Project Report
Summer Internship 2020

Script Counting Application

Interns

Sai Jatin K	PES2UG19CS353
Sneha Jain A	PES2201800030
S Krishna	PES1201801930

Mentors

Sameer Dhole	01FB15EEC197
Vishwas Rajashekar	PES1201700704

PES Innovation Lab
PES University
100 Feet Ring Road,
BSK III Stage,
Bangalore-560085

Abstract

The objective of the project is to develop a method for automatic counting of examination answer booklets through image processing techniques. This helps reduce delay and inaccuracies caused by manual counting. Concepts of pixel profile analysis, curve smoothing and binarization have been employed in the image processing algorithm developed for counting. The methodology has been implemented in Java and integrated with an Android application for easy use. The application has been tested with over 175 images and the error rate is at **1.94 %**.

Contents

List of Figures

Chapter 1

Introduction

The project is concerned with the domains of image processing and Android application development.The goal is to automate counting of examination answer-scripts for usage in educational institutions. Automatic counting helps save time and minimise inaccuracies caused by manual counting.There can be different ways of accomplishing this. We have tackled the problem with image processing.The method designed to count answer booklets makes use of concepts pixel intensity profile analysis.To minimise errors, image binarization and curve smoothening techniques have been employed.The method has been implemented in Java and integrated with an Android application. OpenCV SDK is made use of for implementation of image processing aspects of the algorithm

The Android application development aspect of the project is concerned with development of an interface that allows the user to capture an image with answer booklets or import one from the device. The image thus captured/selected is taken as input to the algorithm.

A Flask server was originally set up to link the application with the Python implementation of the method. The server responds to requests from the application and executes the Python implementation backend and returns results to the application.The Flask framework has now been replaced with native-to-Android implementation of the algorithm for enabling onboard processing.

Figure 1.1: A sample input image with 34 answer-scripts

Figure 1.2: ScriptCounter application

1.1 Problem Statement

Development of an Android application for counting examination answer booklets through image processing techniques.

The following sections of the report elaborate on the different aspects of the project. Chapter 2 describes our survey of literature and concepts derived from the same. Chapter 3 explains the methodology/technique developed to count answer-scripts. It also includes an a section that talks about the different image processing experiments that were tried out in the context of the problem statement. Chapter 3 is concerned with the hardware and software components relevant to the project. The penultimate chapter describes the results of testing and inferences from the same. The report is concluded with final remarks and scope for future work.

Chapter 2

Literature Survey/Related Work

This chapter deals with the literature associated with the problem statement and the concepts made use of in solving the problem. Automatic counting of sheets through image processing is not an extensively studied problem.[1] is a related publication that describes usage of Gabor filters for paper counting. Gabor filter is applied on the image and the intensity profiles are analysed. There are several papers introduce and describe the image processing techniques that can be used to approach the problem. Concepts of Canny edge extraction, probabilistic Hough transforms, binarization and intensity profile smoothening are applied in the project.

Canny edge detector [2] is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.Figure 2.1 is an example of its application to an image. The algorithm involves four stages:noise reduction through Gaussian filter,finding intensity gradient,non maximum suppression,hysteresis thresholding. Canny edge detection is used in this project in the segmentation in conjunction with probabilistic Hough transforms.

A Hough Transform is considered probabilistic [3] if it uses random sampling of the edge points. These algorithms can be divided based on how they map image space to parameter space. One of the easiest probabilistic methods is to choose m edge points from the set M edge points. The complexity of the voting stage reduces from O(M.N) to O(m.N). This works because a

