3. A Brief Introduction To The Work And The Proposed *cGrab-Cut*

This chapter briefly introduces the work that has been carried out. Next, it discusses the dataset that has been created in this work for the testing purpose and also introduces the environment under which the implementation and testing have been carried out. Lastly, it introduces the *cGrab-Cut*, a proposed preprocessing algorithm, for the background removal for Android-based devices.

3.1 A BRIEF ABOUT THE PROPOSED WORK

To achieve the goal and as stated in 1.4.3, the overall work has been divided into four phases, named, preprocessing, feature detection, classification, and text-to-speech conversion. The following figure shows the outline of the work.

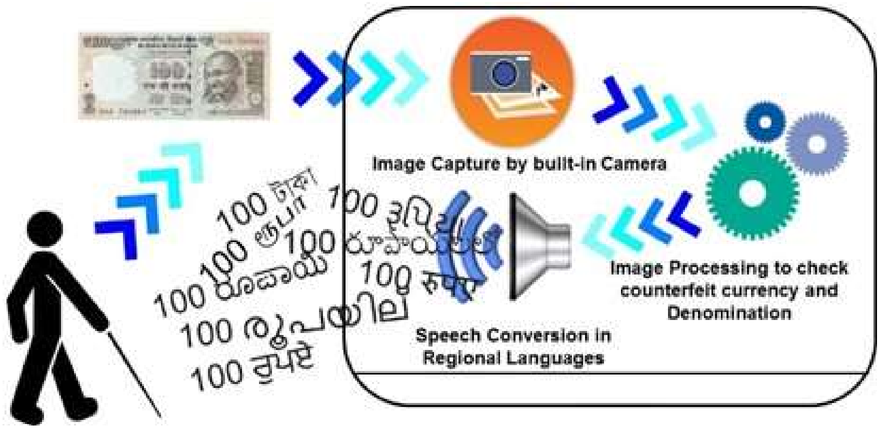
Image Capture by built-in Camera
Image Processing to check counterfeit currency and Denomination
Speech Conversion in Regional Languages
100 টাকা
100 రూపాయలు
100 ரூபாய்
100 ರೂಪಾಯಿ
100 രൂപ
100 रुपया
100 ਰੁਪਏ

Figure 3.1. An overall view of the proposed system

Here, the fourth phase, the Text-to-Speech conversion is simply a technology which is being used to convert the output into speech so that the blind people can listen to it. The text-to-speech conversion has been carried out using Google's API where research point of view there is no contribution. Research point of view, preprocessing, feature detection and classification are the major phases on which the work has been carried out. Firstly, a compromised GrabCut, *cGrab-Cut*, has been developed to reduce the time consumption of GrabCut for Android-based devices. Apart from *cGrab-Cut*, two hybrid feature detectors, the *HORB* – A Histogram based ORB and the *ACORB* – Ant Colony Optimization based ORB have been created and tested. In addition to this, through this work, two different classifiers which are based on the *HORB* and the *ACORB* feature detectors and a bag of visual features: a three-stage hybrid classifier – *HORBoVF* and a two-stage hybrid classifier – *ACORBoVF* have also been proposed and tested. Finally, this research work ends up by implementing one more classifier, *Te₹₹ency*, which is based on CNN and TensorFlow. In any research work, testing is the most crucial phase to check the effectiveness of the proposed approach. The next section briefly tells about the dataset that has been created for the testing purpose of the proposed work and the development as well as the testing environment.

3.2 THE DATASET AND ENVIRONMENT

Since this work is intended for currency recognition, the dataset has been created with all kinds of existing Indian currency notes. The dataset contains the images captured with different orientations, lighting conditions, and image resolutions. For the accurate measurement and to check if the proposed approaches work for all type of images, the dataset has been created using two categories of images: fully visible **(F)** currencies and partially visible **(P)** currencies. The following table shows details about the dataset on which the testing has been carried out:

Denomination	**# of Train Images**		**# of Test Images**	
	Small Set	**Large Set**	**F***	**P****
5	400	1530	130	80
10	620	1570	226	334
20	580	1567	220	340
50	540	1565	228	331
100	660	1587	268	292
200	422	1408	139	52
500	400	1512	160	78
500_old	360	1530	165	327
1000	84	1176	83	354
2000	486	1597	200	96

