Software Specification Report
Text Detection Application using EasyOCR

Kaustubh Sonde

2022-B-06012004A

Contents

1 Executive Summary **3**

2 Introduction **3**
2.1 Purpose . . . 3
2.2 Scope . . . 3
2.3 Definitions, Acronyms, and Abbreviations . . . 3

3 System Overview **3**
3.1 System Architecture . . . 3
3.2 System Interface . . . 4
3.3 Dependencies . . . 4

4 Functional Specifications **4**
4.1 Image Upload . . . 4
4.2 Text Detection . . . 4
4.3 Result Presentation . . . 5
4.4 Resource Management . . . 7

5 Non-functional Requirements **7**
5.1 Performance . . . 7
5.2 Usability . . . 7

6 Technical Architecture **8**
6.1 Component Diagram . . . 8
6.2 Data Flow . . . 8
6.3 Code Structure . . . 9

7 Implementation Details **9**
7.1 Model Initialization . . . 9
7.2 Image Processing . . . 9
7.3 Temporary File Handling . . . 9

8 User Interface Design **10**
8.1 Header Section . . . 10
8.2 Content Section . . . 10
8.3 Results Section . . . 10

9 Future Enhancements **10**
9.1 Functionality Expansion . . . 10
9.2 Performance Improvements . . . 10
9.3 Interface Enhancements . . . 11
9.4 Functionality Expansion . . . 11
9.5 Performance Improvements . . . 11
9.6 Interface Enhancements . . . 11

10 Conclusion **11**

1 Executive Summary

This document provides a comprehensive software specification for a text detection and extraction application built with EasyOCR and Streamlit. The application allows users to upload images containing text and processes them using Optical Character Recognition (OCR) technology to identify, highlight, and extract textual content. The system provides a user-friendly web interface that displays both the original and processed images along with detailed information about the detected text. The application leverages modern computer vision techniques through the EasyOCR library to achieve accurate text detection across various types of images, including documents, signs, and screenshots. It represents a practical implementation of Natural Language Processing (NLP) concepts for text extraction from visual media.

2 Introduction

2.1 Purpose

The purpose of this software is to provide an accessible and efficient tool for text detection and extraction from images. By combining state-of-the-art OCR technology with a web-based interface, the application makes advanced text recognition capabilities available to users without requiring specialized technical knowledge.

2.2 Scope

The application enables users to:

- Upload images containing text in various formats (JPG, JPEG, PNG)
- View visual representations of detected text with bounding boxes
- Access extracted text content with confidence scores
- Process images with English text (with potential for expansion to other languages)

2.3 Definitions, Acronyms, and Abbreviations

- **OCR** - Optical Character Recognition, the technology for converting images of text into machine-encoded text
- **EasyOCR** - An open-source Python library for OCR that supports multiple languages
- **Streamlit** - An open-source Python library for creating web applications
- **OpenCV** (CV2) - Open Computer Vision, a library for computer vision tasks
- **Confidence Score** - A numerical measure indicating the system's certainty in the accuracy of detected text

3 System Overview

3.1 System Architecture

The application follows a straightforward architecture with the following components:

- **Web Interface Layer** - Built with Streamlit to provide user interaction capabilities
- **Processing Layer** - Handles image loading, OCR processing, and visual annotation
- **OCR Engine** - EasyOCR library that performs the core text detection functionality

3.2 System Interface

The system provides a web-based user interface with the following sections:

- Header with application title and description
- File upload widget for image selection
- Display panels showing original and processed images side by side
- Data table showing all detected text elements with confidence scores

3.3 Dependencies

The application relies on the following external libraries:

- Streamlit - For web application interface
- EasyOCR - For text detection and recognition
- OpenCV (cv2) - For image processing and annotation
- NumPy - For numerical operations
- Matplotlib - For visualization support
- PIL (Python Imaging Library) - For image handling
- Temporary file handling modules - For managing uploaded files

