**CHAPTER-1**

**INTRODUCTION**

* 1. **PRELUDE**

In today’s rapidly evolving technological landscape, artificial intelligence (AI) has emerged as a transformative force across a variety of fields, including healthcare, education, security, and assistive technologies. One of the most impactful applications of AI lies in its potential to support individuals with disabilities by enhancing their interaction with the environment and information systems. For the visually impaired community, this opens up new avenues for independence and inclusion. Vision plays a vital role in interpreting and understanding the world, and the lack of it often hinders access to information that is readily available to sighted individuals in the form of text.

Text is everywhere—on street signs, books, menus, forms, nameplates, instruction manuals, product labels, digital displays, and more. Sighted individuals can process this information instantaneously, but for someone who is visually impaired, even basic navigation and comprehension tasks become challenging. Although screen readers and braille display help with digital content, there remains a significant gap when it comes to printed or environmental text.

Recent advancements in deep learning and embedded systems have enabled the development of intelligent, real-time computer vision models that can analyse and interpret images effectively. Leveraging these capabilities, it is now possible to create portable, AI-powered assistants that can detect text from the surroundings and convert it into speech. This project aims to bridge this accessibility gap by designing a real-time text-to-speech system powered by deep learning, specifically targeted to aid the visually impaired in their daily interactions with the world around them.

* 1. **MOTIVATION**

The increasing reliance on digital technologies has highlighted the need for **accessible solutions** that support individuals with visual impairments. Text-to-speech (TTS) systems are a key tool for improving accessibility, as they allow visually impaired individuals to interact with digital content through auditory feedback. However, many existing TTS systems are either limited in their language and accent support, or they do not provide sufficient customization options, making them less adaptable to the diverse needs of users.

The motivation behind this project is to develop an **AI-powered TTS system** that leverages advanced machine learning techniques to provide high-quality, natural-sounding speech synthesis for a wide range of languages, accents, and contexts. By integrating **FPN - FCN** for text detection, **PyTesseract** for text recognition and **eSpeak** for speech conversion, this system aims to provide a seamless experience for users, enabling them to convert printed or displayed text into speech, especially from images and documents.

A key challenge for visually impaired individuals is the ability to access text in images, such as printed books, signs, or documents. Current TTS systems typically focus on static text but lack the capability to handle dynamic or image-based text. This project aims to bridge that gap by incorporating **FPN - FCN** for accurate text detection from images, ensuring that visually impaired users can access a broader range of information. Additionally, the use of **eSpeak** as the TTS engine ensures a lightweight, customizable, and resource-efficient solution, making it suitable for devices with limited computational resources, such as the **Raspberry Pi**.

By combining these technologies, the project seeks to enhance the accessibility of digital content for visually impaired individuals, providing them with an AI-driven, adaptable, and high-performance solution for interacting with written text. The motivation for this work is to not only improve the quality of life for visually impaired users but also contribute to making digital environments more inclusive and universally accessible.

* 1. **PROBLEM STATEMENT**

Visual impairment is a major disability that affects millions of people worldwide. According to the World Health Organization (WHO), over 250 million people live with some form of vision impairment, and a significant portion of them have limited access to assistive technologies. One of the biggest challenges faced by visually impaired individuals is the inability to independently access printed information from physical environments, which negatively impacts their confidence, safety, and autonomy.

Most current assistive technologies for reading rely on either expensive hardware or limited software solutions. Screen readers are excellent tools for digital content, but they cannot interpret printed text in the environment. Braille displays are not always practical, especially for on-the-go use. Some AI-powered glasses do exist, but they are highly expensive, not open-source, and often dependent on cloud-based services, making them inaccessible in offline conditions or rural areas.

Additionally, many text recognition systems require high computing power and are not suitable for real-time use on embedded platforms like the Raspberry Pi. This presents a challenge when designing a portable and cost-effective solution that combines detection, recognition, and speech synthesis in one compact unit. Therefore, there is a strong need for an integrated, low-cost, real-time system that is capable of detecting text from real-world scenes, recognizing it accurately, and converting it into speech without internet dependency—thus helping visually impaired users become more self-reliant in daily activities such as reading signboards, navigating public spaces, or understanding documents.

* 1. **OBJECTIVE**

The **AI-powered Text-to-Speech Assistant for Visually Impaired** is designed with the core aim of making text-based information more accessible in real-time for individuals who are visually impaired. The project combines multiple fields—computer vision, deep learning,

optical character recognition (OCR), and speech synthesis—into a unified embedded system that is affordable, portable, and user-friendly. The key objectives of this project are elaborated as follows:

1. **To build a robust text detection model using a Feature Pyramid Fully Convolutional Network (FPN-FCN)** architecture, which is well-suited for multi-scale text detection tasks. This model generates supervision maps such as kernel text maps, centerline maps, and height maps to accurately localize text regions from the captured images—even in cluttered or noisy backgrounds.
2. **To apply OCR using the Tesseract engine**, an open-source solution that converts the localized text areas into machine-readable strings. Tesseract is chosen for its flexibility, support for multiple languages, and ease of integration into embedded systems.
3. **To incorporate a Text-to-Speech (TTS) engine using eSpeak**, which transforms the recognized text into audio output. eSpeak is a lightweight, offline TTS engine that supports many languages and is well-suited for real-time speech synthesis on constrained devices.
4. **To ensure that the final prototype is portable, affordable, and user-centric**, enabling deployment in real-world conditions without relying on cloud services or external processing.
5. **To evaluate the system's performance through both technical benchmarks and user-based testing**, ensuring that it meets real-time constraints, achieves high detection and recognition accuracy, and effectively assists visually impaired users in daily activities.
   1. **LITERATURE SURVEY**

Table 1 summarizes the survey of works closely related to this project reported in literature.

|  |  |  |
| --- | --- | --- |
| **TITLE OF THE PAPERS &AUTHORS** | **TECHNIQUE PROPOSED IN THE PAPER** | **LIMITATIONS/ REMARKS** |
| "MULDT: Multilingual Ultra-Lightweight Document Text Detection for Embedded Devices" — A. Gayer and V. V. Arlazarov | Proposed MULDT — a lightweight text detector optimized for embedded devices. It is multilingual and processes both scanned documents and photos with high speed and accuracy. | Limited support for highly complex scripts and extreme low-light conditions |
| "HGR-Net: Hierarchical graph reasoning network for arbitrary shape scene text detection" — H. Bi et al. | Proposed a graph-based neural network for detecting arbitrarily shaped text, improving detection accuracy in complex scenarios. | High computational cost; not suitable for real-time embedded systems. |
| "Arbitrary shape text detection via boundary transformer" — S.-X. Zhang et al. | Developed a boundary transformer model for text detection, handling irregular text regions with high precision. | Requires substantial GPU resources; challenging for mobile deployment. |
| "Real-time scene text detection with differentiable binarization" — M. Liao et al. | Introduced a differentiable binarization method for faster and more accurate text detection, especially for document images. | Limited flexibility for multilingual or unseen document types. |
| "p-im2col: Simple yet efficient convolution algorithm" — A. V. Trusov et al. | Introduced an optimized convolution algorithm for fast and efficient text detection, enhancing computational performance. | Computationally intensive for low-end embedded devices. |

* 1. **MODULES OF WORK**

Figure 1 illustrates the key steps involved in the text to speech conversion process.

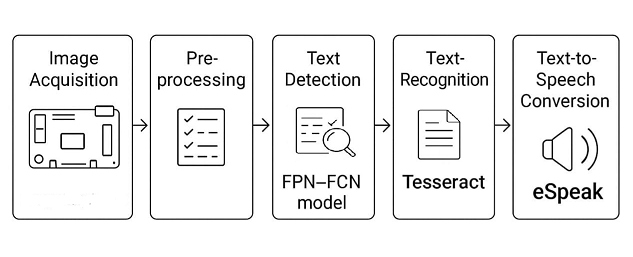


Figure 1 System Modules Overview

Key modules involved in the system workflow are as follows.

1. **Image Acquisition:** Captures text images using Raspberry Pi camera.
2. **Pre-processing:** Enhances images via grayscale conversion, noise removal, and thresholding.
3. **Text Detection:** Uses FPN-FCN to identify and segment text regions in the image.
4. **OCR Recognition**: Extracts readable text from detected regions using Tesseract.
5. **Text-to-Speech:** Converts recognized text into speech using eSpeak

**1.7 ORGANIZATION OF THE REPORT**

**Chapter 1** introduces the project and elaborates on the literature review, motivation, problem statement, objectives, and comparison between existing and proposed systems.

**Chapter 2** describes the hardware components and software models used in the project, explaining each part's specifications and their roles in the system.

**Chapter 3** explains the implementation and methodology in detail, including the circuit diagram, system workflow, coding approach, and integration of hardware and software modules.

**Chapter 4** presents the results obtained from the system, supported by outputs, screenshots, performance evaluations, and detailed discussions on the system's effectiveness.

**Chapter 5** concludes the project by summarizing the work done and discusses the future scope for further development and enhancements.

**CHAPTER-2**

**SYSTEM ARCHITECTURE AND TOOLS**

**OVERVIEW**

**2.1 BLOCK DIAGRAM**

The proposed offline Text-to-Speech (TTS) system is designed to support assistive technology applications, particularly for visually impaired users. It integrates key hardware and software components into a unified architecture, as depicted in Figure 2.

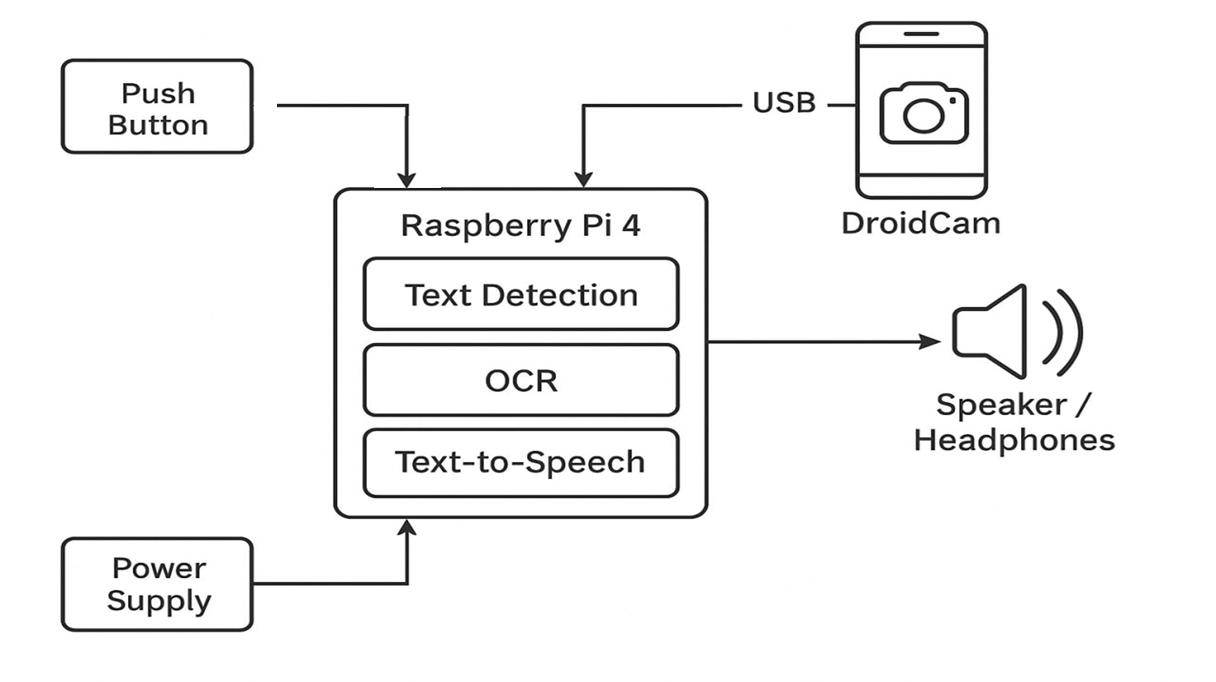
****

Figure 2 Block diagram of the Text-to-Speech Assistant system architecture.

