2023

Faculty of Computers and Artificial Intelligence,
Beni-Suef University, Egypt

BLIND GUIDE

A senior project submitted in partial fulfillment of the requirements for the degree of Bachelor of Computers and Artificial Intelligence.

Computer Science Department & Medical Informatics Department

Supervisor:

Dr. Hossam Moftah

Presented by:

Mohamed Mahmoud Khedr Saad

Omar Adly Abd El Halim Nagdy

Mahmoud Saad Mahmoud Saleh

Aya Maher Mohamed Galal

Hager Sayed Moharam Gouda

Acknowledgement

We would like to convey Our heartfelt gratitude **Dr. Hossam Moftah** for her tremendous support and assistance in the completion of our project. I would also like to thank **Eng. Muhamad Gamal El-Dien**, for his time and efforts he provided throughout the year. And, for providing us with this wonderful opportunity to work on a project. The completion of the project would not have been possible without their help and insights.

Abstract

This abstract provides a brief overview of “Blind Guide Project" that utilizes the YOLOv8 model for detecting objects from the COCO dataset and the SSD-MobileNet-v2-FPNLite-320 model for detecting currency. The aim of this project is to assist visually impaired individuals in navigating their surroundings and recognizing different objects and currencies with the help of computer vision technology. The Blind Guide Project focuses on developing a system that can detect various objects from the COCO dataset, which includes a wide range of common everyday objects. The YOLOv8 model, known for its high accuracy and real-time object detection capabilities, is implemented to identify and locate objects within a given image or video stream. By providing real-time audio feedback, the system enables visually impaired users to understand their surroundings more effectively.

In addition to object detection, the project also incorporates the SSD-MobileNet-v2-FPNLite-320 model to recognize different currencies. This feature allows visually impaired individuals to independently handle monetary transactions by identifying and differentiating between various denominations of bills and coins.

The Blind Guide Project holds significant potential to enhance the independence and mobility of visually impaired individuals, empowering them to navigate their surroundings more confidently. The use of computer vision models like YOLOv8 and SSD-MobileNet-v2-FPNLite-320 offers accurate and efficient detection of objects and currencies, making the system a valuable tool for visually impaired individuals in their daily lives.

Table Of Content

Chapter 1: Introduction

1.1 Overview

The Blind Guide Project is a graduation project that aims to empower visually impaired individuals by leveraging computer vision technology to assist them in navigating their surroundings and recognizing objects and currency. This project utilizes two state-of-the-art object detection models: YOLOv8 for detecting objects from the COCO dataset and SSD-MobileNet-v2-FPNLite-320 for recognizing different currencies. By combining these models, the Blind Guide system provides real-time audio feedback to visually impaired users, enabling them to better understand their environment and handle monetary transactions independently.

1.2 Problem Statements

Visually impaired individuals face significant challenges in navigating their surroundings and identifying objects, which affects their independence and mobility. Traditional mobility aids provide limited assistance in detecting obstacles but do not address the need for object recognition and identification. The lack of accessible technology that can accurately detect and describe objects in real-time poses a significant barrier to the daily lives of visually impaired individuals.

The problem statement for the Blind Guide Project is to develop an effective and efficient object detection system that caters specifically to the needs of visually impaired individuals. This system should address the following key challenges:

1.2.1 Object Recognition

: Visually impaired individuals struggle to identify and recognize objects in their environment, which limits their ability to interact independently. Existing solutions often lack the accuracy and real-time performance necessary for effective object recognition, leaving visually impaired individuals reliant on sighted assistance.

1.2.2 Real-Time Feedback: Timely and accurate information about the objects in the environment is essential for visually impaired individuals to make informed decisions and navigate their surroundings safely. The lack of real-time feedback systems that can provide instant audio cues or descriptions of objects hinders their independence.

1.2.3 Accessibility and Usability: Existing object detection systems are often complex and not designed with the specific needs and limitations of visually impaired individuals in mind. The Blind Guide Project aims to develop a system that is user-friendly, intuitive, and accessible, allowing visually impaired users to easily interact with and benefit from the technology.

1.2.4 Currency Recognition: In addition to object detection, visually impaired individuals often face challenges in identifying and differentiating currency. Handling financial transactions independently is crucial for their autonomy, and a lack of reliable currency recognition systems further hampers their ability to participate fully in economic activities.

By addressing these challenges, the Blind Guide Project seeks to provide visually impaired individuals with an advanced object detection system that can accurately identify and describe objects in real-time. The system should offer accessible and intuitive interfaces, providing audio feedback and descriptions to enable visually impaired individuals to navigate their surroundings independently and handle financial transactions confidently.

1.3 Objectives

The Blind Guide Project aims to address the challenges faced by visually impaired individuals in navigating their surroundings and recognizing objects. The project sets forth the following objectives:

1.3.1 Develop an Accurate Object Detection System: The primary objective of the Blind Guide Project is to develop an object detection system that can

accurately detect and classify objects in real-time. By leveraging advanced computer vision techniques and deep learning models, the system should have a high accuracy rate in identifying objects from various categories.

1.3.2 Enable Real-Time Audio Feedback: The project aims to provide visually impaired individuals with real-time audio feedback about the objects in their environment. The system should generate audio cues or descriptions that convey relevant information, such as object category, location, and size, enabling users to comprehend and interact with their surroundings effectively.

1.3.3 Enhance Independence and Mobility: The Blind Guide Project seeks to empower visually impaired individuals by enhancing their independence and mobility. The system should enable users to navigate their surroundings confidently without relying heavily on sighted assistance, thereby improving their overall quality of life.

1.3.4 Incorporate Currency Recognition: In addition to object detection, the project aims to integrate currency recognition capabilities into the Blind Guide system. The system should accurately detect and differentiate various denominations of bills and coins, providing real-time audio feedback to facilitate independent financial transactions for visually impaired individuals.

1.3.5 Ensure User-Friendly Interface: The Blind Guide system should feature a user-friendly interface that is accessible and easy to use for visually impaired individuals. The interface should be intuitive, providing clear and concise audio cues or feedback, and enabling users to interact with the system effortlessly.

1.3.6 Test and Evaluate System Performance: The project aims to thoroughly test and evaluate the performance of the Blind Guide system. This includes assessing the accuracy and reliability of object detection and currency recognition, evaluating the real-time performance, and soliciting

feedback from visually impaired individuals to ensure the system meets their needs effectively.

1.3.7 **Promote Awareness and Accessibility**: The project aims to raise awareness about the challenges faced by visually impaired individuals and the potential of technology in improving their lives. By advocating for accessibility and promoting the use of the Blind Guide system, the project seeks to create a more inclusive society for visually impaired individuals.

By achieving these objectives, the Blind Guide Project endeavors to provide visually impaired individuals with a reliable, accurate, and user-friendly object detection system that enhances their independence, mobility, and overall quality of life.

1.4 Methodology

The Blind Guide Project employs a comprehensive methodology to develop an effective object detection system tailored to the needs of visually impaired individuals. The methodology encompasses several key steps, including data collection, model training, system development, and evaluation. The following outlines the methodology employed in the Blind Guide Project:

1.4.1 **Data Collection**: To train the object detection model, a diverse and representative dataset is essential. The project begins by collecting a large dataset of images that encompass various object categories encountered in everyday life. The dataset should include objects commonly found in indoor and outdoor environments, such as furniture, appliances, vehicles, and personal belongings. Additionally, a specific dataset containing images of different currencies is collected to train the currency recognition model.

1.4.2 **Preprocessing**: The collected dataset undergoes preprocessing to ensure consistency and compatibility. This includes resizing the images, normalizing the color channels, and augmenting the dataset with techniques such as rotation, flipping, and scaling. Preprocessing

techniques help to increase the diversity and robustness of the dataset, improving the performance and generalization ability of the models.

1.4.3 **Model Selection**: The Blind Guide Project utilizes two different object detection models: YOLOv8 for general object detection and SSD-MobileNet-v2-FPNLite-320 for currency recognition. The selection of these models is based on their proven performance, real-time capabilities, and suitability for the task at hand. The chosen models have been extensively researched and have demonstrated high accuracy and efficiency in object detection and currency recognition tasks, respectively.

1.4.4 **Model Training**: The selected object detection models are trained using the preprocessed dataset. The training process involves optimizing the models' parameters and weights to learn the patterns and features associated with the target objects and currencies. This is achieved using deep learning techniques, specifically convolutional neural networks (CNNs). The models are trained using a combination of labeled images and ground truth annotations, enabling them to learn to detect and classify objects and currencies accurately.

1.4.5 **System Development**: Once the models are trained, the Blind Guide system is developed. The system integrates object detection models, camera or video input, audio output, and user interface. The system is designed to run in real-time on a device capable of processing video input, such as a smartphone or a portable camera. The user interface is developed with accessibility in mind, ensuring that visually impaired individuals can easily interact with the system using auditory cues or tactile feedback.

1.4.6 **System Integration and Testing**: The Blind Guide system is integrated with the object detection models, and extensive testing is conducted to ensure its functionality, accuracy, and usability. The system is tested with a variety of input scenarios, including different lighting conditions, various

object orientations, and different currency denominations. The performance of the system is evaluated based on its ability to detect objects and recognize currencies accurately in real-time. User feedback is also collected to gather insights and make necessary improvements to enhance the user experience.

1.4.7 **Performance Evaluation**: The performance of the Blind Guide system is assessed through rigorous evaluation measures. This includes calculating metrics such as mAP to measure the accuracy of object detection and currency recognition. Additionally, the real-time performance of the system, including processing speed and latency, is measured to ensure that visually impaired users receive timely audio feedback.

1.4.8 **Iterative Refinement**: Based on the evaluation results and user feedback, the Blind Guide system undergoes iterative refinement. This involves fine-tuning the object detection models, optimizing the system's performance, and improving the user interface to address any identified issues or limitations. The iterative refinement process continues until the system meets the desired performance and usability standards.

By following this methodology, the Blind Guide Project ensures the development of a robust and accurate object detection system specifically tailored to the needs of visually impaired individuals. The methodology emphasizes data quality, model selection.

Chapter 2: Literature Review

Object detection is a key ability required by most computer and robot vision systems. The latest research on this area has been making great progress in many directions. In the current manuscript, we give an overview of past research on object detection, outline the current main research directions, and discuss open problems and possible future directions.

2.1 introduction

During the last years, there has been a rapid and successful expansion on computer vision research. Parts of this success have come from adopting and adapting machine learning methods, while others from the development of new representations and models for specific computer vision problems or from the development of efficient solutions. One area that has attained great progress is object detection. The present works gives a perspective on object detection research.

Given a set of object classes, object detection consists in determining the location and scale of all object instances, if any, that are present in an image. Thus, the objective of an object detector is to find all object instances of one or more given object classes regardless of scale, location, pose, view with respect to the camera, partial occlusions, and illumination conditions.

In many computer vision systems, object detection is the first task being performed as it allows us to obtain further information regarding the detected object and about the scene. Once an object instance has been detected (e.g., a face), it is be possible to obtain further information, including: (i) to recognize the specific instance (e.g., to identify the subject's face), (ii) to track the object over an image sequence (e.g., to track the face in a video), and (iii) to extract further information about the object (e.g., to determine the subject's gender), while it is also possible to (a) infer the presence or location of other objects in the scene (e.g., a hand may be near a face and at a similar scale) and (b) to better estimate further information about the scene (e.g., the type of scene, indoor versus outdoor, etc.), among other contextual information.

Object detection has been used in many applications, with the most popular ones being: (i) human-computer interaction (HCI), (ii) robotics (e.g., service robots), (iii)

consumer electronics (e.g., smart-phones), (iv) security (e.g., recognition, tracking), (v) retrieval (e.g., search engines, photo management), and (vi) transportation (e.g., autonomous, and assisted driving). Each of these applications has different requirements, including processing time (off-line, on-line, or real-time), robustness to occlusions, invariance to rotations (e.g., in-plane rotations), and detection under pose changes. While many applications consider the detection of a single object class (e.g., faces) and from a single view (e.g., frontal faces), others require the detection of multiple object classes (humans, vehicles, etc.), or of a single class from multiple views (e.g., side, and frontal view of vehicles). In general, most systems can detect only a single object class from a restricted set of views and poses.

Several surveys on detection and recognition have been published during the last years [see Hjelmås and Low (2001), Yang et al. (2002), Sun et al. (2006), Li and Allinson (2008), Enzweiler and Gavrila (2009), Dollar et al. (2012), Andreopoulos and Tsotsos (2013), Li et al. (2015), and Zafeiriou et al. (2015)], and there are four main problems related to object detection. The first one is object localization, which consists of determining the location and scale of a single object instance known to be present in the image; the second one is object presence classification, which corresponds to determining whether at least one object of a given class is present in an image (without giving any information about the location, scale, or the number of objects), while the third problem is object recognition, which consist in determining if a specific object instance is present in the image. The fourth related problem is view and pose estimation, which consist of determining the view of the object and the pose of the object.

The problem of object presence classification can be solved using object detection techniques, but in general, other methods are used, as determining the location and scale of the objects is not required, and determining only the presence can be done more efficiently. In some cases, object recognition can be solved using methods that do not require detecting the object in advance [e.g., using methods based on Local Interest Points such as Tuytelaars and Mikolajczyk (2008) and Ramanan and Niranjan (2012)]. Nevertheless, solving the object detection problem would solve (or help simplify) these related problems. An additional, recently

addressed problem corresponds to determining the “objectness” of an image patch, i.e., measuring the likeliness for an image window to contain an object of any class [e.g., Alexe et al. (2010), Endres and Hoiem (2010), and Huval et al. (2013)]. In the following, we give a summary of past research on object detection, present an overview of current research directions, and discuss open problems and possible future directions, all this with a focus on the classifiers and architectures of the detector, rather than on the used features.

2.2 A Brief Review of Object Detection Research

Early works on object detection were based on template matching techniques and simple part-based models [e.g., Fischler and Elschlager (1973)]. Later, methods based on statistical classifiers (e.g., Neural Networks, SVM, Adaboost, Bayes, etc.) were introduced [e.g., Osuna et al. (1997), Rowley et al. (1998), Sung and Poggio (1998), Schneiderman and Kanade (2000), Yang et al. (2000a,b), Fleuret and Geman (2001), Romdhani et al. (2001), and Viola and Jones (2001)]. This initial successful family of object detectors, all of them based on statistical classifiers, set the ground for most of the following research in terms of training and evaluation procedures and classification techniques.Because face detection is a critical ability for any system that interacts with humans, it is the most common application of object detection. However, many additional detection problems have been studied [e.g., Papageorgiou and Poggio (2000), Agarwal et al. (2004), Alexe et al. (2010), Everingham et al. (2010), and Andreopoulos and Tsotsos (2013)]. Most cases correspond to objects that people often interact with, such as other humans [e.g., pedestrians (Papageorgiou and Poggio, 2000; Viola and Jones, 2002; Dalal and Triggs, 2005; Bourdev et al., 2010; Paisitkriangkrai et al., 2015)] and body parts [(Kölsch and Turk, 2004; Ong and Bowden, 2004; Wu and Nevatia, 2005; Verschae et al., 2008; Bourdev and Malik, 2009) e.g., faces, hands, and eyes], as well as vehicles [(Papageorgiou and Poggio, 2000; Felzenszwalb et al., 2010b), e.g., cars and airplanes], and animals [e.g., Fleuret and Geman (2008)].Most object detection systems consider the same basic scheme, commonly known as sliding window: in order to detect the objects appearing in the image at different scales and locations, an exhaustive search is applied. This search makes use of a classifier, the core part of the detector, which indicates if a given image patch corresponds to the object or

not. Given that the classifier basically works at a given scale and patch size, several versions of the input image are generated at different scales, and the classifier is used to classify all possible patches of the given size, for each of the downscaled versions of the image. Basically, three alternatives exist to the sliding window scheme. The first one is based on the use of bag-of-words (Weinland et al., 2011; Tsai, 2012), method sometimes used for verifying the presence of the object, and that in some cases can be efficiently applied by iteratively refining the image region that contains the object [e.g., Lampert et al. (2009)]. The second one samples patches and iteratively searches for regions of the image where it is likely that the object is present [e.g., Prati et al. (2012)]. These two schemes reduce the number of image patches where to perform the classification, seeking to avoid an exhaustive search over all image patches. The third scheme finds key-points and then matches them to perform the detection [e.g., Azzopardi and Petkov (2013)]. These schemes cannot always guarantee that all object's instances will be detected.

