethical individuals. The inability to recognize and count money independently is a problem that limits everyday activities for those visually impaired individuals.

Limited Support for Multilingual and Multicultural Contexts

Although there are already similar applications to read banknotes on both Android and iOS platforms, most of them do not provide support for diverse languages

or regional contexts. Most of these applications support only one or a few major languages, leaving non-native speakers or people from multilingual societies in a disadvantageous position. For example, most visually impaired persons in Malaysia speak Malay, Mandarin, or English as their first language, but current applications lack support many of these languages. Moreover, the cultural localization of existing apps is not good enough to pronounce currency values in the local dialect. This linguistic and cultural gap limits the usability and accessibility of the solutions for visually impaired people in multilingual and multicultural countries like Malaysia.

Limited accessibility and functionality of existing solutions

Current solutions for visually impaired individuals often fail in terms of accessibility and functionality. Many apps are not optimized for local currencies such as the Malaysian ringgit, which makes the currency accepted inaccurate. In addition, existing solutions may lack features that visually impaired users need, such as voice feedback or customizable interface design, making them difficult to use. This lack of flexibility and individuality limits the effectiveness of these tools. The lack of solutions hinders the ability of individuals with visual impairments to participate fully in economic activities, further marginalizing them in society.

1.2 Objectives

i. **To develop a mobile application that helps visually impaired individuals to recognize and count Malaysian banknotes and coins**

In this project, the first objective is to create a mobile application to help visually impaired individuals recognize and count Malaysian banknotes and coins accurately. This objective focuses on the development of a real-time recognition system that is both accessible and user-friendly to help visually impaired users in handling currency.

ii. **To provide a fully localized and multilingual application**

This objective aims to ensure that the application supports all the common languages used in Malaysia, including Malay, Mandarin, and English. The app should be done with localized voice feedback, and cultural context, in ensuring that this application truly caters to diverse audiences. With

multilingual support, the app can be more inclusive and address linguistic diversity among the visually impaired in Malaysia.

iii. **To provide visually impaired users with a user-friendly interface with accessibility features**

The goal is to create an easy-to-use interface with some accessibility features such as voice over, flashlight, audio feedback, money marker, voice command, and tutorial, to make sure the overall experience of using the application.

1.3 Project Scope and Direction

The purpose of this project is to develop a mobile application using Flutter for visually impaired individuals to help them in reading and counting new Malaysian currencies, specifically the fourth series of Malaysian banknotes and coins, accurately by detecting one by one in real time. The term "new" refers to the current official currency series issued by Bank Negara Malaysia, which includes updated design, colors, and security features. These updates distinguish them from older currency series, which are either no longer in active circulation or gradually being phased out. The app will utilize YOLOv8 for real-time banknotes and coins identification, also providing real-time feedback using audio feedback in different languages such as Malay, Mandarin, and English. Besides, the app will have a user-friendly user interface to make it accessible to visually impaired users including options such as automatically flashlight to improve visibility, clear audio feedback for interaction, tutorial to guide the user how to use the app, customizable user preferences, and ability to mark scanned money to prevent counting errors. Furthermore, the app will also integrate Flutter TTS (Text-To-Speech) to convert text information into audio feedback for the user and voice command for hands-free operation. The project aims to help visually impaired individuals by developing a free, accessible, and reliable tool for visually impaired users' daily currency handling.

1.4 Contributions

Figure 1.1 MAHAL Logo

This project in the end will contribute to the Malaysian Assistive Hub for Advanced Livelihood (MAHAL) by providing an innovative solution for the Malaysian Association of the Blind (MAB) and the visually impaired community on daily currency recognition and counting. By developing the Real-Time Money Counting App, the independence and quality of life for visually impaired individuals can be enhanced using advanced image recognition and auditory feedback technologies. The app will allow users with vision impairments to identify and count Malaysian coins and banknotes independently, thereby reducing dependence on others for such simple tasks and lessening the possibility of financial exploitation. The mobile application will be accessible to users with a more user-friendly user interface and features that will help visually impaired users solve the challenges they face.

In addition, this project will be a reference for future researchers and developers who are interested in creating assistive technology for the visually impaired community.

1.5 Report Organization

The report of this project is organized into a total of 7 chapters. Chapter 2 mainly reviews existing systems and previous works for currency detection and counting, highlighting their features, strengths, and weaknesses, and comparing them with the proposed system. Chapter 3 focuses on the methodology, hardware, tools to use and the project timeline. Chapter 4 outlines the system design and architecture along with diagrams such as system flowchart, use case diagram, and activity diagram to illustrate the mobile application's operation. Chapter 5 focuses on system implementation which includes setup for hardware and software, system operations, implementation issues and challenges and the overall operations of the system. Followed by Chapter 6 which evaluates the overall system performance, testing setup, result for the test cases, and objectives evaluation. Lastly is Chapter 7 which concludes the report by summarizing the project by discussing the challenges faced during development and suggesting potential improvements for future work.

Chapter 2

Literature Review

2.1 Existing banknote reader application for visually impaired

2.1.1 Seeing AI

Figure 2.1 Seeing AI Logo

Seeing AI is a free mobile app developed by Microsoft, intended to assist visually impaired people in daily tasks [3]. The application use image processing with AI to recognize things, people, and text. Therefore, it is very helpful for various day-to-day tasks for visually impaired users. Among its many features, Seeing AI allows users to use the banknote reader to identify and differentiate between various banknotes. The app not only just performs simple object recognition, but it also allows short text reading, document scanning, people recognition, and much more, which provides a many-sided solution for the visually impaired community.

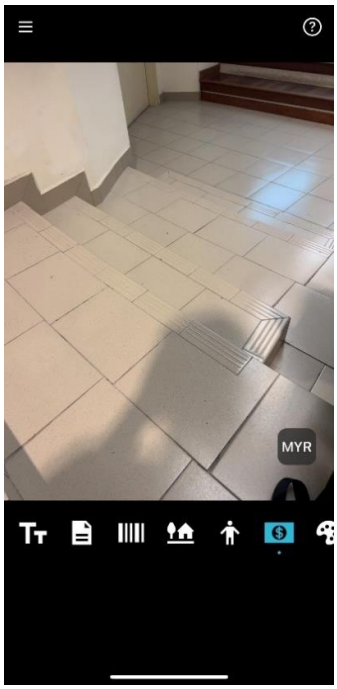


Figure 2.2 Seeing AI Interface

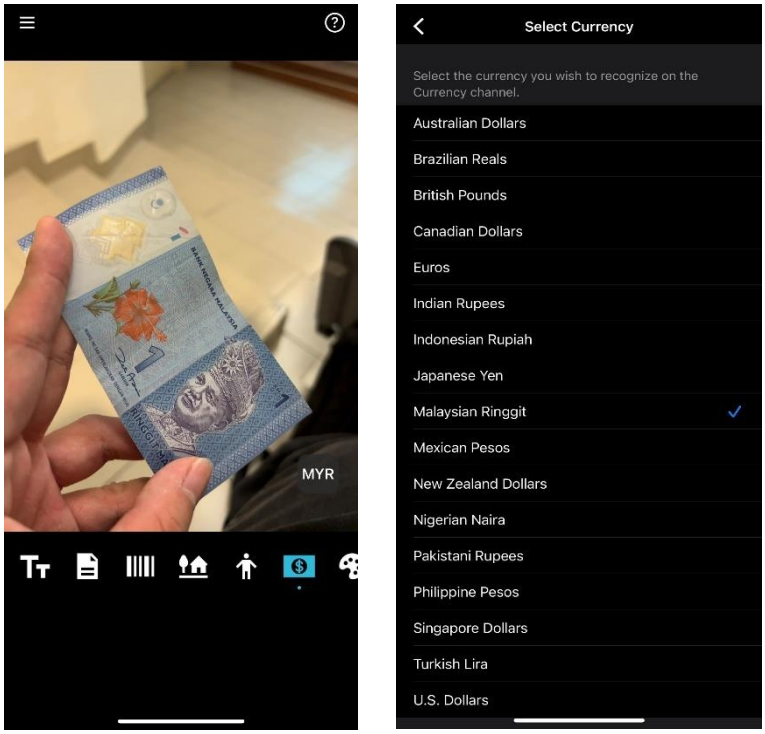


Figure 2.3 Seeing AI Banknote reader (Demo)

Strength of Seeing AI:

The application can recognize a wide range of currencies from different countries. This feature can benefit visually impaired users who need to travel internationally or live in other regions where the environment involves multiple currencies in transactions. By selecting the desired currency, the user can handle the right amount of money confidently no matter in what location which caters to a diverse user base.

When the environment lacks lighting, Seeing AI allows the user to activate the device's flashlight to ensure the currency can be read clearly. This feature greatly enhances the accuracy of the recognition, especially in low-light environments. Visually impaired users might find it difficult to position the camera correctly, but the additional light provided by the flashlight makes sure that the app can capture clear images of banknotes which directly reduces the errors in currency identification.

Furthermore, Seeing AI also provides real-time audio feedback when it recognizes a banknote. This feedback is important for visually impaired users because it allows them to determine the banknote they are handling quick and accurately. The clear audio output provided by the app make sure that users do not have to rely on others' assistance which enhances their independence during transactions.

Weaknesses of Seeing AI:

While Seeing AI supports multiple currencies, it was found that it was unable to recognize the Malaysian RM50 banknote, this has probably happened the same in some of the other country's currencies. This inconsistency can be a significant drawback when the user fails to identify local currency in other regions. Such inaccuracies can make the user question whether the banknote he is holding is probably not a currency, this could reduce the user's confidence when using the app. For visually impaired users who rely heavily on such tools, such inconsistencies can be a source of frustration and stress.

One of the limitations of Seeing AI is its inability to recognize coins, meaning that it can only detect paper currency. For visually impaired users, the lack of coin recognition means that users must still rely on other methods or assistance to handle coins in scenarios such as vending machines. This shortcoming highlights the need for more advanced features that can cover all forms of currency, not just banknotes.

Seeing AI can read the currency quick, however, its performance is highly dependent on the quality of the user's smartphone camera. Users with lower-end devices or older smartphones might experience lower accuracy in currency recognition due to the difficulty of the camera to focus on the currency. This dependency on hardware quality can be an obstacle for some users, especially those who may not have access to the latest technology or have a financial burden on buying good devices. As a result, the app's effectiveness can be different depending on the user's device.

Recommended Solutions

Table 2.1 Recommended Solutions (Seeing AI)

Weakness	Solution
Inconsistent Currency Recognition	Enhance the app's machine learning model by using a more extensive and diverse dataset of banknotes.
Inability to Recognize Coins	Add coin recognition feature that user can active when handling coins.

Dependency on Smartphone Camera Quality	Implement adaptive image processing to tackle lower-resolution images.

Table 2.2 Strength and weaknesses (Seeing AI)

Strength	**Weakness**
• Multi-Currency Support • Able to open device's flashlight • Immediate and clear audio feedback	• Inconsistent Currency Recognition • Inability to recognize coins • Require smartphone camera quality

2.1.2 Cash Reader

Figure 2.4 Cash Reader Logo

Cash Reader is a mobile application developed by Cash Reader R.S.O which specialized designed to assist visually impaired individuals in identifying currency [4]. It supports wide range of curries from around the world, making it a useful tool for individuals who need to handle different currencies. When user use the app, the app will provide immediate audio feedback after it recognizes a banknote to inform the user of the note's information.

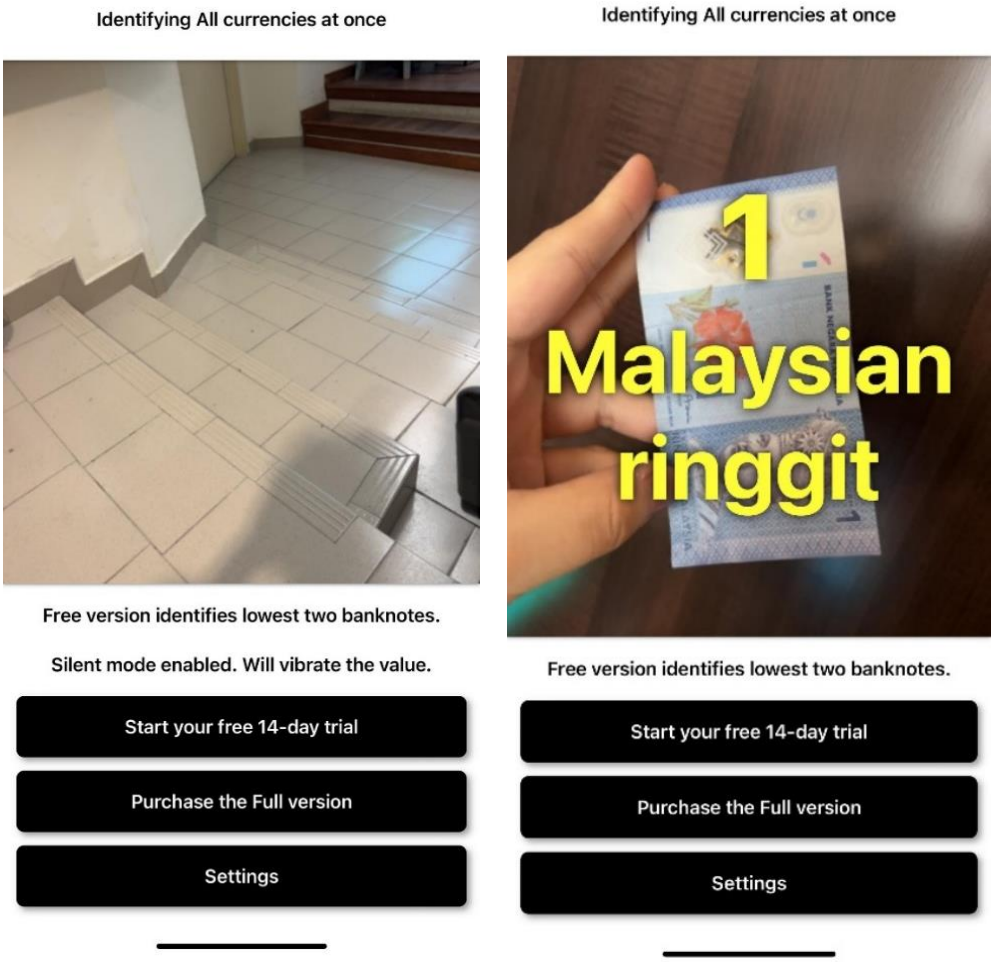


Figure 2.5 Cash Reader Interface

Strength of Cash Reader:

Cash Reader provides a flexible feature where users can choose to detect all currencies at once without having to scroll through a list of countries even it does provide such an option. This can simplify the process of currency reading, especially for users who will have to deal with different currencies regularly. It simplified the process of switching settings when it comes to different currencies, making it convenience for visually impaired users.

Another strength of Cash Reader is it features on user preferences through customizable visual themes. The app allows the users to choose between the default green theme and black and white theme based on their specific visual needs. This flexibility is important for users with different degrees of visual impairment, as it ensures the app can be adjusted to show the best possible contrast and visibility.

In addition to providing audio feedback, Cash Reader also displays the recognized denomination as big text on the screen. This dual feedback system is beneficial for the users who have partial vision, they can prefer to see a visual confirmation of the banknote's value. The text display feature provides an extra layer of guarantee to allow users to cross-verify the audio information.

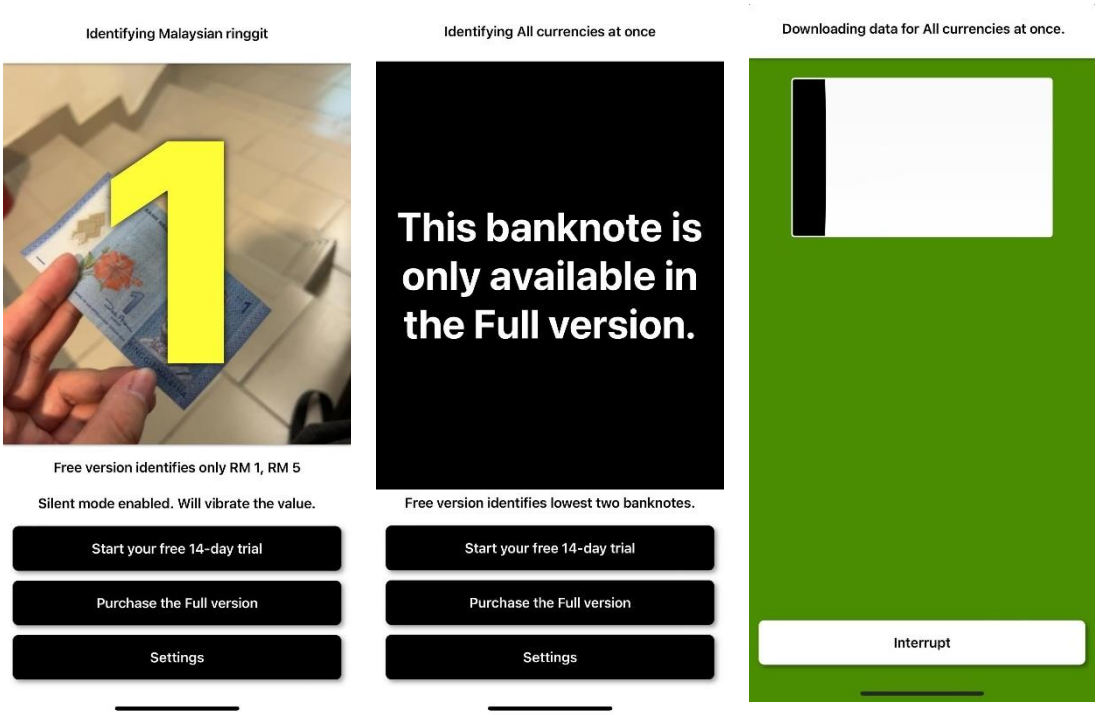


Figure 2.6 Limitations of Cash Reader

Weaknesses of Cash Reader:

While Cash Reader provides a lot of useful features, but users must subscript to its subscription plan to be able to user its full functionality. Non-subscribers can only scan up to RM1 and RM5 in terms of Malaysian Ringgit or all the supported currencies. Even the app allows users to use the free version, the lack of full functionality still makes the tools for free version users inconvenient.

Another weakness of Cash Reader is that if the app's full data is not downloaded, it may only be able to read the numeric value on the banknote rather than fully identifying the currency. For example, it will be showing 1 instead of 1 Malaysian Ringgit on the screen as well as the audio feedback. Therefore, this can be a problem when there are similar-looking notes from different countries at the same time. Users who rely on the app to differentiate between currencies might find this inconvenient, particularly when traveling or dealing with multiple currencies at the same time.

Recommended Solutions

Table 2.3 Recommended Solutions (Cash Reader)

Weakness	Solution
Subscription Requirement	Free to use, ask for donation if the user like to use the app
Incomplete Data Without Full Download	Download all the necessary data when installing the app

Table 2.4 Strengths and weaknesses (Cash Reader)

Strength	Weakness
• Global Currency Detection • Customizable Visual Themes • Text Display Feature	• Subscription Requirement • Incomplete Data Without Full Download

2.1.3 Qatari Money Reader (QAR Reader)

Figure 2.7 Qatari Money Reader (QAR Reader) Logo

QAR Reader is a money ready application developed by Innovation Factory Limited [5]. The purpose of this mobile application aims to assist people with visual impairments to detect and read Qatari Riyal currency notes using smartphone's camera. On top of that, QAR Reader includes other features like reading the Qatari Riyal currency in real-time and works entirely offline, once the currency is detected, the app will inform the user about the value of money in Arabic and English [5].

Figure 2.8 QAR Reader interface

Figure 2.9 Note reading, and currency converter of QAR Reader

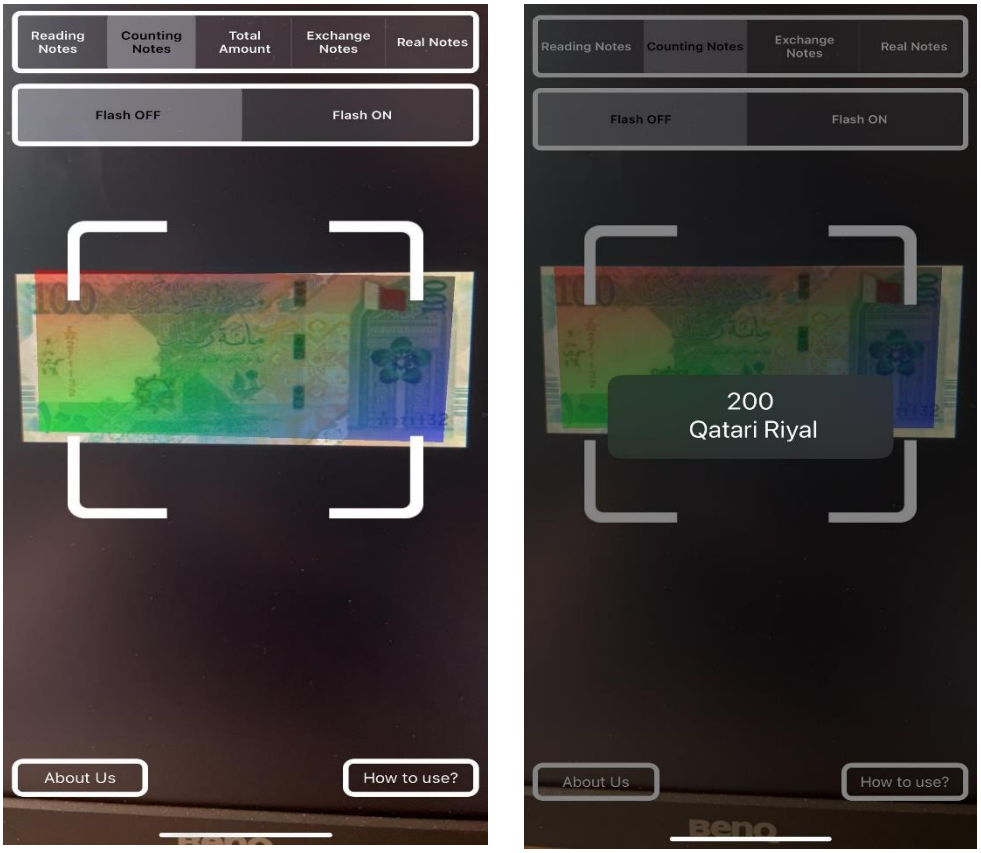


Figure 2.10 Money counting of QAR Reader

Strengths of Qatari Money Reader:

The Qatari Money Reader app uses tactile markers, which are designed on Qatari Riyal notes to help a visually impaired user to identify which bank note the user had scanned. The features help the visually impaired users to able to know which banknotes is not yet scanned which directly reduced the risk of duplicate scanning.

Qatari Money Reader can detect the currency of the current reading note and inform the user to make sure that the note being scanned is one of the Qatari Riyals. This is a very useful feature in a situation when the user has mixed currencies or is not aware of what type of currency it is.

It means that the Qatari Money Reader has a simple and clean user interface, which, regardless of age or technical skills, can be easily learned by any user and worked with. It is designed in such a way that every less experienced user in using technology will be able to operate the app without any difficulty that is why clarity and ease of use have been considered in the design. The direct interface reduces confusion and speeds up the operation when it comes to reaching the key functions of the app.

Weaknesses of Qatari Money Reader:

A significant limitation of the Qatari Money Reader is that it only supports the Qatari Riyal currency. For those users who handle multiple currencies or travel frequently might find this to be a drawback, as they would need to rely on different apps for different currencies. This lack of versatility limits the app's appeal to a broader audience, making it primarily useful only within Qatar or for users who exclusively handle Qatari Riyals.

If the user wants to view the total amount of the notes scanned, then the "Total Amount" button needs to be clicked to show the sum up value on screen. In scenarios where users need to sum up multiple notes in a short period, the lack of automatic display could slow down transactions and might require extra effort from the user, which can be particularly inconvenient for visually impaired individuals who prefer a more seamless experience.

Recommended solutions

Table 2.5 Recommended Solutions (QAR Reader)

Weakness	Solution
Limited to Qatari Riyal	Add feature to detect on other currencies.
Manual Summation of Scanned Amounts	Automatically display the cumulative value on the screen without user must tap the button to see.

Table 2.6 Strength and Weaknesses (QAR Reader)

Strength	Weakness
• Tactile Markers on Currency • Currency Identification • Simple and Clean User Interface	• Limited to Qatari Riyal • Manual Summation of Scanned Amounts

2.1.4 MCT Money Reader

Figure 2.11 MCT Money Reader Logo

MCT Money Reader is a free offline currency recognition app for visually impaired and blind users, allowing users to identify and count a wide range of global banknotes in minimal time with the help of text-to-speech technology [6]. It was developed by MCT Data. The following currencies are supported by this application: US dollar, Euro, and Japanese yen. Moreover, this application can work efficiently under poor light conditions with the support of a phone's flashlight. It also comes in various languages for better accessibility and ease of use. Hence, this app can be very useful for visually impaired individuals.

