CHAPTER 4

TEXTUAL IMAGE BINARIZATION USING GRAPH CUTS

4.1 INTRODUCTION

For last century, the importance of extraction and interpretation of textual information in digital images has grown tremendously. In most of the cases, the text hidden in the image or video clip provides significant piece of information. In last few decades, this has evolved as an important research area. To distinguish the textual area from other image pixels sometimes turn out to be challenging task. Optical Character Recognition (OCR), Content based retrieval of image, segmentation of textual image are some of the important computer vision problems related to images which has confined attention of researchers. Identification of number printed on the number plates of vehicles captured in the CCTV footage of a toll booth, identification of sign boards and hoardings in the scene of highways has also captured attentions of researchers as dealing with video clip and scene analysis. Digital note taking and document archiving are few of the important application of computer vision problems associated with textual image.

After the emergence of social networking sites and mobile phones with good digital cameras, image capturing, sharing and interpreting has become a task which is by and large performed by laymen. In day to day life, we come across many textual images which are not obtained through flatbed scanner but captured using digital camera or mobile phones. Due to portability and high speed mechanism of such devices, acquisition of images through them has become more common and has significantly affected human interaction. However, images produced in this manner don't have uniform brightness and hence traditional methods of document analysis fail miserably in case of images captured by mobile phones and digital camera because of perspective distortion and blur, non-uniform brightness and poor resolution. Most of the systems dealing with document analysis initially binarize the image in order to reduce the computational cost and complexity. Robustness of the binarization achieved at this stage is obviously important as a minor error in the binarization may greatly affect the inferences of document analysis.

The simplest and widely used binarization approach dealing with majority of computer vision problems related to text is global thresholding. If the quality of image is good and with uniform background brightness, pixel intensity called global thresholding is evaluated based on the average intensity of image pixels and the image is segmented in text (foreground) and background using this number. As all images do not have uniform brightness feature, local thresholding technique is employed for segmentation, which imitates the task of global thresholding on small pieces or blocks of images having somewhat uniform intensity throughout the block. Texture based techniques introduced in [8],[9] and methods based on connected components discussed in [45], [65], [67], [77] and [126] are the two main approaches with which textual image segmentation problems are being dealt with. Techniques mentioned in [9],[10] are time consuming and also depends on the font size of the text. For text with variety of font sizes or rather with scanned image of hand written text, this method is not so efficient. This method is highly efficient in detection of text from low resolution textual images. The methods based on the approach of connected component ([45], [65], [67], [77] and [126]) have impressive performance with reference to the time they consume.

In this chapter, the core work of the thesis (i.e. formulation of a mathematical model using graph cuts for a particular application and its implementation) is presented. In chapter 3, various graph cuts

models were discussed. We have formulated a mathematical model to address the task of binarization of textual images using graph cuts terminology. The application has its own significance as in Optical Character Recognition (OCR), binary image of good quality is the first pre-requisite. Most of the prevailing methods of binarization rely in one or other way on thresholding, whether it is local or global or on the approach of connected components.

4.2 DEFINITION OF THE PROBLEM TO BE ADDRESSED

We transform the problem of image binarization into an equivalent optimization problem and attempt to solve it through network flow terminology (i.e. Graph cuts). Max flow min- cut theorem given by Ford and Fulkerson plays a very crucial role in the model as in all graph cuts models.

A digital image of text document (i.e. scanned text) is composed of rectangular arrangement of pixels, each of which represents intensity. Binarization is a process of defining a function $X : V \to \{t', b\}$ (where V is the set of all pixels of the textual image) which reassigns a binary value t' (Text) or b (Background) to every pixel v of the image, subject to the given data. The choice of the function depends on two constraints. 1) The neighboring pixels should be assigned similar value by the function almost everywhere with the exception of the pixels representing boundary of the text. 2) The assignment should be made in light of the data given by scanned image. We have employed these constraints to construct the function. The first step includes construction of an objective function considering the constraint of the problem. Let V be a set of all pixels of the image and $\{g_v : v \in V\}$ be the set of grey values of pixels. The objective function $O : \Omega \to \mathbb{R}$ provides the measure of inappropriateness for every possible binarization function X from Ω and plays a crucial role in the selection of most suitable function

$$\text{i.e. } O(X) = \sum_{\substack{(u,v)\in N \\ u,v\in V}} kp_{uv} + \sum_{v\in V} |X_v - g_v| \tag{4.1}$$

Where, N denotes the set of all pairs of neighboring pixels of V, k is a nonzero constant, p_{uv} denotes the penalty imposed by the objective function O to X for assigning different binary values to the neighboring pixels and is defined as

$$p_{uv} = |X_u - X_v| \tag{4.2}$$

and X_u is defined as,

$$X_u = \begin{cases} 0, & \text{if } X(u) = t' \\ 255, & \text{if } X(u) = b \end{cases} \tag{4.3}$$

The objective function takes care of both the constraints imposed on the required binarization function X. Hence our reduced problem is to find a binarization function X that minimizes the objective function. This optimization problem is computationally expensive as the space all possible binarization functions has dimension $|V|$, which is in thousands. Making the situation worst, the function being non-convex may have many local minima. Our model uses the approach of network flow for the minimization of objective function. We construct a network flow using the objective

function and the scanned image. Minimum cut of the network flow plays a decisive role in the evaluation of the required binarization function.

Given a scanned textual image with pixel set *V*, our model considers its gray value set $\{g_v : v \in V\}$ as an input data. A network flow *G (V, E, W)* using the input data is constructed with vertex set *V* containing vertices v corresponding to each member v of pixel set *V* and a pair of distinguished vertices *s* and *t* for source and sink resp. Every such vertex has a pair of edges e_v^s and e_v^t joining it with source *s* and sink *t* resp. Their edge weights are $|X_v - g_v|$. For every pair of neighbors *u* and *v*, an edge e_{uv}^n with weight $k.p_{uv}$ is created in the network flow. The cut of the network flow is a minimum collection of its edges, whose removal from the network flow makes the terminal vertices *s* and *t* disconnected. A minimum cut of the network flow is a cut with minimum cost. The minimum cut of the network flow gives the partition of the vertices *v* into two sets *S* and *T*. This also gives the most suitable choice of binarization function for the image. All pixels *v* whose corresponding vertices are part of the set S in the induced graph should be assigned binary value t', whereas the remaining pixels of the image should be assigned value *b*. In the next session, we present the results showing that, the model gives the binarization function which minimizes the objective function.

4.3 THEORETICAL JUSTIFICATION OF THE MODEL

In this section, we will prove that, the model efficiently minimizes the objective function and produces the binarization of the image which has minimum value under objective function (4.1). This also proves the superiority of the model among prevailing binarization techniques in light of the objective function.

