Project Report
on
Document Image Analysis for Forensic Application

B. TECH IN INFORMATION TECHNOLOGY
2021 – 2025

University School of Information and Technology

Guru Gobind Singh Indraprastha University

GUIDED BY:

Dr. Anuradha Chug (Int.)

Mr. Bhupendra Kumar (Ext.)

SUBMITTED BY:

Ujjwal Gupta

00516401521

BTECH IT (4th Sem.)

TABLE OF CONTENTS

Table of Contents

TABLE OF CONTENTS 2
1. INTRODUCTION 6
2. PROBLEM AND CHALLENGES 6
3. OBJECTIVES – 7
4. CURRENT STATE OF ART 7
5. SCOPE OF WORK 7
6. DEVELOPMENT ENVIRONMENT– 11
7. TECHNOLOGY USED 11
8. DEVELOPMENT OR CODE SNIPPET 13
9. RESULT AND OUTPUT 30
10. CONCLUSION 33
11. BENEFITS – 33
12. REFERENCES 33

Certificate

I, Ujjwal Gupta, Enroll No. 00516401521 certify that the Project Report (BTECH-IT) entitled "Image Analysis for Forensic Application" is done by me and it is an authentic work carried out by me at C-DAC, Noida. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of the Student
Date:

Certified that the Project Report (BTECH-IT) entitled "Image Analysis for Forensic Application" done by Mr Ujjwal Gupta Roll No. 00516401521, is completed under my guidance.

Signature of the Guide
Date:
Name of the Guide: Mr. Bhupendra Kumar
Designation: Joint Director
Address: C-DAC, Noida

About the Organization

The Centre for Development of Advanced Computing (C-DAC) in Noida stands as a prominent branch of the esteemed C-DAC organization, which holds the distinction of being the premier research and development establishment operating under the Ministry of Electronics and Information Technology (MeitY) within India. C-DAC Noida, in particular, has carved a niche for itself by specializing in cutting-edge research and development across various domains, prominently including high-performance computing, advanced networking, innovative software technologies, and language technologies.

With its steadfast commitment to technological advancement, C-DAC Noida has fostered invaluable collaborations with a myriad of organizations, esteemed academic institutions, and diverse industries. These strategic partnerships have been instrumental in catalyzing pioneering research and development endeavors across an array of specialized areas. Through these collaborative initiatives, C-DAC Noida has exhibited an exceptional capacity to address complex challenges and deliver impactful solutions.

One of the standout features of C-DAC Noida lies in its comprehensive portfolio of training programs and courses. These initiatives are meticulously designed to elevate the skills and knowledge of IT professionals, thereby empowering them to navigate the dynamic landscape of information technology with unparalleled expertise. By offering a diverse array of training opportunities, C-DAC Noida has effectively contributed to the upskilling and professional growth of countless individuals in the IT industry.

In essence, C-DAC Noida stands as a beacon of innovation and excellence within the realm of technology and research. Its unwavering dedication to pushing the boundaries of high-performance computing, networking, software technologies, and language technologies underscores its pivotal role in shaping the technological landscape of India. Through its collaborative ethos, it has not only enriched the domain of research but also facilitated the dissemination of knowledge, ultimately propelling the nation's progress in the digital age.

Acknowledgment

I am pleased to present this **project report titled Image Analysis for Forensic Application** as a culmination of our dedicated efforts and collaboration. I would like to extend my heartfelt gratitude to **Mr. Bhupendra Kumar and Mr. Deepam Ashok Kalkar** for their unwavering commitment and invaluable contributions at every stage of this endeavor. Their collective expertise and dedication have been instrumental in shaping the project's outcomes.

I express my deep appreciation to **Dr. Anuradha Chug** for their guidance, mentorship, and constructive feedback. Their insights have been pivotal in steering the project in the right direction and enhancing its quality. Furthermore, I extend my thanks to **C-DAC, Noida** for their support, be it through resources, expertise, or collaborative opportunities.
Lastly, my gratitude goes to my family and friends for their unwavering encouragement and understanding during this demanding phase of the project.

This report reflects our collective dedication to advancing knowledge and making a meaningful impact. It is my hope that the insights presented herein contribute positively to the relevant field and pave the way for future research and developments.

I extend my sincere thanks once again for the unwavering support and guidance that I have received throughout this endeavor.

Warm regards,

Ujjwal Gupta

1. INTRODUCTION

In the realm of forensic investigations, the scrutiny of questioned documents is a crucial aspect of ascertaining authenticity, authorship verification, and detecting potential forgeries.

The examination of documents plays a pivotal role in legal cases, fraud detection, and other investigative scenarios where the veracity of documents serves as critical evidence. To aid forensic experts in this intricate task, specialized software and toolsets have been developed, collectively known as forensic applications for document images.

These sophisticated applications are designed to analyze and examine questioned documents with precision and efficiency, leveraging advanced image processing techniques and cutting-edge algorithms.

By addressing challenges such as noisy and degraded document images, complex handwriting analysis, and signature verification, these tools significantly enhance the capabilities of forensic experts.

Forensic applications for document images encompass a wide array of features and functionalities tailored to the distinct demands of forensic investigations.

From image preprocessing to extract clear and relevant information to **writer identification** and **signature verification**, these tools empower investigators to uncover the truth hidden within documents.

Forensic applications for document images play a vital role in modern investigative procedures, augmenting the abilities of forensic experts and fostering accuracy, efficiency, and trust in the forensic document examination process. As technological advancements continue, these tools will remain essential assets in unraveling the truth from questioned documents and upholding justice in legal systems worldwide.

2. PROBLEM AND CHALLENGES

- Matching signatures and handwriting in forensic document examination is a difficult job because everyone writes in their own unique way. It's like how everyone has their own way of drawing pictures or writing their name. This makes it hard to find consistent things to compare when trying to figure out if two signatures or pieces of writing were done by the same person.

- The presence of noise and degradation in questioned documents further complicates the verification process, affecting the reliability of the results.

- Additionally, distinguishing between genuine variations and intentional forgeries poses a critical challenge, as it requires a robust and precise method to ensure accurate matching and authentication of signatures and handwriting. Addressing these obstacles is essential for enhancing the efficiency and accuracy of forensic investigations and document authentication processes.

3. OBJECTIVES –

- Implementation of various imaging filter to analyze signature images
- Development of custom image processing algorithms for structural information analysis.