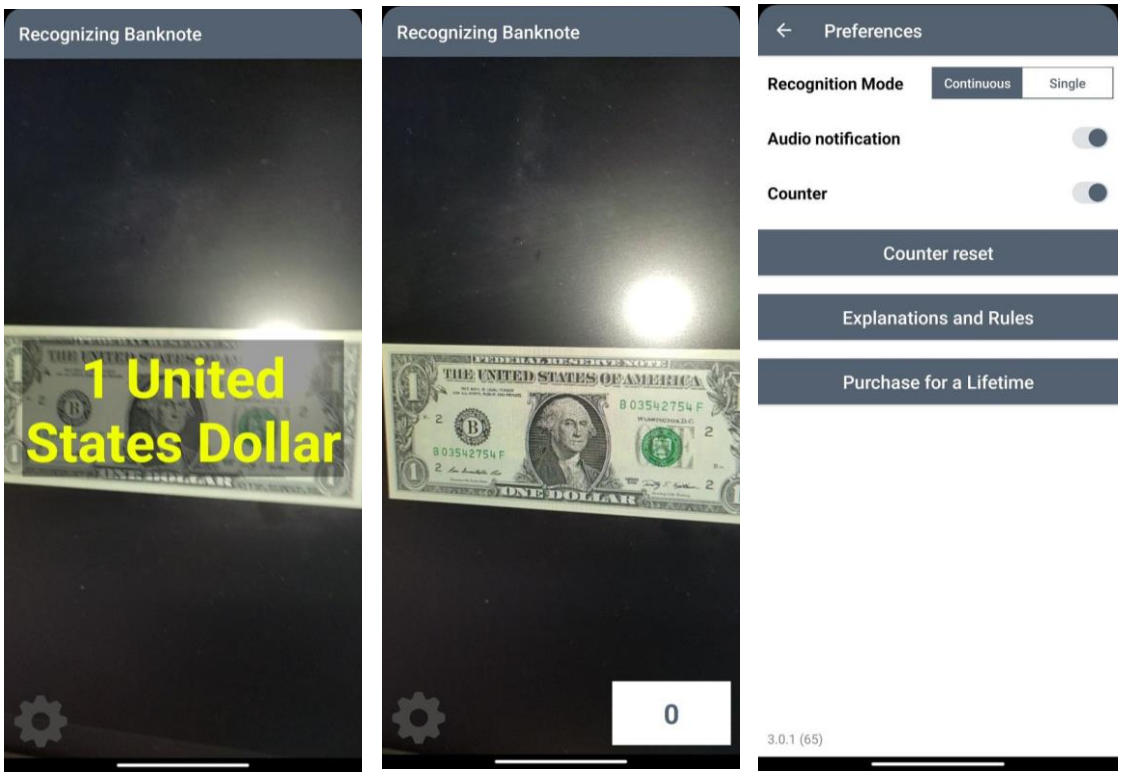


Figure 2.12 MCT Money Reader (Demo)

Strength of MCT Money Reader:

Some of the important features of MCT Money Reader include an inbuilt counter that keeps the numbers for notes scanned. This feature will help users who need to count several notes sequentially, ensuring the right count without necessarily keeping a mental account of the same or using a manual count. It may be useful in an environment where large sums of cash are handled.

It allows the user to select either the continuous scanning mode or the single note scanning mode through its mode selection feature, hence giving flexibility to the application. The continuous mode is fit for situations when a person needs to scan many notes successively with minimal intervention, while single mode allows for more committed and less-burdened processing, one note at a time. This brings in diversity in two different ways: user preferences and situational awareness, which enables easier use of the application.

Weakness of MCT Money Reader:

The major shortcoming of the MCT Money Reader is that one cannot be able to turn the flashlight off when scanning the notes. The flash usually comes on automatically to provide better visibility when taking pictures, but this inability to have a manual override can be an inconvenience to users who find themselves in a brightly lit environment or would not want to use the flash. It can also consume the battery of the device fast, which may be a problem for those users who need this app all day long.

Another important disadvantage of the MCT Money Reader is that it can't scan Malaysian banknotes and recognize them. A reduction in the potential usage area in Malaysia or for those people who have frequent contact with Malaysian money reduces general usability, making the app less flexible compared to other money reader applications that support a wider range of currencies.

MCT Money Reader is only available on Android devices, thus limiting its accessibility for users who own iOS devices. This restriction of platforms is quite a big drawback for those people who prefer using or use exclusively the products of Apple, which reduces the potential user base of the application.

Resolve on weaknesses:

Table 2.7 Recommended Solutions (MCT Money Reader)

Weakness	Solution
Cannot turn off flashlight	Add feature to close on and off flashlight.
Unable to scan Malaysian notes	Train model to be able to scan the Malaysian currencies.
Only available on Android	Develop on both Android and iOS platform.

Table 2.8 Strength and weaknesses (MCT Money Reader)

Strength	Weakness
• Keep track of the number of notes scanned • Option on continuous scanning mode and single note scanning mode	• Cannot turn off flashlight • Unable to scan Malaysian notes • Only available on Android

2.1.5 AR money reader scanner GMoney

Figure 2.13 GMoney Logo

GMoney is a mobile application developed by Beaverhood LLC, designed to facilitate currency conversion using augmented reality [7]. Its main function allows users to calculate the exchange rate of any banknote by pointing the camera of their device on it for instant recognition and conversion. GMoney offers a wide range of currencies worldwide, while its functionality includes automatic conversion, real-time AR banknote frames, and even voice dictation [7]. Also, it works in offline mode, and therefore it is a very handy tool for visually impaired users and people in the far reaches with no access to the internet.

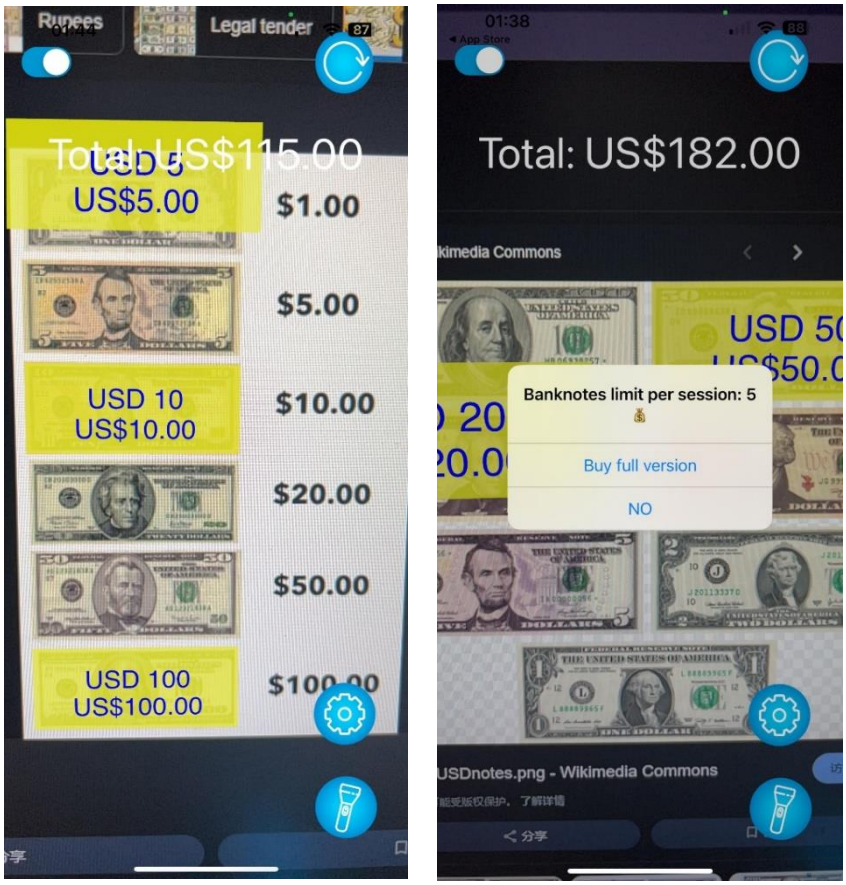


Figure 2.14 GMoney (Demo)

Strength of GMoney:

GMoney used augmented reality markers to further put a layer on top of every banknote while scanning them. Such markers provide a visual clue for visually impaired users to track which notes have been scanned with ease. Furthermore, the value of the money will be displayed on top of the marker. This feature ensures that no currency goes unnoticed, while the user is performing scanning for greater accuracy, hence reducing the rate of errors in counting the money. The markers also appear in real time, making this all one efficient process of scanning.

Another powerful capability of GMoney is the real-time calculation of the total value of banknotes scanned. Immediately after the user scans the banknote, the application will sum up the value for instant feedback on the total. This would benefit users who are visually impaired in fast-paced environments such as retail, where quick and accurate cash handling is essential. Real-time counting saves time, as well as minimizes manual errors, which therefore makes this device an extremely useful tool for rapidly calculating and verifying large sums of money.

Basically, GMoney has one big strength is that it can recognize several banknotes and do that fast. Although it scans the banknotes individually, the app can process each note at high speed, therefore enabling handling large volumes of cash with ease. This rapid scanning helps in situations when visually impaired users need to count several notes in a very short time frame. That efficiency guarantees that the user can handle several notes in a row with only a marginal delay, therefore reducing the time spent counting.

Weaknesses of GMoney:

One disadvantage of the GMoney app is that it has latent responses from its reset button. For example, if the user tries to reset the total value, sometimes the app does not respond immediately; hence, delays arise, which might be annoying in a situation that requires quick move actions. This latency disconnects the work process, especially for those users who want to start another counting session. It cannot provide an immediate response to users, which reduces the efficiency of the application in a critical environment. Though very useful, AR markers can become unstable or "fly away" if the smartphone is moved or shaken during the scanning process. This might make it

unstable to keep the accurate record of scanned notes, especially under certain conditions when the user's hand is unsteady. This could limit the use of the application in cases where any great movement might disturb the placement of markers and could lead to confusion in the count.

The free version of GMoney limits users to scanning a maximum of five notes at a time unless they subscribe to the full version of the app. This limitation can be a disadvantage for the users when they want to scan multiple notes at the same time, then the pop-up window suddenly appears and ask for a subscription. Therefore, this will be both annoying and inconvenient at the same time.

Resolve on weaknesses:

Table 2.9 Recommended Solutions (GMoney)

Weakness	Solution
Latency in Reset Button	Optimize the reset process.
Unstable AR Markers with Movement	Improve the Money Marker.
Manual Summation of Scanned Amounts	Automatically display the cumulative value on the screen without user must tap the button to see.

Table 2.10 Strength and weaknesses (GMoney)

Strength	Weakness
• Real-Time Markers on Scanned Money • Real-Time Value Counting • Detection of Multiple Notes at Once	• Latency in Reset Button • Unstable AR Markers with Movement • Manual Summation of Scanned Amounts

2.2 Comparison of Existing Systems

Table 2.11 Comparison Table of existing systems

Applications / **Features**	Seeing AI	Cash Reader	Qatari Money Reader	MCT Money Reader	AR money reader scanner GMoney	Proposed solution
Tactile Marker			✓		✓	✓
Platform Compatibility	iOS & Android	iOS & Android	iOS	Android	iOS	iOS & Android
Required Subscription		✓		✓	✓	
Real Time	✓	✓	✓	✓	✓	✓
Offline	✓	✓	✓	✓	✓	✓
Recognize Malaysia note	✓	✓				✓
Display recognized value		✓	✓	✓	✓	✓
Recognize coin value						✓
Count number of money scanned				✓		✓
Calculate total amount of money scanned			✓		✓	✓
Voice command						✓
Count tracking of each money classes						✓
Tracking of scanned money					✓	✓
Gesture Detection						✓

2.3 Literature Review on Currency Recognition approach

2.3.1 Currency Recognition For The Visually Impaired People

Abilah CS et al. [8] proposed a mobile-based currency recognition system to assist visually impaired individuals in identifying Indian rupees using machine learning and image processing techniques. The study recognizes the high population of visually impaired people in India and try to address the practical challenge of banknote identification. The system used the Bag of Words (BoW) and Scale-Invariant Feature Transform (SIFT) algorithm which both are effective for object recognition tasks under different lightning, orientation, and scales.

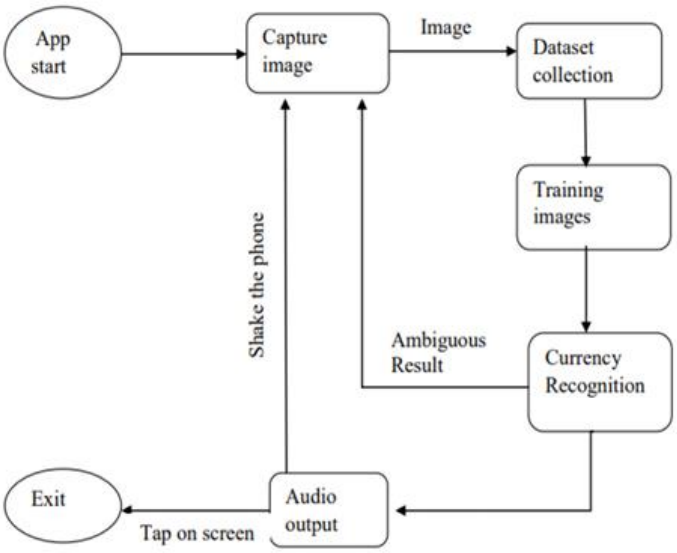


Figure 2.15 System proposed by [8]

The system operates by allowing users to capture images for banknote recognition. A dataset of Indian currency images captured under natural conditions was used to train the recognition model. Using SIFT algorithm as image descriptor to compare and recognize images, the system can maintain robust against noise and different image quality. In terms of classification, the K-Nearest Neighbors (KNN) algorithm was used to match the input currency image to the closest trained example. To enhance recognition accuracy, iterative graph cuts and weak classifiers were used during the segmentation phase to help to isolate the banknote from background noise before feature extraction.

One of the strengths of the system is it able to operate effectively even on low-quality images captured by standard mobile phone cameras which are important to the real-world application. However, the study could have further improved by discussing

performance metrics such as precision, recall, and inference time which limited the understanding on the performance of the method. Furthermore, while the paper discussed visual features like color and shape, it does not explore advanced deep learning techniques such as Convolutional Neural Network (CNN) that could potentially produce higher recognition accuracy based on those visual features mentioned because of its ability to learn hierarchical spatial features from raw image data without the need for manual feature engineering which makes it highly adaptable to complex image patterns.

2.3.2 Malaysian Banknote Reader For Visually Impaired Person

Tan et al. [9] introduced a handheld banknote reader specifically designed to recognize Malaysian banknote. The system integrates perceptual hashing and fuzzy logic to enhance recognition accuracy in real-world conditions. The use of perceptual hashing allows for robust feature extraction from images even when the banknotes are worn or partially damaged, while the fuzzy logic can handle uncertainty in the matching process which improved decision-making reliability.

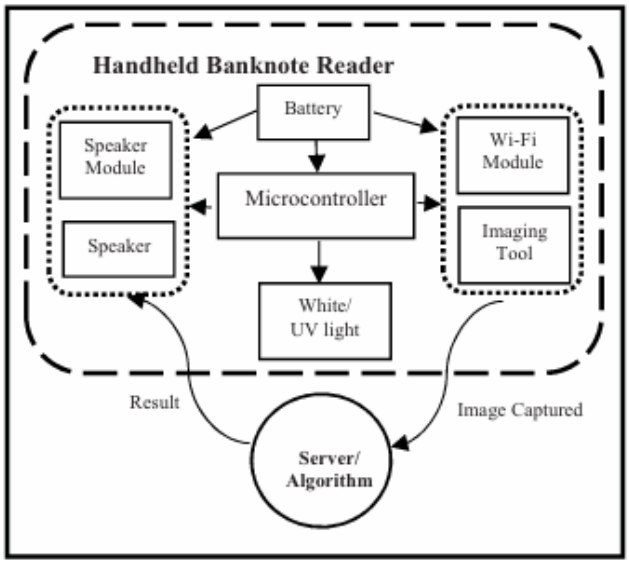


Figure 2.16 Handheld Banknote Reader System [9]

Figure 2.17 Prototype for Malaysian Banknote Reader [9]

The recognition method initially begins with setting up database containing multiple Malaysian banknotes images in different orientation stored on the cloud server. These images are then processed using perceptual hashing to generate hash codes, which are sorted to optimize the retrieval process. During operation, the user presses a button on the device to capture an image of the banknote via the imaging tool. This image will then transmit to the cloud server, where the hash value is computed using the RGB mean from the top-left block of the image.

$$D = \min\{d(x, y) : x, y_n \in C, x \neq y\} \qquad (3)$$

Figure 2.18 Hamming Distance Equation [9]

The system then compares the generated hash of the captured image (denoted as *x*) with the generated hash values stored in the database (denoted as *y*) using the Hamming distance metric, and if a match with at least 90% similarity is found, the input image is considered a match and is classified as the corresponding denomination of banknote in the database.

Previous recognition approaches relying on color and size have face challenges especially in the Malaysian banknote where denominations like RM5 and RM50 have similar color, and the Malaysian banknotes have small dimensional differences. The approach proposed by Tan et al. [9] effectively overcame these limitations, achieving a recognition accuracy of 95% across a dataset of 600 Malaysian banknotes, with a processing time near 1.0 second each reading, therefore offering near real-time performance. However, the system relies on cloud-based processing which introduced limitations where recognition failures might occur in environments with poor or no internet connectivity.

2.3.3 Malaysia coin identification app using deep learning model

The system proposed by [10] used a web-based tool known as Teachable machine to apply deep learning algorithm aimed specifically at recognizing 40 types of Malaysia coins. The primary objective of this application was to introduce the general public to the long history and diversity of Malaysia coins. For each coin class, a dataset consisting of 30 images was prepared to ensure adequate representation and variability. In Teachable Machine, the classification model was trained with different epoch values (50, 80, and 100 epochs) to evaluate the performance trends across different epochs.

Upon completion of training, the best-performing model was exported in TensorFlow Lite format to be integrated into the mobile application developed using Flutter framework.

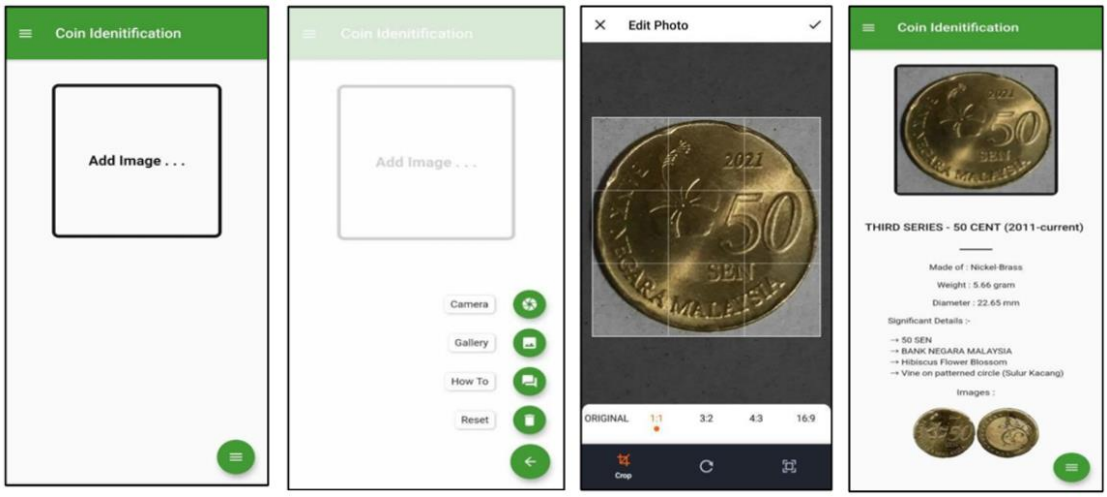


Figure 2.19 Screenshots of proposed method by [10]

Figure 2.19 shows the user interface of the mobile application. Users interact with the application by manually capturing or uploading images of coins then pass to the embedded deep learning model to perform prediction on the coin class and denomination. The proposed system able to achieve mean average accuracy of 80% on the collected dataset demonstrated its feasibility for coin recognition tasks. However, this application is not designed specifically for visually impaired users with accessibility features and does not support real-time coin recognition.

2.3.4 Malaysian Banknote Recognition App for the Visually Impaired using Deep Learning

Hanafiah et al. [11] employed the Single Shot Detector (SSD) MobileNetV2 algorithm to perform real-time classification of Malaysian banknote denomination captured via smartphone cameras. [11] constructed a custom dataset comprising images of Malaysian Ringgit notes (RM1 to RM100), captured under various lighting conditions, angles, and backgrounds to simulate real-world scenarios. The study also applied several image augmentation techniques including image rotation, random flipping, scaling, cropping, and brightness adjustment to improve the model's robustness.

The model training involved fine-tuning a pre-trained SSD MobileNetV2 model. The trained model was then converted into TensorFlow Lite format to be employed on

the mobile application developed using Kotlin programming language within Android Studio for visually impaired users.

Figure 2.20 Output results [11]

Figure 2.20 shows the output results of the mobile application developed by [11]. The system achieved an accuracy of 77% using an input image size of 320x320, a batch size of 32, and 30,000 training steps. In addition to visual output, the application includes an audio feature that announces the detected denomination to the user.

The authors also suggested several future enhancements to improve the system's usability and functionality. These include the voice guidance, vibration feedback, and multilingual voice output such as Malay, English, and Chinese to cater the diversity users in Malaysia. However, the system has some limitations which the accuracy rate of 77% suggests room for improvement especially for real-world reliability. Furthermore, the application is limited to Android platforms only, which reduces accessibility for iOS users.

2.3.5 YOLO-v3 Based Currency Detection and Recognition System for Visually Impaired Persons

Joshi et al. [12] proposed a computer vision-based solution for assisting visually impaired individuals in identifying Indian currency notes. The authors utilized transfer learning on the pre-trained YOLOv3 to develop a real-time currency detection and recognition system. In this study, the model was trained on a custom dataset comprising a total of 3,270 images of Indian currency note captured under different real-world

scenarios such as different lighting angles (both front and side) of the banknotes. To further increase the dataset's complexity and model robustness, the authors applied a variety of image augmentation techniques in combination such as background removal, translation, transformations, reflection, resizing, and addition of random noise.

Figure 2.21 Output results for different rupees denomination [12]

The output results of the system show in Figure 2.21, where it correctly identifies multiple denominations of Indian rupees under different scenarios. According to the authors, the model achieved an average detection accuracy of 95.71% and an average recognition accuracy of 100%, proving the effectiveness of the YOLOv3 model in handling complex scenarios. These scenarios included cluttered backgrounds, partial occlusion, and scale variation, all of which are common in real-world applications.

The system's performance was evaluated using accuracy metrics derived from the test set, which originated from the custom dataset. However, other critical evaluations such as metrics and recall were not discussed. Moreover, the model was not deployed in a mobile application environment, so its real-time performance and usability on handheld devices remain untested.

2.3.6 An Indian Currency Recognition Model for Assisting Visually Impaired Individuals

[13] proposed a system that involves deploying the YOLOv5 model across different platforms which one on the Raspberry-Pi 4 Model B single board computer device, and another by converting the trained model into TensorFlow Lite format to deploy on an Android smartphone to test on the model's performance in real-time. During the image acquisition phase, 2,000 images of unfolded Indian currency were captured simulating real-life scenarios and split into ratio of 7:2:1 through random sampling for train, valid, and test set. The dataset was processed using Roboflow, a computer vision platform that facilitates image dataset management and model training. Data annotation and augmentation in this study were done on the Roboflow. After processing, the dataset was used to train the YOLOv5 model in Google Colab for 3000 epochs. Following training, the model was deployed to both devices mentioned above for performance evaluation.

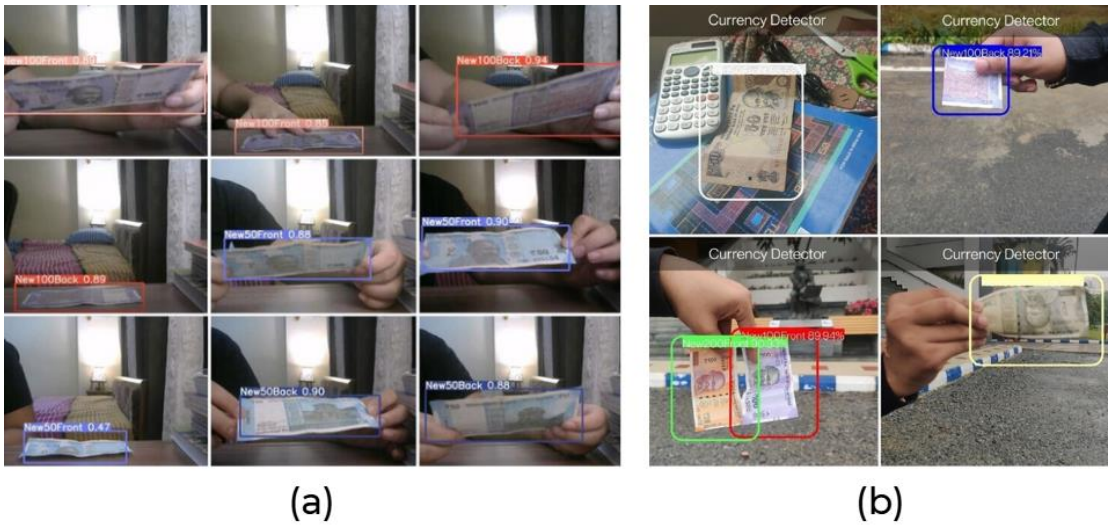


Figure 2.22 Currency notes detected (a) on Raspberry-Pi device and (b) smartphone [13]

Figure above shows the output result from using the model on both the Raspberry-Pi device and on the smartphone, showing the system's capability to detect and recognize currency notes on different platforms in real time. The model achieved an accuracy of 92.71% on the test set, indicating high and reliable performance as a currency recognition system. However, the study was limited to recognizing banknotes only, as coins were not included in the dataset.

2.4 Comparison of Previous Works

Table 2.12 Comparison Table of previous works

Previous Studies	Approach used	Findings	Limitations
[8]	BoW, SIFT, KNN	Recognized Indian currency with robustness to noise	Did not use deep learning technique, missing detailed performance metrics
[9]	Perceptual Hashing, Fuzzy Logic	Achieved 95% accuracy on Malaysian banknotes	Relies on cloud connectivity
[10]	Teachable Machine, CNN	Achieved 80% accuracy on Malaysian coin recognition	Lacks accessibility and real-time support
[11]	SSD MobileNetV2	Achieved 77% accuracy on Malaysian banknotes	Limited to Android, relatively lower accuracy
[12]	YOLOv3	Achieved 95.71% on Indian banknotes	Missing detailed performance metrics, model on mobile performance untested
[13]	YOLOv5	Achieved 92.71% on Indian banknotes	Coins not included

Chapter 3

System Methodology and System Model

3.1 Methodology

The processes of the project were categorized into different phases using the prototyping methodology, which divides the project into several iterative phases. This approach enables continuous improvement by refining the prototype of the system based on the user requirement and overall performance until the final product is complete and acceptable to be used by the visually impaired individuals.