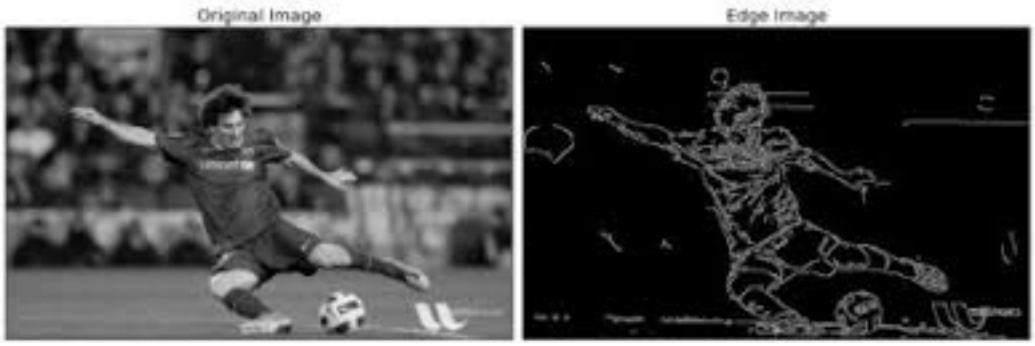


Figure 2.1: Canny Edge Detection applied on an image

random subset of M will fairly represent the edge points and the surrounding noise and distortion. Smaller value of m will result in fast computation but lower accuracy. So the value of m should be appropriately chosen with respect to M.

Another relevant concept is curve smoothening which is carried out with an N-point moving average filter. The moving average filter [4] is a simple Low Pass FIR (Finite Impulse Response) filter commonly used for smoothing an array of sampled data/signal. It takes samples of input at a time and takes the average of those -samples and produces a single output point. It is a very simple LPF (Low Pass Filter) structure that comes handy for scientists and engineers to filter unwanted noisy components from the intended data. As the filter length increases (the parameter) the smoothness of the output increases, whereas the sharp transitions in the data are made increasingly blunt. This implies that this filter has excellent time domain response but a poor frequency response

The concept of adaptive thresholding is explained in [5].It is made use of in binarization. The idea behind adaptive thresholding is that the threshold value for a pixel are determined automatically based on pixel intensities in the area surrounding the pixel. This can tackle inconsistencies in lighting within the input image. The automatically determined threshold can be the mean of the neighbourhood area intensities minus a constant or a Gaussian-weighted sum of the neighbourhood values minus the constant C.

Chapter 3

Main Body

This chapter deals with important aspects of the project such as experimentation, approach to the problem statement and the related hardware/software components.

3.1 Experimentation

The project-work involved a few experiments with image processing techniques for checking their effectiveness with respect to sample images and constructing a suitable solution to the problem of automatic counting.

The first set of exercises was concerned with detection of edges in sample images. Results of Canny [2] and Sobel edge extraction [6] on image 1.1 is shown in Figure 3.1 and 3.2. Though edge detection is not directly used in counting, it can be used along with Hough transformations for segmentation.

Secondly, Hough transforms were experimented with. Sample images were subjected to Hough transforms following Canny edge extraction. Different parameter values were used leading to varied results. A sample result is depicted in Figure 3.3.Probabilistic Hough transforms are preferred to regular Hough transforms because probabilistic Hough transforms work well even for slanted views of booklet stack as evident from figures 3.5 and 3.6

When a given image is converted to black and white there is a clear dis-

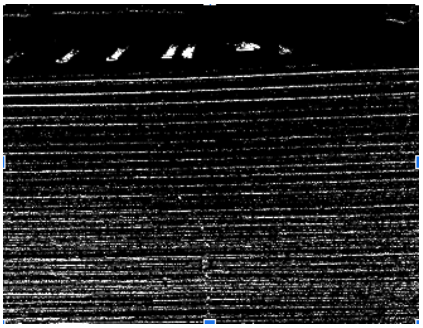


Figure 3.1: Canny Edge Detection

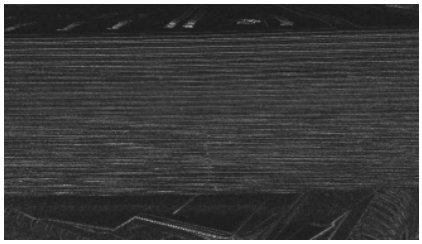


Figure 3.2: Sobel Edge Detection

Figure 3.3: Sample Hough Transform Result

tinction visible suggesting that the edges of the booklets are white and gap in between them black with a little spillover on either side.If this was true,we could change the color of all the pixels on any horizontal line that had at least one white pixel to be entirely white. This would create a clear distinction between black and white lines and thus would allow us to easily count the number of booklets.However, upon experimentation it is found that each and every single horizontal line has at least one white and one black pixel. Thus leading to the failure of this method.