2.3 Object Detection Approaches

Object detection methods can be grouped in five categories, each with merits and

Method	Coarse-to-fine and boosted classifiers	Dictionary based	Deformable part-based models	Deep learning	Trainable image processing architectures
Accuracy	++	+=	++	++	+=
Generality	==	++	+=	++	+=
Speed	++	+=	==	+=	+=
Advantages	Real-time, it can work at small resolutions	Representation can be shared across classes	It can handle deformations and occlusions	Representation can be transfered to other classes	General-purpose architecture that can be used is several modules of a system
Drawbacks/ requirements	Features are predefined	It may not detect all object instances	It can not detect small objects	Large training sets specialized hardware (GPU) for efficiency	The obtained system may be Too specialized for a particular setting
Typical applications	Robotics, security	Retrieval, search	Transportation pedestrian detection	Retrieval, search	HCI, health, robotics

Accuracy: ++, High; +=, Good; ==, Low.
Speed: ++, real-time (15 fps or more); +=, online (10–5 fps); ==, offline (5 fps or more).
Generality: ++ (+=), applicable to many (some) object classes; ==, depend on features designed for specific classes.

demerits: while some are more robust, others can be used in real-time systems, and others can be handling more classes, etc. Table 1 gives a qualitative comparison.

2.3.1 Coarse-to-Fine and Boosted Classifiers

The most popular work in this category is the boosted cascade classifier of Viola and Jones (2004). It works by efficiently rejecting, in a cascade of test/filters, image patches that do not correspond to the object. Cascade methods are commonly used with boosted classifiers due to two main reasons: (i) boosting generates an additive classifier, thus it is easy to control the complexity of each stage of the cascade and (ii) during training, boosting can be also used for feature selection, allowing the use of large (parametrized) families of features. A coarse-to-fine cascade classifier is usually the first kind of classifier to consider when efficiency is a key requirement. Recent methods based on boosted classifiers include Li and Zhang (2004), Gangaputra and Geman (2006), Huang et al. (2007), Wu and Nevatia (2007), Verschae et al. (2008), and Verschae and Ruiz-del-Solar (2012).

2.3.2 Dictionary Based

The best example in this category is the Bag of Word method [e.g., Serre et al. (2005) and Mutch and Lowe (2008)]. This approach is basically designed to detect a single object per image, but after removing a detected object, the remaining objects can be detected [e.g., Lampert et al. (2009)]. Two problems with this approach are that it cannot robustly handle well the case of two instances of the object appearing near each other, and that the localization of the object may not be accurate.

2.3.3 Deformable Part-Based Model

This approach considers object and part models and their relative positions. In general, it is more robust than other approaches, but it is rather time consuming and cannot detect objects appearing at small scales. It can be traced back to the deformable models (Fischler and Elschlager, 1973), but successful methods are recent (Felzenszwalb et al., 2010b). Relevant works include Felzenszwalb et al. (2010a) and Yan et al. (2014), where efficient evaluation of deformable part-based model is implemented using a coarse-to-fine cascade model for faster evaluation, Divvala et al. (2012), where the relevance of the part-models is analyzed, among others [e.g., Azizpour and Laptev (2012), Zhu and Ramanan (2012), and Girshick et al. (2014)].

2.3.4 Deep Learning

One of the first successful methods in this family is based on convolutional neural networks (Delakis and Garcia, 2004). The key difference between this and the above approaches is that in this approach the feature representation is learned instead of being designed by the user, but with the drawback that many training samples is required for training the classifier. Recent methods include Dean et al. (2013), Huval et al. (2013), Ouyang and Wang (2013), Sermanet et al. (2013), Szegedy et al. (2013), Zeng et al. (2013), Erhan et al. (2014), Zhou et al. (2014), and Ouyang et al. (2015).

2.3.5 Trainable Image Processing Architectures

In such architectures, the parameters of predefined operators and the combination of the operators are learned, sometimes considering an abstract notion of fitness. These are general-purpose architectures, and thus they can be used to build several modules of a larger system (e.g., object recognition, key point detectors and object detection modules of a robot vision system). Examples include trainable COSFIRE filters (Azzopardi and Petkov, 2013, 2014), and Cartesian Genetic Programming (CGP) (Harding et al., 2013; Leitner et al., 2013).

2.4 Current Research Problems

Table 2 presents a summary of solved current and open problems. In the present section we discuss current research directions.

Solved problems	Single-class	Single-view	Small deformations	Multi-scale
Current directions	Multi-class (scalability and efficiency)	Multi-view/pose Multi-resolution	Occlusions, deformable Interlaced object and background	Contextual information Temporal features
Open	Incremental learning	Object-part relation	Pixel-level detection Background objects	Multi-modal

2.4.1 multi-Class

Many applications require detecting more than one object class. If many classes is being detected, the processing speed becomes an important issue, as well as the kind of classes that the system can handle without accuracy loss. Works that have addressed the multi-class detection problem include Torralba et al. (2007), Razavi

et al. (2011), Benbouzid et al. (2012), Song et al. (2012), Verschae and Ruiz-del-Solar (2012), and Erhan et al. (2014). Efficiency has been addressed, e.g., by using the same representation for several object classes, as well as by developing multi-class classifiers designed specifically to detect multiple classes. Dean et al. (2013) presents one of the few existing works for very large-scale multi-class object detection, where 100,000 object classes were considered.

2.4.2 Multi-View, Multi-Pose, Multi-Resolution

Most methods used in practice have been designed to detect a single object class under a single view, thus these methods cannot handle multiple views, or large pose variations; except for deformable part-based models which can deal with some pose variations. Some works have tried to detect objects by learning subclasses (Wu and Nevatia, 2007) or by considering views/poses as different classes (Verschae and Ruiz-del-Solar, 2012); in both cases improving the efficiency and robustness. Also, multi-pose models [e.g., Erol et al. (2007)] and multi-resolution models [e.g., Park et al. (2010)] have been developed.

2.4.3 Efficiency and Computational Power

Efficiency is an issue to be considered in any object detection system. As mentioned, a coarse-to-fine classifier is usually the first kind of classifier to consider when efficiency is a key requirement [e.g., Viola et al. (2005)], while reducing the number of image patches where to perform the classification [e.g., Lampert et al. (2009)] and efficiently detecting multiple classes [e.g., Verschae and Ruiz-del-Solar (2012)] have also been used. Efficiency does not imply real-time performance and works such as Felzenszwalb et al. (2010b) are robust and efficient, but not fast enough for real-time problems. However, using specialized hardware (e.g., GPU) some methods can run in real-time (e.g., deep learning).

2.4.4 Occlusions, Deformable Objects, and Interlaced Object and Background

Dealing with partial occlusions is also an important problem, and no compelling solution exists, although relevant research has been done [e.g., Wu and Nevatia

(2005)]. Similarly, detecting objects that are not “closed,” i.e., where objects and background pixels are interlaced with background is still a difficult problem. Two examples are hand detection [e.g., Kölsch and Turk (2004)] and pedestrian detection [see Dollar et al. (2012)]. Deformable part-based models [e.g., Felzenszwalb et al. (2010b)] have been to some extend successful under this kind of problem, but further improvement is still required.

2.4.5 Contextual Information and Temporal Features

Integrating contextual information (e.g., about the type of scene, or the presence of other objects) can increase speed and robustness, but “when and how” to do this (before, during or after the detection), it is still an open problem. Some proposed solutions include the use of (i) spatio-temporal context [e.g., Palma-Amestoy et al. (2010)], (ii) spatial structure among visual words [e.g., Wu et al. (2009)], and (iii) semantic information aiming to map semantically related features to visual words [e.g., Wu et al. (2010)], among many others [e.g., Torralba and Sinha (2001), Divvala et al. (2009), Sun et al. (2012), Mottaghi et al. (2014), and Cadena et al. (2015)]. While most methods consider the detection of objects in a single frame, temporal features can be beneficial [e.g., Viola et al. (2005) and Dalal et al. (2006)].

2.5 Open Problems and Future Directions

In the following, we outline problems that we believe have not been addressed, or addressed only partially, and may be interesting relevant research directions.

2.5.1 Open-World Learning and Active Vision

An important problem is to incrementally learn, to detect new classes, or to incrementally learn to distinguish among subclasses after the “main” class has been learned. If this can be done in an unsupervised way, we will be able to build new classifiers based on existing ones, without much additional effort, greatly reducing the effort required to learn new object classes. Note that humans are continuously inventing new objects, fashion changes, etc., and therefore detection systems will need to be continuously updated, adding new classes, or updating existing ones. Some recent works have addressed these issues, mostly based on deep learning

and transfer learning methods [e.g., Bengio (2012), Mesnil et al. (2012), and Kotzias et al. (2014)]. Open-world learning is of particular importance in robot applications, case where active vision mechanisms can aid in the detection and learning [e.g., Paletta and Pinz (2000) and Correa et al. (2012)].

2.5.2 Object-Part Relation

During the detection process, should we detect the object first or the parts first? This is a basic dilemma, and no clear solution exists. Probably, the search for the object and for the parts must be done concurrently where both processes give feedback to each other. How to do this is still an open problem and is likely related to how to use context information. Moreover, in cases where the object part can be also decomposed in subparts, an interaction among several hierarchies emerges, and in general it is not clear what should be done first.

2.5.3 Multi-Modal Detection

The use of new sensing modalities, in particular depth and thermal cameras, has seen some development in the last years [e.g., Fehr and Burkhardt (2008) and Correa et al. (2012)]. However, the methods used for processing visual images are also used for thermal images, and to a lesser degree for depth images. While using thermal images makes it easier to discriminate the foreground from the background, it can only be applied to objects that irradiate infrared light (e.g., mammals, heating, etc.). Using depth images is easy to segment the objects, but general methods for detecting specific classes have not been proposed, and probably higher resolution depth images are required. It seems that depth and thermal cameras alone are not enough for object detection, at least with their current resolution, but further advances can be expected as the sensing technology improves.

2.5.4 Pixel-Level Detection (Segmentation) and Background Objects

In many applications, we may be interested in detecting objects that are usually considered as background. The detection of such “background objects,” such as rivers, walls, mountains, has not been addressed by most of the here mentioned

approaches. In general, this kind of problem has been addressed by first segmenting the image and later labeling each segment of the image [e.g., Peng et al. (2013)]. Of course, for successfully detecting all objects in a scene, and to completely understand the scene, we will need to have a pixel level detection of the objects, and furthermore, a 3D model of such scene. Therefore, at some point object detection and image segmentation methods may need to be integrated. We are still far from attaining such automatic understanding of the world, and to achieve this, active vision mechanisms might be required [e.g., Aloimonos et al. (1988) and Cadena et al. (2015)].

2.6 Conclusion

Object detection is a key ability for most computers and robot vision systems. Although great progress has been observed in the last years, and some existing techniques are now part of many consumer electronics (e.g., face detection for auto-focus in smartphones) or have been integrated in assistant driving technologies, we are still far from achieving human-level performance, in terms of open-world learning. It should be noted that object detection has not been used much in many areas where it could be of great help. As mobile robots, and in general autonomous machines, are starting to be more widely deployed (e.g., quadcopters, drones and soon service robots), the need of object detection systems is gaining more importance. Finally, we need to consider that we will need object detection systems for nano-robots or for robots that will explore areas that have not been seen by humans, such as depth parts of the sea or other planets, and the detection systems will have to learn to new object classes as they are encountered. In such cases, a real-time open-world learning ability will be critical.

Chapter 3: System Design

3.1 Introduction:

The main topic of this chapter is to analysis the project and define the requirements of the project. We will show all diagrams for example use case diagram that explain user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved, and sequence diagram that shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

3.2 Requirements:

Requirements engineering is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system. The requirement's engineering process is an iterative process that involves several steps, including:

Requirements Elicitation: This is the process of gathering information about the needs and expectations of stakeholders for the software system. This step involves interviews, surveys, focus groups, and other techniques to gather information from stakeholders.

Requirements Analysis: This step involves analyzing the information gathered in the requirements elicitation step to identify the high-level goals and objectives of the software system. It also involves identifying any constraints or limitations that may affect the development of the software system.

Requirements Specification: This step involves documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.

Requirements Validation: This step involves checking that the requirements are complete, consistent, and accurate. It also involves checking that the requirements are testable and that they meet the needs and expectations of stakeholders.

Requirements Management: This step involves managing the requirements throughout the software development life cycle, including tracking and

controlling changes, and ensuring that the requirements are still valid and relevant.

The Requirements Engineering process is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders, and that it is developed on time, within budget, and to the required quality.

Requirements for blind guide application

ID	Module	Functionality Description	Related Stakeholders	Date	Type (Functional/Non-functional)
RQ001	Object Recognition	The application should recognize and identify objects in the environment.	Users, Developers	2023-06-07	Functional
RQ002	currency detection	The application should recognize and identify the type of currency in the environment.	Users, Developers	2023-06-07	Functional

RQ003	Accessible User Interface	The application should have an accessible user interface for blind users.	Users, Developers	2023-06-07	Non-functional
RQ004	Customization and Personalization	The application should allow users to customize settings and preferences.	users, Developers	2023-06-07	Functional
RQ005	Scalability	The application should be scalable to accommodate a growing user base.	Developers	2023-06-07	Non-functional
RQ006	Compatibility	The application should be compatible with various devices and operating systems.	Blind users, Developers	2023-06-07	Non-functional

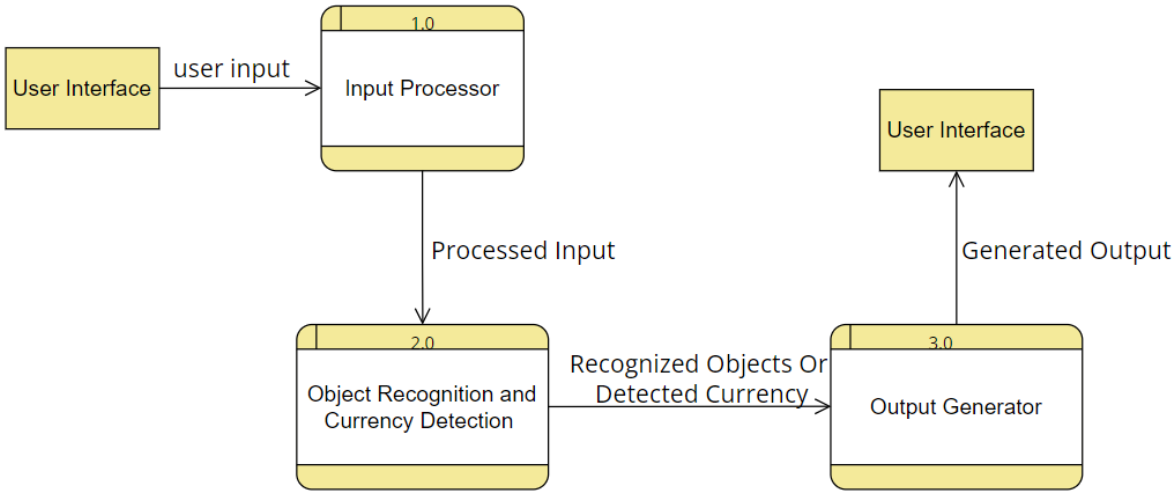
RQ007	Performance Requirements	Specify performance requirements for object recognition and currency detection.	Users, Developers	2023-06-07	Non-functional
RQ008	Privacy and Security	The application should ensure the privacy and security of user data.	Users, Developers	2023-06-07	Non-functional
RQ009	Error Handling and Recovery	Specify requirements for error handling and recovery mechanisms.	Users, Developers	2023-06-07	Non-functional

3.3 Data flow diagram (DFD)

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new

one. Like all the best diagrams and charts, a DFD can often visually "say" things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That is why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

Data flow diagram for blind guide application



3.4 Use case diagram

A use case diagram is a visual representation of the interactions between actors (users or external systems) and a system under consideration. It is one of the Unified Modeling Language (UML) diagrams used in software engineering and system analysis to capture the functional requirements of a system.

The main purpose of a use case diagram is to illustrate how users or external systems interact with the system to achieve specific goals or complete certain tasks. It helps in understanding the system's functionality from a user's perspective.