4. CURRENT STATE OF ART

Examination of handwritten content is a pretty common task performed by document examiners. Authorship of the questioned document being the prime issue, the document-forensic expert manually deals with various aspects of the authenticity of the documents.
In a recent work by Kumar and Bhatia [3], a survey was presented on writer-dependent and independent approaches, incorporating almost every recent work, along with an outlook into future works. A comparative work using handcrafted and auto-derived features (CNN) extracted from intra-variable writing was analyzed by Adak et al. [4] using models like SVM with data augmentation in order to generate large datasets.

5. SCOPE OF WORK

Pre-processing Importance: Image enhancement is crucial in forensic document examination as many suspected documents suffer from natural or deliberately induced noises, making them challenging to analyze.

Handling Varied Document Degradations: Suspected documents often arrive in poor conditions from crime scenes, intentionally damaged to manipulate evidence, requiring manual processing for forensic verification.
Maximum Imaging Filters: For this purpose, a wide range of imaging filters to recover degraded images and facilitate forensic document analysis.

The scope of work mainly includes Implementation of imaging filter to analyze degraded images.
The list of Image Filter operation as follows

Image Filter Operations	Functions
Contour Display	Contour Display is a visualization technique that identifies and highlights the boundaries of objects within a document. By outlining the edges of different elements, such as characters, signatures, or other objects, it offers a clear and distinct representation of the document's structure. This helps forensic experts and investigators to easily identify and analyze individual components of the document for further examination.
Stroke Thinning	Stroke Thinning is a process that reduces the thickness of handwritten strokes within a document. By slimming down the width of each stroke, finer details in characters and signatures become more pronounced. This technique is particularly useful for improving the accuracy of feature extraction algorithms, making it easier to discern unique writing characteristics and enhancing the precision of writer and signature verification.

Stroke Thickening	Conversely, Stroke Thickening increases the thickness of handwritten strokes in a document. By enhancing the visual representation of strokes, characters and signatures may appear bolder and more prominent. This can aid in improving the legibility of handwritten content and make it easier for forensic experts to analyze and compare writing styles for identification and authentication purposes.
Adaptive Smoothing	Adaptive Smoothing is a noise reduction technique that considers local image characteristics. It selectively applies smoothing or blurring to different regions of the document based on their complexity and noise levels. This process preserves important details while effectively reducing unwanted artifacts, ensuring that the document's content remains clear and readable, even in the presence of noise or imperfections.
Contrast Correction	Contrast Correction adjusts the image contrast to enhance the difference between dark and light areas. By increasing the contrast, the distinctions between foreground and background elements become more pronounced, leading to improved visibility and clarity of the document's contents. This helps in better distinguishing individual characters and signatures, aiding in the verification process.
Contrast Stretching	Contrast Stretching is an image enhancement technique that expands the range of pixel intensities in an image. By rescaling the intensity values, it spreads the darkest and lightest pixels across the entire range, resulting in increased contrast and better visibility of details. This process enhances the overall appearance of the image, making it easier to discern fine features and aiding in forensic analysis and document verification.
Gaussian Sharpen	Gaussian Sharpen is an edge enhancement technique that uses a Gaussian filter to emphasize image details. By accentuating the edges, characters and signature strokes become sharper and more well-defined. This sharpening effect is especially valuable when analyzing low-resolution or blurred documents, as it brings out crucial information for accurate forensic examination.
Saturation Correction	Saturation Correction adjusts the intensity of colors in the document. By modifying the saturation levels, the vibrancy and richness of colors are altered, which can aid in better visualizing and interpreting colored elements, such as ink variations or signatures in different colors. This can help forensic experts in analyzing multicolored documents more effectively.
Gamma Correction	Gamma Correction is a brightness adjustment technique that alters the brightness levels of pixel values using a gamma function. This helps in expanding or compressing the pixel intensity values, resulting in better visibility of darker or lighter areas in the document. By adjusting the gamma value, experts can optimize the document's appearance for easier examination and analysis.

Brightness Correction	Brightness Correction is a simple image enhancement technique that alters the overall brightness of the document. By increasing or decreasing the brightness levels, the visibility of the content can be improved, especially in cases where documents are poorly scanned or have uneven illumination. This correction ensures that the document is displayed at an optimal brightness level for analysis.
Mean Filter	The Mean Filter is a smoothing technique that replaces each pixel value with the average of its neighboring pixel values. By calculating the local average, this filter helps to reduce noise and blur artifacts, resulting in a cleaner and more refined representation of the document. It aids in creating a smoother background, making it easier to focus on the handwritten content and signatures.
Median Filter	The Median Filter is another noise reduction technique that replaces each pixel with the median value of its neighboring pixels. Unlike the mean filter, the median filter is robust against outliers and preserves edge details. It effectively removes salt-and-pepper noise and other irregularities, improving the overall image quality and facilitating accurate feature extraction for signature and writer verification.
Bradley Local Threshold	Bradley Local Threshold is an adaptive thresholding technique used to segment the document into text and background regions. It calculates a local threshold for each pixel based on the average intensity of the neighboring pixels. This method helps in isolating handwritten content from the background, making it easier to analyze and process individual components of the document.
Bilateral Smoothing	Bilateral Smoothing is a noise reduction technique that reduces noise while preserving sharp edges. It considers both spatial and intensity differences in the image, ensuring that important edge information is retained while effectively reducing noise. This is particularly useful for handwritten documents, where preserving the sharpness of strokes is crucial for accurate analysis and verification.
Conservative Smooth	Conservative Smooth is a gentle smoothing technique that reduces noise with minimal blurring of important features. It targets only the noisy regions of the document while leaving the rest of the content relatively unaffected. This conservative approach is helpful in maintaining the integrity of the handwritten content, allowing forensic experts to analyze the document without losing critical details.
Histogram Equalize	Histogram Equalization is an image enhancement technique that redistributes pixel intensity values across the entire image histogram. By stretching the intensity range, this process increases the contrast and improves the overall visual appearance of the document. It enhances the visibility of character strokes and signatures, making them more distinguishable for verification and

analysis.

Morphological Opening	Morphological Opening is a morphological operation that removes small noise from the document by eroding the image and then dilating it. This process helps in eliminating small unwanted artifacts and thin lines, creating a smoother and cleaner representation of the document. It prepares the document for further analysis and feature extraction.
Morphological Closing	Morphological Closing is another morphological operation that fills small gaps and holes in the document by dilating the image and then eroding it. This process helps in closing gaps between characters and strokes, ensuring a continuous representation of the handwritten content. It aids in improving the connectivity of strokes for accurate feature extraction and verification.

6. DEVELOPMENT ENVIRONMENT–

The Standards below are the ones used while developing and thus are not the minimum requirements.

