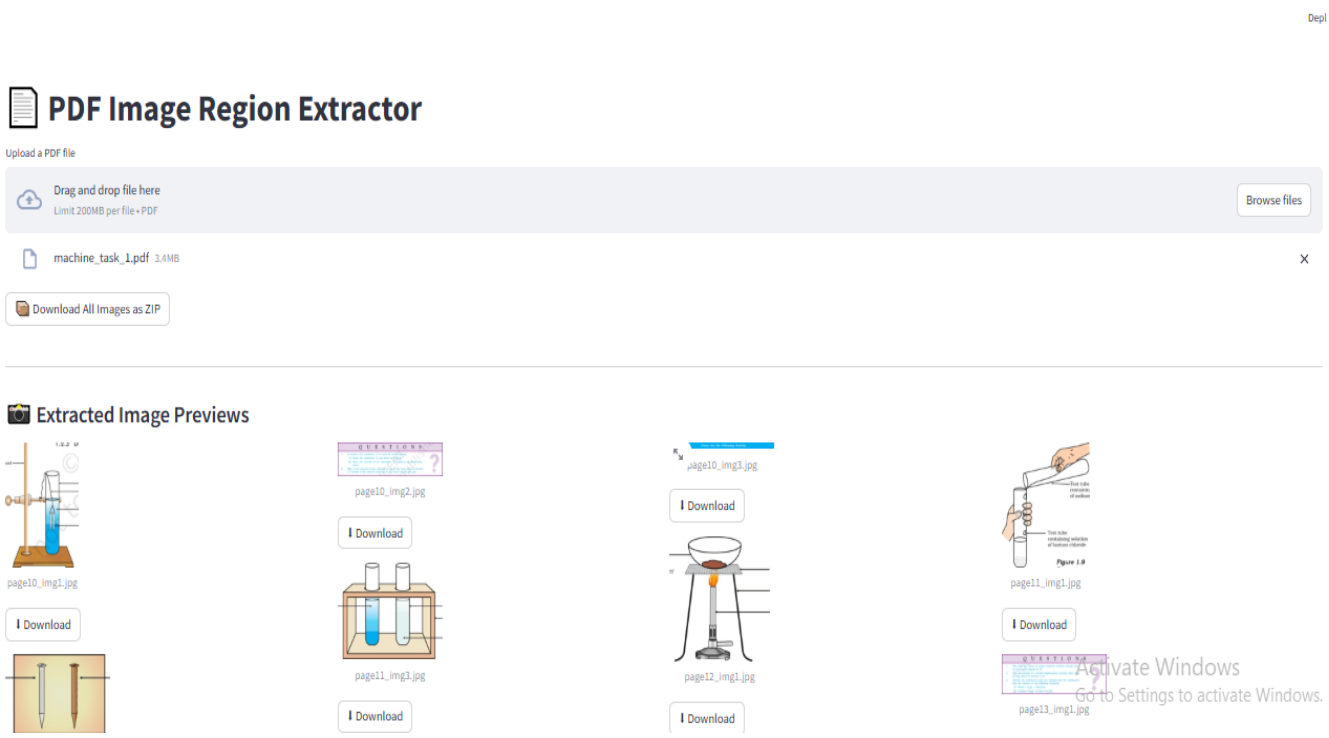
PDF-Image-Exctractor – Documentation

Machine task 1

Github link : https://github.com/abhiramkrishnayanam/PDF-Image-Exctractor



1. Project Overview

The **PDF Image Region Extractor** is a web application built using **Streamlit, PyMuPDF (fitz) , OpenCV (cv2) and Pillow (PIL)** libraries that allows users to upload a PDF file, automatically detect and extract image-like regions from each page, and download the results as a ZIP file. This tool is particularly useful for extracting embedded figures, scanned regions, or illustrations from PDF documents such as research papers, scanned books, and design files.

2. Tools and Libraries Used

Python Libraries:

- **Streamlit** – For building the interactive web application.
- **PyMuPDF (fitz)** – For reading PDF files and rendering pages as images.
- **OpenCV (cv2)** – For image processing and contour detection.
- **NumPy** – For efficient manipulation of image data.
- **Pillow (PIL)** – For handling and displaying image thumbnails in the web app.
- **Zipfile** – To bundle extracted images into a downloadable ZIP file

3. Step-by-Step Approach

Frontend (streamlit_app.py):

1. **Set Page Configuration:**
   Sets the title and layout of the Streamlit web app.
2. **File Upload:**
   Users upload a PDF file using the Streamlit file uploader.
3. **Temporary Directory Creation:**
   A temporary directory is created to store the uploaded PDF and the extracted images.
4. **Image Extraction Function Call:**
   The `extract_image_blocks()` function is called with the PDF path and output folder as arguments.
5. **ZIP File Creation:**
   All extracted images are added to a ZIP archive.
6. **ZIP Download Button:**
   Users can download all extracted images as a ZIP file.
7. **Image Previews and Individual Downloads:**
   Thumbnail previews of each extracted image are displayed with individual download buttons.

