Sheet Counting Application Documentation

Introduction:

The Sheet Counting Application is designed to automate the process of counting individual sheets within a stack using image processing techniques. This tool is intended for use in manufacturing environments where accurate and efficient counting of sheet stacks is crucial. By leveraging computer vision and image processing algorithms, the application offers a reliable method to count sheets with minimal manual intervention.

Objective:

The primary objective of this application is to accurately count the number of sheets in a stack by analysing images taken of the stack. The application is built to handle varying sheet thicknesses and different stack configurations, ensuring robustness and adaptability in various scenarios.

Overall Approach:

The approach to solving the sheet counting problem involves several key steps:

- **Image Pre-processing:** The image is processed to isolate the sheet stack from the background. This involves colour-based masking to identify the relevant areas of the image.
- **Image Enhancement:** Techniques such as Adaptive Histogram Equalization and Gaussian Blur are applied to improve the

contrast and clarity of the image, making edge detection more effective.

- **Sheet Counting:** Edge detection and intensity analysis are used to count the number of sheets based on vertical intensity projections of the processed image.
- **Web Application:** The Flask web framework is used to create an interface where users can upload images, view results, and interact with the application.

The application provides a user-friendly interface for uploading images and receiving real-time feedback on the number of sheets detected. This documentation outlines the approach, tools used, challenges faced, and potential improvements for the application.

Frameworks/Libraries/Tools:

- **Flask**
    - **Purpose:** A lightweight web framework used to develop the web application for image upload and display results. It handles HTTP requests and serves the HTML interface.
- **OpenCV (Open Source Computer Vision Library)**
    - **Purpose:** A comprehensive library for computer vision tasks, including image processing and manipulation. Used for tasks such as colour conversion, image cropping, and edge detection.
- **NumPy**
    - **Purpose:** A fundamental library for numerical computing in Python. It is used for handling arrays and performing operations like vertical projection and peak detection in images.

- **Scipy**
    - **Purpose:** A library for scientific and technical computing. It provides the find_peaks function used for detecting peaks in the intensity projection of the image, which helps in counting the sheets.
- **Matplotlib**
    - **Purpose:** A plotting library for creating static, animated, and interactive visualizations in Python. It is used to visualize images and processing results for debugging and analysis.

Challenges and Solutions:

Challenge 1: Inaccurate Sheet Counting

Description: The initial version of the application struggled with accurately counting sheets, particularly when dealing with varying thicknesses and overlapping sheets.

Solution: Improved image pre-processing and enhancement techniques were applied. Techniques such as Adaptive Histogram Equalization and Gaussian Blur were used to enhance image contrast and clarity. Additionally, the peak detection parameters in the vertical intensity projection were fine-tuned to better handle different sheet thicknesses.

Challenge 2: Isolating the Sheet Stack from the Background

Description: Isolating the sheet stack from the background was challenging due to varying colours and backgrounds in the images.

Solution: Colour-based masking was employed to create masks for different colour ranges (brown and cream) and isolate the relevant areas. Cropping was then applied to focus on the sheet stack. Further, background pixels were set to white to improve processing accuracy.

Challenge 3: Processing Time and Performance

Description: Processing time was an issue with larger images or complex stacks, impacting the performance of the application.

Solution: Optimized image processing steps by reducing the complexity of operations and using efficient algorithms. Applied Gaussian Blur and edge detection judiciously to balance accuracy and performance.

Future Scope:

Enhanced Accuracy

- **Deep Learning Models:** Incorporate deep learning techniques to improve accuracy in detecting and counting sheets, especially in complex or cluttered images. Convolutional Neural Networks (CNNs) can be explored for feature extraction and classification.

User Interface Improvements

- **Real-Time Feedback:** Implement real-time feedback and interactive results on the web interface, allowing users to adjust parameters and see immediate effects.

Database Integration

- **Data Tracking:** Integrate with a database to store and analyse historical data, trends, and performance metrics. This can help in understanding usage patterns and improving the system over time.

Handling Diverse Image Conditions

- **Multi-Condition Support:** Develop techniques to handle different lighting conditions, backgrounds, and sheet types. This could involve adaptive methods or multi-modal image processing techniques.

Scalability

- **Scalable Architecture:** Ensure the application is scalable to handle larger volumes of images or integrate with other systems in a manufacturing environment.