Technical Project Report: Separating Text and Visual Elements from an Image

Abstract

This report details the development of a program designed to separate text and visual elements from images using AWS Rekognition and image segmentation techniques. The program leverages Optical Character Recognition (OCR) for text extraction and basic image segmentation to isolate visual elements. This report covers the approach, chosen technologies, implementation details, and challenges encountered during development.

Table of Contents

1. Introduction

The goal of this project is to develop a program capable of analyzing an image to separate and extract text and visual elements. This functionality can be applied in various domains, such as document digitization, automated content extraction, and image-based information retrieval.

2. Objectives

- Image Analysis: Utilize AWS Rekognition to analyze the uploaded image and identify labels.

- Text Extraction: Implement OCR using AWS Rekognition to extract all text content from the image.

- Visual Element Segmentation: Apply basic image segmentation techniques to isolate individual visual elements.

- Basic HTML Structure: Develop code to output basic html code incorporating generated labels, extracted text and isolated images

3. Technology Stack

- Programming Language: Python

- Vision API: AWS Rekognition

- Libraries: boto3, Pillow, BeautifulSoup

- Development Environment: Visual Studio Code

- Version Control: Git, GitHub

4. Implementation

4.1 Image Analysis

The first step involves analyzing the image using AWS Rekognition. This API provides powerful image analysis capabilities, including object detection, text detection, and more.

Defined a main ImageAnalysis class consisting of methods detect_labels() which analyses the image and identifies labels and detect_text() which extracts text using OCR capabilities of AWS Rekognition in file “image_analysis.py”.

-> Image_analysis.py

```
from pprint import pprint
import logging
import io
from botocore.exceptions import ClientError
from PIL import Image, ImageDraw
from bs4 import BeautifulSoup

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

class ImageAnalysis:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client



    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.
```

```
        :param image_file_name: The file name of the image. The file is opened
and its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
        :return: The RekognitionImage object, initialized with image bytes from
the
                 file.
        """
        with open(image_file_name, "rb") as img_file:
            image = {"Bytes": img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)


    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.

        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels
            )
            labels = [ImageLabel(label) for label in response["Labels"]]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels

    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
```

```
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [ImageText(text) for text in response["TextDetections"]]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

Steps:

1. First we will define analyze_image function to identify labels using AWS Rekognition and show bounding boxes around them.

-> **image_analysis.py**

```
def analyze_image(analyzer):
    """
    Analyze image and highlight the elements and list the analysis labels.
    This function returns a list containing bounding box coordinates and a list
containing labels.
    """

    labels = analyzer.detect_labels(100)
    print(f"\nAnalyzed {len(labels)} labels.")
    for label in labels:
        pprint(label.to_dict())
    names = []
    box_sets = []
    colors = ["aqua", "red", "white", "blue", "yellow", "green", "yellow",
"lightgreen"]

    for label in labels:
        if label.instances:
            names.append(label.name)
            box_sets.append([inst["BoundingBox"] for inst in label.instances])

    print(f"\n\nShowing bounding boxes for {names}.")
    show_bounding_boxes(
        analyzer.image["Bytes"], box_sets, colors[: len(names)]
    )
```

```
    return box_sets, labels
```

2. Now we will use the defined funtion in main.py

- After setting proper configuration and instantiating a Rekognition client, we will define an analyzer object of ImageAnalysis class. Then analyze the image using the following lines of code.

-> **main.py**

```
# creating analyzer object of the ImageAnalysis Class
analyzer = ImageAnalysis.from_file(
    IMG, rekognition_client
)

# analyze labels identified in the image.
box_sets, labels = analyze_image(analyzer)
```

4.2 Text Extraction

For text extraction, the program utilizes the OCR capabilities of AWS Rekognition and writes it into the file called “extracted_text.txt”.

1. First we will define extract_text function to extract text from image using OCR in AWS Rekognition in the image_analysis.py

-> **image_analysis.py**

```
def extract_text(analyzer):
    """
    Extract text content from the given image using Amazon Rekognition. This
function will output extracted text content in a txt file named
"extracted_text.txt and returns the extracted text content."
```

```
    """

    texts = analyzer.detect_text()

    print(f"\nFound {len(texts)} text instances. Here are the texts:")

    text_content = []

    with open("extracted_text.txt", "w+", encoding='utf-8') as f:
        for text in texts:
            content = text.to_dict()['text']
            text_content.append(content)
            pprint(content)
            content += "\n"
            f.write(content)

    show_polygons(
        analyzer.image["Bytes"], [text.geometry["Polygon"] for text in texts],
"aqua"
    )

    return text_content
```

2. Further add the following lines in main.py to extract text from the image using analyzer defined above and save it in extracted_text.txt file. Extracted text will be assigned to the text_content variable.