A Raspberry Pi 4 serves as the central processing unit, initiating the workflow through a push-button interface. Image acquisition is handled via a smartphone connected over USB, offering a flexible and cost-effective alternative to dedicated camera modules. The captured image is then processed through text detection, recognition, and preprocessing stages, with the final speech output generated by an offline TTS engine. The entire system operates without internet dependency, making it suitable for low-resource or remote environments.

* 1. **DESCRIPTION OF COMPONENTS**

The block diagram in figure 2 consists of several key components, each playing a specific role in the overall system. The following sections provide a brief explanation of each component's function and relevance.

**2.2.1 RASPBERRY PI 4 (MODEL B – 4GB )**

The Raspberry Pi 4 serves as the **central processing unit** of the proposed offline Text-to-Speech (TTS) system. It offers the computational power and I/O capabilities necessary for image processing and speech synthesis. Its compact size, versatility, and low power consumption make it ideal for embedded applications. Multiple USB ports facilitate camera connections, while GPIO pins enable interfacing with peripherals like push buttons. Figure 3 shows the Raspberry Pi 4 and its essential components, including the processor, USB ports, and GPIO pins.

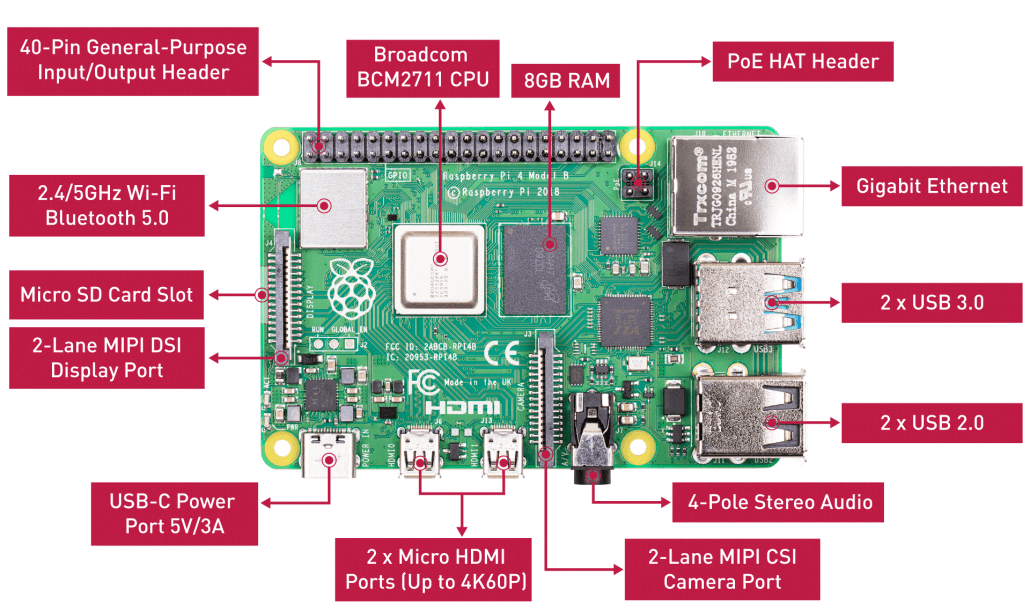


Figure 3 Raspberry pi 4 and key components

**2.2.2 PUSH BUTTON INTERFACE**

The push button acts as the **primary input mechanism**, allowing users to trigger the system's operation. Connected to one of the Raspberry Pi’s GPIO pins, the button provides a simple



Figure 4 Push Button

and reliable way to initiate the workflow of image capture, text detection, and speech synthesis. Figure 4 illustrates the push button's setup and role within the system, offering a straightforward interaction method suitable for a range of applications.

## **2.2.3 SMARTPHONE AS CAMERA (DROIDCAM)**

A smartphone is used in this system to **replace the need for a dedicated USB camera**. By running applications like DroidCam, the smartphone becomes a real-time webcam connected to the Raspberry Pi via USB. This approach is cost-effective and benefits from the smartphone’s high-quality camera and portability, enabling flexible and efficient image acquisition for the system.

## **2.2.4 EARPHONES**

Earphones serve as the **audio output device**, delivering the synthesized speech directly to the user. After processing the captured image and converting detected text into speech, the Raspberry Pi routes the audio output to the earphones. This setup ensures clear, private feedback for the user, enhancing usability in various environments.

## **2.2.5 POWER SUPPLY**

A **stable power source** is critical for the system’s reliability. The Raspberry Pi 4 requires a regulated 5V supply via a USB-C adapter, while the smartphone receives power through the USB connection. Using a reliable power adapter or a high-capacity power bank ensures continuous, uninterrupted operation during extended usage.

* 1. **DEEP LEARNING AND PROCESSING MODELS**

This section outlines the deep learning and image processing models used in the system for detecting, recognizing, and converting text. Each model contributes to a specific stage of the workflow, from feature extraction to final output generation.

**2.3.1. FEATURE PYRAMID FULLY CONVOLUTIONAL NETWORK**

The **Feature Pyramid Fully Convolutional Network (FPN-FCN)** is a powerful deep learning model used for detecting text in natural scene images. It combines the capabilities of a **Feature Pyramid Network (FPN)** with a **Fully Convolutional Network (FCN)** to effectively detect text regions at various scales, orientations, and densities. This makes it highly suitable for assistive applications where image quality and text layout may vary.

#### **Two-Stage Architecture of FPN-FCN**

Text detection using FPN-FCN involves **two major stages**:

* **Neural Network Inference**
* **Post-Processing**
* **Neural Network Inference**

The model consists of several key layers that collaboratively process the input document image, extract features, and localize text regions. The architecture, illustrated in **Figure 5**, depicts the flow from the input layer through various stages of feature extraction and text localization.

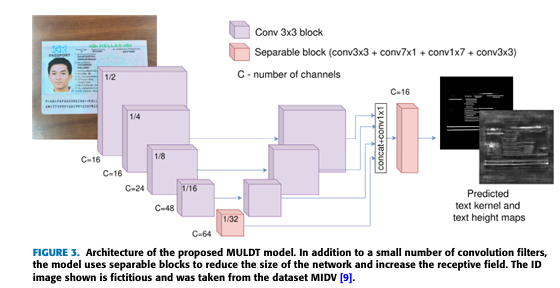
****

Figure 5 Architecture of the model

#### **Input Layer**

The **input layer** is the initial stage where document images of arbitrary sizes are received. It does not alter the input image but serves as the entry point for raw image data into the network. This layer enables the model to handle images in different resolutions and orientations, which is crucial for real-world applications where document formats vary. The image data is then passed on to the subsequent layers for further processing and feature extraction.

#### **Convolutional Layers**

Once the image is received, the **convolutional layers** apply **3x3 filters** to the input. These filters are essential for capturing local spatial patterns such as **edges**, **textures**, and **shapes**, which are vital for distinguishing text from background elements. **Figure 5** demonstrates how the convolutional layers gradually learn hierarchical features:

* **Early Layers:** These layers focus on detecting fundamental features such as edges and textures.
* **Deeper Layers:** These layers identify more complex structures like letter shapes, word boundaries, and text regions.

The **ReLU activation function** is applied after each convolution to introduce non-linearity, which is key for the model to capture complex relationships in the data and improve text recognition capabilities.

#### **Feature Pyramid Network (FPN)**

The **Feature Pyramid Network (FPN)**, shown in **Figure 5**, enhances the model's ability to detect text at different scales. It generates multi-scale feature maps by combining high-level semantic features with low-level spatial details. The FPN allows the model to detect small-scale text (e.g., footnotes) as well as large-scale text (e.g., titles or headers). This multi-scale feature fusion is crucial for robust text detection across varying text sizes in a single document.

* **Top-Down Pathway:** Higher-level features are upsampled and merged with lower-level feature maps, ensuring that both detailed spatial information and abstract semantic content are preserved.

#### **Separable Convolutions**

To enhance computational efficiency, the model utilizes **separable convolutions**, which are visualized in **Figure 5**. This approach divides the traditional convolution operation into **depthwise convolution** (where filters are applied to individual channels) and **pointwise convolution** (1x1 convolutions that combine channel-wise information). This reduces the number of parameters and accelerates computation, making the model more efficient while maintaining high performance in text detection.

#### **Upsampling and Concatenation**

As depicted in **Figure 5**, the model then uses **upsampling** to increase the resolution of the feature maps, which restores the spatial details lost during downsampling. **Concatenation** is then applied to merge feature maps from different scales, combining low-level spatial details with high-level semantic information. This process ensures that the final feature maps contain rich multi-scale data, allowing for precise text localization in the document image.

#### **Output Layer: Text Kernel Map**

The final output is the **text kernel map**, as shown in **Figure 5**. This map highlights the regions where text is likely to be located. Higher values in the kernel map indicate areas containing text, while lower values correspond to non-text regions. This text localization is essential for tasks like **optical character recognition (OCR)** and further image segmentation. The **text kernel map** is key in guiding the model to accurately localize and detect text within the input image.

Each layer in this architecture, as illustrated in **Figure 5**, plays a critical role in ensuring efficient and accurate text detection. The combination of convolutional layers, FPN, separable convolutions, upsampling, and concatenation enables the model to detect text across different scales, while maintaining computational efficiency and high accuracy.

### **POST-PROCESSING**

Post-processing aims to refine model predictions and generate accurate bounding boxes around detected text regions, crucial for subsequent OCR tasks.

