Politecnico di Bari

Dipartimento di Ingegneria Elettrica e dell'Informazione – DEI

Corso di Laurea Magistrale in Ingegneria Informatica

Optical Character Recognition for Visually Impaired People

Professor:
Prof.ssa Marina Mongiello

Student:
Davide MINERVINI

1. Introduction:

1.1: Importance of Optical Character Recognition

Optical Character Recognition (OCR) is a transformative technology that enables the conversion of printed or handwritten text into machine-readable digital text. It utilizes sophisticated algorithms to analyze the shapes and patterns of characters in images or scanned documents, extracting the textual information contained within them.

OCR technology has gained significant prominence in recent years due to its ability to automate the process of text recognition and data extraction. By leveraging advanced image processing techniques, OCR algorithms can accurately interpret and decipher text from a wide range of sources, including physical documents, scanned images, and photographs.

The primary objective of OCR is to enhance the efficiency and accuracy of data extraction. By eliminating the need for manual data entry, OCR significantly reduces the time and effort required to process large volumes of textual information. This automation not only increases productivity but also minimizes the likelihood of human error, leading to improved data quality and reliability.

Moreover, OCR plays a vital role in the digitization and searchability of documents. It facilitates the conversion of paper-based documents into digital formats, enabling efficient storage, retrieval, and management of textual content. The ability to search within digitized documents empowers individuals and organizations to quickly locate specific information, improving overall accessibility and productivity.

Furthermore, OCR technology has brought about significant advancements in accessibility and inclusivity. By converting text in images or documents into speech or braille, OCR enables individuals with visual impairments to access and comprehend textual content that was previously inaccessible. This fosters equal access to information, empowering individuals with disabilities to participate more fully in society.

In recent years, OCR has found widespread application across various industries and sectors. From automated invoice processing in the finance industry to passport recognition at airports, OCR technology has revolutionized data processing, document management, and information retrieval. The integration of OCR with other technologies such as natural language processing, machine learning, and artificial intelligence further expands its capabilities, enabling advanced features such as language translation, sentiment analysis, and data extraction from unstructured documents.

As OCR continues to evolve, ongoing research and development efforts aim to improve its accuracy, expand language support, and enhance integration with emerging technologies. The future implications of OCR are promising, with the potential to revolutionize data processing, information accessibility, and automation in numerous domains.

1.2: Characteristics of Optical Character Recognition

Normally, a complete OCR process includes 5 main steps:

1. Noise attenuation
2. Image binarization (to black and white)
3. Text segmentation
4. Character recognition
5. Post-processing, such as spell checking

These steps are very effective when applied to document text, which when collected by a scanner, is generally aligned and has clear contrast between text and its uniform background. However, taking pictures from a portable camera, especially the one embedded inside a mobile device, may lead to various artifacts in the images and as a result, causes even the best available OCR engine to fail.

1: Different issues arising in camera-captured documents: (a) shading (b) flooding (c) blur (d) skew (e) tilted (f) misalignment

A. Lighting Condition

An embedded camera inside the mobile phone has far less control of lighting conditions than scanners. Uneven lighting is common, due to both the physical environment (shadows, reflection, fluorescents) and uneven response from the devices. Further complications occur when trying to use artificial light, i.e. flash, which results in light flooding.

Binarization has long been recognized as a standard method to solve the lightning issue. The goal of binarization process is to classify image pixels from the given input grayscale or color document into either foreground (text) or background and as a result, reduces the candidate text region to be processed by later processing steps.

B. Text Skew

When OCR input is taken from a hand-held camera or other imaging device whose perspective is not fixed like a scanner, text lines may get skewed from their original orientation. Feeding such a rotated image to our OCR engine produces extremely poor results. A skew detection process is needed before calling the recognition engine. If any skew is detected, an auto-rotation procedure is performed to correct the skew before processing text further.