- **Hardware Environments:**

OS	Windows 11
Processor	Intel i5
RAM	8GB
SSD	250GB
Hard Disk	500GB

- **Software Environments:**

IDE	PyCharm Community Edition 2023.2
Programming Language	Python (3.10 version 64-bit)
Image Processing library	Open Source Computer Vision Library (OpenCV)
Python Library	Numpy
	Pandas
	Matplotlib

7. TECHNOLOGY USED

➢ **Open-Source Computer Vision Library (OpenCV)**

- OpenCV (Open-Source Computer Vision) is a widely used open-source library that empowers developers with tools and functions to perform a myriad of computer vision tasks.
- Renowned for its versatility and efficiency, OpenCV offers a comprehensive range of image and video processing capabilities, pattern recognition algorithms, and machine learning tools.
- From object detection and facial recognition to image manipulation and augmented reality applications, OpenCV plays a pivotal role in enabling the development of cutting-edge computer vision solutions across diverse industries, fostering innovation and transformation in the field of visual data analysis and interpretation.

➢ **Numpy**

- NumPy (Numerical Python) stands as a foundational library in the Python ecosystem, providing essential tools for numerical computations and data manipulation.
- Recognized for its efficient array operations and mathematical functions, NumPy forms the bedrock for scientific computing and data analysis.
- Its multidimensional array objects, along with an extensive collection of mathematical routines,

enable users to effortlessly perform complex calculations, data transformations, and statistical analyses.

- With its seamless integration into the Python programming language, NumPy facilitates rapid development of data-driven applications, making it an indispensable tool for researchers, engineers, and data scientists seeking to harness the potential of numerical computing and analysis.

➢ **Pandas**

- Pandas, a pivotal library in the Python ecosystem, empowers data analysts and scientists with robust tools for data manipulation, analysis, and exploration.
- Leveraging its core data structures—Series and DataFrame—Pandas offers a seamless platform for handling structured and tabular data.
- With its intuitive syntax and versatile functions, Pandas facilitates tasks such as data cleaning, transformation, aggregation, and visualization.
- Whether it's loading data from various sources, conducting complex data operations, or summarizing insights, Pandas provides a versatile toolkit that streamlines the data analysis process.
- Renowned for its ability to handle missing data, time series, and heterogeneous datasets, Pandas plays a vital role in simplifying data-centric workflows, making it a cornerstone for anyone involved in data-driven decision-making and exploration.

➢ **Matplotlib**

- Matplotlib, a fundamental data visualization library in Python, empowers researchers, analysts, and scientists to create compelling and informative visual representations of their data.
- With a rich collection of plotting functions, Matplotlib offers a versatile platform for generating a wide range of static, interactive, and publication-quality visualizations.
- From line plots and scatter plots to bar charts, histograms, and heatmaps, Matplotlib provides the tools needed to effectively communicate insights and trends within data.
- Its customizable features enable users to fine-tune every aspect of their visualizations, from labels and colors to annotations and layouts.
- By offering a seamless integration with various data analysis libraries, Matplotlib serves as an indispensable resource for conveying complex data relationships and patterns, fostering a deeper understanding of data-driven narratives.

