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 an Android application for automatic counting of examination answer booklets through image processing techniques. Concepts of Canny edge extraction, Probabilistic Hough transforms, binarization and intensity profile smoothening have been used.The input image is segmented to remove background and is then subjected to pixel intensity analysis.The algorithm has been implemented in Python. A Flask server has been set up for integrating the Python implementation with the Android application.Results of testing with sample images have been satisfactory.

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 of edge detection and pixel intensity profile analysis.To minimise errors, image binarization and curve smoothening techniques have been employed.The method has been implemented in Python using the OpenCV module.

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 has been set up to link the application with the Python script. The server responds to requests from the application and executes the Python implementation backend and returns results to the application.The Flask framework is temporarily used to pave way for testing the technique designed for counting booklets. Upon completion of testing, the Python program will be ported to Java for onboard processing in Android.

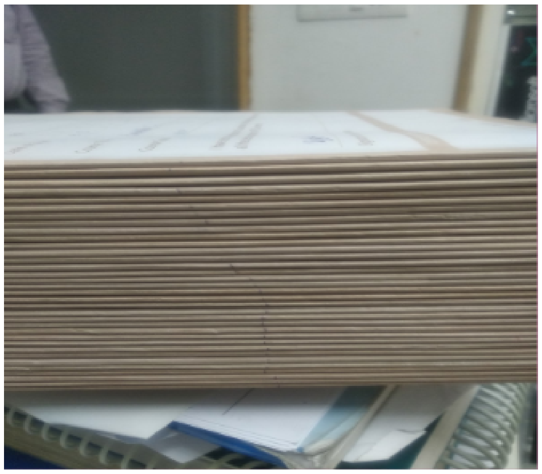


Figure 1.1: A sample input image with 34 answer-scripts

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 progress achieved with the objective so far. Chapter 2 described 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