Pixel intensity profile analysis led to the fact that gaps between two booklets were indicated by **dips** in the pixel intensities. This could be used in the counting process. But the intensity profiles obtained from the image cannot

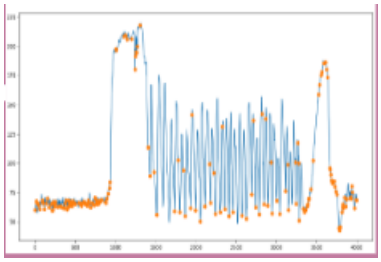


Figure 3.4: Intensity profile along a vertical section of the sample image with orange points indicating dips

Figure 3.5: Regular Hough transforms applied on input image

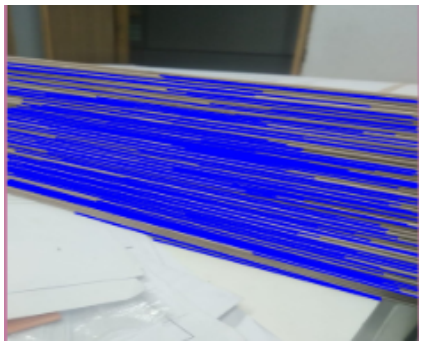


Figure 3.6: Probabilistic Hough transforms applied on input image

be used directly for dip-counting as there can be dips that do not pertain to gaps between booklets. This can be observed in Figure 3.4

These false positives can be eliminated by binarization of image before dip counting. The profile obtained is rectangular and makes the process of counting dips easier. Binarization is accomplished by image thresholding. Adaptive thresholding [5] is favoured over simple thresholding [7] as the former can work even with inconsistent lighting conditions. This can be observed in Figures 3.7 and 3.8.

An alternative technique to handle "fake dips" is smoothing the profile. This can be done with an N-point average moving filter. Dips are counted after smoothing.

These experiments helped in the process of developing a reasonably fool-

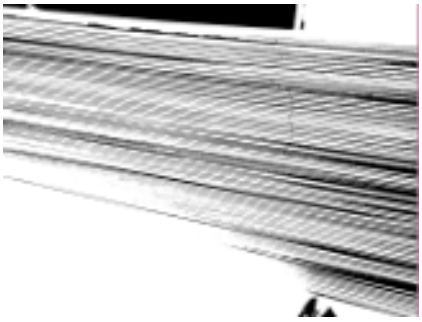


Figure 3.7: Simple Thresholding leading to white patches on the left

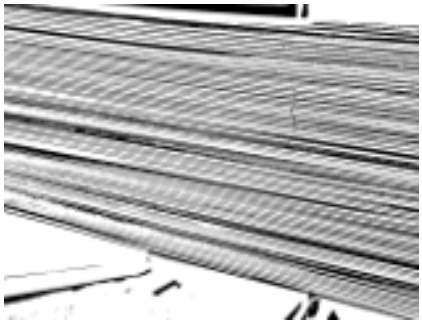


Figure 3.8: Adaptive thresholding producing better results

proof technique that can be applied for a variety of conditions.

3.2 Methodology

In this section, we shall present the technique for counting answer booklets in the input image. There are two steps involved in this process:
1.Image Segmentation
2.Dip Counting
Let us discuss the two steps individually.

3.2.1 Image Segmentation - Hough Transform Based and Manual cropping

The input image is segmented for background removal. Only that portion of the image with the booklets is retained. Segmentation is necessary because presence of background may result in inaccurate counts. A Hough-Transform based method can be used for this purpose. It works as follows:
Firstly, the edges of the image are extracted through Canny edge detection [2]. Probabilistic Hough transforms are applied using the edges that have been extracted. A set of lines in the region of booklets is obtained. The image is segmented/cropped based on the coordinates of the top-most and bottom-most lines. This process is depicted in Figure 3.9

