Visvesvaraya Technological University, Belagavi – 590018

MINI-PROJECT REPORT
ON

OPTICAL CHARACTER RECOGNITION

Submitted in partial fulfillment for the award of degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE & ENGINEERING

Submitted by

Keerthana	4SO21CS073
Laksha	4SO21CS075
M Kushi Suvarna	4SO21CS082
Mantraj Tryphena David	4SO21CS083

Under the Guidance of

Ms Shiji Abrahman
Assistant Professor, Department of CSE

DEPT. OF COMPUTER SCIENCE AND ENGINEERING
ST JOSEPH ENGINEERING COLLEGE
An Autonomous Institution
(Affiliated to VTU Belagavi, Recognized by AICTE, Accredited by NBA)
Vamanjoor, Mangaluru - 575028, Karnataka
2023-24

ST JOSEPH ENGINEERING COLLEGE

An Autonomous Institution

(Affiliated to VTU Belagavi, Recognized by AICTE, Accredited by NBA)

Vamanjoor, Mangaluru - 575028, Karnataka

DEPT. OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Mini-project work entitled **"Optical Character Recognition"** carried out by

Keerthana	**4SO21CS073**
Laksha	**4SO21CS075**
M Kushi Suvarna	**4SO21CS082**
Mantraj Tryphena David	**4SO21CS083**

the bonafide students of VI semester Computer Science & Engineering in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2023-2024. It is certified that all suggestions indicated during internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Ms Shiji Abrahman	**Dr Sridevi Saralaya**
Mini-Project Coordinator	HOD-CSE

Abstract

This project focuses on developing an automated handwritten text recognition system using advanced machine learning techniques. The system integrates Optical Character Recognition (OCR) with modern text detection algorithms to efficiently convert handwritten text into a digital format. Our approach utilizes YOLO for text detection and Tesseract for character recognition. The project includes implementing a Flask-based web application where users can upload images, which are then processed through the YOLO model and Tesseract OCR. Our experimental work involves assessing the system's performance in real-world scenarios, analyzing text extraction accuracy, and optimizing processing times. The successful implementation of this system suggests promising future developments, such as creating a solution specifically for converting handwritten text from answer booklets into digital text. This enhancement will benefit educational institutions, exam boards, and administrative offices by automating the transcription process, thereby improving efficiency and accuracy in managing handwritten documents.

Table of Contents

List of Figures

Chapter 1

Introduction

The efficient processing of handwritten documents is becoming increasingly vital in today's world. Handwritten text recognition remains a challenging task due to the variability in individual handwriting styles and the complex nature of interpreting handwritten text accurately. This project seeks to address these challenges by developing an automated system designed to convert handwritten words into digital text. By leveraging advanced machine learning techniques, including Optical Character Recognition (OCR) and text detection algorithms, this system aims to streamline the transcription process and enhance the accuracy of text extraction from handwritten documents. The system will be implemented as a user-friendly web application, facilitating easy upload and processing of handwritten images

1.1 Background

Handwritten text recognition has been a challenging area of research due to the diverse and often inconsistent nature of human handwriting. As the volume of handwritten documents continues to grow, there is a pressing need for automated solutions that can accurately and efficiently convert these documents into digital formats. This project focuses on the specific aspect of handwritten text recognition, which involves converting handwritten words into digital text. By using advanced machine learning techniques, the project aims to provide a more accurate and efficient solution for processing handwritten documents, thus simplifying and speeding up the transcription process.

By integrating YOLO for text detection and Tesseract OCR for text recognition, the system aims to provide a robust solution for extracting text from handwritten documents. The combination of these technologies not only enhances the accuracy of text extraction but also improves the system's ability to handle varied handwriting styles.

1.2 Problem statement

The conversion of handwritten words into digital text presents a significant challenge due to the variability and complexity of handwriting styles. Each individual's handwriting is unique, and factors such as slant, spacing, and character formation can vary widely. These differences make it difficult for traditional Optical Character Recognition (OCR) systems to accurately interpret and convert handwritten text. Furthermore, handwriting can include cursive writing, mixed cases, and different writing instruments, adding to the complexity. This project aims to create a system that accurately and efficiently converts handwritten words into digital text, making it easier to handle and process handwritten documents. The primary goal is to create a robust solution capable of handling diverse handwriting styles, ensuring that the converted text is as accurate as possible.

1.3 Scope

This project focuses on developing a system to convert handwritten words into digital text. The scope includes implementing a machine learning-based approach that utilizes YOLO for text detection and Tesseract OCR for character recognition. The project involves creating a Flask-based web application where users can upload images of handwritten documents, which are then processed to extract and convert the text. The primary goal is to enhance accuracy and efficiency in recognizing handwritten text, with a future plan to handle handwritten answer booklets for educational institutions and administrative purposes.

Chapter 2

Software Requirements Specification

2.1 Introduction

This project specifies the requirements for a system designed to convert handwritten text into digital text. The system uses advanced machine-learning techniques to detect and recognize handwritten characters from images. It provides a web-based interface for users to upload images, which are then processed to extract and display the text.

2.2 Functional Requirements

- **User Authentication:** Users can securely log in and sign up to access the system.
- **Image Upload:** Users can upload images of handwritten documents via a user-friendly web interface. The system can support common image formats, including JPEG, PNG, and GIF.
- **Text Detection:** The system can use the YOLO model to identify and locate regions of text within the uploaded images.
- **Text Recognition:** The system can employ Tesseract OCR to convert the detected handwritten text into digital format after detecting text regions.
- **Result Display:** The system can display the converted text to the user in a clear and readable format.

2.3 Non-Functional Requirements

- **Performance**: The system processes and recognizes text from uploaded images quickly, providing results promptly.
- **Usability**: The web interface is user-friendly, allowing users to upload images and view results easily.

- **Compatibility**: The system works with common web browsers and supports major image formats like JPEG, PNG, and GIF.
- **Reliability**: The system operates consistently, handling errors gracefully and avoiding crashes or issues with invalid inputs.

2.4 User Interface Requirements

The user interface is designed to ensure a seamless and intuitive experience for users. It is structured to facilitate easy interaction with the system and provide clear instructions for each task.

2.4.1 Login and Signup Pages

The login and signup pages are designed with a clean and straightforward layout. Users can enter their login credentials with ease, and the forms include fields for username and password.

2.4.2 Image Upload Interface

The image upload interface is designed to allow users to easily select and upload images of handwritten documents. It supports common image formats such as JPEG, PNG, and GIF. The interface features an upload button and provides feedback on the status of the upload, including notifications for both successful and unsuccessful attempts.

2.4.3 Results Page

The results page displays the converted text extracted from the uploaded images in a readable format. It also shows the uploaded image with detected text regions, if applicable. Users can view the recognized text and have the option to upload another image through a dedicated button.

2.5 Software Requirements

This project requires several software components and configurations to function effectively:

2.5.1 Web Application Framework

This project utilizes the Flask-based web application framework. Flask is chosen for its simplicity and ease of integration with machine learning models, facilitating user interaction through a web interface where users can upload images and view results.

2.5.2 Machine Learning Libraries

This project employs the following machine-learning libraries:

- **YOLO (You Only Look Once):** Used for detecting text within uploaded images.
- **Tesseract OCR:** Used to convert the detected handwritten text into digital format.

These libraries must be installed and properly configured to ensure accurate text recognition and processing.

2.5.3 Web Browser Compatibility

This project's web application is compatible with common web browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge, to ensure accessibility for all users.