THEOREM 4.3.1

For any scanned textual image, the set Ω *of all corresponding binarization functions and the set C of all cuts on the network flow corresponding to the image are in one to one correspondence.*

Proof: Let $X : V \to \{t', b\}$ be a binarization function corresponding to the given scanned image. Then, there exists a cut C = *{S, T}* on the network flow corresponding to the image defined as follows:

S = $\{v \mid X_v = 0\}$ and T = $\{v \mid X_v = 255\}$

Note that, $X_v = 0$, when $X(v) = t'$ and $X_v = 255$ when $X(v) = b$.

This proves that, every binarization function gives rise to a cut on the network flow constructed for the textual image under consideration.

Conversely, let C = *{S, T}* on the network flow for the given textual image. Then, we can define *X* as follows:

$X(v) = t'$, if $v \in S$

$X(v) = b$, if $v \in T$

This proves the theorem.

THEOREM 4.3.2

The minimum cut on the network flow constructed for the textual image gives binarization which minimizes the objective function.

Proof: let X be the binarization corresponding to any cut C of the network flow. First, we show that, cost of the cut C is $O(X)$.

Note that, cost of any cut is sum of weights of the edges which are member of the cut set. In case of our network flow, there are two types of edges: (i) non-terminal edges e^n_{uv} joining neighboring vertices u and v of the network flow (ii) Terminal edges e^s_v and e^t_v connecting vertex v with the terminal vertices s and t respectively.

Thus, the cost of C is given by,

$$|C| = \sum_{\substack{(u,v)\in N \\ e^n_{uv}\in C}} |e^n_{uv}| + \sum_{e^s_u \in C} |e^s_u| + \sum_{e^t_u \in C} |e^t_u|. \qquad (4.4)$$

Claim 1: For every vertex v, exactly one of the edges e^s_v and e^t_v can be part of the cut C.

If both of the edges are part of the cut C, the cut C' obtained by removing one of the edges e^s_v and e^t_v is again a cut, which is contradiction with the fact that C is a cut, as no proper subset of a cut can be a cut. This proves that, at most one of the edges e^s_v and e^t_v can be in C.

If the cut C contains none of the edges e^s_v and e^t_v then, the terminals s and t stays connected even after removal of all edges of C, (because of the existing path s - e^s_v - v - e^t_v - t in $G \setminus C$) which contradicts with the fact that, C is a cut.

This proves claim 1.

As a result, every vertex v of the network contributes to exactly one of the last two terms of (4.4). Thus, (4.4) becomes,

$$|C| = \sum_{\substack{(u,v)\in N \\ e^n_{uv}\in C}} |e^n_{uv}| + \sum_{u\in V} \left\{ |e^s_u|, |e^t_u| \right\}$$

$$|C| = \sum_{\substack{(u,v)\in N \\ e^n_{uv}\in C}} |e^n_{uv}| + \sum_{u\in V} |X_u - g_u| \qquad (4.5)$$

As $|e^n_{uv}| = k.p_{uv}$, (4.5) becomes,

$$|C| = \sum_{\substack{(u,v)\in N \\ e^n_{uv}\in C}} k.p_{uv} + \sum_{u\in V} |X_u - g_u| \qquad (4.6)$$

Claim 2: For every pair of neighboring vertices *u* and *v*, the edge e^n_{uv} is part of *C* iff either e^s_u and e^t_v are both in *C* or e^t_u and e^s_v are both in *C*.

Assume that, e^s_u and e^t_v are in C but e^t_u and e^s_v are not. If possible, assume that, $e^n_{uv} \notin C$. Then there is a path s- e^s_v - v - e^t_u - t joining *s* and *t* in G \ C, which is a contradiction with the fact that *C* is a cut. Hence, in this case e^n_{uv} must be in *C*.

Assume that, e^s_u and e^s_v are in C but e^t_u and e^t_v are not. If possible, assume that, $e^n_{uv} \in C$. Then, $C \setminus \{e^n_{uv}\}$ is still a cut, which is a contradiction with the fact that, C is a cut and no proper subset of *C* can be a cut.

Remaining two cases can be addressed by analogous argument. Thus, claim 2 is proved.

The first term in the expression of cost of C mentioned in (4.4) represents the sum of weights of all e^n_{uv} corresponding to neighboring vertices *u* and *v* which are connected to different terminals in the induced graph G \ C. The edges e^n_{uv} not contained in *C*, are the ones, for which corresponding *u* and *v* are assigned same binary value by *X*. Hence, weights of such e^n_{uv} will be zero.

Thus, equation (4.6) becomes,

$$|C| = \sum_{\substack{(u,v)\in N \\ u,v\in V}} k.p_{uv} + \sum_{u\in V} |X_u - g_u|$$

But, by (4.1),

$$\text{i.e. } O(X) = \sum_{\substack{(u,v)\in N \\ u,v\in V}} kp_{uv} + \sum_{v\in V} |x_v - g_v|$$

Thus, $|C|$ = O(*X*).

This proves that, the cost of the minimum cut C minimizes the objective function O(*X*).

Note that, we start with an arbitrary binary labeling of the image and then refine it through graph cut terminology in order to improve it in terms of its value under objective function (4.1). The time complexity of the model is dependent on the initial binary labeling. This suggests that, the choice of the initial binarization function assigned to the function should be made tactfully.

