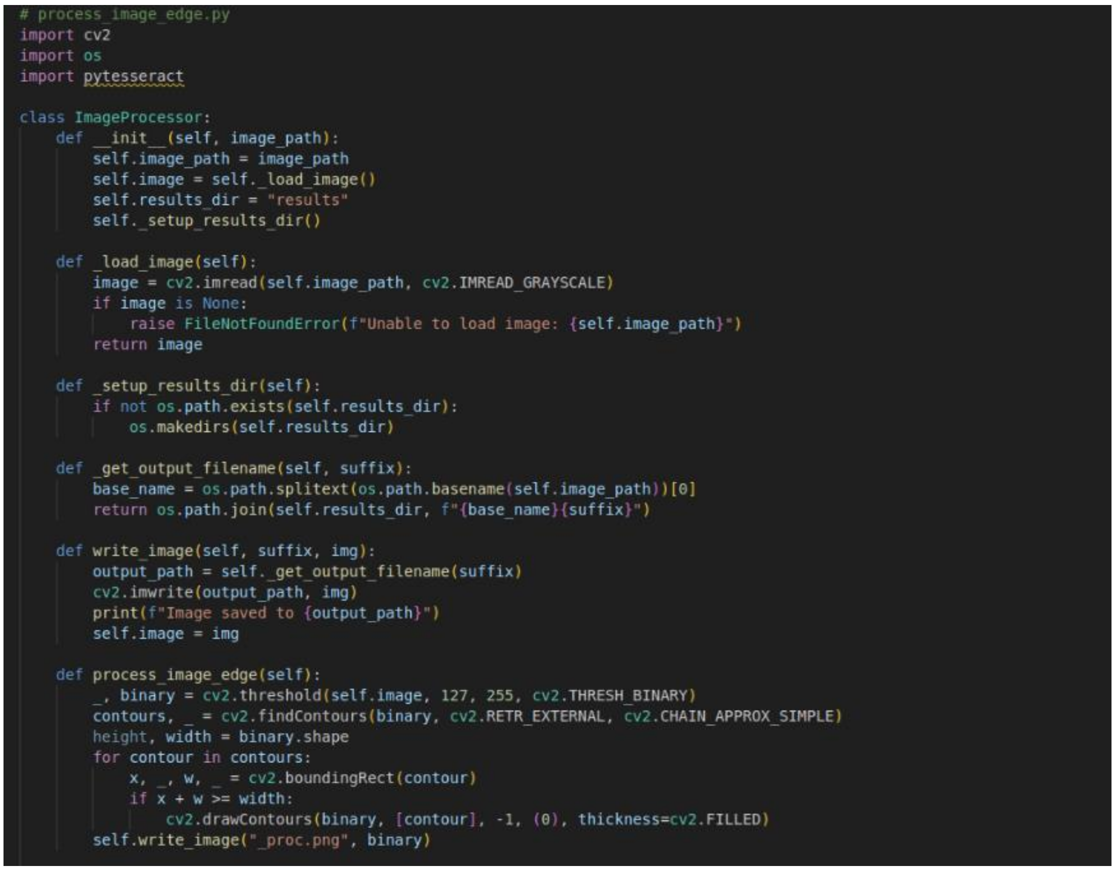
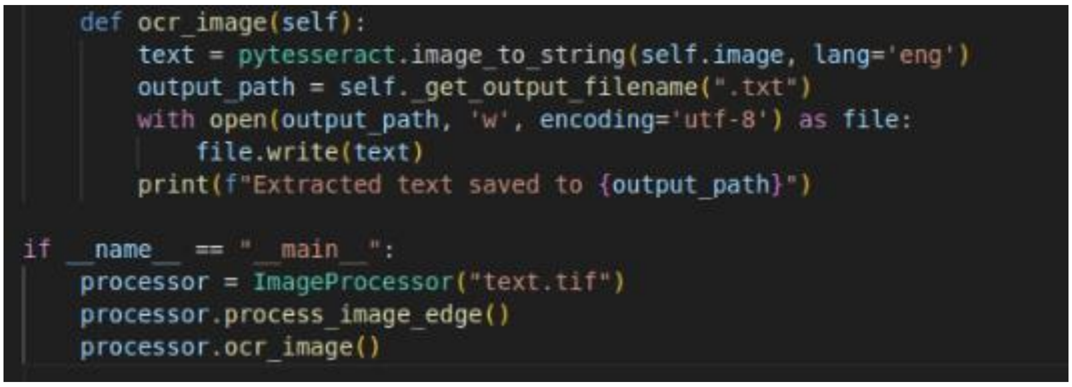
1. text.tif

Code:



```python
# process_image_edge.py
import cv2
import os
import pytesseract

class ImageProcessor:
    def __init__(self, image_path):
        self.image_path = image_path
        self.image = self._load_image()
        self.results_dir = "results"
        self._setup_results_dir()

    def _load_image(self):
        image = cv2.imread(self.image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            raise FileNotFoundError(f"Unable to load image: {self.image_path}")
        return image

    def _setup_results_dir(self):
        if not os.path.exists(self.results_dir):
            os.makedirs(self.results_dir)

    def _get_output_filename(self, suffix):
        base_name = os.path.splitext(os.path.basename(self.image_path))[0]
        return os.path.join(self.results_dir, f"{base_name}{suffix}")

    def write_image(self, suffix, img):
        output_path = self._get_output_filename(suffix)
        cv2.imwrite(output_path, img)
        print(f"Image saved to {output_path}")
        self.image = img

    def process_image_edge(self):
        _, binary = cv2.threshold(self.image, 127, 255, cv2.THRESH_BINARY)
        contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        height, width = binary.shape
        for contour in contours:
            x, _, w, _ = cv2.boundingRect(contour)
            if x + w >= width:
                cv2.drawContours(binary, [contour], -1, (0), thickness=cv2.FILLED)
        self.write_image("_proc.png", binary)
```



```python
    def ocr_image(self):
        text = pytesseract.image_to_string(self.image, lang='eng')
        output_path = self._get_output_filename(".txt")
        with open(output_path, 'w', encoding='utf-8') as file:
            file.write(text)
        print(f"Extracted text saved to {output_path}")

if __name__ == "__main__":
    processor = ImageProcessor("text.tif")
    processor.process_image_edge()
    processor.ocr_image()
```

Approach:

1. Detailed Steps for Image Processing

(1) Image Loading

First, the input image is loaded using OpenCV in grayscale mode. In grayscale mode, the image contains a single channel of pixel intensity values (ranging from 0 to 255), representing the brightness levels from black to white. This simplifies computation and provides a unified format for subsequent processing.