#### **Resizing the Prediction Maps**

Since images are resized during preprocessing, the prediction maps (text map and height map) are resized back to the original image dimensions using interpolation, ensuring correct alignment with the real-world pixel coordinates.

#### **Thresholding the Text Map**

The text probability map is binarized using a threshold (typically 0.4). Pixels with probabilities above the threshold are considered text, creating a clear binary separation between text and background.

#### **Morphological Dilation**

Morphological dilation using a 7×7 kernel (iterated three times) merges fragmented text regions, fills small gaps, and strengthens text connectivity, improving the coherence of detected regions.

#### **Connected Component Analysis**

Connected component analysis identifies groups of connected text pixels. Bounding boxes are drawn around each component, and components smaller than a defined area threshold are discarded to remove noise.

#### **Removal of Nested Bounding Boxes**

Nested bounding boxes — where one box lies entirely within another — are removed to avoid redundant detections and ensure clarity in text localization.

#### **Removal of Overlapping Boxes Based on IoU**

Bounding boxes with a high Intersection over Union (IoU) overlap (>0.5) are merged to minimize redundant or overlapping detections, ensuring each text region is uniquely identified.

#### **Drawing Final Bounding Boxes**

The final bounding boxes are drawn over a copy of the original grayscale image, visually highlighting detected text areas and allowing easy verification of detection performance.

#### **Note on Height Map Usage**

Although the model predicts a height map alongside the text map, the post-processing currently uses only the text map for detection. Future work could leverage height map information to further enhance detection quality.

**2.3.2 PYTESSERACT**

**Pytesseract** is a Python wrapper for **Tesseract**, an open-source Optical Character Recognition (OCR) engine developed by Google. It is used to recognize and extract text from images, making it an essential tool for applications involving text recognition, image analysis, and document scanning. Pytesseract allows Python users to interact with the Tesseract engine directly, providing an easy-to-use interface for extracting text from images.

### **Key Features of Pytesseract:**

1. **Text Extraction**: It extracts printed text from images in a variety of formats, such as PNG, JPG, TIFF, or PDF.
2. **Multi-language Support**: Tesseract can recognize text in various languages, and Pytesseract allows specifying the language of the text for improved accuracy.
3. **Preprocessing Capabilities**: While Pytesseract itself doesn’t provide advanced image processing, it can be integrated with other libraries like OpenCV and PIL (Python Imaging Library) for preprocessing, such as thresholding, resizing, or noise removal, before passing the image to Tesseract for OCR.

**2.3.3 ESPEAK**

**eSpeak** is an open-source text-to-speech (TTS) engine that converts written text into spoken words. It is lightweight, works on multiple platforms (Linux, Windows, and others), and supports various languages. **eSpeak** is commonly used in applications that require speech synthesis, such as accessibility tools, voice assistants, and other systems where converting text to speech is essential.

Here’s an overview of how **eSpeak** works, along with how to use it for TTS in Python:

### **Key Features of eSpeak:**

1. **Multi-language Support**: eSpeak supports a wide range of languages, including English, Spanish, French, German, and many more, allowing users to generate speech in multiple languages.
2. **Customizable Speech**: You can adjust various aspects of the generated speech, including speed, pitch, and volume.
3. **Lightweight**: eSpeak is optimized for low resource usage, making it suitable for applications on devices with limited processing power, such as embedded systems or Raspberry Pi.
4. **Phoneme-based**: eSpeak uses phoneme-based synthesis, meaning it breaks down text into phonetic components, which makes it more flexible in handling various languages and accents.

**CHAPTER 3**

**IMPLEMENTATION AND METHODOLOGY**

**3.1 CIRCUIT DIAGRAM**

The **Figure 6** illustrates the complete hardware setup used in the project.  
It shows the interconnection between the Raspberry Pi, camera, push button, and other peripherals

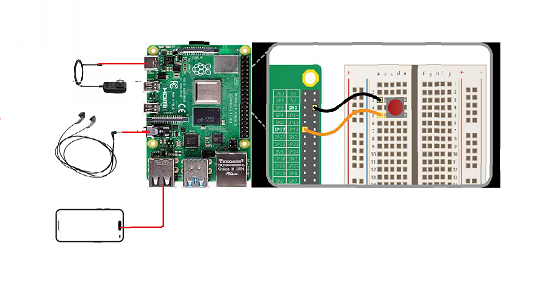
****

Figure 6 Circuit connection

## **3.1.1 HARDWARE INTERCONNECTION OVERVIEW**

The project utilizes the Raspberry Pi 4 as the **core controller**, coordinating real-time input and processing tasks. Key hardware devices include a USB-connected smartphone for capturing images, a push button for user interaction, and earphones for delivering audio output. Together, these elements create a closed-loop assistive technology system.

## **3.1.2 POWER SUPPLY SETUP**

The Raspberry Pi 4 is powered through its **USB Type-C port** using a 5V/3A adapter. A stable supply prevents operational issues such as device disconnection or unexpected shutdowns. For portable applications, a high-capacity power bank with 5V/3A output and overcurrent protection can be employed, ensuring uninterrupted system performance.

## **3.1.3 PUSH BUTTON WIRING**

The push button is connected between **GPIO17 (physical pin 11)** and **GND** on the Raspberry Pi. The internal pull-up resistor is enabled in the software configuration to maintain a defined HIGH state when idle and detect a LOW state upon button press. This method simplifies circuit design while ensuring reliable operation.

## **3.1.4 CAMERA INTEGRATION**

Instead of a traditional USB camera, a **smartphone using DroidCam** is connected via USB and recognized as a webcam by the Raspberry Pi. Captured frames are processed in real-time using OpenCV libraries, providing high-quality input for text detection and OCR modules.

## **3.1.5 EARPHONE CONNECTION**

Audio output is delivered through the **3.5mm jack** or a **USB audio device** connected to the Raspberry Pi. After text recognition, synthesized speech is generated using the eSpeak engine and routed to the earphones for private user feedback.

## **3.1.6 OVERALL SYSTEM FLOW**

When the push button is pressed, the Raspberry Pi initiates the image capture, processes the image for text detection and OCR recognition, and finally generates the speech output delivered to the earphones. This efficient sequence of operations allows visually impaired users to receive real-time auditory information from their environment.

**3.2 WORKFLOW**

Figure 7 illustrates the overall workflow of the proposed system.  
It outlines the step-by-step process from image capture to text recognition and conversion into speech using TTS.

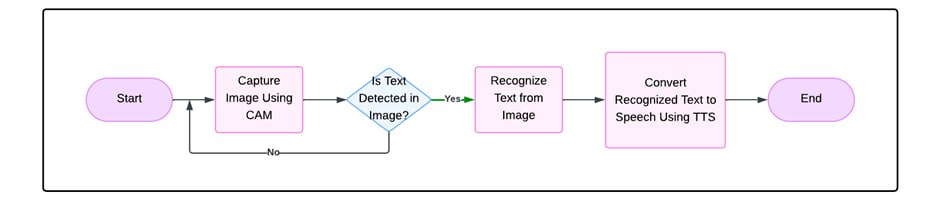
****

Figure 7 System workflow

#### **3.2.1 SYSTEM INITIALIZATION**

The operation begins as soon as power is supplied to the Raspberry Pi 4. The Linux-based operating system boots up and a pre-configured Python script is automatically launched through system startup services. During this initialization phase, the system configures the GPIO pins, setting the push button input with an internal pull-up resistor for stable detection. The USB camera is initialized using OpenCV, readying it to capture high-quality frames. Simultaneously, the deep learning-based text detection model built with Feature Pyramid Networks and Fully Convolutional Networks (FPN-FCN) is loaded into memory. The eSpeak text-to-speech engine is also initialized, ensuring that both text recognition and speech synthesis can occur without any delay. At the end of this phase, all hardware and software components are active and prepared for immediate user interaction.

#### **3.2.2 WAITING FOR USER INPUT**

Once initialized, the system transitions into an idle monitoring state where it continually polls the GPIO pin to detect a button press. It follows an event-driven approach to optimize power and processing resources, executing heavy tasks only when necessary. Lightweight polling ensures minimal processor load while keeping the system alert to any user input. Electrical stability is maintained through internal pull-up resistors, reducing the risk of false triggers due to noise. This monitoring phase ensures the system is ready to react instantly to the user's command.

#### **3.2.3 FRAME CAPTURE AND PRE-PROCESSING**

Upon detecting a button press, the system captures a frame using the camera. This frame serves as the primary input for the text detection process. To enhance detection accuracy, the captured frame undergoes pre-processing. It is first converted to grayscale to simplify data, followed by the application of noise reduction filters such as Gaussian blur. Additional techniques like edge enhancement may be used to make textual elements more distinct. The image is resized to match the deep learning model’s input requirements, ensuring optimal performance during text detection.

#### **3.2.4 TEXT DETECTION**

The pre-processed frame is fed into the loaded deep learning model, which uses FPN-FCN architecture to detect text regions at multiple scales. The model produces prediction maps highlighting probable text areas. Further refinement is carried out using thresholding to eliminate weak detections and Non-Maximum Suppression (NMS) to merge overlapping boxes, resulting in precise identification of text-containing regions. This stage ensures that only valid and clean text zones are passed on for recognition.

#### **3.2.5 OPTICAL CHARACTER RECOGNITION**

Once the text regions are detected, they are individually cropped and sent to the Tesseract OCR engine for recognition. Tesseract analyzes each region by segmenting it into lines, words, and characters, followed by feature extraction and pattern matching. Using its trained language models, Tesseract predicts the most likely characters and words, applying post-processing algorithms to correct minor errors. The final output is a machine-readable text string accurately representing the content in the captured frame.

#### **3.2.6 TEXT-TO-SPEECH (TTS) CONVERSION**

The recognized text is then passed to the eSpeak text-to-speech engine. The text undergoes normalization to ensure fluency, including handling numbers, abbreviations, and punctuation appropriately. eSpeak converts the normalized text into phonemes and synthesizes an audio waveform, which is played through the Raspberry Pi’s audio output. Users hear the spoken version of the detected text promptly, with adjustable parameters like pitch, speed, and volume for a personalized experience.

#### **3.2.7 RETURNING TO IDLE STATE**

After delivering the speech output, the system clears temporary data from memory to maintain high performance. It seamlessly returns to its idle monitoring mode, ready for the next button press. This cyclical workflow ensures continuous operation without manual resets, delivering a responsive and reliable user experience.

**3.3 DATASET DESCRIPTION**

The dataset used to train the FPN-FCN model includes images from public datasets like SROIE (Scanned Receipts OCR and Information Extraction) and FUNSD (Form Understanding in Noisy Scanned Documents). These datasets were selected for their diversity, containing text instances ranging from handwritten characters on receipts to structured and unstructured text in forms and noisy scanned documents.