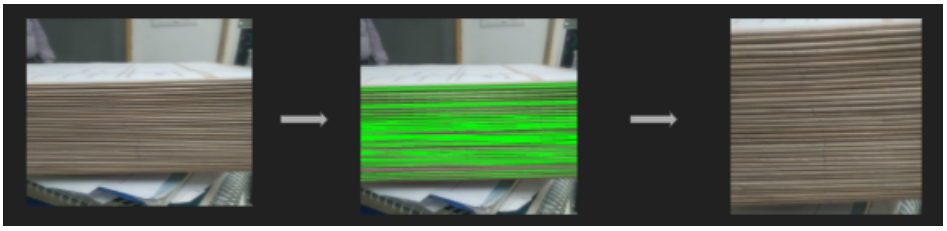


Figure 3.9: Image Segmentation based on Probabilistic Hough Transforms

However, this method of segmentation has been found to be unreliable for a few views of the booklet stack and hence been removed from the methodology. The user of the application is expected to crop the input image to remove background. For this purpose, a cropping feature has been included in the application. It can be seen in fig. 3.10

Figure 3.10: Cropper Activity in ScriptCounter

3.2.2 Dip Counting

The segmented image is converted to a gray-scale image and is subjected to pixel intensity analysis. It has been observed that gaps between booklets are marked by a **dip** in the intensity. This concept is applied in counting

answer booklets. There can however be false positives i.e dips that do not correspond to "inter-booklet gaps". To eliminate "fake dips", we can either binarize the image or smooth the intensity profile before counting dips. This leads to this equation:

$$Numberofbooklets = Numberofdips + 1 \tag{3.1}$$

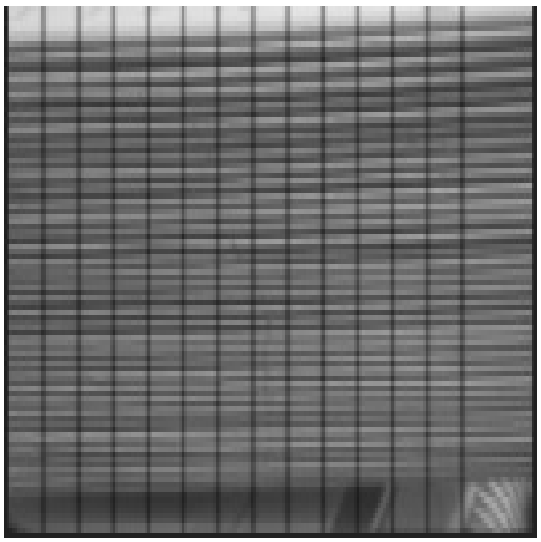


Figure 3.11: Vertical Sections along which dips are counted

Dip counting is done along multiple vertical sections (as indicated by lines in Figure 3.11) of the segmented image for enhanced accuracy. Both binarization and curve smoothening methods are independently used and the results are merged. The **most frequently occurring count** is returned as final result.

The flow of this algorithm is shown in Figure 3.12

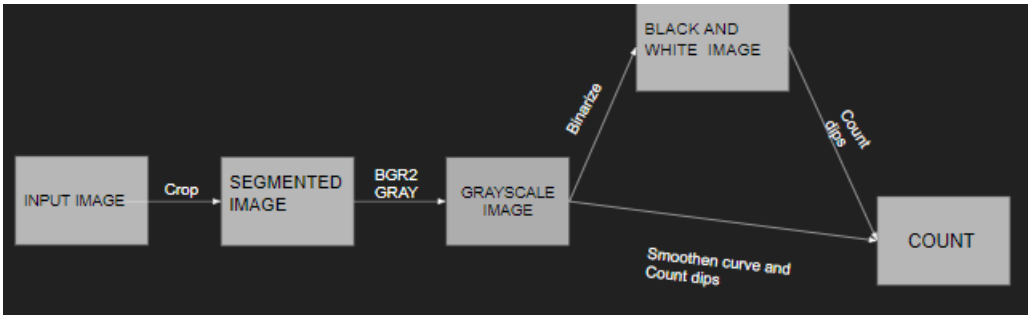


Figure 3.12: Algorithm flow

Binarization

The segmented image in grayscale is binarized i.e converted to a black and white image. The resulting image has only two intensities - 0 and 255. A dip is marked by a shift from 255 to 0. Binarization results in a clear demarcation between booklets(in white) and gaps between booklets(in black). This can be observed in Figure 3.13. The intensity profile of a binarized image is rectangular(as seen in Figure 3.14).
Binarization is performed by adaptive thresholding [5] of gray-scale segmented image. The threshold for a given pixel is the Gaussian-weighted sum of the neighbourhood values minus a constant C.