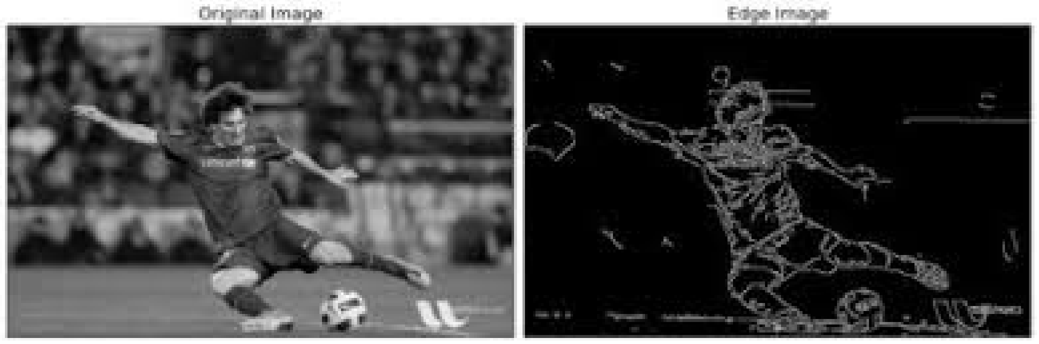


Figure 2.1: Canny Edge Detection applied on an image

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-

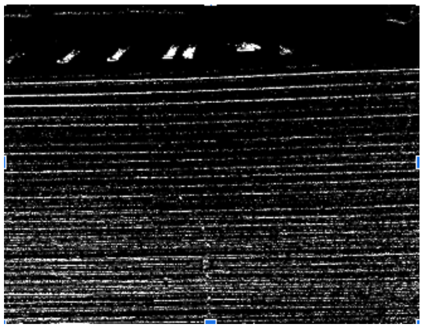


Figure 3.1: Canny Edge Detection

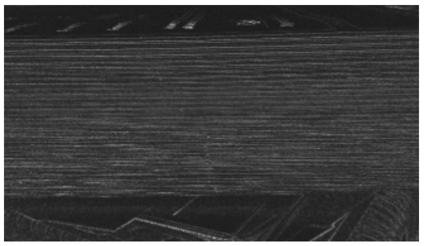


Figure 3.2: Sobel Edge Detection

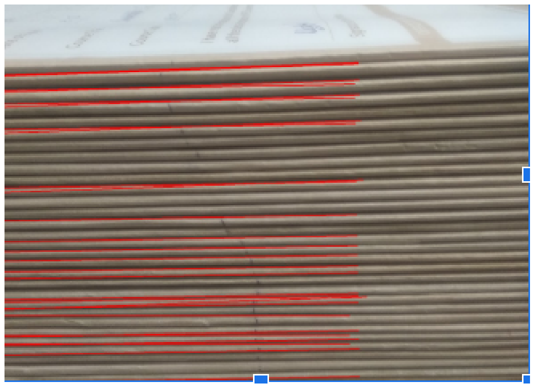


Figure 3.3: Sample Hough Transform Result

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

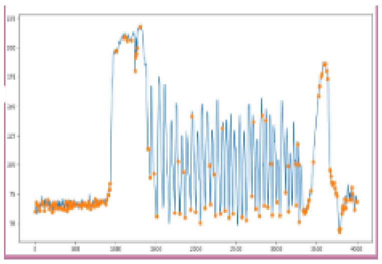


Figure 3.4: Intensity profile along a vertical section of the sample image with orange points indicating dips

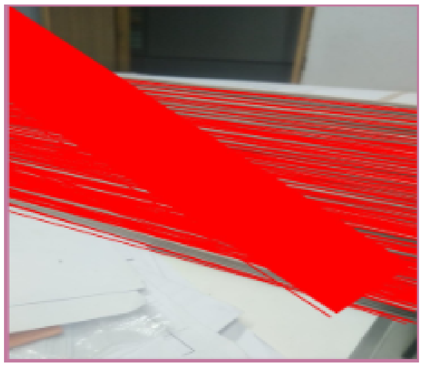


Figure 3.5: Regular Hough transforms applied on input image

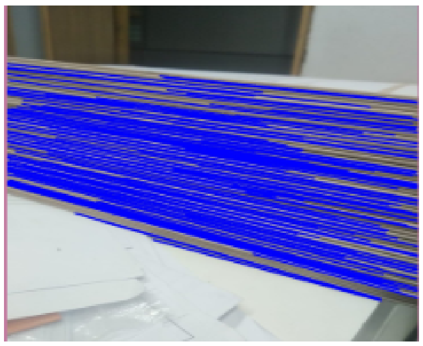


Figure 3.6: Probabilistic Hough transforms applied on input image

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 **??**.

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

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

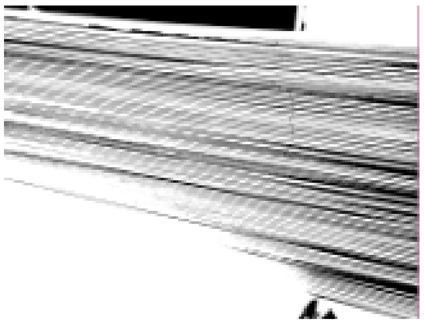


Figure 3.7: Simple Thresholding leading to white patches on the left

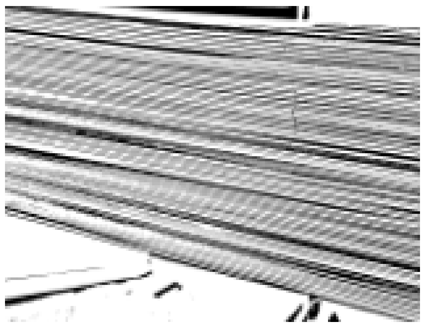


Figure 3.8: Adaptive thresholding producing better results

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

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. 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