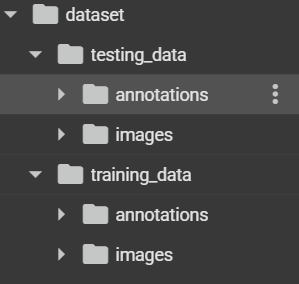


Figure 8 Organisation of dataset

The dataset consists of thousands of images, each with ground truth annotations in the form of bounding boxes or segmentation masks. For training, additional supervision maps are generated: kernel text maps, centerline maps, and height maps. These maps help the model focus on key text regions, understand text alignment and orientation, and distinguish between text instances of varying heights.

The FUNSD dataset contains 149 training images and annotations, and 50 images for testing. **Figure 8** shows the dataset organization, and **Figure 9** presents sample images.

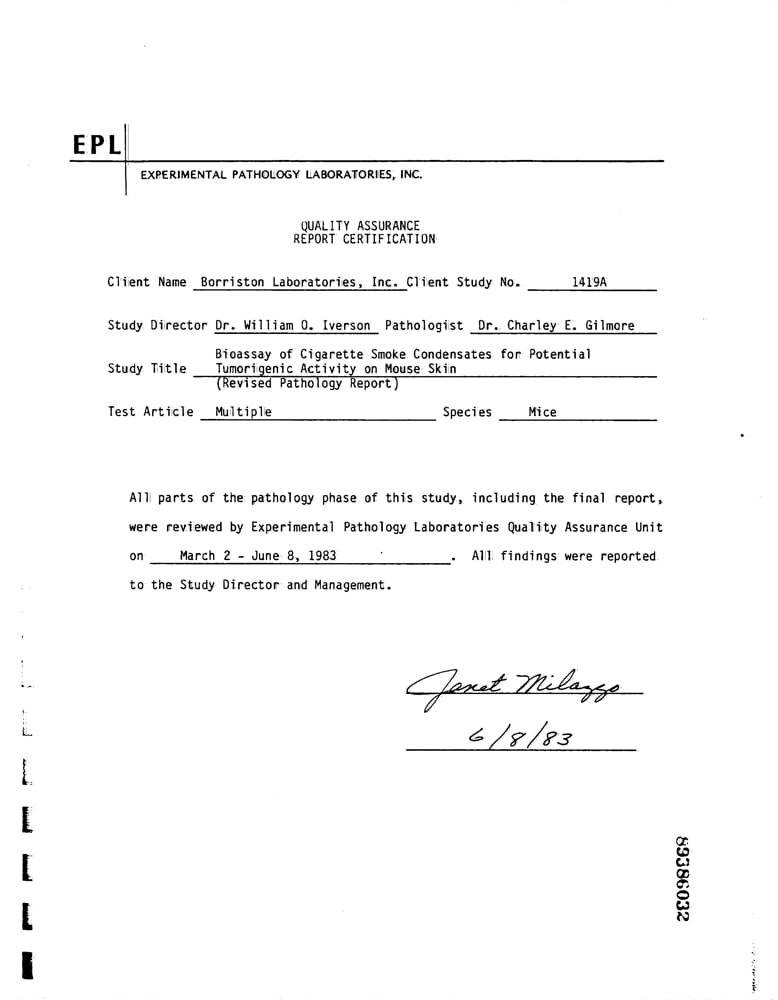
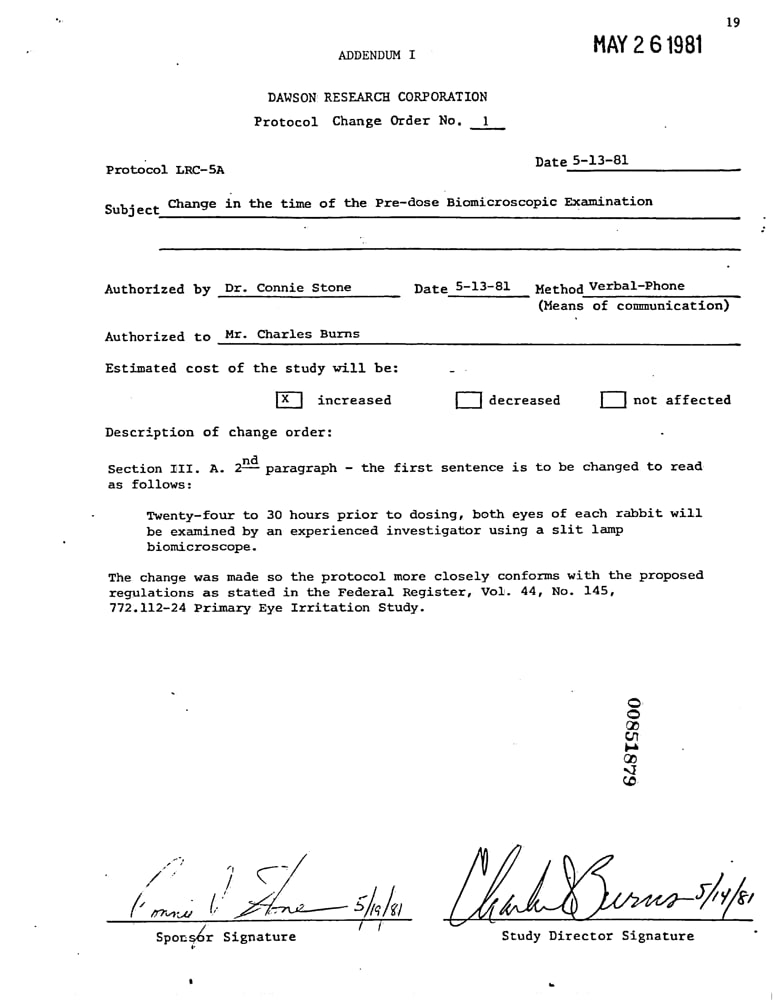
 

Figure 9 Sample dataset

### **3.4 INTRODUCTION TO FPN-FCN MODEL**

The Feature Pyramid Network (FPN) combined with the Fully Convolutional Network (FCN) is implemented to achieve robust text detection in document images. FPN enables the detection of text across multiple scales by constructing a multi-level feature pyramid, while FCN facilitates pixel-level classification of text regions by predicting dense output maps. Together, these architectures ensure that the system effectively detects text instances of various sizes and orientations within complex document layouts.

### **3.4.1 PREPROCESSING FOR TEXT DETECTION**

The preprocessing stage is crucial for optimizing input image quality before passing it to the FPN-FCN model. It consists of the following steps:

* **Grayscale Conversion**: The first step involves converting the image to grayscale. Since text detection primarily depends on character structure and contrast, grayscale conversion simplifies the data and retains only essential information, reducing computational complexity.
* **Noise Reduction**: Techniques such as Gaussian blur are applied to remove unwanted noise. This prevents artifacts from misleading the model and improves detection accuracy.
* **Edge Enhancement**: This step boosts the prominence of textual edges, helping the model distinguish text from background more effectively.
* **Image Resizing**: The image is resized such that both dimensions are divisible by 8, aligning with the model's downsampling operations. This adjustment ensures compatibility with the network architecture and prevents dimensional mismatches during up sampling.

**3.4.2 DATA PREPARATION**

The training images were preprocessed to create binary text kernel maps using their corresponding annotations. A custom data generator, implemented using TensorFlow’s Sequence class, dynamically loads training images and produces kernel maps on the fly. Multi-scale training was also employed, randomly resizing images between 320 and 1280 pixels (multiples of 8) to improve robustness across text sizes.

### **3.4.3 MODEL ARCHITECTURE AND TRAINING**

The model combines Feature Pyramid Network (FPN) and Fully Convolutional Network (FCN), consisting of three convolutional blocks with downsampling operations that progressively reduce the spatial dimensions of the input. The final prediction layer uses a 1×1 convolution followed by sigmoid activation to generate a text kernel map that highlights potential text regions.

Training was conducted using the **Adam optimizer** with an initial learning rate of **0.001** and **binary cross-entropy loss**. The model was trained for **50 epochs** with a **batch size of 1**, and multi-scale training was applied to improve generalization across different text sizes. Checkpoints were saved after each epoch to preserve model progress.

### **3.4.4 INFERENCE AND POST-PROCESSING**

During inference:

* The image is resized to dimensions divisible by 8,
* Passed through the trained FPN-FCN model,
* The predicted text kernel map is resized back to the original image size.

#### **Thresholding**

The soft probability text map generated by the FPN-FCN model is binarized by comparing each pixel value to a predefined threshold. Pixels with values above the threshold are classified as text and set to white, while others are considered background and set to black. This segmentation process effectively isolates candidate text regions while retaining essential structural information needed for accurate localization.

#### **Dilation**

To address fragmentation in the binary mask, **morphological dilation** is applied. This operation expands the white regions, effectively connecting broken or disjointed components of text. It ensures that individual letters or parts of words that were separated during earlier processing stages are properly unified into coherent text regions.

#### **Bounding Box Generation**

After the binary text map has been refined through thresholding and morphological operations such as dilation, the next crucial step in the text detection pipeline is to accurately localize the detected text regions. This is achieved using **Connected Component Analysis (CCA)**—a technique that identifies groups of adjacent pixels which share the same label (in this case, white pixels representing potential text regions). Each connected component represents a distinct text candidate that can potentially be enclosed within a bounding box.

Once these components are identified, the system calculates a **bounding rectangle** for each one. This rectangle tightly encloses the entire connected region, providing a spatial boundary that defines where the text appears within the image. These bounding boxes are characterized by their top-left coordinates (x, y), along with their width and height.

To improve the reliability of the detection process and reduce false positives, the system applies **post-processing filters** based on two key properties: **area** and **aspect ratio**. Very small components, which often arise due to noise or artifacts, are discarded using an area threshold, ensuring that only sufficiently large regions are retained. Similarly, aspect ratio checks are used to eliminate irregularly shaped boxes that are unlikely to contain readable text—for example, boxes that are extremely narrow or disproportionately tall.

This filtering mechanism helps in retaining only well-formed, significant text regions for the next stage of the pipeline. The output of this step is a refined set of bounding boxes that accurately enclose meaningful text segments. These regions are then cropped and forwarded to the **Optical Character Recognition (OCR)** stage—typically handled by the Tesseract engine—for converting the detected text into a machine-readable format.

**3.5 TEXT RECOGNITION USING TESSERACT**

Tesseract is an open-source Optical Character Recognition (OCR) engine used to extract text from images. It accurately converts the detected text regions into editable and readable text format.

## **3.5.1. LOADING THE TRAINED MODEL**

The trained FPN-FCN model, responsible for identifying text regions, is loaded. Since a custom loss function was used during training, it must be redefined before loading to ensure compatibility. The model weights and structure are restored, making it ready for inference on new inputs.