Figure 3.13: Binarized Image

Figure 3.14: Intensity profile of the binarized image

Curve Smoothing

Alternatively, intensity profile of segmented grayscale image can be smoothened by application of an N-point moving average filter. Smoothening helps re-

move false positives / "fake dips" from consideration. A window size of 5 is used for the smoothening filter. A smoothened curve is shown in Fig 3.15.

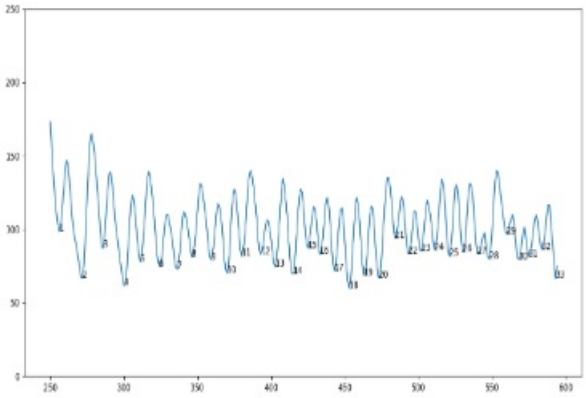


Figure 3.15: Smoothened Intensity Profile

3.3 Hardware and Software Components

This section is meant for describing the software/programming aspects of the project. It can be divided into three sub-sections: Java implementation with OpenCV and Android User Interface. These components have been shown in Figure 3.16

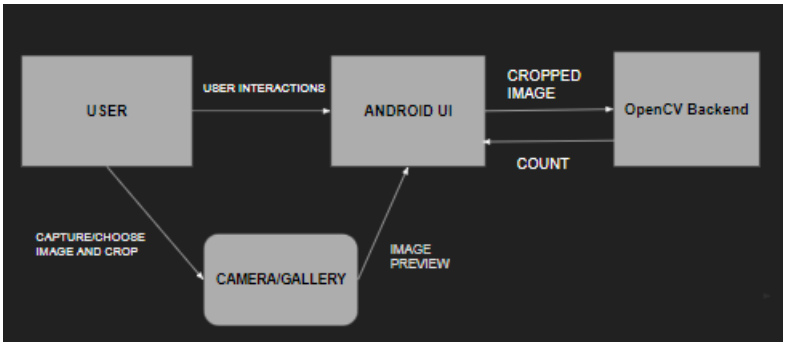


Figure 3.16: Software Components

3.3.1 Java Implementation of the algorithm

The technique described in section 3.2 has been implemented in Android Java. Image processing techniques are performed by invoking methods of OpenCV 3.4.1 (Java Wrapper for Android). The dip techniques of binarization and curve smoothing are implemented in separate classes and both work on the input image represented as a **Mat**.

3.3.2 Android User Interface

An Android application **ScriptCounter** is developed using Android Studio IDE.The app accepts the image from the user in two ways - one by importing images from one of the storage repositories including Gallery,Google Drive and Photos and the other by taking an image through the phone camera. This image can be cropped by the user using an interactive tool. Finally this serves as the input image for the image processing algorithm. The user flow can be seen in fig 3.18.