If the image cannot be loaded (e.g., due to incorrect file path or file corruption), an error is raised to notify the user that the image could not be accessed.

(2) Binary Thresholding

After loading, the image undergoes binary thresholding, a process that converts the grayscale image into a binary image with only two distinct pixel values:

- White (255) represents the foreground.
- Black (0) represents the background.

A threshold value of 127 is used, meaning pixels with intensity greater than or equal to 127 are set to white, while others are set to black. This enhances the contrast between the main structures of the image, making it easier to detect contours later.

(3) Contour Detection

Edge detection techniques are applied to identify contours in the image. Contours are curves formed by joining continuous points along the boundaries of an object. This step extracts only the external contours of the image, ignoring any nested internal details (e.g., small holes).

The contour data is returned as a list, with each contour containing multiple coordinate points that define the geometric boundary of the contour.

(4) Removing Contours Intersecting the Border

Contours that intersect with the right border of the image are filtered and removed:

- For each contour, a bounding rectangle (the smallest rectangle enclosing the contour) is calculated.
- The right edge of the bounding rectangle is checked to see if it extends beyond the width of the image. If so, the contour is deemed intersecting with the right border.
- Such contours are filled with black pixels, effectively removing them from the binary image.

This step ensures that distracting elements are cleared from the image, leaving only meaningful regions to enhance the accuracy of subsequent OCR processing.

(5) Saving the Processed Image

The processed image, now cleaned and thresholded, is saved to a specified path with a new filename. This allows users to verify the output or use it for further operations.

2. Detailed Steps for Optical Character Recognition (OCR)

(1) Overview of OCR Technology

Optical Character Recognition (OCR) is a technology that converts text in images into editable text.
This program uses pytesseract, a Python wrapper for the Tesseract OCR engine. It can extract English letters, numbers, and symbols from images and convert them into corresponding text data.

(2) OCR Processing the Image

Once the image has been cleaned, it is processed by the OCR engine:

- The processed image (in grayscale or binary format) is loaded.
- The language is set to English (lang='eng') to focus the engine on recognizing English characters.
- The engine analyzes the pixel distribution patterns in the image, matches them to known text patterns, and extracts recognizable characters.

(3) Saving Extracted Text

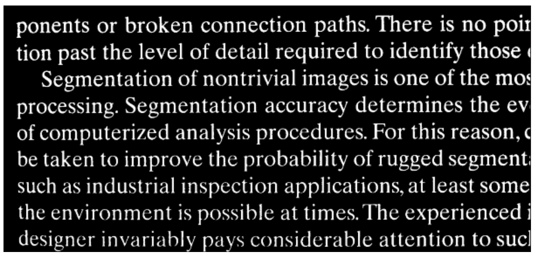
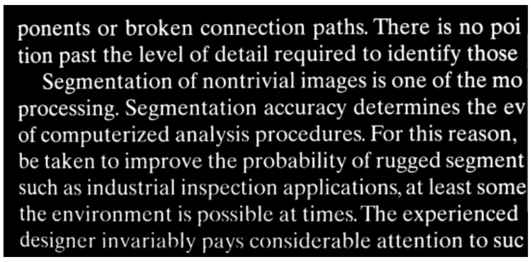
The text extracted by the OCR engine is saved into a .txt file for further use or review. The saved file has the following characteristics:

- All detected text from the image is saved in sequence.
- The file is stored with UTF-8 encoding to ensure cross-platform compatibility.

3. Summary

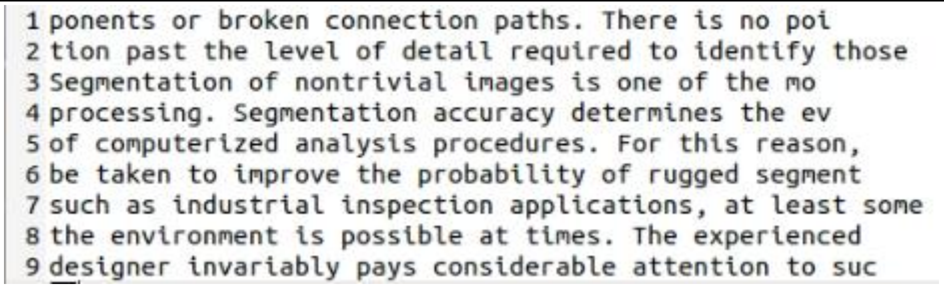
- **Image Cleaning and Optimization:**
  By performing binary thresholding and border contour removal, the program reduces noise and distractions in the image, making text regions easier to detect and extract.
- **Accurate Text Extraction:**
  The OCR tool is used to recognize and extract textual content from the image, converting it into a machine-readable format for digital preservation.

Result:

	
Origin	**Process**

ponents or broken connection paths. There is no poir
tion past the level of detail required to identify those
Segmentation of nontrivial images is one of the mos
processing. Segmentation accuracy determines the ev
of computerized analysis procedures. For this reason, c
be taken to improve the probability of rugged segment
such as industrial inspection applications, at least some
the environment is possible at times. The experienced i
designer invariably pays considerable attention to sucl

