Sheet Counter Web Application

Harshavardhan Manavalan | **Sheet Counter Web Application** | [Date]

Project Report: Sheet Counter Web Application

1. Overall Approach

The Sheet Counter Application is designed to automate the process of counting and analyzing sheets from images. The project employs computer vision techniques to efficiently detect and count sheets, catering to industries where precise sheet counting is essential.

Approach Summary:

1. **Image Preprocessing:**
    - **Convert images to a suitable format.**
    - **Apply preprocessing techniques to enhance quality for better analysis.**
2. **Sheet Detection:**
    - **Utilize image processing algorithms to identify sheets in images.**
3. **Counting Mechanism:**
    - **Implement a counting algorithm to accurately determine the number of sheets.**
4. **Data Storage and Output:**
    - **Save the results and provide a summary of findings.**

Recording Demonstration:

For a visual walkthrough of the application, view the demonstration recording

https://www.loom.com/share/85952c3ca68048afb9eb8eec7aa694d2?sid=1360e464-cd58-49d7-a7cf-f893e98b45bf

2. Frameworks, Libraries, and Tools

1. Python

- **Purpose: The primary programming language used for developing the image processing and counting algorithms.**

2. OpenCV

- **Purpose: A library for computer vision tasks used for image processing operations such as filtering, thresholding, and contour detection.**

3. NumPy

- **Purpose: Supports large, multi-dimensional arrays and matrices, facilitating numerical operations essential for image processing.**

4. Matplotlib

- **Purpose: Used for visualizing results, including displaying images and summaries of counts.**

5. Git

- **Purpose: Version control system for managing code changes and collaboration.**

6. GitHub

- **Purpose: Platform for hosting the code repository, tracking issues, and collaborative development.**

3. Challenges and Solutions

1. Image Quality and Variability

- **Challenge: Variability in image quality and lighting conditions affected sheet detection accuracy.**
- **Solution: Implemented preprocessing techniques like histogram equalization and adaptive thresholding to standardize image quality.**

2. Accurate Sheet Detection

- **Challenge: Differentiating sheets from other elements in the image.**
- **Solution: Utilized contour detection and morphological operations to accurately isolate and count sheets.**

3. Integration and Deployment

- **Challenge: Ensuring a reliable deployment pipeline and integrating various tools.**
- **Solution: Configured Git for version control and GitHub for repository management, streamlining the deployment process and documentation.**

4. Future Scope

1. Enhanced Accuracy

- **Improvement: Integrate machine learning models to improve detection accuracy under varied conditions.**

2. Multiformat Support

- **Improvement: Extend functionality to support additional image formats and sources, such as PDFs and real-time camera feeds.**

3. User Interface

- **Improvement: Develop a graphical user interface (GUI) to enhance user experience, allowing easy image uploads and result viewing.**

4. Advanced Analytics

- **Improvement: Incorporate advanced analytics to provide insights and trends based on sheet counts for business decision-making.**

5. Script Content

Below is the Python script used for processing and counting sheets:

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

def preprocess_image(image_path):
    # Load image
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # Apply GaussianBlur to smooth image
    blurred = cv2.GaussianBlur(img, (5, 5), 0)
    # Apply adaptive thresholding
    _, thresh = cv2.threshold(blurred, 127, 255, cv2.THRESH_BINARY_INV)
    return thresh

def count_sheets(thresh_image):
    # Find contours
    contours, _ = cv2.findContours(thresh_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # Count number of contours
    num_sheets = len(contours)
    return num_sheets

def main():
    image_path = 'path/to/image.jpg'
    processed_image = preprocess_image(image_path)
    num_sheets = count_sheets(processed_image)
    print(f"Number of sheets detected: {num_sheets}")
    # Display the processed image with contours
    img = cv2.imread(image_path)
    contours, _ = cv2.findContours(processed_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"Detected Sheets: {num_sheets}")
    plt.show()

if __name__ == "__main__":
    main()
```

6. HTML Code

Here is a basic HTML code snippet that could be used to display results in a web interface:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sheet Counter Application</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        .container {
            max-width: 800px;
            margin: 0 auto;
        }
        h1 {
            text-align: center;
        }
```

```
        .results {
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Sheet Counter Application</h1>
        <div class="results">
            <h2>Results</h2>
            <p><strong>Number of sheets detected:</strong> <span
id="sheetCount">Loading...</span></p>
            <img src="path/to/result_image.jpg" alt="Processed Image" style="max-width: 100%;
height: auto;">
        </div>
    </div>
    <script>
        // This script would be used to dynamically update results if connected to a backend
        document.getElementById('sheetCount').innerText = 'Placeholder for dynamic data';
    </script>
</body>
</html>
```