1. Purpose

The project aims to process images of handwritten or printed documents by enhancing text visibility and removing uneven backgrounds. This will address issues like uneven lighting and camera distortions, producing clean, readable outputs suitable for digitization and analysis.

---

2. Main Functionalities

2.1. Preprocessing

Key preprocessing steps include:

1. **Noise Reduction**: Smooths images to remove artifacts (e.g., Gaussian Blur).
2. **Contrast Adjustment**: Enhances faint text visibility.
3. **Binarization**: Separates text from the background using thresholding methods.
4. **Edge Preservation**: Retains fine text details with Bilateral Filtering.
5. **Morphological Filtering**: Connects fragmented text and reduces noise.
6. **Gradient Analysis**: Highlights text transitions for better segmentation.

2.2. Post-Processing

1. **Perspective Correction**: Corrects skew caused by camera angles by detecting paper edges and applying a perspective transformation.
2. **Background Removal**: Removes uneven backgrounds while preserving text using thresholding and morphological operations.
3. **Text Detection and Highlighting**: Identifies text regions through contour-based methods, with filtering for geometric constraints. Text is highlighted using bounding boxes.
4. **Edge Enhancement**: Sharpens faint edges for improved text clarity.

---

3. Implementation Details

- **Languages and Tools**: Python 3.x with OpenCV and NumPy for image processing.
- **Input Requirements**:
  - Images captured via smartphone or scanner.
  - Document fully visible, ideally A4 size, with a contrasting background.
  - Adequate resolution (min 300 DPI for scans or 8 MP for photos).
  - Minimal skew or extreme angles.
- **Output**: Cleaned, enhanced images with text highlighted and backgrounds removed, saved in grayscale or binarized formats.

---

4. Expected Outcome

- Improved text clarity.
- Removal of uneven backgrounds.
- Correct camera-induced skew.

- Aligned, enhanced outputs suitable for digitization and archiving.

---

5. Contributions

1. **Noise Reduction**
   - **Contribution**: Implement custom smoothing using Gaussian filters.
   - **Libraries**: OpenCV's `cv2.GaussianBlur` and `cv2.fastNlMeansDenoising`.
2. **Contrast Adjustment**
   - **Contribution**: Develop scripts for local contrast enhancement using histograms.
   - **Libraries**: NumPy for intensity scaling.
3. **Binarization**
   - **Contribution**: Experiment with adaptive thresholding and Otsu's method for effective separation.
   - **Libraries**: OpenCV's `cv2.adaptiveThreshold`.
4. **Morphological Filtering**
   - **Contribution**: Use dilation/erosion to clean text and remove noise.
   - **Libraries**: OpenCV's `cv2.morphologyEx`.
5. **Gradient Analysis**
   - **Contribution**: Implement Sobel-based gradient analysis for better text segmentation.
   - **Libraries**: OpenCV's `cv2.Sobel`.
6. **Perspective Correction**
   - **Contribution**: Detect document edges and apply transformations to correct skew.
   - **Libraries**: OpenCV's `cv2.findContours` and `cv2.getPerspectiveTransform`.
7. **Background Removal**
   - **Contribution**: Use adaptive thresholding and morphological operations to remove backgrounds.
   - **Libraries**: OpenCV's `cv2.adaptiveThreshold` and `cv2.morphologyEx`.
   - **Possible Improvements**:
     - Add edge detection (`cv2.Canny`) for better separation.
     - Use gradient-based segmentation (`cv2.Sobel`) for faint text.
     - Enhance contrast with local histogram equalization.
8. **Text Detection and Highlighting**
   - **Contribution**: Detect text using contours and filter by size and aspect ratio.
   - **Libraries**: OpenCV's `cv2.findContours` and `cv2.boundingRect`.
   - **Possible Improvements**:
     - Preprocess with edge detection for improved accuracy.
     - Use rotated bounding boxes (`cv2.minAreaRect`) for skewed text.
     - Merge fragmented regions based on proximity.