2.6 Hardware Requirements

This project requires specific hardware resources to support its operations:

2.6.1 Server Specifications

The server running the Flask application and machine learning models should have the following specifications:

- **CPU:** A modern multi-core processor to handle the computational demands of text detection and recognition.
- **RAM:** At least 8 GB of RAM to ensure smooth operation and performance.
- **Storage:** Sufficient storage space for image files and user data, with scalability considerations.

2.6.2 User Hardware

Users accessing this project's web application should have:

- **Internet Connection:** A stable internet connection for seamless interaction with the web application.
- **Computer or Mobile Device:** A computer or mobile device with a modern web browser.

Chapter 3

System Design

3.1 System Architecture

The system architecture is designed to facilitate efficient processing of handwritten text. The main components include the front-end interface, the back-end server, and the machine learning models.

Data flows from the user uploading an image to the server, where it is processed by the YOLO model and Tesseract OCR. The results are then sent back to the user interface for display.

3.2 Component Design

3.2.1 Front-End Design

The front end of the application is built using HTML, CSS, and JavaScript. It includes a simple layout for uploading images and viewing results. The interface is designed to be user-friendly and responsive.

- **Image Upload Interface:** A simple form for users to upload images of handwritten documents. The design ensures that users can easily select and submit their files.
- **Result Display:** After processing, the results are displayed on the same page, showcasing the extracted text alongside the original image for comparison.
- **Navigation:** It has links to the login, signup, and other functional pages. Ensures users can easily find the features they need.
- **Responsive Design:** The layout adjusts to various screen sizes and devices, providing a consistent experience across desktops, tablets, and smartphones.

3.2.2 Back-End Design

The back-end is implemented using Flask, a Python web framework. It handles user requests, processes images using the YOLO model for text detection and Tesseract OCR for text recognition, and returns the results to the front-end.

- **Image Upload Handling:** This "Uploads" folder is used to temporarily store the images that users upload via the web interface. When a user uploads an image, it is saved to this folder. This is where the initial, unprocessed images are kept before any processing occurs.
- **Text Detection and Recognition:** Utilizes the YOLO model to detect text regions within the uploaded images and Tesseract OCR to recognize and extract text from these regions. This step involves processing the image to identify and read handwritten text.
- **Result Generation:** Compiles the detected text and processed image, preparing them for display on the front end. The results are sent back to the user interface for clear and immediate viewing.
- **Detected Folder Creation:** This folder is used to store images that have been processed by the YOLO model and potentially modified by the text detection step.

3.2.3 Machine Learning Integration

The YOLO model is used to detect text regions in the uploaded images, while Tesseract OCR is employed to recognize the text within these regions. This section details how these models are integrated into the Flask application and how they interact with the data.

3.3 Database Design

Schema: The project does not utilize a database for storing data. All processing is performed in-memory and results are managed within the application's runtime.

3.4 User Interface Design

3.4.1 Layout and Navigation

The user interface is designed with a clean and intuitive layout. The main features include an image upload form and a results display area. Navigation is straightforward, allowing users to upload images and view results without difficulty.

3.5 Security Design

3.5.1 Authentication and Authorization

User authentication is managed through secure login and signup processes, ensuring that only authorized users can access the system.

3.5.2 Data Protection

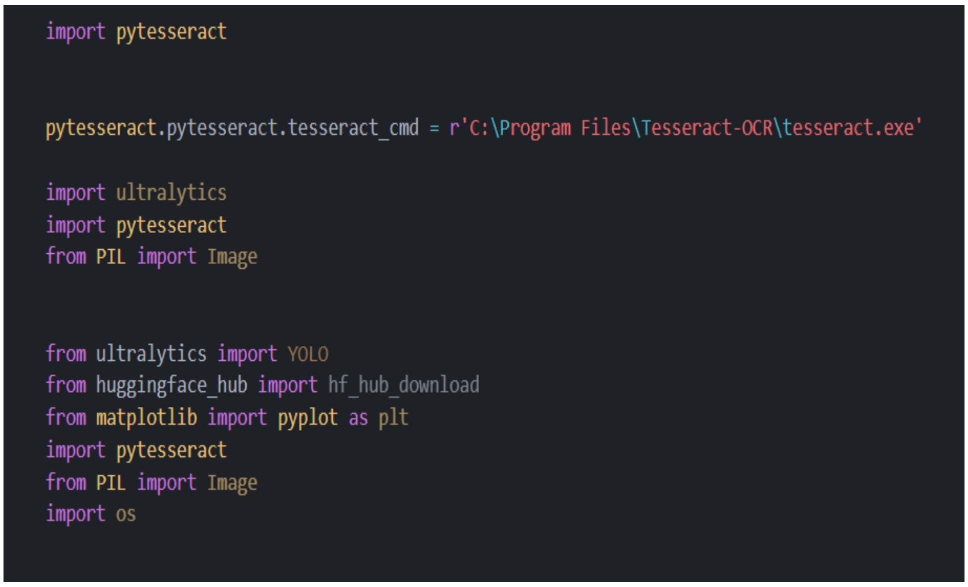
Data protection measures are in place to safeguard uploaded images and recognized text, ensuring that user data is handled securely.

Chapter 4

Implementation

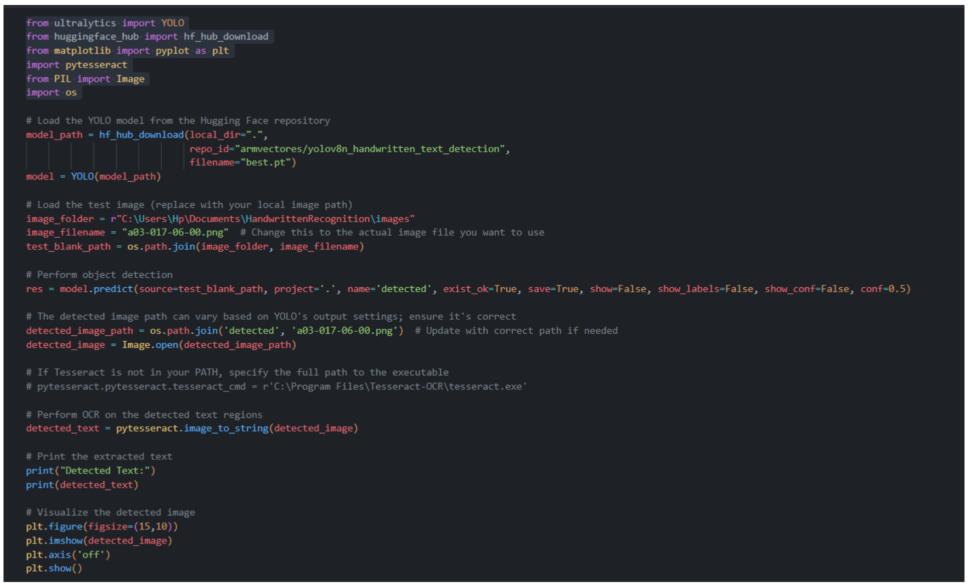
4.1 Text Detection

The text detection process is implemented using Jupyter Notebooks, which allow for interactive development and testing of machine learning models. It employs the YOLO (You Only Look Once) model to identify and locate text regions within images. The system utilizes this model to detect text areas, which are then processed for text recognition. This approach ensures effective handling of various handwriting styles and image conditions.



```
import pytesseract


pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

import ultralytics
import pytesseract
from PIL import Image


from ultralytics import YOLO
from huggingface_hub import hf_hub_download
from matplotlib import pyplot as plt
import pytesseract
from PIL import Image
import os
```