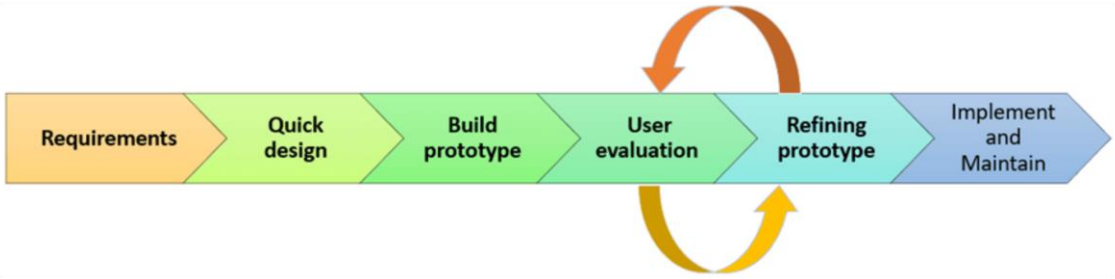


Figure 3.1 Prototyping Development Processes [14]

Requirement Gathering and analysis

In this initial phase, the specific needs of visually impaired users were identified. This includes defining the core functionalities such as real-time money detection, accessible user interface design, and audio feedback features. The project scope, problem statements, and objectives were outlined based on these requirements, ensuring the prototype addresses the actual user challenges.

Quick Design

A preliminary design of the app was created, focusing on accessibility and usability. Wireframes and mock-ups were developed to visualize the user interface, showing large buttons and simple navigation which are specifically suitable for visually impaired users. Furthermore, the architecture design of the app is also in process using diagrams like a use case diagram and activity diagram.

Prototype Building

The initial prototype of the application was built with the following components:

- **Front-End Development**: The app's front-end will be designed using Flutter. The app should also allow access to the camera, provide functionality to stream using camera, and turn on/off the flashlight.
- **Machine Learning Model Integration**: A YOLOv8 model was trained in Python and converted to TensorFlow Lite to perform real-time recognition of Malaysian banknotes and coins on mobile devices.
- **Audio Feedback and Storage**: Design Text-To-Speech feature to provide real-time audio feedback by using the Flutter TTS (Text-To-Speech) plugin. Then use shared preferences to implement the local storage feature to hold some user settings information while being offline.

User Evaluation

The prototype was then evaluated by the users to assess its usability, functionality, and performance. Testers interacted with the app and provided feedback on key areas such as detection speed, accuracy, and ease of navigation.

Prototype Refining

Based on the feedback from user, the prototype was further refined to address identified issues and enhance the user experience. The improvements included adjustments to the detection performance, smoothness of the overall user experience, and enhancing the user interface for better accessibility. The cycle of user evaluation and refinement was then repeated multiple times to ensure the final product able to align with user needs closely.

Implement and Maintenance

After iterative refinements and satisfactory user evaluations, the final version of the application was prepared for deployment. Ongoing maintenance, user feedback collection, and updates will continue to ensure that the application remains effective, accessible, and relevant to its intended users.

3.2 Tools

3.2.1 Hardware

The project's hardware consists of an Android mobile smartphone, an iOS mobile smartphone and a laptop. When used for developing, designing, and testing apps, a laptop boosts productivity by offering plenty of screen real estate for multitasking during the design and development stages. While mobile applications are tested on a mobile device with varying screen sizes and operating systems.

Table 3.1 Specifications of laptop

Description	Specifications
Model	Huawei Matebook D15
Processor	12th Gen Intel(R) Core (TM) i7-12700H 2.30 GHz
Operating System	Windows 11
Graphic	Intel(R) Iris(R) Xe Graphic
Memory	16.0 GB
Storage	460GB

Table 3.2 Specifications of Android Smartphone

Description	Android Smartphone Specification
Model	Redmi Note 10
Processor	Octa-core Max 2.20GHz
Manufacturer	Redmi
RAM	6GB
Storage	128 GB
Operating System	MIUI Global Version 14.0.9

Table 3.3 Specifications of iOS Smartphone

Description	iOS Smartphone Specification
Model	iPhone 13 Pro Max
Processor	A15 Bionic process
Manufacturer	Apple

RAM	6GB
Storage	256 GB
Operating System	iOS Version 18.1

3.2.2 Software

- **Visual Studio Code**

  A flexible code editor that seamlessly combines robust developer features like debugging and code completion with simplicity. It may be accessed on Windows, Linux, and macOS. Visual Studio Code will be used as the editor for writing and managing both Dart and Python code [15].

- **Google Colab**

  A cloud-based Jupyter notebook environment. It supports Python programming with access to GPU and TPU resources, making it ideal for machine learning and deep learning tasks. In this project, Google Colab is used for training the YOLOv8 model with the use of its computational resources to speed up the model development and experimentation.

- **Google Drive**

  Cloud storage solution that is used in this project for managing datasets, model checkpoints, and other project-related files. It can be mounted in the Google Colab for easy access of the uploaded datasets

- **TensorFlow Lite**

  Lightweight version of TensorFlow which designed for deploying machine learning models on mobile and embedded devices. It is used in this project to convert the trained YOLOv8 model into a format compatible with mobile devices, to enable efficient real-time object detection.

- **LabelImg**

  LabelImg is an open-source graphical image annotation tool that supports creating bounding box annotations for object detection models [16]. It is used

to annotation images of Malaysian banknotes and coins, creating the dataset required for training the YOLOv8 model.

- **Anaconda Prompt**

  Command-line interface which used to install and open the LabelImg software for this project.

Coding Language

- **Python**

  Language used to develop and train machine learning models for banknote and coin recognition.

- **Dart**

  Language used with Flutter to develop the whole mobile app as well as user interface, real-time camera access and user interaction.

Technologies

- **Flutter**

  Flutter is an open-source framework for building natively compiled, multi-platform applications from a single codebase.

- **YOLOv8 (You Only Look Once) model**

  YOLOv8 is an advanced real-time object detection model that can be used to accurately and quickly recognize and classify objects.

3.2.3 Flutter plugins

Table 3.4 Flutter plugins

Plugin	Version
camera	^0.10.0+4
light	^3.0.1
flutter_vision	^1.1.4
flutter_tts	^4.2.2

shared_preferences	^2.5.2
vibration	^3.1.3
just_audio	^0.9.46
easy_localization	^3.0.7+1
speech_to_text	^7.0.0
permission_handler	^11.0.1

3.4 Timeline

The development of this project is divided into two phases which are Project I and Project II:

Project I contributes for 30% of the overall project, which focuses on model training and initial integration for detection on mobile devices. This phase involves refining the YOLOv8 model for accurate detection and classification of Malaysian banknotes and coins, including dataset collection, preprocessing, augmentation, and iterative training. It also includes testing the model to evaluate its performance on determine the denomination of the money.

Project II contributes 70% and will be focusing on the full development of the mobile app for both Android and iOS platforms and further enhancement of the YOLOv8 model. This phase integrates the trained model into a user-friendly interface for real-time detection and counting, while also implementing some accessibility features such as audio feedback, flashlight, and vibration notifications for visually impaired users.

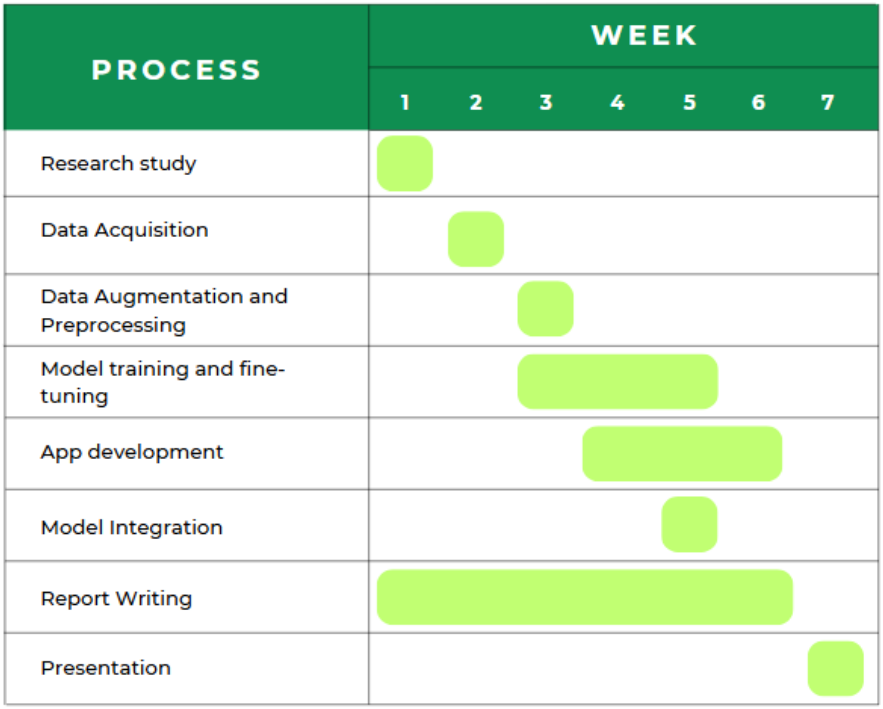


Figure 3.2 Timeline Gantt Chart for FYP 1

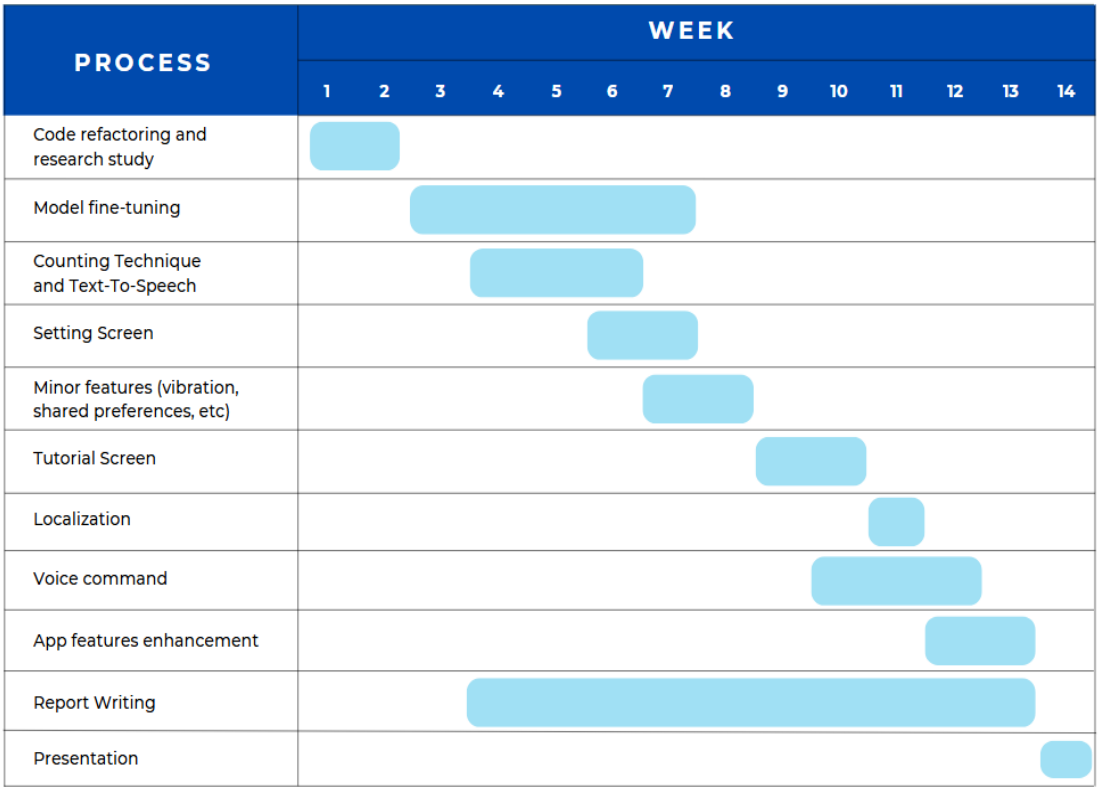


Figure 3.3 Timeline Gantt Chart for FYP 2

Chapter 4
System Design

4.1 System flowchart

4.1.1 Overall system flowchart

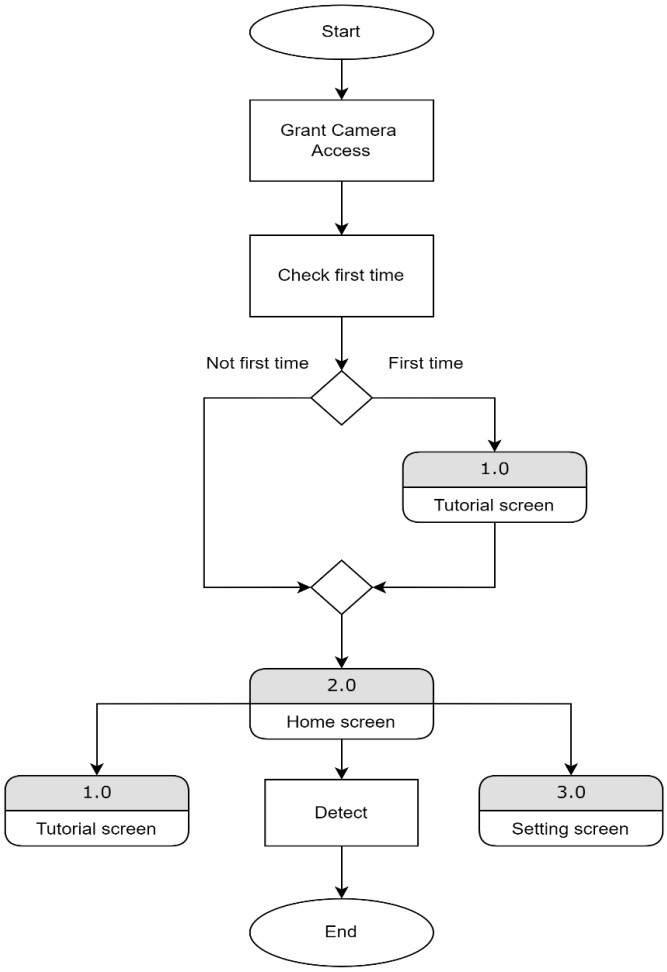


Figure 4.1 Overall System Flowchart

Figure above outlined the overall flow of the system. Initially, the system will request the user's permission to access the device's camera. Next, it will check whether it is the first-time user using this system. If so, the system will navigate the user to the tutorial screen that provides an overview of the system's functions and how to use it. After that, user will proceed to the home screen which is where the main interface of the system where the detection and classification of the banknote take place. On the home screen, the user can select whether to go to tutorial screen again or setting screen. Once the user finishes the money counting process, the system flow ends.

4.1.2 Tutorial screen flowchart

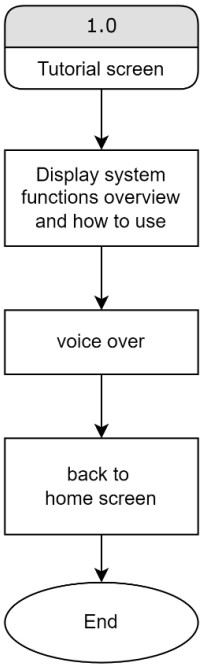


Figure 4.2 Sub-Flowchart of tutorial screen

4.1.3 Home screen flowchart

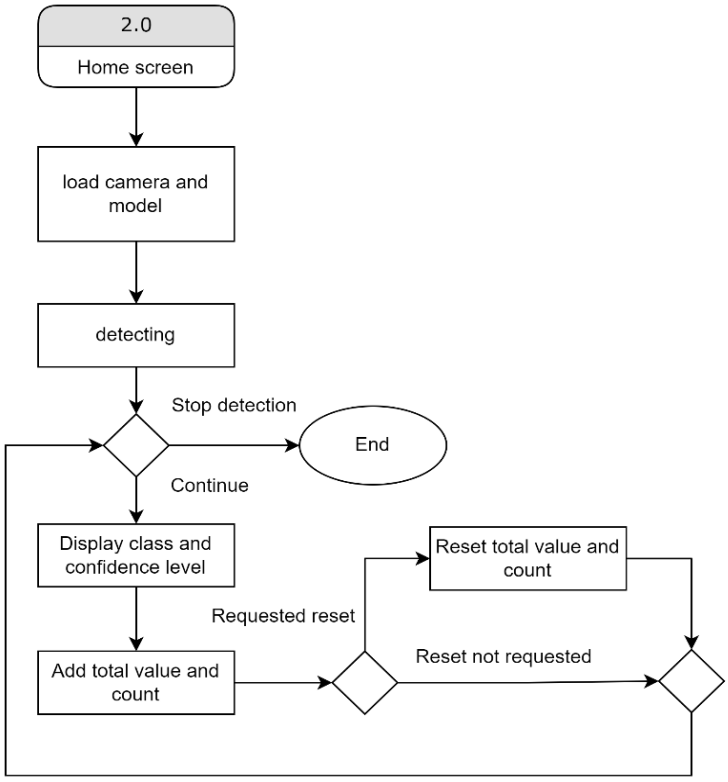


Figure 4.3 Sub-Flowchart of Home screen

4.1.4 Setting screen flowchart

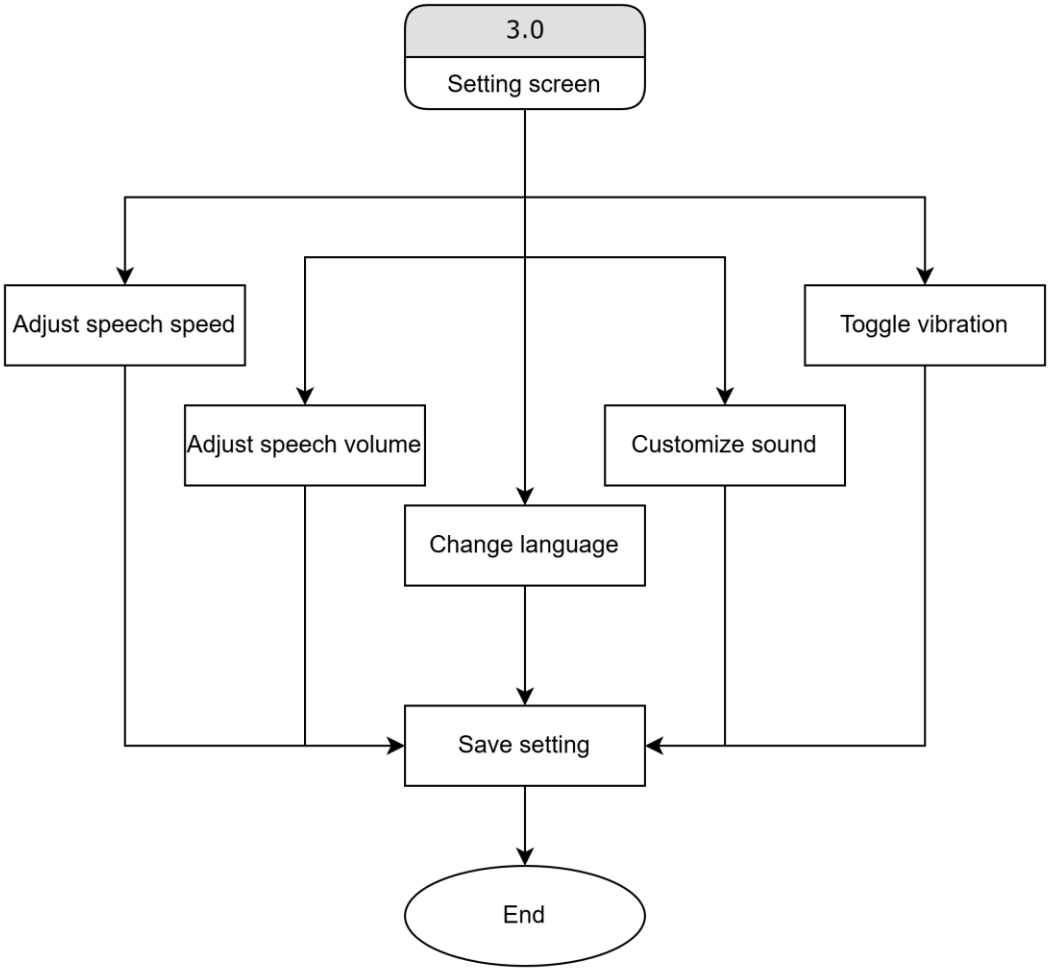


Figure 4.4 Sub-Flowchart of Setting screen

In the setting screen, after the user completes the settings, the changes are saved and the flow ends.

4.2 Use Case Diagram

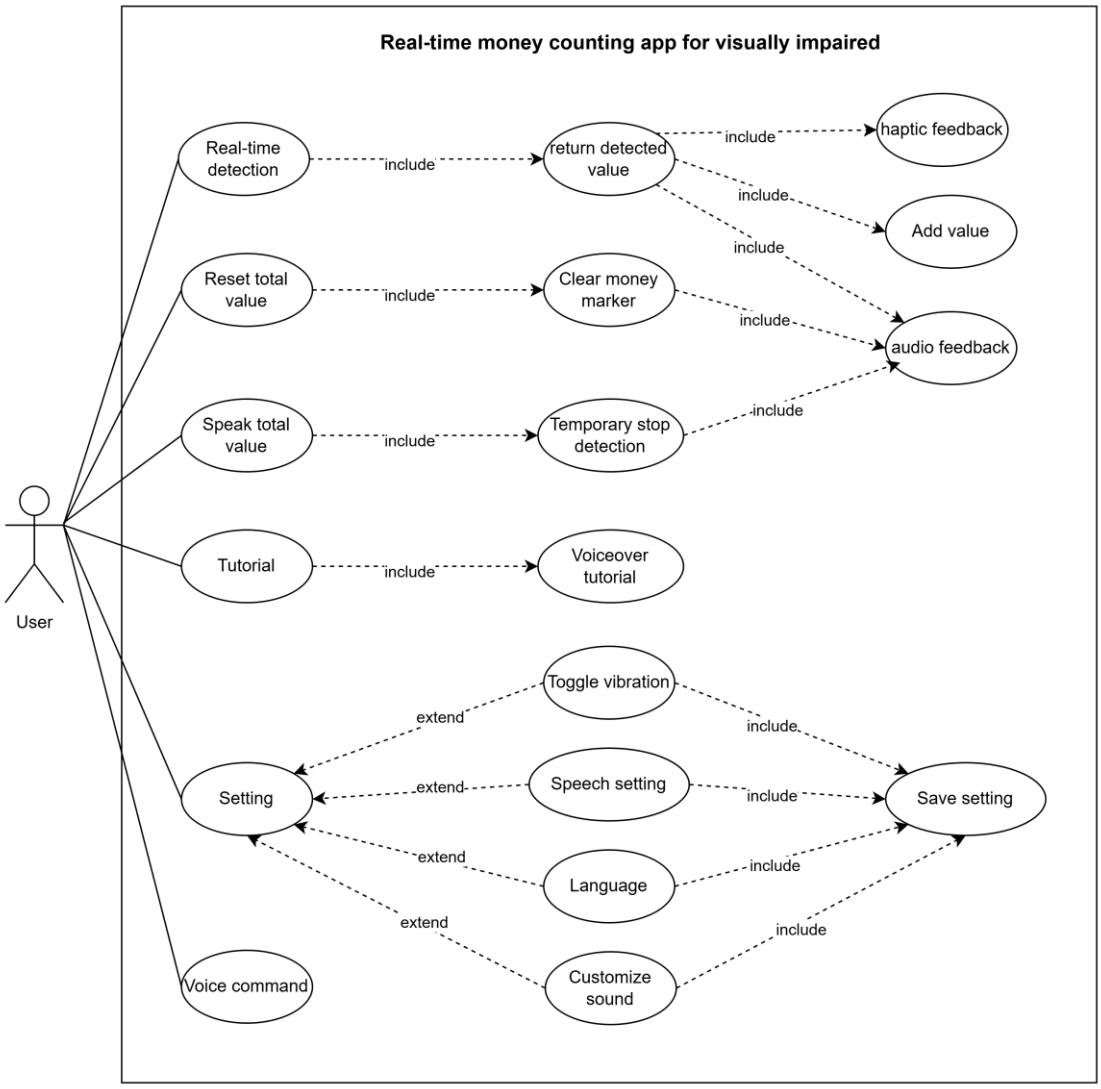


Figure 4.5 Use case Diagram

The use case diagram shows what users can do within the mobile application. The user can perform real-time detection on Malaysian banknotes and coins, with the app displaying the denomination on the screen along with audio and haptic feedback. The detected value is added to the total for money counting. Secondly, the user can reset the total value to start a new counting session or when accidentally scanning the same money. Additionally, a button allows to speak the total scanned value in the current scanning session on the screen provided with audio feedback. The app also includes a tutorial page with voiceover instructions and a settings menu for the user to set customization such as contrast, voice volume, voice feedback speed, language, and sound customization. Lastly, the app also include voice command feature for user to perform tasks hands-free.

4.3 Activity Diagram

The activity diagram shows the overall process of the application, detailing the key steps and interactions within the system. The diagram provides a high-level view of how the application processes user inputs, detects and classifies banknotes and coins, and delivers the output in an accessible manner.