4 Functional Specifications

4.1 Image Upload

- **File Format Support:** The system shall provide a file upload widget that accepts JPG, JPEG, and PNG image formats. This restriction ensures compatibility with the EasyOCR processing pipeline while covering the most common image formats.
- **Temporary Storage:** Uploaded images shall be temporarily stored in the server's file system using Python's tempfile module. This approach prevents memory overload for large images while ensuring data privacy by automatic cleanup after processing.
- **Image Preview:** The system shall display the original image upon successful upload at an appropriate resolution within the interface. The image shall be scaled proportionally to fit within the column width while maintaining aspect ratio for optimal viewing.
- **Error Handling:** The system shall validate uploaded files to ensure they are valid image files of the supported formats. If validation fails, the system shall display an informative error message to guide the user.
- **File Size Handling:** The system shall accept images up to 5 MB in size to accommodate high-resolution documents while preventing server overload from extremely large files.

4.2 Text Detection

- **OCR Processing:** The system shall utilize EasyOCR to detect text regions within the uploaded image with support for varying text sizes, orientations, and fonts commonly found in documents, signage, and digital content.
- **Language Support:** The system shall identify text with primary support for English language content. The EasyOCR model shall be specifically initialized for English to optimize performance and accuracy for this language.

- **Confidence Measurement:** The system assigns confidence scores (ranging from 0.0 to 1.0) to each detected text element, representing the probability of correct recognition. This quantitative measure helps users assess the reliability of each detection.
- **Visual Annotation:** The system shall visually annotate the detected text by drawing green rectangular bounding boxes around text regions with a line thickness of 3 pixels for clear visibility. The boxes shall precisely encompass the text regions based on the coordinates provided by EasyOCR.
- **Text Labeling:** The system shall display the detected text content and its confidence score (formatted to two decimal places) above each bounding box using a standardized font (FONT_HERSHEY_SIMPLE with size 0.8 and green color for consistency with the bounding boxes.
- **Multi-line Text Handling:** The system shall properly detect and process multi-line text blocks, treating them according to EasyOCR's segmentation logic, which may identify them as separate elements or as a single element depending on spacing and layout.
- **Special Character Support:** The system shall support the detection of alphanumeric characters, common punctuation, and special symbols as supported by the EasyOCR English language model.

4.3 Result Presentation

- **Dual Image Display:** The system shall display the original image and the annotated image side by side in a two-column layout to facilitate easy comparison. Each column shall occupy approximately 50% of the available screen width.
- **Section Headers:** Each display panel shall have clear subheadings ("Original Image" and "Detected Text") with consistent formatting to establish visual hierarchy and improve navigation.
- **Tabular Results:** The system shall present all elements of the detected text in a structured table format with three columns: a sequential number (starting from 1), the actual text content, and the confidence score formatted in two decimal places.
- **Table Styling:** The results table shall use Streamlit's native table styling with alternating row colors, clear column headers, and appropriate cell padding for optimal readability.
- **Empty Results Handling:** The system must explicitly inform the user when no text is detected in the image through a clear message stating "No text detected in the image." This prevents confusion when processing images without textual content.
- **Sorting Capability:** The detection results shall be presented in a logical order based on the position of text in the image (generally top to bottom, left to right), corresponding to natural reading order when possible.
- **Status Indication:** During processing, the system shall display a spinner animation with accompanying text "Detecting text..." to indicate that analysis is in progress and prevent user confusion during processing delays.