8. DEVELOPMENT OR CODE SNIPPET

---MAIN.PY---

```
from contour_display import highlight_contours
from stroke_thinning import thin_strokes
from stroke_thickening import thicken_strokes
from adaptive_smoothing import adaptive_smooth
from contrast_correction import contrast_correction
from contrast_stretching import contrast_stretching
from gaussian_sharpening import gaussian_sharpen
from saturation_correction import saturation_correction
from gamma_correction import gamma_correction_and_display
from Brightness_correction import adjust_and_show_brightness
from mean_filter import mean_filter_and_display
from median_filter import median_filter_and_display
from bradley_local_threshold import bradley_local_thresholding
from bilateral_smoothing import bilateral_smooth
from Conservative_smooth import conservative_smooth
from Histogram_equalise import histogram_equalize_and_display
from Morphological_opening import morphological_opening_and_display
from Morphological_closing import morphological_closing_and_display


def main():
    # Get the source image from the user
    image_path = input("Enter the path to the image: ")
    while True:
        print("Select an operation:")
        print("1. Contour Display")
        print("2. Stroke Thinning")
        print("3. Stroke Thickening")
        print("4. Adaptive Smoothing")
        print("5. Contrast Correction")
        print("6. Contrast Stretching")
        print("7. Gaussian Sharpening")
        print("8. Saturation Correction")
        print("9. Gamma Correction")
        print("9b. Brightness Correction")
        print("10. Mean Filter")
        print("11. Median Filter")
        print("12. Bradley Local Threshold")
        print("13. Bilateral Smoothing")
        print("14. Conservative Smooth")
        print("15. Histogram Equalize")
        print("16. Morphological Opening")
        print("17. Morphological Closing")
        print("0. Exit")

        choice = input("Enter the number corresponding to the operation: ")

        if choice == "1":
            output_filename = input("Enter the output image filename (without 
extension): ")
            threshold_value = int(input("Enter the threshold value: "))
            contour_color = tuple(map(int, input("Enter the contour color 
(comma-separated RGB values): ").split(',')))
```

```
            contour_thickness = int(input("Enter the contour thickness: "))
            image_path = highlight_contours(image_path, output_filename,
threshold_value, contour_color, contour_thickness)

        elif choice == "2":
            output_path = input("Enter output file name or directory: ")
            # Get additional parameters for stroke thinning
            threshold_value = int(input("Enter threshold value: "))
            min_neighbors = int(input("Enter min neighbors: "))
            max_neighbors = int(input("Enter max neighbors: "))
            max_transitions = int(input("Enter max transitions: "))
            image_path = thin_strokes(image_path, output_path, threshold_value,
min_neighbors, max_neighbors,max_transitions)

        elif choice == "3":
            output_path = input("Enter output file name or directory: ")
            # Get additional parameters for stroke thickening
            kernel_size = int(input("Enter kernel size: "))
            iterations = int(input("Enter number of iterations: "))
            image_path = thicken_strokes(image_path, output_path, (kernel_size,
kernel_size), iterations)

        elif choice == "4":
            output_path = input("Enter output file name or directory: ")
            kernel_size_input = int(input("Enter the kernel size: "))
            image_path = adaptive_smooth(image_path, output_path,
kernel_size_input)

        elif choice == "5":
            output_path = input("Enter output file name or directory: ")
            alpha = float(input("Enter contrast factor: "))
            beta = float(input("Enter brightness factor: "))
            image_path = contrast_correction(image_path, output_path, alpha,
beta)

        elif choice == "6":
            output_path = input("Enter output file name or directory: ")
            image_path = contrast_stretching(image_path, output_path)

        elif choice == "7":
            output_path = input("Enter output file name or directory: ")
            sigma = float(input("Enter the sigma value for Gaussian blur: "))
            image_path = gaussian_sharpen(image_path, output_path, sigma)

        elif choice == "8":
            output_path = input("Enter output file name or directory: ")
            alpha = float(input("Enter saturation scale factor: "))
            beta = float(input("Enter saturation shift factor: "))
            image_path = saturation_correction(image_path, output_path, alpha,
beta)

        elif choice == "9":
            output_path = input("Enter output file name or directory: ")
            gamma_value = float(input("Enter the gamma value: "))
            image_path = gamma_correction_and_display(image_path, gamma_value,
output_path)

        elif choice == "9b":
            output_path = input("Enter output file name or directory: ")
```

```
            brightness_factor = float(input("Enter the brightness factor: "))
            image_path = adjust_and_show_brightness(image_path,
brightness_factor, output_path)

        elif choice == "10":
            output_path = input("Enter output file name or directory: ")
            kernel_size = int(input("Enter the kernel size: "))
            mean_filter_and_display(image_path, kernel_size,output_path)

        elif choice == "11":
            output_path = input("Enter output file name or directory: ")
            kernel_size = int(input("Enter the kernel size: "))
            image_path = median_filter_and_display(image_path, kernel_size,
output_path)

        elif choice == "12":
            output_path = input("Enter output file name or directory: ")
            window_size = int(input("Enter the window size: "))
            threshold = int(input("Enter the threshold percentage: "))
            image_path = bradley_local_thresholding(image_path, output_path,
window_size, threshold)

        elif choice == "13":
            output_path = input("Enter output file name or directory: ")
            diameter = int(input("Enter the diameter: "))
            sigma_color = float(input("Enter the sigma color: "))
            sigma_space = float(input("Enter the sigma space: "))
            image_path = bilateral_smooth(image_path, output_path, diameter,
sigma_color, sigma_space)

        elif choice == "14":
            output_path = input("Enter output file name or directory: ")
            h = float(input("Enter the h value: "))
            search_window = int(input("Enter the search window size: "))
            patch_window = int(input("Enter the patch window size: "))
            image_path = conservative_smooth(image_path, output_path, h,
search_window, patch_window)

        elif choice == "15":
            output_path = input("Enter output file name or directory: ")
            image_path = histogram_equalize_and_display(image_path,
output_path)

        elif choice == "16":
            output_path = input("Enter output file name or directory: ")
            kernel_width = int(input("Enter the kernel width: "))
            kernel_height = int(input("Enter the kernel height: "))
            kernel_size = (kernel_width, kernel_height)
            image_path = morphological_opening_and_display(image_path,
kernel_size, output_path)

        elif choice == "17":
            output_path = input("Enter output file name or directory: ")
            kernel_width = int(input("Enter the kernel width: "))
            kernel_height = int(input("Enter the kernel height: "))
            kernel_size = (kernel_width, kernel_height)
            image_path = morphological_closing_and_display(image_path,
kernel_size, output_path)
```

```
        elif choice == "0":
            exit()

        else:
            print("Invalid choice. Please enter a valid number.")


if __name__ == "__main__":
    main()
------------------------------CONTOUR_DISPLAY.py------------------------------
import cv2
import matplotlib.pyplot as plt


def highlight_contours(image_filename, output_filename=None,
threshold_value=150, contour_color=(0, 255, 0), contour_thickness=2):
    # Read the image
    image = cv2.imread(image_filename)

    # Convert the image to grayscale
    img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply binary thresholding
    ret, thresh = cv2.threshold(img_gray, threshold_value, 255,
cv2.THRESH_BINARY)

    # Detect the contours on the binary image using cv2.CHAIN_APPROX_SIMPLE
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on the original image for `CHAIN_APPROX_SIMPLE`
    image_copy = image.copy()
    cv2.drawContours(image_copy, contours, -1, contour_color,
contour_thickness, cv2.LINE_AA)

    # Display the images using Matplotlib
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(image_copy, cv2.COLOR_BGR2RGB))
    plt.title('Image with Contours')
    plt.axis('off')
    plt.tight_layout()
    # Save the image with drawn contours if output_filename is provided
    if output_filename:
        cv2.imwrite(output_filename, image_copy)
    plt.show()
    return output_filename




-------------------------------STROKE_THINNING.py------------------------------
```

```
import cv2
import matplotlib.pyplot as plt

def thin_strokes(input_image_path, output_image_path, threshold_value=128,
min_neighbors=2, max_neighbors=6, max_transitions=1):
    # Read the source image using cv2.imread
    original_image = cv2.imread(input_image_path)
    source_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
    # Threshold the image to convert it to binary
    _, binary_image = cv2.threshold(source_image, threshold_value, 255,
cv2.THRESH_BINARY)

    # Ensure the image is binary and invert it (foreground pixels become 0,
background pixels become 1)
    binary_image = 1 - (binary_image // 255)

    def check_pixel_deletion(x, y):
        neighbors = [
            binary_image[y - 1, x],
            binary_image[y - 1, x + 1],
            binary_image[y, x + 1],
            binary_image[y + 1, x + 1],
            binary_image[y + 1, x],
            binary_image[y + 1, x - 1],
            binary_image[y, x - 1],
            binary_image[y - 1, x - 1]
        ]
        transitions = sum((a, b) == (0, 1) for a, b in zip(neighbors,
neighbors[1:] + neighbors[:1]))
        return (binary_image[y, x] == 1 and int(min_neighbors) <=
sum(neighbors) <= int(max_neighbors) and transitions == max_transitions
                and neighbors[0] * neighbors[2] * neighbors[4] == 0 and
neighbors[2] * neighbors[4] * neighbors[6] == 0)

    # Perform the Zhang-Suen Thinning Algorithm
    rows, cols = binary_image.shape
    changing = True
    while changing:
        changing = False
        # First sub-iteration
        to_delete = []
        for y in range(1, rows - 1):
            for x in range(1, cols - 1):
                if check_pixel_deletion(x, y):
                    to_delete.append((x, y))
        for x, y in to_delete:
            binary_image[y, x] = 0
            changing = True
        # Second sub-iteration
        to_delete = []
        for y in range(1, rows - 1):
            for x in range(1, cols - 1):
                if check_pixel_deletion(x, y):
                    to_delete.append((x, y))
        for x, y in to_delete:
            binary_image[y, x] = 0
            changing = True

    # Invert the binary image back to its original format
```

```
    binary_image = 1 - binary_image

    # Convert the binary image back to the original color format
    color_image = cv2.cvtColor(binary_image * source_image, cv2.COLOR_GRAY2BGR)

    # Display the images using Matplotlib
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB))
    plt.title('Output Image')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    # Save the color image to the destination path
    cv2.imwrite(output_image_path, color_image)
    return output_image_path

----------------------------stroke_thickening.py------------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt

def thicken_strokes(image_path, output_path, kernel_size=(3, 3), iterations=1):
    # Read the grayscale image
    original_image = cv2.imread(image_path)
    image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
    # Invert the grayscale image (text will become black and background will
become white)
    inverted_image = cv2.bitwise_not(image)

    # Create a structuring element for dilation
    kernel = np.ones(kernel_size, dtype=np.uint8)

    # Perform dilation on the inverted grayscale image
    dilated_inverted_image = cv2.dilate(inverted_image, kernel,
iterations=int(iterations))

    # Invert the dilated image back to the original orientation
    dilated_image = cv2.bitwise_not(dilated_inverted_image)

    # Create a single Matplotlib window with two subplots
    plt.figure(figsize=(10, 5))

    # Original Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')