**3.5.2. OCR FOR RECOGNITION**

The detected text region image (from the previous stage) is preprocessed appropriately to match the model input size requirements. It is then fed into the loaded model for inference. The model predicts two outputs: a text probability map and a height map. However, only the text probability map is used for further processing, while the height map is discarded, as it is not required for text recognition.

The predicted text probability map undergoes thresholding to create a binary mask. Morphological operations like dilation are applied to connect text components.  
Connected component analysis is then performed to extract bounding boxes around each text region. These bounding boxes are sorted in reading order to maintain the correct sequence.

Each cropped bounding box is preprocessed with adaptive thresholding to enhance clarity. Tesseract OCR is applied to recognize the text within each region, using a specific configuration to improve accuracy .The recognized texts are cleaned to remove any special characters and duplicates, and then compiled into the final output.

### **3.6 ESPEAK SYNTHESIS**

Once the text has been successfully extracted from an image using OCR (Tesseract), it is passed to the eSpeak text-to-speech (TTS) engine for conversion into audible speech. This step is crucial for the accessibility feature of the system, especially for visually impaired users.

#### **3.6.1 TEXT INPUT AND PREPROCESSING**

The input to eSpeak is the final recognized text from the OCR module. Before the actual speech synthesis begins, eSpeak preprocesses this text to ensure it is suitable for conversion. This includes:

* Removing unnecessary characters or symbols,
* Converting numbers into their word equivalents (e.g., "101" becomes "one hundred and one"),
* Handling punctuation to control speech pauses and flow,
* Selecting the appropriate voice model and language based on user or system settings.

#### **3.6.2 PHONEME CONVERSION**

eSpeak then converts the cleaned text into phonemes, the basic sound units of speech. It uses a rule-based approach and a built-in dictionary to ensure correct pronunciation, even handling words not present in the dictionary using phonetic rules. This enables support for multiple languages and accurate speech generation.

#### **3.6.3 SPEECH SYNTHESIS**

Using formant synthesis, eSpeak generates speech from the phoneme sequence. This approach models the human vocal tract rather than relying on pre-recorded audio, allowing for lightweight and flexible TTS functionality. The system controls intonation, pitch, and emphasis to produce natural-sounding speech.

#### **3.6.4 AUDIO OUTPUT**

Finally, eSpeak outputs the synthesized speech as an audio waveform (usually PCM format), which is played through headphones or speakers connected to the Raspberry Pi. Speech rate and volume can be adjusted for user comfort. This real-time conversion ensures an efficient and user-friendly experience.

**3.7 HARDWARE SETUP-RASPBERRY PI INTEGRATION**

This section explains how various hardware components are connected and configured with the Raspberry Pi. Proper integration ensures seamless data flow between the camera, button, and audio output for real-time operation.

## **3.7.1 OPERATING SYSTEM INSTALLATION AND CONFIGURATION**

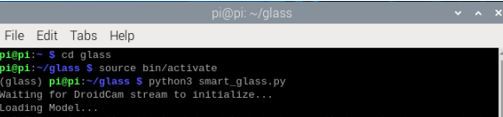
For setting up the Raspberry Pi hardware, the Raspberry Pi OS was installed using the Raspberry Pi Imager tool. After flashing the OS onto the SD card, an initial configuration was performed where the device was connected to Wi-Fi, SSH was enabled for secure remote access, and the time zone and locale settings were properly configured. The operating system and its packages were updated immediately to ensure stability and compatibility for the subsequent project dependencies.

**3.7.2 REMOTE ACCESS CONFIGURATION**

To enable headless operation (i.e., working without a dedicated monitor or keyboard), remote access tools were configured. RealVNC Viewer was enabled through the Raspberry Pi’s settings, allowing complete desktop access remotely. Additionally, WinSCP was set up to manage file transfers between the PC and the Raspberry Pi. It provided an efficient way to upload, download, and remotely edit files, helping in fast development and debugging without physical access to the Pi.

## **3.7.3 PYTHON ENVIRONMENT AND SOFTWARE TOOLS SETUP**

A dedicated Python virtual environment named glass was created to isolate project-specific libraries and dependencies, preventing conflicts with system-wide packages. Essential libraries such as OpenCV for image processing, NumPy for numerical computations, and additional modules related to text detection and text-to-speech (TTS) conversion were installed within this environment. For OCR (Optical Character Recognition), the Tesseract engine was integrated to accurately extract text from images. For converting recognized text to speech, the eSpeak library was utilized. The environment setup ensured that all necessary software tools worked seamlessly on the Raspberry Pi, optimizing the performance of the text-to-speech assistant system.

  
**Figure 10 Virtual Environment Activation and Script Execution**

**CHAPTER – 4**

**RESULTS AND DISCUSSION**

**4.1 OVERVIEW**

This chapter presents the results obtained from the implementation of the proposed system. The process involves several key stages, each designed to enhance the performance and efficiency of the pipeline. In this section, we will explore each stage in detail, providing relevant output images to illustrate the effects and outcomes of each step. The results showcase how the various components of the system work together to achieve the desired performance, highlighting the improvements and challenges encountered along the way.

The implementation of the system focuses on efficiently processing and analyzing input images, applying preprocessing techniques, detecting text regions, and finally recognizing the text content. These stages are crucial for ensuring that the system operates at optimal speed and accuracy. The following sections will delve into the specific outcomes of each processing phase.

**4.2 PREPROCESSING RESULTS**

Figure 10 illustrates the result of applying grayscale conversion to an input image. This step ensures that the system can concentrate on the text regions, which are typically darker or have high contrast compared to the background. In the example, an image originally measuring 860×1280 pixels is resized slightly to 864×1280 pixels to meet the system’s processing requirements. This minor resizing does not alter the core structure of the image but ensures that the input dimensions match the expected size for the pipeline.

The grayscale image output shown in Figure 10 highlights the significant reduction in complexity while maintaining the integrity of text regions. By eliminating unnecessary color information, the system is better equipped to handle the subsequent stages, such as text detection and recognition, without being overwhelmed by irrelevant data.

For example, an image originally having dimensions **860×1280** was resized to **864×1280** to meet this requirement, as shown in **Figure 11**.



Figure 11 demonstrates the output after applying adaptive thresholding

**4.3 TEXT DETECTION RESULTS**

The grey scale image is passed to the neural network and then the output (figure 12) is post processed. The post processing includes thresholding (figure 13), dilating (figure 14) and at last generating bounding box (figure 15).

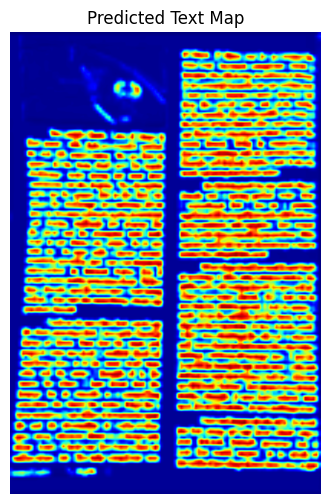


Figure 12 model output

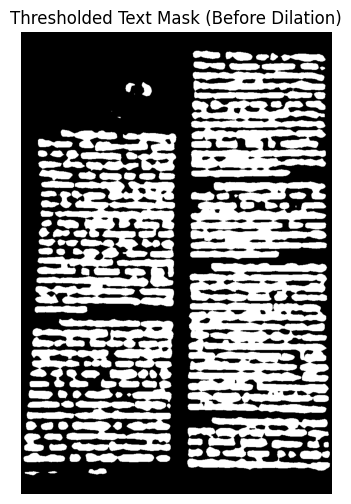


Figure 13 output after applying threshold

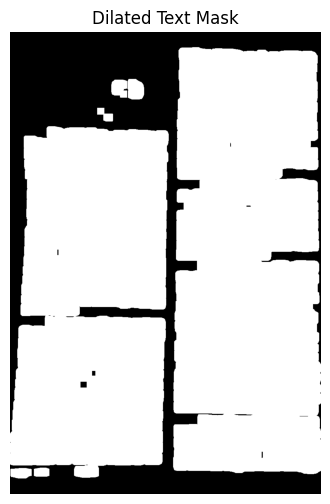


Figure 14 dilated output

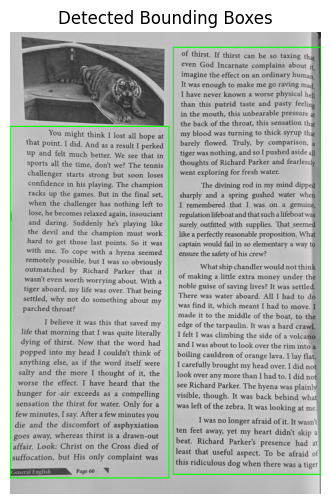


Figure 15 final text detection

**4.4 PERFORMANCE METRICES**

Evaluating the effectiveness and efficiency of the proposed system requires a thorough analysis using appropriate performance metrics. Performance metrics provide quantitative measures that help assess how well the system performs in terms of accuracy, speed, reliability, and resource utilization. These metrics are essential not only for validating the system's functionality but also for identifying areas where further optimizations can be made. In the context of the smart glass project, performance evaluation focuses on different stages such as text detection, recognition accuracy, response time, and the overall smoothness of the text-to-speech conversion pipeline. A detailed analysis of these metrics offers valuable insights into the real-world applicability of the system and highlights its strengths and potential areas for future improvement.

**4.4.1 IOU SCORE**

Intersection over Union (IoU) is a widely used evaluation metric in the field of computer vision, particularly for tasks such as object detection, instance segmentation, and semantic segmentation. It provides a quantitative measure of how accurately a predicted bounding box or mask overlaps with the ground truth annotation. In simple terms, IoU measures the degree of agreement between the predicted result and the actual labeled data by computing the ratio of their overlapping area to the total combined area. Mathematically, IoU is defined as the area of intersection between the predicted and ground truth regions divided by the area of their union. The resulting value ranges from 0 to 1, where a value of 1 indicates a perfect match between prediction and ground truth, and a value of 0 indicates no overlap at all. A higher IoU score reflects better localization and detection accuracy by the model. In practice, thresholds such as 0.5 or 0.75 are often used to determine whether a detection is considered successful. For example, an IoU ≥ 0.5 might be treated as a correct detection, while values below the threshold are considered false positives. IoU thus serves as a robust and interpretable metric for evaluating the performance of detection algorithms and is critical for comparing different models or configurations during training and validation.

**4.4.2 DICE COEFFICIENT:**

The **Dice Coefficient**, also known as the **Dice Similarity Score (DSC)**, is a statistical metric used to evaluate the accuracy of segmentation models, particularly in image analysis and medical imaging tasks. It quantifies how closely the predicted segmentation mask matches the ground truth mask by measuring the degree of overlap between the two. Mathematically, the Dice Coefficient is calculated as twice the area of intersection between the predicted and actual masks divided by the total number of pixels in both masks. The formula is typically expressed as:

where A is the set of predicted pixels and B is the set of ground truth pixels.