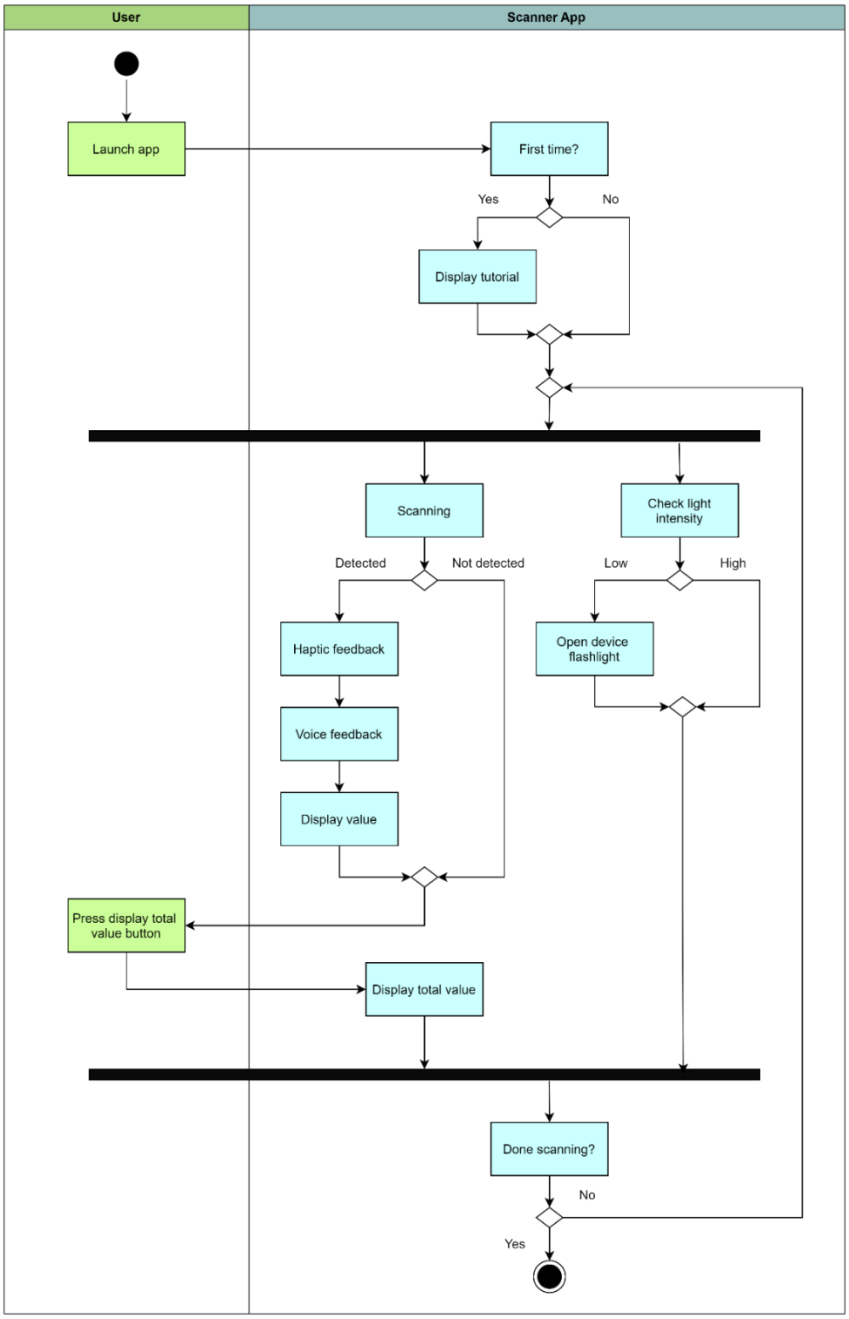


Figure 4.6 Activity Diagram

4.4 Prototype Design

4.4.1 Wireframes Prototype Design

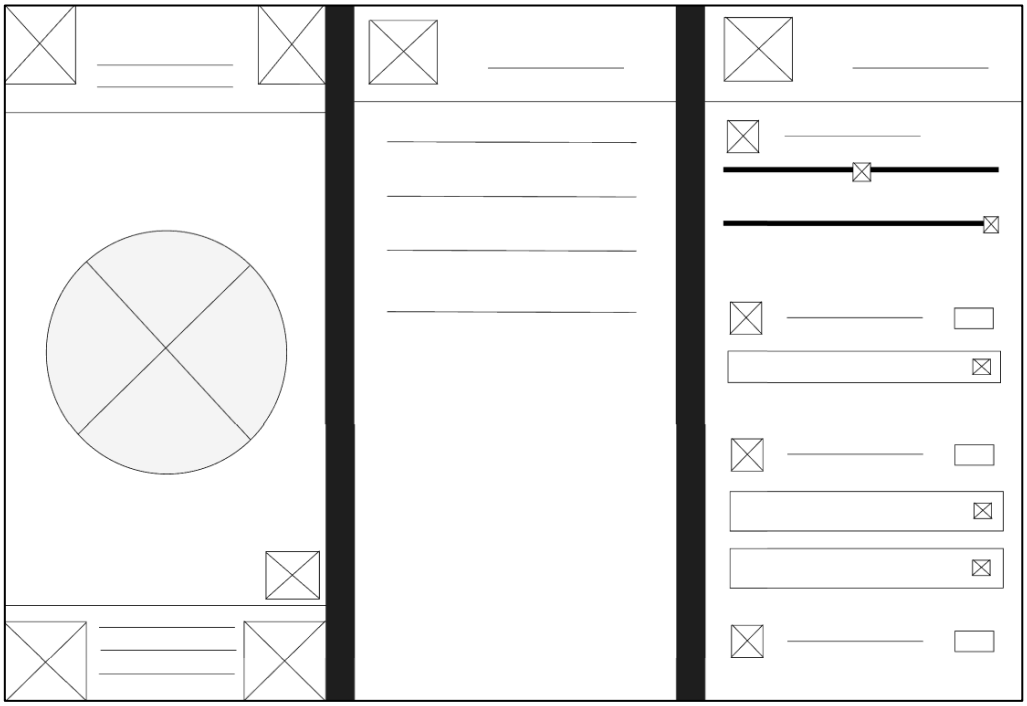


Figure 4.7 Wireframe Prototype Design

4.4.2 Low-fidelity prototype

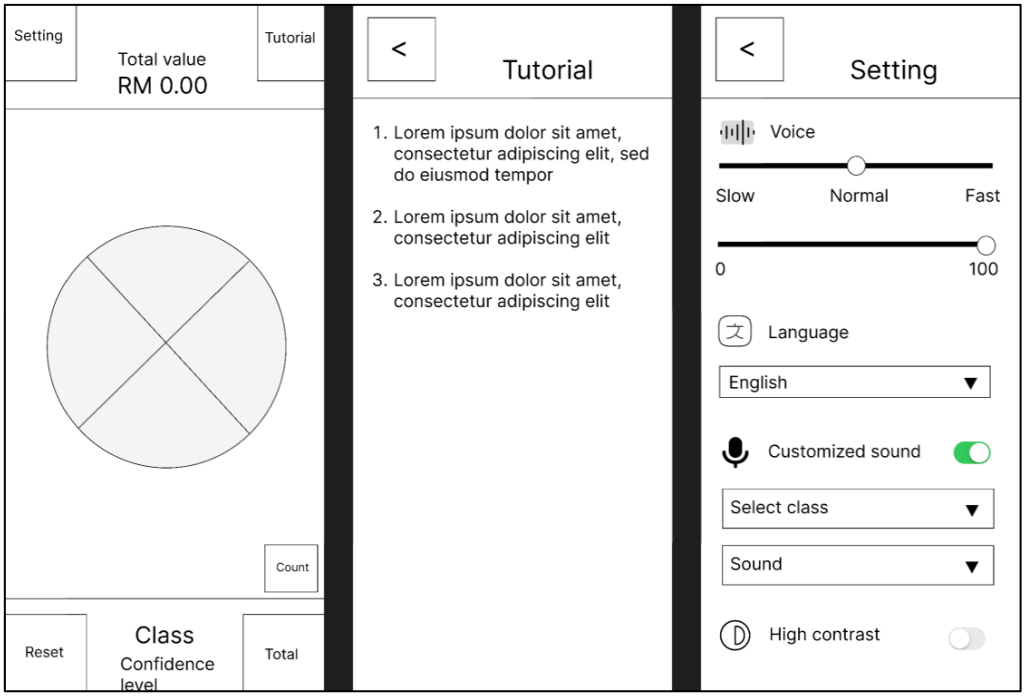


Figure 4.8 Low-Fidelity Prototype

4.4.3 High-fidelity prototype

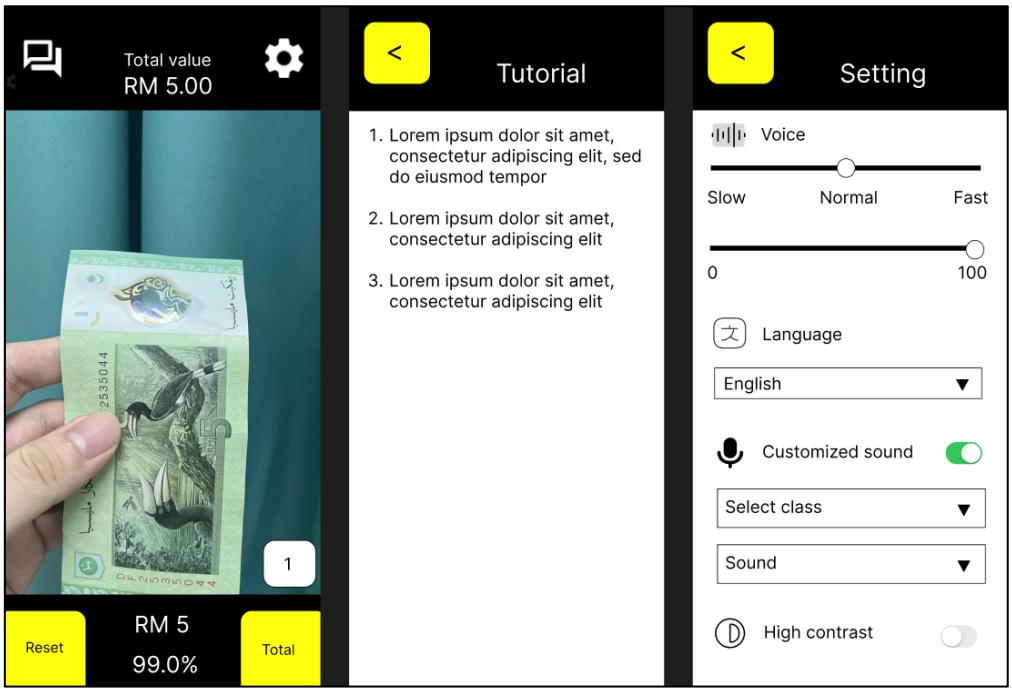


Figure 4.9 High-fidelity prototype

4.5 Storyboard

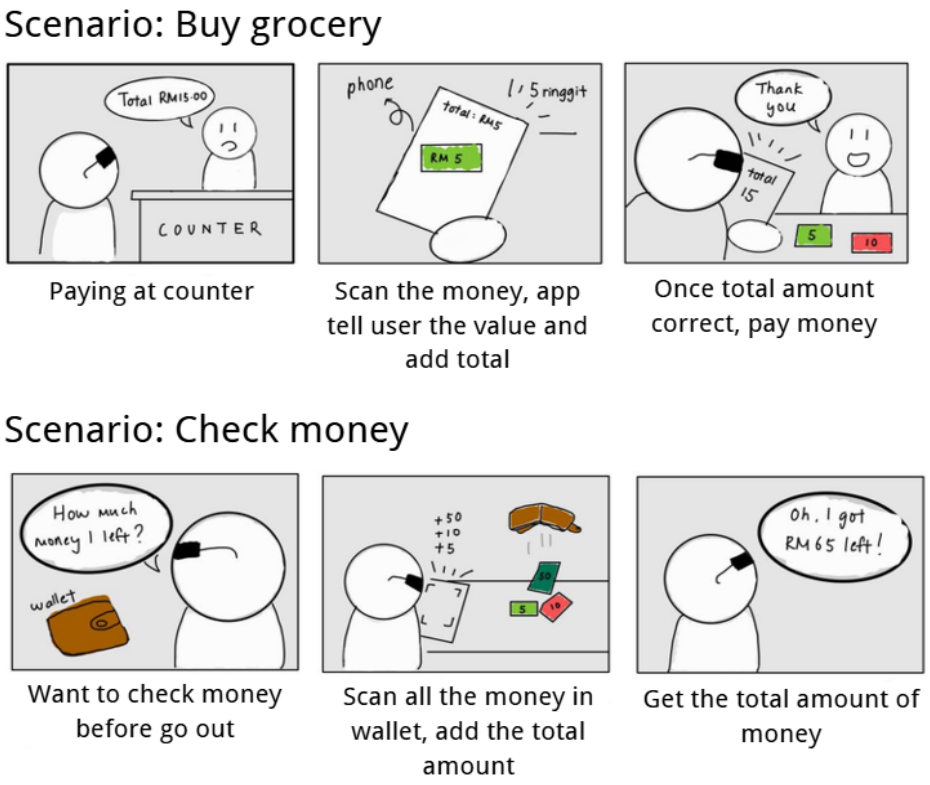


Figure 4.10 Storyboard

4.6 System Architecture

4.6.1 YOLOv8 Architecture

In this project, YOLOv8 was chosen because it represents a mature and well-established version of YOLO by offering reliable performance for real-time applications. YOLOv8 has been released quite a while, therefore it is stable and being widely used within the machine learning community, also there is sufficient documentation available. While newer versions such as YOLOv9, YOLOv10, or YOLO11 may potentially result in higher performance, but they are currently incompatible with the existing Flutter plugin, which only supports up to YOLOv8. This limitation is illustrated in Figure 4.11, where attempts to load models of newer YOLO versions result in errors.

```
, java.lang.Exception: Model version must be yolov5, yolov8 or yolov8seg
yolo_model(FlutterVisionPlugin.java:222)
```

Figure 4.11 Error loading other versions of YOLO models

The YOLOv8 model offers various variations such as detection, classification, and segmentation, each exists for specific tasks. For this project, the detection variant was chosen because the main goal of this project is to detect and recognize real-time video frames, which aligns perfectly with the capabilities of the detection model. Unlike classification, it only identifies the type of object without its location and is not typically optimized for real-time object detection which will be slower for real-time processing. Additionally, YOLOv8 offers multiple model sizes such as n (nano), s (small), m (medium), l (large), and x (extra-large) to tailor to diverse resource requirements and performance needs. Among these sizes, YOLOv8n (nano) was selected for its lightweight design, making it ideal for mobile applications where computational efficiency and speed are important. Despite being the smallest model in the YOLOv8 sizes, it can achieve satisfied speed and accuracy for detecting currencies while minimizing processing time and resource consumption. This choice ensures that the app can run smoothly on a wide range of devices, especially those with limited hardware capabilities.

The YOLOv8 model integrate advancements from previous YOLO versions and include components to enhance feature extraction, processing efficiency, and detection accuracy. The architecture of YOLOv8 model mainly consists of three key blocks: **Backbone**, **Neck**, and **Head** [17].

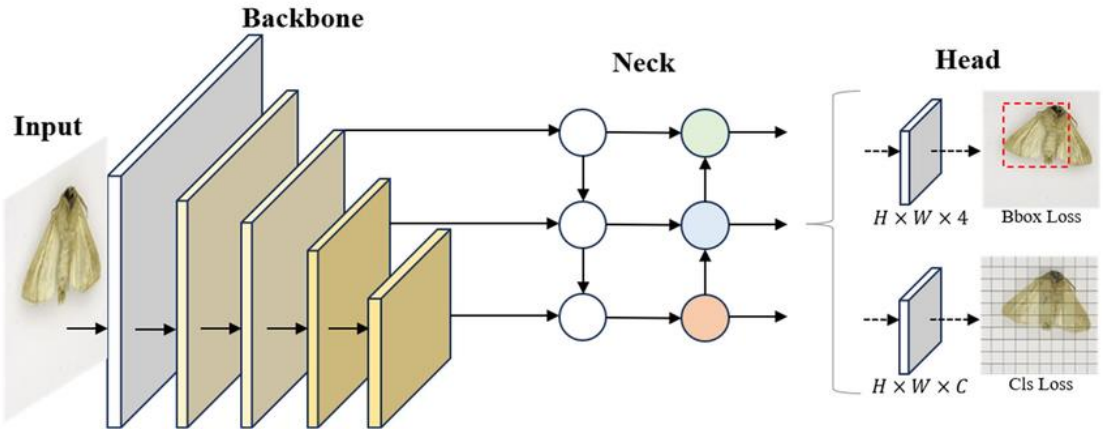


Figure 4.12 Overall detection architecture of YOLOv8 [18]

Backbone: This block is like the "eyes" of the model, it extracts important details from the input image. YOLOv8 uses a specially designed Convolutional Neural Network (CNN) called CSPDarknet53 which breaks down the image into layers to find patterns like edges, shapes, and textures by using a method called cross-stage partial (CSP) connections to make the process faster and more accurate by sharing information across layers.

Neck: The neck is like the "brain" which combining all details found by the backbone. It merges the features from different layers to make sure the model can detect objects of all sizes. YOLOv8 uses a C2f module to mix detailed low-level information with high-level abstract features to improve accuracy, especially for detecting small objects.

Head: It is like the "decision-maker" which decides what are the objects and where are them. It makes the prediction based on the combination of bounding boxes, confidence scores and class probabilities for each grid cell in the feature map.

Beyond these key blocks, YOLOv8's architecture is composed of several basic blocks: **Convolutional Block**, **C2f block**, **SPPF (Spatial Pyramid Pooling – Fast) block**, and **Detect block**.

Convolutional Block

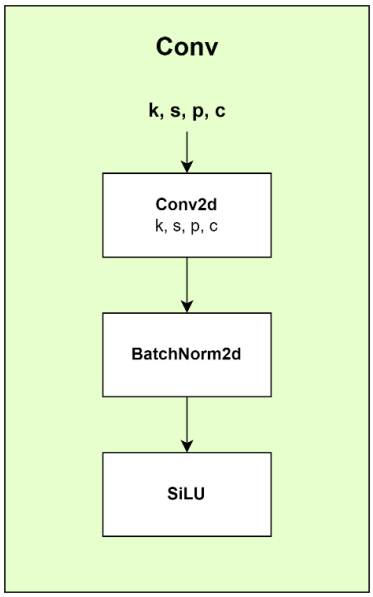


Figure 4.13 Convolutional Block

Figure 3.13 illustrates the Convolutional Block which is the most frequently used block and foundational component of the YOLOv8 architecture, serving as the primary feature extractor. It consists of three key components: 2D Convolutional layer (Conv2d), 2D Batch Normalization (BatchNorm2d), and SiLU (Sigmoid-Weighted Linear Unit) activation function.

The Conv2d layer is known as 2D convolution layer, it applies filters (kernels) and computing dot products between the filter and local regions of the input. BatchNorm2d normalizes these feature maps, ensuring consistent distribution and stabilizing the training process. Finally, the SiLU which is the Sigmoid Linear Unit is an activation function that introduces non-linearity by weighting inputs with their sigmoid values, resulting in smooth gradients and enhanced performance, specifically for capturing tiny details.

C2f Block

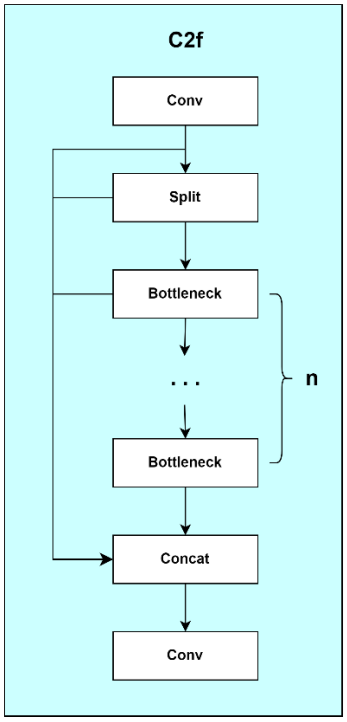


Figure 4.14 C2f Block

The C2f Block (Cross Stage Partial Fusion) was refined from YOLOv7, to optimize information flow between layers to enhance feature reuse and computational efficiency. It divides the feature maps into two parts: one is processed through convolutional layers, while the other bypasses and merges back later. This structure retains original features while learning new ones, improving gradient flow and reducing vanishing gradient risks. Its partial fusion mechanism minimizes redundant computations, keeping the YOLOv8 model lightweight and fast. By integrating shallow and deep layer information, the C2f Block ensures robust feature extraction, important for high detection accuracy.

SPPF (Spatial Pyramid Pooling – Fast) Block

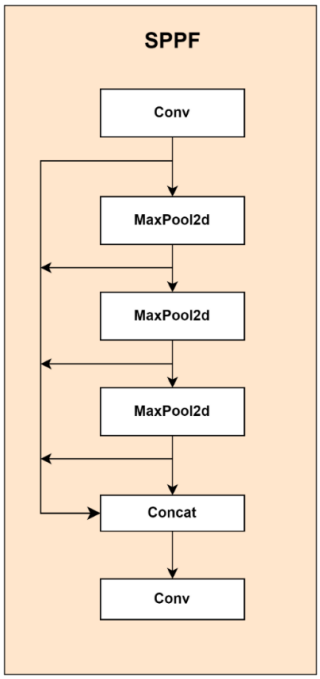


Figure 4.15 SPPF Block

The SPPF Block (Spatial Pyramid Pooling – Fast) includes a convolutional block followed by three MaxPool2d layers. The feature maps from these layers are concatenated and passed to another convolutional block.

Spatial Pyramid Pooling (SPP) basically divides an image into grids, then pools feature from each grid, and enables networks to handle images of different sizes by capturing multi-scale information [19].

Pooling layers are used to down samples the spatial dimensions of the input to reduce computational complexity and focus on dominant features. It retains only maximum value in each region of the tensor input while the rest are discarded. In MaxPool2d, pooling is applied across both height and width and is defined by parameters such as kernel size and stride.

Detect Block

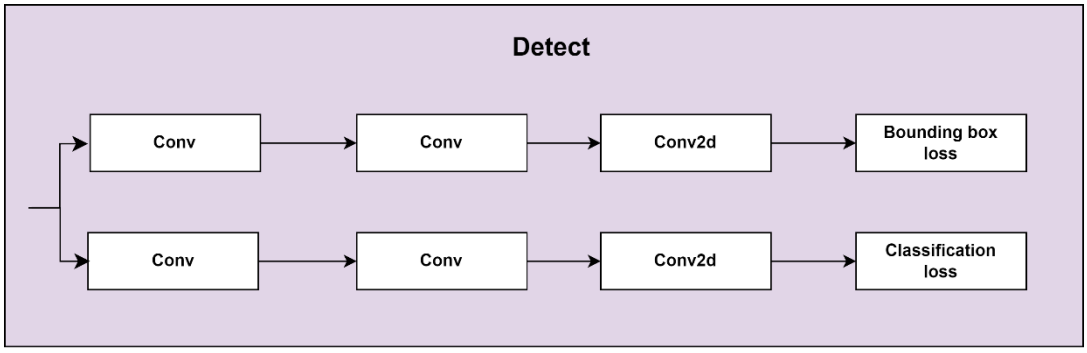


Figure 4.16 Detect Block

The Detect Block is the final component in YOLOv8, it is responsible for generating detection results. Unlike previous versions of YOLO, YOLOv8 is anchor-free which means it directly predicts object's centre rather than offsets from anchor boxes. Therefore, it could reduce the number of predictions and speed up the post-processing process. This block includes two tracks, one for bounding box predictions and another for class predictions. Each track includes two convolutional blocks followed by a Conv2d layers which computes the Bounding Box loss and Class loss.

By combining these foundational blocks, the detailed Backbone, Neck, and Head of the YOLOv8 model is as shown below:

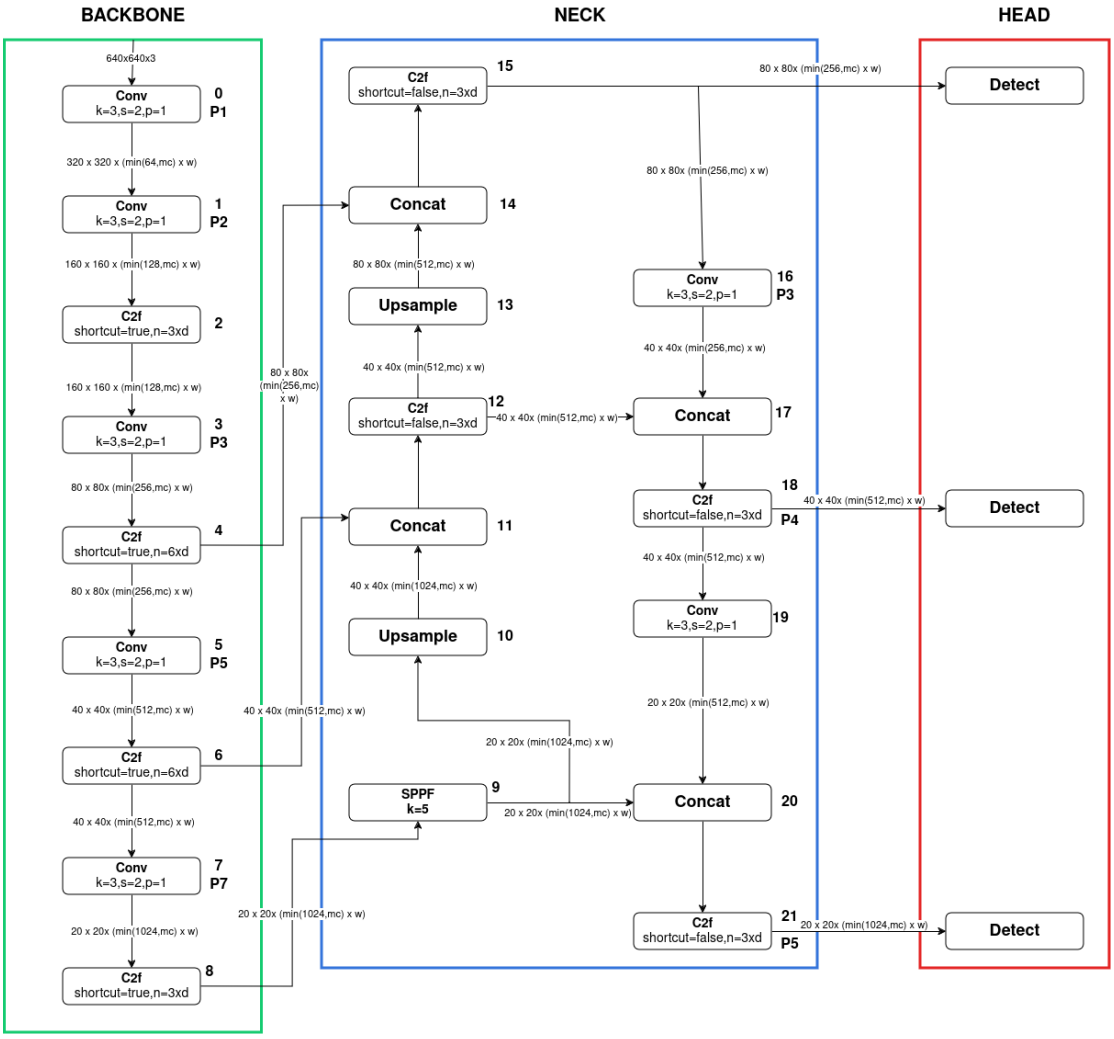


Figure 4.17 YOLOv8 detailed architecture [19]

4.6.2 Mobile Application Architecture

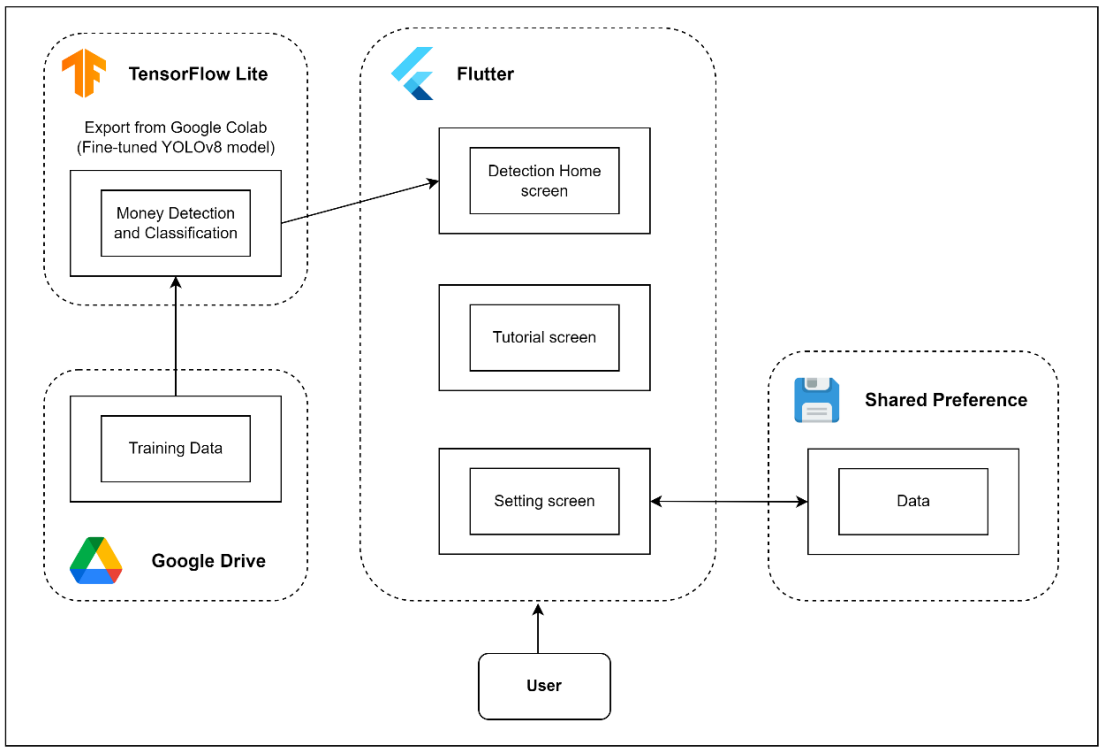


Figure 4.18 Mobile Application Architecture

The overall system architecture for the Real-time money counting app for visually impaired is as shown as above. The architecture integrates various components with Flutter serving as the primary framework for the entire application. The Flutter framework is used to design and manage the user interface, logic, and interactions within the app. It compromises three main blocks: **Detection Home Screen**, **Tutorial Screen**, and **Setting Screen**.