Figure 4.1: Library Imports and Initialization

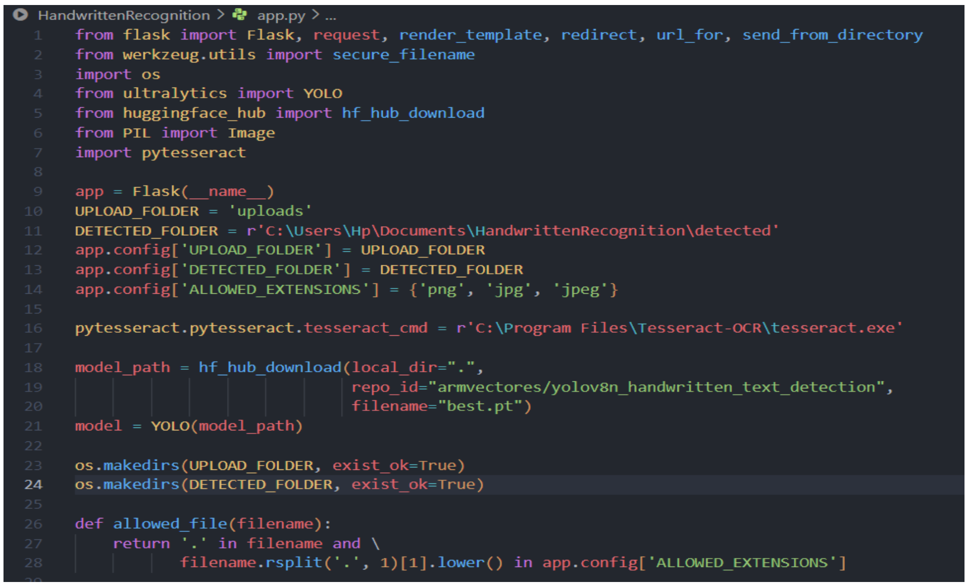


```python
from ultralytics import YOLO
from huggingface_hub import hf_hub_download
from matplotlib import pyplot as plt
import pytesseract
from PIL import Image
import os

# Load the YOLO model from the Hugging Face repository
model_path = hf_hub_download(local_dir=".",
                             repo_id="armvectores/yolov8n_handwritten_text_detection",
                             filename="best.pt")
model = YOLO(model_path)

# Load the test image (replace with your local image path)
image_folder = r"C:\Users\Hp\Documents\HandwrittenRecognition\images"
image_filename = "a03-017-06-00.png"  # Change this to the actual image file you want to use
test_blank_path = os.path.join(image_folder, image_filename)

# Perform object detection
res = model.predict(source=test_blank_path, project='.', name='detected', exist_ok=True, save=True, show=False, show_labels=False, show_conf=False, conf=0.5)

# The detected image path can vary based on YOLO's output settings; ensure it's correct
detected_image_path = os.path.join('detected', 'a03-017-06-00.png')  # Update with correct path if needed
detected_image = Image.open(detected_image_path)

# If Tesseract is not in your PATH, specify the full path to the executable
# pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Perform OCR on the detected text regions
detected_text = pytesseract.image_to_string(detected_image)

# Print the extracted text
print("Detected Text:")
print(detected_text)

# Visualize the detected image
plt.figure(figsize=(15,10))
plt.imshow(detected_image)
plt.axis('off')
plt.show()
```

Figure 4.2: YOLO Model Loading and Text Extraction

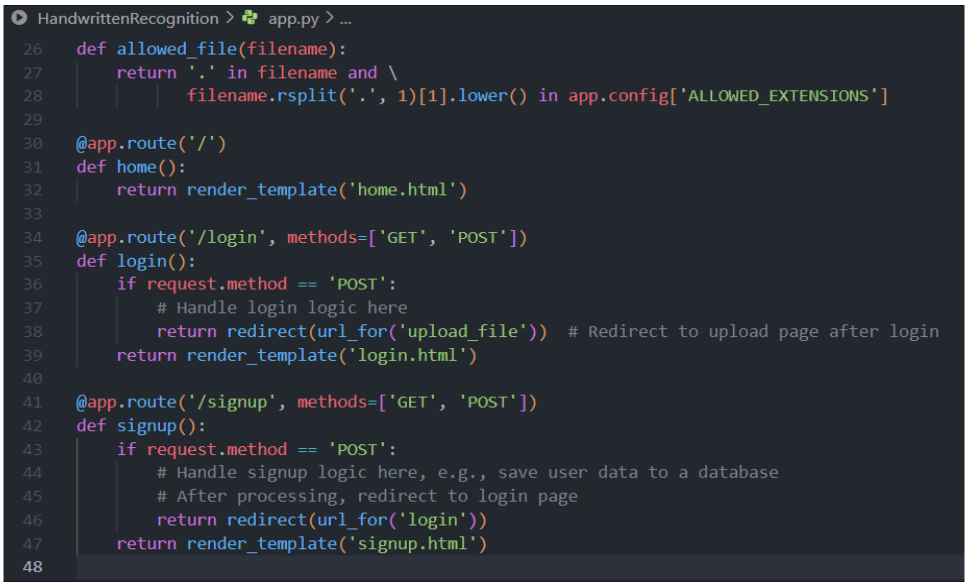
4.2 Integration with Python Backend

The integration of Python code into the web application is achieved using Flask, a Python web framework. Flask handles user requests and processes images using the YOLO model for text detection and Tesseract OCR for text recognition. It ensures smooth communication between the front end and the machine learning components, returning processed results to the user.



```python
from flask import Flask, request, render_template, redirect, url_for, send_from_directory
from werkzeug.utils import secure_filename
import os
from ultralytics import YOLO
from huggingface_hub import hf_hub_download
from PIL import Image
import pytesseract

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
DETECTED_FOLDER = r'C:\Users\Hp\Documents\HandwrittenRecognition\detected'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['DETECTED_FOLDER'] = DETECTED_FOLDER
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg'}

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

model_path = hf_hub_download(local_dir=".",
                             repo_id="armvectores/yolov8n_handwritten_text_detection",
                             filename="best.pt")
model = YOLO(model_path)

os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(DETECTED_FOLDER, exist_ok=True)

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']
```

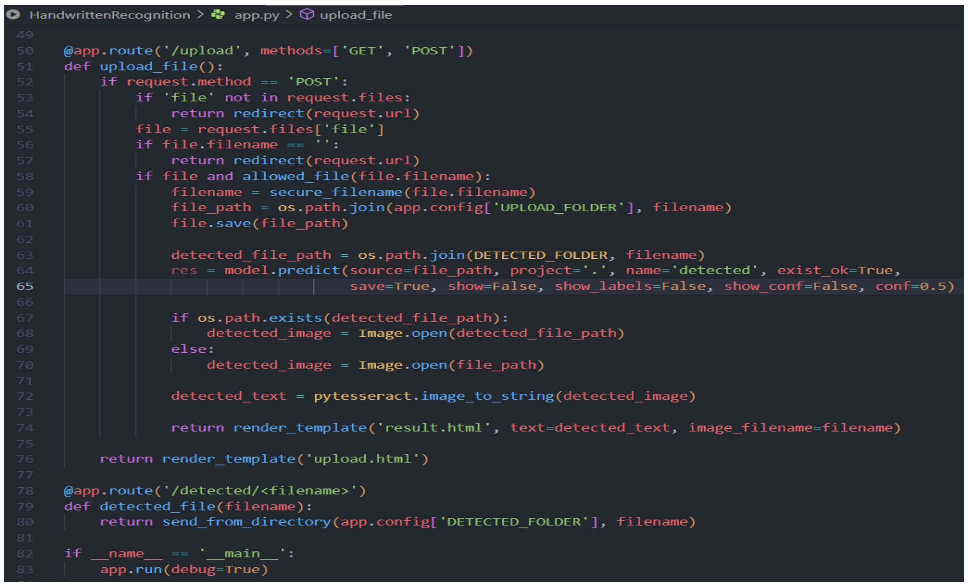
Figure 4.3: Application and Model Initialization