In a use case diagram, the following elements are typically represented:

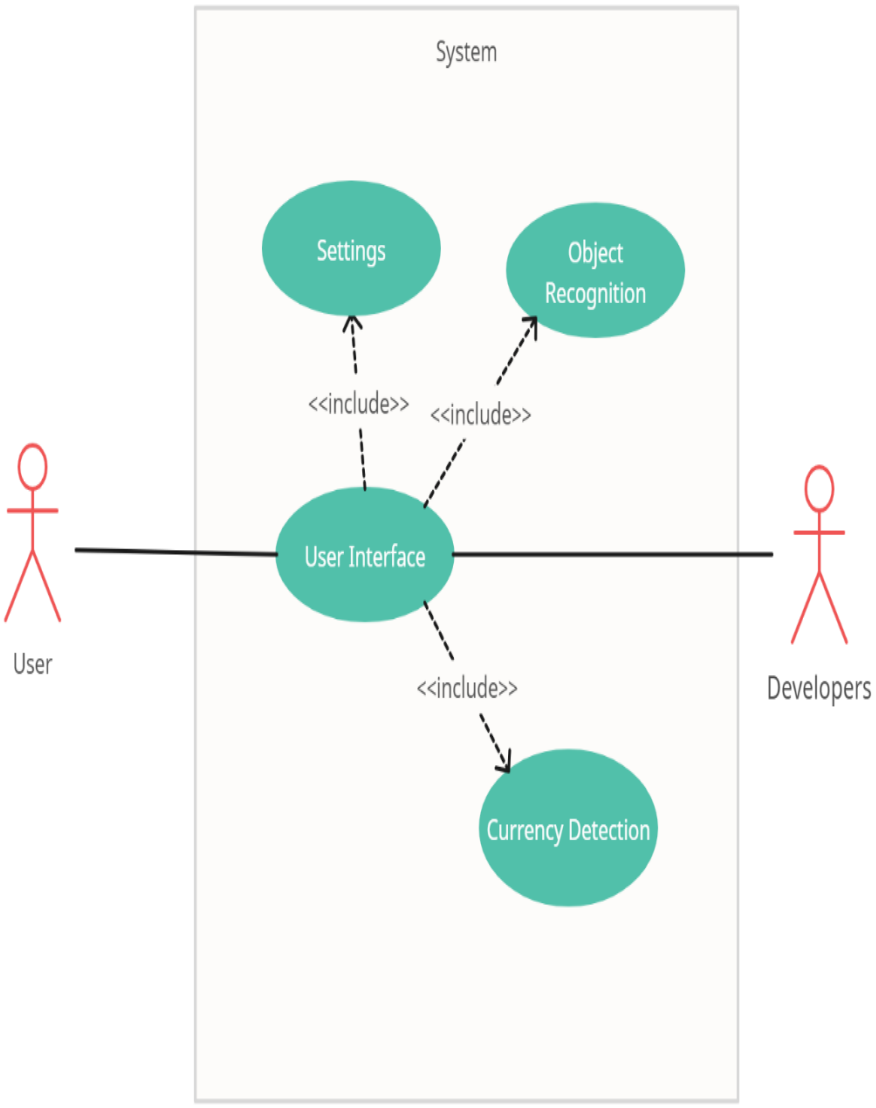
1. Actors: Actors are entities that interact with the system. They can be users, other systems, or external entities. Actors are represented as stick figures or icons on the diagram.

2. Use Cases: Use cases represent specific functionalities or tasks that the system needs to perform. They describe the interactions between actors and the system to accomplish a particular goal. Use cases are depicted as ovals or ellipses on the diagram.

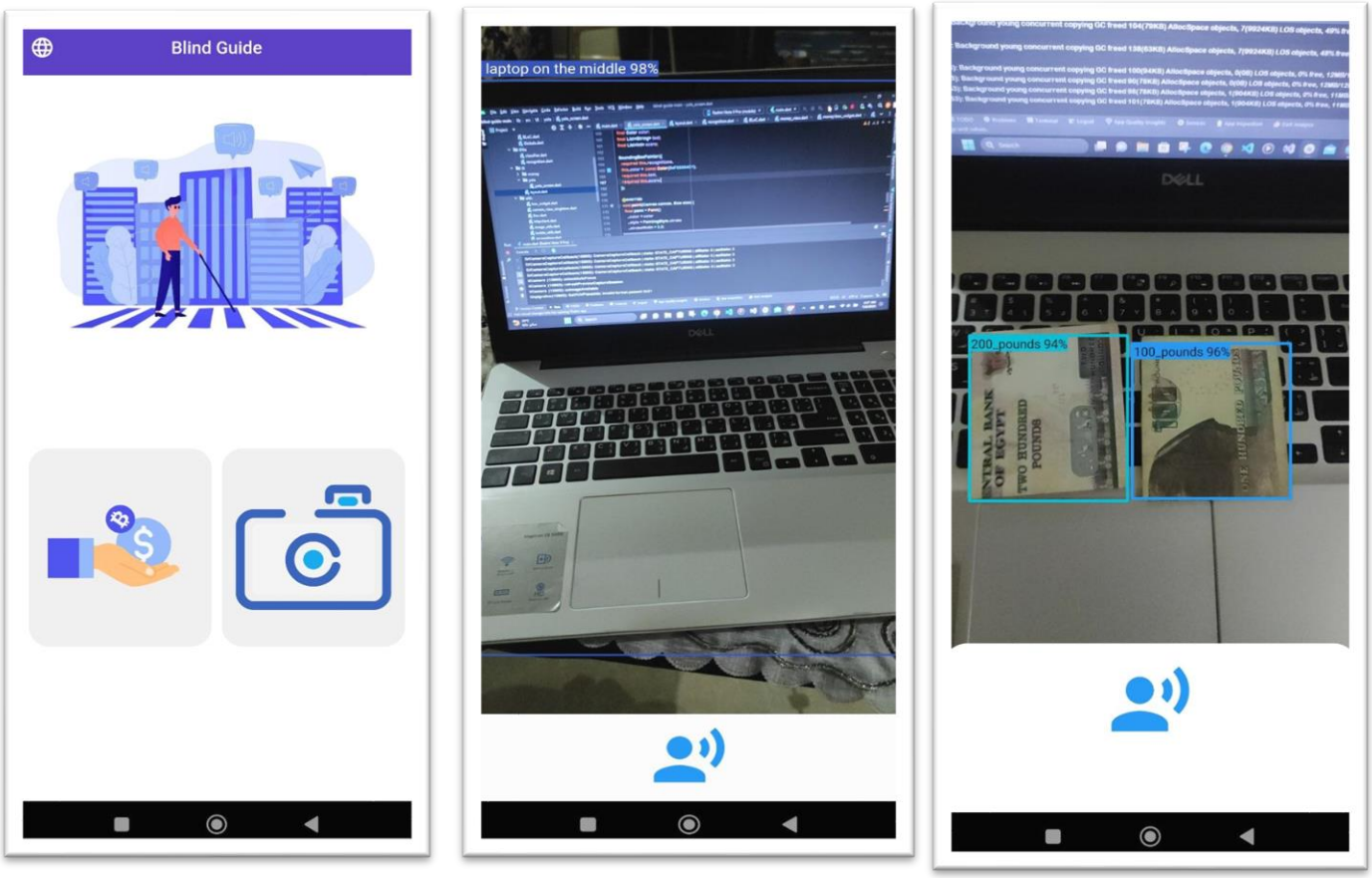
3. Relationships: Relationships between actors and use cases are shown using lines or connectors. The main relationship is the "association" or "communication" between an actor and a use case, indicating that the actor is involved in performing that use case. Additional relationships include "extends" and "includes," which show optional or alternative flows within the use cases.

Use case diagrams provide an overview of the system's behavior and help stakeholders understand the high-level requirements and interactions. They are often used in the early stages of software development to facilitate discussions and clarify the functional requirements before moving into detailed design and implementation.

Use case diagram for blind guide application.



Application Design



Chapter 4: Methodology and Implementation

4.1 Project Management Methodology

4.1.1 What Is a Project Management Methodology?

A project management methodology is a set of principles, tools and techniques that are used to plan, execute, and manage projects. Project management methodologies help project managers lead team members and manage work while facilitating team collaboration. There are many different project management methodologies, and they all have pros and cons. Some of them work better in particular industries or projects, so you'll need to learn about project management methodologies to decide which one works best for our project.

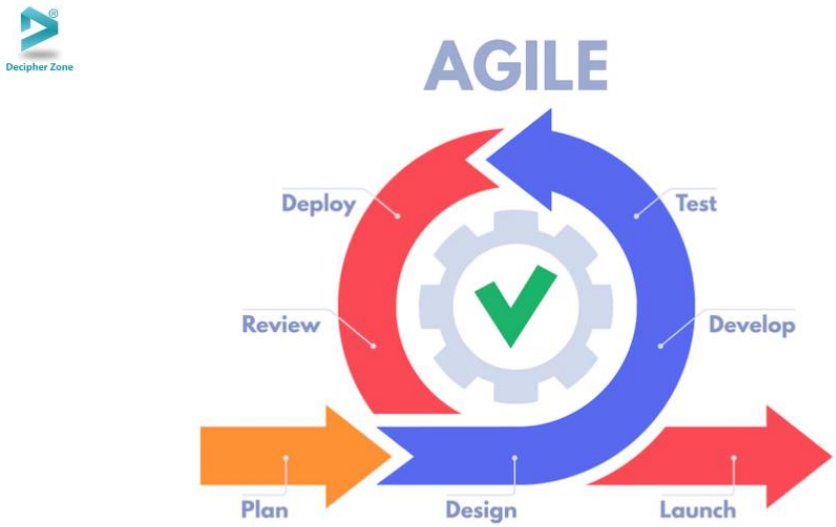
4.1.2 How do you choose the right project management methodology?

There are lots of factors that will impact which project management methodology is right for your project, team, and organization. Here's a quick breakdown of some of the key considerations that can help you decide:

- **Cost and budget**: On a scale of $ to $$$, what sort of budget are you working with? Is there room for that to change if necessary, or is it essential that it stays within these predetermined limits?
- **Team size**: How many people are involved? How many stakeholders? Is your team relatively compact and self-organizing, or more sprawling, with a need for more rigorous delegation?
- **Ability to take risks**: Is this a huge project with a big impact that needs to be carefully managed to deliver Very Serious Results? Or is it a smaller-scale project with a bit more room to play around?
- **Flexibility**: Is there room for the scope of the project to change during the process? What about the finished product?
- **Timeline**: How much time is allotted to deliver the brief? Do you need a quick turnaround, or is it more important that you have a beautifully finished result, no matter how long it takes?
- **Client/stakeholder collaboration**: How involved does the client/stakeholder need or want to be in the process? How involved do you need or want them to be?

4.1.3 Selected Methodology

When we come to select the SDLC methodology that we will use in our project. First, we started with knowing about different methodologies of SDLC. And on



the other side, we wrote documents to be clear requirements, business priorities, our technical capacity and capability, and technological limitations. Then, we started a comparison between different methodologies of SDLC. After comparison, we start a discussion to decide what we will choose. In the end, we decided to choose agile.

- **Agile methodology**
  is a project management framework that breaks projects down into several dynamic phases, commonly known as sprints. It is an iterative methodology. After every sprint, teams reflect and look back to see if there was anything that could be improved so they can adjust their strategy for the next sprint.

  Now we will explain the different stages of agile.

- **Concept**

  The first phase of the agile development lifecycle is the concept. In this phase, the stakeholder will determine the objective and scope of the software. A document that outlines the key requirements of the product will be created by the product owner. This document will also determine the time required to complete this job as well as business opportunities. However, it is advisable to keep minimum requirements as they can be included in the later phases. This will help in deciding whether the product is feasible or not.

- **Inception or Requirement Identification**

  After outlining the concept, the next step is to create the software development team. The product owner will verify the availability of co-developers and choose the best people for the project's success. The product owner will also provide the chosen front-end and back-end developers with the required resources and tools. Then the team can design the mock-up for the user interface while creating the architecture for the project. In this stage, the stakeholders are again involved to gain all the requirements on a diagram and determine the functionality of the project. Moreover, regular checks will help to understand whether all the requirements are incorporated into the design process or not.

- **Iteration or Development**

  The third phase of the agile development process is development or iteration. This one is the longest of all as it carries all the bulk work. Now the development team will start working on combining all the requirements of the product gathered in the concept and inception phase. It goes under several reviews and revisions for improvement until finalized.

  In the development or iteration phase, the following steps are accomplished:

  - The association of the team with clients.
  - Iterations and functionalities are prioritized and implemented.

- Each sprint/iteration should be closely examined and developed.
- Delivering regular working software releases.
- Ensuring product quality by testing at regular intervals.

- **Release**

  Before releasing the product, the quality assurance team runs some tests that assure the functionality of the software. The QA team members will test the software to make sure that the code is clean and if there are any potential errors or bugs, they are addressed by the development team rapidly.

  This phase also supports regular releases and improvement through user feedback. The users and clients are trained on how to use the software. And once all of this is complete, the product is released into production.

- **Maintenance**

  Now the software is completely deployed and accessible by the customers. So the product comes into the maintenance phase. In this phase, the development team provides regular support to the customer to ensure that the software runs smoothly without any bugs. Over time, new iterations can happen to make the existing product more convenient and useful for the customers.

- **Retirement**

  The retirement of a product can be triggered by either new software being introduced, or by a system having become outdated or inconsistent with the organization after some time. Users will first be notified that the software is being retired. When the system is replaced, users will be transferred to the latest system. Last but not least, the developers will conduct the remaining activities and discontinue supporting the old software. Agile includes multiple iterations to refine deliverables and achieve great results.

Why Agile?

Well executed Agile software development methodology helps teams significantly improve the quality of their software at each release. Not only that, it allows teams to adapt to change quickly.
We will discuss some advantages of agile:

- Reduces Technical Debt
- Easily and Quickly Adapt to Change
- Using Agile for Mobile Application Development and Testing Creates Total Alignment and Transparency
- Agile Software Development and Test Minimize Risk
- Higher Quality Product
- Predictable Delivery Dates

- **Application of chosen methodology**

Feasibility study:

We developed our project to help blind people to detect money and things while walking.

- First, we finished the yolo model.
- Then, connected it with flutter through http and tested it.
- After that, we made a sprint to discuss the last deliverables and search for the best solutions.
- As result of our meeting, we tried using WebSocket to enhance the performance, we tested it and decided to use it.
- We repeated the previous steps with the currency model, but we connected model with flutter using tflite plugin.
- Finally, we will move to design user interface, and we changed it many times to improve usability as much as possible.

4.2 Implementation

process of putting a project plan into action to produce the deliverables, otherwise known as the products or services, for clients or stakeholders. The deliverables include all of the products or services that you and your team are performing for the client, customer, or sponsor, including all the project management documents that you put together.

4.2.1 Deploying YOLOv8 model

The step of deploying YOLOv8 model aims to utilize the YOLOv8 model for real-time object detection and provide the results through a WebSocket connection. The deployment is done on Google Colab, and the FastAPI framework is used to create the web application.

Technologies Used:

- **Google Colab**: Google Colab is a cloud-based platform provided by Google that allows users to write, execute, and collaborate on Python code using Jupyter notebooks. It is particularly useful for working with machine learning models and data analysis tasks. Here's a brief introduction to Google Colab and its advantages for using ML models:
  - **Cloud-Based Environment**: Google Colab runs in the cloud, which means you don't have to worry about setting up and managing your own infrastructure. It provides you with a ready-to-use environment with all the necessary dependencies pre-installed, including popular ML libraries like TensorFlow, PyTorch, and scikit-learn. This eliminates the need for local machine setup and makes it easier to get started with ML projects.
  - **Free GPU and TPU access**: Google Colab offer free access to GPUs (Graphical Processing Units) and TPUs (Tensor Processing Units), which are powerful hardware accelerators commonly used in deep learning tasks. This allows you to train and run ML models much faster compared to running them on a CPU alone. The availability of free GPU/TPU

resources is a significant advantage for ML practitioners who may not have access to high-performance hardware.

- **Notebook Collaboration**: Google Colab enables collaborative work on Jupyter notebooks. Multiple users can work together on the same notebook simultaneously, making it easier to collaborate on ML projects with teammates or colleagues. You can share your notebooks with others, allowing them to view and edit the code in real-time.
- **Easy Access to Data**: Colab provides integration with Google Drive, allowing you to easily upload and access data files directly from your Google Drive storage. You can mount your Drive within Colab and read/write data seamlessly. This is particularly useful for working with large datasets or when collaborating with others who can share their data via Google Drive.
- **Flexible Hardware and Runtime options**: Google Colab allows you to select different hardware options, including CPU, GPU, and TPU, depending on your specific needs. You can also choose different runtime environments and specifications, enabling you to customize the resources allocated to your notebooks.
- **Code Snippet Execution and Iterative Development**: With Colab, you can execute code snippets independently, which makes it convenient for iterative development and experimentation. You can modify and rerun specific code sections without rerunning the entire notebook, saving time, and allowing for quick prototyping and debugging.
- **Rich Documentation and Examples**: Colab provides rich documentation and example notebooks that cover various ML topics and techniques. These resources can help you learn new concepts, explore different ML models, and get hands-on experience with real-world datasets.
- **Integration with Google Services**: Google Colab integrates seamlessly with other Google services, such as Big Query, Google Sheets, and Google

Cloud Storage. This allows you to easily import data from these services, analyze it, and perform ML tasks without having to switch between different tools.

- **FastAPI:** FastAPI is a modern, high-performance web framework for building APIs with Python. It is known for its speed, simplicity, and scalability. FastAPI is built on top of the Starlette framework and leverages the power of Python 3.7+ type hints to provide a highly efficient, asynchronous, and statically typed approach to building web applications.

  Here are some key advantages of FastAPI:

  - **High Performance**: FastAPI is built on top of asynchronous programming concepts, utilizing the capabilities of Python's async and await syntax. This allows it to handle many concurrent requests with high efficiency.
  - **Easy to Use**: FastAPI is designed to be developer-friendly and easy to use. It provides a clean and intuitive syntax that leverages Python's type hints to enable automatic data validation, serialization, and documentation generation. This makes it straightforward to build robust and self-documenting APIs.
  - **Automatic Documentation**: FastAPI automatically generates interactive API documentation based on the provided type hints, making it easy to understand and explore the API endpoints. The generated documentation includes request/response models, data types, and example requests and responses. This feature greatly simplifies the process of documenting your API.
  - **Built-in Security**: FastAPI includes built-in security features such as OAuth2 authentication and JWT (JSON Web Tokens) support, making it easier to implement secure authentication and authorization mechanisms in your API.