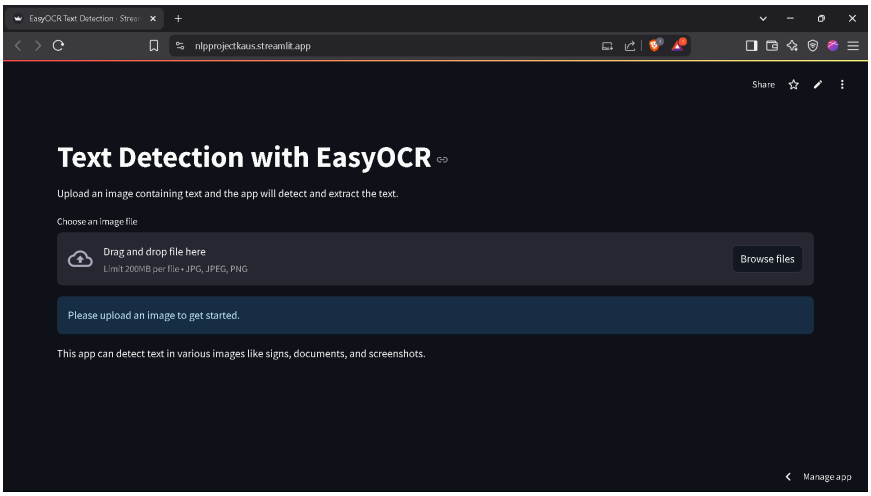

Figure 1: Main Interface of the Application

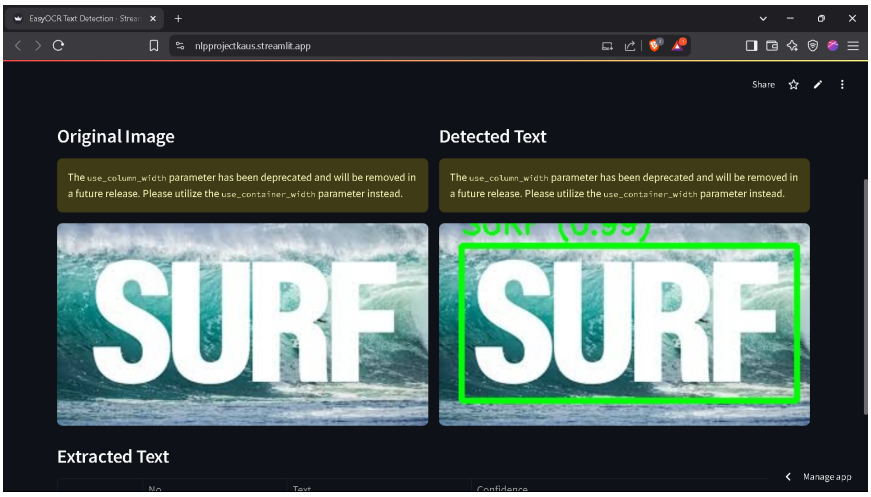

Figure 2: Dual Image Display

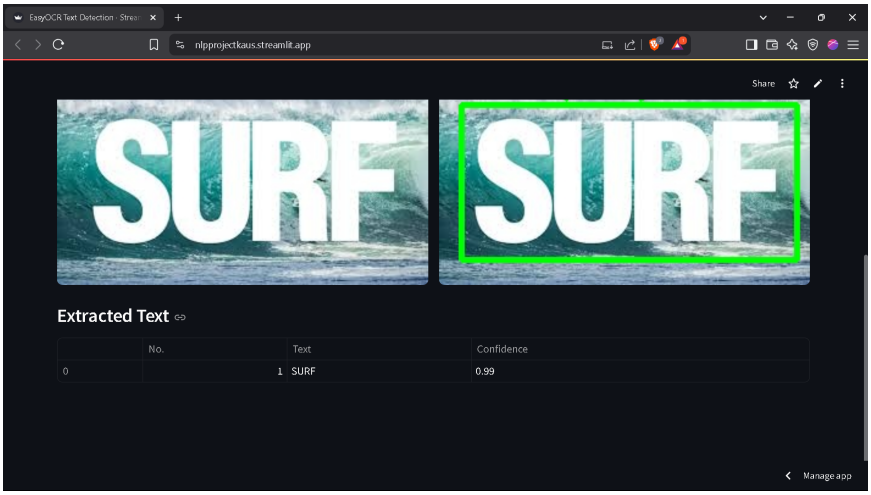

Figure 3: Tabular Results

4.4 Resource Management

- **Temporary File Cleanup:** The system shall explicitly remove temporary files after processing using os.unlink() to prevent the accumulation of user data and conserve storage space on the hosting server.
- **Model Caching:** The system shall implement caching for the EasyOCR model using Streamlit's @st.cache_resource decorator to prevent redundant loading of the large model file between user sessions, significantly reducing memory usage and improving responsiveness.
- **Memory Management:** The system must efficiently manage memory by releasing resources associated with processed images that are no longer needed, preventing memory leaks during extended usage sessions.
- **Concurrency Handling:** The system should properly handle concurrent user sessions without resource conflicts, ensuring that each user's data remain isolated and secure while maintaining consistent performance across multiple simultaneous connections.
- **Processing Queue Management:** When deployed in a multi-user environment, the system shall manage processing requests in a queue to prevent server overload and ensure fair resource allocation across users.