HandwrittenRecognition > app.py > ...

```python
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Handle login logic here
        return redirect(url_for('upload_file'))  # Redirect to upload page after login
    return render_template('login.html')

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        # Handle signup logic here, e.g., save user data to a database
        # After processing, redirect to login page
        return redirect(url_for('login'))
    return render_template('signup.html')
```

Figure 4.4: Routing and User Authentication



HandwrittenRecognition > app.py > upload_file

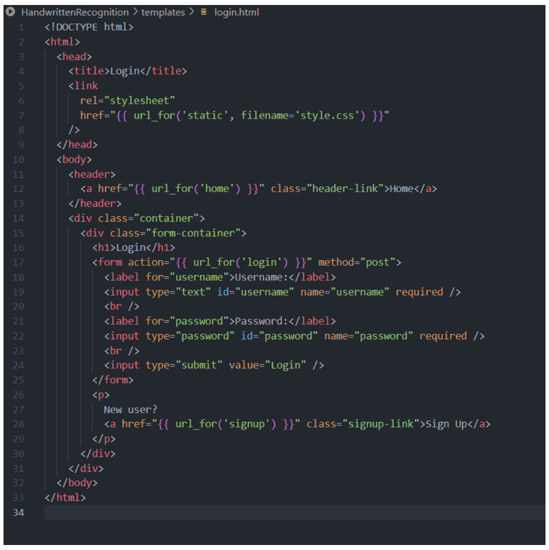
```python
@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(file_path)

            detected_file_path = os.path.join(DETECTED_FOLDER, filename)
            res = model.predict(source=file_path, project='.', name='detected', exist_ok=True,
                                save=True, show=False, show_labels=False, show_conf=False, conf=0.5)

            if os.path.exists(detected_file_path):
                detected_image = Image.open(detected_file_path)
            else:
                detected_image = Image.open(file_path)

            detected_text = pytesseract.image_to_string(detected_image)

            return render_template('result.html', text=detected_text, image_filename=filename)

    return render_template('upload.html')

@app.route('/detected/<filename>')
def detected_file(filename):
    return send_from_directory(app.config['DETECTED_FOLDER'], filename)

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 4.5: File Upload and Text Detection Processing

4.3 Front-End Development

4.3.1 Front-End Design

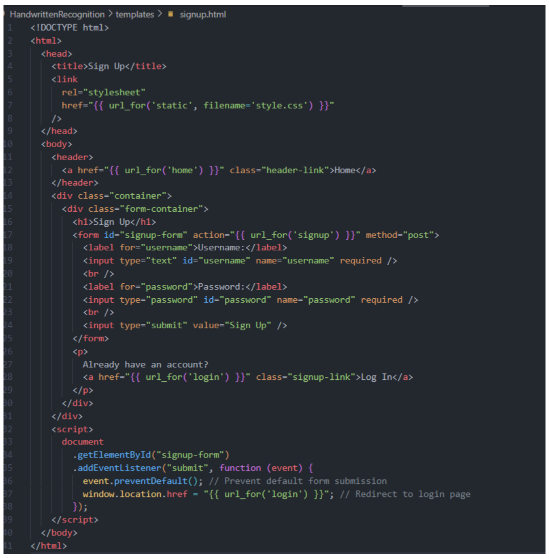
The front end of the application is developed using HTML, CSS, and JavaScript to provide a user-friendly interface. The layout includes an image upload form and result display areas. Users can easily upload images and view the extracted text with minimal effort. The design is kept clean and intuitive, enhancing the user experience. The CSS styles are stored separately in the ‘style.css‘ file, ensuring a clear separation between the design and functionality.



HandwrittenRecognition > templates > login.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Login</title>
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body>
    <header>
      <a href="{{ url_for('home') }}" class="header-link">Home</a>
    </header>
    <div class="container">
      <div class="form-container">
        <h1>Login</h1>
        <form action="{{ url_for('login') }}" method="post">
          <label for="username">Username:</label>
          <input type="text" id="username" name="username" required />
          <br />
          <label for="password">Password:</label>
          <input type="password" id="password" name="password" required />
          <br />
          <input type="submit" value="Login" />
        </form>
        <p>
          New user?
          <a href="{{ url_for('signup') }}" class="signup-link">Sign Up</a>
        </p>
      </div>
    </div>
  </body>
</html>
```