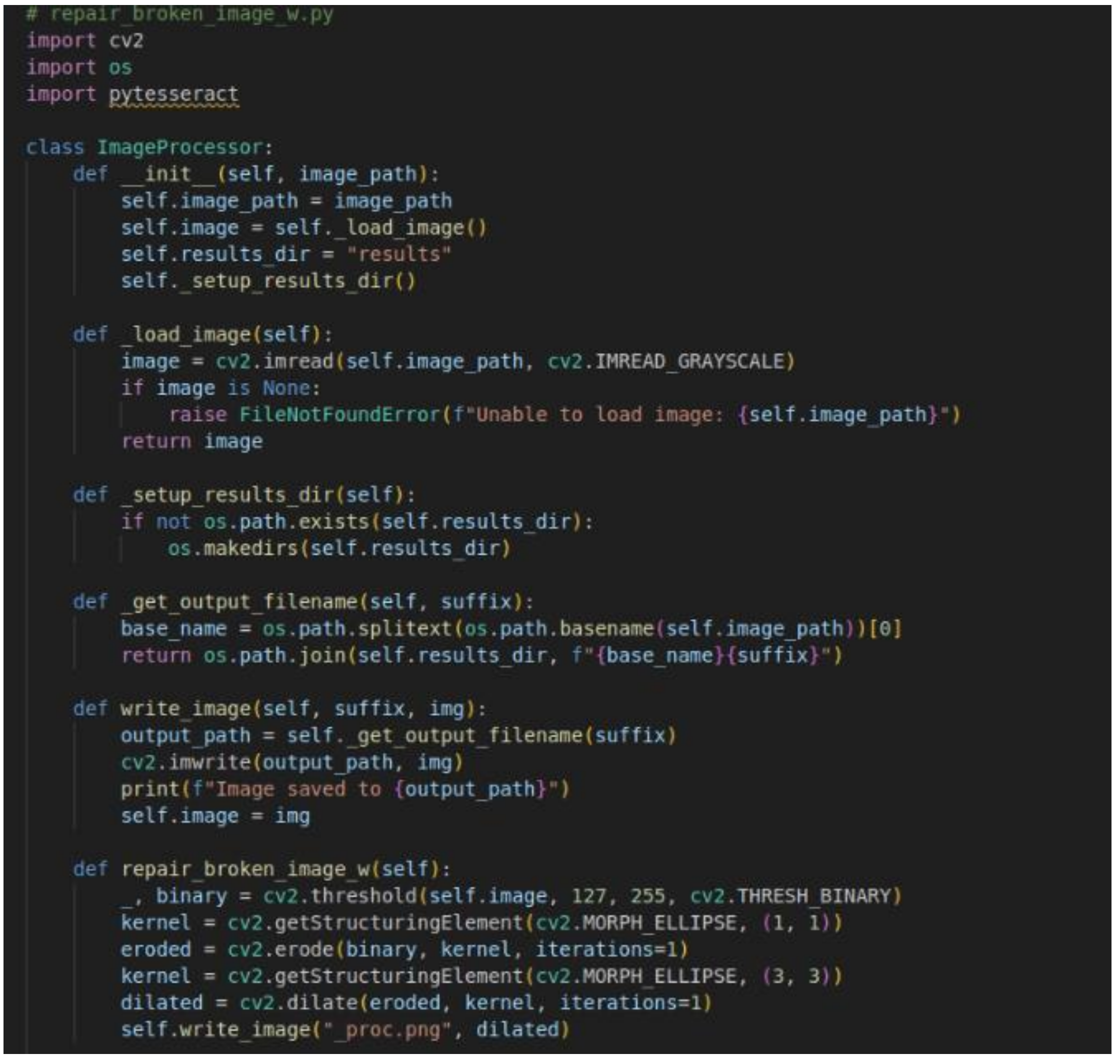
ponents or broken connection paths. There is no poi
tion past the level of detail required to identify those
Segmentation of nontrivial images is one of the mo
processing. Segmentation accuracy determines the ev
of computerized analysis procedures. For this reason,
be taken to improve the probability of rugged segment
such as industrial inspection applications, at least some
the environment is possible at times. The experienced
designer invariably pays considerable attention to suc


Txt file

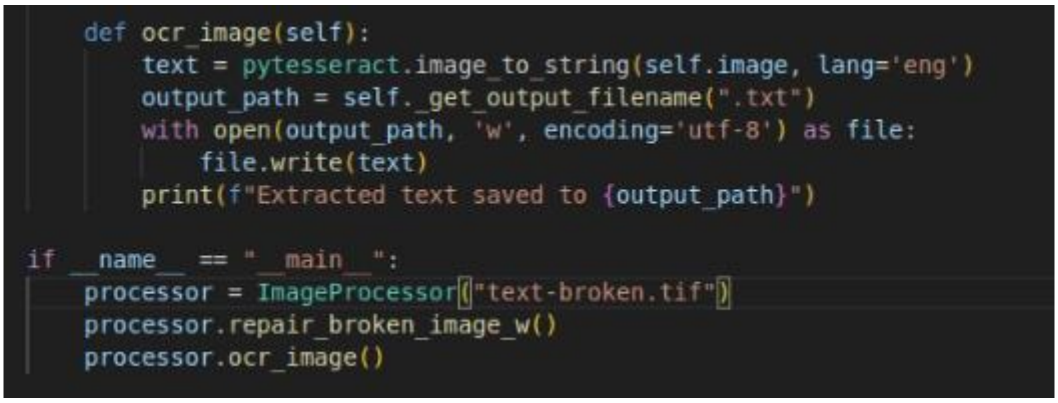
```
ponents or broken connection paths. There is no poi
tion past the level of detail required to identify those
Segmentation of nontrivial images is one of the mo
processing. Segmentation accuracy determines the ev
of computerized analysis procedures. For this reason,
be taken to improve the probability of rugged segment
such as industrial inspection applications, at least some
the environment is possible at times. The experienced
designer invariably pays considerable attention to suc
```

2. text broken.tif

Code:



```python
# repair_broken_image_w.py
import cv2
import os
import pytesseract

class ImageProcessor:
    def __init__(self, image_path):
        self.image_path = image_path
        self.image = self._load_image()
        self.results_dir = "results"
        self._setup_results_dir()

    def _load_image(self):
        image = cv2.imread(self.image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            raise FileNotFoundError(f"Unable to load image: {self.image_path}")
        return image

    def _setup_results_dir(self):
        if not os.path.exists(self.results_dir):
            os.makedirs(self.results_dir)

    def _get_output_filename(self, suffix):
        base_name = os.path.splitext(os.path.basename(self.image_path))[0]
        return os.path.join(self.results_dir, f"{base_name}{suffix}")

    def write_image(self, suffix, img):
        output_path = self._get_output_filename(suffix)
        cv2.imwrite(output_path, img)
        print(f"Image saved to {output_path}")
        self.image = img

    def repair_broken_image_w(self):
        _, binary = cv2.threshold(self.image, 127, 255, cv2.THRESH_BINARY)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (1, 1))
        eroded = cv2.erode(binary, kernel, iterations=1)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
        dilated = cv2.dilate(eroded, kernel, iterations=1)
        self.write_image("_proc.png", dilated)
```



```python
    def ocr_image(self):
        text = pytesseract.image_to_string(self.image, lang='eng')
        output_path = self._get_output_filename(".txt")
        with open(output_path, 'w', encoding='utf-8') as file:
            file.write(text)
        print(f"Extracted text saved to {output_path}")

if __name__ == "__main__":
    processor = ImageProcessor("text-broken.tif")
    processor.repair_broken_image_w()
    processor.ocr_image()
```

Approach:

1. Image Processing Steps

(1) Loading the Image

The process starts by loading the input image in grayscale mode using OpenCV. Grayscale mode simplifies the image by reducing it to a single channel of intensity values ranging from 0 (black) to 255 (white). This step helps focus on the structural and text details of the image without the complexity of color information.