Flutter:

Detection Home Screen

The Detection Home Screen is the primary interface where users interact with the app's detection feature, it will be performing two main functions:

- **Model Loading and Integration**

  The trained YOLOv8 model is loaded and initialized through this screen to perform real-time currency detection and classification. Once the model processes with the video feed by device camera, it will return the predicted value in a list back to the Flutter to further process the output to display to the user.

- **Camera Initialization for Video Streaming**

  The Detection Home Screen will also initialize the device's camera to enable live video streaming. This feature will be continuously streams the camera feed to capture real-time images of the user's surroundings. The live video stream is processed frame by frame, then each frame is sent to YOLOv8 model for detection and classification

- **Light intensity detection and Flashlight Activation**

  The Flutter plugin will be integrated into this screen, it will be initialized after initialization of the camera because the flashlight will be using camera flashlight, else the camera will not be opened. After initialization of this plugin, it will keep returning the lux value which is the light intensity value. If the measured lux value falls below a threshold (e.g., 30lux, indicating a dark environment), the app will automatically activate the device's camera flashlight to improve the clarity of images passed to the YOLOv8 model so that the detection accuracy remains consistent, even in low-light environments.

Tutorial Screen

The Tutorial Screen serves as an onboarding and guidance tool for new users. This screen includes voice-guided step-by-step instructions to familiarize users with the app's feature. It explains how to use the camera and position on holding money for scanning, how to interpret feedback, and how to navigate other features such as settings and resetting totals. The voiceover feature ensures that visually impaired users can fully understand and utilize most of the app's functionalities with minimal assistance.

Setting Screen

The Setting Screen allows users to personalize the app according to their preferences, to make the use of the experience more accessible to visually impaired users. Users can modify the following settings:

- **Voice Feedback**

  Allows users to modify the speed and volume of audio output based on their personal preferences. This feature ensures that users can comfortably interpret the denomination of detected banknotes and coins.

- **Language Preferences**

  This setting enables users to select the preferred language for audio and visual feedback. By supporting multiple languages, the app caters to a diverse audience. For example, the app could support Malay, English, and Chinese to meet the linguistic needs of users in Malaysia.

- **Sound Customization**

  This unique feature provides users with the ability to customize audio feedback for specific currency denominations. For example, instead of announcing "RM10" aloud, the app could play a sound that the user has assigned to RM10 class to prevents others nearby from overhearing the exact denomination detected, which will be useful in environment where privacy is a concern.

- **Toggle Vibration**

  This option allows users to enable or disable the vibration feedback feature within the application. Vibration feedback provides an additional cue to help visually impaired individuals to receive confirmation through the tactile signals.

Shared Preferences:

Shared Preferences are an ideal solution for storing small amounts of persistent data as key-value pairs, such as user-selected settings, due to their simplicity and efficiency [20]. Therefore, the Setting Screen uses Shared Preferences, a lightweight key-value data storage solution, to save user preferences locally on the device. These preferences are automatically retrieved and applied each time the app is opened, ensuring consistent user experience without the need for reconfiguration. Shared Preferences is preferred because it does not require a dedicated database and is straightforward to implement.

TensorFlow Lite:

The architecture relies on TensorFlow Lite to deploy the trained YOLOv8 model that is trained in Google Colab. The YOLOv8 will be exported into (.tflite) format to integrate into Flutter because TensorFlow Lite is a lightweight, mobile-optimized framework that enables fast and efficient inference directly on user's device.

Additionally, by using this method, the need for internet connectivity can be eliminated to enable the detection on the app can be worked offline too.

Google Drive:

The architecture also includes Google Drive as a data source used during the model training phase. The processed datasets of Malaysian banknotes and coins are stored on Google Drive, ensuring easy access while training in Google Colab. Although Google Drive is primarily used during development, it remains part of the system architecture because there will be a need for ongoing updates to the model in future. For example, if new banknotes designs are introduced or improvements to the detection are required, Google Drive can act as a repository for new training data. This makes sure that the app can stay relevant and reliable over time.

Chapter 5

System Implementation

5.1 Software Setup

To begin the development process, several software tools must be installed and configured on the device.

1. **Flutter (version: 3.24.5)**

- Go to Flutter official website: https://docs.flutter.dev/get-started/install
- Select installation package based on the operating system and download the Flutter SDK zip file by following the instructions provided on the page.
- Once the download is complete, extract the file and add the bin directory of Flutter to the system path.
- In terminal of visual studio code, run command ‘flutter doctor’, if there are no issues found then the installation is completed.

2. **Visual Studio Code (version: 1.95.3)**

- Download from the official website: https://code.visualstudio.com/download
- After successfully installed the software, go to the extension of the Visual Studio Code and install the Flutter extension to allow Flutter development in the software.

3. **Android Studio (version: 2024.1.1)**

- Download from the official website: https://developer.android.com/studio
- After completely installed and set up, follow the instructions to download required Android SDK
- In the SDK manager of Android Studio, install Android SDK Command-line Tools
- Lastly, proceed to the terminal of visual studio code to enable Android License by using the command: flutter doctor --android-licenses

4. **Anaconda Prompt (version: 23.3.1)**

- Download from the official website: https://www.anaconda.com/download/success
- Follow the instructions and set up the directory path

5. **LabelImg (version: 1.8.6)**

- Download zip file of LabelImg from GitHub: https://github.com/HumanSignal/labelImg
- Open anaconda prompt and type the following command to install and run the software:
    - conda install pyqt=5
    - conda install -c anaconda lxml
    - pyrcc5 -o libs/resources.py resources.qrc
    - python labelImg.py
    - python labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]

5.2 Hardware Setup

To enable application debugging on physical devices, some settings need to be applied to the device, the settings may be different based on the device type, in this project:

- Go to Setting and tap MIUI version for 7-10 times to enable the Developer options on the device.
- After that, go to Additional Settings > Developer options and turn on “USB debugging” then the device is ready to install and show return debug message to Visual Studio code.
- To enhance the debugging experience, wireless debugging can also be activated. This involves connecting the device and computer to the same network and using the terminal in Visual Studio Code to run the command: adb connect [IP address]:[port number].

5.2.1 Emulator Settings

The emulator used in this project is a Pixel 8 running Android 13.0 with API 33.

Figure 5.1 Android Emulator

5.3 Settings and Configuration

In the android/app/build.gradle file, the compileSdkVersion must be set to at least 35. This is necessary to support the latest Android APIs and features used within the application.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <application
```

Figure 5.2 Configuration in AndroidManifest.xml

In the AndroidManifest.xml file located in the “android/app/src/main” directory, the permissions as shown in the Figure 5.2 is added.

- RECORD_AUDIO permission is for enabling speech recognition feature which allow the app to capture input from the device’s microphone.
- VIBRATE permission is to enable the app to provide vibration feedback.

By adding these permissions, the application can request the necessary access from the user at runtime.

5.4 YOLOv8 model implementation

In this section, we describe the implementation of YOLOv8 model, the implementation process required a range of steps, from data acquisition and preprocessing to model training, evaluation and integration.

5.4.1 Data Acquisition

Data acquisition is one of the most important stages for training the model. Due to the limitations of available datasets on Malaysian currency, the data acquisition will be capturing the currency images manually to create a custom dataset. There are ten distinct currency classes, including six banknote denominations (RM1, RM5, RM10, RM20, RM50 and RM100), four coin classes (5 cent, 10 cent, 20 cent, and 50 cent). The images were captured using an iPhone 13 Pro Max, which was selected due to its advanced camera system capable of producing high-quality images with visible detail, which important for detecting the tiny features of the banknotes and coins accurately.

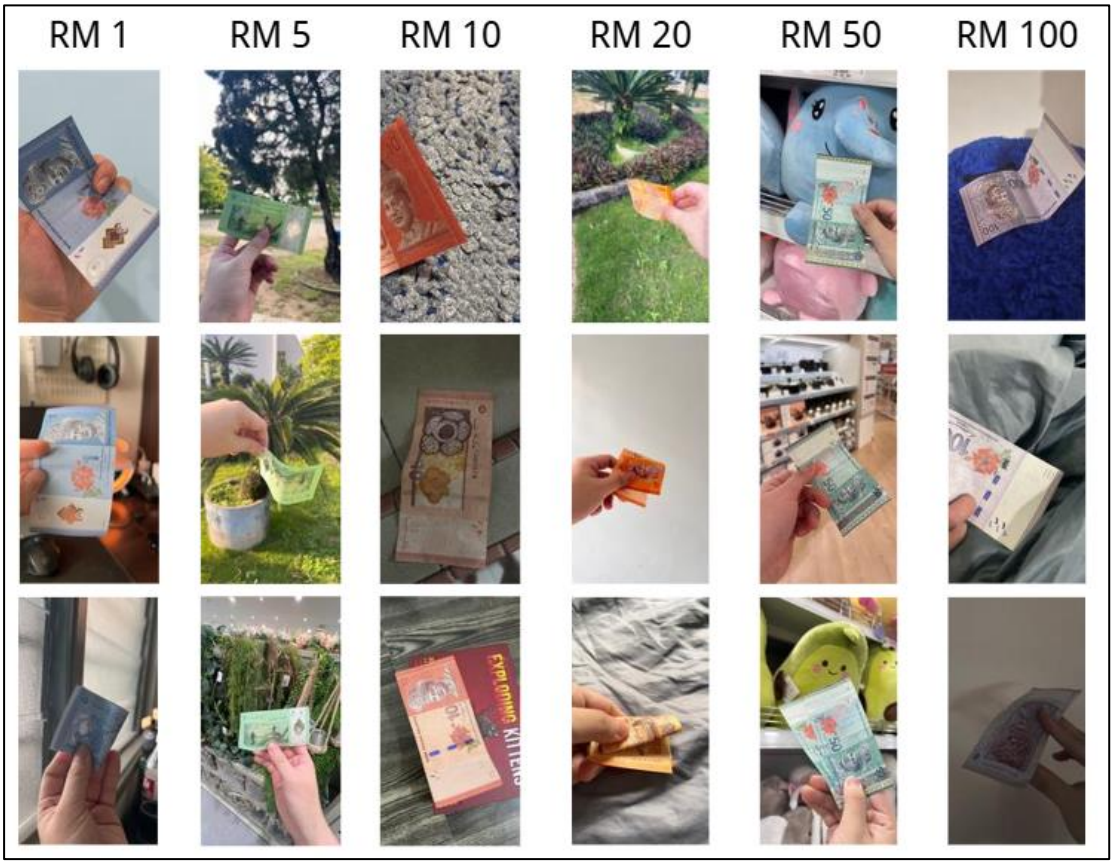


Figure 5.3 Example images for banknote classes

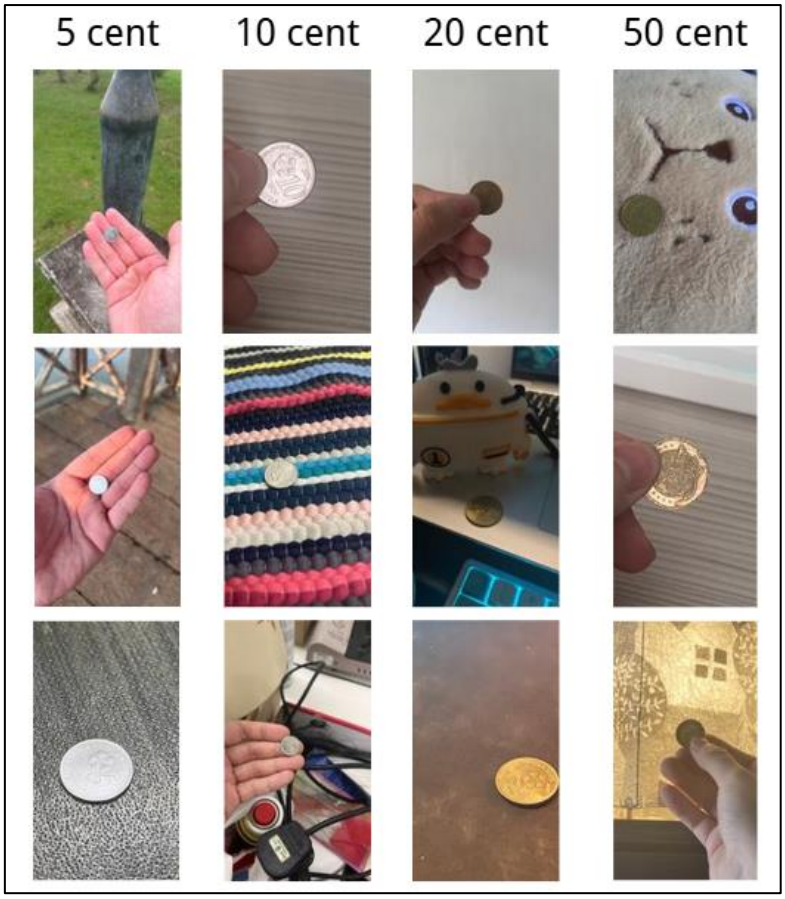


Figure 5.4 Example images for coin classes

To make sure the dataset reflected realistic conditions under which the app would perform, the images were captured from various angles to simulate the different orientations users might have when holding the currency. Additionally, different lighting conditions were simulated too such as bright, natural light, and dark indoor light, which commonly occurs in real-world environments. On top of that, to further increase the dataset's robustness, some images included partial occlusion, where parts of the banknotes or coins were hidden or blocked for example people holding a corner of the money, to make sure that the model could still detect and classify the visible parts accurately. Furthermore, some images captured contain multiple labels to help the model to differentiate between the classes more effectively. In total, 2,168 images were captured with at least 200 images for each class to ensure there is enough training data for model training.

Table 5.1 Number of images captured for each class

Class	Number of images captured
5 cents	203
10 cents	201
20 cents	200
50 cents	200
RM 1	221
RM 5	209
RM 10	202
RM 20	204
RM 50	201
RM 100	242

Table 5.2 Number of images captured for mixed currencies

Mixed currencies (multi label)	Number of images
	85

5.4.2 Data Preprocessing and Augmentation

Once the raw images were captured, the next step is preprocessing the data to make sure it was the correct format for training the YOLOv8 model. Since YOLOv8 required images in the JPEG format, the original images captured in (HEIC) format were converted to JPG to maintain compatibility with the training pipeline. The preprocessing begins by annotating the images and was carried out using the LabelImg software.

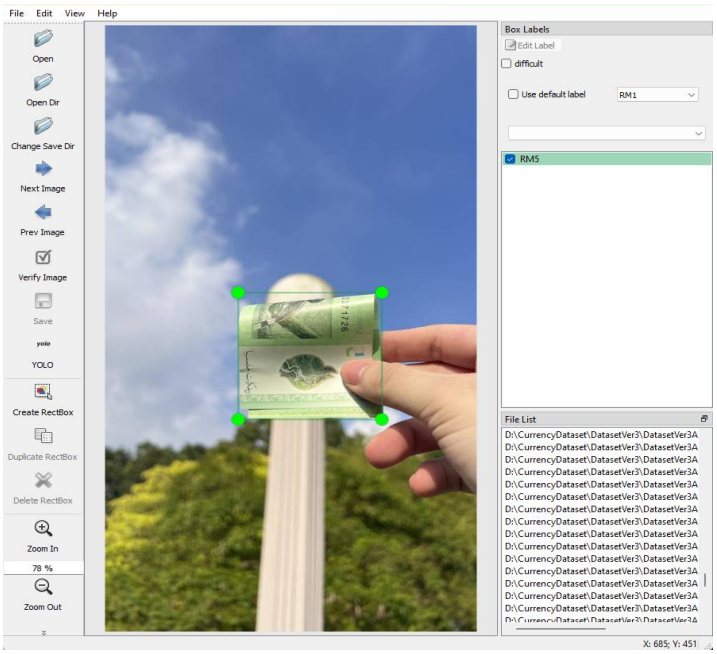


Figure 5.5 Data labelling in LabelImg

Figure 5.5 illustrates the annotation process within LabelImg. In this process, each image was labelled carefully with bounding boxes draw around the target object. These bounding boxes should be marking precisely and tight to the object to help YOLOv8 model to learn the characteristics of each class. A class label was assigned to every annotated object which representing the denomination of the banknote or coin to differentiate between objects.

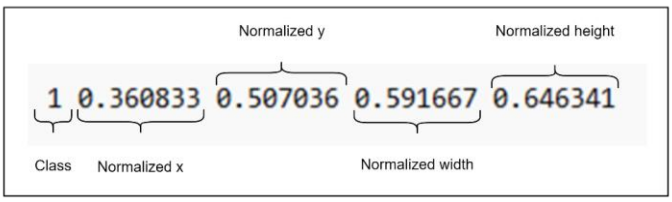


Figure 5.6 YOLOv8 format label text file

After the labelling process, the annotations were automatically exported into YOLOv8-compatible format in a text file for each saved image annotation. As shown in Figure 5.6, the text file format consists of five components: class id, normalized x-coordinate, normalized y-coordinate, normalized width, and the normalized height of the bounding box. The use of normalized values ensures that the annotations are resolution-independent, allowing the model to process images of different sizes without losing the accurate position.

Following the annotation step, the dataset was then randomly split into three subsets which are training, validation, and testing. These subsets were organized into separate folders named train, valid, and test with a ratio of 7:2:1. The training set that compromised 70% of the dataset was used to train the model, while the validation set (20%) was used to fine-tune the model and prevent overfitting. The testing set (10%) was reserved for the final evaluation of the model's performance on unseen data.

```
# Add RandomBrightnessContrast
if random.random() > 0.5:
    augmentations.append(A.RandomBrightnessContrast(brightness_limit=random.uniform(0.1, 0.3),
                                                    contrast_limit=random.uniform(0.1, 0.3), p=1.0))

# Add RandomGamma
if random.random() > 0.5:
    augmentations.append(A.RandomGamma(gamma_limit=(random.randint(90, 150), random.randint(120, 180)), p=1.0))

# Add GaussianBlur
if random.random() > 0.5:
    augmentations.append(A.GaussianBlur(blur_limit=(3, random.choice([3, 5, 7])), p=1.0))

# Add Flip
if random.random() > 0.5:
    augmentations.append(A.HorizontalFlip(p=1.0))  # Flip horizontally

# Add Random Resized Crop
if random.random() > 0.5:
    augmentations.append(A.RandomResizedCrop(height=416, width=416,
                                             scale=(random.uniform(0.9, 1.0), 1.0), p=1.0, interpolation=cv2.INTER_NEAREST))

# Ensure there's always at least one transformation
if len(augmentations) == 0:
    augmentations.append(A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=1.0))
```

Figure 5.7 Data Augmentation parameters

After completing the splitting process, data augmentation was performed to improve the dataset diversification. Data augmentation is an important process to generate additional variations of the images by applying random transformations. In this project, each image was augmented to create two new variations, and these variations were generated using a random combination of transformations such as brightness, Gaussian blur, gamma, horizontal flip, and random sized crop. With these

augmentations, some real-world conditions can be simulated such as poor lighting, motion blur, different orientations, and different distances to let the model learn to generalize better and maintain the performance under different scenarios. As a result of data augmentation, the images on the training set increased from 1,517 images to 4,551 images (1,517 original + (2 variations per image = 3034)). Figure 5.8 shows examples of two variation of augmented data from the first original image.

Figure 5.8 Example variation of augmented data