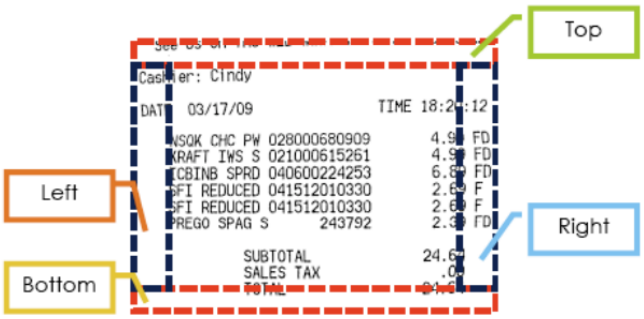
C. Perception Distortion (Tilt)

Perception distortion occurs when the text plane is not parallel to the imaging plane. It happens a lot if using a hand-held camera to take pictures. The effect is that characters further away look smaller and distorted, and parallel-line assumptions no longer hold in the image. From our experience, small to mild perception distortion causes significant degradation in performance of our OCR engine.

Instead of applying image processing techniques to correct the distortion, we take advantage of the embedded orientation sensors to measure the tilt of the phone. Users are prevented from taking pictures if the camera is tilted to some extent, and as a result, the chances of perception distortion are reduced considerably. If the text source itself is titled, a facility to calibrate the phone to any orientation is provided so that the imaging plane is parallel to the text plane.

D. Misalignment

Text misalignment happens when the camera screen covers a partial text region, in which irregular shapes of the text characters are captured and imported as inputs to the OCR engine. Misalignment issues occur a lot especially when people casually take their pictures. The OCR result is significantly affected by misalignment. Also misaligned images may lead to loss of important data. Firstly, the camera captured image is defined as misaligned if any of the four screen borders of the phone cuts through a single line of text, either horizontally, or vertically. Based on this definition, in order to detect misalignment, a 10-pixel wide margin along each border is defined, as depicted in figure below:



2: Definition of four 10-pixel wide margins

If any foreground pixel is detected within the margins, there is a high probability that the text is being cut. Therefore, we conclude that the image is misaligned. Considering issues of imperfect lighting conditions, a binarization preprocessing step is necessary before we can

infer whether we have detected any black dot or not. To overcome these problems, the ***Sauvola binarization algorithm*** is used.

E. Blur (Out Of Focus)

Since many digital cameras are designed to operate over a variety of distances, focus becomes a significant factor. Sharp edge response is required for the best character segmentation and recognition. At short distances and large apertures, even slight perspective changes can cause uneven focus. AutoFocus API, provided by Android SDK, is used to avoid any blurring seen in out of focus images. Whenever we start the application, a camera autofocus handler object is declared and initiated so that the camera itself can focus on the text sources automatically.

1.3: Related Works and Literature

In this work, two different implementations will be discussed. The first one will use ***Firebase ML Kit*** to perform OCR, while the second one will perform OCR through ***Tesseract OCR*** and then read the recognized characters out loud through ***Text-to-Speech API***.
The information used and depicted in this report is mostly referring to the libraries and APIs documentation. Let's discuss the proposed implementations in the next section.

2. Firebase OCR Implementation

The implementation of this app uses Firebase ML Toolkit library to perform OCR on an image snapped with the phone camera. Let's discuss its peculiarities.

2.1: Firebase ML Toolkit

Firebase ML Kit is a powerful mobile SDK provided by Google that enables developers to integrate machine learning capabilities into their Android applications without requiring extensive knowledge of machine learning algorithms. One of the key features of Firebase ML Kit is its support for Optical Character Recognition (OCR), which allows developers to extract text from images.

Firebase ML Kit's OCR capabilities provide a convenient and efficient way to extract text from various sources such as photos, screenshots, or camera input. It leverages advanced machine learning models and algorithms to accurately detect and recognize text within images.

By using Firebase ML Kit's OCR functionality, developers can implement features in their Android applications that involve text recognition and analysis. For example, developers can build apps that can scan and extract text from documents, receipts, business cards, or any other text-containing images. This opens up a wide range of possibilities for automating data entry, extracting information, or enhancing user experiences.