5 Non-functional Requirements

5.1 Performance

- **Model Loading Efficiency:** The application shall load the EasyOCR model efficiently using Streamlit's caching mechanism (@st.cache_resource). The model shall be loaded only once per server instance rather than per user request, reducing the initialization overhead from approximately 5-10 seconds to near-instantaneous for all but the first user.
- **Processing Feedback:** The application shall provide immediate visual feedback during processing through a prominently displayed spinner animation with the text "Detecting text..." This feedback shall appear immediately after the user uploads an image and remain until processing completes.
- **Processing Time:** The application should process most standard images (up to 2000×2000 pixels) in under 10 seconds on modern hardware (8GB RAM, quad-core processor). Processing time may increase linearly with image complexity (number of text elements) and resolution.
- **Response Time:** The application shall respond to user interactions (such as button clicks and file selection) within 500ms to maintain a sense of immediate feedback and control.
- **Image Rendering Speed:** Both the original and annotated images shall be rendered within 2 seconds of becoming available, regardless of the image dimensions. Streamlit's image scaling capabilities shall be utilized to optimize rendering performance.
- **Memory Utilization:** The application shall maintain a reasonable memory footprint, not exceeding 1GB RAM per user session under normal operation with typical image sizes. This constraint ensures scalability when deployed to standard cloud hosting environments.
- **Bandwidth Efficiency:** The application shall optimize image transfer between client and server, utilizing appropriate compression when displaying results to minimize bandwidth usage while maintaining visual clarity of text elements.

5.2 Usability

- **Intuitive Interface:** The interface shall follow a logical top-to-bottom flow (upload → view original → view processed → examine results) that aligns with users' mental models of document processing workflows. New users should be able to successfully process an image within 30 seconds of accessing the application without prior instruction.

- **Clear Instructions:** The application shall provide concise instructional text at each step of the process. The initial state shall display the message "Please upload an image to get started" along with supported formats, establishing clear first-step guidance.
- **Contextual Information:** The application shall provide additional context about its capabilities through the subtitle text: "This app can detect text in various images like signs, documents, and screenshots." This sets appropriate user expectations about the system's capabilities.
- **Visual Hierarchy:** The interface shall employ consistent visual hierarchy through section headers, spacing, and font sizing to clearly differentiate between functional areas (upload, original image, processed image, and results table).
- **Responsive Layout:** The layout shall adapt to different screen sizes through Streamlit's responsive design capabilities. On smaller screens, the two-column layout shall reflow to a single column while maintaining the logical sequence of elements.
- **Color Accessibility:** The application shall use a color scheme that maintains sufficient contrast ratios (minimum 4.5:1 for normal text) to ensure readability for users with visual impairments. The bright green (0, 255, 0) used for bounding boxes provides strong contrast against most image backgrounds.
- **Feedback Mechanisms:** The application shall provide clear feedback for all user actions, including successful uploads, processing status, and completion of analysis. Error states shall be communicated with specific, actionable messages rather than technical error co

6 Technical Architecture

6.1 Component Diagram

The application consists of the following major components:

Component	Description
User Interface	Streamlit-based web interface that handles user interactions and result display
OCR Model Loader	Cached resource that initializes and maintains the EasyOCR model
Image Processor	Component that handles image reading, OCR execution, and visual annotation
Result Formatter	Component that converts OCR results into tabular display format
File Manager	Component that handles temporary file creation and cleanup

6.2 Data Flow

1. User uploads an image through the Streamlit interface
2. Image is saved to a temporary file
3. Original image is displayed in the interface
4. The OCR model processes the image and returns text detection results
5. Image Processor annotates the detected text regions on a copy of the image
6. Annotated image is displayed in the interface
7. Text detection results are formatted into a table
8. Results table is displayed in the interface
9. Temporary file is deleted after processing completes

6.3 Code Structure

The application code is organized into the following logical sections:

