# Image Text Extraction

**APP:** (imagetextextraction.streamlit.app)

**Approach:** The primary goal was to develop an application capable of analysing an uploaded image, performing image segmentation, and extracting text. The application provides a structured HTML file with extracted text and images of the detected visual elements.

## Chosen Technologies:

- **Streamlit:** Used to build the web application interface.
- **Google Cloud Vision API:** Utilized for accurate text detection.
- **Ultralytics YOLOv8:** Employed for object detection and segmentation.
- **OpenCV:** Utilized for image processing tasks.
- **Base64:** Used for embedding images in the HTML report.

## Implementation Details:

### Detect Text:

- Utilises the Google Cloud Vision API to detect text in the provided image.
- Reads the image file, sends it to the API, and processes the response to extract text and bounding boxes.

### Image Segmentation:

- It uses the YOLOv8 model to detect objects and segment the image.
- Reads the image file, performs predictions with the model, and saves the annotated image.

### Draw Annotations:

- Reads the image using OpenCV.
- Draws bounding boxes around the detected text using coordinates from the API response.
- Returns the annotated image.

### Save Segmented Objects:

- Iterates over segmented objects detected by the YOLO model.
- Creates masks for each object, applies the masks to the original image, and saves the segmented objects as separate images.
- Converts segmented images to base64 for embedding in HTML.

### Generate HTML:

- Generates HTML content with embedded images and extracted text.
- Constructs HTML with base64-encoded images and text paragraphs.

## Challenges Encountered:

### Writing HTML File:

- Initially, embedding images directly into the HTML file was problematic. This was resolved by using base64 encoding to embed images as data URLs.

**Colour Inversion:**

- OpenCV uses BGR colour format, whereas other libraries use RGB.

**Reading Image Files:**

- Handling various image formats and ensuring proper processing was challenging, especially with OpenCV and file encoding.

**Encoding Issues:**

- Ensuring proper encoding while writing the HTML file was necessary to handle all characters and prevent errors.

**Image Segmentation:**

- Accurately segmenting images, particularly those with fewer details, took time.

**Deploying on Streamlit Cloud:**

- During deployment, issues were encountered regarding saving image paths and accessing Google Cloud Vision API credentials.

## Results and Limitations:

**Annotated Image:**

- The resulting annotated image has a blue-coloured box drawn around the detected text, and each pixel is labelled with objects identified by the YOLO model.
- A download file to download html that sort the extracted content into appropriate tags

**Object Detection Limitation:**

- The YOLOv8 model used in this implementation can detect up to 80 class objects, which is a limitation as it may not detect objects outside these predefined classes.