Firebase ML Kit's OCR API provides a simple and intuitive interface for developers to integrate text recognition into their apps. It offers options to detect and recognize text in

real-time from live camera input or to process static images. The API also supports language detection, enabling multilingual OCR capabilities.

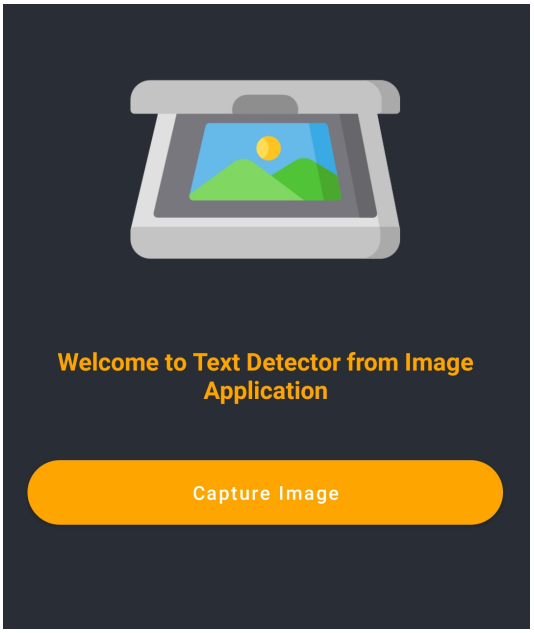
Additionally, Firebase ML Kit's OCR functionality provides robust text recognition even in challenging conditions, such as low light, skewed angles, or complex backgrounds. It also supports text recognition in different languages and provides flexibility in customizing recognition settings to meet specific application requirements.

By utilizing Firebase ML Kit's OCR capabilities, developers can significantly reduce the development effort and complexity involved in implementing OCR functionality in their Android applications. They can focus on building user-friendly interfaces and leveraging the extracted text for various purposes, such as translation, data analysis, or further processing.

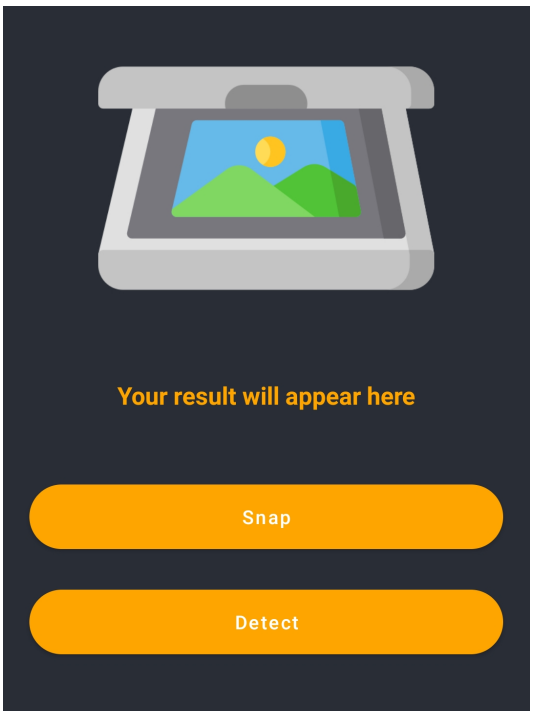
Overall, Firebase ML Kit's OCR functionality empowers Android developers to incorporate powerful text recognition capabilities into their applications, unlocking new possibilities for automating tasks, enhancing productivity, and improving user experiences.