(2) Binary Thresholding

Binary thresholding is applied to convert the grayscale image into a binary image, where each pixel is assigned a value of either 0 (black) or 255 (white). This is determined by a fixed threshold value of 127:

- Pixels with intensity greater than or equal to 127 are set to 255.
- Pixels with intensity below 127 are set to 0.

This transformation enhances the separation between the foreground (text or shapes) and the background, making the structural features of the image more prominent and easier to repair.

(3) Morphological Operations

To repair broken or fragmented parts of the image (e.g., broken lines, disconnected text), two morphological operations are performed:

- **Erosion:**
  The binary image is eroded using a small elliptical kernel of size (1, 1). Erosion reduces the thickness of white regions (foreground) by removing pixels along their boundaries. This step eliminates small noise and helps refine the edges of the text or shapes.
- **Dilation:**
  After erosion, dilation is applied using a larger elliptical kernel of size (3, 3). Dilation increases the thickness of white regions by adding pixels around their boundaries. This step reconnects fragmented components (e.g., broken lines or characters) and restores the continuity of shapes.

The combination of erosion and dilation is often referred to as "closing." It helps clean up and enhance the binary image while preserving the integrity of the textual or structural elements.

(4) Saving the Processed Image

The repaired binary image is saved to a specified directory with a modified filename. This allows users to verify the changes visually or use the cleaned image for further processing, such as OCR.

2. Optical Character Recognition (OCR)

(1) Overview of OCR

OCR (Optical Character Recognition) converts text in images into machine-readable and editable text formats. This program leverages the Tesseract OCR engine through its Python wrapper pytesseract. The OCR step focuses on extracting text content from the repaired binary image.

(2) Processing the Image with OCR

The cleaned binary image, now free of noise and disconnected components, is passed to the OCR engine. The steps involved include:

1. The engine interprets the pixel patterns in the image.
2. It matches these patterns to known textual templates (letters, numbers, symbols) using its pre-trained model.
3. Extracted text is output as a string. In this case, the language is explicitly set to English (lang='eng'), optimizing the engine for recognizing English text.

(3) Saving the Extracted Text

The extracted text is saved into a .txt file, ensuring the information can be reused or further analyzed. Key details include:

- The text is stored sequentially, preserving the order detected by the OCR engine.
- UTF-8 encoding is used to ensure compatibility across different platforms and applications.

3. Summary

Image Repair Goals:

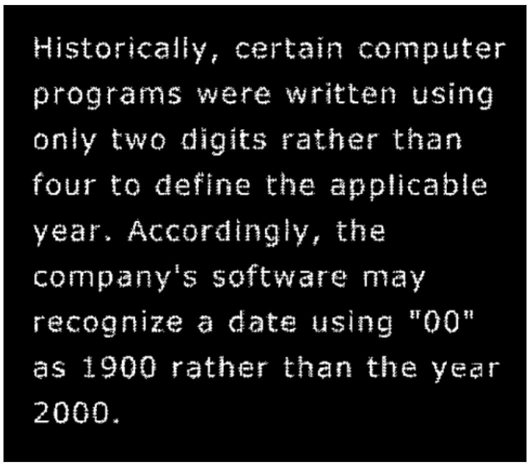
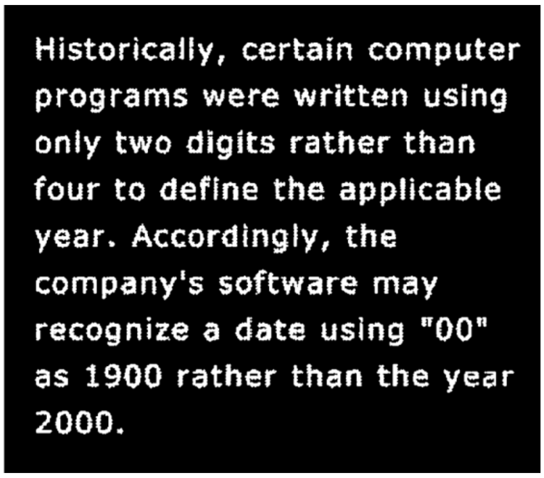
- Remove noise and artifacts from the image.

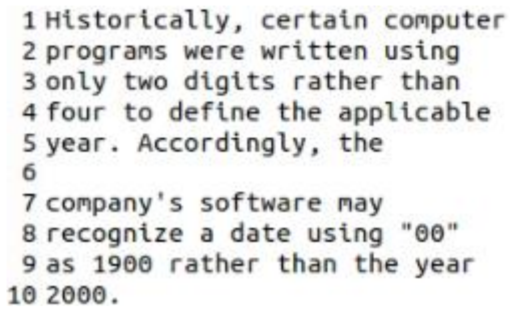
- Repair broken or fragmented text and shapes, making them continuous and well-defined.
- Optimize the image for text recognition by ensuring clear boundaries and minimal distortion.

OCR Goals:

- Accurately extract textual content from images.
- Convert non-editable image text into digital formats for documentation, analysis, or archiving.

Result:

 Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.	 Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.
Origin	**Process**