4.4 IMPLEMENTATION AND RESULTS

The model is implemented using Java programming language. The input image is first converted into grayscale in order to facilitate the computation. The initial binary labeling is obtained through thresholding. Then after, the network flow is constructed, where weights of the edges are decided based on the binary value assigned to a particular pixel by the initial labeling. The code finds the most cost effective (with reference to cost / value / penalty assigned by the objective function) possible binary labeling in the move space. Following is the code for implementation of the model in Java programming language:

```
/******************************************************************************
* Compilation: javac FordFulkerson3.java
* Execution: java FordFulkerson3 V E
* Dependencies: FlowNetwork.java FlowEdge.java Queue.java
*
* Ford-Fulkerson algorithm for computing a max flow and
* a min cut using shortest augmenthing path rule.
*
******************************************************************************/
import java.awt.image.BufferedImage;
import java.awt.image.Raster;
import java.awt.image.ColorModel;
import java.io.IOException;
import java.io.File;
import java.io.PrintWriter;
import javax.imageio.ImageIO;
public class FordFulkerson3 {
        private boolean[] marked; // marked[v] = true iff s->v path in residual graph
        private FlowEdge[] edgeTo; // edgeTo[v] = last edge on shortest residual s->v path
        private double value; // current value of max flow
        // max flow in flow network G from s to t
        public FordFulkerson3(FlowNetwork G, int s, int t) {
            if (s < 0 || s >= G.V()) {
            throw new RuntimeException("Source s is invalid: " + s);
            }
            if (t < 0 || t >= G.V()) {
            throw new RuntimeException("Sink t is invalid: " + t);
            }
            if (s == t) {
            throw new RuntimeException("Source equals sink");
            }
            value = excess(G, t);
            if (!isFeasible(G, s, t)) {
            throw new RuntimeException("Initial flow is infeasible");
            }
            // while there exists an augmenting path, use it
            while (hasAugmentingPath(G, s, t)) {
                // compute bottleneck capacity
                double bottle = Double.POSITIVE_INFINITY;
                for (int v = t; v != s; v = edgeTo[v].other(v)) {
                bottle = Math.min(bottle, edgeTo[v].residualCapacityTo(v));
                }
```

```
                    // augment flow
                    for (int v = t; v != s; v = edgeTo[v].other(v)) {
                    edgeTo[v].addResidualFlowTo(v, bottle);
                    }
                    value += bottle;
                }
                // check optimality conditions
                assert check(G, s, t);
}
// return value of max flow
public double value() {
return value;
}
// is v in the s side of the min s-t cut?
public boolean inCut(int v) {
return marked[v];
}
// is there an augmenting path?
// if so, upon termination edgeTo[] will contain a parent-link representation of such a path
private boolean hasAugmentingPath(FlowNetwork G, int s, int t) {
        edgeTo = new FlowEdge[G.V()];
        marked = new boolean[G.V()];
        // breadth-first search
        Queue<Integer> q = new Queue<Integer>();
        q.enqueue(s);
        marked[s] = true;
        while (!q.isEmpty()) {
              int v = q.dequeue();
              for  (FlowEdge e : G.adj(v)) {
                    int w = e.other(v);

                    // if residual capacity from v to w
                    If  (e.residualCapacityTo(w) > 0) {
                        if  (!marked[w]) {
                            edgeTo[w] = e;
                            marked[w] = true;
                            q.enqueue(w);
                        }
                    }
              }
        }
        // is there an augmenting path?
        return marked[t];
}

// return excess flow at vertex v
private double excess(FlowNetwork G, int v) {
        double excess = 0.0;
        for  (FlowEdge e : G.adj(v)) {
              if (v == e.from()) excess -= e.flow();
              else excess += e.flow();
        }
        return excess;
}
// return excess flow at vertex v
```

```
private boolean isFeasible(FlowNetwork G, int s, int t) {
        double EPSILON = 1E-11;
        // check that capacity constraints are satisfied
        for  (int v = 0; v < G.V(); v++) {
              for (FlowEdge e : G.adj(v)) {
                  if (e.flow() < -EPSILON || e.flow() > e.capacity() + EPSILON) {
                  System.err.println("Edge does not satisfy capacity constraints: " + e);
                  return false;
                  }
              }
        }
        // check that net flow into a vertex equals zero, except at source and sink
        if (Math.abs(value + excess(G, s)) > EPSILON) {
            System.err.println("Excess at source = " + excess(G, s));
            System.err.println("Max flow = " + value);
            return false;
        }

        if (Math.abs(value - excess(G, t)) > EPSILON) {
            System.err.println("Excess at sink = " + excess(G, t));
            System.err.println("Max flow = " + value);
            return false;
        }
        for (int v = 0; v < G.V(); v++) {
            if (v == s || v == t) continue;
                else if (Math.abs(excess(G, v)) > EPSILON) {
                      System.err.println("Net flow out of " + v + " doesn't equal zero");
                      return false;
                }
        }
        return true;
}
// check optimality conditions
private boolean check(FlowNetwork G, int s, int t) {

        // check that flow is feasible

        if ( ! isFeasible(G, s, t)) {
            System.err.println("Flow is infeasible");
            return false;
        }
        // check that s is on the source side of min cut and that t is not on source side
        if  (!inCut(s)) {
              System.err.println("source " + s + " is not on source side of min cut");
              return false;
        }
        if  (inCut(t)) {
            System.err.println("sink " + t + " is on source side of min cut");
            return false;
        }
        // check that value of min cut = value of max flow

        double mincutValue = 0.0;
        for (int v = 0; v < G.V(); v++) {
              for  ( FlowEdge e : G.adj(v) ) {
```