1. Imports and configuration
2. Page setup and UI elements
3. Model initialization with caching
4. Image processing function
5. Main application flow
6. File handling and cleanup

7 Implementation Details

7.1 Model Initialization

The EasyOCR model is initialized using Streamlit's cache mechanism to prevent reloading:

```
@st.cache_resource
def load_model():
    return easyocr.Reader(['en'])
```

7.2 Image Processing

Image processing is handled by a dedicated function that detects text and visualizes results:

```
def process_image(image_path, reader):
    # Read the image
    img = cv2.imread(image_path)

    # Perform OCR
    result = reader.readtext(image_path)

    # Create a copy of the image for drawing
    img_with_boxes = img.copy()

    # Draw bounding boxes and text on the image
    for detection in result:
        top_left = tuple([int(val) for val in detection[0][0]])
        bottom_right = tuple([int(val) for val in detection[0][2]])
        text = detection[1]
        confidence = detection[2]

        # Draw rectangle
        cv2.rectangle(img_with_boxes, top_left, bottom_right, (0, 255, 0), 3)

        # Add text above the box
        cv2.putText(img_with_boxes, f"{text} ({confidence:.2f})",
                    (top_left[0], top_left[1] - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

    # Convert BGR to RGB for displaying with Streamlit
    img_with_boxes_rgb = cv2.cvtColor(img_with_boxes, cv2.COLOR_BGR2RGB)

    return img_with_boxes_rgb, result
```

7.3 Temporary File Handling

The application uses Python's tempfile module to manage uploaded images:

```
# Create a temporary file to save the uploaded image
temp_file = tempfile.NamedTemporaryFile(delete=False)
temp_filename = temp_file.name
temp_file.write(uploaded_file.getvalue())
temp_file.close()

# ... processing ...

# Clean up the temporary file
os.unlink(temp_filename)
```

8 User Interface Design

The application interface uses a clean, functional design with the following elements:

8.1 Header Section

- Title: "Text Detection with EasyOCR"
- Brief description explaining the application's purpose
- File uploader widget with supported file format indicators

8.2 Content Section

- Two-column layout displaying original and processed images side by side
- Subheadings for each section ("Original Image" and "Detected Text")
- Loading spinner to indicate processing status

8.3 Results Section

- Table displaying extracted text with column headers for number, text content, and confidence score
- Information message when no text is detected
- Instructions message when no image is uploaded

9 Future Enhancements

9.1 Functionality Expansion

- Support for additional languages beyond English
- Implementation of text search and highlight features
- Addition of image preprocessing options (contrast adjustment, noise reduction)
- Implementation of batch processing for multiple images
- Export options for extracted text (CSV, JSON, TXT)

9.2 Performance Improvements

- Advanced caching strategies for processed results
- Parallel processing for improved performance on multi-core systems
- Image size optimization before processing

9.3 Interface Enhancements

- Implementation of a dark mode option
- Addition of interactive elements for manual text selection
- Implementation of a results filtering system
- Visualization of confidence scores through color gradients

9.4 Functionality Expansion

- Support for additional languages beyond English
- Implementation of text search and highlight features
- Addition of image preprocessing options (contrast adjustment, noise reduction)
- Implementation of batch processing for multiple images
- Export options for extracted text (CSV, JSON, TXT)

9.5 Performance Improvements

- Advanced caching strategies for processed results
- Parallel processing for improved performance on multi-core systems
- Image size optimization before processing

9.6 Interface Enhancements

- Implementation of a dark mode option
- Addition of interactive elements for manual text selection
- Implementation of a results filtering system
- Visualization of confidence scores through color gradients

10 Conclusion

The Text Detection Application with EasyOCR represents an effective implementation of modern OCR technology in a user-friendly interface. Through its combination of leading libraries like EasyOCR, OpenCV, and Streamlit, the application provides valuable functionality for extracting textual information from images.

The modular design and well-structured code base allow for straightforward maintenance and future expansion. The current implementation fulfills the core requirements of text detection and extraction while establishing a foundation for additional features and improvements.

This application showcases the practical application of Natural Language Processing techniques in solving real-world problems related to text extraction from visual media.