The value of the Dice Coefficient ranges from 0 to 1, where a value of **1** indicates a perfect match (complete overlap) and a value of **0** represents no overlap at all. A higher Dice score signifies better segmentation performance. Unlike simple pixel accuracy, the Dice Coefficient is especially useful in scenarios involving **imbalanced datasets**, where the region of interest (such as text in an image or a tumor in medical scans) occupies a small portion of the image. In such cases, traditional accuracy may be misleading, while the Dice score provides a more meaningful assessment of the model’s ability to correctly identify and delineate the relevant regions. Because of its sensitivity to both false positives and false negatives, the Dice Coefficient is considered a robust and reliable metric for evaluating segmentation quality in real-world applications.

**4.4.3 PRECISION**

**Precision** is a critical evaluation metric used in classification, object detection, and segmentation tasks, including text detection. It measures the **proportion of correctly identified instances** among all the instances that the model has predicted as positive. In the context of text detection, precision tells us **how many of the predicted text regions are actually correct**—that is, how many of them truly correspond to actual text in the image. Mathematically, precision is defined as

Here, **True Positives** refer to text regions that were correctly predicted by the model, while **False Positives** are those regions that were predicted as text but do not actually contain any. A high precision value indicates that the model makes very **few false positive predictions**, meaning it is good at avoiding incorrect detections. This is particularly important in applications where false alarms can be disruptive—such as assistive technologies for visually impaired users—where hearing non-existent or irrelevant text can lead to confusion. However, precision should often be considered alongside **recall**, as focusing too much on precision alone can result in the model becoming overly conservative, missing out on real text regions. Therefore, precision plays a vital role in evaluating how reliable and accurate a model's positive predictions are in real-world scenarios.

**4.4.4 RECALL**

**Recall** is a key performance metric used to evaluate how effectively a model identifies all relevant instances, particularly in object detection and segmentation tasks such as text localization. In the context of text detection, recall measures **how many of the actual text regions present in an image were correctly detected** by the model. It reflects the model’s ability to find all true instances of the target class—in this case, areas that truly contain text. Mathematically, recall is calculated as:

Here, **True Positives (TP)** are the correctly detected text regions, while **False Negatives (FN)** are the actual text regions that the model failed to detect. A high recall value indicates that the model is successful in capturing most of the actual text present in the image, minimizing the chance of missing critical information. This is especially important in assistive systems for the visually impaired, where missing important text could lead to an incomplete or misleading understanding of the environment. However, a model with high recall may sometimes produce more false positives, which is why recall is usually considered in tandem with **precision**. Balancing both helps in building a system that not only finds most of the relevant text but also avoids detecting irrelevant or incorrect regions.

**4.4.5 PIXEL ACCURACY:**

**Pixel Accuracy** is a fundamental evaluation metric in image segmentation tasks that measures how well a model classifies each individual pixel into the correct category—typically, in binary segmentation, as either **text** or **background**. In the context of text detection systems using segmentation-based approaches (such as FPN-FCN), pixel accuracy reflects the proportion of pixels that are correctly labeled by the model, regardless of the number or size of the text regions. It is calculated as the number of correctly classified pixels (both text and background) divided by the total number of pixels in the image. The formula is expressed as:

For instance, if a pixel that belongs to a text region is correctly labeled as "text," or if a background pixel is correctly labeled as "background," both are considered accurate predictions. A high pixel accuracy indicates that the model performs well at a granular level, correctly labeling the majority of the image. However, this metric may become **less informative in imbalanced datasets**, where background pixels significantly outnumber text pixels. In such cases, the model might achieve high accuracy by simply predicting most pixels as background. Therefore, while pixel accuracy provides a quick and intuitive measure of segmentation performance, it is often used alongside other metrics like the **Dice Coefficient** or **Intersection over Union (IoU)** to gain a more comprehensive understanding of model performance.

The selected performance metrics have demonstrated their effectiveness in providing a comprehensive evaluation of the system’s performance. Each metric contributed valuable insights into different aspects of the model’s accuracy, precision, and operational efficiency. Their systematic application enabled a detailed performance analysis, facilitating informed optimizations throughout the development process. Overall, the metrics proved instrumental in validating the robustness, reliability, and real-time suitability of the proposed solution for visually impaired users.

**4.5 OVERALL COMPARISON**

In figure 16, the comparison between DBNet and FPN-FCN across multiple performance metrics demonstrates that FPN-FCN consistently outperforms DBNet by a slight margin in all evaluated areas. Both models show a steady improvement from Mean IoU to Pixel Accuracy, with noticeable peaks at Precision and Pixel Accuracy. Although Recall scores are lower compared to other metrics for both models, FPN-FCN still maintains a marginal advantage. Overall, FPN-FCN delivers higher precision, better segmentation overlap (as shown by IoU and Dice scores), and superior pixel-wise accuracy, making it a more robust choice for the given task.

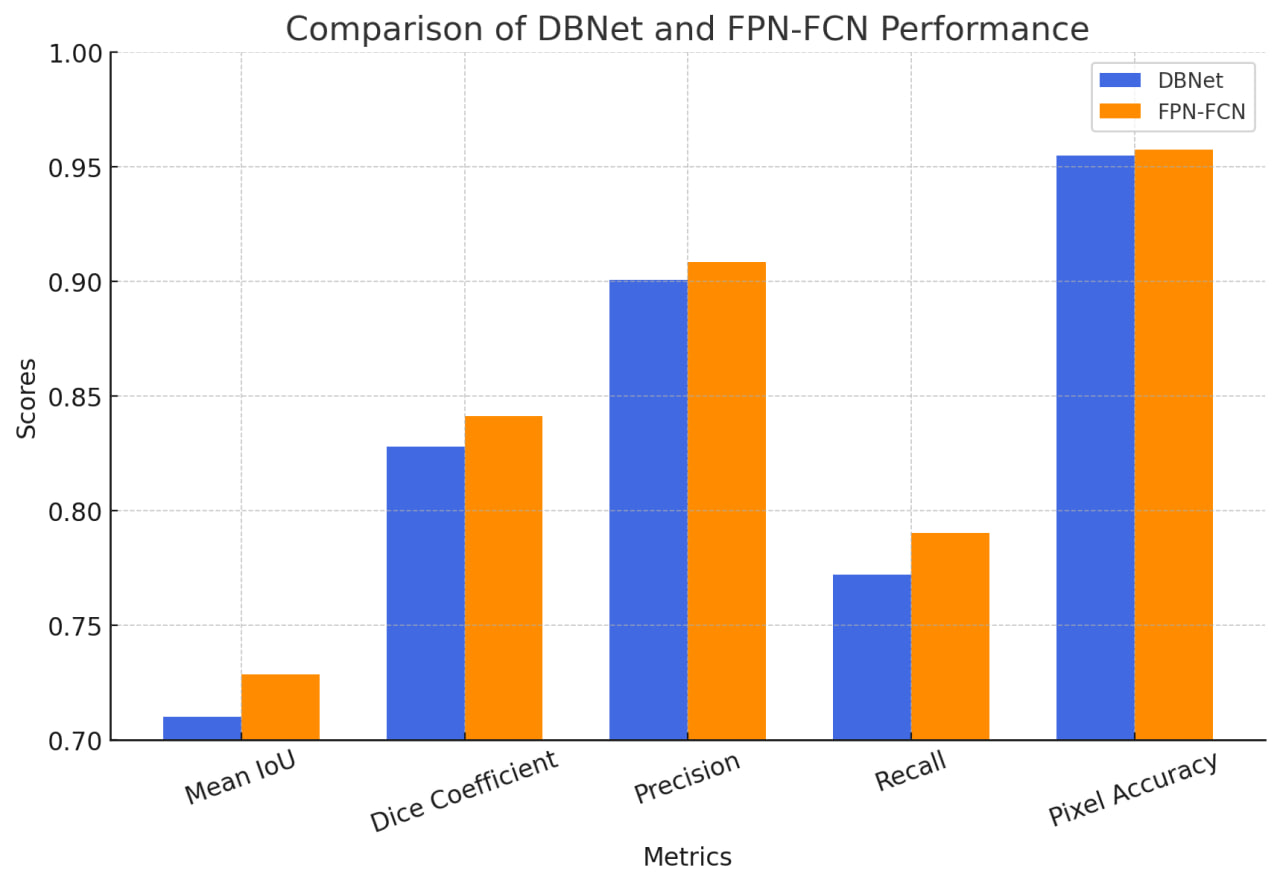
****

Figure 16 Bar Graph of Comparison of DBNet and FPN-FCN Performance

**Table 2** presents a comparative analysis of performance metrics between the FPN-FCN and DBNet models. It highlights their effectiveness in terms of accuracy, precision, recall, and F1-score for text detection tasks.