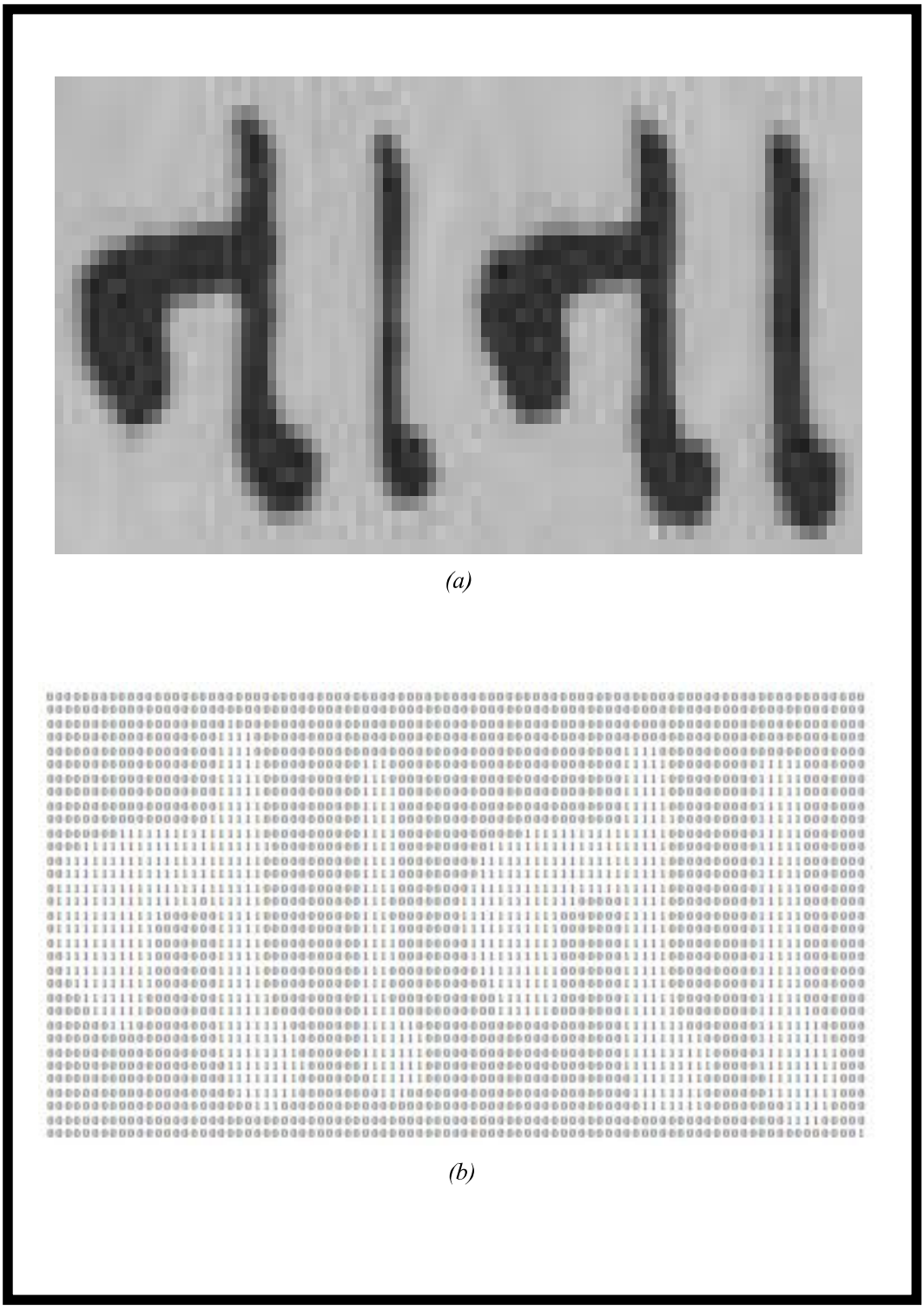
```
                if ((v == e.from()) && inCut(e.from()) && !inCut(e.to()))
                mincutValue += e.capacity();
            }
        }
        double EPSILON = 1E-11;
        if  ( Math.abs(mincutValue - value) > EPSILON ) {
            System.err.println("Max flow value = " + value + ", min cut value = " + mincutValue);
            return false;
        }
        return true;
}
// test client that creates random network, solves max flow, and prints results

public static void main(String[] args) {

        // create flow network with image of file args[0]

        long strt = System.currentTimeMillis();
        String fileName = args[0];
        File imageFile = new File(fileName);
        BufferedImage image = null;
        try {
            image = ImageIO.read(imageFile);
        } catch (IOException e) {
            e.printStackTrace();
        }
        Raster rstr = image.getData();
        int width = image.getWidth();
        int height = image.getHeight();
        int[][] intensities = new int[height][width];
        int type = image.getType();
        for ( int i = 0; i < height; i ++ ) {
              for (  int j = 0; j < width; j ++ ) {
                  if (type == BufferedImage.TYPE_BYTE_GRAY) {
                      intensities[i][j] = rstr.getSample(j,i,0);
                  }
              }
        }
        //PrincetonFlows gc = new PrincetonFlows(image);

        //In in = new In(args[0]);

        FlowNetwork G = new FlowNetwork(intensities,height,width);
        StdOut.println(G);
        int s = 0, t =G.V() - 1 ;

        // compute maximum flow and minimum cut

        FordFulkerson3 maxflow = new FordFulkerson3(G, s, t);
        StdOut.println("Max flow from " + s + " to " + t);
        for (int v = 0; v < G.V(); v++) {
            for (FlowEdge e : G.adj(v)) {
                if ((v == e.from()) && e.flow() > 0)
                    StdOut.println(" " + e);
            }
```

```
        }

        // print min-cut
        int rows = height;
        int cols = width;
        int numberOfVertices = G.V();
        if ( numberOfVertices != rows*cols +2 ) {
            System.out.println(" Error in image data ");
            System.exit(0);
        }
        int [] linear = new int[numberOfVertices];
        for ( int i = 0; i < numberOfVertices; i ++)
            linear[i] = 1;

        StdOut.print("Min cut: ");
        for  ( int v = 0; v < numberOfVertices; v++) {
              if  ( maxflow.inCut(v)) {
                  StdOut.print(v + " ");
                  linear[v] = 0;
              }
        }
        StdOut.println();

        StdOut.println("Max flow value = " + maxflow.value());
        System.out.println(" Binarized image ");
        int[][] binmat = new int[rows][cols];

        for ( int i = 1; i <= rows * cols ; i ++) {
              System.out.print(" "+linear[i]);
              if ( i % cols == 0 )
                  System.out.println();
        }
        long ends = System.currentTimeMillis();
        long secs = (ends-strt)/1000;
        System.out.println(secs + " seconds taken to binarize image "+args[0]);
        System.err.println(secs + " seconds taken to binarize image "+args[0]);
    }
}
```

We implemented the code on various textual image segments for binarization, results of which are presented in next few pages. The code worked considerably well in case of image segments of smaller dimensions. Due to limitations of the computing system (parallel computing environment and configuration of the system), all the implementations were made on a dual core system. The binarization resulted through the code are of good quality but the running time is considerably high. The present code cannot handle the entire document (of A4 size) at single go and hence needs refinement. However, the quality of the binarization is considerably good and proves the efficiency of the mathematical model implemented through the code.

Due to limitations of programming skills and computing infrastructure, the code is not time efficient and the real time implementation using the code is not practically feasible. Thus, the mathematical model needs to be re-implemented through efficient coding.

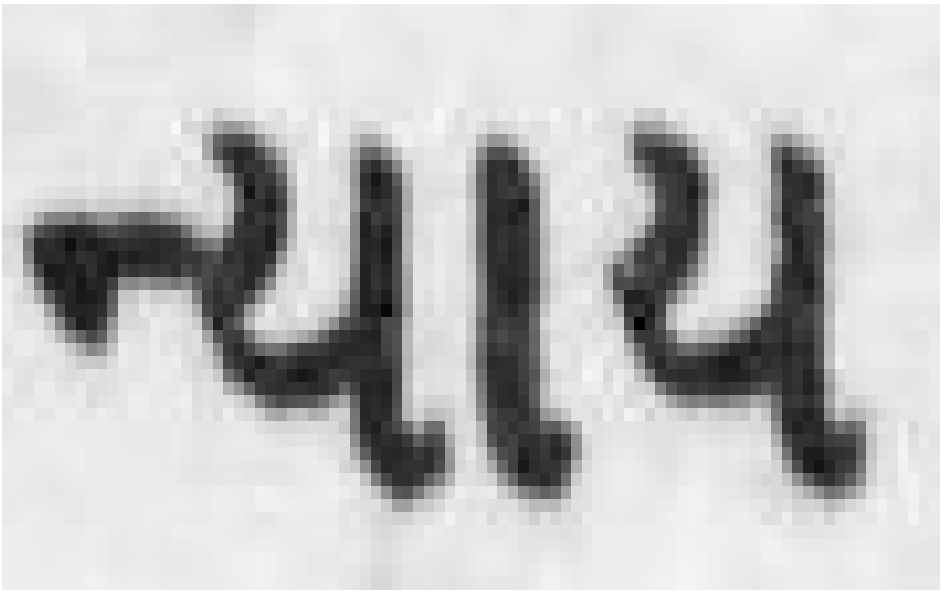
The results obtained through the code are given below:



(a)

(b)

Figure: 4.1(a) Input image (b) Binarization of the image



(a)