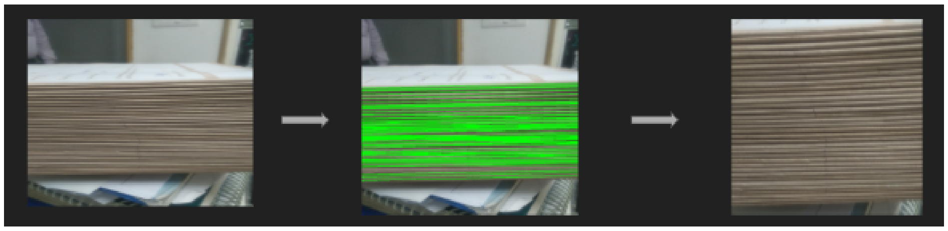


Figure 3.9: Image Segmentation based on Probabilistic Hough Transforms

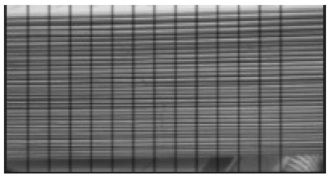


Figure 3.10: Vertical Sections along which dips are counted

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 smoothen the intensity profile before counting dips. This leads to this equation:

$$Number of booklets = Number of dips + 1 \tag{3.1}$$

Dip counting is done along multiple vertical sections (as indicated by lines in Figure 3.10) 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 3.11

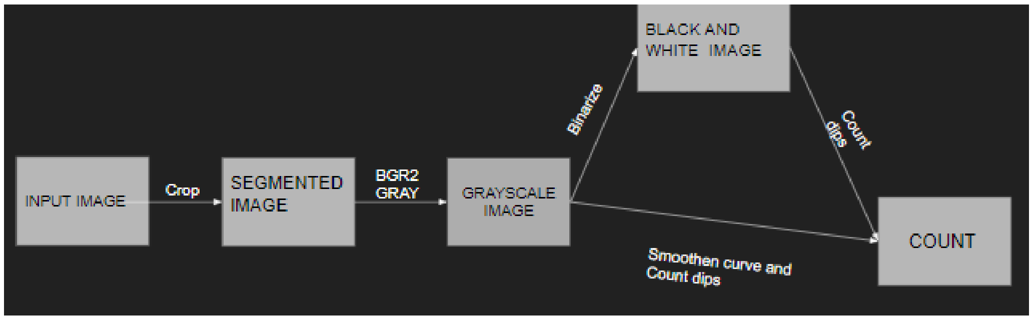


Figure 3.11: Algorithm flow

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 3.12. The intensity profile of a binarized image is rectangular(as seen in Figure 3.13).
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.

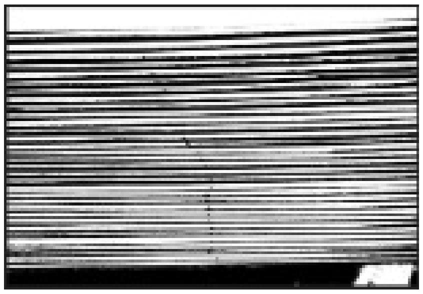


Figure 3.12: Binarized Image

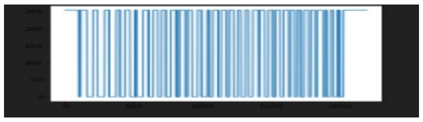


Figure 3.13: Intensity profile of the binarized image

Curve Smoothening

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

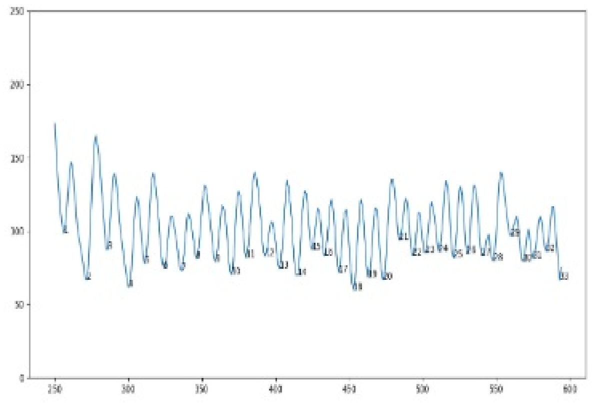


Figure 3.14: Smoothened Intensity Profile

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 3.14.