(a) Code Snippet for Login Page

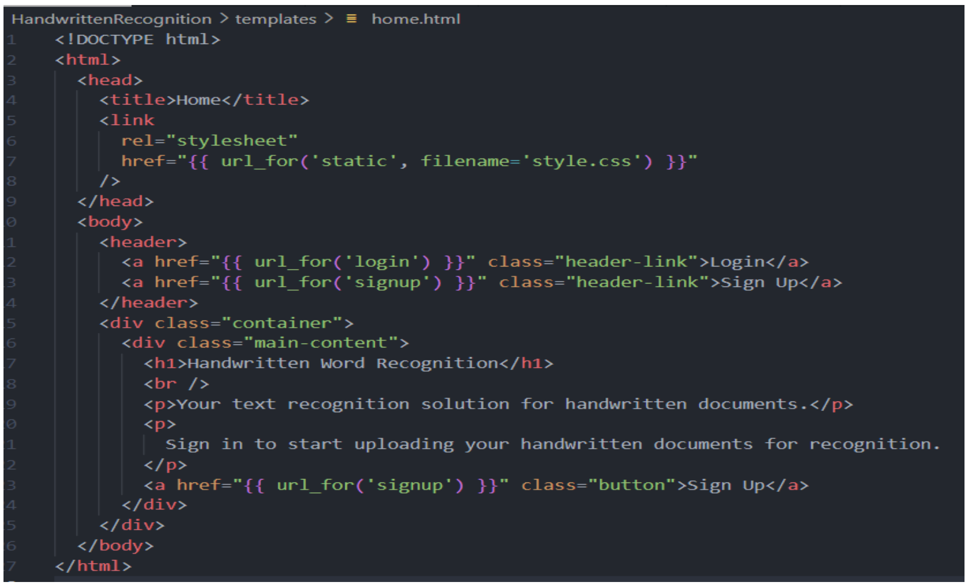


HandwrittenRecognition > templates > signup.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Sign Up</title>
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body>
    <header>
      <a href="{{ url_for('home') }}" class="header-link">Home</a>
    </header>
    <div class="container">
      <div class="form-container">
        <h1>Sign Up</h1>
        <form id="signup-form" action="{{ url_for('signup') }}" method="post">
          <label for="username">Username:</label>
          <input type="text" id="username" name="username" required />
          <br />
          <label for="password">Password:</label>
          <input type="password" id="password" name="password" required />
          <br />
          <input type="submit" value="Sign Up" />
        </form>
        <p>
          Already have an account?
          <a href="{{ url_for('login') }}" class="signup-link">Log In</a>
        </p>
      </div>
    </div>
    <script>
      document
        .getElementById("signup-form")
        .addEventListener("submit", function (event) {
          event.preventDefault(); // Prevent default form submission
          window.location.href = "{{ url_for('login') }}"; // Redirect to login page
        });
    </script>
  </body>
</html>
```

(b) Code Snippet for Signup Page

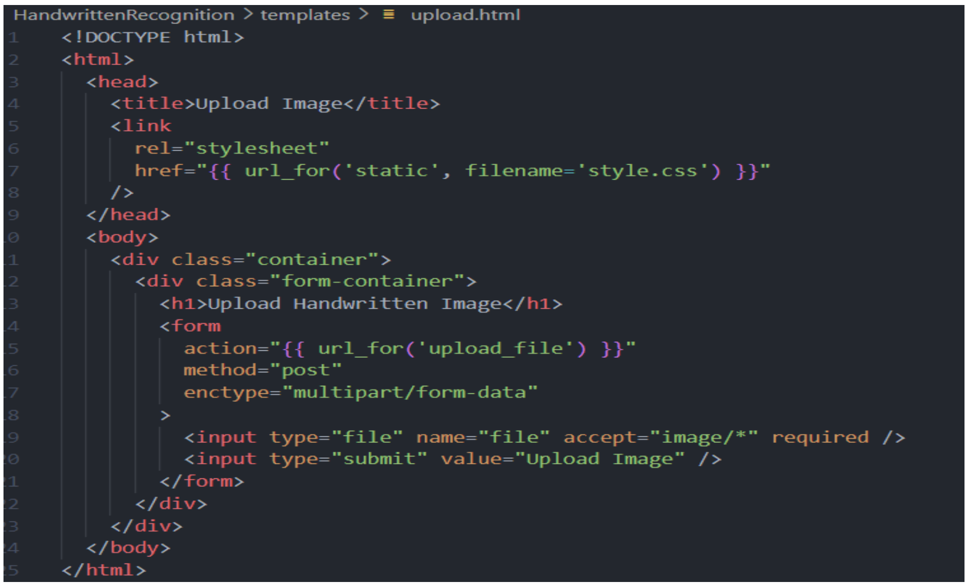
Figure 4.6: Code Snippets for Login and Signup Pages



HandwrittenRecognition > templates > home.html

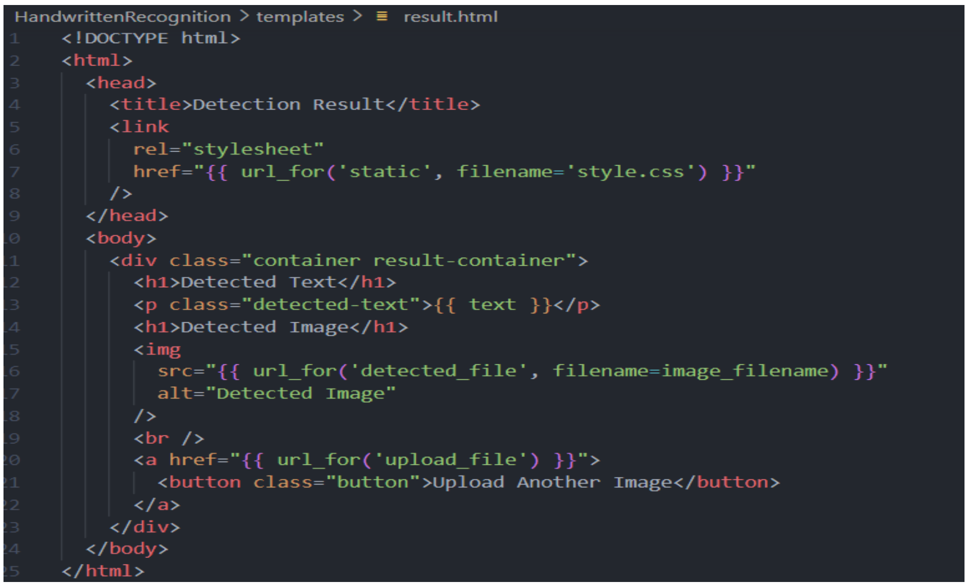
```html
<!DOCTYPE html>
<html>
  <head>
    <title>Home</title>
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body>
    <header>
      <a href="{{ url_for('login') }}" class="header-link">Login</a>
      <a href="{{ url_for('signup') }}" class="header-link">Sign Up</a>
    </header>
    <div class="container">
      <div class="main-content">
        <h1>Handwritten Word Recognition</h1>
        <br />
        <p>Your text recognition solution for handwritten documents.</p>
        <p>
          Sign in to start uploading your handwritten documents for recognition.
        </p>
        <a href="{{ url_for('signup') }}" class="button">Sign Up</a>
      </div>
    </div>
  </body>
</html>
```

Figure 4.7: Code Snippets for Home Page



```html
<!DOCTYPE html>
<html>
  <head>
    <title>Upload Image</title>
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body>
    <div class="container">
      <div class="form-container">
        <h1>Upload Handwritten Image</h1>
        <form
          action="{{ url_for('upload_file') }}"
          method="post"
          enctype="multipart/form-data"
        >
          <input type="file" name="file" accept="image/*" required />
          <input type="submit" value="Upload Image" />
        </form>
      </div>
    </div>
  </body>
</html>
```

Figure 4.8: Code Snippets for Upload Page

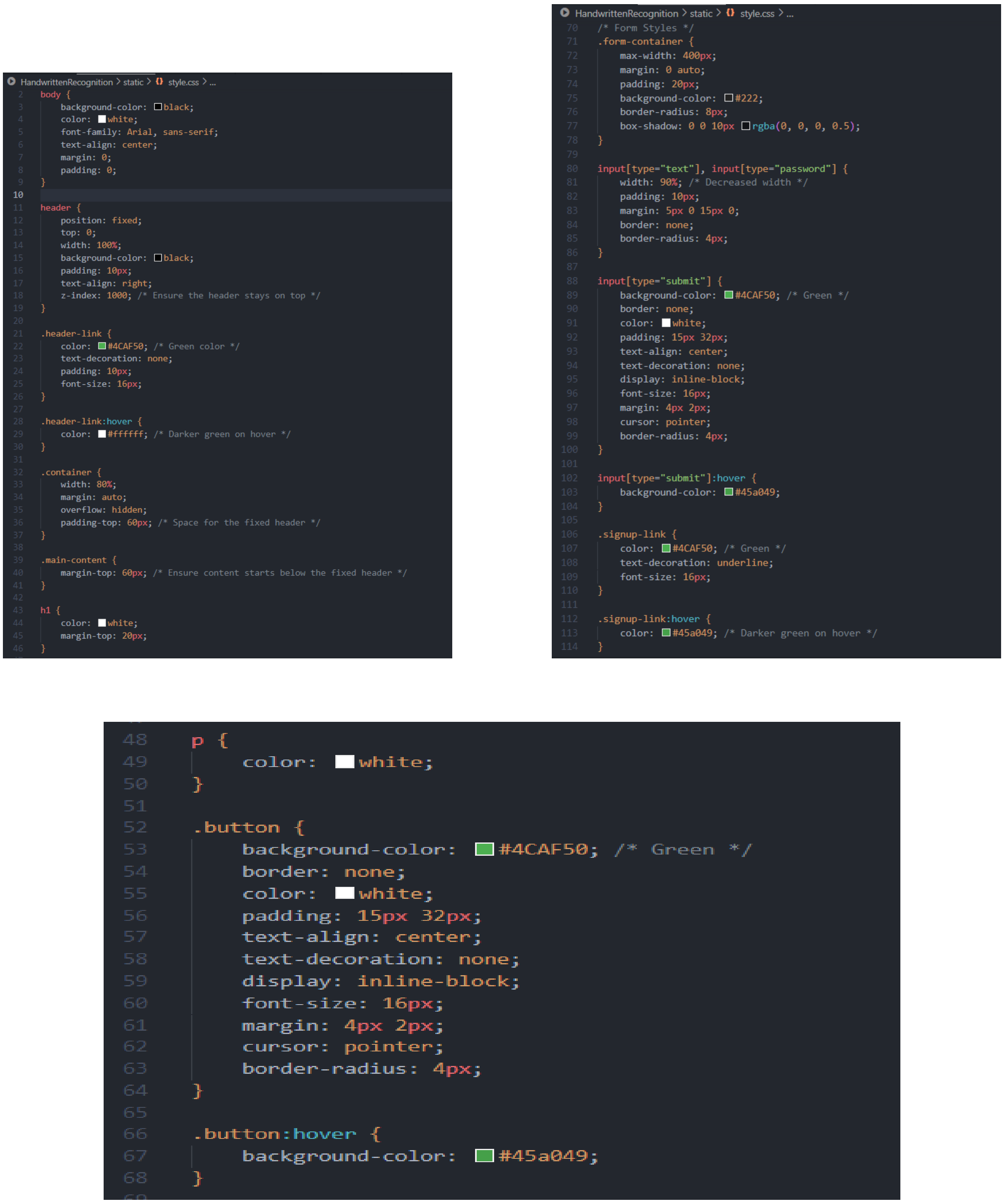