2.2: App General UI

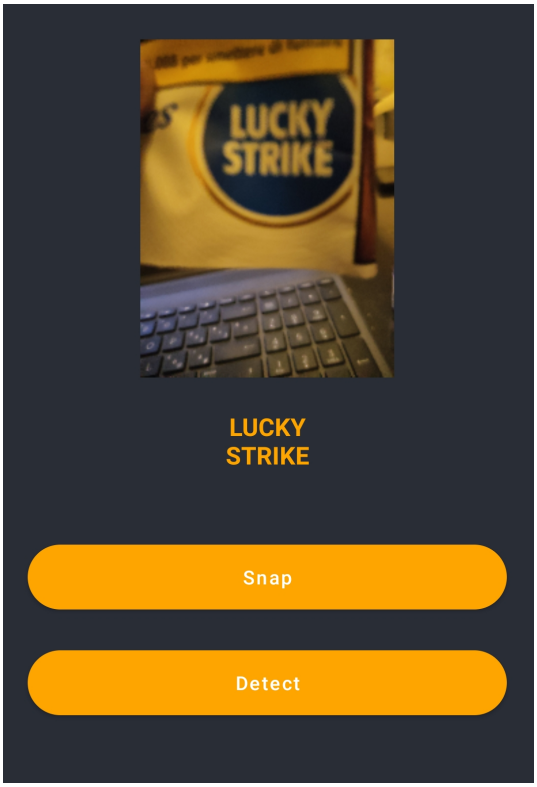
The App's general UI as we open it looks like this:



The "**Capture Image**" button allows us, through a permission request, to use the smartphone's camera and take a picture of the text on which we're going to perform OCR on. After the permission is granted, the app will have this layout:



Where the “**Snap**” button is used to take the picture, while the “**Detect**” button performs OCR on the snapped image. Here’s an example of how the interface looks after the use of the **Detect** button:



2.3: App Main Activity Code

The code for the ***Main Activity*** of the app is available at this GitHub link: https://github.com/davoidadmin/FirebaseOCR/blob/master/app/src/main/java/com/example/ocrgeeksforgeeks/MainActivity.java.

Here's a general overview of what the code does:

1. After the necessary imports and declarations, we can find **onCreate()** Method:
    - This method is called when the activity is created.
    - The layout is set using the setContentView() method, which associates the XML layout file with the activity.
    - The captureBtn button is initialized and assigned to the corresponding UI element using findViewById().
    - A click listener is attached to the captureBtn button using an anonymous inner class implementing the OnClickListener interface.
    - When the button is clicked, the onClick() method is triggered.
2. Then, we can find the **onClick()** Method:
    - This method is called when the captureBtn button is clicked.
    - An Intent object i is created to launch the ScannerActivity.
    - The ScannerActivity is specified as the target activity for the intent using ScannerActivity.class.
    - The startActivity() method is called, passing the intent i to start the ScannerActivity.

The purpose of this code is to set up the main activity of an OCR-based Android application. It initializes the UI elements and handles the button click event. When the button is clicked, it launches the ScannerActivity to perform the scanning and OCR functionality.

Now, let's see what the ***Scanner Activity*** does. The code for this activity can be found at this link: https://github.com/davoidadmin/FirebaseOCR/blob/master/app/src/main/java/com/example/ocrgeeksforgeeks/ScannerActivity.java

1. After the necessary imports and declarations, we can find onCreate() Method:
    - This method is called when the activity is created.
    - The layout is set using the setContentView() method, associating the XML layout file with the activity.
    - UI elements are initialized and assigned to their respective variables using findViewById().
    - Click listeners are set for the detect button (detectBtn) and snap button (snapBtn).
2. Button Click Listeners:

- detectBtn click listener: Calls the detectText() method to perform OCR on the captured image.
- snapBtn click listener: Checks for camera permissions and either captures an image or requests permission.