Table 3.1 The dataset description

*F – Fully Visible Images

**P – Partially Visible Images

The development and testing of this work have been carried out in two different environments, an Android-based mobile device and a Laptop-PC for python based implementation. The following table shows the configuration of the development and testing environment:

Specifications	Mobile Device	Laptop-PC
Model	RedMi 1S	Dell INSPIRON N5110
Processor	Max 1.6 GHz Quad-Core	Intel® Core-i3-2310M CPU with 2.10 GHz
RAM	1 GB	6 GB
Operating System	Android Kitkat 4.4	Windows10 Enterprise
System Type	4.4.4KTU84P 3.4.0-g9ada2c2	64-bit Operating System with the x64-based processor

Table 3.2 The Development & Testing Environment

3.3 The *cGrab-Cut* – A COMPROMISED YET OPTIMIZED GRABCUT FOR ANDROID DEVICES

3.3.1 Introduction to the *cGrab-Cut*

The GrabCut is an iterative and interactive background removal algorithm based on the graph cut theory which was designed to solve the Min-Cut/Max-Flow problem and K-Gaussian model [30]. However, in [123], Liu et al. noted that the time complexity of the GrabCut is too much when there is a background clutter. They proposed a superpixel based GrabCut to improve the time performance of the algorithm. In this, they first extract the blocks of superpixels and then split the picture. They experimentally showed that the algorithm has improved in terms of time complexity. In another similar work, Suriya et al. [87] used the GrabCut for the foreground detection, i.e., segmentation. But the implementation of their algorithm in Java on Android device was not found as efficient as it is in C++. The steps of their algorithms are given below:

1. Initial resizing of the image
2. Dynamic calculation of *Threshold* value based on the resized image (number of rows and columns)
3. Initialize the *rectangle* and *counter*
4. Repeat masking and GrabCut iterations until the *counter* reaches to *Threshold*
5. Resize the image

The GrabCut implementation of Suriya et. al.

The problems which were found, through the experiments, with their algorithm are:

1. Twice resizing of the image.

2. Dynamic calculation of Threshold value based on the resized image. The higher the number of rows and columns in terms of pixels, the larger the threshold value would be.
3. Repeated masking and the GrabCut iterations which are dependent on the threshold value. If the threshold value is larger than the number of iterations will be large.
4. The larger size of the initial rectangle

These factors were consuming more time and memory both. Even if only one GrabCut execution with one iteration is considered, which is not in the case, the complexity of the algorithm becomes *O(t)*, where *t* is the threshold value calculated dynamically based on the size of the image. To address these issues, the attempts are made, experimentally to optimize the GrabCut in terms of time and memory both. Following are the steps of the algorithm:

```
1. Initialize the rectangle from center region of the image
2. Perform GrabCut on inputImage
3. Perform masking of inputImage
4. Convert inputImage into grayscale
5. Perform inverse binary thresholding on inputImage to create segmentedImage
6. Display segmentedImage
```

The cGrab-Cut algorithm

Here, instead of resizing the image, it starts with a fixed sized rectangle and applied the GrabCut with inverse binary thresholding and masking. The result which was obtained is remarkable. Due to the calling of the GrabCut and masking only once along with inverse binary thresholding, the time and memory consumption by the algorithm was considerably low. Considering only GrabCut execution, it leads to complexity of algorithm as *O(1)*. The time consumption has been reduced by 57% of the original time taken by the implementation of Suriya et. al. While testing, it was found that many times the output image the proposed algorithm implementation was not a perfect cut but the output can be used for further processing as majority of the background part gets removed by the *cGrab-Cut*. The meaning of a perfect cut and a rough cut is shown in the figure given below.