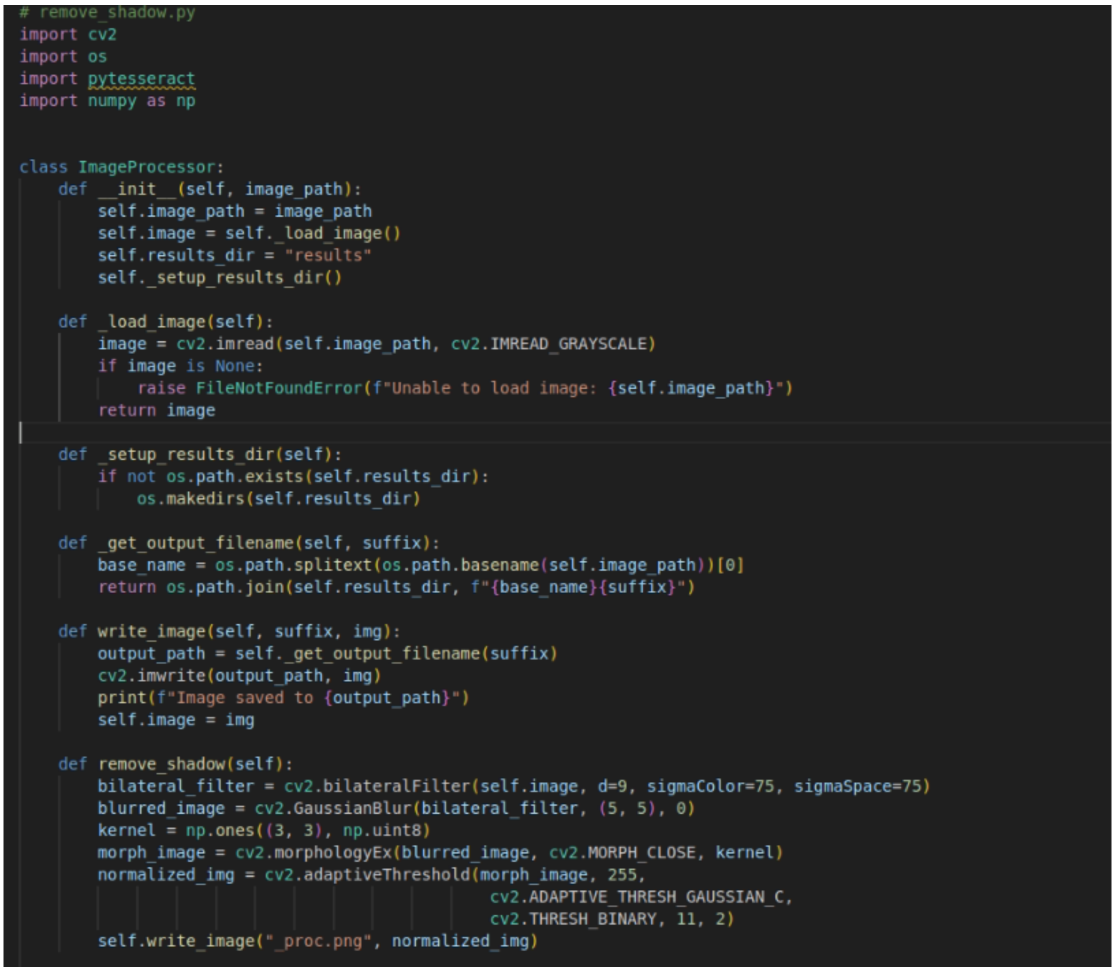


```
Historically, certain computer
programs were written using
only two digits rather than
four to define the applicable
year. Accordingly, the

company's software may
recognize a date using "00"
as 1900 rather than the year
2000.
```

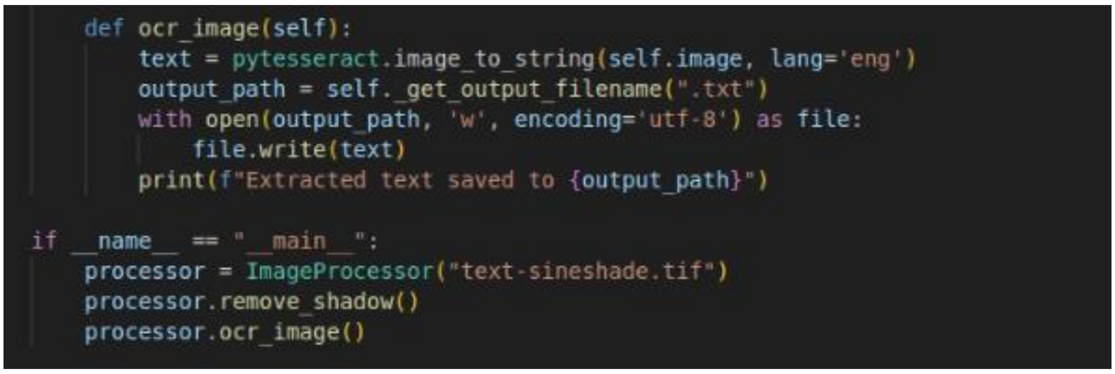
Txt file

3. text-sineshade.tif

Code:



```python
# remove_shadow.py
import cv2
import os
import pytesseract
import numpy as np


class ImageProcessor:
    def __init__(self, image_path):
        self.image_path = image_path
        self.image = self._load_image()
        self.results_dir = "results"
        self._setup_results_dir()

    def _load_image(self):
        image = cv2.imread(self.image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            raise FileNotFoundError(f"Unable to load image: {self.image_path}")
        return image

    def _setup_results_dir(self):
        if not os.path.exists(self.results_dir):
            os.makedirs(self.results_dir)

    def _get_output_filename(self, suffix):
        base_name = os.path.splitext(os.path.basename(self.image_path))[0]
        return os.path.join(self.results_dir, f"{base_name}{suffix}")

    def write_image(self, suffix, img):
        output_path = self._get_output_filename(suffix)
        cv2.imwrite(output_path, img)
        print(f"Image saved to {output_path}")
        self.image = img

    def remove_shadow(self):
        bilateral_filter = cv2.bilateralFilter(self.image, d=9, sigmaColor=75, sigmaSpace=75)
        blurred_image = cv2.GaussianBlur(bilateral_filter, (5, 5), 0)
        kernel = np.ones((3, 3), np.uint8)
        morph_image = cv2.morphologyEx(blurred_image, cv2.MORPH_CLOSE, kernel)
        normalized_img = cv2.adaptiveThreshold(morph_image, 255,
                                               cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                               cv2.THRESH_BINARY, 11, 2)
        self.write_image("_proc.png", normalized_img)
```



```python
    def ocr_image(self):
        text = pytesseract.image_to_string(self.image, lang='eng')
        output_path = self._get_output_filename(".txt")
        with open(output_path, 'w', encoding='utf-8') as file:
            file.write(text)
        print(f"Extracted text saved to {output_path}")

if __name__ == "__main__":
    processor = ImageProcessor("text-sineshade.tif")
    processor.remove_shadow()
    processor.ocr_image()
```

Approach:

1. Image Processing Method

(1) Loading the Image

- **Steps**: The program first uses OpenCV to load the input image in grayscale mode. This simplifies a multi-channel color image (e.g., RGB) into a single-channel intensity image. Each pixel is represented by an intensity value ranging from 0 (black) to 255 (white).
- **Objective**: To simplify the image processing workflow by reducing data dimensions and focusing on structural features and contrast (e.g., differentiating shadows from text).
- **Error Handling**: If the image cannot be loaded, the program raises an error, prompting the user to check the file path or integrity of the image.