    # Thickened Image
    plt.subplot(1, 2, 2)
```

```
    plt.imshow(cv2.cvtColor(dilated_image, cv2.COLOR_BGR2RGB))
    plt.title('Thickened Image')
    plt.axis('off')

    # Display the subplots
    plt.tight_layout()
    plt.show()

    # Save the final image
    cv2.imwrite(output_path, dilated_image)
    return output_path
-------------------------------Adaptive Smoothing.py--------------------------
import cv2
import matplotlib.pyplot as plt

def adaptive_smooth(image_path, output_path, kernel_size=5):
    # Load image
    img = cv2.imread(image_path)

    # Display the original and smoothed images using Matplotlib
    plt.figure(figsize=(10, 5))

    # Original Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')

    # Add median filter to image
    img_median = cv2.medianBlur(img, int(kernel_size))

    # Smoothed Image
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(img_median, cv2.COLOR_BGR2RGB))
    plt.title('Smoothed Image')
    plt.axis('off')

    # Display the images
    plt.tight_layout()
    plt.show()

    # Save the output image
    cv2.imwrite(output_path, img_median)
    return output_path

------------------------contrast_correction.py-------------------------------
import cv2
import matplotlib.pyplot as plt

def contrast_correction(image_path, output_path, alpha, beta):
    # Read the image in BGR format (OpenCV default)
    image = cv2.imread(image_path)

    # Display the input and output images using Matplotlib
    plt.figure(figsize=(10, 5))

    # Input Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
    plt.title('Input Image')
    plt.axis('off')

    # Perform contrast correction using the formula: corrected_pixel = alpha * 
original_pixel + beta
    corrected_image = cv2.convertScaleAbs(image, alpha=float(alpha), 
beta=float(beta))

    # Output Image
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(corrected_image, cv2.COLOR_BGR2RGB))
    plt.title('Output Image')
    plt.axis('off')

    # Display the images
    plt.tight_layout()
    plt.show()

    # Save the output image
    cv2.imwrite(output_path, corrected_image)
    return output_path
----------------------------contrast_stretching.py----------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt

def contrast_stretching(image_path, output_path):
    # Read the image
    image = cv2.imread(image_path)

    # Display the input and output images using Matplotlib
    plt.figure(figsize=(10, 5))

    # Input Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Input Image')
    plt.axis('off')

    # Get user-defined parameters
    for channel in range(3):  # Loop through each color channel (R, G, B)
        while True:
            try:
                min_val = int(input(f"Enter the minimum value for channel 
{channel}: "))
                max_val = int(input(f"Enter the maximum value for channel 
{channel}: "))
                break
            except ValueError:
                print("Invalid input. Please enter integer values.")

        # Linearly scale the pixel intensities between 0 and 255
        stretched_channel = ((image[:, :, channel] - min_val) / (max_val - 
min_val)) * 255
        stretched_channel = np.clip(stretched_channel, 0, 255)

        # Replace the channel in the stretched image
        image[:, :, channel] = stretched_channel
```

```
    # Convert the pixel values to integers
    stretched_image = image.astype('uint8')

    # Output Image
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(stretched_image, cv2.COLOR_BGR2RGB))
    plt.title('Output Image')
    plt.axis('off')

    # Display the images
    plt.tight_layout()
    plt.show()

    # Save the output image
    cv2.imwrite(output_path, stretched_image)
    return output_path
---------------------------gaussian_sharpening.py------------------------------
import cv2

import matplotlib.pyplot as plt

def gaussian_sharpen(image_path, output_path, sigma=1.0):
    # Read the image
    image = cv2.imread(image_path)

    # Display the input and output images using Matplotlib
    plt.figure(figsize=(10, 5))

    # Input Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Input Image')
    plt.axis('off')

    # Apply Gaussian blur to the image
    blurred = cv2.GaussianBlur(image, (0, 0), sigma)

    # Calculate the sharpened image by subtracting the blurred image from the
original image
    sharpened_image = cv2.addWeighted(image, 1.5, blurred, -0.5, 0)

    # Output Image
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(sharpened_image, cv2.COLOR_BGR2RGB))
    plt.title('Sharpened Image')
    plt.axis('off')

    # Display the images
    plt.tight_layout()
    plt.show()

    # Save the output image
    cv2.imwrite(output_path, sharpened_image)
    return output_path
---------------------------saturation_correction.py---------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
def saturation_correction(image_path, output_path, alpha=1.0, beta=0):
    # Read the image
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Unable to read the image.")
        return

    # Convert the image to the HSV color space
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    if hsv_image is None:
        print("Error: Unable to convert the image to HSV color space.")
        return

    # Split the HSV image into its individual channels (Hue, Saturation, Value)
    h, s, v = cv2.split(hsv_image)

    # Convert the saturation channel to the appropriate data type (uint8)
    s = s.astype(np.uint8)

    # Perform saturation correction by applying an affine transformation to the
Saturation channel
    corrected_s = np.clip(alpha * s + beta, 0, 255).astype(np.uint8)

    # Merge the corrected Saturation channel with the original Hue and Value
channels
    corrected_hsv = cv2.merge([h, corrected_s, v])

    # Convert the corrected HSV image back to the BGR color space
    corrected_image = cv2.cvtColor(corrected_hsv, cv2.COLOR_HSV2BGR)

    # Display the input and output images using Matplotlib
    plt.figure(figsize=(10, 5))

    # Input Image
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Input Image')
    plt.axis('off')

    # Output Image
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(corrected_image, cv2.COLOR_BGR2RGB))
    plt.title('Corrected Image')
    plt.axis('off')

    # Display the images
    plt.tight_layout()
    plt.show()

    # Save the corrected image
    cv2.imwrite(output_path, corrected_image)
    return output_path
---------------------------gamma_correction.py---------------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt


def gamma_correction_and_display(image_path, gamma_value=1.0, output_path=""):
```

```
    try:
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    corrected_image = np.power(original_image / 255.0, gamma_value) * 255

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(corrected_image.astype(np.uint8),
cv2.COLOR_BGR2RGB))
    plt.title(f"Gamma-Corrected Image (Gamma = {gamma_value})")
    plt.axis("off")
    plt.show()
    # Save the corrected image
    cv2.imwrite(output_path, corrected_image)
    return output_path
---------------------------Brightness_correction.py----------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt


def adjust_and_show_brightness(image_path, brightness_factor, output_path=""):
    try:
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    corrected_image = cv2.convertScaleAbs(original_image,
alpha=brightness_factor, beta=0)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(corrected_image, cv2.COLOR_BGR2RGB))
    plt.title(f"Brightness Corrected Image (Factor = {brightness_factor})")
    plt.axis("off")
    plt.show()
    cv2.imwrite(output_path, corrected_image)
    return output_path
```

```
--------------------------------mean_filter.py-----------------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt

def mean_filter_and_display(image_path, kernel_size,output_path):
    try:
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    def mean_filter(image, kernel_size):
        return cv2.blur(image, (kernel_size, kernel_size))

    filtered_image = mean_filter(original_image, kernel_size)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(filtered_image, cv2.COLOR_BGR2RGB))
    plt.title(f"Mean Filtered Image (Kernel Size: {kernel_size})")
    plt.axis("off")
    plt.show()
    cv2.imwrite(output_path, filtered_image)
    return output_path
----------------------------------Median Filter.py-----------------------------
import cv2
import matplotlib.pyplot as plt


def median_filter_and_display(image_path, kernel_size, output_path):
    def median_filter(image, kernel_size):
        return cv2.medianBlur(image, kernel_size)

    try:
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    filtered_image = median_filter(original_image, kernel_size)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")
```

```
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(filtered_image, cv2.COLOR_BGR2RGB))
    plt.title(f"Median Filtered Image (Kernel Size: {kernel_size})")
    plt.axis("off")
    plt.savefig(output_path)
    plt.show()
    return output_path
------------------------Bradley Local Threshold.py----------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt


def bradley_local_thresholding(image_path, output_path, window_size=30,
threshold=10):
    try:
        gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if gray_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    # Calculate the integral image
    integral_image = cv2.integral(gray_image)
    height, width = gray_image.shape

    # Apply the Bradley Local Thresholding algorithm
    thresholded_image = np.zeros((height, width), dtype=np.uint8)
    for y in range(height):
        for x in range(width):
            x1, y1, x2, y2 = max(0, x - window_size // 2), max(0, y -
window_size // 2), min(width - 1,

x + window_size // 2), min(
                height - 1, y + window_size // 2)
            area = (x2 - x1) * (y2 - y1)
            threshold_sum = integral_image[y2, x2] - integral_image[y1, x2] -
integral_image[y2, x1] + integral_image[
                y1, x1]
            if gray_image[y, x] * area <= threshold_sum * (100 - threshold) /
100:
                thresholded_image[y, x] = 0
            else:
                thresholded_image[y, x] = 255

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(gray_image, cmap='gray')
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(thresholded_image, cmap='gray')
    plt.title(f"Bradley Local Thresholding (Window Size: {window_size},
Threshold: {threshold}%)")
    plt.axis("off")
```

```
    plt.savefig(output_path)
    plt.show()
    return output_path


---------------------------------bilateral_smoothing.py------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt

def bilateral_smooth(image_path, output_path, diameter, sigma_color,
sigma_space):
    try:
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    # Apply bilateral smoothing using the cv2.bilateralFilter function
    smoothed_image = cv2.bilateralFilter(original_image, diameter, sigma_color,
sigma_space)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(smoothed_image, cv2.COLOR_BGR2RGB))
    plt.title("Bilateral Smoothed Image")
    plt.axis("off")

    plt.savefig(output_path)
    plt.show()
    return output_path

---------------------------------Conservative_Smooth.py------------------------
import cv2
import numpy as np
import matplotlib.pyplot as plt

def conservative_smooth(image_path, output_path, h, search_window,
patch_window):
    try:
        original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    # Apply the Non-Local Means filter for conservative smoothing
    smoothed_image = cv2.fastNlMeansDenoising(original_image, None, h,
search_window, patch_window)
```

```
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(original_image, cmap='gray')
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(smoothed_image, cmap='gray')
    plt.title("Conservative Smoothed Image")
    plt.axis("off")

    plt.savefig(output_path)
    plt.show()
    return output_path
---------------------------------Histogram_Equalize.py------------------------
import cv2
import matplotlib.pyplot as plt


def histogram_equalize_and_display(image_path,output_path):
    try:
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid 
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    # Convert the image to grayscale if it's in color
    if len(original_image.shape) == 3:
        original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)

    equalized_image = cv2.equalizeHist(original_image)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(equalized_image, cmap='gray')
    plt.title("Histogram Equalized Image")
    plt.savefig(output_path)
    plt.axis("off")
    plt.show()
    return output_path
---------------------------------Morphological_Opening.py----------------------

import cv2
import numpy as np
import matplotlib.pyplot as plt

def morphological_opening_and_display(image_path, kernel_size=(5, 
5),output_path=""):
    try:
```