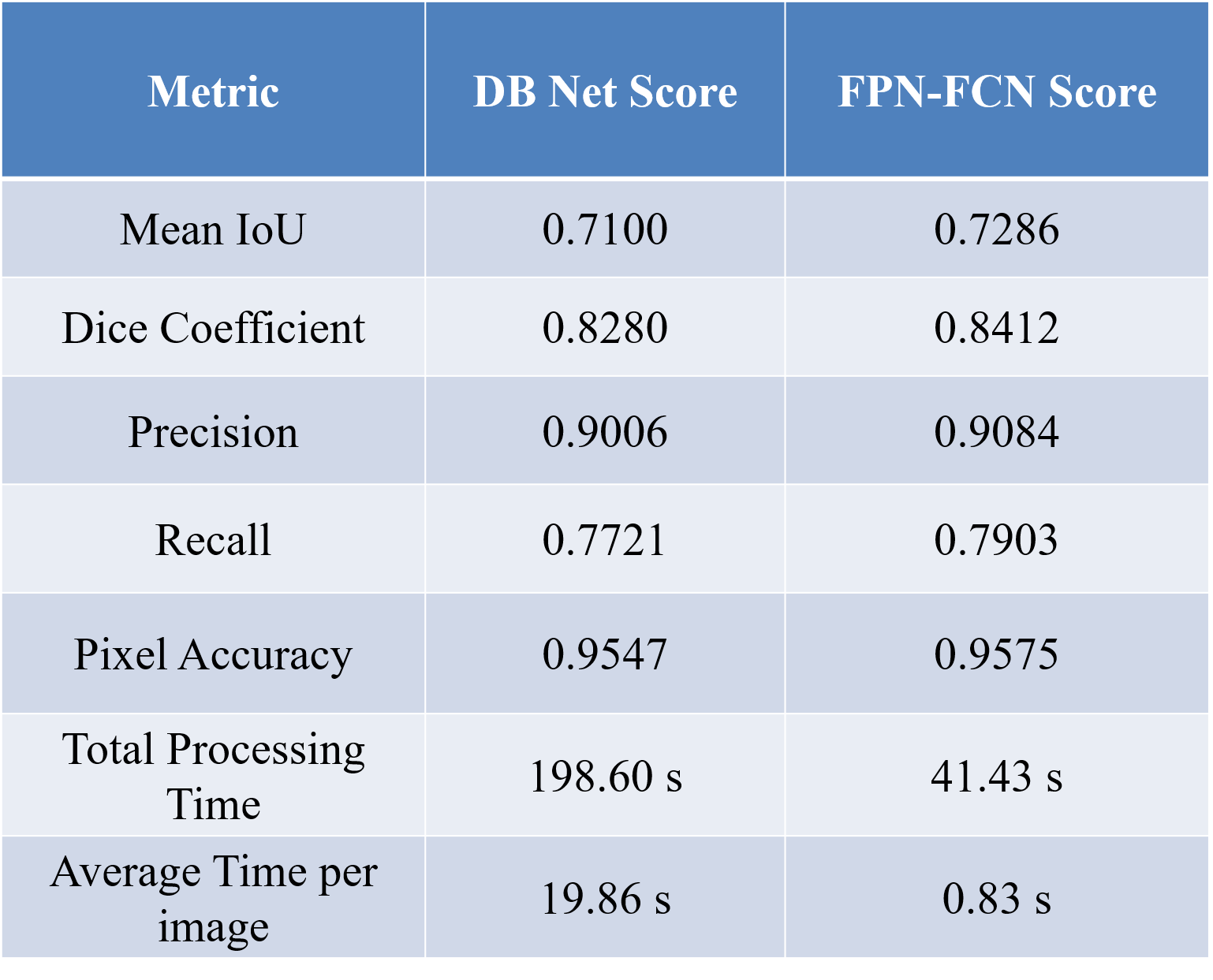


Table 2 Comparison of Performance Metrics between FPN-FCN and DBNet Models

**Inference:**

This FPN-FCN model outperforms DBNet in both accuracy and efficiency:

* Higher Accuracy: Achieves better Mean IoU, Dice Coefficient, Precision, Recall, and Pixel Accuracy.
* Faster Processing: 4.8× faster total processing time and 24× faster per image, making it ideal for real-time application.

**4.6 RESULT**

The results demonstrate the successful execution of the proposed system in real-time scenarios. Key outputs such as detected text, recognized content, and audio feedback validate the system’s effectiveness.

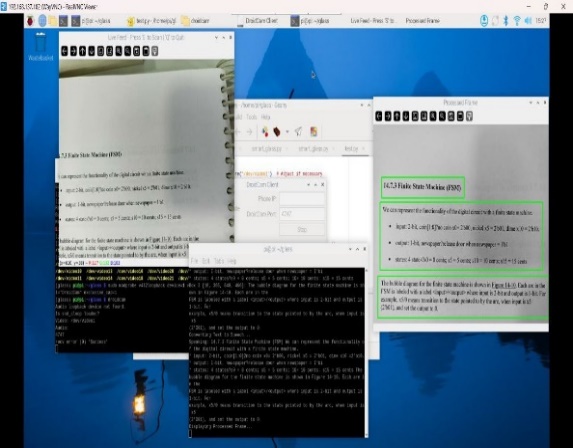
 

Figure 17 final output

**4.7 CHALLENGES FACED**

During the development process, several technical and hardware-related challenges were encountered. These included camera compatibility issues, real-time processing delays, and OCR accuracy under varying lighting conditions.

**4.7.1 ATTEMPT WITH RASPBERRY PI 3**

Initially, the project was implemented using a Raspberry Pi 3 as the processing unit. However, several performance-related challenges were encountered during the development phase. The primary issue was the limited computational power of the Raspberry Pi 3, which resulted in significant delays during image processing tasks. Moreover, there were persistent compatibility issues with TensorFlow, as the version supported by Raspberry Pi 3 was outdated and lacked many optimizations necessary for efficient model execution. The combination of hardware limitations and software incompatibility severely impacted the system's ability to run real-time text detection and recognition models, ultimately making the Raspberry Pi 3 unsuitable for the project’s performance requirements.

**4.7.2 EVALUATION OF PI CAMERA**

Following the issues faced with the processing unit, the choice of camera hardware was also evaluated. Initially, a **Pi Camera** was used for image acquisition due to its ease of integration with the Raspberry Pi’s native camera interface. The Pi Camera was simple to set up and did not require complex configurations, making it a convenient choice. However, it soon became evident that the Pi Camera suffered from **low resolution and limited image quality**. These limitations negatively affected the accuracy of both text detection and OCR processes, especially when dealing with smaller fonts or documents with poor lighting conditions. As a result, the Pi Camera was deemed insufficient for the system's requirements.

**4.7.3 EXPERIMENTATION WITH USB WEBCAM**

To overcome the image quality issues, a **USB webcam** was tested as the next alternative. Although USB webcams offer relatively better resolution compared to Pi Cameras, the development phase revealed several shortcomings. **Compatibility problems** arose frequently, especially with drivers and device recognition on Raspberry Pi operating systems. Additionally, **focus issues** were encountered, as many USB webcams lacked the ability to maintain sharp focus at varying distances. Despite multiple adjustments and trials, the USB webcam setup still failed to deliver the consistent and clear image quality needed for reliable text detection and recognition.

**4.7.4 FINAL DECISION: SMARTPHONE CAMERA**

After exploring multiple hardware options, the final decision was to **utilize a smartphone camera** connected over USB as the primary image acquisition device. Smartphones inherently come equipped with **high-resolution cameras** capable of capturing detailed and clear images under various lighting conditions. This transition significantly **improved the accuracy of text detection and OCR** operations, as the system could now work with higher quality inputs. The smartphone camera offered flexibility, ease of use, and superior imaging performance compared to previous options, thereby becoming the ideal choice for the project’s requirements.

**CHAPTER – 5**

**CONCLUSION AND FUTURE WORK**

### **5.1 SUMMARY**

This project introduces a fully offline, AI-powered **Text-to-Speech (TTS) assistant system** built around the Raspberry Pi 4 platform, specifically designed to aid **visually impaired individuals** in accessing textual information from their surroundings. The proposed system combines several intelligent modules including **image acquisition**, **text detection**, **OCR-based text recognition**, and **speech synthesis**, creating a seamless and efficient pipeline from input to audible output.

The user initiates the system through a **simple push-button interface**, which triggers the Raspberry Pi to capture an image using a **high-resolution mobile phone camera**. This choice of using a smartphone camera was intentional, as it significantly improves the clarity and resolution of the captured images, enhancing both **text detection** and **recognition accuracy**.

Captured images are passed through a **Feature Pyramid Fully Convolutional Network (FPN-FCN)** for text region detection. These detected regions are then processed by the **Tesseract OCR engine** to extract readable text from the image. Finally, the recognized text is fed into the **eSpeak TTS engine**, which converts it into **clear, synthesized speech**.

The entire system is designed to function **entirely offline**, ensuring data privacy, low latency, and practical usability in low-resource or remote environments without internet access. Furthermore, the architecture is **modular and scalable**, meaning future enhancements—such as improved speech quality, real-time video feed support, and multilingual support—can be integrated without overhauling the entire design.

This project not only addresses a critical accessibility challenge but also showcases the potential of affordable embedded AI solutions in real-world assistive technologies.

### **5.2 KEY OUTCOMES**

The major accomplishments of the project can be summarized as follows:

* **End-to-End Pipeline**: Successfully developed a complete and integrated image-to-speech pipeline that includes image capture, deep learning-based text detection, OCR recognition, and speech output—all controlled via a simple hardware interface.
* **Offline and Cost-Effective**: Leveraged open-source tools and hardware to develop a system that works entirely offline, removing the need for costly cloud APIs or internet connectivity. This makes the system highly suitable for **rural areas**, **schools for the blind**, and **resource-constrained environments**.
* **Hardware Optimization**: The decision to use a **mobile phone camera** over a basic Raspberry Pi camera led to significantly better **text recognition accuracy**, especially for small or low-contrast text. This trade-off maintained low cost while improving performance.
* **Modular Software Design**: Individual modules like the FPN-FCN model, OCR engine, and TTS engine are independently operable and replaceable, allowing easy upgrades or replacements in future iterations.
* **User-Centric Design**: With a push-button interface and audio feedback, the system provides an **accessible, intuitive experience** for visually impaired users. The latency from image capture to audio output remains within acceptable limits, ensuring **real-time usability**.
* **Scalability**: The system’s modular design allows for future integrations such as support for **multiple languages**, **emotion-based TTS models**, **voice-based interaction**, and **Edge AI acceleration**.

### **5.3 LIMITATIONS**

Despite the system's strong performance, there are certain limitations that must be acknowledged:

* **Handwritten Text Recognition**: The system struggles with handwriting or cursive text, as both the OCR engine and the text detector are primarily trained on printed text datasets.
* **Lighting Sensitivity**: Poor lighting conditions or glare can affect image clarity and lead to inaccurate OCR results. Advanced preprocessing for lighting correction is not yet implemented.
* **Basic Speech Quality**: While eSpeak is lightweight and offline, the speech output is robotic compared to more modern neural TTS models. This may reduce user comfort over long periods.
* **Limited Interaction**: Currently, the system only supports a single trigger-action interaction. There is no feedback or control mechanism (e.g., repeat, pause, or stop commands).

### **5.4 FUTURE SCOPE**

Looking ahead, there are various promising directions in which this system can be improved and extended:

#### **5.4.1 REPLACE ESPEAK WITH NEURAL TTS MODELS**

Integration of advanced TTS models such as **Tacotron 2**, **FastSpeech**, or **VITS** would greatly enhance the **naturalness and expressiveness** of the speech. These models, when

optimized for edge devices, can provide real-time performance with better human-like output.

#### **5.4.2 MULTILINGUAL AND TRANSLATION SUPPORT**

By enabling support for multiple languages and possibly real-time translation, the system can be made more inclusive for users who speak different native languages. Libraries like **langdetect** and **Google's Translate API (offline versions)** can be considered.

#### **5.4.3 REAL-TIME VIDEO STREAM TEXT DETECTION**

Instead of processing one image per trigger, the system can be enhanced to process **continuous video feeds** and extract text in real-time. This would help in scenarios like reading street signs, labels, or product information while moving.

#### **5.4.4 EDGE AI ACCELERATION**

To meet real-time demands, AI accelerators like **Google Coral TPU**, **NVIDIA Jetson Nano**, or **Intel Movidius NCS** can be integrated. These accelerators will offload the model inference task and allow **faster and more power-efficient processing**.