```
path: /content/Dataset/MyDrive/Ver6Augment/extracted_data/AugmentedDataset
train: train/images
val: valid/images
test: test/images

nc: 10
names: ['RM1', 'RM5', 'RM10', 'RM20', 'RM100', 'RM50', '5 cent', '10 cent', '20 cent', '50 cent']
```

Figure 5.9 Dataset .yaml file

Then, a YAML (Yet Another Markup Language) configuration file as shown in Figure 5.9 is then created to define the dataset structure for the YOLOv8 model. This file specified the information about the dataset, including the paths to the training, validation, and testing sets in Google Drive, as well as the total number of classes and their corresponding labels. The structure of the file ensured that the dataset could be loaded and interpreted correctly during the training process.

Finally, the entire processed dataset together with the .yaml file was uploaded to Google Drive to be mounted for model training in Google Colab. By mounting the Google Drive in the Colab environment, the dataset was directly accessible during the training process and remained persistent, unlike the session storage in Google Colab which is temporary and will gets cleared when the session ends.

5.4.3 Model Training

The model training started by loading the YOLOv8 pre-trained model, in this case (YOLOv8n.pt) is selected for its lightweight architecture and efficiency in real-time detection, making it suitable for identifying Malaysian currency. Transfer learning was applied to let the model learn about detecting on the Malaysian currency. Transfer learning is important for this task because it allows the model to utilize its pre-existing knowledge of object detection while learning to recognize the unique features of Malaysian banknotes and coins. The training process involved fine-tuning the model with a custom dataset of labelled currency images. The parameters for the training process are listed in table below:

Table 5.3 Model training parameters

Parameter	Value
imgsize	640
Batch	16
epochs	50
lr0	0.01

imgsize (Image Size) - 640: This parameter defines the resolution of the input images feed into the model during training. It is selected as 640 pixels because we want to make sure a balance between computational efficiency and sufficient detail for detecting small features on Malaysian currency because even higher imgsize can improve detection accuracy, but it may increase the memory consumption and training time.

Batch - 16: The batch size specifies the number of images processed in each iteration of training, it was set to 16 to provide a good balance between memory usage and the stability of gradient updates during backpropagation because larger batch size might speed up training but may require more GPU memory while smaller sizes might result in noisier gradients.

Epochs - 50: The number of epochs defines how many times the entire dataset will be passed through during model training. In this case, the model is trained for 50 epochs to allow it to improve the ability to detect and classify currency gradually.

lr0 (Learning rate) – 0.01: The learning rate defines the step size for updating the model's weight during each iteration of training. A value of 0.01 is selected to make sure that the model can learn in a moderate speed without premature converging. A higher learning rate might cause model to converge faster but makes the training unstable, while a lower learning rate make sure the stability but will slow down the training process.

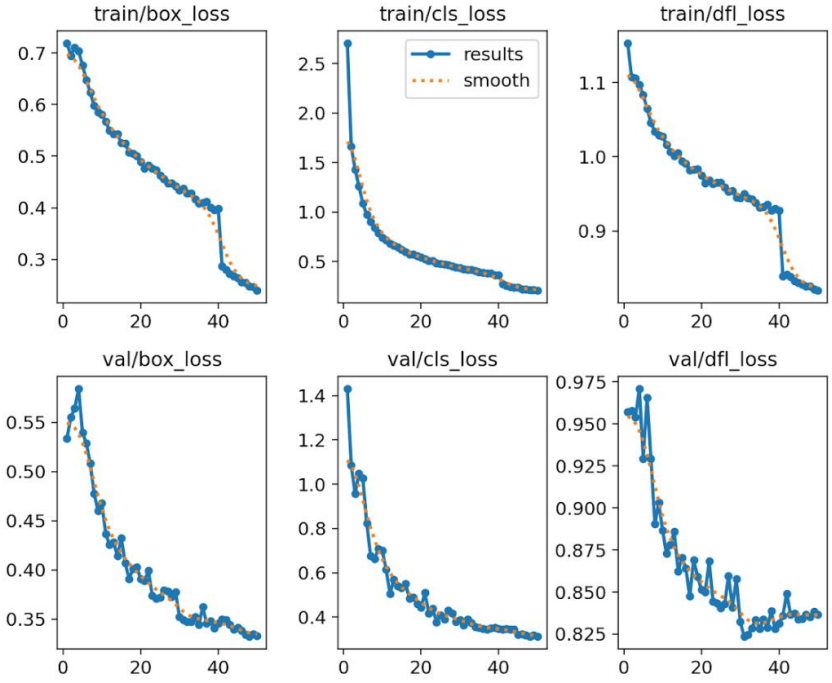


Figure 5.10 Training result

The whole training process took around 2 hours with the use of Google Colab's T4 GPU to complete, Figure 5.10 shows the training and validation results of the model over 50 epochs, there are a few observations can be made:

Loss Function: The top row which are the train/box_loss, train/cls_loss, and train/dfl_loss, decrease steadily throughout the training which indicated that the model becomes better on minimizing errors in bounding box regression, classification, and distribution focal loss (DFL). Besides, the bottom row val/box_loss, val/cls_loss, and val/dfl_loss also shows a smooth reduction which shows that the model can generalize well on the validation dataset too. Furthermore, it is observed that there is a huge reduction in the last 10 epochs for training dataset, this is because of Ultralytic's automatic disabling of mosaic augmentation to fine-tune the performance of model to let it more focus on precise detection rather than generalized learning.

5.4.4 Model Fine-tuning

Upon evaluating the initial training results, it was found that the model underperformed in detecting the coin classes.

Figure 5.11 CLAHE and exposure augmentation on additional coin dataset

To address this, the dataset was expanded with additional images of coins. Due to the characteristics of coins that can reflect the light under direct lighting or sunlight, therefore augmentation techniques such as CLAHE (Contrast Limited Adaptive Histogram Equalization) and exposure adjustment as shown in Figure 5.11 were applied specifically to the coin classes to improve detection under different lighting conditions.

CLAHE: Improves local contrast in images and enhances edge definitions by redistributing light intensity more evenly which makes the coin feature such as inscriptions, edges, and textures more visible in poor light or uneven lighting environments.

Exposure Adjustment: Simulates variations in brightness levels by artificially increasing or decreasing the image exposure. This helps the model generalize better to conditions ranging from low indoor lighting to bright outdoor sunlight, where the details on the coin might not visible.

As a result of the expansion, 800 images of coins were added with 200 images per coin class and split into train, valid, and test set with ratio of 7:2:1.

Then model was further fine-tuned for an additional 40 epochs using the enhanced dataset. This approach is known as continual training where the training resumes from a previously trained model rather than starting from scratch. This allows the model to retain previously learned features such as banknotes while further introducing new knowledge from the updated dataset. The parameters for model fine-tuning are as below:

Table 5.4 Model fine-tuning parameters

Parameter	Value
imgsize	640
Batch	16
epochs	40
lr0	0.01
patience	10

To prevent the model from overfitting during this phase, early stopping was used with a patience value of 10, meaning that the training would stop if there were no improvement shown in the validation loss for 10 epochs.

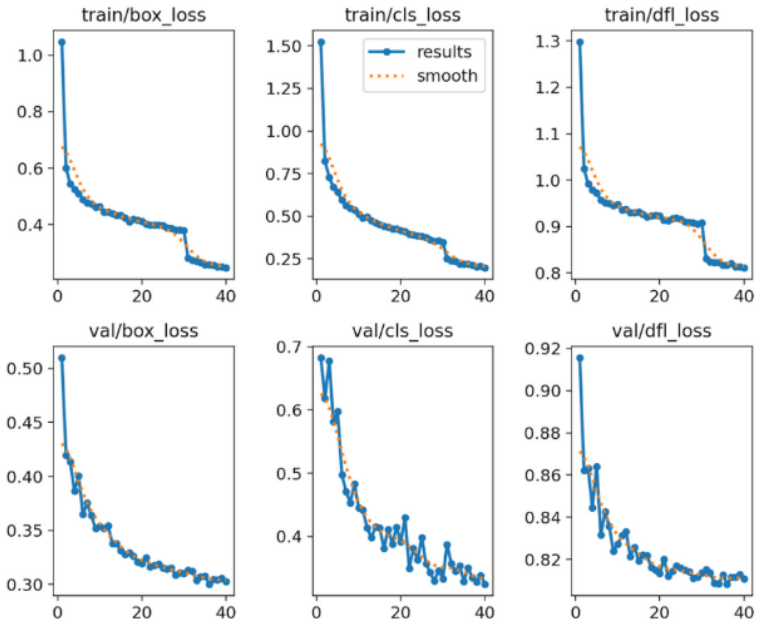


Figure 5.12 Training result for fine-tuning

The Figure 5.12 shows the training and validation results during the fine-tuning phases. From the graph, it can be observed that all loss functions continued to decrease, indicating that the model was still learning effectively. This result also suggests that the additional and augmented coin data enabled the model to better generalize under varying visual conditions. In terms of overfitting issue, the gradual and consistent loss reduction shows that the model did not overfit but further improved from the enhanced dataset and exposure-based augmentation.

5.4.5 Model Exportation and Conversion

Once the evaluation was done, the model was converted into TensorFlow lite format pending to be integrated into the mobile app. During the conversion of the model, there will be two versions of the model which are best_float32.tflite and best_float16.tflite. These versions are mainly different in terms of resource efficiency and accuracy. The best_float32.tflite model can provide maximum accuracy but may lack optimized performance and resource efficiency. For this project, the best_float32.tflite model was selected to integrate into the app to focus on the accuracy of currency detection.

```
[ ]  model.export(format='tflite')
```

```
TensorFlow Lite: starting export with tensorflow 2.17.1...
TensorFlow Lite: export success ✅ 0.0s, saved as 'runs/detect/train2/weights/best_saved_model/best_float32.tflite' (11.7 MB)

Export complete (54.1s)
Results saved to /content/Dataset/MyDrive/Ver6Augment/runs/detect/train2/weights
Predict:         yolo predict task=detect model=runs/detect/train2/weights/best_saved_model/best_float32.tflite imgsz=640
Validate:        yolo val task=detect model=runs/detect/train2/weights/best_saved_model/best_float32.tflite imgsz=640 data=/cont
Visualize:       https://netron.app
'runs/detect/train2/weights/best_saved_model/best_float32.tflite'
```

Figure 5.13 TFlite conversion

5.5 System Operation

5.5.1 Model Integration

```
flutter:
  uses-material-design: true

  assets:
    - assets/best_float32.tflite
    - assets/classes.txt
```

Figure 5.14 Model assets in pubspec.yaml

To integrate the exported YOLOv8 model into the Flutter app, the TensorFlow lite format of the model together with a text file that contains the class names were copied into the asset directory of the Flutter application directory. After that, the pubspec.yaml (file that store dependencies in Flutter) of the application was updated to include the paths to these files, making sure that they are accessible to the app during runtime as shown in Figure 5.14.

```
Future<void> loadYoloModel() async {
  try {
    await vision.loadYoloModel(
      labels: 'assets/classes.txt',
      modelPath: 'assets/model.tflite',
      modelVersion: 'yolov8',
      numThreads: 3,
      useGpu: true,
    );
```

Figure 5.15 Function to load model

After updating the pubspec.yaml, a loadYoloModel function was implemented in the application to load the YOLOv8 model as shown in Figure 5.15. This function is responsible for initializing the YOLOv8 model using the Flutter Vision plugin, configuring the model version, and setting the number of threads for inference. By using more than one thread, the function can optimize the resource usage to ensure the performance of the model and system load for smooth operation during detection.

Figure 5.16 Home screen user interface with bounding box

To test the model's prediction accuracy. A bounding box function was integrated into the app's user interface. This feature visually displays the detected banknotes by outlining them with bounding boxes based on the model's prediction coordinates. These bounding boxes can help to verify whether the model correctly identifies the actual currency being detected. The bounding box feature is an important tool for testing and debugging, allowing us to validate the model's performance in real time and identify potential inaccuracies or misclassifications.

```
Future<void> yoloOnFrame(CameraImage image) async {
  if (!mounted || !_isDetecting) return;

  _frameCount++;
  if (_frameCount % PROCESS_EVERY_N_FRAMES != 0) return;
```

Figure 5.17 Frame skipping on model prediction

Continuous model prediction on every video frame can significantly affect the device's performance which will lead to increased computational load, reduced battery

life, and overheating. To overcome this, a frame-skipping mechanism was implemented. Instead of processing predictions on every frame, the app processes predictions once every five frames which is an acceptable interval for model processing as shown in Figure 5.17. This optimization reduces the frequency of model inference making the overall user interface able to process smoothly. The implementation of frame skipping also helps to reduce latency, maintaining a smooth user experience while ensuring that the app can provide accurate feedback during detection.

5.5.2 Counting Technique

To add up the total amount of money scanned by the user, a custom counting technique was implemented in the application. This technique is specifically designed to address the challenges associated with real-time object detection results when using the YOLOv8 model on mobile devices.

The YOLOv8 model will continuously return detection results for every frame that perform detection, which causes the same money to be detected repeatedly which is overcounting. Without an appropriate counting strategy, the value of the same money would be added multiple times which leads to incorrect total amounts. Therefore, there is a need for an efficient and mobile-friendly object tracking and counting mechanism to ensure that each banknote or coin is counted only once, even if it remains visible across multiple frames.

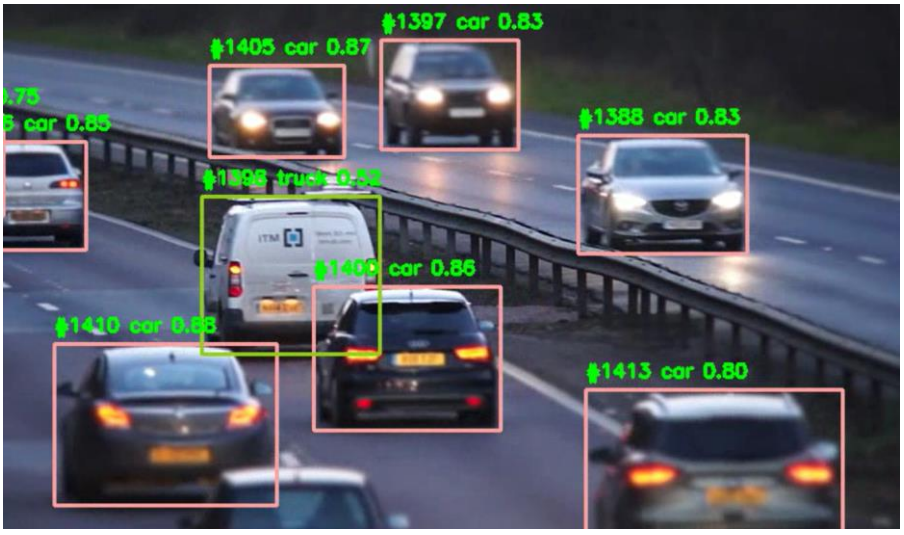


Figure 5.18 ByteTrack on vehicle tracking [21]

On desktop environments, object tracking algorithms such as ByteTrack and DeepSORT are commonly used to assign unique IDs to the detected objects based on their intersection-over-union (IoU) and motion between frames as shown in Figure 5.18. These algorithms allow for robust object tracking by distinguishing between new and existing objects. However, these solutions often need higher computational resources and are not available for mobile devices making them unsuitable for this project which focuses on accessibility and mobile devices usage in real time.

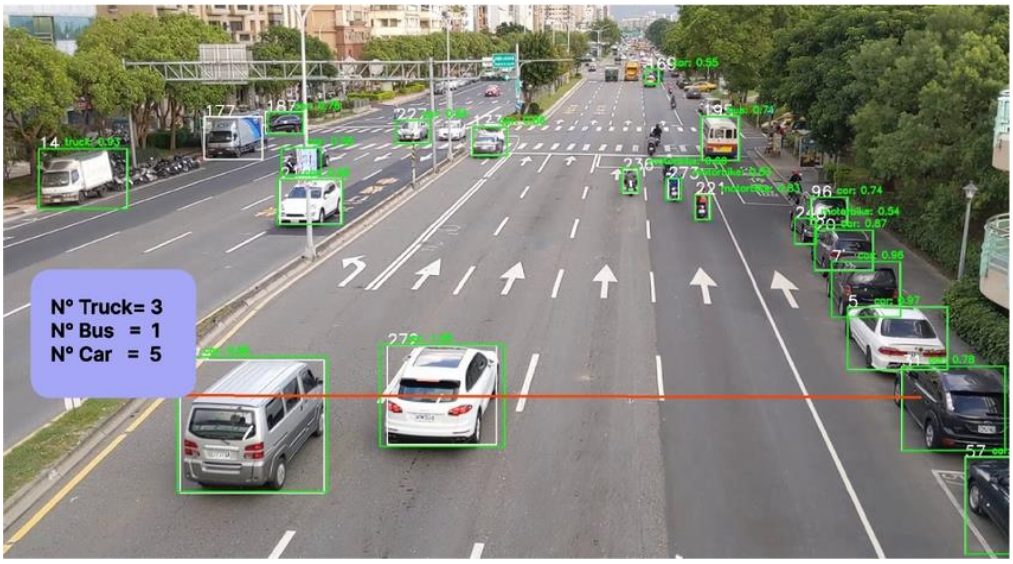


Figure 5.19 YOLOv8 counting technique [22]

While YOLOv8 also offers a built-in counting method, it relies on a virtual line counting approach as shown in Figure 5.19, where an object is counted only when it crosses a predefined line on the screen. This method is not applicable to the requirements of this project, as banknotes and coins in this application can appear anywhere in the camera frame, and visually impaired user might also not know the current placement of the money. Furthermore, YOLOv8 does not provide tracking itself, it requires integration with external algorithms such as ByteTrack for object tracking which as mentioned earlier, is not feasible to mobile-based real-time applications.

Therefore, to overcome these challenges, a lightweight object tracking and counting technique that does not rely on computationally intensive tracking libraries was implemented, each detected banknote is assigned a unique identifier based on its position and confidence score. This approach enables the application to distinguish

between newly detected money and previously detected ones, even if minor movements such as slight shifts, rotations, or camera shakes occur.

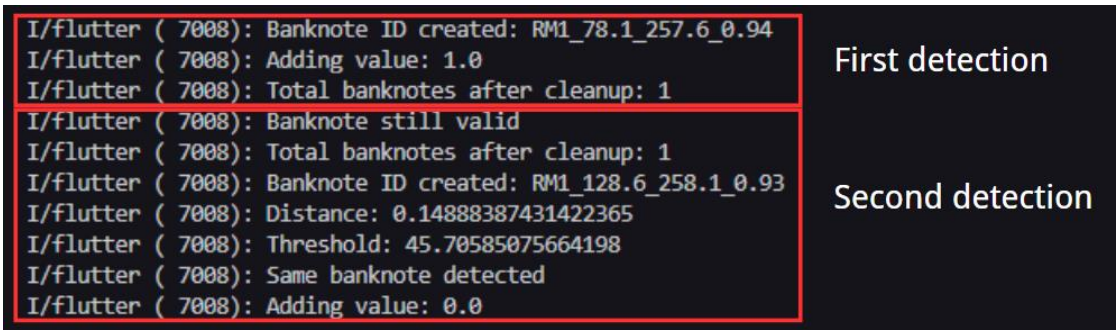


Figure 5.20 Money Counting output for same banknote

Initially, when a detection result is received from the YOLOv8 model, the application will generate a unique ID by combining the detection class (denomination), the x-y position of the bounding box center and bounding box size.

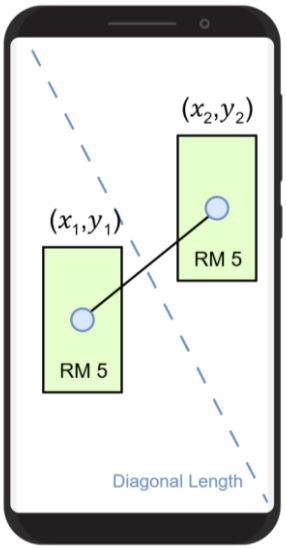


Figure 5.21 Euclidean Distance application on device

To further improve robustness, the system compares new detections against previously processed banknotes using a normalized distance calculation based on the screen size and computes the Euclidean distance between the center points of the current detection and previously detected money as shown in Figure 5.21. The Euclidean distance between two points (x_1, y_1) and (x_2, y_2) is computed as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This method is used because it can measure the direct spatial distance between two points effectively, providing a reliable way to determine proximity regardless of minor positional changes. If the class labels match and the distance between two detections is below a predefined threshold which is 5% of the screen's diagonal length, it is considered the same object and is not counted again as shown in Figure 5.22. This prevents repeated counting and ensures that the total amount reflects only truly new banknotes that appeared on the camera view.

```
I/flutter ( 7008): Total banknotes after cleanup: 1
I/flutter ( 7008): Banknote id: RM1_78.1_257.6_0.94 expired, removing...
I/flutter ( 7008): Total banknotes after cleanup: 0
I/flutter ( 7008): Banknote ID created: RM1_172.8_238.0_0.94
I/flutter ( 7008): Adding value: 1.0
```

Figure 5.22 Money Counting output for "expired" banknote

To further enhance the robustness of the algorithm, a timeout mechanism has been implemented where each detected result is assigned with a timestamp when it is first identified. If a banknote is not seen again within a certain duration (3 seconds in this project), it will be considered as "expired" and removed from the list of tracked banknotes as shown in Figure 5.22. This ensures that if the user removes the money from the camera view or introduces another money, the system still remains responsive and keep updating without stuck on the old detections. When new money is confirmed, the system will then add the count for the corresponding denomination on the map, the total value of the scanned money will also being updated too. Through this approach, the algorithm is able to maintain the performance of the system without the need for heavy computational resources for money counting. Figure 5.23 and Figure 5.24 below show some partial code of how the counting technique was implemented.

```
// Update last seen timestamps and track new banknotes
Set<String> countedThisFrame = {};
double newValue = 0.0;
Map<String, int> newCounts = Map.from(denominationCounts);

// Process new detections
newDetections.forEach((id, detection) {
  String tag = detection['tag'];

  // Check if this is a new banknote or a previously seen one that has timed out
  if (!_processedBanknotes.containsKey(id)) {
    bool isNewBanknote = true;

    // Check if this might be a slightly moved version of an existing banknote
    _processedBanknotes.forEach((existingId, existingData) {
      if (_isSameBanknote(context, detection, existingData)) {
        print("Same banknote detected");
        isNewBanknote = false;
        _lastSeenBanknotes[existingId] = currentTime;
      }
    });

    if (isNewBanknote && !countedThisFrame.contains(tag)) {
      // Count new banknote
      newCounts[tag] = (newCounts[tag] ?? 0) + 1;
      newValue += _moneyDenomination.values[tag] ?? 0;
      countedThisFrame.add(tag);
      _processedBanknotes[id] = detection;
      _lastSeenBanknotes[id] = currentTime;
    }
  } else {
    // Update timestamp for existing banknote
    _lastSeenBanknotes[id] = currentTime;
  }
});
```

Figure 5.23 Code for updating detection result

```
String _createBanknoteId(Map<String, dynamic> detection) {
  // Create a unique ID based on position and confidence
  List<double> box = detection['box'];
  return '${detection['tag']}_${box[0].toStringAsFixed(1)}_${box[1].toStringAsFixed(1)}_${box[4].toStringAsFixed(2)}';
}

bool _isSameBanknote(BuildContext context, Map<String, dynamic> detection1, Map<String, dynamic> detection2) {
  List<double> pos1 = detection1['position'];
  List<double> pos2 = detection2['position'];

  double center1X = (pos1[0] + pos1[2]) / 2;
  double center1Y = (pos1[1] + pos1[3]) / 2;
  double center2X = (pos2[0] + pos2[2]) / 2;
  double center2Y = (pos2[1] + pos2[3]) / 2;

  double screenWidth = MediaQuery.of(context).size.width;
  double screenHeight = MediaQuery.of(context).size.height;

  double normalizedX1 = center1X / screenWidth;
  double normalizedY1 = center1Y / screenHeight;
  double normalizedX2 = center2X / screenWidth;
  double normalizedY2 = center2Y / screenHeight;

  double distance = _calculateDistance(normalizedX1, normalizedY1, normalizedX2, normalizedY2);

  // 5% diagonal length
  double threshold = 0.05 * sqrt(pow(screenWidth, 2) + pow(screenHeight, 2));

  return distance < threshold && detection1['tag'] == detection2['tag'];
}

double _calculateDistance(double x1, double y1, double x2, double y2) {
  return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}
```

Figure 5.24 Code for ID creation, same money detection and Euclidean distance calculation

5.5.3 Localization

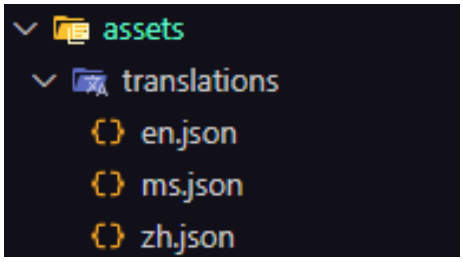


Figure 5.25 JSON files of languages

In order to accommodate users from different linguistic backgrounds, the application allows users to switch the interface language according to their preferences. This localization functionality is implemented using Flutter's easy_localization plugin, which simplifies the management and integration of multiple languages within the application. The first step in implementing localization involves preparing translation files for each supported language. JSON files were created for English, Chinese, and Malay, and stored in the assets/translation's directory as shown in Figure 5.25.

Figure 5.26 Key-value pair in language JSON files

Each JSON file follows a key-value structure, Figure 5.26 shows the partial content in each JSON files:

- The key represents the term used throughout the application.
- The value represents the translation of the key in the respective language.

After creating the translation files, additional setup is required to enable Flutter to recognize and utilize these resources. Specifically, two types of files need to be generated which are **codegen_loader.g.dart** and **locale_keys.g.dart**.

Codegen_loader.g.dart:

The first step involves generating a loader file that reads the translation JSON files into the application. The following command is executed:

flutter pub run easy_localization:generate -S "assets/translations" -O "lib/translations"

- -S "assets/translations" specifies the source directory containing the translation files.
- -O "lib/translations" specifies the output directory where the generated Dart file will be placed.

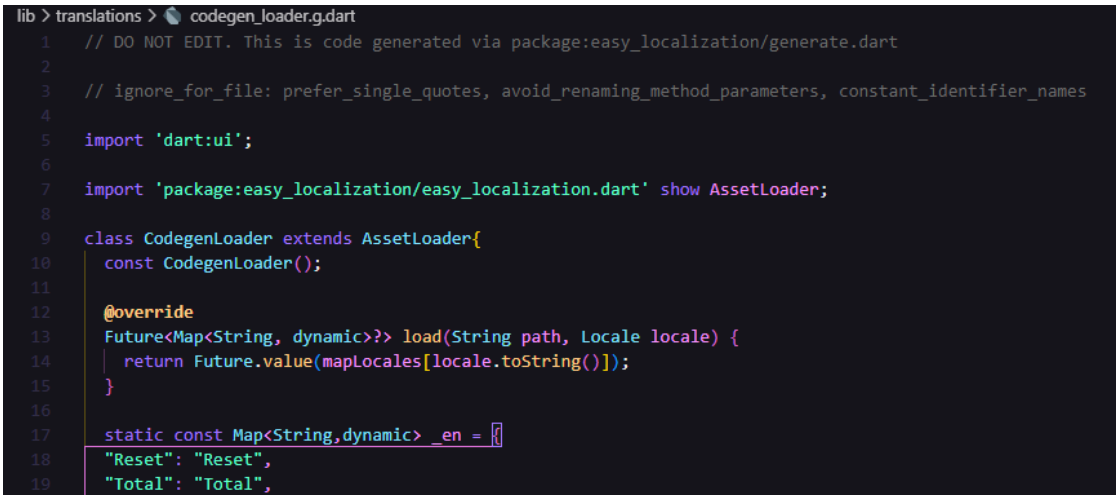


Figure 5.27 Codegen_loader.dart

After executing the command, a codegen_loader.g.dart file as shown in Figure 5.27 is created inside the lib/translation's directory. This file automatically loads and parses the JSON files at runtime, making them available for use within the application.

Locale_keys.g.dart:

The next step involves generating a Dart file that contains all translation keys as constants. This approach improves code maintainability and reduces potential runtime errors due to typos in the keys. The following command is used:

flutter run easy_localization:generate -S "assets/translations" -O "lib/translations" -o "locale_keys.g.dart" -f keys

- -o "locale_keys.g.dart" specifies the name of the output file for locale keys.
- -f keys instructs the generator to create keys instead of a loader.

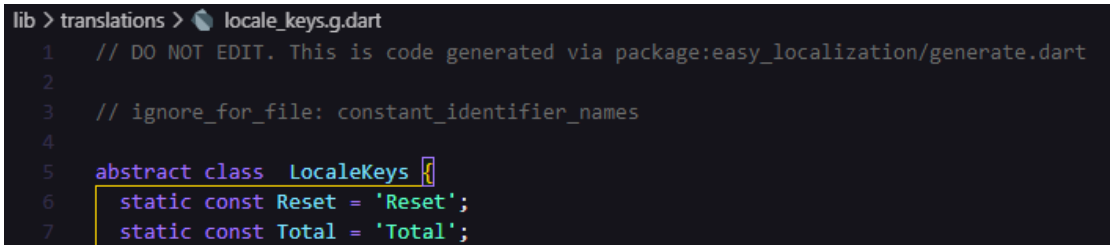


Figure 5.28 Locale_keys.g.dart

The generated locale_keys.g.dart file as shown in Figure 5.28 provides a class where each translation key is represented as a constant, making it easier and safer to access localized strings programmatically such as “Text(LocaleKeys.setting.tr())” instead of hardcoding "setting" as a string, this method ensures better compile-time checking.

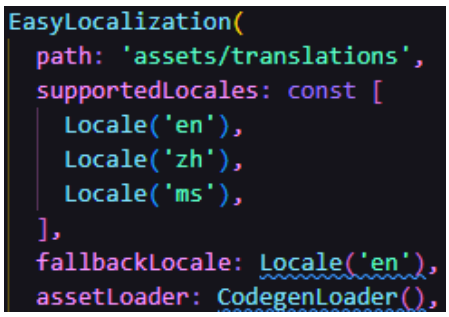


Figure 5.29 Initialization of Easy Localization

Once the generated files are ready, it is necessary to initialize localization during the application startup. In the main.dart file, the EasyLocalization widget is used to wrap the application, specifying the supported locales and the path to the translation files as shown in Figure 5.29.

5.5.4 Setting Screen

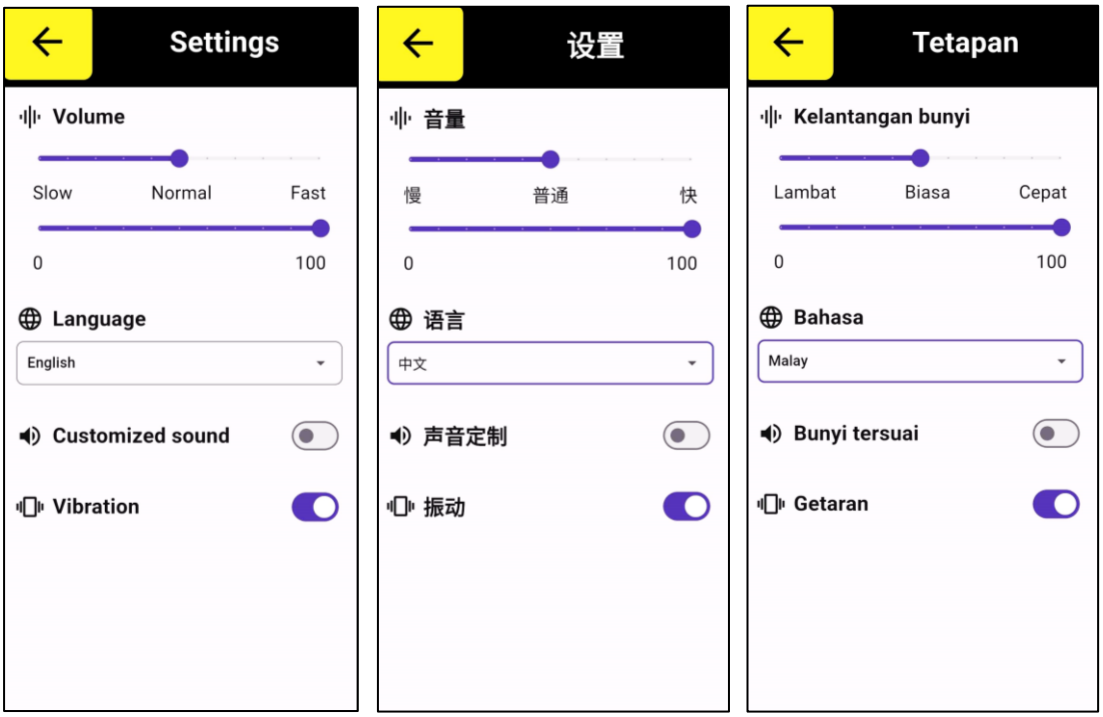


Figure 5.30 Setting Screen in different languages

Figure 5.30 shows the setting screen displayed in different languages. The setting screen is designed to allow the user to customize their experience based on preferences. Any changes made by the user in the Setting Screen are immediately written to the device's internal storage by shared preferences. Once there is a change in the setting, the application will automatically update its user interface to reflect the new configurations. For example, when the user selects a new language, the entire application's interface is updated in real time to display text in the newly selected language without the need to restart the application.

FlutterSharedPreferences.xml