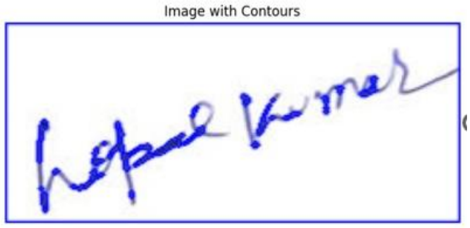
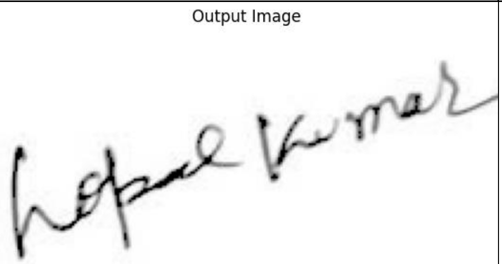
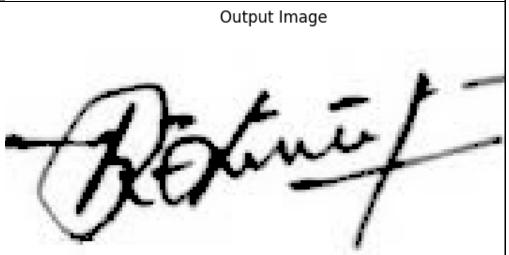
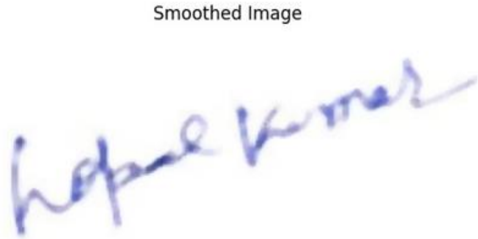
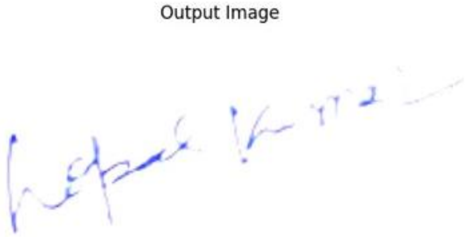
```
        # Read the image
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    # Convert the image to grayscale
    gray_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
    # Define the kernel for morphological operations
    kernel = np.ones(kernel_size, np.uint8)
    # Perform morphological opening
    opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(opened_image, cmap='gray')
    plt.title("Morphological Opening")
    plt.axis('off')
    plt.savefig(output_path)
    plt.show()
    return output_path
---------------------------------Morphological_closing.py----------------------


import cv2
import numpy as np
import matplotlib.pyplot as plt


def morphological_closing_and_display(image_path, kernel_size, output_path):
    try:
        # Read the image
        original_image = cv2.imread(image_path)
        if original_image is None:
            raise FileNotFoundError("Image not found. Please provide a valid
image path.")
    except Exception as e:
        print(f"Error: {e}")
        return

    # Convert the image to grayscale
    gray_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
    # Create a kernel for morphological operations
    kernel = np.ones(kernel_size, np.uint8)
    # Perform morphological closing
    closed_image = cv2.morphologyEx(gray_image, cv2.MORPH_CLOSE, kernel)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')
    plt.subplot(1, 2, 2)
```