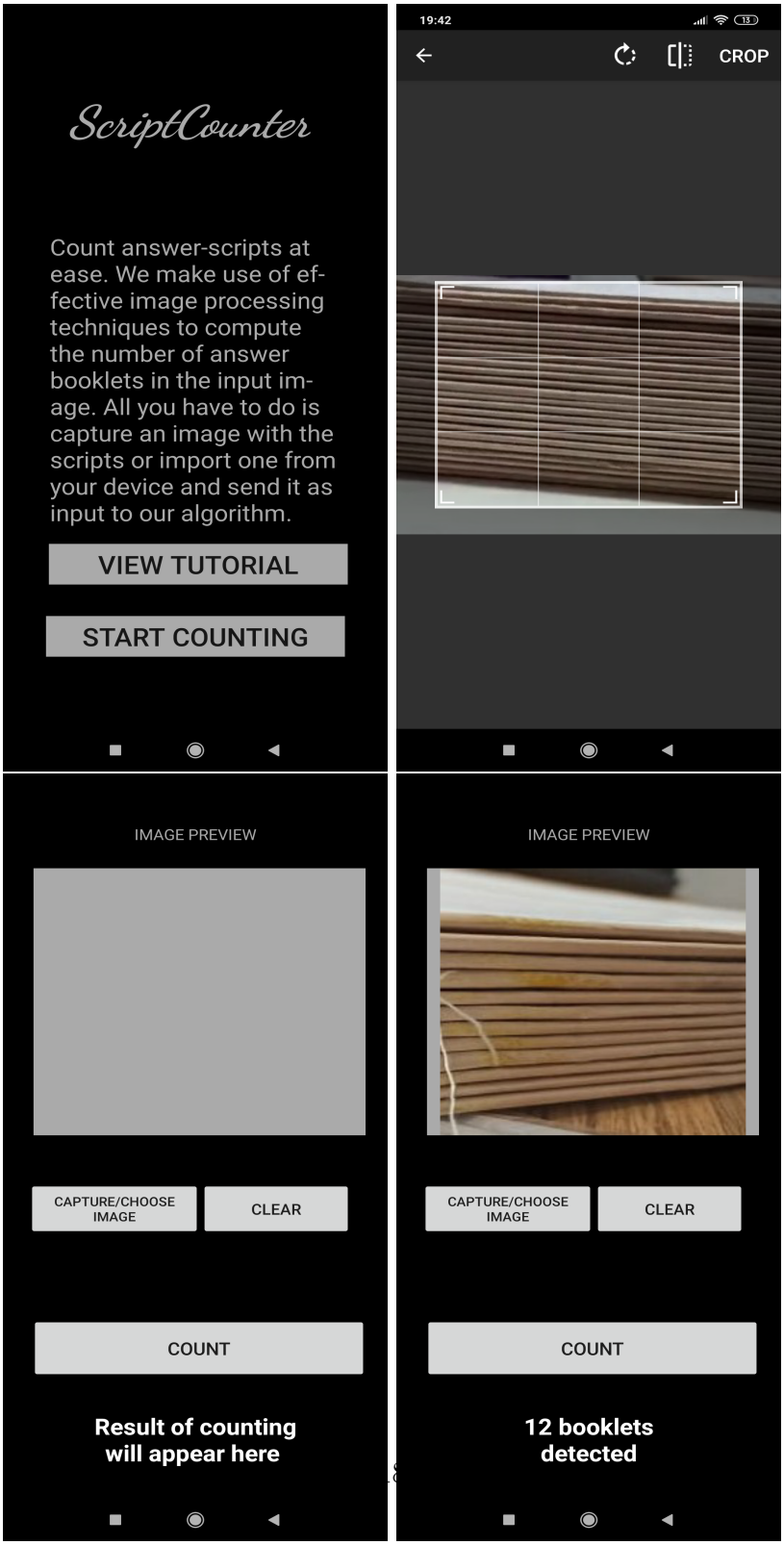


Figure 3.17: Screenshots from the application

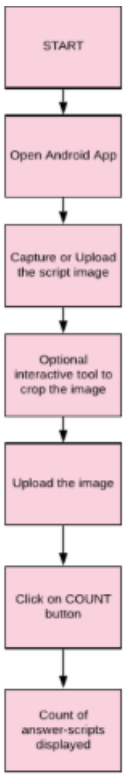
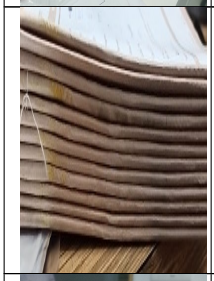