4. Backend (pdf_image_extractor.py) – Detailed Explanation

1. PDF Loading

Library Used: `PyMuPDF (fitz)`

```
pdf_file = fitz.open(pdf_path)
```

- The PDF is loaded using `fitz.open()`, which creates a `Document` object.
- For each page, `page.get_pixmap(dpi=300)` renders the page as an image at 300 DPI, which improves quality and makes small image elements more distinguishable.

```
pix = pdf_file[page_number].get_pixmap(dpi=300)
```

- `pix.samples` gives raw pixel bytes.
- `pix.n` tells the number of color channels (3 for RGB, 4 for RGBA).

2. Image Conversion

Libraries Used: `NumPy`, `OpenCV`

```
img_data = pix.samples
img_size = (pix.height, pix.width, pix.n)
img_np = np.frombuffer(img_data, dtype=np.uint8).reshape(img_size)
```

- Raw image data from PyMuPDF is converted into a NumPy array to allow manipulation using OpenCV.
- This transformation is essential to apply image processing techniques.

```
if pix.n == 4:
    img_cv = cv2.cvtColor(img_np, cv2.COLOR_RGBA2BGR)
else:
    img_cv = cv2.cvtColor(img_np, cv2.COLOR_RGB2BGR)
```

- Depending on whether the image has an alpha (transparency) channel (`RGBA`), we use `cv2.cvtColor` to convert it into a format OpenCV can work with (`BGR`).

3. Grayscale Conversion & Thresholding

Library Used: `OpenCV`

```
gray = cv2.cvtColor(img_cv, cv2.COLOR_BGR2GRAY)
```

- Converts the color image to grayscale.
- Grayscale simplifies the image and reduces computational complexity.

```
_, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY_INV)
```

- Applies binary inverse thresholding:
  - Pixels brighter than 200 become black (`0`),
  - Pixels darker than 200 become white (`255`).
- This highlights potential image or region areas while suppressing the white background.

4. Contour Detection

Library Used: `OpenCV`

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

- Detects the boundaries of all external connected components (white blobs from thresholding).

- `RETR_EXTERNAL` ensures that only outermost contours are found (i.e., no nested ones).
- Each contour is a list of boundary coordinates.

5. Region Filtering

Library Used: `OpenCV`, `OS`

```
x, y, w, h = cv2.boundingRect(cnt)
if w > 50 and h > 50:
```

- A bounding rectangle is drawn around each detected contour.
- Filters out noise and very small regions (like text characters or dots) by ignoring those with width/height less than 50 pixels.

6. Image Saving

Libraries Used: `OpenCV`, `OS`

```
roi = img_cv[y:y+h, x:x+w]
save_path = os.path.join(output_folder, f"page{page_number+1}_img{img_count+1}.jpg")
cv2.imwrite(save_path, roi)
```

- The Region of Interest (ROI) is extracted (i.e., cropped).
- The cropped image is saved using `cv2.imwrite()` in the specified `output_folder`.
- File names follow the convention `page{page_number}_img{image_number}.jpg` for easy tracking.

Challenges Faced & Resolutions

Challenges

- **Non-Embedded Images:** Many PDFs had images as scanned or rasterized regions, not proper embedded images, making extraction difficult.
- **Mixed Content:** Extracted regions sometimes included parts of text or page artifacts along with images.
- **Inconsistent Results:** Variations in PDF quality and layout caused inconsistent image extraction and cropping accuracy.

Resolutions (Future Improvements)

- Use **adaptive thresholding** and **morphological operations** to better isolate images.
- Implement **text detection and masking** (e.g., OCR-based) to remove text from extracted regions.
- Explore **deep learning segmentation models** for precise image extraction.
- Add **manual region selection** in the Streamlit UI for user-controlled cropping.

Conclusions

This project successfully demonstrates a practical approach to extracting image regions from PDFs using a combination of PyMuPDF and OpenCV. By rendering PDF pages as images and applying image processing techniques, the tool can isolate and save multiple image blocks from complex documents.

While the current method works well for clearly defined images, challenges remain when dealing with scanned PDFs or documents where images and text overlap. Future improvements such as adaptive preprocessing, text masking, and advanced segmentation can enhance accuracy and usability.

Overall, the project provides a solid foundation for PDF image extraction with a user-friendly Streamlit interface, enabling users to easily upload PDFs, preview extracted images, and download them conveniently.