```
1  <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
2  <map>
3      <string name="flutter.volume">
       VGhpcyBpcyB0aGUgcHJlZml4IGZvciBEb3VibGUu1.0</string>
4      <string name="flutter.classSounds">{&amp;quot;RM 100&amp;quot;:&amp;quot;Sound 1
       &amp;quot;,&amp;quot;RM 1&amp;quot;:&amp;quot;Sound 1&amp;quot;,&amp;quot;RM 10&amp;quot;:&amp;quot;None
       &amp;quot;,&amp;quot;50 sen&amp;quot;:&amp;quot;Sound 3&amp;quot;}</string>
5      <string name="flutter.locale">en</string>
6      <boolean name="flutter.enableCustomSound" value="false" />
7      <string name="flutter.language">English</string>
8      <string name="flutter.voiceRate">
       VGhpcyBpcyB0aGUgcHJlZml4IGZvciBEb3VibGUu0.5</string>
9      <boolean name="flutter.enableVibration" value="true" />
10 </map>
```

Figure 5.31 FlutterSharedPreferences.xml

Figure 5.31 shows the shared preferences xml file retrieved from the device's internal storage. This file is responsible for storing the application's settings persistently. From the figure, the settings are stored as key-value pairs which is a simple form so that each setting can be retrieved quickly when the application is launched.

5.5.5 Voice Command

The voice command functionality is integrated into the application using the speech_to_text plugin. This feature is implemented to enhance the user experience for visually impaired individuals by allowing them to interact with the app using their voice instead of touch-based navigation. Through voice commands, users can use the applications more conveniently and independently. For example, users can say "change to Chinese", the application will capture the keyword "Chinese" and switch the language setting to Chinese automatically. Similarly, users can ask, "How many 1 ringgit", and the application will respond with the count of RM 1 notes detected during the current session via audio feedback, making the application more accessible.

```
void startListening(Function(String) onResultCallback) {
  if (!_speech.isListening && _speech.isAvailable) {
    _audioPlayerService.playStt();
    _speech.listen(
      localeId: _currentLocaleId,
      onResult: (SpeechRecognitionResult result) {
        if (result.finalResult) {
          onResultCallback(result.recognizedWords.toLowerCase());
        }
      },
      listenFor: const Duration(seconds: 10),
      pauseFor: const Duration(seconds: 3),
      partialResults: true,
      cancelOnError: false,
      listenMode: stt.ListenMode.dictation,
    );
    _isListening = true;
  }
}
```

Figure 5.32 Code listening for speech

Figure 5.32 displays the startListening function which activates the speech recognition service. When this function is being called, an audio will be played to notify users that the system is now actively listening for voice input. The locale of the speech recognizer is set based on the current language selection to ensure the speech

recognition engine listens for and correctly interprets the expected language. In terms of the listening duration, the speech recognizer is set to capture the input for a maximum duration of 10 seconds, providing sufficient time for users to complete their speech, and it will stop listening when no speech is captured for 3 seconds to ensure the listening just on time without wasting device resources. Furthermore, the speech recognizer is configured to continue listening even if errors occur so that when the selected language does not match the language spoken by user, it still can be activated and remain smooth user experience. The listen mode is set to "dictation" to optimize the system for recognizing continuous natural speech input to avoid fail to capture certain keywords.

Once the speech is recognized and finalized, the recognized words are converted to lowercase before being passed back to the functions that call the service because the commands are case-sensitive, setting it in lowercase and avoiding the errors due to inconsistent capitalization.

```
child: GestureDetector(
  onLongPress: () {
    widget.speechToTextService.startListening((String recognizedText) {
      _handleVoiceCommand(recognizedText);
    });
  },
  onDoubleTap: () {
    widget.speechToTextService.startListening((String recognizedText) {
      _handleVoiceCommand(recognizedText);
    });
  },
```

Figure 5.33 Code Gesture Detector to activate voice command

In addition, the gesture detector is implemented to trigger the voice commands. Figure 5.33 illustrates how a GestureDetector widget is used across all screens of the application. By keeping the speech recognition service running continuously in the background, it might consume a lot of device's resources which will reduce application's performance and affect battery health. Therefore, the voice command has been set to activate only when needed through user gestures such as long press or double tap on the screen.

5.5.6 Tutorial Screen

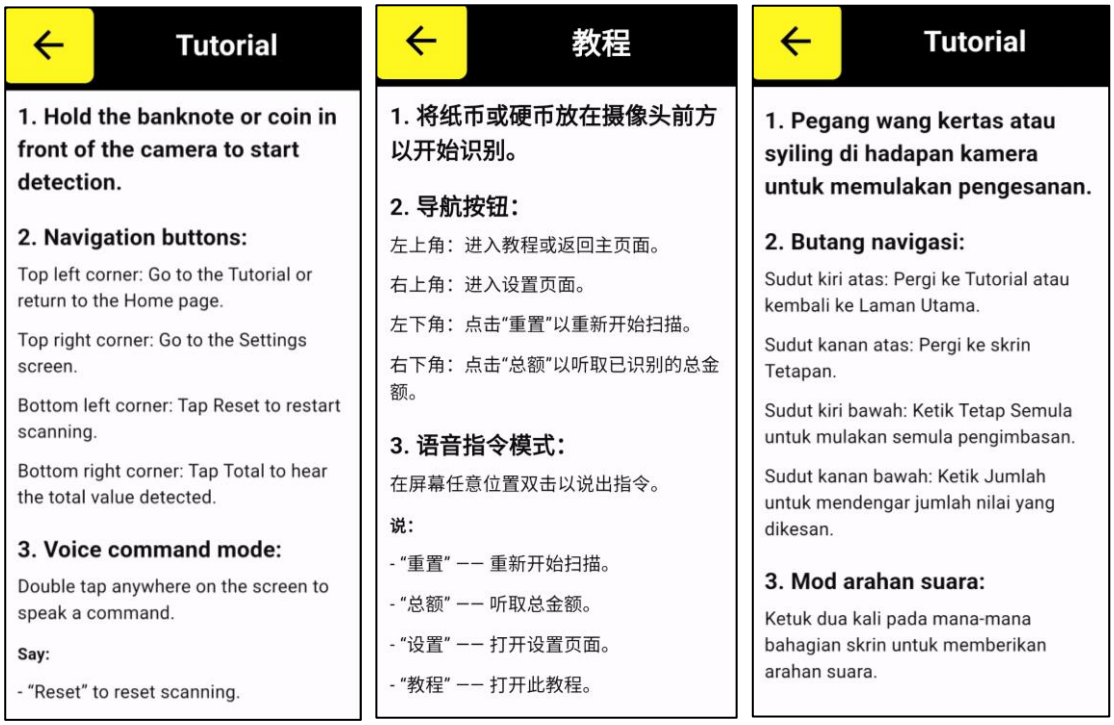


Figure 5.34 Tutorial Screen in different languages

Figure 5.34 shows the tutorial screen in English, Chinese, and Malay. The content on this screen dynamically changes based on the current language setting selected by the user. When a user launches the application for the first time, the system will check automatically and navigate the user to the tutorial screen. When user entered this screen, the text-to-speech (TTS) service will be activated to read the tutorial content aloud which helps the visually impaired individuals to understand the basic usage of the application without needing to rely on visual cues.

5.5.7 Other features

Below are some features designed to improve the usability, accessibility and user experience for the visually impaired individuals:

- **TalkBack and VoiceOver Compatibility**

The application is fully compatible with TalkBack (Android) and VoiceOver (iOS) screen readers. When these features are enabled, the users will receive spoken feedback for each user interface element on the first tap and activate it with double tap. Without further setting on the user interface, the screen readers may only tell the user that the element is a "button" without any further details. Therefore, the app utilizes semantic widgets with proper labels and hints to allow these screen readers to inform users about the function of each button. This enables the user to navigate to each screen and perform actions smoothly.

- **Text-to-Speech**

Automatically speaks out the denomination of detected result, providing real-time audio feedback for the visually impaired user.

- **Total Speak out**

Speak out the total amount of the scanned money at the top of the screen.

- **Reset Button**

Clears all currently scanned results within the active session, allowing users to start a new scan.

- **Automatically Flashlight Control**

Automatically enable device's flashlight in low-light environments by detecting surrounding light intensity through the device's light sensor to enhance visibility during scanning.

- **Haptic Feedback**

Provides vibration feedback when a button is pressed to let the user confirm that the action had been performed.

- **Audio Playback**

Plays a short sound when specific actions such as reset or adjusting settings are successfully performed, giving users additional auditory confirmation.

5.6 Implementation Issues and Challenges

One of the primary challenges in this project is the limitation of the dataset. There is lack of publicly available datasets online that include both Malaysian banknotes and coins that meets our expectation, which makes us need to create a custom dataset for model training. This process involves data acquisition and preprocessing, which is time-consuming. The self-captured data must be label tightly to ensure accurate model training. Even there are datasets that include Malaysian currency, they failed to meet the specific requirements of this project for example the project focuses on real-time detection with the expectation that users will hold the banknotes or coins in front of the device camera. Therefore, the dataset must include images of people holding banknotes and coins in various angles and environments to make sure that the model performs accurately in real-world scenarios.

Another significant challenge is the high similarity between the coins. For example, both 50 cent and 20 cent coins are gold, while 10 cent and 5 cent coins are silver, which makes differentiation become challenging under different lighting conditions. The light reflection from the coins due to some angle makes it more difficult to detect. When coins reflect light at certain angles, some details on the coin may become not visible, which leads to a higher likelihood of misclassification. Additionally, the size difference between coins is little and requires precise feature extraction by the model, which can be further impacted by noise or inconsistent. Therefore, robust data augmentation techniques are required and need to carefully fine-tune the model to overcome real-world complexities.

The next challenge is the lack of native object counting and tracking mechanisms available within the mobile device environments, which impacted the ability to accurately count currency items in real time. While the YOLOv8 model is good at object detection, its output returns a new set of detections for each processed video

frame without inherent object tracking capabilities which cause the same object to be detected repeatedly across successive frames, leading to duplicated counting on the same banknote or coin. Therefore, a custom counting technique is needed to make sure that the value of the detected result could be avoided from adding to the total value again even if the object moved slightly such as rotation or flipping.

Chapter 6

System Evaluation and Discussion

6.1 Performance Metrics

This section presents the evaluation metrics used to evaluate the performance of the trained YOLOv8 model on the test dataset. The selected metrics provide insights into the model's accuracy, robustness, and ability to classify different denominations of Malaysian banknotes and coins correctly. The main evaluation metrics that used to evaluate the model performance are **mAP50** and **mAP50-95**.

6.1.1 Precision and Recall

Precision and recall are fundamental metrics used to evaluate to performance of object detection models.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Figure 6.1 Formula for calculating precision

Precision measures the proportion of positive instances being predicted correctly (true positive) among all instances predicted as positive. If the precision is higher, indicates that less false positive meaning the model makes fewer incorrect positive predictions.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figure 6.2 Formula for calculating recall

Recall measures the proportion of positive instances that predicted correctly (true positive) among all actual positive instances. If the recall is higher, indicates that the model able to identifies most of the relevant objects successfully with less false negatives.

6.1.2 Mean Average Precision (mAP)

$$\text{mAP@}\alpha = \frac{1}{n}\sum_{i=1}^{n} \text{AP}_i \quad \text{for n classes}$$

Figure 6.3 Formula for calculating Mean Average Precision

The mAP50-95 metrics represents the mean average precision calculated over a range of intersection over Union (IoU) thresholds from 0.50 to 0.95. This metric provides a comprehensive evaluation by considering different levels of overlap between predicted and ground truth bounding boxes.

On the other hand, mAP50 is the mean average precision calculated specifically at an IoU threshold of 0.50, offering a more tolerant evaluation that focuses on the model's ability to detect objects with some degrees of overlap.

6.1.3 Confusion Matrix

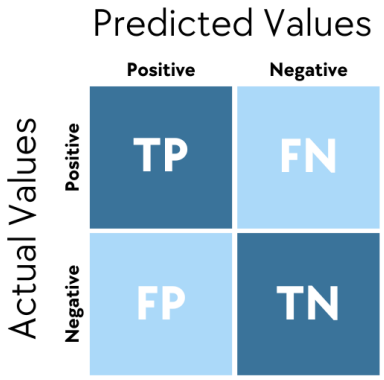


Figure 6.4 Confusion Matrix

The confusion matrix is used to visualize the performance of the classification of the model by showing the number of correct and incorrect predictions across each class such as each denomination of banknotes and coins. It provides a clear breakdown of true positives, false positives, true negatives, and false negatives, providing a detailed insight into the misclassifications between the classes.

6.2 Model Evaluation

The figure below summarizes the mean training results across all classes and epochs.

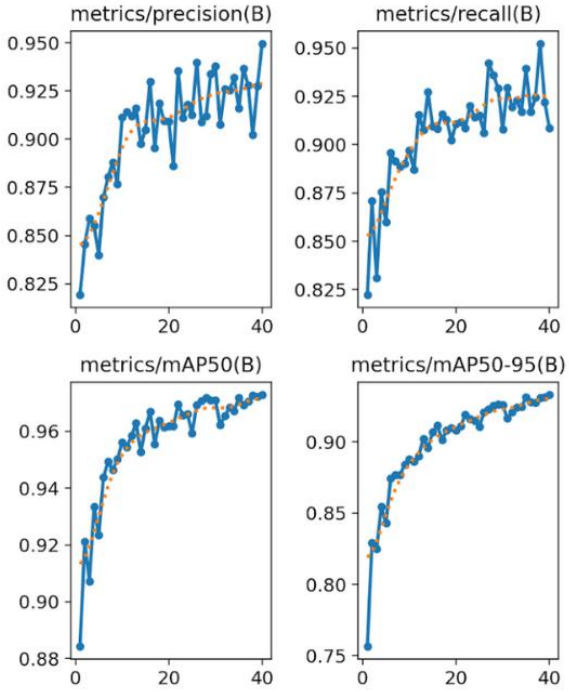


Figure 6.5 Metrics performance during fine-tuning

Based on the observed results, the mAP50 curve (bottom left) gradually increases and eventually stabilizes around 0.98, which indicated that the model is highly reliable in detecting objects with moderate overlap. Additionally, the precision (top left), recall (top right) and mAP50-95 curve (bottom right) which is mean average precision across IoU thresholds ranging from 50% to 95%, further supports this observation. The steady increase in these metrics shown that the model's robustness in detecting objects with different levels of overlapping and positioning, demonstrated its adaptability to diverse real-world environments.

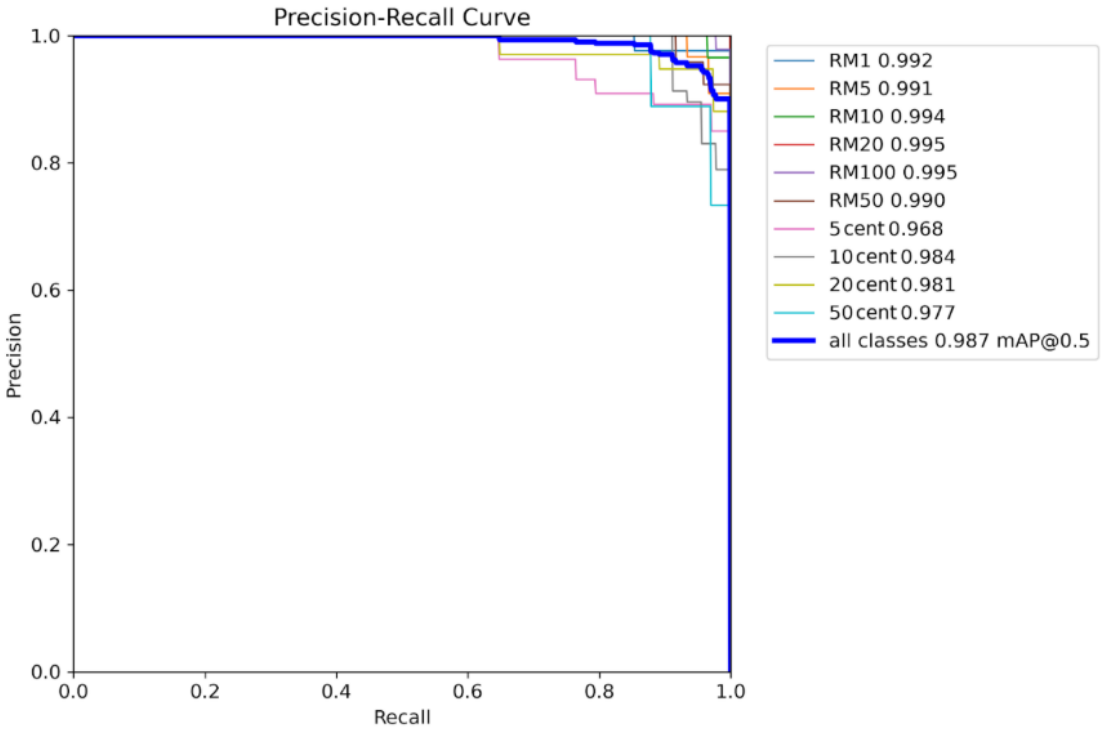


Figure 6.6 Precision-Recall Curve

The Precision-Recall (PR) Curve further illustrates the relationship between precision and recall across different confidence thresholds, providing a deeper understanding of the model's performance in detecting different classes under different conditions. From the PR curve:

Banknote Classes: The classes 'RM1' and 'RM100' achieved a high mAP50 with values of 0.992 and 0.995. This indicates that the detection performance on these classes is good, meaning that the model is highly confident and accurate in identifying these classes. Other banknotes classes such as 'RM5' (0.991), 'RM10' (0.994), and 'RM20' (0.995), also performed well but the 'RM50' banknote achieved the lowest mAP50 among the banknotes at 0.990, which is still a good performance but slightly lower than others.

Coin classes: However, the performance for coin classes is quite low. '20 cent' and '10 cent' coins achieved mAP50 values of 0.981 and 0.984, reflecting reasonable detection performance but not as strong as banknotes. The '5 cent' (0.968) and '50 cent' (0.977) classes show the weakest performance. These lower values suggest that the model struggles to consistently identify these coins compared to the banknotes.

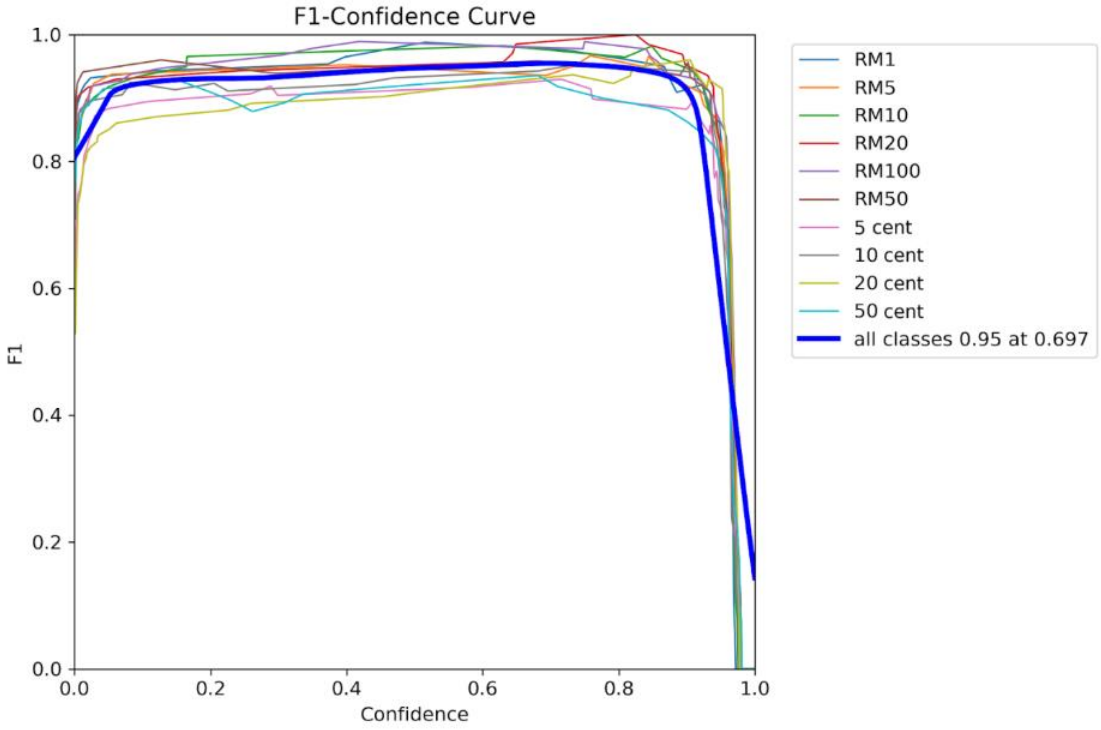


Figure 6.7 F1-Confidence Curve

Based on the observations from the curve, it can be used to support the findings from the precision-recall curve. From the graph, it is evident that coin classes such as, '5 cent', '10 cent', '20 cent' and '50 cent' fall below the mean F1 confidence, indicating the model's performance for these classes is relatively weaker compared to others but still acceptable. The best macro-average F1 score (thick blue line) achieved 0.95 at a confidence threshold of 0.697 meaning that the model can perform reliably at this setting, indicates that it can tolerate some uncertainty during detections without high performance loss.

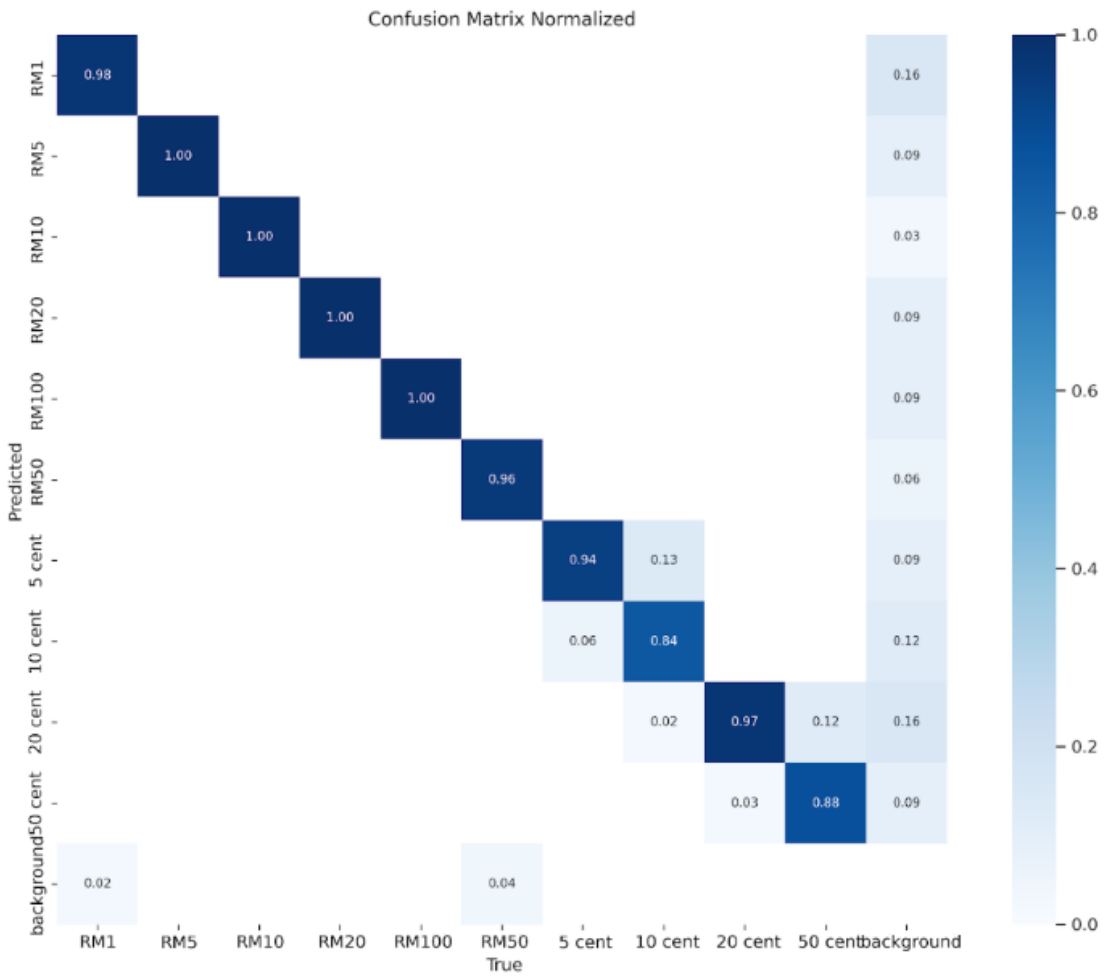


Figure 6.8 Normalized confusion matrix

The normalized confusion matrix shown that the banknotes class further provides a detailed view of how well the model distinguishes between different classes by showing the proportion of correct and incorrect predictions of each class. The diagonal cells represent correctly classified object, the banknotes classes perform well, with ‘RM5’, ‘RM10’, ‘RM20’ and ‘RM100’ achieving perfect accuracy (1.00) and minimal confusion with background. However, there is a small classification for ‘RM50’, where 4% of predictions are misclassified as ‘background, this is probably due to overlapping features between these two classes.

In contrast, the coin classes show higher levels of confusion, for example, ’10 cent’ faces high misclassification with 13% of predictions labelled as ’5 cent’. The other coin classes also shown different levels of misclassification, indicating that the model struggles in detecting and classifying the between the coin classes.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	293	350	0.951	0.961	0.987	0.944
RM1	35	41	0.976	0.979	0.992	0.942
RM5	25	30	0.909	0.967	0.991	0.903
RM10	23	28	0.965	0.987	0.994	0.97
RM20	28	33	0.979	1	0.995	0.957
RM100	34	45	0.978	0.981	0.995	0.95
RM50	21	24	0.958	0.955	0.99	0.889
5 cent	34	34	0.889	0.971	0.968	0.952
10 cent	44	45	0.982	0.911	0.984	0.975
20 cent	37	37	0.874	1	0.981	0.958
50 cent	30	33	1	0.863	0.977	0.946

Speed: 1.0ms preprocess, 4.0ms inference, 0.0ms loss, 2.3ms postprocess per image

Figure 6.9 Overall performance of model on Test Set

Overall, the mAP50 for all classes is 0.987 and mAP50-95 of 0.944 as shown in Figure 6.9, indicating that the model achieved a good performance but still requires improvement in detecting coin classes and the ‘RM50’ banknote.

6.3 Testing Setup and Result

6.3.1 Detection on All Money Classes

The model was tested for its capability to detect all targeted Malaysian banknotes and coins by scanning them under real-world conditions 10 times for each class. Below are the detect results for each class:

Table 6.1 Testing Result of model

Class	Detection Success Rate (%) [n=10]	Highest Confidence Score	Least Confidence Score	Remarks
RM 1	100	97.10	90.36	Detected reliably in all conditions
RM 5	100	97.79	91.28	Detected reliably in all conditions
RM 10	100	96.56	91.03	Detected reliably in all conditions
RM 20	100	96.20	88.75	Detected reliably in all conditions
RM 50	100	96.80	89.04	Detected reliably in all conditions
RM 100	100	97.09	91.19	Detected reliably in all conditions
5 cents	70	93.76	83.21	Lower detection due to poor camera focus and visual similarity to 10 cents
10 cents	80	95.12	86.56	Lower detection due to poor camera focus and visual

				similarity to 5 cents and 20 cents in certain lighting
20 cents	80	97.66	89.02	Lower detection due to poor camera focus and visual similarity to 50 cents
50 cents	100	98.72	90.30	Detected reliably in all conditions

6.3.2 Home Screen

Table 6.2 Test Cases for Home Screen

Test Case ID	Test Description	Input / Action	Expected Result	Actual Result
TC001	First time use	Launch the app	Navigate to tutorial if launch the app for the first time	Pass
TC002	Load model	Launch the app	Model is loaded successfully, and system is ready for inference	Pass
TC003	Open device's camera	Launch the app	Device's camera opens and keep streaming	Pass
TC004	Display detected value and confidence score	Point camera at banknote or coin	Detected denomination and confidence are displayed on screen	Pass

TC005	Prevent adding the same banknote	Keep the same banknote or coin within the camera view	Same banknote or coin is not added multiple times to total value	Pass
TC006	Add total value	Detect multiple banknotes or coins in one session	Detected values are added and updated in total value	Pass
TC007	Display total value of the current scanning session	Launch the app	Total accumulated value is displayed at top of screen	Pass
TC008	Speak out total value	Tap on 'Total' button	Device vibrates and total value is read out with text to speech	Pass
TC009	Reset scanning session	Tap on 'Reset' button	Device vibrates, audio plays, all detected values and total scanned value are cleared and speak reset successfully	Pass
TC010	Open Flashlight	Light intensity < 30 lux	Flashlight turns on	Pass
TC011	Close Flashlight	Light intensity > 30 lux	Flashlight turns off	Pass
TC012	Haptic feedback on button press	Press any interactive button	Device vibrates to confirm button press	Pass
TC013	Audio feedback on button press	Press any interactive button	Audio feedback plays to confirm action	Pass

TC014	Open Tutorial	Tap on 'Tutorial' button	Device vibrates, audio plays, and navigates to Tutorial screen	Pass
TC015	Open Settings	Tap on 'Settings' button	Device vibrates, audio plays, navigates to Settings screen	Pass
TC016	Activate voice command	Long press or double tap on home screen	Voice command activated and listened for speech to perform action	Pass
TC017	Count of denomination tracking	Activate voice command and speak [class] (e.g "1 ringgit")	Device speaks the count recorded for the requested denomination detected in the current session	Pass

6.3.3 Setting Screen

Table 6.3 Test Cases for Setting Screen

Test Case ID	**Test Description**	**Input / Action**	**Expected Result**	**Actual Result**
TC018	Retrieve setting values	Launch the app	Settings in shared preferences are loaded	Pass
TC019	Speak out "Setting"	Open Setting Screen	The word "Setting" is spoken aloud using text to speech	Pass

TC020	Adjust speech volume	Move volume slide in Settings	System adjusts speech volume accordingly	Pass
TC021	Adjust speech speed	Move speed slider in Settings	System adjusts speech speed accordingly	Pass
TC022	Change language	Select different language option	App language changes to selected option	Pass
TC023	Customize sound for each class	Select denomination and assign custom sound	Custom sound is assigned and played for selected denomination	Pass
TC024	Toggle vibration	Enabled/disable vibration toggle in Settings	Device vibrates and plays audio, then haptic feedback is turned on/off based on toggle	Pass
TC025	Return to home screen	Tap on “Back” button	Navigate back to Home Screen	Pass
TC026	Activate voice command	Double tap or long press on setting screen	Voice command activated and listened for speech to perform action	Pass

6.3.4 Tutorial Screen

Table 6.4 Test Cases for Tutorial Screen

Test Case ID	Test Description	Input / Action	Expected Result	Actual Result
TC027	Speak out "Tutorial"	Open Tutorial Screen	The word "Tutorial" is spoken aloud using speech to text	Pass
TC028	Speak out tutorial content	Open Tutorial Screen	The content on the Tutorial Screen is spoken aloud using text to speech	Pass
TC029	Return to home screen	Tap on "Back" button	Navigate back to Home Screen	Pass
TC030	Activate voice command	Double tap or long press on Tutorial Screen	Voice command activated and listened for speech to perform action	Pass

6.3.5 System Speed Performance Testing

The model inference time in the system was tested using a stopwatch function implemented in the application code. The stopwatch was started when the image capture and processing began, and stopped immediately after the model returned the detection result.

