Parallel Implementation of Otsu's Binarization Approach on GPU

Brij Mohan Singh, Rahul Sharma
Department of CSE,
College of Engineering Roorkee,
Roorkee-247667,Uttarakhand, India

Ankush Mittal
Director (Research),
Graphic Era University, Dehradun-248002, Uttarakhand, India

Debashish Ghosh
Department of E&C,
IIT Roorkee,
Roorkee-247667,Uttarakhand, India

ABSTRACT
Fast algorithms are important for efficient image processing systems for handling large set of calculations. To speedup the processing, parallel implementation of an algorithm can be done using Graphics Processing Unit (GPU). GPU is general purpose computation hardware; programmability and low cost make it productive. Binarization is widely used technique in the image analysis and recognition applications. In this paper, we investigate the accuracy and performance characteristics of GPUs on well known global binarization Otsu's approach for Optical Character Recognition systems. The main goal of this research work is to make binarization faster for recognition of a large number of document images on GPU. The algorithm is implemented using Compute Unified Device Architecture (CUDA). Experimental results show that parallel implementation achieved an average speedup of 1.6x over the serial implementation when running on a GPU named GeForce 9500 GT having 32 cores. Otsu's method is also evaluated using PSNR, F-measure, NRM, and IND evaluation measures.

General Terms
Document Analysis and Recognition, Image Processing, Pattern Recognition.

Keywords
Binarization; CUDA; GPU; OCR; Parallelization.

1. INTRODUCTION
Binarization is an active research area in the field of Document Image Processing. Binarization (thresholding) converts grey image into binary image. Binarization of document images is the first most important step in pre-processing of scanned documents to save all or maximum subcomponents such us text, background and image [1]. Binarization computes the threshold value that differentiate object and background pixels [2]. Color and grey level image processing consumes lots of processing powers. But binary images decrease computational load and increase efficiency of the systems.

Binarization has many applications such as medical image processing, document image analysis, face recognition etc. [3] Binarization can be classified into two categories: global and adaptive. Global methods [4-9] are based on the finding a single threshold value for the entire image, and adaptive methods [10-15] are based on the local information obtained from the candidate pixel and is needed for the calculation of threshold value for each pixel. If illumination of input image is not equal (evenly illuminated), local methods might perform better. If image has equal illumination then global methods can work better. But global methods cannot handle any of the image degradation and not able to remove noise. Local methods are significantly more time-consuming and computationally expensive.

Fast and accurate algorithms are necessary for Optical Character Recognition (OCR) systems to perform operations on document images. To speedup the processing, parallel implementation of an algorithm can be done using Graphics Processing Unit (GPU) as general purpose computation hardware; programmability and low cost make it productive [16].

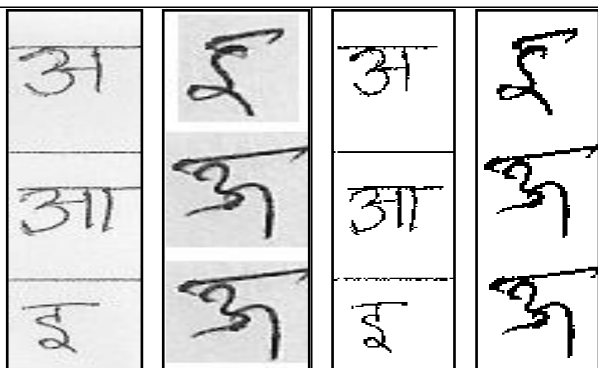
Some studies of GPU implementation are in [17-22]. To reduce the computation of numerical problems, parallel implementations on GPUs have been applied in [22-25].Parallel implementations on GPUs to handwritten character recognition were proposed in [26-30]. Oh at al. parallelized Neural Networks on GPU. In [31], GPU was used to implement the matrix multiplication of a Neural Network to enhance the time performance. Jung [32] proposed a Neural Network based text localization in color images. Recently, Singh et al. proposed parallel implementation of well known profiling based segmentation algorithm for Devanagari character recognition on GPU [33].

2. NVIDIA CUDA
The programmable GPU has evolved due to growing need for real-time and high definition 3D graphics processing. It has evolved into multithreaded, highly parallel and multi-core chip system with excellent computational and high memory bandwidth [34]. To fulfill the dream of parallelization, CUDA™ was introduced by NVIDIA in November 2006 [35]. It is a general purpose parallel computing architecture. It contains new instruction set architecture and parallel programming model. CUDA provides a new software environment that allows developers to use C as a high-level programming language that enable a straightforward implementation of parallel algorithms

and supports heterogeneous computation where applications use both the CPU and GPU. Serial implementations of algorithm run on CPUs and parallel implementations run on GPUs. CPU and GPU have own memory space when executing programs and allows simultaneous computation on both the CPU and GPU without contention for memory resources. Many languages such as FORTRAN, C++, OpenCL, and DirectX Compute will be supported in the future. The development of application software that transparently scales its parallelism to leverage the increasing number of processor cores such as GPUs are challenging. To run the CUDA programs, the CUDA Toolkit for compiling and build a CUDA application in conjunction with Microsoft Visual Studio and CUDA SDK includes sample projects that have all the necessary project configuration and build files to perform one-click builds using Microsoft Visual Studio.

3. OTSU'S METHOD OF BINARIZATION

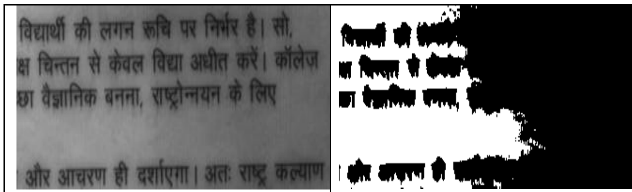
The most well-known global binarization method was proposed by N. Otsu [9]. Otsu's method works better where clear separation between foreground and background exists or where image illumination is not variable as shown in fig. 1. Unfortunately, real life document images possess various kinds of degradations (e.g. illumination contrast, skewed, stains, and noise) that weaken thresholding proposed by N. Otsu as shown in fig.2.