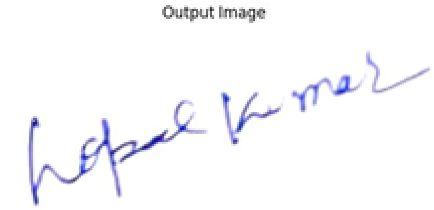
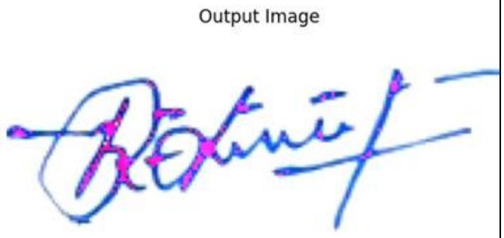
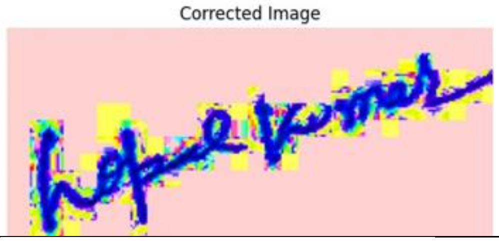
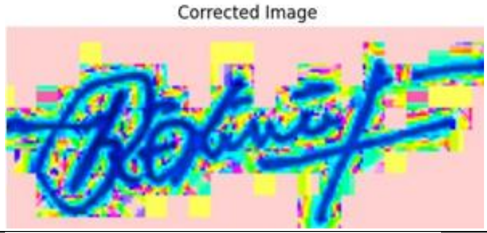
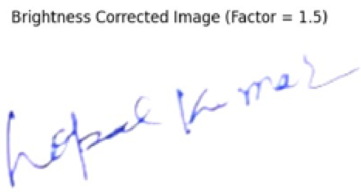
```
    plt.imshow(closed_image, cmap='gray')
    plt.title("Morphological Closing")
    plt.axis('off')
    plt.savefig(output_path)
    plt.show()
    return output_path
```

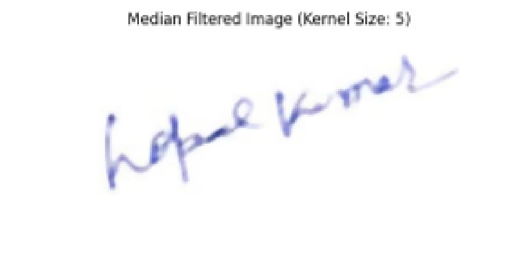
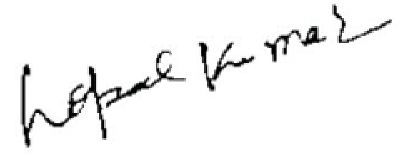
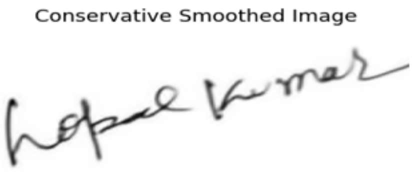
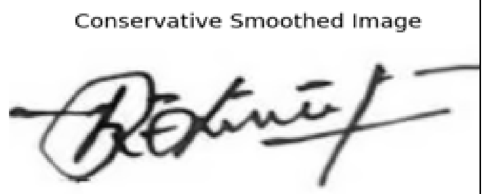
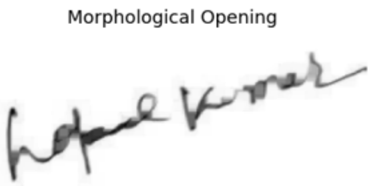
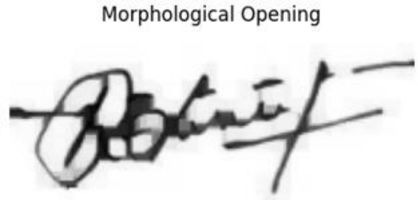
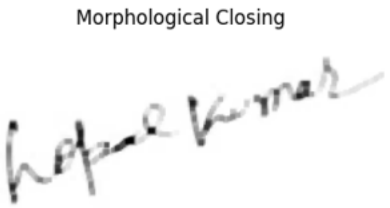
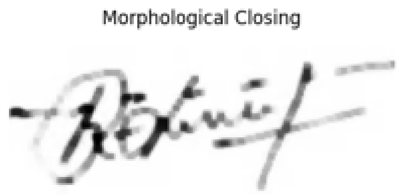
----------------------------------PROGRAM TO DETECT HORIZONTAL AND VERTICAL-------------------------------- LINES