3. Permission Handling Methods:
    - checkPermissions(): Checks if the camera permission is granted.
    - requestPermission(): Requests camera permission if it is not granted.
    - onRequestPermissionsResult(): Callback method for handling permission results.
        - If the camera permission is granted, a toast message is displayed, and the captureImage() method is called.
        - If the permission is denied, a toast message is displayed.
4. Image Capture Handling:
    - captureImage(): Creates an Intent to capture an image using the device's camera app.
        - If a camera app is available, the startActivityForResult() method is called to capture the image.
    - onActivityResult(): Callback method for handling the result of the image capture.
        - If the capture is successful, the captured image bitmap is extracted from the Intent and displayed in the captureIV ImageView.
5. Text Detection:
    - detectText(): Creates an InputImage object from the captured image bitmap and initializes a TextRecognizer using default options.
        - The TextRecognizer processes the image using the process() method, which returns a Task<Text> object.
        - The addOnSuccessListener is called when the text detection is successful.
            - The detected text is extracted from the Text object and appended to the result StringBuilder.
            - The detected text is set as the text for the resultTV TextView.
        - The addOnFailureListener is called if there is an error during text detection, displaying a toast message with the error details.

This code handles image capture, text detection using ML Kit Vision's TextRecognizer, and updating the UI with the detected text.

The entire project link, with all the necessary files to run the app on Android Studio is available at: https://github.com/davoidadmin/FirebaseOCR

3. Tesseract OCR and Text-to-Speech Implementation

For this second implementation, Tesseract OCR engine has been used, together with Text-to-Speech API. Let’s discuss their peculiarities:

3.1: Tesseract OCR

Tesseract OCR, an open-source OCR engine, has emerged as a leading solution in this field, offering powerful text recognition capabilities and robust performance.

Tesseract OCR has a rich history that dates back to its development at Hewlett-Packard Laboratories. Initially released as a proprietary software, Tesseract OCR transitioned into an open-source project, allowing developers worldwide to contribute to its evolution. Currently maintained and supported by the open-source community, Tesseract OCR benefits from the collective expertise and continuous improvements driven by a diverse group of contributors.

One of the notable features of Tesseract OCR is its exceptional accuracy in recognizing text from images. It can effectively handle various languages, font styles, and complex document layouts, making it suitable for a wide range of applications. Additionally, Tesseract OCR provides extensive language support through its language data files, enabling precise recognition and extraction of text in different languages.

Integration of Tesseract OCR into applications and frameworks is straightforward, allowing developers to leverage its capabilities for text recognition tasks. Whether it's mobile apps, desktop software, or web applications, Tesseract OCR can be seamlessly integrated, offering reliable and efficient OCR functionality. Its performance is optimized to deliver fast and accurate results, even in challenging scenarios involving low-resolution images or poor-quality scans.

3.2: Text-to-Speech API

Text-to-Speech (TTS) is a technology that converts written text into spoken words. It enables computers and software applications to generate human-like speech, allowing users to listen to text content rather than reading it. TTS APIs provide developers with the tools and resources to integrate this functionality into their applications, opening up a range of possibilities for enhancing accessibility, user experience, and communication.

In recent years, TTS technology has made significant advancements, thanks to advancements in machine learning and natural language processing. Modern TTS APIs utilize deep learning models and neural networks to create more natural and expressive speech, mimicking the nuances of human language and intonation. This

has led to significant improvements in speech quality and the overall user experience.

TTS APIs offer a variety of features and customization options to suit different application requirements. Developers can choose from a range of voices with different genders, accents, and languages, allowing them to tailor the speech output to match their application's target audience or specific use case. Additionally, developers can control parameters such as speech rate, pitch, and volume to further fine-tune the generated speech.

The applications of TTS API are vast and diverse. They can be integrated into various domains and industries to provide value-added functionalities. Some common use cases include:

- Accessibility: Text-to-Speech API enables visually impaired individuals to access digital content by converting written text into spoken words. It enhances the accessibility of websites, e-books, documents, and other online platforms.

- Language Learning: Language learning applications can leverage Text-to-Speech to provide pronunciation assistance, enabling learners to hear correct pronunciations of words and phrases in different languages.

- Assistive Technologies: Text-to-Speech can be integrated into assistive technologies such as screen readers and communication aids, enabling individuals with disabilities to interact with computers and devices.