(2) Shadow Removal

Shadows can interfere with text extraction. The program removes shadows through a multi-step process involving filtering and morphological operations:

1. **Bilateral Filtering**:
    - Bilateral filtering is a nonlinear filtering technique that smooths the image while preserving edges.
    - **Parameters**:
        - d=9: Diameter of the pixel neighborhood.
        - sigmaColor=75: Standard deviation for intensity differences, controlling the range of filtering.
        - sigmaSpace=75: Standard deviation for spatial distance, controlling the influence range of neighboring pixels.
    - **Objective**: To smooth intensity values, reduce shadow intensity fluctuations, and retain details (e.g., text edges).
2. **Gaussian Blur**:
    - Applies further smoothing to the image filtered by the bilateral filter using a 5x5 Gaussian kernel.
    - **Objective**: To further reduce the effect of shadows and smooth local intensity variations.
3. **Morphological Closing**:
    - Uses a 3x3 kernel to perform morphological closing (dilation followed by erosion).
    - **Objective**: To eliminate small shadows or gaps and smooth transitions in larger bright or dark regions, making text boundaries more distinct.
4. **Adaptive Thresholding**:

    - Applies Gaussian adaptive thresholding to convert the processed image into a binary image.
    - **Parameters**:
        - cv2.ADAPTIVE_THRESH_GAUSSIAN_C: Uses Gaussian weights to calculate the local threshold.
        - blockSize=11: Size of the local region used for threshold calculation.
        - C=2: Fine-tuning constant for the threshold.
    - **Objective**: Adjust the threshold dynamically based on local brightness, ensuring proper separation of shadows and text for improved readability.
5. **Saving the Result**:
    - The shadow-removed, binarized image is saved to a specified directory for further inspection or processing.

2. Optical Character Recognition (OCR)

(1) Introduction to OCR

- **Optical Character Recognition (OCR)** is a technology that converts text within an image into machine-readable text.
- The program uses the Tesseract OCR engine via its Python interface, pytesseract.
- Preprocessing ensures the image is shadow-free and binarized, providing optimal conditions for text recognition.

(2) OCR Process

1. **Image Parsing**:
    - Tesseract divides the processed binary image into individual text regions.
    - Each region's pixel patterns are matched against internal font templates.
    - The results are converted into string form for output.
2. **Language Setting**:
    - lang='eng' specifies English as the recognition language, optimizing accuracy for English characters.
    - Appropriate language settings enhance recognition by using corresponding font templates.

(3) Saving Extracted Text

- **Steps**:
    - The extracted text is saved in a .txt file with UTF-8 encoding.
    - Text regions are arranged in the order they appear in the image, maintaining the original document structure.
- **Objective**:
    - To make textual information reusable, such as for searching, analysis, or archival purposes.

3.Summary

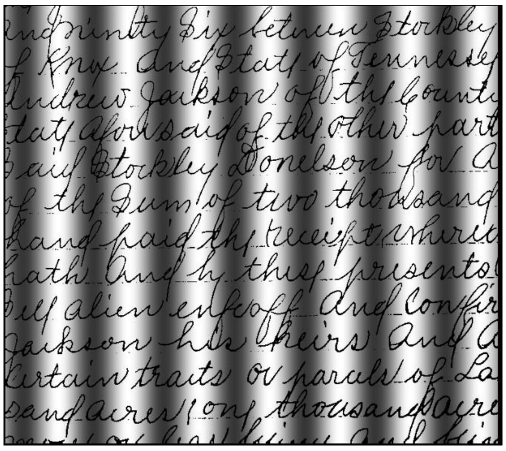
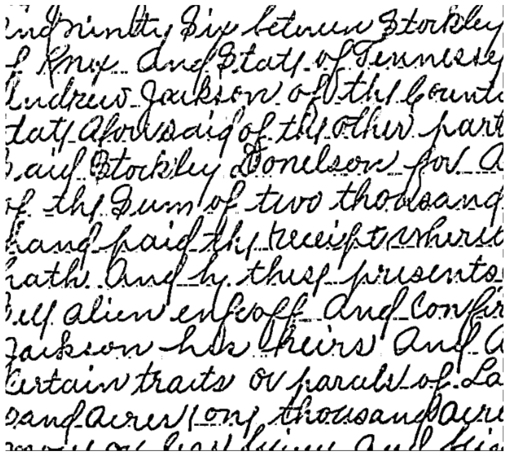
Objectives of Image Processing:

- To remove shadows and noise.
- To enhance text boundaries, contrast, and clarity.
- To prepare the image with clear and continuous text patterns for OCR.

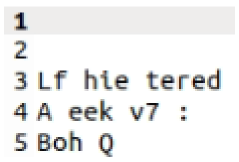
Objectives of OCR:

- To accurately extract text from images.
- To automate data digitization processes.

Result:

	
Origin	**Process**

nd ninety Six between Stokley
f Knox And State of Tennessee
Andrew Jackson of the County
tate aforesaid of the other part
said Stockley Donelson for A
of the Sum of two thousand
hand paid the receipt whereof
hath And by these presents
ell alien enfeoff And Confir
Jackson his heirs And A
certain traits or parcels of La
sand acres (one thousand acre