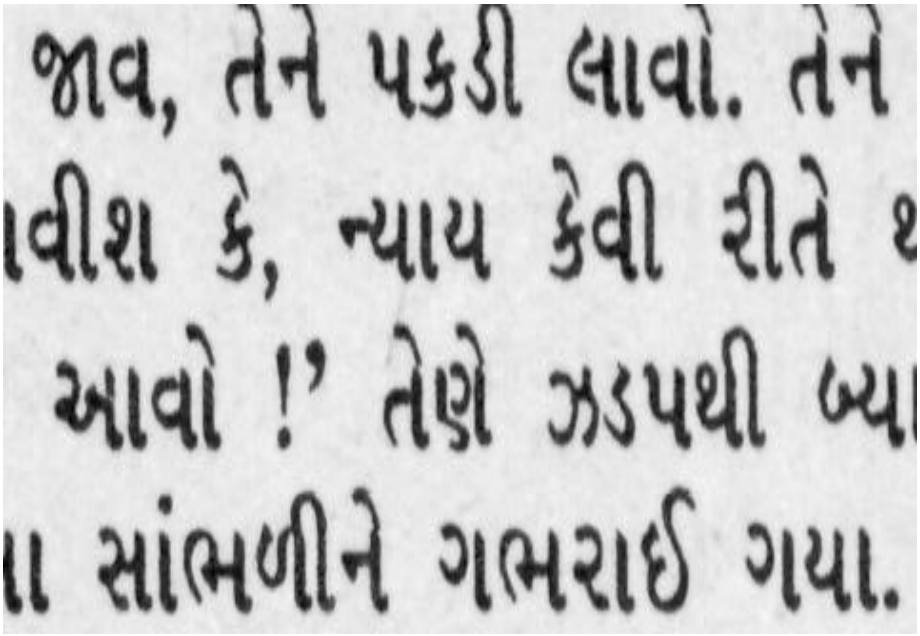
(b)

Figure: 4.2(a) Input image (b) Binarization of the image

(a)

(b)

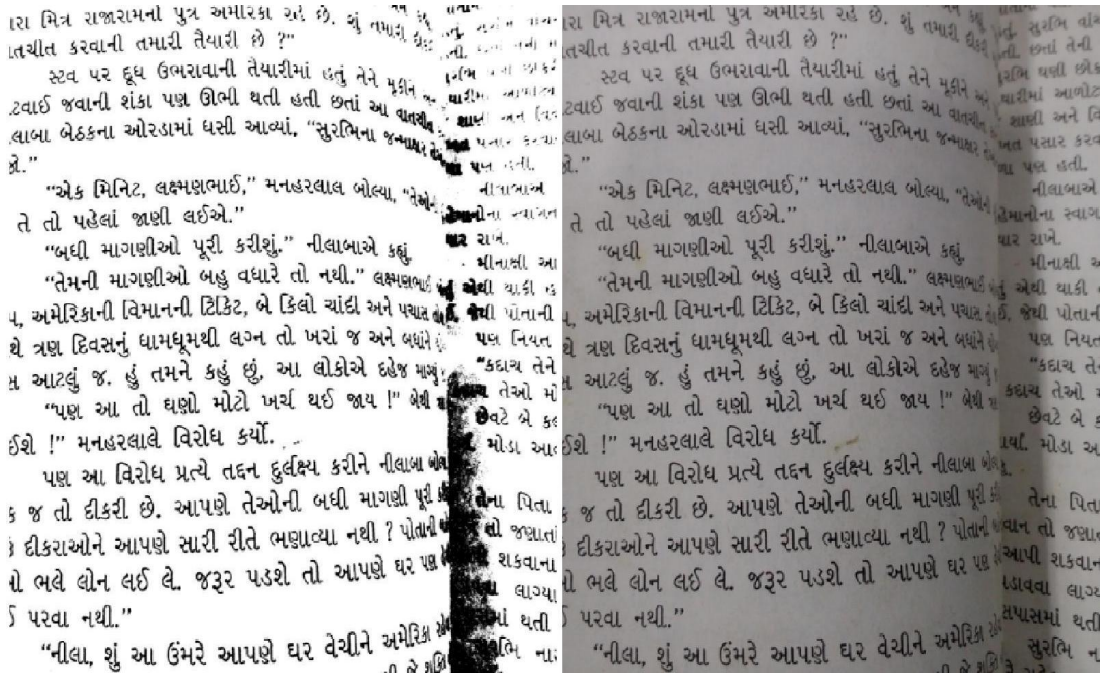
Figure: 4.3(a) Input image (b) Binarization of the image



(a)

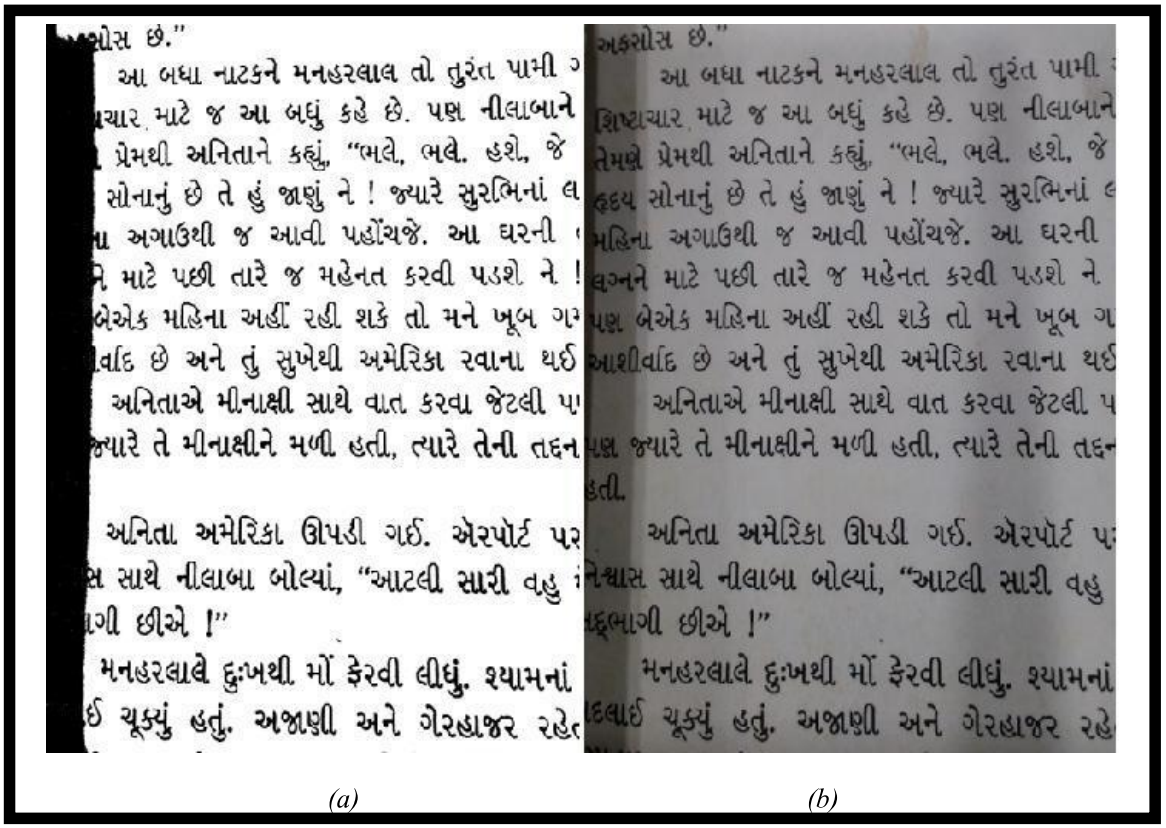
(b)