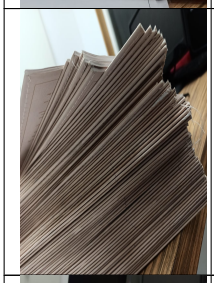
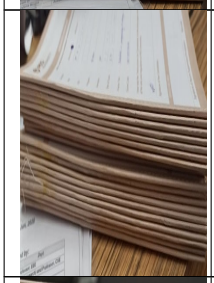
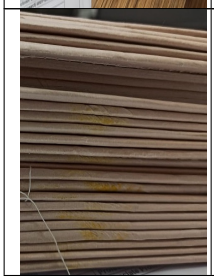
Figure 3.18: Application User Flow

Chapter 4

Results and Discussion

This chapter is concerned with the results of testing the application and inferences thereof. Table 4.1 shows the predicted counts for a few images.

Image	Expected Count	Obtained Count	Accuracy Level	Remark
	46	46	100	Slightly slanted stack
	12	12	100	Slanted stack
	34	34	100	-

	48	48	100	-
	61	61	100	Stack seen partially
	40	40	100	-
	18	18	100	Slightly slanted stack
	25	24	96	Stack slighlty misaligned

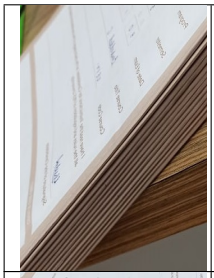
	12	13	91.67	Stack highly slanted
	66	55	83.33	Camera far from stack

Table 4.1: Sample Results

4.1 Testing Results

The methodology was tested with images of different angles and views. Two rounds of testing were conducted.

4.1.1 Testing - 1

The first round of testing involved 75 images. The Python implementation of the methodology was iteratively executed on the dataset. The input image, cropped image, dip diagram, binarization output, expected and obtained counts have been recorded in the report of the testing (link to report provided in Appendix A). The overall error rate for this set of pictures was found to be **5.2%**. The relatively high rate of error is due to the fact that dataset comprises of a wide variety of images including very tilted and misaligned stacks for which the algorithm is expected to under-perform.

4.1.2 Testing - 2

In this round, 175 images were tested on the ScriptCounter application. Impractical views were ignored so as to understand how the app functions for

typical angles and views of the booklet stack. Results with screenshots have been recorded in the report for this round of testing. (link in Appendix). Overall error rate for the 175 images is at about **1.94%** (slightly inflated on consideration of errors in manual counting). This figure provides a better picture of the effectiveness of the methodology for it is based on more practical and acceptable types of pictures.

4.2 General Sources of Errors

In this section, we shall list down the commonly encountered problematic cases leading to errors with an example for each.

4.2.1 Misaligned Stack

This refers to stacks of booklets where individual booklets do not coincide with one another. As a result, the booklet closer to the camera poses a region of shadow on the ones farther from the camera (i.e the ones pushed inside). Such booklets go un-detected. An example can be seen in the image 4.1

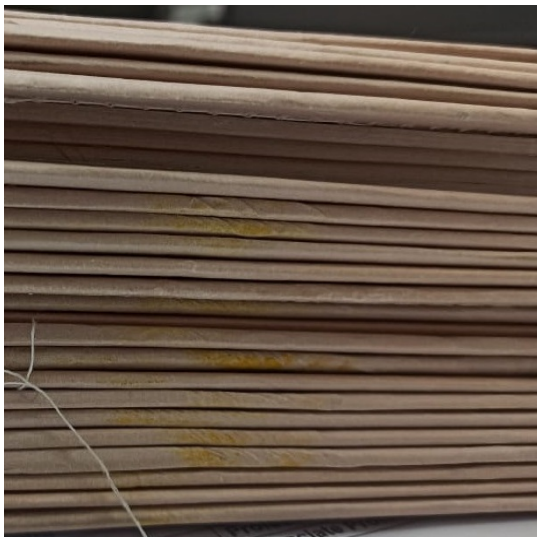


Figure 4.1: Misaligned Stack

4.2.2 Tilted View

The algorithm developed works best for stacks that are not very tilted. Very tilted booklet stacks like in 4.2 tend to mislead in the sense that the mode of the dip counts get deviated from the correct count. So such images are prone to errors. Rotation is an option to consider for future work.