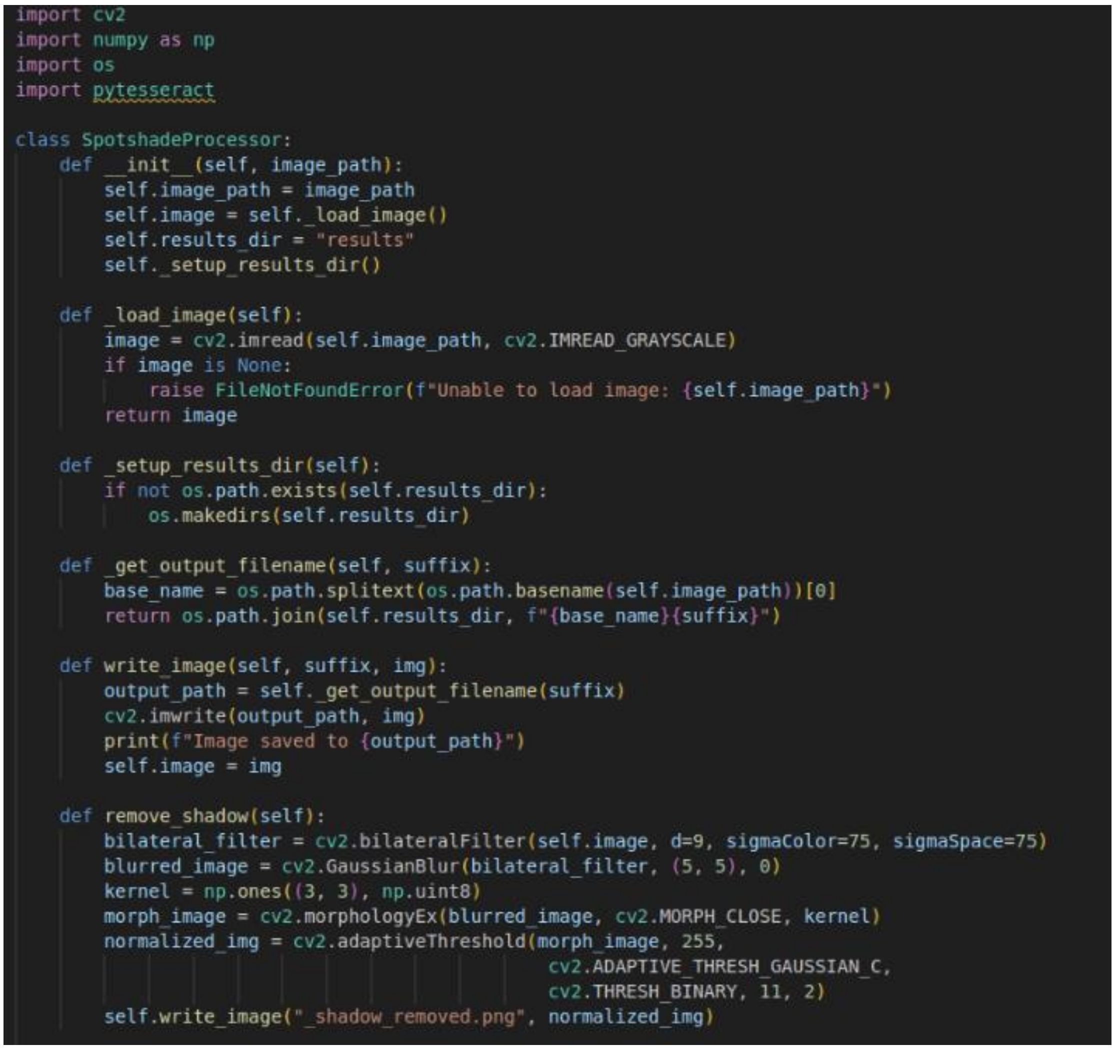


```


Lf hie tered
A eek v7 :
Boh Q
```

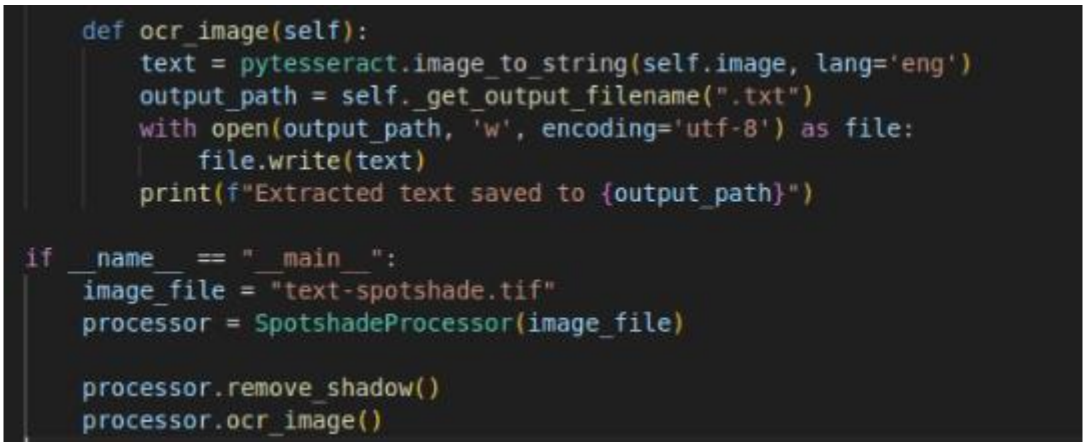
Txt file

4. test-spotshade.tif

Code:



```python
import cv2
import numpy as np
import os
import pytesseract

class SpotshadeProcessor:
    def __init__(self, image_path):
        self.image_path = image_path
        self.image = self._load_image()
        self.results_dir = "results"
        self._setup_results_dir()

    def _load_image(self):
        image = cv2.imread(self.image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            raise FileNotFoundError(f"Unable to load image: {self.image_path}")
        return image

    def _setup_results_dir(self):
        if not os.path.exists(self.results_dir):
            os.makedirs(self.results_dir)

    def _get_output_filename(self, suffix):
        base_name = os.path.splitext(os.path.basename(self.image_path))[0]
        return os.path.join(self.results_dir, f"{base_name}{suffix}")

    def write_image(self, suffix, img):
        output_path = self._get_output_filename(suffix)
        cv2.imwrite(output_path, img)
        print(f"Image saved to {output_path}")
        self.image = img

    def remove_shadow(self):
        bilateral_filter = cv2.bilateralFilter(self.image, d=9, sigmaColor=75, sigmaSpace=75)
        blurred_image = cv2.GaussianBlur(bilateral_filter, (5, 5), 0)
        kernel = np.ones((3, 3), np.uint8)
        morph_image = cv2.morphologyEx(blurred_image, cv2.MORPH_CLOSE, kernel)
        normalized_img = cv2.adaptiveThreshold(morph_image, 255,
                                               cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                               cv2.THRESH_BINARY, 11, 2)
        self.write_image("_shadow_removed.png", normalized_img)
```



```
    def ocr_image(self):
        text = pytesseract.image_to_string(self.image, lang='eng')
        output_path = self._get_output_filename(".txt")
        with open(output_path, 'w', encoding='utf-8') as file:
            file.write(text)
        print(f"Extracted text saved to {output_path}")

if __name__ == "__main__":
    image_file = "text-spotshade.tif"
    processor = SpotshadeProcessor(image_file)

    processor.remove_shadow()
    processor.ocr_image()
```

Approach:

1. Image Processing Steps

(1) Load Image

The process starts by loading the input image in grayscale mode using OpenCV. Grayscale mode simplifies the image into a single-channel intensity value ranging from 0 (black) to 255 (white). This step focuses on the image's structure and text details without the interference of color information, making it easier for subsequent processing and analysis.