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: Python implementation, Android User Interface and Flask Server. These components have been shown in Figure 3.15

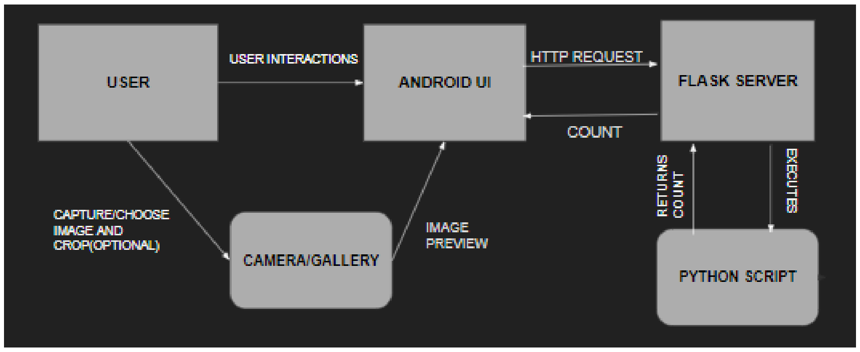


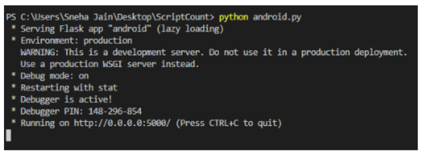
Figure 3.15: Software Components

3.3.1 Python Implementation

The technique described in section 3.2 has been implemented in Python 3.7. Image processing techniques are performed by invoking methods of OpenCV 3.4.1 (Python Wrapper). The script applies the algorithm on a given image and returns the booklet count.It is executed by the Flask server when the Android application makes an HTTP request.