- Voice-Enabled Applications: Text-to-Speech APIs can power voice assistants, chatbots, virtual agents, and other voice-enabled applications, enhancing user interactions and providing a more natural and intuitive user experience.

- Multimedia and Entertainment: Text-to-Speech technology can be used to generate voice-overs for multimedia content, audiobooks, podcasts, and other forms of media, adding a personalized and engaging touch.

By leveraging a Text-to-Speech API, developers can harness the power of speech synthesis to create inclusive, engaging, and interactive applications that cater to a broader audience and deliver information in a more accessible and convenient format.

3.3: App General UI

The layout is basically equivalent to the previous implementation, adding a new "**Read Text**" button to read the detected string out loud through Text-to-Speech API. This guarantees modularity in our project, dividing the whole process in a pipeline that can be expanded in a partial way.

3.4: App Main Activity Code

The code for the ***Main Activity*** of the app is available at this GitHub link:

https://github.com/davoidadmin/TesseractOCR/blob/master/app/src/main/java/com/example/tesseract_ocr/MainActivity.java

Here's a general overview of what the code does:

1. **onCreate()** Method:
    - This method is called when the activity is created.
    - The layout is set using the `setContentView()` method, which associates the XML layout file with the activity.
    - UI elements (buttons, preview view) are initialized and assigned to their respective variables using `findViewById()`.
    - An instance of the `TesseractOCR` class is created by passing the `AssetManager` obtained from the activity.
    - An instance of `TextToSpeech` is created and initialized with the activity as the context and the activity itself as the listener.
2. **Button Click Listeners**:
    - `mCaptureButton`, `mOCRButton`, and `mReadButton` have click listeners attached to them.
    - The `onClick()` methods for each button handle the corresponding actions:
        - `mCaptureButton`: Checks camera permission and either opens the camera or requests the permission.
        - `mOCRButton`: Calls the `doOCR()` method, which performs optical character recognition (OCR) on the captured image using the `TesseractOCR` instance.
        - `mReadButton`: Calls the `readText()` method, which reads the recognized text aloud using the `TextToSpeech` instance.
3. **Camera Handling** Methods:
    - `openCamera()`: Requests an instance of `ProcessCameraProvider` using `ProcessCameraProvider.getInstance(this)`.
        - When the provider is available, it binds the camera preview to the `mPreviewView` using the `bindPreview()` method.

        - Exceptions are caught and appropriate error messages are displayed.
    - `bindPreview()`: Creates a `Preview` object and a `CameraSelector` object for the back-facing camera.
        - It associates the preview with the `mPreviewView` and binds the camera with the lifecycle of the activity.
        - Exceptions are caught and appropriate error messages are displayed.
4. **OCR and Text-to-Speech** Methods:
    - `doOCR()`: Performs OCR on the captured image (`mImageBitmap`) using the `TesseractOCR` instance.
        - If the image bitmap is available, the `getResults()` method of `TesseractOCR` is called to obtain the OCR result.
        - The result is logged for debugging purposes.
    - `readText()`: Reads the recognized text aloud using the `TextToSpeech` instance.
        - If the image bitmap is available, the `getUTF8Text()` method of `TesseractOCR` is called to obtain the text.
        - The text is passed to `mTTS.speak()` to be spoken out loud.
5. **Other Methods and Callbacks**:
    - `onInit()`: Callback method of `TextToSpeech.OnInitListener` interface, called when the `TextToSpeech` engine is initialized.
        - Checks if the language is supported, and logs an error message if not.
    - `onDestroy()`: Lifecycle method called when the activity is destroyed.
        - Performs cleanup tasks such as destroying the `TesseractOCR` instance, shutting down `TextToSpeech`, and terminating the camera executor.
    - `onRequestPermissionsResult()`: Callback method for handling permission results.
        - Checks if the camera permission is granted and either opens the camera or displays a permission denied message.