(2) Shadow Removal

The main goal of this step is to remove shadows that may obscure text or important structural elements, thereby improving text visibility for further processing. The shadow removal process includes the following stages:

(a) Bilateral Filter
The image is first passed through a bilateral filter, a type of filter that removes noise while preserving edges. The bilateral filter considers both spatial distance and intensity differences between pixels, ensuring that text edges are preserved while smoothing out shadowed areas. This filtering step prepares the image for the next processing stages.

(b) Gaussian Blur
After applying the bilateral filter, a Gaussian blur is applied to further smooth the image. This step helps reduce any remaining noise, creating a more uniform background that aids in the subsequent thresholding process.

(c) Morphological Operations
Morphological operations such as erosion and dilation are used to clean up the image. The erosion operation reduces the thickness of the white regions (foreground) by removing boundary pixels, which helps eliminate small noise and refine the edges of text or shapes. The dilation operation increases the thickness of the white regions by adding pixels around the boundaries, reconnecting fragmented parts (such as broken lines or text). The combination of erosion and dilation is known as "closing," which helps to enhance and clean up the binary image while maintaining the integrity of text or structural elements.

(d) Adaptive Thresholding
Adaptive thresholding is applied to convert the image into a binary image. In this step, all pixel values greater than the threshold are set to white (255), and those below the threshold are set to black (0). Adaptive thresholding dynamically adjusts the threshold based on the neighboring pixels, enabling effective binarization even under varying lighting conditions.

(3) Save Processed Image

The processed image (with shadows removed and structures enhanced) will be saved to the specified directory with a modified filename. This allows the user to review the results or use the cleaned image for subsequent steps, such as OCR.

2. Optical Character Recognition (OCR)

(1) Overview of OCR

Optical Character Recognition (OCR) converts text in an image into a machine-readable and editable text format. This program uses the Tesseract OCR engine via its Python interface pytesseract to perform the processing. The goal is to extract the text content from the processed image.

(2) Extract Text Using OCR

The cleaned binary image (free from shadows and noise) is passed to the OCR engine for processing. The OCR engine performs the following steps:

- The engine analyzes the pixel patterns in the image.
- These patterns are matched with trained templates (letters, numbers, symbols, etc.).

- The extracted text is returned as a string.

The language is set to English (lang='eng') to optimize recognition for English text.

(3) Save Extracted Text

The extracted text is saved to a .txt file to ensure the information is reusable or available for further analysis. Specific details include:

- The text is saved in the order detected by the OCR engine, preserving the original structure of the text.
- The text file is encoded in UTF-8 to ensure compatibility across platforms and applications.

3. Purpose and Applications

Image Processing Goals:

- **Remove Noise and Imperfections**: Eliminate noise and unwanted artifacts from the image.
- **Repair Broken or Fragmented Text and Shapes**: Ensure text and structural elements are continuous and clear.
- **Enhance the Image for OCR**: Strengthen the boundaries of the text in the image and reduce distortion to improve OCR performance.

OCR Goals:

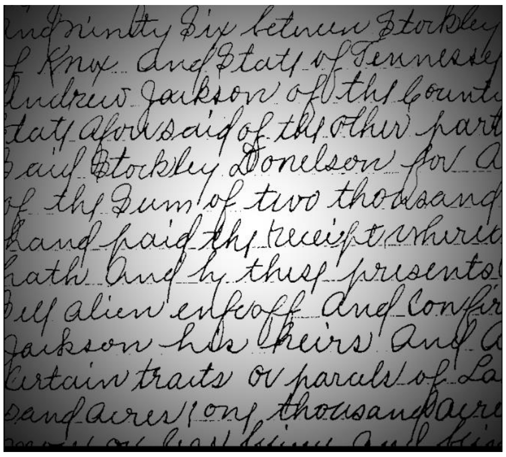
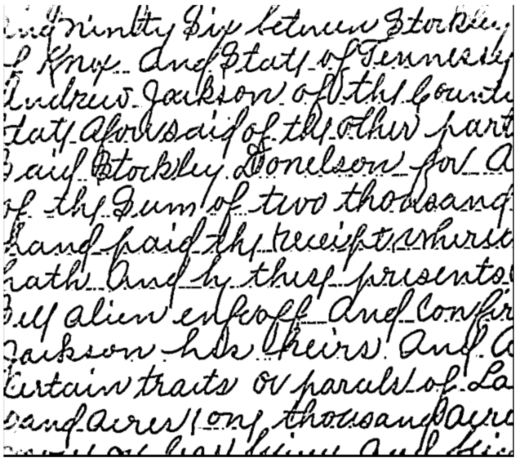
- **Accurately Extract Text**: Extract text content from the image with high accuracy.
- **Convert to Editable Format**: Convert non-editable text in the image to a digital format for easy storage, analysis, or further processing.

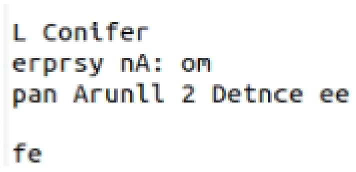
Common Applications:

- **Document Digitization**: Convert scanned documents, damaged records, or historical manuscripts into digital formats.
- **Extract Content from Degraded or Noisy Images**: Retrieve readable information from blurry or noisy images.
- **Improve Text Recognition Accuracy**: Enhance the image quality to improve the accuracy of OCR, supporting automated data extraction and analysis.

This method ensures efficient image processing and text extraction, even when dealing with images containing noise or fragmented text. It is suitable for tasks such as document digitization, image cleanup, and OCR applications, especially for improving precision and clarity.

Result:

 ind ninety Six between Stockley / f Knox And State of Tennessee / Andrew Jackson of the Count / tate aforesaid of the other part / aid Stockley Donelson for A / f the Sum of two thousand / hand paid the receipt whereo / hath And by these presents / ell alien enfeoff And Confir / Jackson his heirs And a / ertain tracts or parcels of La / sand acres (one thousand acre / [illegible]	 ind ninety Six between Stockley / f Knox And State of Tennessee / Andrew Jackson of the Count / tate aforesaid of the other part / aid Stockley Donelson for A / f the Sum of two thousand / hand paid the receipt whereo / hath And by these presents / ell alien enfeoff And Confir / Jackson his heirs And a / ertain tracts or parcels of La / sand acres (one thousand acre / [illegible]
Origin	**Process**



```
L Conifer
erprsy nA: om
pan Arunll 2 Detnce ee

fe
```

Txt file