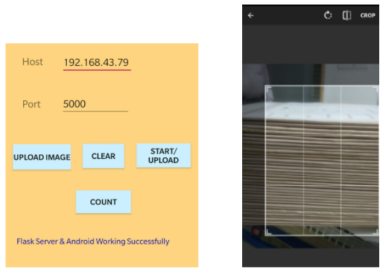
3.3.2 Android User Interface

An Android application has been developed using Android Studio IDE.The app accepts the user image in two ways. One by importing images from the gallery 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 provided to the server using requests.For running the Python program, a HTTP request using *okhttp* is made to the flask server that is set up on a local machine. The number of booklets detected is received as a response of the server, that is displayed in the app. The user-flow is indicated in the Figure 3.18 Screenshots from the interface can be seen in Fig. 3.17



```
PS C:\Users\Sneha Jain\Desktop\ScriptCount> python android.py
 * Serving Flask app "android" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 148-296-854
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figure 3.16: Flask Server



Host 192.168.43.79

Port 5000

Flask Server & Android Working Successfully

Figure 3.17: Screenshots of Application

3.3.3 Flask Server

A Flask server has been deployed on a local machine. It can execute the image processing script for the image of the stacked-up answer scripts, uploaded by the user through the Android UI. By running the count algorithm on the

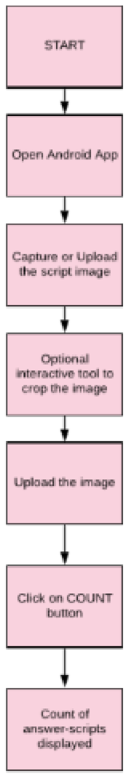

START
Open Android App
Capture or Upload the script image
Optional interactive tool to crop the image
Upload the image
Click on COUNT button
Count of answer-scripts displayed

Figure 3.18: Application User Flow

server, the number of booklets is computed, which is returned for display in the Android application. A screenshot of the terminal showing the running server can be seen in Fig. 3.16 Deployment of Flask server is temporary and only for development and testing purposes. The Python implementation will be ported to Java by means of OpenCV SDK for Android.

Chapter 4

Results and Discussion

This chapter deals with results obtained on testing the application with a few sample images. Table 4.1 shows results for four images along with the expected count.

The first image in Table 4.1 is already cropped. The number of booklets in it is correctly predicted to be 27. The second one has a background that is has to be removed through segmentation. Here again, the prediction is accurate. The third image shows a slanted view of the booklet-stack making it harder to detect booklets. The prediction is off by 1. The inaccuracy is probably because of under-cropping due to slanting. The fourth picture contains 80 booklets. Only 76 are detected. This is because of the combined effect of slanting and irregular arrangement of booklets that makes it difficult to discern even for a human eye in zoomed view.

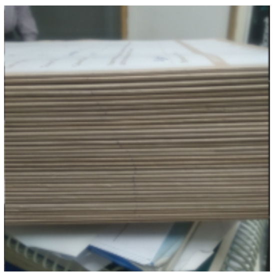


Figure 4.1: Expected View

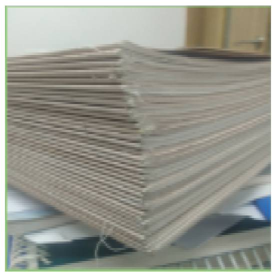


Figure 4.2: Misleading View

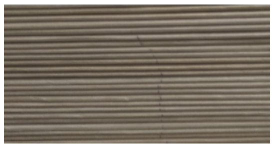
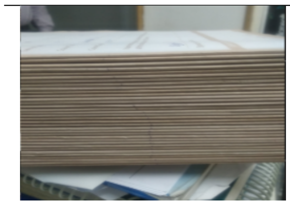
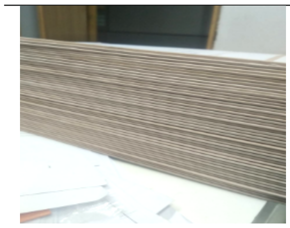
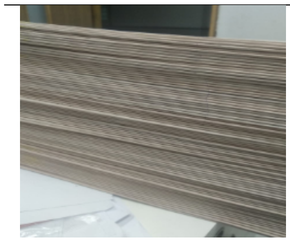
Image	Actual Count	Predicted Count
	27	27
	34	34
	46	47
	80	76

Table 4.1: A few sample results along with expected counts

4.1 Limitations

It can be seen from the table 4.1 that a few inaccuracies are prevalent in the technique. In general, they can be attributed to cropping errors and false positives. It also has to be noted that the technique works only for the front-view of the booklet stack(i.e edges of the booklets facing the camera). Top or lateral views do not lead to accurate results.Expected and misleading views can be seen in Figures . These limitations should be overcome on further testing and analysis.

Chapter 5

Conclusions and Future Work

In conclusion, it can be stated despite inaccuracies, the attempt to automate counting of booklets has been reasonably successful with encouraging results. The application developed will pave way for extensive testing in the future. Scope for future work includes analysing results by additional testing on more images followed by refinement of methodology for improving accuracy and porting of Python script OpenCV SDK for onboard processing on Android, making it suitable for deployment.

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. Test Images : https://drive.google.com/drive/folders
/1Mdp3E038ELqdBE3Y2WvoFCqdGeO8TuVQ

3. Program Files : https://drive.google.com/drive/folders
/1odilBC1fCE2qzZfN9UiiVOZkG05whxTO

4. Reports : https://drive.google.com/drive/folder0
s/1MgP8MDY$_ctB3eF5kbAVey6hfKfpaO4ZO$

$5.LatestPresentation : https : //docs.google.com/presentation/d/$
$1km9KymoL5IKuBizVIvvL06UBPzeCOD3B93mMvRkeU-U/editslide =$
$id.g81f27b2eed_{04}$