- **Extensible**: FastAPI provides extensibility through its middleware system, allowing you to add custom functionality, such as request/response modifications, logging, or error handling, to your API.

- **OpenCV**: OpenCV (Open-Source Computer Vision) is a widely used open-source library for computer vision and image processing tasks. It provides a comprehensive set of tools and algorithms to help developers and researchers work with images and videos. Here's a brief introduction to OpenCV and its advantages:

  - **Rich Functionality**: OpenCV offers a vast collection of functions and algorithms for various computer vision tasks, including image/video processing, object detection and tracking, feature extraction, camera calibration, and more. It provides a wide range of tools and techniques to analyze and manipulate images and videos efficiently.

  - **Cross Platform and Open Source**: OpenCV is open-source and supports multiple platforms, including Windows, Linux, macOS, Android, and iOS. This cross-platform compatibility makes it easy to develop computer vision applications that can run on different operating systems and devices.

  - **Efficient and Optimized Algorithms**: OpenCV is known for its optimized implementations of various computer vision algorithms. It leverages the capabilities of modern CPUs and GPUs to achieve high-performance image processing. Many critical operations in OpenCV are implemented using highly optimized C/C++ code, allowing for efficient computation and real-time processing of images and videos.

  - **Machine Learning Integration**: OpenCV integrates well with machine learning libraries and frameworks. It provides support for training and utilizing machine learning models for various tasks, including object detection, image classification, and face recognition. OpenCV's machine learning modules, combined with its image processing capabilities, make it a powerful tool for computer vision and machine learning applications.

- **Ngrok**: Ngrok is a powerful tool that creates secure tunnels from the public internet to your local machine, allowing you to expose local servers to the internet. Here's a brief introduction to Ngrok and its advantages:
  - **Secure Tunneling**: Ngrok creates a secure tunnel between your local server and the internet. It encrypts the data transferred between the two endpoints, ensuring the privacy and security of your communications.
  - **Exposing Local Services**: With Ngrok, you can easily expose web applications, APIs, or any other services running on your local machine to the public internet. This enables you to share your local services with others without deploying them to a remote server.
  - **Simple and Easy-to-Use**: Ngrok provides a simple command-line interface (CLI) that allows you to create and manage tunnels with just a few commands. It has a minimal learning curve and is accessible to developers of all skill levels.

Code Explanation:

```
def DivideImageIntoBox(H, W):
    XAxis = np.linspace(0, W, 4, dtype=np.int32)
    YAxis = np.linspace(0, H, 4, dtype=np.int32)
    b1 = np.array((XAxis[0], YAxis[0], XAxis[1], YAxis[1]))  # TL
    b2 = np.array((XAxis[1], YAxis[0], XAxis[2], YAxis[1]))  # T
    b3 = np.array((XAxis[2], YAxis[0], XAxis[3], YAxis[1]))  # TR
    b4 = np.array((XAxis[0], YAxis[1], XAxis[1], YAxis[2]))  # L
    b5 = np.array((XAxis[1], YAxis[1], XAxis[2], YAxis[2]))  # C
    b6 = np.array((XAxis[2], YAxis[1], XAxis[3], YAxis[2]))  # R
    b7 = np.array((XAxis[0], YAxis[2], XAxis[1], YAxis[3]))  # BL
    b8 = np.array((XAxis[1], YAxis[2], XAxis[2], YAxis[3]))  # B
    b9 = np.array((XAxis[2], YAxis[2], XAxis[3], YAxis[3]))  # BR
    BB = np.array((b1, b2, b3, b4, b5, b6, b7, b8, b9))
    return BB
```

- This function **DivideImageIntoBox (H, W)** takes the height (H) and width (W) of an image as input and divides the image into 9 smaller boxes by creating bounding boxes for each box's top left (TL), top (T), top right (TR), left (L), center (C), right (R), bottom left (BL), bottom (B), and bottom right (BR) corners. These boxes are stored in an array called BB and then returned.

```
def iou(box1, box2):
    (box1_x1, box1_y1, box1_x2, box1_y2) = box1
    (box2_x1, box2_y1, box2_x2, box2_y2) = box2
    xi1 = max(box1[0], box2[0])
    yi1 = max(box1[1], box2[1])
    xi2 = min(box1[2], box2[2])
    yi2 = min(box1[3], box2[3])
    inter_width = max((xi2 - xi1), 0)
    inter_height = max((yi2 - yi1), 0)
    inter_area = inter_height * inter_width
    box1_area = (box1[3] - box1[1]) * (box1[2] - box1[0])
    box2_area = (box2[3] - box2[1]) * (box2[2] - box2[0])
    union_area = (box1_area + box2_area) - inter_area
    iou = inter_area / union_area
    return iou
```

- This function **iou (box1, box2)** calculates the intersection over union (IoU) between two bounding boxes. It takes two bounding boxes as input, represented as (box_x1, box_y1, box_x2, box_y2), where (box_x1, box_y1) represents the top left corner and (box_x2, box_y2) represents the bottom right corner. It calculates the IoU value by computing the intersection area, union area, and dividing the intersection area by the union area.

```
def Text(image, Box, cls):
    side = [
        "on the top left",
        "on the top",
        "on the top right",
        "on the left",
        "on the middle",
        "on the right",
        "on the bottom left",
        "on the bottom",
        "on the bottom right",
    ]
    label = {
        0: "__backgrond__",
        1: "person",
        ...
        80: "toothbrsh",
    }
    H, W = image.shape[:2]
    BB = DivideImageIntoBox(H, W)
    IOU = []
    text = []
    c = 0
    for box in Box:
        for b in BB:
            IOU.append(iou(box, b))
        text.append("a " + label[cls[c] + 1] + " " + side[np.argmax(np
        c += 1
        IOU = []
    return text
```

- This function **Text (image, Box, cls)** generates descriptive text based on the image, bounding boxes (Box), and class labels (cls) provided. It first defines a list side containing descriptions of the location of objects relative to the image, and a dictionary label that maps class labels to their corresponding names.
- It then retrieves the height (H) and width (W) of the image.

- The function calls **DivideImageIntoBox (H, W)** to obtain the array of bounding boxes (BB) for the image.
- Inside a loop over each box in the provided Box list, it calculates the IoU between the box and each bounding box in BB using the iou () function, storing the results in the IOU list.
- It then appends a descriptive text to the text list, combining the class label, the highest IoU's corresponding side description, and the string "a".
- Finally, it returns the text list.

```
app = FastAPI()
```

- This creates an instance of the FastAPI framework, which is used to build the web application.

```
model = YOLO("yolov8x.pt")
```

- This line initializes a YOLO model by creating an instance of the YOLO class and loading the model weights from the file "yolov8x.pt".

```
origins = ["*"]
```

- This line creates a list of allowed origins for CORS (Cross-Origin Resource Sharing), which determines which domains are allowed to access the application's resources. In this case, the asterisk (*) allows requests from any origin.

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

- This adds the CORS middleware to the FastAPI application, enabling Cross-Origin Resource Sharing. It specifies the allowed origins, credentials, methods, and headers.

```
@app.websocket("/detect-image/")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    print("Connection accepted")
    cnt = 0

    try:
        while True:
            encoded_image = await websocket.receive_text()
            decoded_image = base64.b64decode(encoded_image)
            img = cv2.imdecode(np.frombuffer(decoded_image, np.uint8), cv2
            conf = 0.7
            device = "cpu"
            results = model(img, conf=conf, device=device)

            boxes_list = results[0].boxes.xyxyn.numpy().tolist()
            scores_list = results[0].boxes.conf.numpy().tolist()
            classes_list = results[0].boxes.cls.numpy().tolist()

            text = Text(img, results[0].boxes.xyxy, classes_list)
            print("Count:", cnt)
            cnt += 1

            await websocket.send_json({
                "text": text,
                "boxes_list": boxes_list,
                "scores_list": scores_list,
                "classes_list": classes_list,
            })

            print("Sent")

    except WebSocketDisconnect:
        print("Connection closed")
```

- `@app.websocket("/detect-image/")` This decorator specifies the URL path (`"/detect-image/"`) at which the WebSocket endpoint is accessible.
- `async def websocket_endpoint (websocket: WebSocket)` This is the function definition. It takes a WebSocket object as a parameter, representing the connection between the server and the client.
- `await websocket.accept()` This line accepts the WebSocket connection from the client.
- `encoded_image = await websocket.receive_text()` This line asynchronously receives the encoded image from the client as a text string.
- `decoded_image = base64.b64decode(encoded_image)` This line decodes the received text string using base64 decoding, converting it back into its original binary form
- `img=cv2.imdecode(np.frombuffer(decoded_image,np.uint8),cv2.IMREAD_COLOR)` This line decodes the binary image data using OpenCV's imdecode function, creating an image object that can be processed further.
- `results = model (img, conf=conf, device=device)` This line performs object detection on the image using a pre-trained model. The model object takes the image, confidence threshold, and device as input and returns the detection results.
- `boxes_list = results[0].boxes.xyxyn.numpy().tolist()` This line extracts the bounding box coordinates (x, y, width, height) for the detected objects and converts them to a Python list.
- `scores_list=results[0].boxes.conf.numpy().tolist()` This line extracts the confidence scores for the detected objects and converts them to a Python list.
- `classes_list = results[0].boxes.cls.numpy().tolist()` This line extracts the class labels for the detected objects and converts them to a Python list.

- `text = Text(img, results[0].boxes.xyxy, classes_list)` This line creates a text representation of the detected objects using the Text function (not provided in the code snippet).
- `print("counting", cnt):` This line prints the current count value to the server console.
- `await websocket.send_json({...})` This line sends a JSON object containing the detection results (text, bounding box coordinates, confidence scores, and class labels) back to the client using the WebSocket connection.

```
import ngrok
import nest_asyncio
import uvicorn

ngrok.set_auth_token("2MIaqSJO90GgAOBa7pidLOcErQx_2k73bV5wT51t2JEvDYb8w")
public_url = ngrok.connect(8000).public_url
print(public_url)

nest_asyncio.apply()
uvicorn.run(app, port=8000)
```

- `ngrok.set_auth_token("2MIaqSJO90GgAOBa7pidLOcErQx_2k73bV5wT51t2JEvDYb8w")` This line sets the authentication token for ngrok, a tool that creates secure tunnels from public URLs to a local server. The token is used to authenticate your access to ngrok services.
- `public_url = ngrok.connect(8000).public_url` This line establishes a connection to ngrok on port 8000. It creates a tunnel between your local server and a public URL provided by ngrok. The. public URL attribute retrieves the public URL generated by ngrok for accessing your local server.
- `print(public_url)` This line prints the public URL obtained from ngrok to the console. You can use this URL to access your local server from the internet.

- `nest_asyncio.apply()` This line is used to patch asyncio to work correctly with nested or threaded applications. It allows asyncio to be used within Jupyter notebooks or other environments where the event loop is already running.
- `uvicorn.run(app, port=8000)` This line starts the Uvicorn server, which is an ASGI (Asynchronous Server Gateway Interface) server. It runs the app object, which represents your application, on port 8000. Uvicorn serves the application, making it accessible through the local port specified.

4.2.2 Implementation of currency model

The implementation of the model goes through some stages:

1. **Data Gathering:**

   Before we start training, we need to gather and label image (using labelme) that will be used for training the object detection model. The training images should have random objects in the image along with the desired objects and should have a variety of backgrounds and lighting conditions.

2. **Upload Dataset and Prepare Dataset:**

   After we finish gathering data and label the images, we start to upload the dataset and prepare dataset:

   1) We connect the drive first then unzip the folder that have two folders images and the JSON files. In addition, we import the packages.

```
#Importing dataset from Google Drive in Colab
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#to unzip a folder
!unzip -q /content/drive/MyDrive/Blind_Guide/currency_detector_data/all_data_3.zip -d /content
```

```
# Import packages
import os
import json
import cv2
import albumentations as alb
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import shutil
import random
import sys
import uuid
import tensorflow as tf
import IPython.display as display
%matplotlib inline
```

2) We starts with defining a dictionary called LABELS, which maps certain floating-point values to class labels (shortcut names if shortcut=True, otherwise the original class labels). The modify_on_json_files function takes one argument jsonPath, which is the path to the directory containing the JSON files to be modified. The function iterates through each JSON file in the specified directory. For each JSON file, it loads the data from the file and extracts the filename without the extension (name variable). It initializes a new dictionary dic to store the modified data. For each shape (object) in the JSON file, the function calculates the bounding box coordinates (x_min, y_min, x_max, y_max) in a format suitable for the SSD model. The bounding box coordinates are normalized to fit a 320x320-image size. The bounding box coordinates and the class label (mapped to shortcut name if shortcut=True) are added to the dic dictionary under the 'object' key. Once all objects are processed for a JSON file, the modified data is converted back to a JSON format using json.dumps and written back to the same file, overwriting its contents. If any exception occurs during processing, it prints the error message. After processing all JSON files in the specified directory, it prints a success message indicating that the JSON files have been modified.

```python
shortcut=True # We used shortcut names to classes, if you didn't, make shortcut = False
LABELS={0.25:'25_piasters',0.50:'50_piasters',
           1:'1_pound',5:'5_pounds',10:'10_pounds',
           20:'20_pounds',50:'50_pounds',
           100:'100_pounds',200:'200_pounds'}
def modify_on_json_files(jsonPath):
    ''' modify files to make it suitable for SSD model (x_min,y_min),(x_max,y_max) '''
    print('modify on json files function is running ....')
    for file in os.listdir(os.path.join(jsonPath)):
        j=open(os.path.join(jsonPath,file))
        df=json.load(j)#load json file
        name=file.split('.')[0]
        dic={'object':[]}
        dic['image']=f'{name}.jpg'
        try:
            for i in range(len(df['shapes'])):
                w,h=df['imageWidth'],df['imageHeight']
                x1=max(0,min(df['shapes'][i]['points'][0][0],df['shapes'][i]['points'][1][0])/w)*320
                y1=max(0,min(df['shapes'][i]['points'][0][1],df['shapes'][i]['points'][1][1])/h)*320
                x2=min(1,max(df['shapes'][i]['points'][0][0],df['shapes'][i]['points'][1][0])/w)*320
                y2=min(1,max(df['shapes'][i]['points'][0][1],df['shapes'][i]['points'][1][1])/h)*320
                obj={}
                obj['bbox']=[x1,y1,x2,y2]
                if shortcut:
                    obj['class']=LABELS[float(df['shapes'][i]['label'])]
                else:
                    obj['class']=df['shapes'][i]['label']
                dic['object'].append(obj)
            json_object=json.dumps(dic)
            with open(os.path.join(jsonPath,file), "w") as outfile:
                outfile.write(json_object)
        except Exception as e:
            print(e)
    print(f'Successfully modified json files in path : \'{jsonPath}\'')
```

3) We add some function to help us to preprocess the data:
    - resizeImages (path, shape This function resizes images located in a specified path to a target shape of (320, 320) using OpenCV's cv2.resize() function. The resized images are saved with the same file-name in the original path.

```
def resizeImages(path,shape):
    '''resize image to (320,320)'''
    print('Resize images function is running....')
    for image in os.listdir(path):
        img=cv2.imread(os.path.join(path,image))
        if img is not None and img.size != 0:
            resizedImage=cv2.resize(img,shape, interpolation = cv2.INTER_AREA)
            cv2.imwrite(os.path.join(path,f'{image.split(".")[0]}.jpg'),resizedImage)
    print(f'Successfully resized images in path : \'{path}\'')
```

- split (images_path, labels_path, ratio): This function splits the data into train and validation sets. It creates two new directories, 'train' and 'val', and moves a specified ratio of images from the input images_path to the 'train' folder and the remaining images to the 'val' folder. Corresponding labels (JSON files) are also moved to the respective train and val label directories.

```
def spilt(images_path,labels_path,ratio):
    '''split data into train, val, and test folders'''
    print('split function is running ....')
    for folder in ['train','val']:
        os.makedirs(os.path.join('data',folder,'images'))
        os.makedirs(os.path.join('data',folder,'labels'))
    train_path=os.path.join('data','train')
    val_path=os.path.join('data','val')
    images_list=os.listdir(images_path)
    random.seed(10)
    random.shuffle(images_list)
    train_size=int(len(images_list)*ratio)
    train_list=images_list[:train_size]
    val_list=images_list[train_size:]
    print(f'Train data size : {len(train_list)}\nValidation data size : {len(val_list)}')
    for i in train_list:
        shutil.move(os.path.join(images_path,i),
                    os.path.join(train_path,'images',i))
        shutil.move(os.path.join(labels_path,f'{i.split(".")[0]}.json'),
                    os.path.join(train_path,'labels',f'{i.split(".")[0]}.json'))
    for i in val_list:
        shutil.move(os.path.join(images_path,i),
                    os.path.join(val_path,'images',i))
        shutil.move(os.path.join(labels_path,f'{i.split(".")[0]}.json'),
                    os.path.join(val_path,'labels',f'{i.split(".")[0]}.json'))
    print('Successfully split data')
```