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def preprocess_image(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, threshold = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)
    return threshold

def extract_vertical_lines(image):
    vertical_kernel = np.ones((6, 1), dtype=np.uint8)
    vertical_lines = cv2.morphologyEx(image, cv2.MORPH_OPEN, vertical_kernel)
    return vertical_lines

def extract_horizontal_lines(image):
    horizontal_kernel = np.ones((1, 6), dtype=np.uint8)
    horizontal_lines = cv2.morphologyEx(image, cv2.MORPH_OPEN,
horizontal_kernel)
    return horizontal_lines

if __name__ == "__main__":
    print("Vertical and Horizontal Line Detection Between Alphabets")
    print("Please provide the path to the signature image.")

    image_path = input("Enter the path to the signature image: ")

    preprocessed_image = preprocess_image(image_path)

    vertical_lines = extract_vertical_lines(preprocessed_image)
    horizontal_lines = extract_horizontal_lines(preprocessed_image)

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 3, 1)
    plt.imshow(preprocessed_image, cmap='gray')
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(vertical_lines, cmap='gray')
    plt.title("Detected Vertical Lines")
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(horizontal_lines, cmap='gray')
    plt.title("Detected Horizontal Lines")
    plt.axis("off")

    plt.tight_layout()
    plt.show()
```

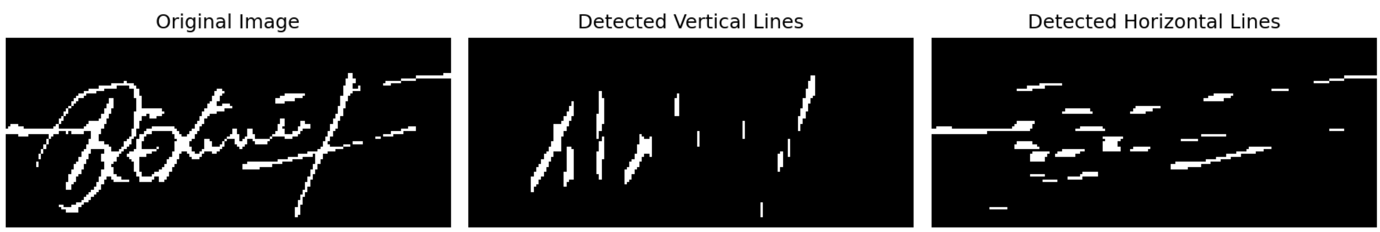
9. RESULT AND OUTPUT

Input Image		
Image Filter Operations	**Output1**	**Output 2**
Contour Display Threshold value: 100 Enter the contour color: 255,0,0 Enter the contour thickness: 1		
Stroke Thinning Threshold value: 100 Min neighbors: 2 Max neighbors: 8 Max transitions: 4		
Stroke Thickening Kernel size: 3 Iteration Number: 2		
Adaptive Smoothing kernel size: 5		
Contrast Correction Contrast factor: 2 Brightness factor: 1		

Contrast Stretching Minimum value for channel 0: 20 Maximum value for channel 0: 200 Minimum value for channel 1: 20 Maximum value for channel 1: 200 Minimum value for channel 2: 20 Maximum value for channel 2: 200		
Gaussian Sharpen **sigma** = 4 (standard deviation of the Gaussian Distribution).		
Saturation Correction **Saturation scale factor:** 60 **Saturation shift factor:** 45		
Gamma Correction **Gamma value:** 3.5		
Brightness Correction **Brightness factor:** 1.5		
mean_filter **Kernel Size:** 3		

Median Filter **Kernel Size:** 3		
Bradley Local Threshold **window size:** 25 **threshold percentage**: 30		
bilateral smoothing **Diameter**: 9 **Sigma color**: 75 **Sigma space:** 75		
Conservative-Smooth **Enter the h value:** 10 **Enter the search window size:** 7 **Enter the patch window size**: 5		
Histogram Equalize		
Morphological Opening **kernel width**: 5 **kernel height**: 5		
Morphological closing **kernel width**: 3 **kernel height**: 3		

HORIZONTAL AND VERTICAL LINES OUTPUT



10.CONCLUSION

The array of image filter operations is a versatile toolkit crucial in forensic document examination. These techniques are essential for overcoming challenges like degradation, noise, and visibility issues in document analysis. They enhance the accuracy and depth of forensic investigations.

To address document degradation, operations like Stroke Thinning, Stroke Thickening, Adaptive Smoothing, and Gaussian Sharpen are employed, allowing experts to reverse or mitigate adverse effects.

For noise reduction, Adaptive Smoothing, Mean Filter, Median Filter, and Bilateral Smoothing strategically filter out unwanted artifacts, ensuring legibility.

Enhancement techniques like Contrast Correction, Contrast Stretching, Saturation Correction, and Gamma Correction improve visibility and enable precise extraction of information.

Segmentation tools, such as Contour Display, Bradley Local Threshold, Morphological Opening, and Closing, aid in isolating and analyzing specific components, facilitating anomaly detection.

These operations optimize forensic document examination, vital for verification and authentication processes in legal proceedings.

Through this project I learned about OpenCV module and various processes which can modify an image.

11.BENEFITS –

- Efficient Forensic Analysis: The proposed semi-automated solution streamlines the process of analyzing questioned documents and verifying handwritten content, reducing the time and effort required for manual examination.

- Enhanced Accuracy: By utilizing advanced techniques like image processing filters and feature extraction, the solution improves the accuracy of forensic document examination and signature verification, minimizing errors and false identifications.

- Comprehensive Functionality: The solution covers various aspects of forensic document examination, including noise removal, segmentation, and writer/signature verification, providing a comprehensive solution for different aspects of document analysis.

12.REFERENCES

1. Lee, S., Cha, S.H., Srihari, S.N.: Combining macro and micro features for writer identification. In: Kantor, P.B., Kanungo, T., Zhou, J. (eds.) Document Recognition and Retrieval IX. International Society for Optics and Photonics, vol. 4670, pp. 155-166, San Jose, California, United States (2001)
2. CEDAR-FOX homepage. https://cedar.buffalo.edu/NIJ/objectives.html
3. Kumar, A., Bhatia, K.: A survey on offline handwritten signature verification system using writer dependent and independent approaches. In: 2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA), pp. 1-6. IEEE, Bareilly, India (2016). A survey on offline handwritten signature verification system using writer dependent and independent approaches | IEEE Conference Publication | IEEE Xplore
4. Adak, C., Chaudhuri, B.B., Blumenstein, M.: An empirical study on writer identification and verification from intra-variable individual handwriting. IEEE Access 7, 24738-24758 (2019). https://doi.org/10.1109/ACCESS.2019.2899908