Figure: 4.4(a) input image (b) Binarization of the image



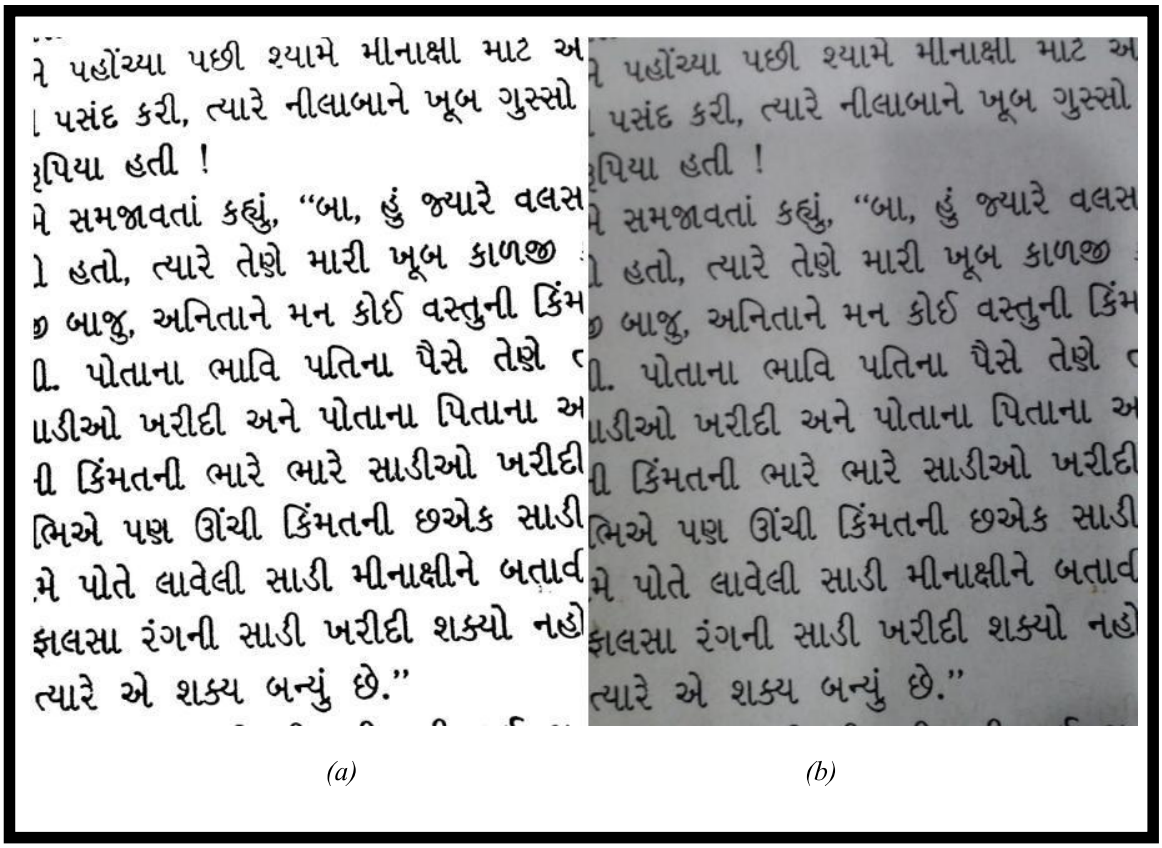
(a) *(b)*

Figure: 4.5 (a) Binarization of the image (b) input image



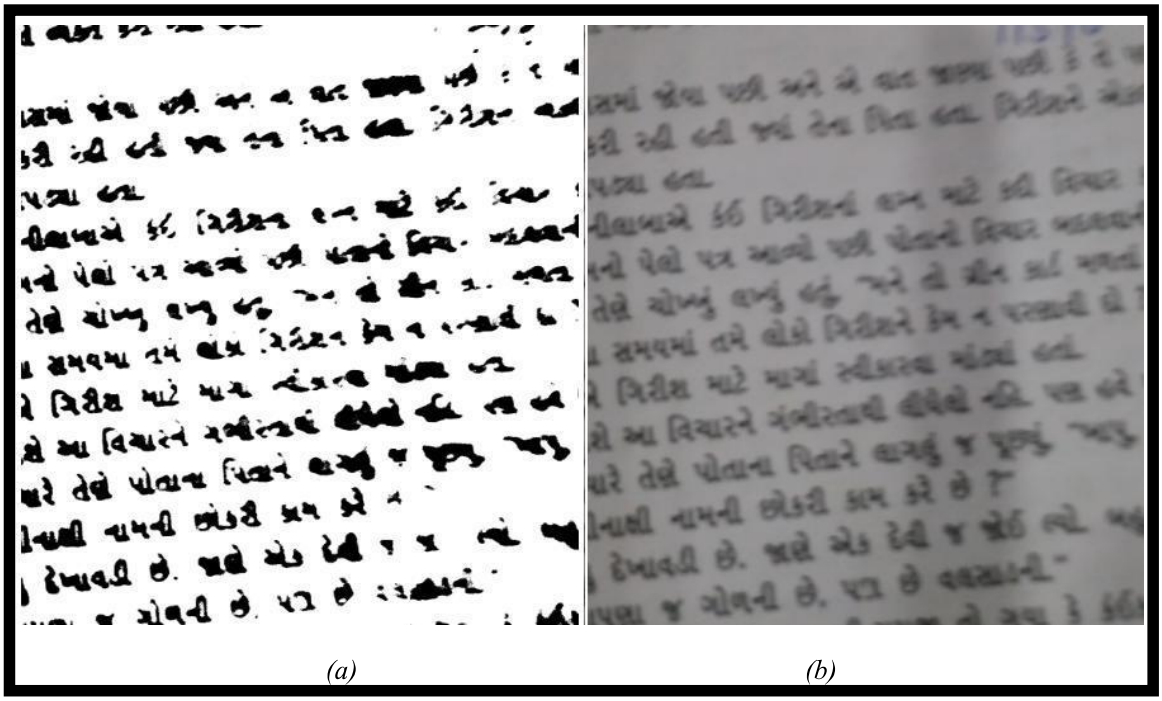
(a) *(b)*

Figure: 4.6 (a) Binarization of the image (b) input image



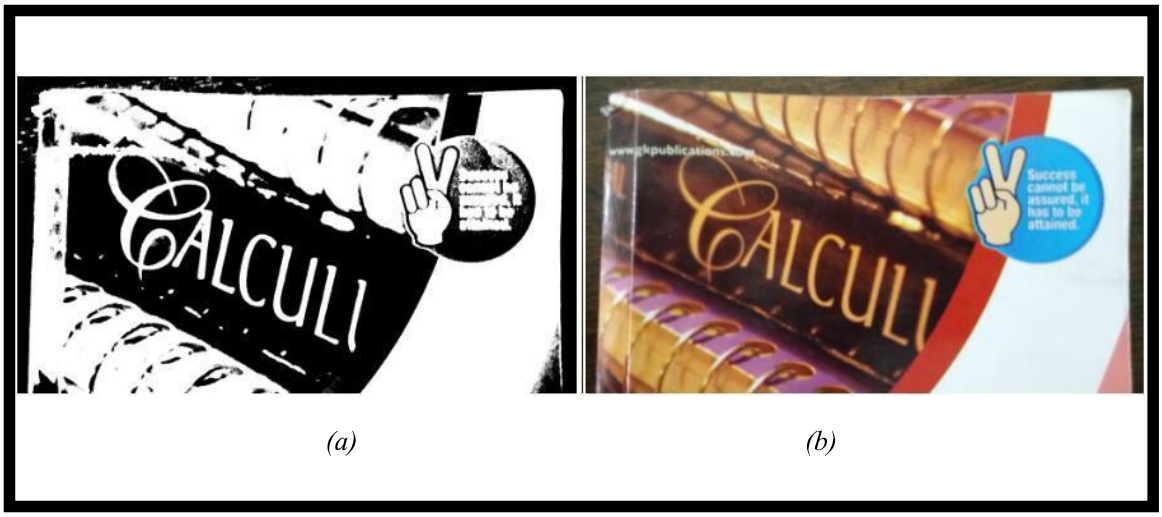
(a) *(b)*

Figure: 4.7 (a) Binarization of the image (b) input image



(a) *(b)*

Figure: 4.8 (a) Binarization of the image (b) input image



(a) *(b)*

Figure: 4.9 (a) Binarization of the image (b) input image

(a) *(b)*

Figure: 4.10 (a) Binarization of the image (b) input image

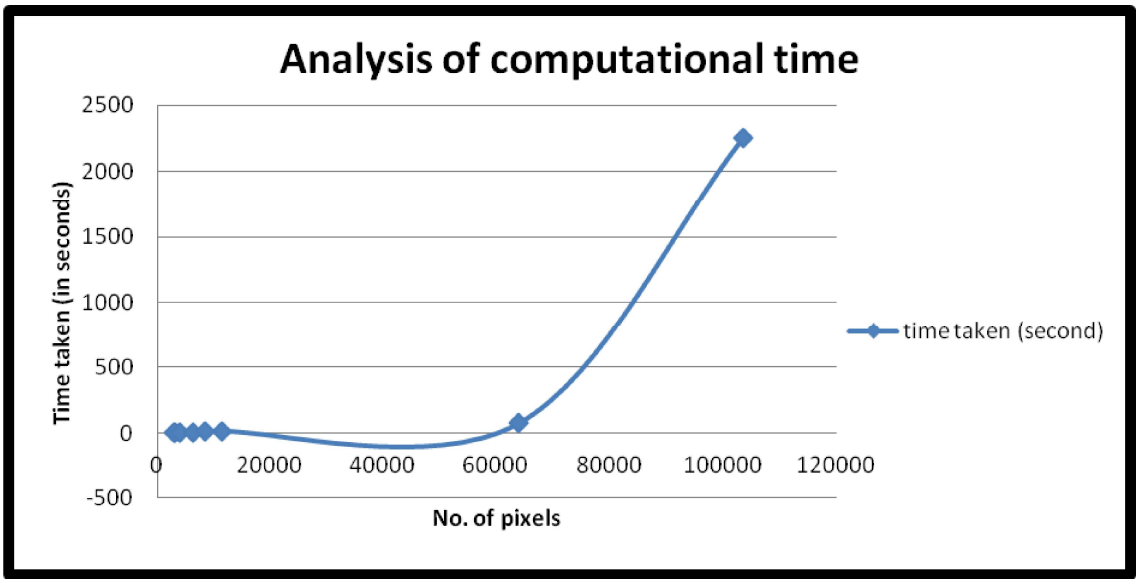


Figure 4.11 various image sizes versus Time required for binarization

As discussed earlier, the code turns out be quite expensive in terms of time it takes to handle the image for binarization, especially for larger image segments. Table 4.1 presents the time taken by the code for the image segments of various dimensions. For images of more than one tenth of a million pixels, the code takes about 38 minutes, which is very high time when compared to other popular binarization methods. Methods based on global