```html
<!DOCTYPE html>
<html>
  <head>
    <title>Detection Result</title>
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='style.css') }}"
    />
  </head>
  <body>
    <div class="container result-container">
      <h1>Detected Text</h1>
      <p class="detected-text">{{ text }}</p>
      <h1>Detected Image</h1>
      <img
        src="{{ url_for('detected_file', filename=image_filename) }}"
        alt="Detected Image"
      />
      <br />
      <a href="{{ url_for('upload_file') }}">
        <button class="button">Upload Another Image</button>
      </a>
    </div>
  </body>
</html>
```

Figure 4.9: Code Snippets for Result Page

4.4 System Integration

This project integrates the front-end, back-end, and machine learning components into a unified application. The front end allows users to upload images, which are then processed by the back end. The back end handles the text detection and recognition using machine learning models. The results are then displayed back to the user. This integration ensures a smooth workflow from uploading an image to receiving the recognized text, providing users with a seamless and efficient experience.



```css
body {
    background-color: black;
    color: white;
    font-family: Arial, sans-serif;
    text-align: center;
    margin: 0;
    padding: 0;
}

header {
    position: fixed;
    top: 0;
    width: 100%;
    background-color: black;
    padding: 10px;
    text-align: right;
    z-index: 1000; /* Ensure the header stays on top */
}

.header-link {
    color: #4CAF50; /* Green color */
    text-decoration: none;
    padding: 10px;
    font-size: 16px;
}

.header-link:hover {
    color: #ffffff; /* Darker green on hover */
}

.container {
    width: 80%;
    margin: auto;
    overflow: hidden;
    padding-top: 60px; /* Space for the fixed header */
}

.main-content {
    margin-top: 60px; /* Ensure content starts below the fixed header */
}

h1 {
    color: white;
    margin-top: 20px;
}
```

```css
/* Form Styles */
.form-container {
    max-width: 400px;
    margin: 0 auto;
    padding: 20px;
    background-color: #222;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
}

input[type="text"], input[type="password"] {
    width: 90%; /* Decreased width */
    padding: 10px;
    margin: 5px 0 15px 0;
    border: none;
    border-radius: 4px;
}

input[type="submit"] {
    background-color: #4CAF50; /* Green */
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    border-radius: 4px;
}

input[type="submit"]:hover {
    background-color: #45a049;
}

.signup-link {
    color: #4CAF50; /* Green */
    text-decoration: underline;
    font-size: 16px;
}

.signup-link:hover {
    color: #45a049; /* Darker green on hover */
}
```

```css
p {
    color: white;
}

.button {
    background-color: #4CAF50; /* Green */
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    border-radius: 4px;
}