The entire project link, with all the necessary files to run the app on Android Studio is available at: https://github.com/davoidadmin/TesseractOCR

4. Conclusions

Let’s now make a comparison between the implemented tools to understand which one is the best candidate to reach our goal.

Firebase ML Kit and Tesseract OCR are both popular tools for implementing Optical Character Recognition in mobile applications. However, they have distinct characteristics that make them suitable for different use cases. When considering the app UI to be completely vocal assistant-driven, catering to visually impaired individuals, there are several pros and cons to consider for each solution:

Firebase ML Kit:

Pros:

- ➢ Integration with other Firebase services: Firebase ML Kit offers seamless integration with other Firebase services, such as Cloud Firestore and Cloud Functions, providing a comprehensive ecosystem for building mobile applications.
- ➢ Cloud-based OCR: Firebase ML Kit's OCR capabilities are predominantly cloud-based, allowing for more powerful and accurate text recognition. The OCR processing happens on remote servers, which can handle complex OCR tasks and deliver reliable results.
- ➢ Wide language support: Firebase ML Kit supports a wide range of languages for OCR, enabling multilingual applications and expanding accessibility for users globally.
- ➢ Continuously updated models: Firebase ML Kit benefits from Google's continuous updates and improvements to its machine learning models, ensuring the latest advancements in OCR accuracy and performance.

Cons:

- ❖ Internet dependency: Firebase ML Kit's cloud-based OCR requires an active internet connection, which may not always be available or reliable. Offline functionality might be limited, impacting the user experience in areas with poor connectivity.
- ❖ Pricing considerations: Firebase ML Kit has a pricing model based on usage, and extensive OCR usage may incur costs, especially if the application processes a large volume of images with OCR requests.

Tesseract OCR:

Pros:

- ➢ Open-source and free: Tesseract OCR is an open-source library, making it free to use and modify. It offers flexibility and control over the OCR implementation, allowing customization and integration into specific application requirements.
- ➢ Offline functionality: Tesseract OCR can be implemented to work entirely offline, without relying on an internet connection. This ensures consistent OCR functionality even in areas with limited or no internet access.
- ➢ Community support: Tesseract OCR benefits from a vibrant and active community of developers contributing to its development and providing support. There are various online resources, forums, and documentation available for assistance and troubleshooting.
- ➢ On-device processing: Tesseract OCR performs OCR processing locally on the device, eliminating the need for network communication and reducing latency.

Cons:

- ❖ Limited language support: Tesseract OCR has somewhat limited language support compared to Firebase ML Kit. While it supports several languages, the range might be narrower than what Firebase ML Kit offers.
- ❖ Development complexity: Integrating and configuring Tesseract OCR requires additional development effort compared to using Firebase ML Kit. Developers need to handle the implementation details, training data, and preprocessing of images for optimal OCR results.

Considering the goal of making the app UI completely vocal assistant-driven to aid visually impaired users, Tesseract OCR may be a better fit due to the following reasons:

- Offline functionality: Tesseract OCR's ability to perform OCR processing offline ensures that the vocal assistant-driven UI remains functional even without an internet connection, providing uninterrupted accessibility to visually impaired users.
- Control and customization: Tesseract OCR's open-source nature allows developers to tailor the OCR implementation to specific requirements, ensuring a seamless integration with the vocal assistant-driven UI and enabling fine-grained control over the user experience.

- Community support: The active community supporting Tesseract OCR can provide guidance, assistance, and potential contributions to enhance the accessibility features for visually impaired users.

However, it's worth noting that Firebase ML Kit's cloud-based OCR can offer more robust OCR capabilities and continuous model improvements, which might be advantageous in scenarios where internet connectivity is reliable and extensive language support is critical.

Ultimately, the choice between Firebase ML Kit and Tesseract OCR depends on the specific needs and constraints of the application. Considering the vocal assistant-driven UI and accessibility for visually impaired users, Tesseract OCR's offline functionality and customization options make it a stronger candidate.