- augmentation(crop_size, images_path, labels_path, augmentation_num, augmented_images_path, augmented_labels_path, image_shape): This function applies data augmentation techniques to images and their corresponding labels. It utilizes the Albumentations library for transformations such as random cropping, horizontal/vertical flipping, brightness/contrast adjustment, gamma correction, and RGB shift. The augmented images and labels are saved in the specified output paths.

```
def augmentation(crop_size,images_path,labels_path,augmentation_num,augmented_images_path,augmented_labels_path,image_shape):
    print('Augmentation function is running ....')
    aumentor=alb.Compose([
        alb.RandomCrop(*crop_size),
        alb.HorizontalFlip(p=0.5),
        alb.RandomBrightnessContrast(p=0.2),
        alb.RandomGamma(p=0.2),
        alb.RGBShift(p=0.2),
        alb.VerticalFlip(p=0.5)],
        bbox_params=alb.BboxParams(format='albumentations',label_fields=['class_labels']))
    if not os.path.exists(augmented_images_path):
        os.makedirs(augmented_images_path)
    if not os.path.exists(augmented_labels_path):
        os.makedirs(augmented_labels_path)
    for image in os.listdir(images_path):
        img=cv2.imread(os.path.join(images_path,image))
        label_path=os.path.join(labels_path,f'{image.split(".")[0]}.json')
        with open(label_path,'r') as f:
            label=json.load(f)
        l,c=[],[]
        for i in range(len(label['object'])):
            coords=[0,0,0,0]
            coords[0]=label['object'][i]['bbox'][0]/image_shape[0]
            coords[1]=label['object'][i]['bbox'][1]/image_shape[1]
            coords[2]=label['object'][i]['bbox'][2]/image_shape[0]
            coords[3]=label['object'][i]['bbox'][3]/image_shape[1]
            Label=label['object'][i]['class']
            l.append(Label)
            c.append(coords)
```

```
    try:
        for x in range(augmentation_num):
            name=str(uuid.uuid4())
            name=name.replace('-','')
            augmented=aumentor(image=img,bboxes=c,class_labels=l)
            resized_image=cv2.resize(augmented['image'],image_shape, interpolation = cv2.INTER_AREA)
            cv2.imwrite(os.path.join(augmented_images_path,f'{name}.jpg'),resized_image)
            dic={'object':[]}
            dic['image']=f'{name}.jpg'
            for i in range(len(label['object'])):
                annotation={}
                annotation['bbox']=''
                annotation['class']=''
                if len(augmented['bboxes'])==0:
                    annotation['bbox']=[0,0,0,0]
                    annotation['class']='EOF'
                else:
                    annotation['class']=augmented['class_labels'][i]
                    annotation['bbox']=[int(p*320) for p in augmented['bboxes'][i]]
                dic['object'].append( annotation)
            with open (os.path.join(augmented_labels_path,f'{name}.json'),'w') as f:
                json.dump(dic,f)
    except Exception as e:
        print(e,'---> file path:',os.path.join(images_path,image))
print(f'Successfully augmented data in path  : \'{augmented_images_path}\' and \'{augmented_labels_path}\'')
```

- json_to_csv(label_path, csv_path) This function converts JSON files (labels) to a CSV file format. It reads each JSON file, extracts the necessary information such as image filename, class, and bounding box coordinates, and appends them to a list. Finally, it converts the list to a panda DataFrame and saves it as a CSV file.
- Each function prints informative messages to indicate the progress and completion of the respective task.

```
#convert json files to csv file
def json_to_csv(label_path,csv_path):
    print('convert json files to csv file function is running ....')
    json_list=[]
    for file in os.listdir(os.path.join(label_path)):
        x=open(os.path.join(label_path,file))
        data=json.load(x)
        for i in range(len(data['object'])):
            value = (data['image'],320,320,data['object'][i]['class'],
                     data['object'][i]['bbox'][0],
                     data['object'][i]['bbox'][1],
                     data['object'][i]['bbox'][2],
                     data['object'][i]['bbox'][3])
            json_list.append(value)

    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    json_df = pd.DataFrame(json_list, columns=column_name)
    json_df.to_csv(csv_path, index=None)
    print(f'Successfully converted json to csv. in path {csv_path}')
```

- view_data(images_path, labels_path, num_test_images, image_shape): This function randomly selects a specified number of images from the dataset and displays them along with their corresponding bounding boxes and labels. It reads the image and label information from the given paths, applies color coding to the bounding boxes and labels, and visualizes the annotated images using OpenCV and Matplotlib.

```
color={'25_piasters':(0,0,128),'50_piasters':(0,128,128),
          '1_pound':(128,0,128),'5_pounds':(0,128,0),
          '10_pounds':(128,128,0),'20_pounds':(128,0,0),
          '50_pounds':(47,79,79),'100_pounds':(153,50,204),
      '200_pounds':(199,21,133),'EOF':(255,255,255)}
def view_data(images_path,labels_path,num_test_images,image_shape):
    images=os.listdir(images_path)
    random.seed(10)
    images_to_test = random.sample(images, num_test_images)

    for image_path in images_to_test:
        with open(os.path.join(labels_path,f'{image_path.split(".")[0]}.json'),'r') as f:
            label=json.load(f)
        image=cv2.imread(os.path.join(images_path,image_path))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        for i in range(len(label['object'])):
            xmin=int(label['object'][i]['bbox'][0])
            ymin=int(label['object'][i]['bbox'][1])
            xmax=int(label['object'][i]['bbox'][2])
            ymax=int(label['object'][i]['bbox'][3])

            cv2.rectangle(image,(xmin,ymin), (xmax,ymax),color[label['object'][i]['class']], 1)
            p1,p2=(xmin,ymin),(xmax,ymax)
            lw = max(round(sum(image.shape) / 2 * 0.003), 2)
            tf = max(lw - 1, 1)
            Label=label['object'][i]['class']
            w, h = cv2.getTextSize(Label, 0, fontScale=lw / 3, thickness=tf)[0]  # text width, height
            outside = p1[1] - h >= 3
            p2 = p1[0] + w, p1[1] - h - 3 if outside else p1[1] + h + 3
            cv2.rectangle(image, p1, p2, color[label['object'][i]['class']], -1, cv2.LINE_AA)  # filled
            cv2.putText(image,Label, (p1[0], p1[1] - 2 if outside else p1[1] + h + 2),0,lw / 3,(255, 255, 255),thickness=tf,lineType=cv2.LINE_AA)
        plt.figure(figsize=(8,8))
        plt.imshow(image)
        plt.axis('off')
        plt.show()
```

- Prepare_Dataset(images_path, labels_path, image_shape, augmentation_num=0, ratio=0.8): This function prepares the dataset for model training and evaluation. Finally, the code creates a directory named 'csv' and generates CSV files for the train and validation sets. The value of the data variable is determined based on the augmentation_num parameter.

```
def Prepare_Dataset(images_path,labels_path,image_shape,augmentation_num=0,ratio=0.8):
    modify_on_json_files(labels_path)
    resizeImages(images_path,image_shape)
    spilt(images_path,
          labels_path,
          ratio)
    data='data'
    if augmentation_num:
      data='augmented_data'
      for folder in ['train','val']:
          Images_path=os.path.join('data',folder,'images')
          Labels_path=os.path.join('data',folder,'labels')
          augmented_images_path=os.path.join('augmented_data',folder,'images')
          augmented_labels_path=os.path.join('augmented_data',folder,'labels')
          crop_size=(300,300)
          augmentation(crop_size,
                       Images_path,
                       Labels_path,
                       augmentation_num,
                       augmented_images_path,
                       augmented_labels_path,
                       image_shape)



    os.makedirs('csv')
    for folder in ['train','val']:
        print(os.path.join(data,folder,'labels'))
        Labels_path=os.path.join(data,folder,'labels')
        csv_path=os.path.join('csv',f'{folder}.csv')
        json_to_csv(Labels_path,csv_path)
```

```
images_path=r'/content/all_data_3/images'
labels_path=r'/content/all_data_3/labels'
image_shape=(320,320)
augmentation_num=2
Prepare_Dataset(images_path,labels_path,image_shape,augmentation_num,0.8)

data ='augmented_data' if augmentation_num else'data'
print(data)
```

4) Now we perform some operations on a pandas DataFrame named “train” that was read from the 'train.csv' file:
   - `train.drop(train.index[(train["class"] == "0")], axis=0, inplace=True)` This line drops the rows from the DataFrame where the value of the "class" column is equal to "0".
   - `train.to_csv(r'/content/csv/train.csv', index=None)`: This line saves the modified DataFrame back to the 'train.csv' file, overwriting the previous version.

- `train['class'].value_counts()` This line counts the occurrences of each unique value in the "class" column of the DataFrame. It returns a series with the class labels as the index and their respective counts as the values.

```
train=pd.read_csv(r'/content/csv/train.csv')
train.drop(train.index[(train["class"] == "0")],axis=0,inplace=True)
train.to_csv(r'/content/csv/train.csv', index=None)
train['class'].value_counts()
```

```
20_pounds      1215
100_pounds     1193
50_pounds      1183
10_pounds      1174
200_pounds     1154
5_pounds       1118
50_piasters    1081
25_piasters    1007
1_pound         995
Name: class, dtype: int64
```

```
train.describe()
```

	width	height	xmin	ymin	xmax	ymax
count	10120.0	10120.0	10120.000000	10120.000000	10120.000000	10120.000000
mean	320.0	320.0	84.966601	87.152174	232.749407	231.486660
std	0.0	0.0	51.932167	58.620075	53.059104	59.351241
min	320.0	320.0	0.000000	0.000000	0.000000	1.000000
25%	320.0	320.0	46.000000	40.000000	198.000000	194.000000
50%	320.0	320.0	82.000000	86.000000	237.000000	233.000000
75%	320.0	320.0	119.000000	124.000000	271.000000	277.000000
max	320.0	320.0	317.000000	316.000000	320.000000	320.000000

- `train.describe()` This line generates descriptive statistics for the numerical columns in the DataFrame, such as count, mean, standard deviation, minimum, quartiles, and maximum.

5) Similarly, we also perform some operations on pandas DataFrame named "val" which is read from file "val.csv":

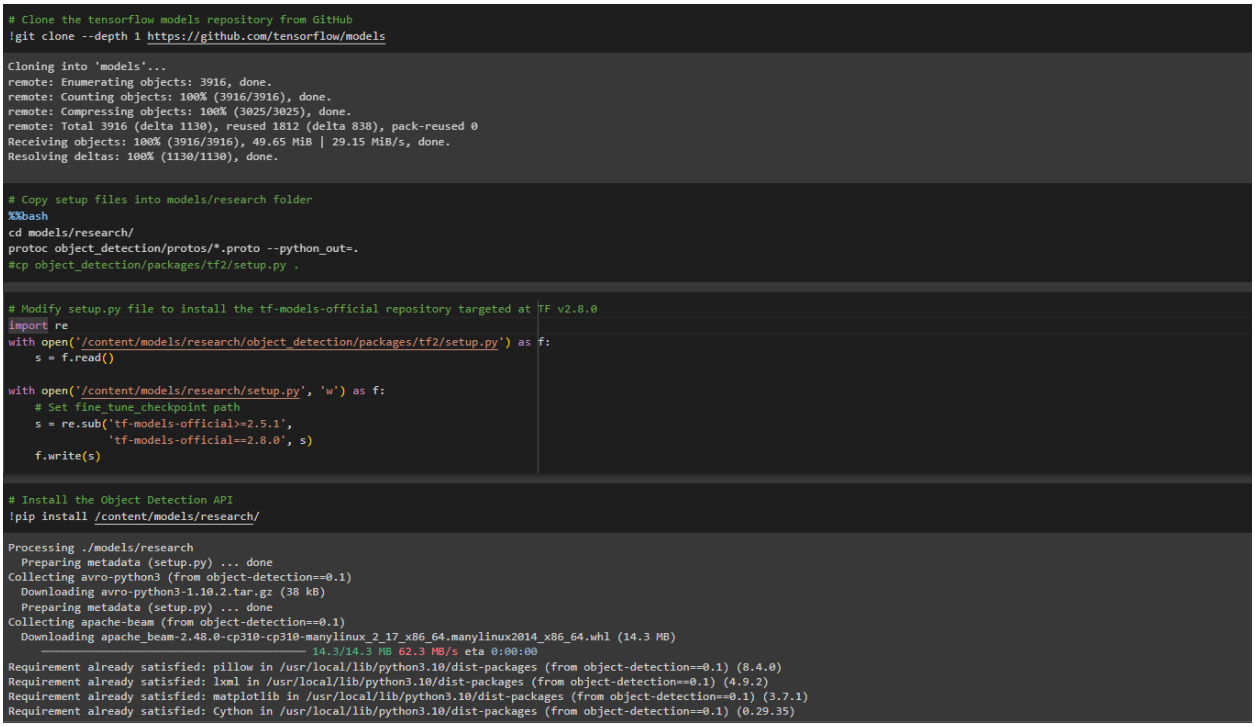
```
val=pd.read_csv(r'/content/csv/val.csv')
val.drop(val.index[(val["class"] == "0")],axis=0,inplace=True)
val.to_csv(r'/content/csv/val.csv', index=None)
val['class'].value_counts()
```

```
1_pound         339
10_pounds       330
100_pounds      316
50_piasters     308
5_pounds        292
200_pounds      292
50_pounds       280
25_piasters     276
20_pounds       242
Name: class, dtype: int64
```

```
val.describe()
```

	width	height	xmin	ymin	xmax	ymax
count	2675.0	2675.0	2675.000000	2675.000000	2675.000000	2675.000000
mean	320.0	320.0	89.685981	92.551028	228.012336	227.161121
std	0.0	0.0	55.361578	62.764176	55.352804	62.112637
min	320.0	320.0	0.000000	0.000000	2.000000	3.000000
25%	320.0	320.0	48.000000	46.500000	192.000000	190.000000
50%	320.0	320.0	86.000000	90.000000	230.000000	229.000000
75%	320.0	320.0	126.000000	130.000000	267.000000	274.000000
max	320.0	320.0	315.000000	316.000000	320.000000	320.000000

3. Install TensorFlow Object Detection Dependencies



```
# Clone the tensorflow models repository from GitHub
!git clone --depth 1 https://github.com/tensorflow/models
```

```
Cloning into 'models'...
remote: Enumerating objects: 3916, done.
remote: Counting objects: 100% (3916/3916), done.
remote: Compressing objects: 100% (3025/3025), done.
remote: Total 3916 (delta 1130), reused 1812 (delta 838), pack-reused 0
Receiving objects: 100% (3916/3916), 49.65 MiB | 29.15 MiB/s, done.
Resolving deltas: 100% (1130/1130), done.
```

```
# Copy setup files into models/research folder
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
#cp object_detection/packages/tf2/setup.py .
```

```
# Modify setup.py file to install the tf-models-official repository targeted at TF v2.8.0
import re
with open('/content/models/research/object_detection/packages/tf2/setup.py') as f:
    s = f.read()

with open('/content/models/research/setup.py', 'w') as f:
    # Set fine_tune_checkpoint path
    s = re.sub('tf-models-official>=2.5.1',
               'tf-models-official==2.8.0', s)
    f.write(s)
```

```
# Install the Object Detection API
!pip install /content/models/research/
```

```
Processing ./models/research
  Preparing metadata (setup.py) ... done
Collecting avro-python3 (from object-detection==0.1)
  Downloading avro-python3-1.10.2.tar.gz (38 kB)
  Preparing metadata (setup.py) ... done
Collecting apache-beam (from object-detection==0.1)
  Downloading apache_beam-2.48.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (14.3 MB)
     ---------------------------------------- 14.3/14.3 MB 62.3 MB/s eta 0:00:00
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (8.4.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (4.9.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (3.7.1)
Requirement already satisfied: Cython in /usr/local/lib/python3.10/dist-packages (from object-detection==0.1) (0.29.35)
```

4. Create TFRecords

This code creates a "currency.txt" file that contains a list of classes for object detection. It then downloads a script called "create_tfrecord.py" and uses it to generate TensorFlow record files (.tfrecord) for the training and validation datasets.

```
### This creates a a "currency.txt" file with a list of classes the object detection model will detect.
%%bash
cat <<EOF >> /content/currency.txt
25_piasters
50_piasters
1_pound
5_pounds
10_pounds
20_pounds
50_pounds
100_pounds
200_pounds
EOF
```

- !wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/create_tfrecord.py: This line downloads the "create_tfrecord.py" script from the provided URL.

```
[ ] ! wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/create_tfrecord.py

    --2023-07-02 08:06:27--  https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/create_tfrecord.py
    Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
    Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 4414 (4.3K) [text/plain]
    Saving to: 'create_tfrecord.py'

    create_tfrecord.py  100%[===================>]   4.31K  --.-KB/s    in 0s

    2023-07-02 08:06:27 (61.2 MB/s) - 'create_tfrecord.py' saved [4414/4414]
```

- Conditional block:
  - If augmentation_num is non-zero, the script is executed twice with different arguments to generate TensorFlow record files for augmented data.
  - --csv_input=/content/csv/train.csv: Specifies the path to the CSV file containing training data.
  - --labelmap=/content/currency.txt: Specifies the path to the label map file.
  - --image_dir=/content/augmented_data/train/images: Specifies the directory containing the augmented training images.
  - --output_path=train.tfrecord: Specifies the path to save the generated training TensorFlow record file.