.button:hover {
    background-color: #45a049;
}
```

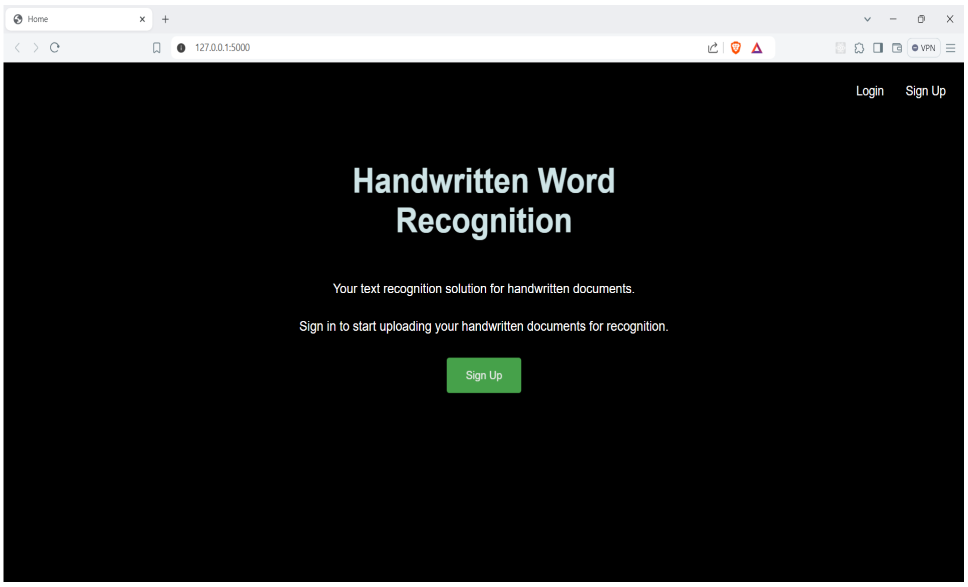
Figure 4.11: Stylesheets

Chapter 5

Results and Discussion

5.1 Home Page

The home page introduces users to the handwritten text recognition system. It provides an easy interface for users to navigate to the login or signup pages. Key features include options to upload images for text detection and access previous results. The design is clean, ensuring an easy start for all users.



Handwritten Word Recognition

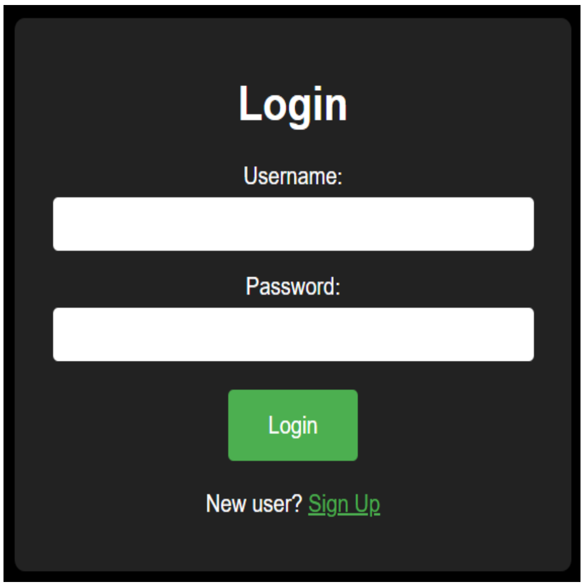
Your text recognition solution for handwritten documents.

Sign in to start uploading your handwritten documents for recognition.

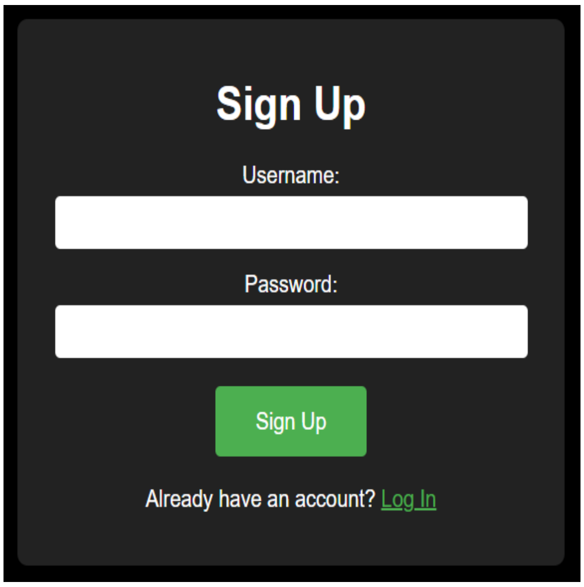
Figure 5.1: Home Page

5.2 Login and Signup

The login and signup pages are designed to facilitate secure user authentication. The login page enables existing users to access the system, while the signup page allows new users to create an account. Both pages ensure a smooth user experience with clear prompts and straightforward forms.



(a) Login Page



(b) SignUpt Page

5.3 Upload Section

The upload section provides a user-friendly interface for submitting handwritten documents. Users can upload images in various formats, and the system processes these images for text detection and recognition.

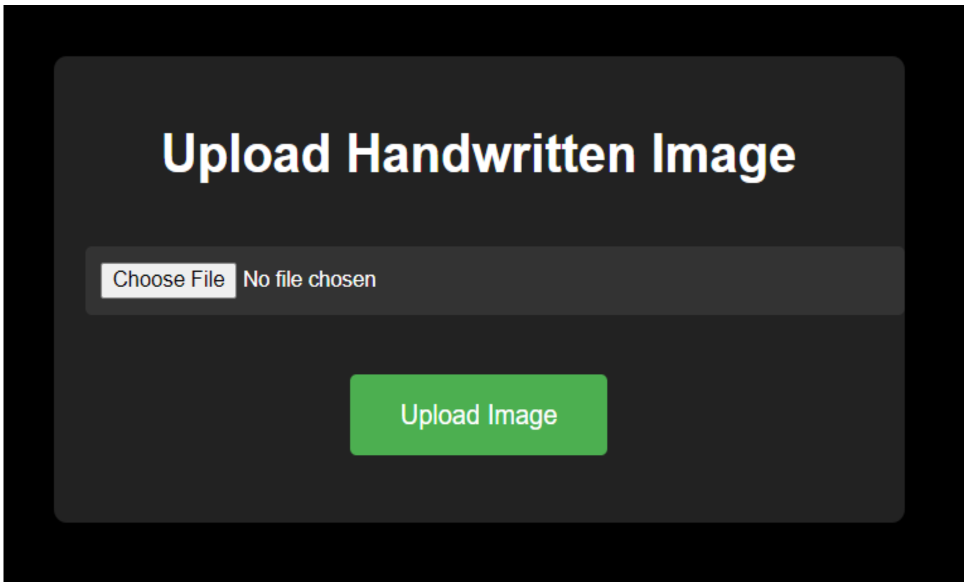


Figure 5.3: Upload Page

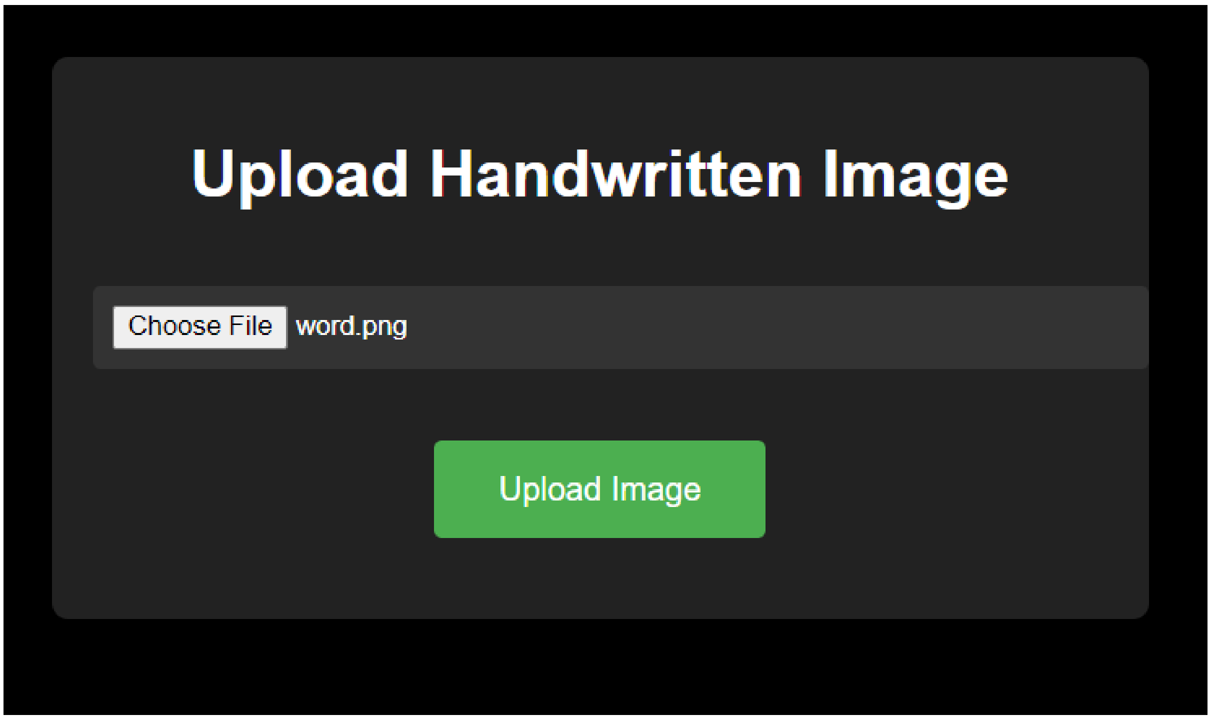
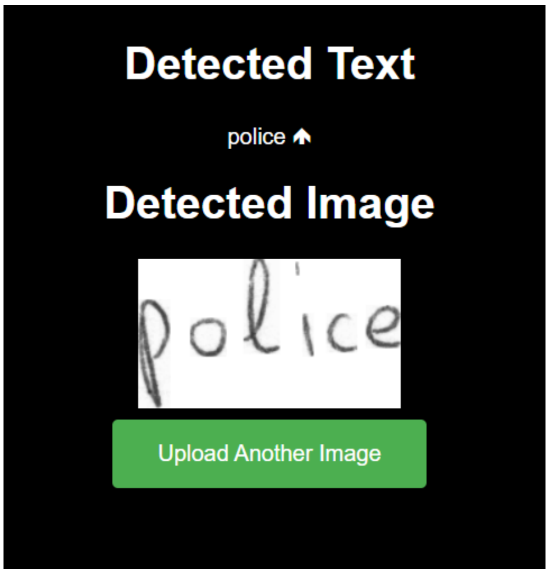


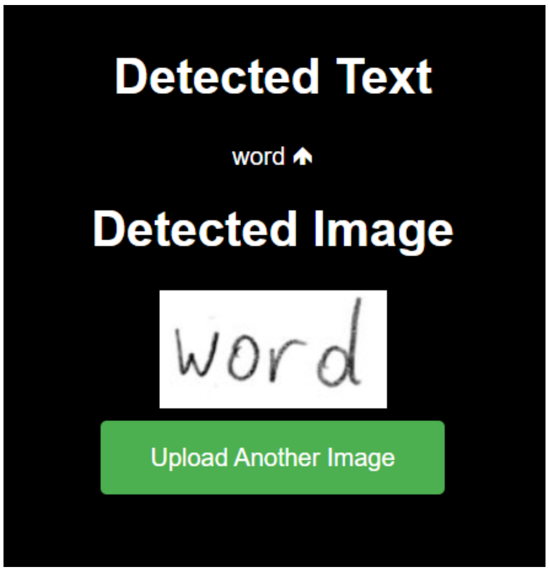
Figure 5.4: Upload Page

5.4 Result Section

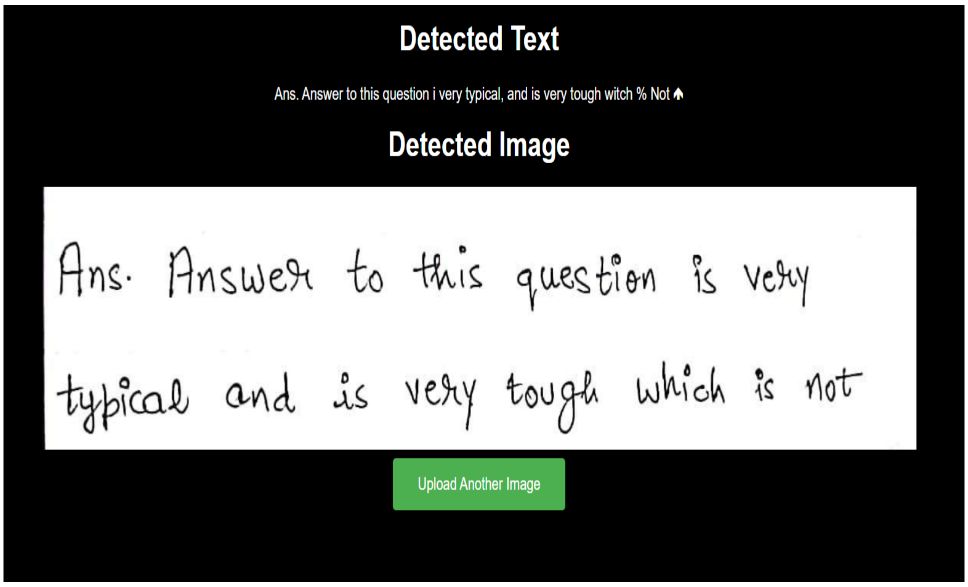
The result section displays the extracted text from the uploaded handwritten documents. It presents the recognized text in a clear format, allowing users to review and utilize the transcribed content.



(a) Result 1



(b) Result 2



Detected Text

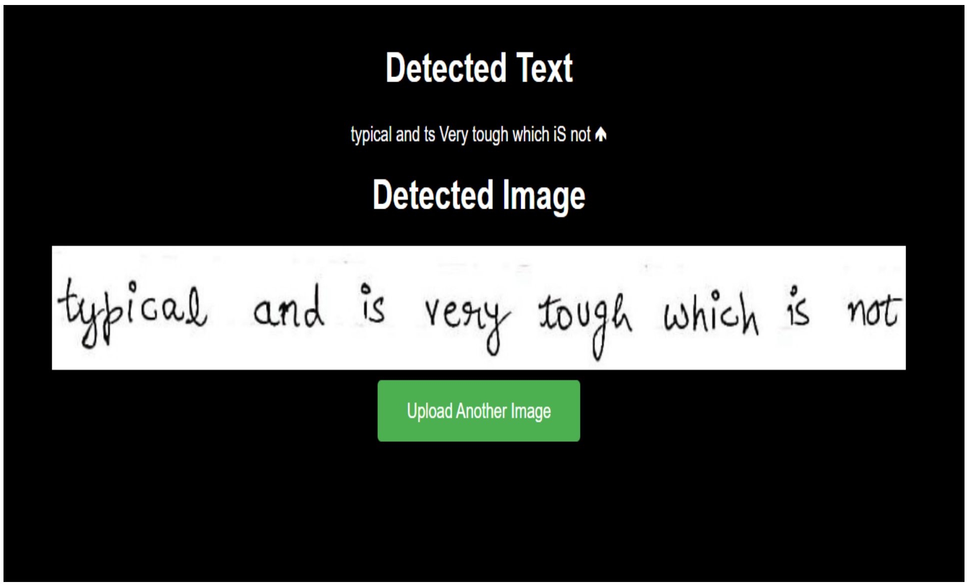
Ans. Answer to this question i very typical, and is very tough witch % Not

Detected Image

Ans. Answer to this question is very typical and is very tough which is not

Upload Another Image

Figure 5.6: Result 3



Detected Text

typical and ts Very tough which iS not

Detected Image

typical and is very tough which is not

Upload Another Image

Figure 5.7: Result 4

Chapter 6

Conclusion and Future Work