```
final stopwatch = Stopwatch()..start();

final results = await _processFrame(image);
stopwatch.stop();

print('Inference time: ${stopwatch.elapsedMilliseconds} ms');
```

Figure 6.10 Code to test model inference time

```
I/flutter (25381): Inference time: 50 ms
```

Figure 6.11 YOLOv8 Model Inference time in system

Figure 6.11 shown that the model inference time is around 50 milliseconds (ms), indicating that the model can perform detections efficiently with minimal delay.

Additionally, the overall performance of the Flutter application was tested using the Flutter DevTools (Performance tool) for 30 seconds under detection of banknote. The tool logged frame rendering statistics during continuous app usage to identify potential bottlenecks and UI lag.

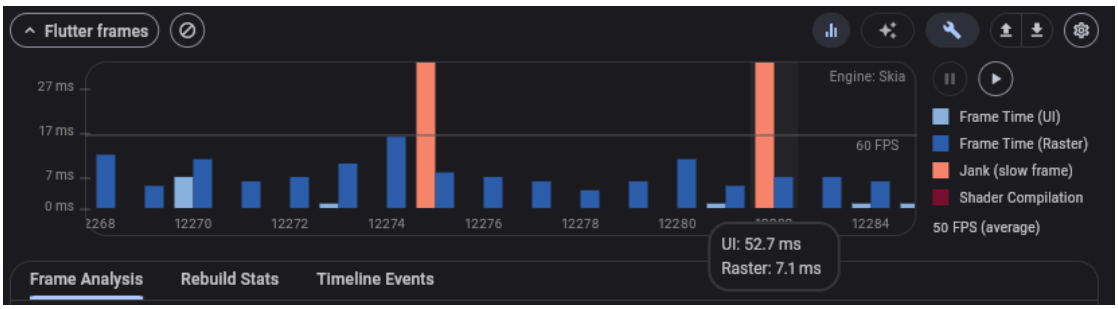


Figure 6.12 Frame rendering analysis from Flutter DevTools

As shown in Figure 6.12, during detection events (which occurred every 5 frames), a Jank (slow frame) was observed. In the Jank frame, the UI thread time was recorded with highest 52.7ms and raster thread time was 7.1ms. However, despite occasional slow frames, the average FPS (frame per second) during the entire testing period remained at around 50fps, indicating acceptable real-time performance.

6.4 Objectives Evaluation

Overall, all the objectives stated have been achieved successfully. The primary objective in this project was to develop a mobile application that could help visually impaired individuals to recognize and count Malaysian banknotes and coins. This objective has been achieved through the integration of the YOLOv8 object detection mode, which shown reliable and precise detection of all targeted banknote and coin classes with the custom counting technique to track and count detected object while avoid duplication of previously recognized items.

Following by second objective which was to provide a fully localized and multilingual application. The app has been successfully developed with support for three major languages: Malay, Mandarin, and English. Users can select their preferred language, and all key functionalities, including voice feedback for detected denominations and tutorial content, are available in the selected language. Testing confirmed that language switching functions correctly and that the pronunciation and clarity of spoken feedback are appropriate for users in all supported languages, making the application inclusive for diverse users in Malaysia.

The final objective was to offer visually impaired users a user-friendly interface equipped with accessibility features. This objective has also been fully achieved with applicates features a simple and intuitive interface with larger and higher contrast buttons and clear labels to make sure the easy navigation. Accessibility functionalities such as voice feedback for both currency detection and app navigation, an audio guided tutorial screen explaining app usage, and voice command support for hands free operation were successfully implemented and tested. Additionally, semantic widgets were also integrated to provide descriptive labels and hints for each button, enhanced compatibility with screen readers such as VoiceOver and TalkBack, further improved accessibility for visually impaired users.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

In this project, the real-time money counting app for visually impaired users utilized the computer vision technologies YOLOv8 to detect and classify Malaysian banknotes and coins. The application offers a range of features, including voice command interaction, dynamic flashlight control based on light intensity, and the integration of a machine learning model (YOLOv8) for accurate real-time money detection. The model was trained using a custom dataset of Malaysian currency with the transfer learning technique, achieving acceptable performance in terms of mAP50-95 of 94.4%, mAP50 of 98.7%, precision of 95.1%, and recall of 96.1%. This provides robust real-time object detection even in different lighting conditions, making the system more adaptable to real-world environments.

This project aimed to address the challenge faced by visually impaired individuals in handling cash transactions. By enabling users to detect, recognize, count individual banknotes and coins, the app offers a solution that promotes greater independence in managing finances. In addition, this project also contributes to the Malaysian Assistive Hub for Advanced Livelihood (MAHAL) by providing an innovative solution for the Malaysian Association of the Blind (MAB) and the visually impaired community for daily currency recognition and counting.

In terms of the app's functionality, the app successfully integrates accessibility features such as haptic feedback, audio feedback on detected denominations, voice commands for hand-free operation, audio-guided tutorial, and support for three languages (Malay, Mandarin, and English) to cater to the Malaysia's diverse linguistic landscape. Moreover, the app is also compatible with device's screen readers such as TalkBack (Android) and VoiceOver (iOS) by implementing semantic widgets to provide descriptive labels and hints for interface elements to enhance accessibility for users with visual impairments. To count and track the detected results, a custom tracking technique based on Euclidean distance calculation was implemented due to lack of tracking technique able to use on Flutter. This mechanism prevents double-

counting of the same banknote or coin while it remains in the camera's view. Each detection result is tracked and only added once towards the total value, thereby improving the accuracy and trustworthiness of the system.

7.2 Recommendation

Although the YOLOv8 model demonstrated high performance in terms of both mAP50 and mAP50-90 on both validation and test set, a slight drop in detection accuracy was observed when the model was deployed on mobile devices, this decline is more notable for coin detection. One contributing factor may be the limited resolution and image quality captured by the older phone cameras. Additionally, the application implements a frame-skipping mechanism to optimize inference speed, which may result in blurred frames being passed to the model which further impacted the detection accuracy. To address this, future work could focus on augmenting the training dataset with more images that simulate lower resolution, blurred, and noisy scenarios to mimic the image captured in the older devices so that the model can maintain robustness across a wider range of hardware.

Currently, the application is designed to handle a single banknote at a time, but it can detect and count up to two banknotes simultaneously. However, the detection accuracy will be reduced once there are more than two banknotes held together, particularly if one is being blocked by another. Future enhancements should consider the implementation of advanced image processing techniques, such as instance segmentation to improve the app's ability to accurately detect and count multiple overlapping banknotes and coins simultaneously.

Furthermore, the application only supports the recognition of the fourth series of Malaysian currency. In future, the dataset could be expanded to include older Malaysian currency series (third series or below) and currencies from other countries to further broaden the app's usability. Therefore, the application can become not only more comprehensive for Malaysian users but also extend its usability to visually impaired users from other countries.

REFERENCES

[1] World Health Organization, “Blindness and vision impairment,” *World Health Organization*, Aug. 10, 2023. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment

[2] “Is the Failure to Produce Tactile Currency Really a Matter of Discrimination?” *www.nfb.org*, 2007. https://www.nfb.org/images/nfb/publications/bm/bm07/bm0702/bm070202.htm

[3] Microsoft Corporation, "Seeing AI," Version 4.0.2, iOS. Microsoft Corporation, 2024. [Online]. Available: https://apps.apple.com/us/app/seeing-ai/id999886767. Accessed: Aug. 27, 2024.

[4] Cash Reader s.r.o, “Cash Reader” Version 1.66, iOS. Cash Reader s.r.o, 2024. [Online]. Available: https://apps.apple.com/my/app/cash-reader-aid-to-the-blind/id1344802905. Accessed: Aug 27, 2024.

[5] Innovation Factory Limited, “Qatari Money Reader - قارئ العم,” *Google.com*, 2021. https://play.google.com/store/apps/details?id=com.mada.arabicmoneyreader&pcampaignid=web_share (accessed Aug. 30, 2024).

[6] M. Data, “MCT Money Reader,” *Google.com*, 2021. https://play.google.com/store/apps/details?id=com.mctdata.ParaTanima&pcampaignid=web_share (accessed Aug. 30, 2024).

[7] Beaverhood LLC, “AR money reader scanner GMoney,” *App Store*, Sep. 23, 2018. https://apps.apple.com/us/app/ar-money-reader-scanner-gmoney/id1430788405 (accessed Aug. 30, 2024).

[8] Abilash CS, Dinakaran M, Hari Vignesh R, and A Anju, “Currency Recognition For The Visually Impaired People,” *2022 IEEE Delhi Section Conference (DELCON)*, Feb. 2022, doi: https://doi.org/10.1109/delcon54057.2022.9753373.

[9] C. J. Tan, W. K. Wong, and T. S. Min, “Malaysian Banknote Reader For Visually Impaired Person,” IEEE Xplore, Sep. 01, 2020. https://ieeexplore.ieee.org/abstract/document/9250979?casa_token=4mWzRucHVy4AAAAA:eHS_Z5QCkZYMGftb4Jty3qteCOID_ROSk-oeaFX67Rfb79OIyM9xGLel2fbjKTKkRJ9yv35IxBidIw

[10] D. Q. Mohd Nazly, P. Isawasan, K. Ahmad Salleh, and S. K. Sugathan, “Malaysia coin identification app using deep learning model,” *Bulletin of Electrical Engineering and Informatics*, vol. 12, no. 4, pp. 2506–2512, Aug. 2023, doi: https://doi.org/10.11591/eei.v12i4.4601.

[11] N. I. M. Hanafiah, Y. Mahmud, M. H. bin Hussin, and S. N. K. Kamarudin, “Malaysian Banknote Recognition App for the Visually Impaired Using Deep Learning,” *2023 4th International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, pp. 287–292, Sep. 2023, doi: https://doi.org/10.1109/aidas60501.2023.10284630.

[12] R. C. Joshi, S. Yadav, and M. K. Dutta, “YOLO-v3 Based Currency Detection and Recognition System for Visually Impaired Persons,” *IEEE Xplore*, Feb. 01, 2020. https://ieeexplore.ieee.org/abstract/document/9077137

[13] Pasumarthy, Rutvi Padhy, R. Yadav, G. Subramaniam, and M. Rao, “An Indian Currency Recognition Model for Assisting Visually Impaired Individuals,” Nov. 2022, doi: https://doi.org/10.1109/rasse54974.2022.9989624.

[14] M. Martin, “Prototyping Model in Software Engineering: Methodology, Process, Approach,” *Guru99.com*, Oct. 24, 2019. https://www.guru99.com/software-engineering-prototyping-model.html

[15] Microsoft, “Visual Studio Code,” *Visual Studio Code*, Apr. 14, 2016. https://code.visualstudio.com/docs/editor/whyvscode

[16] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg

[17] J. Torres, “YOLOv8 Architecture: A Deep Dive into its Architecture - YOLOv8,” *Yolov8*, Jan. 15, 2024. https://yolov8.org/yolov8-architecture/

[18] J. Yin, P. Huang, D. Xiao, and B. Zhang, “A Lightweight Rice Pest Detection Algorithm Using Improved Attention Mechanism and YOLOv8,” *Agriculture*, vol. 14, no. 7, pp. 1052–1052, Jun. 2024, doi: https://doi.org/10.3390/agriculture14071052.

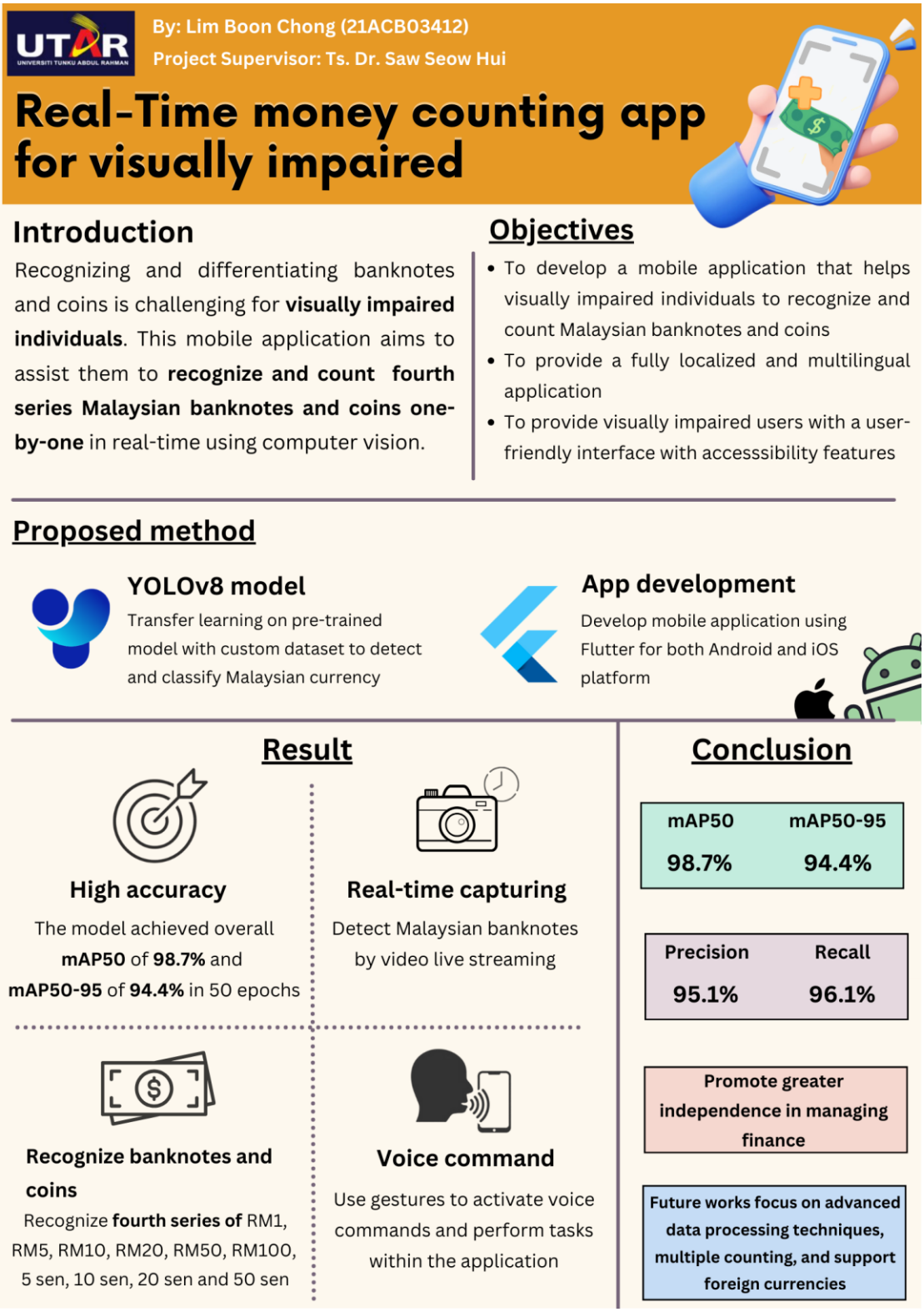
[19] A. Timilsina, “YOLOv8 Architecture Explained!” *Medium*, Mar. 17, 2024. https://abintimilsina.medium.com/yolov8-architecture-explained-a5e90a560ce5

[20] Zorbey Torunoğlu, “Android SharedPreferences | Medium,” *Medium*, May 04, 2024. https://medium.com/@zorbeytorunoglu/android-shared-preferences-ed97627bb0e1 (accessed Dec. 03, 2024).

[21] C. Writer, “What is ByteTrack? A Deep Dive.,” *Roboflow Blog*, Aug. 21, 2024. https://blog.roboflow.com/what-is-bytetrack-computer-vision/

[22] A. Albouchi, S. Messaoud, S. Bouaafia, M. A. Hajjaji, and A. Mtibaa, “Implementation of an improved multi-object detection, tracking, and counting for autonomous driving,” *Multimedia Tools and Applications*, vol. 83, no. 18, pp. 53467–53495, Nov. 2023, doi: https://doi.org/10.1007/s11042-023-17444-w.

POSTER


By: Lim Boon Chong (21ACB03412)
Project Supervisor: Ts. Dr. Saw Seow Hui
Real-Time money counting app for visually impaired
Introduction
Recognizing and differentiating banknotes and coins is challenging for visually impaired individuals. This mobile application aims to assist them to recognize and count fourth series Malaysian banknotes and coins one-by-one in real-time using computer vision.
Objectives
To develop a mobile application that helps visually impaired individuals to recognize and count Malaysian banknotes and coins
To provide a fully localized and multilingual application
To provide visually impaired users with a user-friendly interface with accesssibility features
Proposed method
YOLOv8 model
Transfer learning on pre-trained model with custom dataset to detect and classify Malaysian currency
App development
Develop mobile application using Flutter for both Android and iOS platform
Result
High accuracy
The model achieved overall mAP50 of 98.7% and mAP50-95 of 94.4% in 50 epochs
Real-time capturing
Detect Malaysian banknotes by video live streaming
Recognize banknotes and coins
Recognize fourth series of RM1, RM5, RM10, RM20, RM50, RM100, 5 sen, 10 sen, 20 sen and 50 sen
Voice command
Use gestures to activate voice commands and perform tasks within the application
Conclusion
mAP50 98.7%
mAP50-95 94.4%
Precision 95.1%
Recall 96.1%
Promote greater independence in managing finance
Future works focus on advanced data processing techniques, multiple counting, and support foreign currencies