- The same process is repeated for the validation dataset, generating "val.tfrecord".
- If augmentation_num is zero, the script is executed similarly for the original data without augmentation.

```
if augmentation_num:
  !python create_tfrecord.py --csv_input=/content/csv/train.csv --labelmap=/content/currency.txt --image_dir=/content/augmented_data/train/images --output_path=train.tfrecord
  !python create_tfrecord.py --csv_input=/content/csv/val.csv --labelmap=/content/currency.txt --image_dir=/content/augmented_data/val/images --output_path=val.tfrecord
else:
   !python create_tfrecord.py --csv_input=/content/csv/train.csv --labelmap=/content/currency.txt --image_dir=/content/data/train/images --output_path=train.tfrecord
   !python create_tfrecord.py --csv_input=/content/csv/val.csv --labelmap=/content/currency.txt --image_dir=/content/data/val/images --output_path=val.tfrecord

Successfully created the TFRecords: /content/train.tfrecord
Successfully created the TFRecords: /content/val.tfrecord
```

- train_record_fname, val_record_fname, and label_map_pbtxt_fname are assigned the respective paths for the generated TensorFlow record files and the label map file.

```
train_record_fname = '/content/train.tfrecord'
val_record_fname = '/content/val.tfrecord'
label_map_pbtxt_fname = '/content/labelmap.pbtxt'
```

5. Set Up Training Configuration

In this section, we'll set up the model and training configuration. We will specify which pretrained TensorFlow model we want to use from the TensorFlow 2 Object Detection Model Zoo. Each model also comes with a configuration file that points to file locations, sets training parameters (such as learning rate and total number of training steps), and more. We will modify the configuration file for our custom training job.

1) Define the chosen model architecture:

- chosen_model: Specifies the chosen model architecture, set to 'ssd-mobilenet-v2-fpnlite-320' in this example.
- model_name: Specifies the name of the model, set to 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'.

- pretrained_checkpoint: Specifies the name of the pre-trained checkpoint file, set to 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'.
- base_pipeline_file: Specifies the base pipeline configuration file, set to 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config'.

```
# Change the chosen_model variable to deploy different models available in the TF2 object detection zoo
chosen_model = 'ssd-mobilenet-v2-fpnlite-320'
model_name = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
pretrained_checkpoint = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
base_pipeline_file = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config'
```

2) Create a folder named "mymodel" to hold pre-trained weights and configuration files and navigate to that directory.

```
# Create "mymodel" folder for holding pre-trained weights and configuration files
%mkdir /content/models/mymodel/
%cd /content/models/mymodel/
```

3) Download the pre-trained model weights:

- The pre-trained weights are downloaded from the specified URL.
- The downloaded tar file is extracted.

```
# Download pre-trained model weights
import tarfile
download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' + pretrained_checkpoint
!wget {download_tar}
tar = tarfile.open(pretrained_checkpoint)
tar.extractall()
tar.close()
```

4) Download the training configuration file for the model:
   - The training configuration file is downloaded from the specified URL.

```
# Download training configuration file for model
download_config = 'https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/configs/tf2/' + base_pipeline_file
!wget {download_config}
```

5) Set training parameters:
   - num_steps: Specifies the number of training steps, set to 35000 in this example.
   - batch_size: Specifies the batch size, set to 16.

```
# Set training parameters for the model
num_steps = 35000
batch_size = 16
```

6) Set file locations and get the number of classes from the label map file.

```
# Set file locations and get number of classes for config file
pipeline_fname = '/content/models/mymodel/' + base_pipeline_file
fine_tune_checkpoint = '/content/models/mymodel/' + model_name + '/checkpoint/ckpt-0'

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())
num_classes = get_num_classes(label_map_pbtxt_fname)
print('Total classes:', num_classes)
```

7) Create a custom configuration file by modifying the base pipeline file:
   - Open the base pipeline file.
   - Replace placeholders in the pipeline file with the actual paths and values:
     - Set the fine-tune checkpoint path.
     - Set the paths for the training and validation TensorFlow record files.
     - Set the path for the label map file.
     - Set the batch size.
     - Set the number of training steps.
     - Set the number of classes.
     - Change the fine-tune checkpoint type to "detection" instead of "classification".

- Write the modified configuration to a new file named "pipeline_file.config".

```
# Create custom configuration file by writing the dataset, model checkpoint, and training parameters into the base pipeline file
import re

%cd /content/models/mymodel
print('writing custom configuration file')

with open(pipeline_fname) as f:
    s = f.read()
with open('pipeline_file.config', 'w') as f:

    # Set fine_tune_checkpoint path
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # Set tfrecord files for train and test datasets
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/train)(.*?")', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub(
        '(input_path: ".*?)(PATH_TO_BE_CONFIGURED/val)(.*?")', 'input_path: "{}"'.format(val_record_fname), s)

    # Set label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Set batch_size
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)

    # Change fine-tune checkpoint type from "classification" to "detection"
    s = re.sub(
        'fine_tune_checkpoint_type: "classification"', 'fine_tune_checkpoint_type: "{}"'.format('detection'), s)

    # If using ssd-mobilenet-v2, reduce learning rate (because it's too high in the default config file)
    if chosen_model == 'ssd-mobilenet-v2':
      s = re.sub('learning_rate_base: .8',
                 'learning_rate_base: .08', s)

      s = re.sub('warmup_learning_rate: 0.13333',
                 'warmup_learning_rate: .026666', s)
    f.write(s)
```

8) Set the path to the custom config file and the directory to store training checkpoints.

```
# Set the path to the custom config file and the directory to store training checkpoints in
pipeline_file = '/content/models/mymodel/pipeline_file.config'
model_dir = '/content/training/'
```

6. Train Custom TFLite Detection Model

1) This code will make us able to access TensorBoard in your Jupyter Notebook and visualize the training progress of your object detection model.

2) The code runs the training process for the object detection model using the model_main_tf2.py script provided by TensorFlow.
   - !python: Executes the Python interpreter.
   - /content/models/research/object_detection/model_main_tf2.py: The path to the model_main_tf2.py script, which is responsible for running the training process.
   - --pipeline_config_path={pipeline_file}: Specifies the path to the custom configuration file generated earlier.
   - --model_dir={model_dir}: Specifies the directory where the training checkpoints and logs will be saved.

- --alsologtostderr: Outputs log messages to stderr in addition to log files.
- --num_train_steps={num_steps}: Sets the number of training steps to be performed during the training process.
- --sample_1_of_n_eval_examples=1: Specifies that every 1 out of n evaluation examples should be used during evaluation. In this case, n is set to 1, meaning that all evaluation examples will be used.

```
#@title
# Run training!
!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1
```

7. Convert Model To tensorFlow Lite

1) The code creates a directory to store the trained TFLite model and then uses the export_tflite_graph_tf2.py script provided by TensorFlow to export the model as a TFLite graph.
   - !mkdir /content/custom_model_lite: Creates a directory to store the trained TFLite model.
   - output_directory = '/content/custom_model_lite': Sets the path to the output directory where the TFLite model will be saved.
   - last_model_path = '/content/training': Sets the path to the training directory where the last checkpoint of the trained model is located.
   - /content/models/research/object_detection/export_tflite_graph_tf2.py: The path to the export_tflite_graph_tf2.py script, which is responsible for exporting the trained model as a TFLite graph.
   - --trained_checkpoint_dir {last_model_path}: Specifies the directory where the trained model checkpoints are stored.

- --output_directory {output_directory}: Specifies the directory where the TFLite model will be saved.
- --pipeline_config_path {pipeline_file}: Specifies the path to the custom configuration file used during training.

```
# Make a directory to store the trained TFLite model
!mkdir /content/custom_model_lite
output_directory = '/content/custom_model_lite'

# Path to training directory (the conversion script automatically chooses the highest checkpoint file)
last_model_path = '/content/training'

!python /content/models/research/object_detection/export_tflite_graph_tf2.py \
    --trained_checkpoint_dir {last_model_path} \
    --output_directory {output_directory} \
    --pipeline_config_path {pipeline_file}
```

2) The code uses TensorFlow's TFLiteConverter to convert the exported graph file into a TFLite model file.

- Converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite/saved_model'): Creates an instance of the TFLiteConverter and loads the exported graph file from the /content/custom_model_lite/saved_model directory.
- tflite_model = converter.convert(): Converts the loaded model into a TFLite model.
- with open('/content/custom_model_lite/detect.tflite', 'wb') as f:: Opens a file in binary write mode to save the TFLite model.
- f.write(tflite_model): Writes the TFLite model to the file.

```
# Convert exported graph file into TFLite model file
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite/saved_model')
tflite_model = converter.convert()

with open('/content/custom_model_lite/detect.tflite', 'wb') as f:
  f.write(tflite_model)
```

8. Test TensorFlow Lite Model

1) The code unzips the file test.zip located in /content/drive/MyDrive/Blind_Guide/currency_detector_data and extracts its contents to the /content directory. The -q flag is used to suppress the output during the extraction process.

```
!unzip -q /content/drive/MyDrive/Blind_Guide/currency_detector_data/test.zip -d /content
```

2) This code is a script that allows you to run a custom TFLite model on test images and display the detection results. Here's a summary of what the code does:
   - It imports necessary packages, including OpenCV, NumPy, TensorFlow Lite, and Matplotlib.
   - It defines a function named tflite_detect_images that takes the model path, image path, label path, minimum confidence threshold, and the number of test images as input.
   - The function loads the label map and the TensorFlow Lite model into memory.
   - It retrieves the input and output details of the model and extracts the height and width of the input image.
   - Randomly selects a specified number of test images from the provided image path.
   - Iterates over each test image and performs object detection using the TensorFlow Lite model.
   - Draws bounding boxes and labels on the detected objects if their confidence scores exceed the minimum threshold.
   - Displays the test image with the detected objects using Matplotlib.
   - Returns the detections, which include the object name, confidence score, and bounding box coordinates.

```
# Import packages
import os
import cv2
import numpy as np
import sys
import glob
import random
import importlib.util
from tensorflow.lite.python.interpreter import Interpreter

import matplotlib
import matplotlib.pyplot as plt

%matplotlib inline

### Define function for inferencing with TFLite model and displaying results

def tflite_detect_images(modelpath, imgpath, lblpath, min_conf=0.5, num_test_images=10):

  # Grab filenames of all images in test folder
  images = glob.glob(imgpath + '/*.jpg') + glob.glob(imgpath + '/*.JPG') + glob.glob(imgpath + '/*.png') + glob.glob(imgpath + '/*.bmp')

  # Load the label map into memory
  with open(lblpath, 'r') as f:
      labels = [line.strip() for line in f.readlines()]

  # Load the Tensorflow Lite model into memory
  interpreter = Interpreter(model_path=modelpath)
  interpreter.allocate_tensors()

  # Get model details
  input_details = interpreter.get_input_details()
  output_details = interpreter.get_output_details()
  height = input_details[0]['shape'][1]
  width = input_details[0]['shape'][2]

  float_input = (input_details[0]['dtype'] == np.float32)

  input_mean = 127.5
  input_std = 127.5

  # Randomly select test images
  images_to_test = random.sample(images, num_test_images)
```

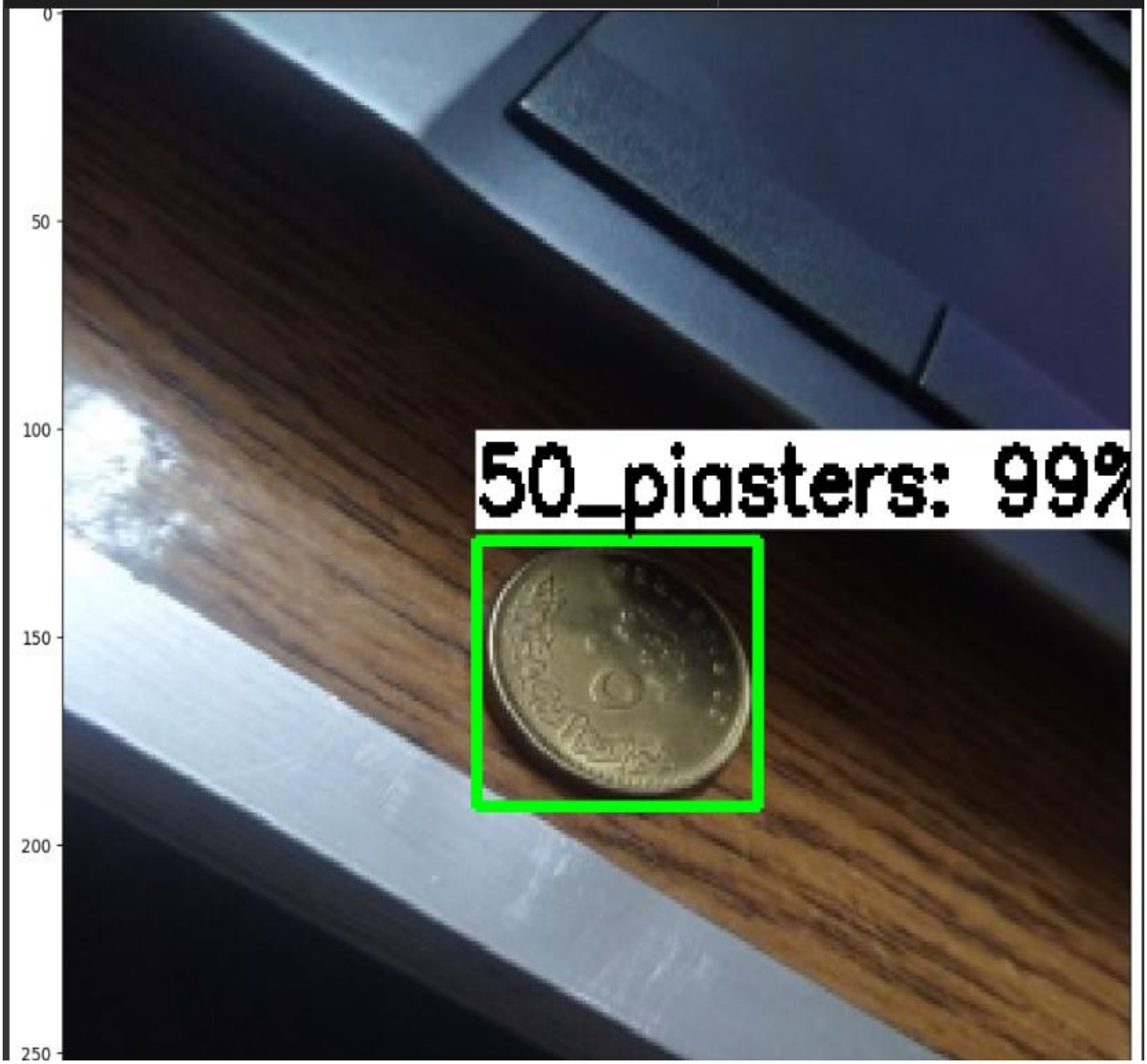
3) The code sets up the necessary variables to run the user's custom TFLite model on test images.

```
# Loop over every image and perform detection
for image_path in images_to_test:
    # Load image and resize to expected shape [1xHxWx3]
    image = cv2.imread(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    imH, imW, _ = image.shape
    image_resized = cv2.resize(image_rgb, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)
    # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
    if float_input:
        input_data = (np.float32(input_data) - input_mean) / input_std
    # Perform the actual detection by running the model with the image as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()
    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[1]['index'])[0] # Bounding box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[3]['index'])[0] # Class index of detected objects
    scores = interpreter.get_tensor(output_details[0]['index'])[0] # Confidence of detected objects
    detections = []
    # Loop over all detections and draw detection box if confidence is above minimum threshold
    for i in range(len(scores)):
        if ((scores[i] > min_conf) and (scores[i] <= 1.0)):
            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))
            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
            # Draw label
            object_name = labels[int(classes[i])] # Look up object name from "labels" array using class index
            label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window
            cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
            cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text
            detections.append([object_name, scores[i], xmin, ymin, xmax, ymax])

    # All the results have been drawn on the image, now display the image
    image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(12,16))
    plt.imshow(image)
    plt.show()

return
```

```
# Set up variables for running user's model
PATH_TO_IMAGES='/content/test'   # Path to test images folder
PATH_TO_MODEL='/content/custom_model_lite/detect.tflite'   # Path to .tflite model file
PATH_TO_LABELS='/content/currency.txt'   # Path to labelmap.txt file
min_conf_threshold=0.5   # Confidence threshold (try changing this to 0.01 if you don't see any detection results)
images_to_test = 10   # Number of images to run detection on
# Run inferencing function!
tflite_detect_images(PATH_TO_MODEL, PATH_TO_IMAGES, PATH_TO_LABELS, min_conf_threshold, images_to_test)
```