This project successfully developed a system for converting handwritten words into digital text using advanced machine-learning techniques. By integrating YOLO for text detection and Tesseract OCR for text recognition, the system effectively processes and digitizes handwritten documents. The user-friendly web application facilitates image uploads, detects text, and easily displays results. The use of Flask as the backend framework ensures seamless communication between the front end and the machine learning components, resulting in a highly efficient and reliable application. . Our project uses the YOLOv8 Handwritten Text Detection model to identify text regions in images, which is provided by armvectores on Hugging Face [**1**].

The system is implemented as a user-friendly web application, providing a seamless experience for users to upload images of handwritten documents. The web application not only facilitates easy image uploads but also displays the recognized text in a clear and accessible format. The system's ability to handle diverse handwriting styles and its user-friendly interface makes it a valuable tool for digitizing handwritten documents. Future improvements could focus on enhancing the accuracy of text recognition.

While our project meets its objectives, future work could focus on improving the accuracy of text recognition by incorporating additional machine-learning models or techniques. Evaluation metrics and markers should be developed to assess the performance of the text recognition system comprehensively. Additionally, expanding the system to handle more diverse handwriting styles and integrating real-time processing capabilities could significantly enhance its applicability and robustness. These improvements will help in refining the system's accuracy and efficiency, making it more valuable for broader applications in educational institutions.

Chapter 7

References

1. [**yolov8˙handwritten˙text˙detection**]. Our project uses the YOLOv8 Handwritten Text Detection model to identify text regions in images, which is provided by armvectores on Hugging Face.
2. [**flask˙documentation**]. Our web application backend was developed using Flask, a lightweight and flexible web framework for Python
3. [**tesseract˙website**]. Tesseract OCR was used for text recognition in our project.
4. [**huggingface.co/models**]. Our project uses this for pre-trained models.