Image Text and Visual Element Extraction Program

Introduction

This report documents the development of a Python program that extracts text and visual elements from an image, and then generates an HTML file with the extracted content. The program leverages Optical Character Recognition (OCR) and image processing techniques to achieve its objectives.

Approach

The main approach of the program is divided into three primary tasks:

1. **Text Extraction**: Using OCR to extract text from the image.
2. **Visual Element Segmentation**: Using image processing techniques to identify and extract visual elements from the image.
3. **HTML Generation**: Creating an HTML file that embeds the extracted text and visual elements.

Technologies Used

The program utilizes the following technologies and libraries:

- **Python**: The programming language used to develop the program.
- **Tesseract OCR**: An open-source OCR engine used to extract text from images.
- **Pillow (PIL)**: A Python Imaging Library (PIL) fork used for opening and manipulating images.
- **OpenCV**: An open-source computer vision and machine learning software library used for image processing tasks.
- **Base64**: A module used to encode binary data (visual elements) as ASCII strings for embedding in HTML.

Implementation Details

Required Libraries

Before running the program, the following libraries need to be installed:

- **Tesseract OCR**: Download and install Tesseract OCR from https://github.com/tesseract-ocr/tesseract.
- **OpenCV**: Install using pip:

```
pip install opencv-python
```

- **Pillow**: Install using pip:

```
pip install pillow
```

- **Pytesseract**: Install using pip:

```
pip install pytesseract
```

Text Extraction

The text extraction functionality is implemented using the Tesseract OCR engine. The following steps are taken to extract text from an image:

1. **Open the Image**: The image is opened using the Pillow library.
2. **Extract Text**: The text is extracted from the image using Tesseract OCR.

```
def extract_text(image_path):
    image = Image.open(image_path)
    text = pytesseract.image_to_string(image)
    return text
```

Visual Element Segmentation

The visual element segmentation is implemented using OpenCV. The following steps are taken to segment visual elements from an image:

1. **Read the Image**: The image is read using OpenCV.
2. **Convert to Grayscale**: The image is converted to grayscale.
3. **Thresholding**: Apply binary thresholding to segment the image.
4. **Find Contours**: Contours of the visual elements are identified.
5. **Extract Visual Elements**: Each visual element is extracted and encoded as a Base64 string.

```
def segment_visual_elements(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    visual_elements = []
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)
        visual_element = image[y:y+h, x:x+w]
        _, buffer = cv2.imencode('.png', visual_element)
        visual_element_base64 = base64.b64encode(buffer).decode('utf-8')
        visual_elements.append(visual_element_base64)

    return visual_elements
```

HTML Generation

The HTML generation functionality creates an HTML file that includes the extracted text and visual elements. The text is formatted with HTML `<br>` tags, and the visual elements are embedded as Base64-encoded images.

```
def generate_html(text, visual_elements):
    html_content = "<html>\n<head>\n<title>Extracted
Content</title>\n</head>\n<body>\n"
    formatted_text = text.replace('\n', '<br>')
    html_content += f"<p>{formatted_text}</p>\n"

    for element in visual_elements:
        html_content += f'<img src="{element}" alt="Visual
Element">\n'

    html_content += "</body>\n</html>"
    return html_content
```

Main Execution

The main execution block orchestrates the entire process by calling the functions defined above and saving the generated HTML content to a file.

```
if __name__ == "__main__":
    image_path = "IvV2y.png"  # Replace with your image path

    text = extract_text(image_path)
    print("Extracted Text:")
    print(text)

    visual_elements = segment_visual_elements(image_path)

    html_content = generate_html(text, visual_elements)

    html_file_path = "output.html"
    with open(html_file_path, "w", encoding="utf-8") as html_file:
        html_file.write(html_content)

    print(f"HTML content saved as: {html_file_path}")
```

Running the Program

1. Ensure your input image is in the same directory as your script or provide the correct path to the image.
2. Run the script in Visual Studio Code's terminal:

```
python main.py
```

3. After running the script, an HTML file named `output.html` will be generated in the same directory. You can open this file in a web browser to see the extracted text and embedded visual elements.

Challenges Faced

Accuracy of OCR

The primary challenge encountered was the accuracy of the OCR results. Tesseract OCR, while being an excellent open-source tool, sometimes provides inaccurate results, especially with complex or low-quality images. For better accuracy and reliability, paid APIs such as Google Vision API, Microsoft Azure Computer Vision, or Amazon Textract could be used. However, these APIs involve a billing process and associated costs.

Image Segmentation

Another challenge was accurately segmenting visual elements in various types of images. The current implementation uses basic thresholding and contour detection techniques, which may not work well for all image types and complexities. More advanced image processing techniques or machine learning models could potentially improve segmentation accuracy.

Conclusion

The developed program successfully extracts text and visual elements from an image and generates an HTML file with the extracted content. While the current implementation demonstrates the core functionality, improvements can be made in terms of OCR accuracy and visual element segmentation by using advanced techniques and paid APIs. The project showcases a practical application of OCR and image processing using Python and open-source libraries.

Code:

```
import cv2
import pytesseract
from PIL import Image
import base64
import io

pytesseract.pytesseract.tesseract_cmd =
r'C:\Users\sanja\AppData\Local\Programs\Tesseract-OCR\tesseract.exe'

def extract_text(image_path):
    # Open the image using PIL
    image = Image.open(image_path)
```

```
    # Use pytesseract to extract text
    text = pytesseract.image_to_string(image)

    return text

def segment_visual_elements(image_path):
    # Read the image using OpenCV
    image = cv2.imread(image_path)

    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply thresholding to segment the image
    _, thresh = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # Find contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    visual_elements = []
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)
        visual_element = image[y:y+h, x:x+w]

        # Convert the visual element to a Base64 string
        _, buffer = cv2.imencode('.png', visual_element)
        visual_element_base64 = base64.b64encode(buffer).decode('utf-8')
        visual_elements.append(visual_element_base64)

    return visual_elements

def generate_html(text, visual_elements):
    html_content = "<html>\n<head>\n<title>Extracted
Content</title>\n</head>\n<body>\n"

    # Replace new lines with <br> tags for HTML
    formatted_text = text.replace('\n', '<br>')

    # Add text to HTML content
    html_content += f"<p>{formatted_text}</p>\n"

    # Add visual elements to HTML content
    for element in visual_elements:
        html_content += f'<img src="{element}" alt="Visual
Element">\n'

    html_content += "</body>\n</html>"

    return html_content

if __name__ == "__main__":
    image_path = "IvV2y.png"  # Replace with your image path

    # Extract text from image
    text = extract_text(image_path)
```

```
    print("Extracted Text:")
    print(text)

    # Segment visual elements in image
    visual_elements = segment_visual_elements(image_path)

    # Generate HTML content
    html_content = generate_html(text, visual_elements)

    # Save the HTML content to a file
    html_file_path = "output.html"
    with open(html_file_path, "w", encoding="utf-8") as html_file:
        html_file.write(html_content)

    print(f"HTML content saved as: {html_file_path}")
```

To Contact me:

Ch. Sanjan Shiv

+91 – 8247657647

sanjanshiva123@gmail.com