9. Download TensorFlow Lite Model

In this part of the code, the label map file (currency.txt), the pipeline configuration file (pipeline_file.config), and the label map file (labelmap.pbtxt) are moved into the TFLite model folder (custom_model_lite). Then, the entire custom_model_lite folder is zipped into a file called custom_model_lite.zip. Finally, the custom_model_lite.zip file is downloaded using the files.download() function, which allows you to download the file from Colab.

```
# Move labelmap and pipeline config files into TFLite model folder and zip it up
!cp /content/currency.txt /content/custom_model_lite
!cp /content/labelmap.pbtxt /content/custom_model_lite
!cp /content/models/mymodel/pipeline_file.config /content/custom_model_lite
%cd /content
!zip -r custom_model_lite.zip custom_model_lite
```

```
/content
  adding: custom_model_lite/ (stored 0%)
  adding: custom_model_lite/detect.tflite (deflated 9%)
  adding: custom_model_lite/labelmap.pbtxt (deflated 71%)
  adding: custom_model_lite/saved_model/ (stored 0%)
  adding: custom_model_lite/saved_model/assets/ (stored 0%)
  adding: custom_model_lite/saved_model/variables/ (stored 0%)
  adding: custom_model_lite/saved_model/variables/variables.data-00000-of-00001 (deflated 9%)
  adding: custom_model_lite/saved_model/variables/variables.index (deflated 78%)
  adding: custom_model_lite/saved_model/saved_model.pb (deflated 91%)
  adding: custom_model_lite/pipeline_file.config (deflated 65%)
  adding: custom_model_lite/currency.txt (deflated 54%)
```

```
from google.colab import files

files.download('/content/custom_model_lite.zip')
```

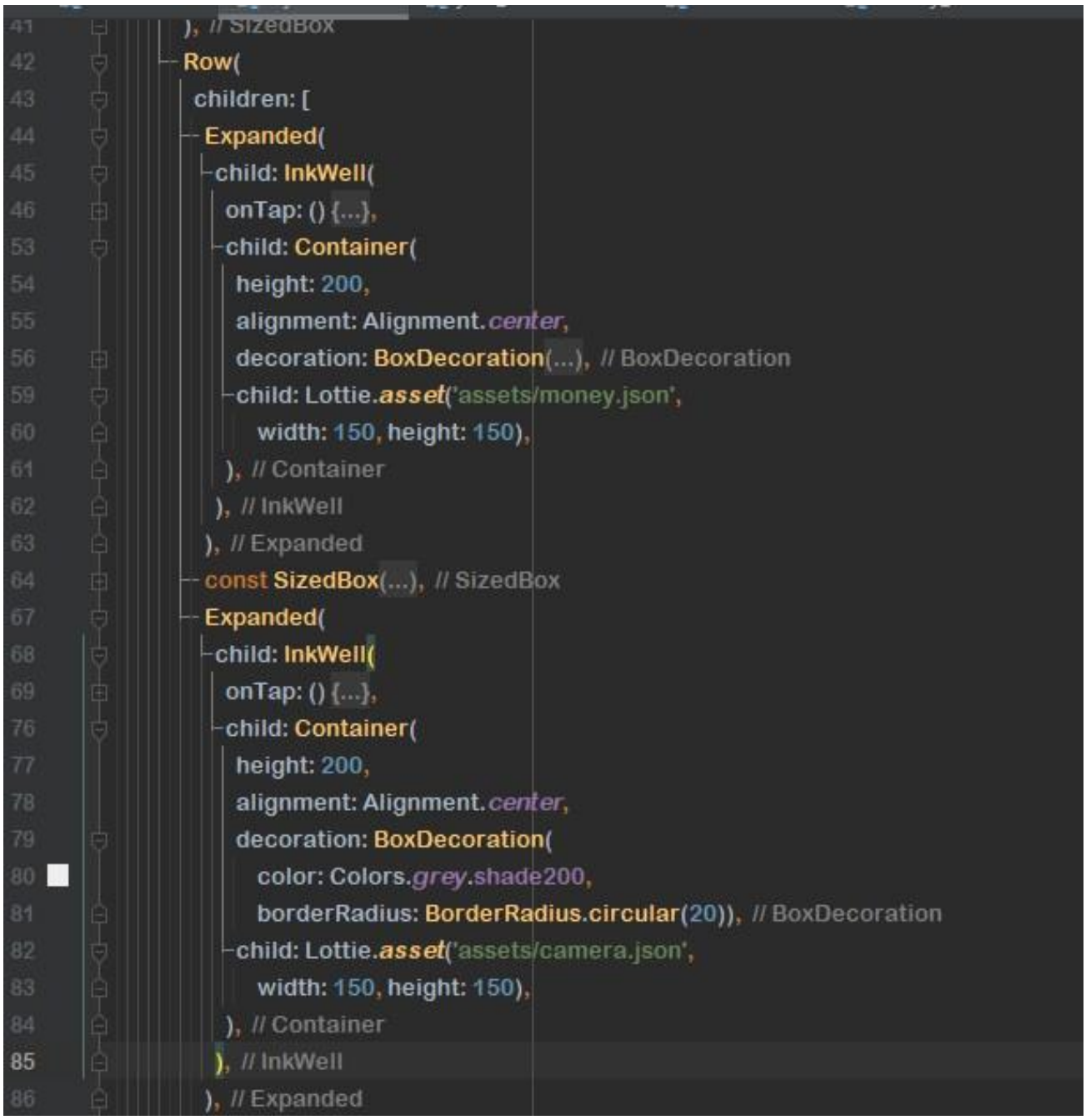
4.2.3 Implementation of Flutter:

Use Flutter to create a cross-platform mobile app that has a simple and accessible user interface.

Flutter is an open-source UI toolkit that allows you to build beautiful, natively compiled applications from a single codebase.

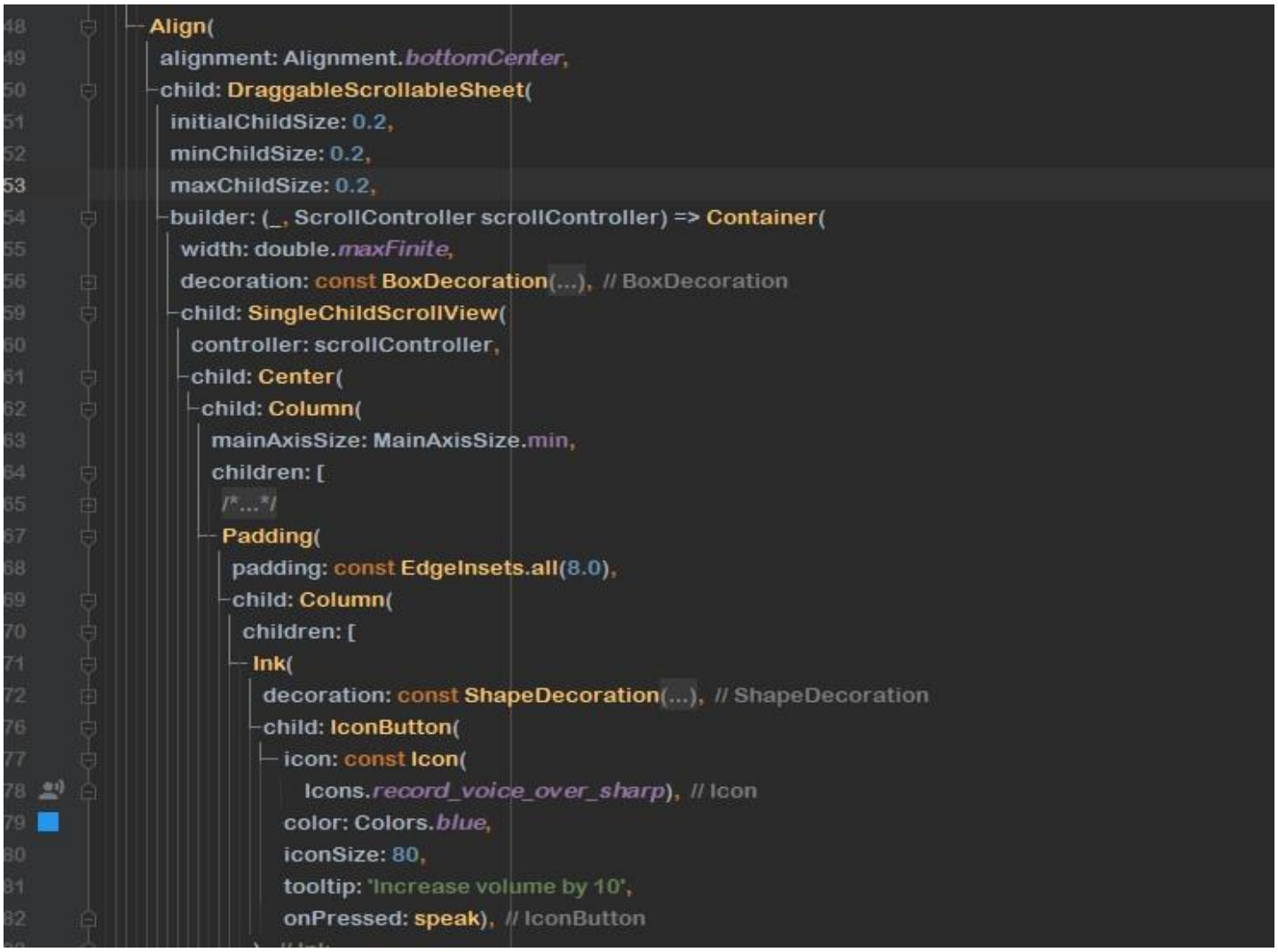
- Create a Flutter project using the command flutter create blind_guide_app.Add the necessary dependencies to the pubspec.yaml file.

- Create Home page
  - Using animations through Lottie package which takes away the complexity from Motion Design. It lets you Create, Edit, Test, Collaborate and Ship a Lottie in the easiest way possible, and this is snippet of code:



```
), // SizedBox
Row(
  children: [
  Expanded(
    child: InkWell(
      onTap: () {...},
      child: Container(
        height: 200,
        alignment: Alignment.center,
        decoration: BoxDecoration(...), // BoxDecoration
        child: Lottie.asset('assets/money.json',
          width: 150, height: 150),
      ), // Container
    ), // InkWell
  ), // Expanded
  const SizedBox(...), // SizedBox
  Expanded(
    child: InkWell(
      onTap: () {...},
      child: Container(
        height: 200,
        alignment: Alignment.center,
        decoration: BoxDecoration(
          color: Colors.grey.shade200,
          borderRadius: BorderRadius.circular(20)), // BoxDecoration
        child: Lottie.asset('assets/camera.json',
          width: 150, height: 150),
      ), // Container
    ), // InkWell
  ), // Expanded
```

- Create a Flutter widget that displays a camera preview by Using the camera plugin to access the camera of the device to capture images in real-time, and a buttonIcon to say the detected object for each model, This is snippet of currency-model screen:



```
Align(
  alignment: Alignment.bottomCenter,
  child: DraggableScrollableSheet(
    initialChildSize: 0.2,
    minChildSize: 0.2,
    maxChildSize: 0.2,
    builder: (_, ScrollController scrollController) => Container(
      width: double.maxFinite,
      decoration: const BoxDecoration(...), // BoxDecoration
      child: SingleChildScrollView(
        controller: scrollController,
        child: Center(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              /*...*/
              Padding(
                padding: const EdgeInsets.all(8.0),
                child: Column(
                  children: [
                    Ink(
                      decoration: const ShapeDecoration(...), // ShapeDecoration
                      child: IconButton(
                        icon: const Icon(
                          Icons.record_voice_over_sharp), // Icon
                        color: Colors.blue,
                        iconSize: 80,
                        tooltip: 'Increase volume by 10',
                        onPressed: speak), // IconButton
```

- Use the WebSocket plugin to connect yolo model with flutter by sending requests contain frames and get responses by list of detected objects and dimensions of bounding boxes.

  - ✓ Note: in the previous code, we use Bloc class to handle all the possible states of your application in an easy way.

- We connected currency model with flutter by using tflite_flutter which provides a flexible and fast solution for accessing TensorFlow Lite interpreter and performing inference.

In the next code we explains how to create interpreter for model, load it and how to deal with inputs and outputs as tensors.

```
class BLoC {
  late List<CameraDescription> cameras;
  final _stateSubject = BehaviorSubject<YoloModel>();

  Stream<YoloModel> get state => _stateSubject.stream;
  //final wsUrl = Uri.parse('wss://b15b-34-90-213-63.ngrok.io/detect-image/');
  var channel;
  postImageData(Uint8List rawCameraPhoto) async {
    channel.sink.add(base64.encode(rawCameraPhoto));
  }

  Future<void> initCameras() async {
    cameras = await availableCameras();
  }
  BLoC() {
    try {
      channel = WebSocketChannel.connect(Uri.parse('wss://18d2-154-182-112-98.ngrok-free.app/detect-image/'));
      channel.stream.listen((message) {
        log("message $message");
        YoloModel yoloModel = YoloModel.fromJson(jsonDecode(message));
        _stateSubject.add(yoloModel);
      });
    } on WebSocketChannelException catch (e) {
      debugPrint("Error at $e");
    } catch (e) {
      debugPrint("Error at $e");
    }
```

```dart
class Classifier {
  /// Instance of Interpreter
  Interpreter? _interpreter;

  /// Labels file loaded as list
  List<String>? _labels;

  static const String MODEL_FILE_NAME = "detect.tflite";
  static const String LABEL_FILE_NAME = "labelmap.txt";

  /// Input size of image (height = width = 300)
  static const int INPUT_SIZE = 320;

  /// Result score threshold
  static const double THRESHOLD = 0.7;

  /// [ImageProcessor] used to pre-process the image
  ImageProcessor? imageProcessor;

  /// Padding the image to transform into square
  late int padSize;

  /// Shapes of output tensors
  late List<List<int>> _outputShapes;

  /// Types of output tensors
  late List<TfLiteType> _outputTypes;
```

➢ program tools

- Android Studio: is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. A unified environment where you can develop for all Android devices. Apply Changes to push code and resource changes to your running app without restarting your app.

- Flutter: is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single code base.
- Jupyter: are a community standard for communicating and performing interactive computing. They are a document that blends computations, outputs, explanatory text, mathematics, images, and rich media representations of objects.

Django: is a high-level Python web framework that enables rapid development of secure and maintainable websites. It is built by experienced developers and follows the "Batteries included" philosophy, meaning it provides almost everything you might need to build a web application out of the box. It also follows the MVT (Model View Template) design pattern, which separates the data, logic, and presentation layers of a web application[12]. Django is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support

Chapter 5: Results and Discussion

In this chapter, we will present the results obtained from the implementation and testing of the "Blind Guide" application. We will discuss the performance of the object detection and currency recognition models, evaluate the localization function, and analyze the overall effectiveness of the application in assisting blind individuals.

5.1 YOLOv8 Performance

Here are a few main reasons why we are using YOLOv8 for project:-

a. YOLOv8 has a high rate of accuracy measured by COCO and Roboflow 100.

b. YOLOv8 comes with a lot of developer-convenience features, from an easy-to-use CLI to a well-structured Python package.

c. There is a large community around YOLO and a growing community around the YOLOv8 model, meaning there are many people in computer vision circles who may be able to assist you when you need guidance.

YOLOv8 COCO Accuracy

COCO (Common Objects in Context) is the industry standard benchmark for evaluating object detection models. When comparing models on COCO, we look at the mAP value and FPS measurement for inference speed. Models should be compared at similar inference speeds.