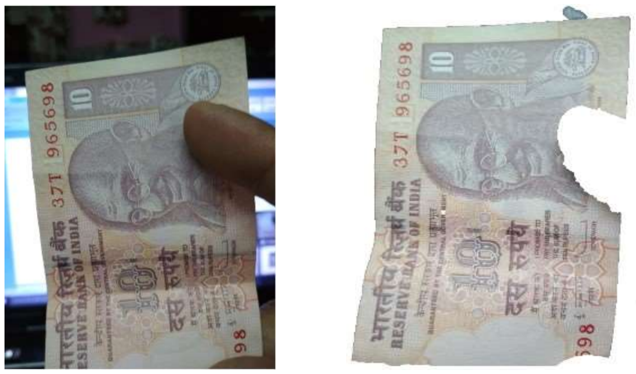

Figure 3.2. A Perfect GrabCut output for the image shown in rectangle

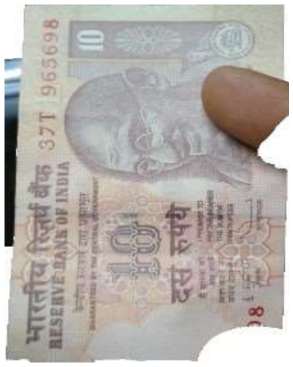

Figure 3.3. A *cGrab-Cut* Output

As it can be seen figure 3.3 that a perfect cut is not obtained yet the majority of the unnecessary part is being removed here. Hence, the output is acceptable for further processing. This is a trade-off that is being carried out with a little compromise in output to improve the performance in terms of time and memory, both. Due to this compromise, it has been named as *cGrab-Cut*.

3.3.2 Performance testing of the *cGrab-Cut*

The following graph shows the average time, per image, taken by the GrabCut, 18.983 seconds, and *cGrab-Cut*, 8.219 seconds, for a set of 100 images.

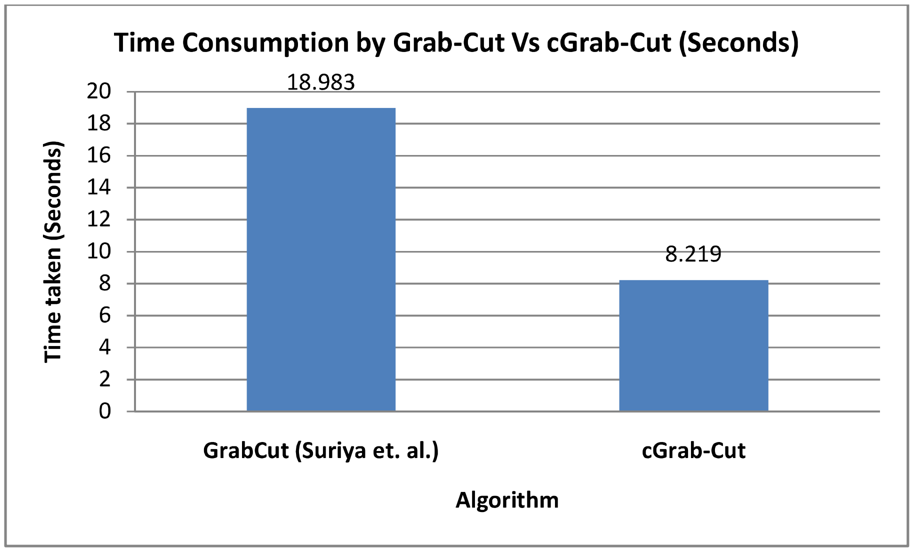

Figure 3.4. Time reduction in the GrabCut by the cGrab-Cut

As stated above, the *cGrab-Cut* is a trade-off based java implementation of the GrabCut wherein; some quality is being compromised to achieve the faster performance. There were around 100 images tested (100X2, for the GrabCut and *cGrab-Cut*) to check the performance in terms of time. It can be observed from the figure that the average time of the GrabCut implementation of Suriya et al. [87] had been reduced by 57% in *cGrab-Cut*.

SUMMARY

In the beginning, this chapter discussed the proposed approaches briefly. The second section showed the details about the testing dataset and the environment under which the development and testing are carried out. Finally, the third section explained the proposed *cGrab-Cut* algorithm for preprocessing in details along with the results. The next chapter discusses the two proposed feature detectors in detail with the result analysis.