```
 # outputs final image as final_analyzed_image.jpg
text_content = extract_text(analyzer)
```

4.3 Visual Element Segmentation

Basic image segmentation techniques are used to isolate visual elements. OpenCV provides functions for this purpose.

Steps:

1. First define isolate_features function in image_analysis.py to crop and save the visual elements shown by analyze_image function using bounding boxes in isolated or seperate images.

-> **image_analysis.py**

```
def isolate_features(image_bytes, box_sets):
    """
    Isolate the analyzed elements and save the isolated elements or image named
like element0.jpg, element1.jpg, element2.jpg, ...
    """

    image = Image.open(io.BytesIO(image_bytes))
    im = []
    modified_box_set = []

    for box in box_sets:
        if box not in modified_box_set:
            modified_box_set.append(box)

    for boxes in modified_box_set:
        for box in boxes:
            left = image.width * box["Left"]
            top = image.height * box["Top"]
            right = (image.width * box["Width"]) + left
            bottom = (image.height * box["Height"]) + top
            im.append(image.crop((left, top, right, bottom)))
    count = 0
    for i in im:
        i = i.convert('RGB')
        i.save(f'element{count}.jpg')
        count += 1
        i.show()

    return count
```

2. Now use the isolate_features function to isolate and segment visual elements shown below.

```
# isolates image elements into seperate images
feauture_count = isolate_features(analyzer.image["Bytes"], box_sets)
```

4.4. Basic HTML Structure

We will write a code to output a basic html code incorporating <p> tags for identified labels and extracted text, and <img> tag for the isolated and segmented images.

Steps:

1. Define a function createHTML() in the image_analysis.py file to generate <p> tag and <img> tag with necessary text and src for labels, extracted text, isolated images respectively.

-> **image_analysis.py**

```
def createHTML(labels, text_content, count):
    """
    creates a basic html structure with labels, text data and isolated images and
outputs a file "index.html".
    """

    html_template = """<html>
    <head></head>
    <body>
    </body>
    </html>
    """

    soup = BeautifulSoup(html_template, 'html.parser')

    text_label = soup.new_tag("h3")
    text_label.string = "Analyzed following labels from the image: "
    soup.body.append(text_label)

    for label in labels:
        tag = soup.new_tag("p")
        tag.string = label.to_dict()['name']
        soup.body.append(tag)

    text_label = soup.new_tag("h3")
    text_label.string = "Text Content Extracted from the image: "
```

```
    soup.body.append(text_label)

    for text in text_content:
        tag = soup.new_tag("p")
        tag.string = text
        soup.body.append(tag)

    for i in range(count):
        img_tag = soup.new_tag("img")
        img_tag['src'] = f'./element{i}.jpg'
        soup.body.append(img_tag)

    html_content = soup.contents

    with open('index.html', 'w') as html_file:
        for item in html_content:
            html_file.write(str(item))

    return html_content
```

2. Now call the createHTML() function in main.py with the necessary arguments derived in the image analysis, text extraction and isolate elements code.

-> **main.py**

```
# creates a basic html structure with identified labels, extracted text content, isolated visual elements
createHTML(labels, text_content, feauture_count)
```

5. Testing and Validation

Testing involves validating the text extraction accuracy and the effectiveness of visual element segmentation.

Text Extraction Testing:

- Compare extracted text with the expected text from sample images.

- Evaluate accuracy using metrics such as precision and recall.

Visual Segmentation Testing:

- Visual inspection of segmented images to ensure individual elements are correctly isolated.

6. Challenges and Solutions

- OCR Accuracy: Variations in font size, style, and image quality can affect OCR accuracy. Preprocessing techniques such as noise reduction and binarization help improve results.

- Segmentation Precision: Complex backgrounds and overlapping elements pose challenges. Using advanced segmentation algorithms like GrabCut can enhance precision.

7. Conclusion

The developed program successfully separates text and visual elements from images using OCR and image segmentation techniques. AWS Rekognition provides robust capabilities for text extraction, while OpenCV facilitates effective visual element segmentation.

8. References

- “AWS Rekognition Documentation,” [AWS Rekognition](https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html).