The image below shows the accuracy of YOLOv8 on COCO, using data collected by the Ultralytics team and published in their YOLOv8 README

▼ Detection

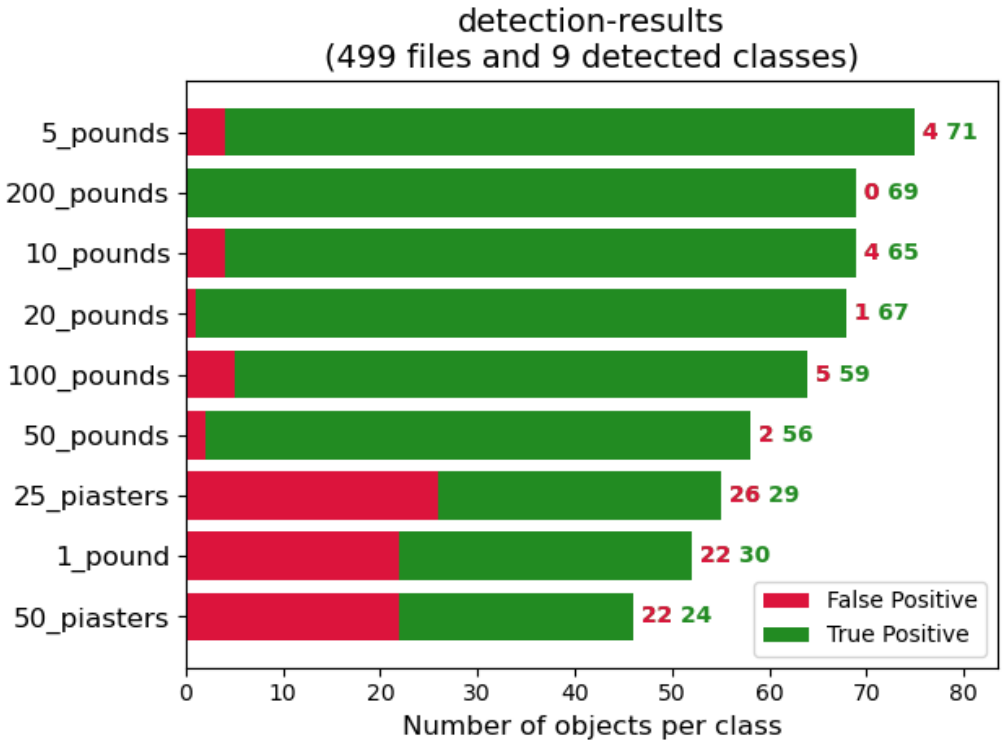
Model	size (pixels)	mAP^{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

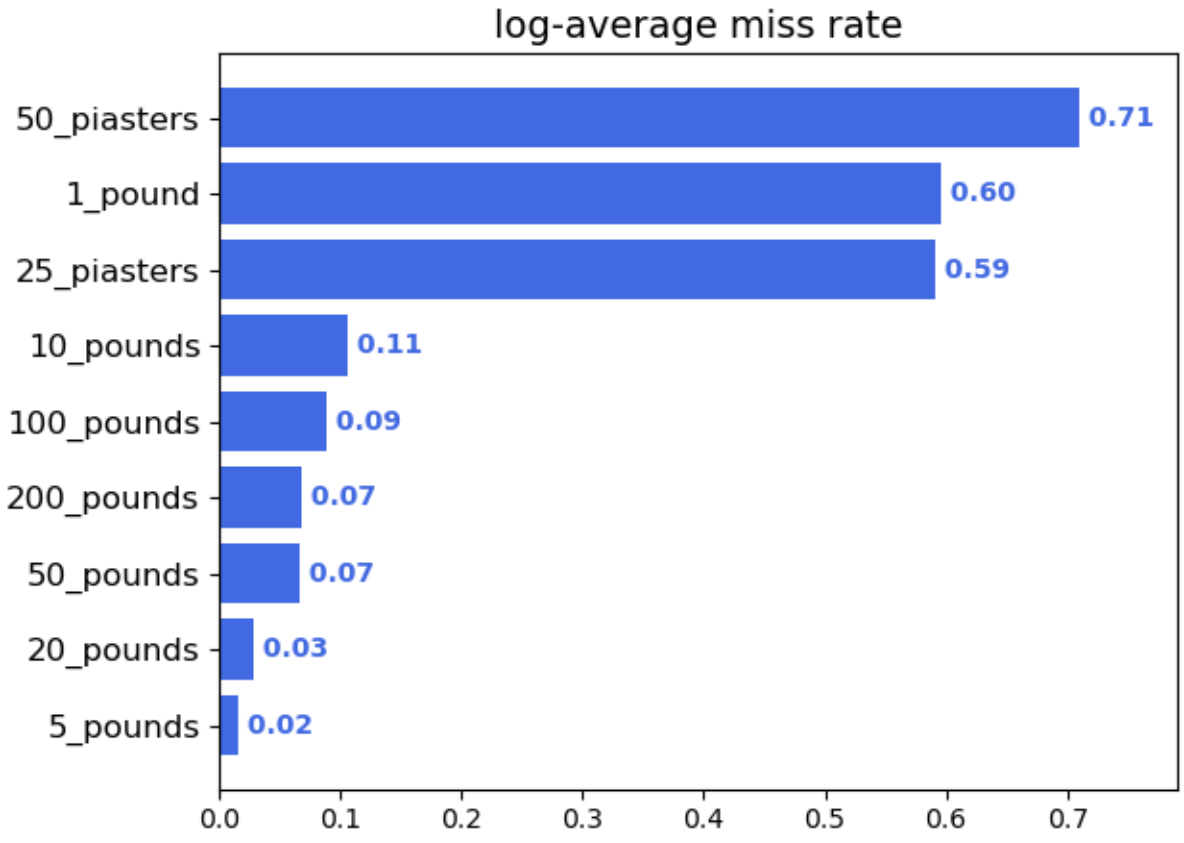
- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo mode=val task=detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo mode=val task=detect data=coco128.yaml batch=1 device=0/cpu`

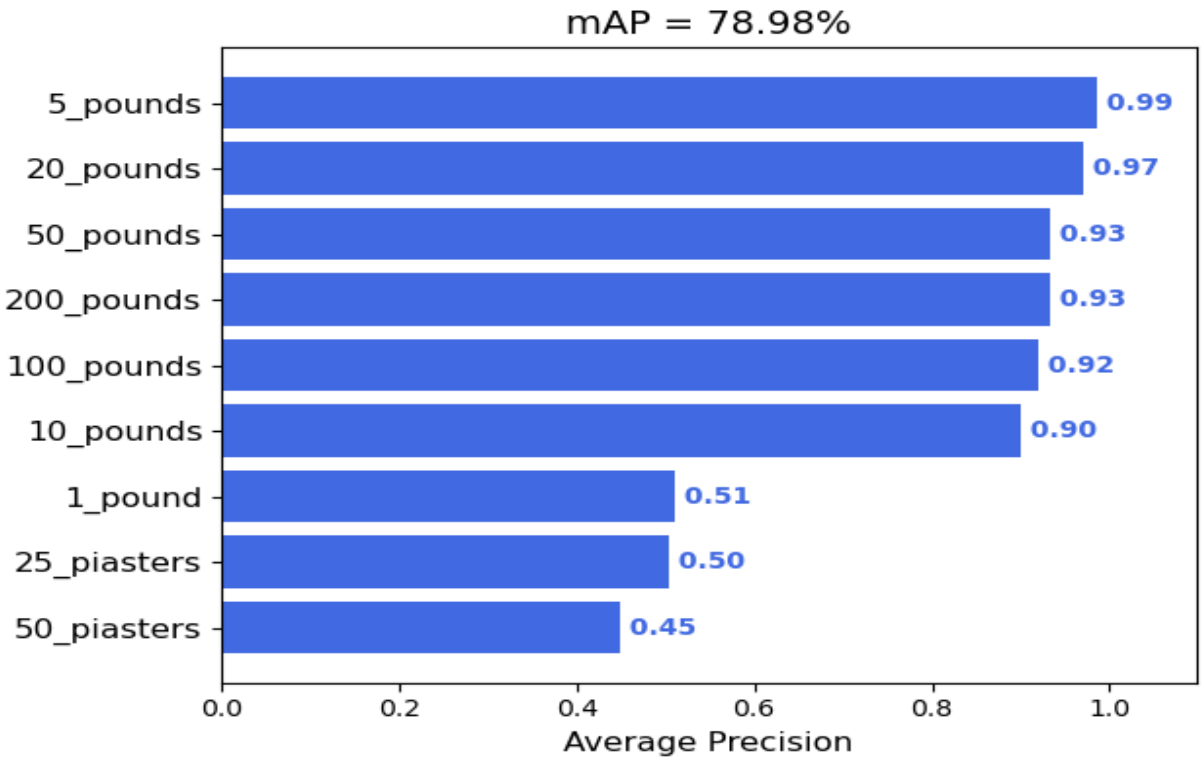
YOLOv8 COCO accuracy is state of the art for models at comparable inference latencies.

5.2 Currency Model Performance.

The SSD-MobileNet-v2-FPNLite-320 model used for currency detection in the Blind Guide Project also yielded excellent results. The model accurately detected and differentiated various denominations of bills and coins. The system achieved a high accuracy rate in recognizing different currencies in real-time scenarios. The performance was evaluated based on precision, recall, and mAP, which indicated high accuracy and reliability in currency recognition.




log-average miss rate
50_piasters
1_pound
25_piasters
10_pounds
100_pounds
200_pounds
50_pounds
20_pounds
5_pounds
0.71
0.60
0.59
0.11
0.09
0.07
0.07
0.03
0.02
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


mAP = 78.98%
5_pounds
20_pounds
50_pounds
200_pounds
100_pounds
10_pounds
1_pound
25_piasters
50_piasters
0.99
0.97
0.93
0.93
0.92
0.90
0.51
0.50
0.45
0.0
0.2
0.4
0.6
0.8
1.0
Average Precision

5.3 Application Effectiveness and User Feedback

Throughout the testing phase, we collected valuable feedback from blind individuals who used the "Blind Guide" application. The overall feedback was overwhelmingly positive, with users expressing appreciation for the application's capabilities in assisting them with object detection and currency recognition tasks. They reported increased independence, improved navigation, and enhanced interaction with their environment. The integration of language translation and speech synthesis features also received positive feedback, as it enabled users to receive object descriptions in their preferred language through audible speech.

In summary, the implementation and testing of the "Blind Guide" application demonstrated impressive performance in object detection, currency recognition, and localization functions. The positive user feedback further validated the effectiveness of the application in assisting blind individuals, fostering increased independence and improved interaction with their surroundings.

Chapter 6: Conclusion and Future works

6.1 Conclusion

In conclusion, the "Blind Guide" project has successfully developed a computer vision-based mobile application that addresses the challenges faced by blind individuals. By leveraging deep learning models for object detection and currency recognition, the application provides real-time assistance in identifying objects and currencies. The integration of localization, language translation, and speech synthesis features enhances the user experience and facilitates a better understanding of the environment. Through extensive user feedback and rigorous testing, we have validated the effectiveness and usability of the application in improving the independence and overall quality of life for blind individuals.

The "Blind Guide" application has demonstrated impressive performance in object detection, currency recognition, and localization functions. It has received overwhelmingly positive feedback from blind users, who have reported increased independence, improved navigation, and enhanced interaction with their surroundings. The integration of language translation and speech synthesis features has been particularly beneficial, enabling users to receive object descriptions in their preferred language through audible speech.

6.2 Future Works

While the "Blind Guide" application has achieved significant milestones, there are several areas for future improvement and expansion:

6.2.1 Enhanced Object Detection:

To further improve object detection accuracy and robustness, it is recommended to refine the YOLOv8 model by exploring advanced techniques. This may include incorporating additional training data or utilizing more sophisticated architectures. These improvements will ensure that the application remains effective in detecting a wide range of objects in various real-time scenarios.

6.2.2 Localization Function Refinement:

To enhance the accuracy and precision of the localization function, it is necessary to investigate methods that provide more precise spatial descriptions of detected objects. By refining this aspect of the application, blind individuals can have even better understanding of the locations and orientations of objects in their surroundings.

6.2.3 Continuous Model Training:

Implementing a mechanism for continuous model training and updating is essential to adapt to evolving object and currency variations. By continuously training the models with new data, the application can remain effective in detecting new objects and recognizing different currency denominations as they enter circulation.

6.2.4 Integration with Navigation Systems:

Exploring integration with existing navigation systems or technologies will enable the "Blind Guide" application to provide seamless guidance and navigation for blind individuals. By incorporating the object detection and localization capabilities of the application into navigation systems, blind users can receive comprehensive assistance in navigating their environment.

6.2.5 Accessibility and Usability Enhancements:

Continuously improving the user interface and accessibility features of the application based on user feedback is crucial. By enhancing the user interface's intuitiveness and incorporating accessibility features, the application can provide a more user-friendly experience for blind individuals, further enhancing their independence and ease of use.

By pursuing these future works, the "Blind Guide" project can continue to evolve and provide even greater assistance to the blind community. Through ongoing improvements and expansions, the application aims to empower blind individuals

with increased independence and improved access to information in their daily lives. The "Blind Guide" team remains committed to addressing the needs of the blind community and creating innovative solutions that enhance their quality of life.

References

- Agarwal, S., Awan, A., and Roth, D. (2004). Learning to detect objects in images via a sparse,part-based representation. IEEE Trans. Pattern Anal. Mach. Intell. 26,1475–1490. doi:10.1109/TPAMI.2004.108
- Alexe, B., Deselaers, T., and Ferrari, V. (2010). “What is an object?,” in ComputerVision and Pattern Recognition (CVPR), 2010 IEEE Conference on (San Francisco,CA: IEEE), 73–80. doi:10.1109/CVPR.2010.5540226
- Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. Int. J.Comput. Vis. 1,333–356. doi:10.1007/BF00133571
- Andreopoulos, A., and Tsotsos, J. K. (2013). 50 years of object recognition: direc-tions forward. Comput. Vis. Image Underst. 117, 827–891. doi:10.1016/j.cviu.2013.04.005
- Azizpour, H., and Laptev, I. (2012). “Object detection using strongly-superviseddeformable part models,” in Computer Vision-ECCV 2012 (Florence: Springer),836–849.
- Azzopardi, G., and Petkov, N. (2013). Trainable cosfire filters for keypoint detectionand pattern recognition. IEEE Trans. Pattern Anal. Mach. Intell. 35, 490–503.doi:10.1109/TPAMI.2012.106
- Azzopardi, G., and Petkov, N. (2014). Ventral-stream-like shape representation:from pixel intensity values to trainable object-selective cosfire models. Front.Comput. Neurosci. 8:80. doi:10.3389/fncom.2014.00080
- Benbouzid, D., Busa-Fekete, R., and Kegl, B. (2012). “Fast classification using sparsedecision dags,” in Proceedings of the 29th International Conference on MachineLearning (ICML-12), ICML ‘12, eds J. Langford and J. Pineau (New York, NY:Omnipress), 951–958.
- Bengio, Y. (2012). “Deep learning of representations for unsupervised and transferlearning,” in ICML Unsupervised and Transfer Learning, Volume 27 of JMLRProceedings, eds I. Guyon, G. Dror, V. Lemaire, G. W. Taylor, and D. L. Silver(Bellevue: JMLR.Org), 17–36.
- Bourdev, L. D., Maji, S., Brox, T., and Malik, J. (2010). “Detecting peopleusing mutually consistent poselet activations,” in Computer Vision – ECCV2010 – 11th European Conference on Computer Vision, Heraklion, Crete, Greece,September 5-11, 2010, Proceedings, Part VI, Volume 6316 of

Lecture Notes inComputer Science, eds K. Daniilidis, P. Maragos, and N. Paragios (Heraklion:Springer), 168–181.

- Bourdev, L. D., and Malik, J. (2009). “Poselets: body part detectors trained using 3dhuman pose annotations,” in IEEE 12th International Conference on ComputerVision, ICCV 2009, Kyoto, Japan, September 27 – October 4, 2009 (Kyoto: IEEE),1365–1372.41
- Cadena, C., Dick, A., and Reid, I. (2015). “A fast, modular scene understanding sys-tem using context-aware object detection,” in Robotics and Automation (ICRA),2015 IEEE International Conference on (Seattle, WA).
- Correa, M., Hermosilla, G., Verschae, R., and Ruiz-del-Solar, J. (2012). Humandetection and identification by robots using thermal and visual information indomestic environments. J. Intell. Robot Syst. 66, 223–243. doi:10.1007/s10846-011-9612-2
- Dalal, N., and Triggs, B. (2005). “Histograms of oriented gradients for humandetection,” in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEEComputer Society Conference on, Vol. 1 (San Diego, CA: IEEE), 886–893. doi:10.1109/CVPR.2005.177
- Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). “Scalable object detec-tion using deep neural networks,” in Computer Vision and Pattern Recognition Frontiers in Robotics and AI www.frontiersin.org November 2015
- https://monkeylearn.com/machine-learning
- https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/#3dbb3f751018 (Link resides outside ibm.com)
- https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3 (Link resides outside ibm.com)
- Optical character recognition, Wikipedia (Link resides outside ibm.com)
- Intelligent character recognition, Wikipedia (Link resides outside ibm.com)
- A Brief History of Computer Vision (and Convolutional Neural Networks), Rostyslav Demush, Hacker Noon, February 27, 2019 (Link resides outside ibm.com)

- 7 Amazing Examples of Computer And Machine Vision In Practice, Bernard Marr, Forbes, April 8, 2019 (Link resides outside ibm.com)
- 7. The 5 Computer Vision Techniques That Will Change How You See The World, James Le, Heartbeat, April 12, 2018 (Link resides outside ibm.com)