threasholding take comparatively very small time for images of same size. Figure 4.11 shows that the time required for binarization of the image using our code grows exponentially with image size after a fixed stage. This is very serious drawback of our model. However, this deficiency could be easily overcome by better coding. Due to our limited programming skills, the code we wrote has vast scope for improvement.

Dimension of the scanned image	No. of pixels in the image	Time taken (in Seconds)
91 X 33	3003	2
91 X 35	3185	2
89 X 45	4005	3
100 X 63	6300	5
92 X 92	8464	10
120 X 95	11400	15
320 X 200	64000	75
480 X 216	103680	2258

Table 4.1 Time required for binarization for various image sizes

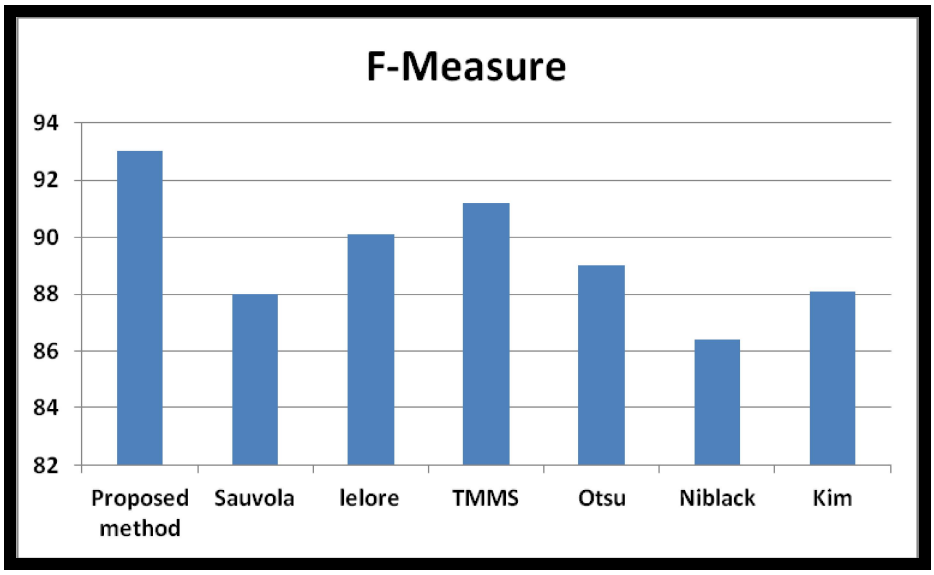


Figure 4.12 Bar graph showing various binarizatin methods and corresponding F-measure

There are many evaluation techniques to measure the quality of the binarization based on two main approaches: (1) pixel based accuracy evaluation and (2) OCR based evaluation. We have used some of the most prevalent evaluation techniques to measure the quality of our binarization results. The results are presented in the table 4.2.