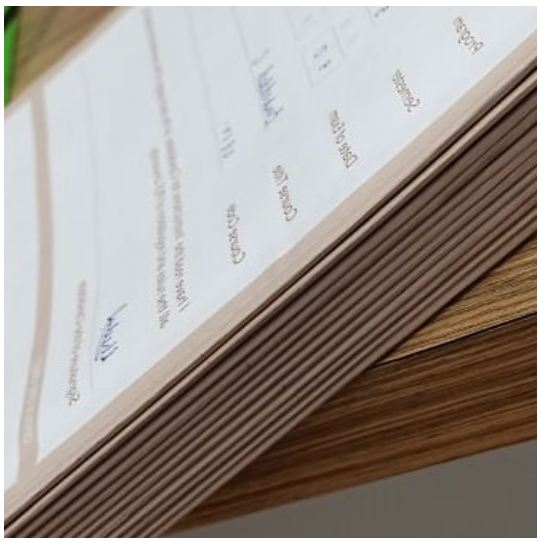


Figure 4.2: Tilted View

4.2.3 Projection distance problem in large stacks

It has been observed that in images with a large number of booklets (say ¿80), the booklets in the bottom appear very thin. As a result, they may not be detected. This problem seems to alleviate on the usage of flash in capturing the picture. Flash also helps reduce false positives within a booklet.

Figure 4.3: Images captured without flash and with flash

4.3 Constraints on the input image

Based on the results of testing, we can list down the conditions for the input image for the app to give out an accurate count :

1. Booklet stack is fairly horizontal i.e angle of tilt less than 30 deg
2. Booklets are unicolored
3. Large stacks are captured with flash
4. Booklets are well arranged

Chapter 5

Conclusions and Future Work

In conclusion, it can be said that the attempt to automate counting of booklets has been quite successful with encouraging results. The application developed will pave way for more extensive testing in the future. Scope for future work includes testing on physical booklets captured by camera (instead of already taken images), exploration of some image enhancement techniques for images of large booklets taken without flash and improvement in accuracy for misaligned stacks.

Bibliography

[1] C. Jiqiu, L. Qirong, and L. Qianming, "Design and implementation of paper counting algorithm based on gabor filter," pp. 281–285, 12 2018.

[2] Wikipedia contributors, "Canny edge extractor." https://en.wikipedia.org/wiki/Canny$_e dge_d etector$.

[3] Jayrambhia, "Probabilistic hough transforms." https://jayrambhia.com/blog/probabilistic-hough-transform.

[4] Gaussian Waves, "Curve smoothening." https://www.gaussianwaves.com/.

[5] Tutorials Point, "Adaptive thresholding with opencv." https://www.tutorialspoint.com/opencv/opencv$_a daptive_t hreshold.htm : : text = Adaptive$2020.

[6] GeeksForGeeks, "Sobel edge extractor." https://www.geeksforgeeks.org/python-program-to-detect-the-edges-of-an-image-using-opencv-sobel-edge-detection/.

[7] OpenCV Docs, "Simple(global) thresholding." https://opencv-python-tutroals.readthedocs.io/en/latest/py$_t utorials/py_i mgproc/py_t hresholding/py_t hresholding.html$.

Chapter 6

Appendix A

6.1 Important Links

1. Research Papers: https://drive.google.com/drive/folders/1LxmpUL6CxY9eu1Biv0nlFGxuMRb4Uinu

2. GitHub Repo : https://github.com/SnehaJain0531/ScriptCounting

3.Report of Testing - 1 : https://drive.google.com/file/d/15GlFxQeskXX2QiHJqIMQtN983gW2SSK8/view?usp=sharing

4.Report of Testing - 2 : https://drive.google.com/file/d/12nszrTlBAjX94gP1hnwY-45cybGN2sfj/view?usp=sharing

5. Latest Presentation : https://docs.google.com/presentation/d/1AJILIO-IpRTBbGOdtzcRL7ybLlrY5QUTqAqh3f4Segw/edit?usp=sharing