#### **5.4.5 VOICE-CONTROLLED INTERFACE**

Adding **speech recognition capabilities** (e.g., using Vosk or Picovoice) would allow users to issue voice commands like “repeat,” “next,” or “pause,” enabling a **hands-free experience**—especially useful for users with mobility challenges.

#### **5.4.6 COMPANION MOBILE APP**

A dedicated mobile app could be developed to:

* Customize voice and language settings
* View or save detected text
* Connect wirelessly to the Pi via Bluetooth/Wi-Fi for control

#### **5.4.7 IMPROVED PREPROCESSING TECHNIQUES**

Implementing **adaptive thresholding, glare reduction**, and **contrast enhancement** would improve OCR accuracy, especially under varying environmental lighting conditions.

### **5.5 FINAL REMARKS**

This project represents a successful step toward the integration of embedded AI systems into practical, real-world assistive technologies. By combining deep learning, OCR, and speech synthesis in a compact and cost-effective manner, the system empowers visually impaired individuals to access printed text information independently. With continued development and user-centric refinements, this solution has the potential to contribute meaningfully to the domain of accessible technology, while also serving as a platform for future research and innovation in smart automation and AI-driven assistive devices.

**REFERENCES**

1. **A. Gayer and V. V. Arlazarov**, "MULDT: Multilingual Ultra-Lightweight Document Text Detection for Embedded Devices," *IEEE Access*, vol. 12, pp. 170530–170540, Oct. 2024. doi: 10.1109/ACCESS.2024.3474616.
2. **H. Bi et al.**, "HGR-Net: Hierarchical graph reasoning network for arbitrary shape scene text detection," *IEEE Transactions on Image Processing*, vol. 32, pp. 4142–4155, Aug. 2023. doi: 10.1109/TIP.2023.3294822.
3. **S.-X. Zhang et al.**, "Arbitrary shape text detection via boundary transformer," *IEEE Transactions on Multimedia*, vol. 26, pp. 1747–1760, Mar. 2023. doi: 10.1109/TMM.2023.3232151.
4. **M. Liao et al.**, "Real-time scene text detection with differentiable binarization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 5, pp. 11474–11481, May 2022. doi: 10.1109/CVPR.2017.283.
5. **A. V. Trusov et al.**, "p-im2col: Simple yet efficient convolution algorithm," *IEEE Access*, vol. 9, pp. 168162–168184, Feb. 2021. doi: 10.1109/ACCESS.2021.3059295.

***Appendix***

**Source code**

import cv2

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import pytesseract

import subprocess

from tensorflow.keras.models import load\_model

# ==============================

# Text-to-Speech Function

# ==============================

def text\_to\_speech(text):

if text.strip(): # Ensure there's text to speak

print(f"Speaking: {text}")

subprocess.run(["espeak","-s", "100","-p", "50","-v", "en+f3", text], check=True)

else:

print("No text detected for speech conversion.")

# ==============================

# Load the Trained Model

# ==============================

def custom\_loss(alpha=0.5):

def loss(y\_true, y\_pred):

y\_true\_resized = tf.image.resize(y\_true, tf.shape(y\_pred)[1:3])

L\_c\_pos = -y\_true\_resized \* tf.keras.backend.log(y\_pred + tf.keras.backend.epsilon())

L\_c\_neg = -(1 - y\_true\_resized) \* tf.keras.backend.log(1 - y\_pred + tf.keras.backend.epsilon())

return tf.keras.backend.mean(L\_c\_pos + alpha \* L\_c\_neg)

return loss

def load\_fpn\_fcn\_model(model\_path):

return load\_model(model\_path, custom\_objects={'custom\_loss': custom\_loss()})

# ==============================

# Preprocess the Frame (Instead of Image File)

# ==============================

def preprocess\_frame(frame):

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

orig\_h, orig\_w = gray\_frame.shape

new\_h = ((orig\_h // 8) + 1) \* 8 if orig\_h % 8 != 0 else orig\_h

new\_w = ((orig\_w // 8) + 1) \* 8 if orig\_w % 8 != 0 else orig\_w

resized\_frame = cv2.resize(gray\_frame, (new\_w, new\_h)) / 255.0

resized\_frame = np.expand\_dims(resized\_frame, axis=[0, -1])

return resized\_frame, (orig\_h, orig\_w), gray\_frame

# ==============================

# Predict Text Map

# ==============================

def predict\_maps(model, frame):

image, (orig\_h, orig\_w), original\_frame = preprocess\_frame(frame)

pred\_text\_map, pred\_height\_map = model.predict(image)

pred\_text\_map = cv2.resize(np.squeeze(pred\_text\_map), (orig\_w, orig\_h))

pred\_height\_map = cv2.resize(np.squeeze(pred\_height\_map), (orig\_w, orig\_h))

return original\_frame, pred\_text\_map, pred\_height\_map

# ==============================

# Generate Bounding Boxes from Text Map

# ==============================

def remove\_overlapping\_boxes(bounding\_boxes, iou\_threshold=0.5):

def iou(box1, box2):

x1, y1, x2, y2 = box1

x1\_, y1\_, x2\_, y2\_ = box2

inter\_x1 = max(x1, x1\_)

inter\_y1 = max(y1, y1\_)

inter\_x2 = min(x2, x2\_)

inter\_y2 = min(y2, y2\_)

inter\_area = max(0, inter\_x2 - inter\_x1) \* max(0, inter\_y2 - inter\_y1)

box1\_area = (x2 - x1) \* (y2 - y1)

box2\_area = (x2\_ - x1\_) \* (y2\_ - y1\_)

union\_area = box1\_area + box2\_area - inter\_area

return inter\_area / union\_area if union\_area > 0 else 0

filtered\_boxes = []

#bounding\_boxes.sort(key=lambda b: (b[2] - b[0]) \* (b[3] - b[1]), reverse=True) # Sort by area (largest first)

while bounding\_boxes:

current\_box = bounding\_boxes.pop(0)

filtered\_boxes.append(current\_box)

bounding\_boxes = [

box for box in bounding\_boxes if iou(current\_box, box) < iou\_threshold

]

return filtered\_boxes

# ==============================

# Generate Bounding Boxes

# ==============================

def generate\_bounding\_boxes(text\_map, original\_image, threshold=0.4, area\_ratio\_threshold=0.01):

h, w = original\_image.shape[:2]

text\_map\_resized = cv2.resize(text\_map, (w, h))

text\_map\_resized = (text\_map\_resized \* 255).astype(np.uint8)

\_, text\_mask = cv2.threshold(text\_map\_resized, int(threshold \* 255), 255, cv2.THRESH\_BINARY)

kernel = np.ones((7,7), np.uint8)

text\_mask = cv2.dilate(text\_mask, kernel, iterations=3)

num\_labels, labels, stats, \_ = cv2.connectedComponentsWithStats(text\_mask)

boxed\_image = cv2.cvtColor(original\_image, cv2.COLOR\_GRAY2BGR)

bounding\_boxes = []

image\_area = w \* h

for i in range(1, num\_labels):

x, y, w, h, area = stats[i]

if area > area\_ratio\_threshold \* image\_area: # Filter based on relative area

bounding\_boxes.append((x, y, x + w, y + h))

# Step 1: Remove completely nested boxes

filtered\_boxes = []

for box in bounding\_boxes:

x1, y1, x2, y2 = box

inside = False

for other in bounding\_boxes:

if other == box:

continue

ox1, oy1, ox2, oy2 = other

if ox1 <= x1 and oy1 <= y1 and ox2 >= x2 and oy2 >= y2: # If inside another box

inside = True

break

if not inside:

filtered\_boxes.append(box)

# Step 2: Remove smaller overlapping boxes

final\_boxes = remove\_overlapping\_boxes(filtered\_boxes, iou\_threshold=0.5)

# Draw final bounding boxes

for x1, y1, x2, y2 in final\_boxes:

cv2.rectangle(boxed\_image, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Sort final boxes top-to-bottom, left-to-right

final\_boxes.sort(key=lambda box: (box[1], box[0]))

return boxed\_image, final\_boxes

# ==============================

# Recognize Text Using Tesseract OCR

# ==============================

def recognize\_text(original\_frame, bounding\_boxes, lang="eng"):

recognized\_text = []

custom\_config = f"--psm 6 -l {lang}"

for i, (x1, y1, x2, y2) in enumerate(bounding\_boxes):

roi = original\_frame[y1:y2, x1:x2]

roi = cv2.cvtColor(roi, cv2.COLOR\_GRAY2RGB) if len(roi.shape) == 2 else roi

text = pytesseract.image\_to\_string(roi, config=custom\_config).strip() or "[No Text Detected]"

recognized\_text.append({"box": (x1, y1, x2, y2), "text": text})

print(f"Box {i+1} [{x1}, {y1}, {x2}, {y2}]: {text}")

return recognized\_text

# ==============================

# Real-Time Webcam Processing

# ==============================

def run\_real\_time\_ocr(model\_path, lang="eng"):

print("Loading Model...")

model = load\_fpn\_fcn\_model(model\_path)

print("Starting Webcam...")

cap = cv2.VideoCapture(1) # Open webcam

#cap = cv2.VideoCapture("http://192.168.31.244:8080/video")

while cap.isOpened():

ret, frame = cap.read()

if not ret:

print("Failed to capture image.")

break

cv2.imshow("Live Feed - Press 'S' to Scan | 'Q' to Quit", frame)

key = cv2.waitKey(1) & 0xFF

if key == ord('s'): # Press 's' to scan

print("Processing Frame...")

frame = (frame \* 0.6).astype(np.uint8)

original\_frame, text\_map, height\_map = predict\_maps(model, frame)

print("Generating Bounding Boxes...")

boxed\_frame, bounding\_boxes = generate\_bounding\_boxes(text\_map, original\_frame)

print("Performing OCR...")

recognized\_texts = recognize\_text(original\_frame, bounding\_boxes, lang)

# Remove empty text and placeholders

valid\_texts = [item["text"] for item in recognized\_texts if item["text"].strip() and item["text"] != "[No Text Detected]"]

if valid\_texts: # Only speak if valid text exists

full\_text = " ".join(valid\_texts)

print("Converting Text to Speech...")

text\_to\_speech(full\_text)

else:

print("No text detected, skipping speech.")

print("Displaying Processed Frame...")

cv2.imshow("Processed Frame", boxed\_frame)

elif key == ord('q'): # Press 'q' to quit

break

cap.release()

cv2.destroyAllWindows()

print("Webcam Closed.")

# ==============================

# Run the Real-Time Pipeline

# ==============================

if \_\_name\_\_ == "\_\_main\_\_":

model\_path = r"D:\PTU FILES\final project\fpn\_fcn\_checkpoint\_epoch\_50.h5"

run\_real\_time\_ocr(model\_path, lang="eng")