We have used PERR, MSE and F-measure as evaluation techniques to measure the quality of binarization.

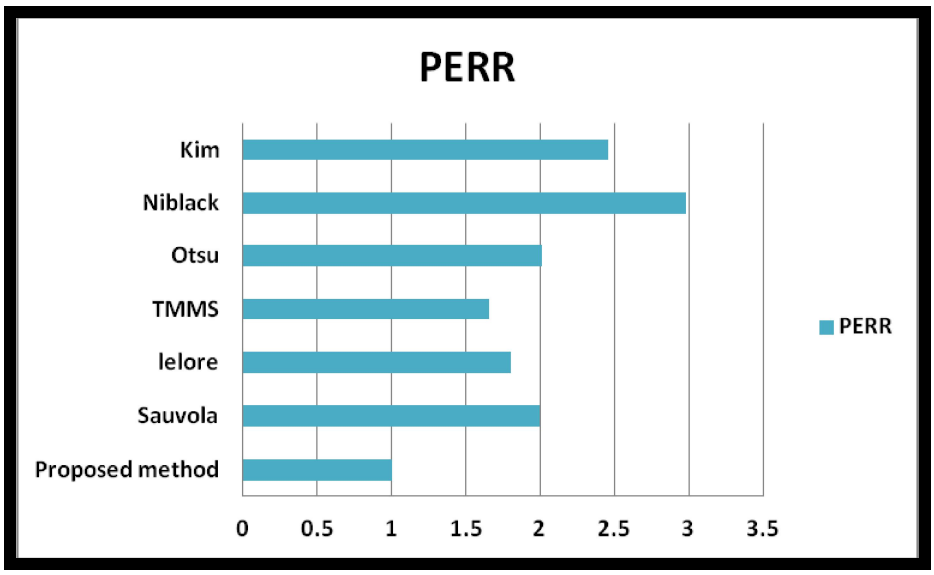


Figure 4.13 Bar graph showing various binarization methods and corresponding PERR value

Pixel Error Rate abbreviated as PERR is defined as $PERR = \frac{\text{Pixelerror}}{M\,x\,N}$. It gives the total number of misidentified pixels in the binarization (i.e., total number of those pixels which are of black colour in binarized image but of white colour in the original image and those which are white in the binarized one but are black in the original image). If *x(i, j)* represents the value of the pixel situated at i^{th} row and j^{th} column of the original image *x* and If *y(i, j)* represents the value of the corresponding pixel (i.e. pixel situated at i^{th} row and j^{th} column) of the binarized image *y*, the Mean Square Error rate (MSE) is defined as,

$$MSE = \frac{\sum_i \sum_j e(i,j)^2}{M\,x\,N}.$$

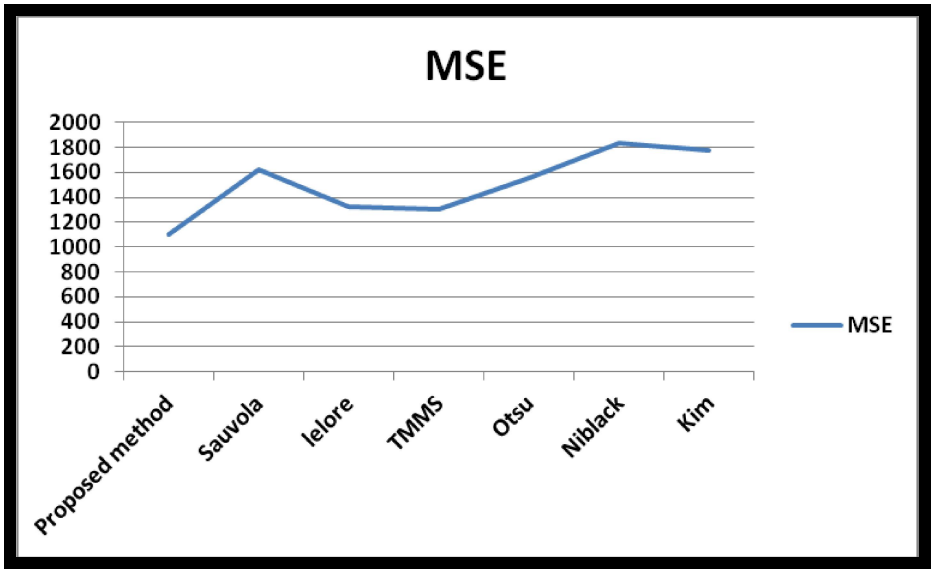


Figure 4.14 Line graph showing various binarization methods and corresponding MSE

As another measurement parameter, we have also used F-measure, which is defined as,

$$FM = \frac{2\,x\ \text{Recall}\ \ x\ \text{Precision}}{\text{Recall}\ \ +\ \text{Precision}}.$$

From the definition of Pixel Error Rate (PERR), smaller value of the PERR indicates that the algorithm for which PERR is measured is very accurate in terms of binarization results. In the same manner, smaller value of MSE (Mean Square Error) leads to the conclusion that, the quality of binarization produced by the algorithm is of high quality. On the contrary to this, the other third measurement technique, F-measure is quite opposite. Smaller value of F-measure proves superiority of the binarization technique.

Note that, our algorithm gives the maximum F-measure value (of 93) among all other methods listed in the table. It is 5 units higher than one of the most popular binarization method called Sauvola method. Thus, with respect to F –measure our method turns out to be superior to the popular methods listed in the table. The other two measures namely MSE and PERR measure some form of error in the binarization results. Hence, smaller values of these measures for the algorithm prove it better from the

others. As shown in the table, our method has the value of 1101.021 for MSE and 1.00312 for PERR, which is quite less than other binarization methods.

Method	FM	MSE	PERR
Proposed method	93	1101.021	1.00312
Sauvola	88	1622.132	2.00012
lelore	90.1	1321.001	1.80114
TMMS	91.2	1300.564	1.65842
Otsu	89	1551.256	2.01239
Niblack	86.4	1832.534	2.98560
Kim	88.1	1776.723	2.45601

Table 4.2 Comparison of our algorithm (Proposed method) with popular binarization algorithms

In nutshell, the new binarization method based on our mathematical model produces results better than existing and popular binarization methods. The underlying mathematical model efficiently minimizes the objective function and hence theoretically guarantees significant results. However, the only limitation emerging out of experimental results is high computational time, which needs to be addressed by better programming skills and re-coding of the algorithm.