(a)	(b)	(c)	(d)
Input Images		Binarized images by Otsu's Method	

Figure 1: Sample of Binarization

This method is based on the pixels of an image are separated into two classes C_0 (e.g. objects) and C_1 (background) and by a threshold T. C_0 denotes pixels with gray level [1, ..., T] and C_1 denotes pixels with gray level [T + 1, ..., L]. The gray level histogram is normalized and regarded as a probability distribution:



Input Images	Binarized image by Otsu's Method

Figure 2: Otsu's binarization over degraded image

$$\sigma_{\omega}^{2}(t)=\omega_1(t)\sigma^2{}_1(t)+\omega_2(t)\sigma^2{}_2(t) \quad (1)$$

Where ωi (Weights) are the probabilities of the two classes and $\sigma^2{}_i$ variances of classes. According to Otsu, intra-class variance minimizing is the same as maximizing inter-class variance:

$$\sigma^2{}_b(t)=\sigma^2-\sigma^2{}_\omega(t)=\omega_1(t)\omega_2(t)[\mu_1(t)-\mu_2(t)]^2 \quad (2)$$

which is expressed in terms of class probabilities ωi and class means μ_i which in turn can be updated iteratively.

The sequential algorithm is implemented in C++ and making use of C++ Standard Template Library. GCC 3.4.2 (mingw-special) compiler is used and thread model is win32. The following pseudo-code outlines the structure of sequential code of Otsu's method.

```
int bin[256], bin1, bin2;
for( int i=0; i< 256; i++)
        bin[i] = 0;
for( int i=0; i<nop; i++)
        bin[image[i]]++;
for( int i=0; bin[i]==0; i++)
        bin1 = i;
for( int i=255; bin[i]==0; i--)
        bin2 = i;
double nh[256], ch[256], m[256];
for( int i=0; i<256; i++)
        nh[i] = bin[i]/nop;
ch[0] = nh[0];
m[0] = 0.0;
for( int =1; i<256; i++){
        ch[i] = ch[i-1] + nh[i];
        m[i] = m[i-1] + i*nh[i];
}
double mean = m[255], max=0;
int threshold = 0;
for( i=bin1; i<=bin2; i++){
        bcv = mean*ch[i]-m[i];
        bcv *= bcv/(ch[i]*(1.0-ch[i]));
        if( max<bcv){
                max = bcv;
```

```
                threshold = I;
        }
}
for( int i=0; i<nop; i++)
        outImage[i] = (image[i]>threshold)?WHITE:BLACK;
where
'nop' is number of pixels in image
```

4. PARALLEL IMPLEMENTATION

In the second set of experiment the algorithm is parallelized using CUDA platform. In CUDA, it is assumed that both host and device maintain their own DRAM. Host memory is allocated using malloc and device memory is allocated using cudaMalloc. CUDA threads are assigned a unique thread ID that identifies its location within the thread, block and grid. The following pseudo-code outlines the structure of parallel code of Otsu's method.

```
Main()
{
        Define block and grid
        Kernal_Histogram(numOfPixels, image, histogram)
        Calculate threshold from histogram
        Kernal_Threshold(numOfPixels, image, threshold)
}
Kernal_Histogram(numOfPixels, image, histogram){
        Declare shared memory subHist[numOfThreads][256];
        for each data pixels in window
                subHist [threadId][ currentPixelValue]++;
        end for
        Apply scan to merge subHist into histogram
}

Kernal_Threshold(numOfPixels, image, threshold){
        for each data pixels in window
                if(currentPixelValue<threshold)
                        currentPixelValue = BLACK;
                else    currentPixelValue = WHITE;
        end for
}
```

5. HARDWARE SPECIFICATIONS

All the experiments are carried out using the hardware specifications of GPU: GeForce 9500 GT, 1 MB DDR2, No of Processors = 4, No of core =32, RAM 1 GB, Frequency 1.35 GHz, DDR2 and CPU: Intel Core 2 Duo, 2.66 GHZ, No of cores available =2, No of thread=1, No of thread/core=1, Physical Memory =2 GB, DDR2

6. EVALUATION MEASURES

- F-Measure: it is a way of combining recall and precision scores into a single measure of performance

$$F-Measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \quad (2)$$

Where $Recall = \frac{TP}{TP+FN}$ and $Precision = \frac{TP}{TP+FP}$

- Peak Signal to Noise Ratio (PSNR): the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

$$PSNR = 10 \cdot \log\left(\frac{C^2}{MSE}\right) \quad (3)$$

Where $MSE = \frac{\sum_{x=1}^{M}\sum_{y=1}^{N}\left(I(x,y)-I'(x,y)\right)^2}{MN}$, I is the original image and I' is binarized.

- Negative Rate Metric (NRM): It is based on pixel-wise mismatches between the xground truth and observations in a frame.

$$NRM = \frac{NR_{FN} + NR_{FP}}{2} \quad (4)$$

Where $NR_{FN} = \frac{FN}{FN+TP}$ and $NR_{FP} = \frac{FP}{FP+TN}$

- Information to Noise Difference (IND): It is a method to test the quality of the binarized image based on information and noise.

$$IND = Ivalue - Nvalue \quad (5)$$

Where $Ivalue = \frac{TP}{NB_{GT}}$ and $Nvalue = \frac{FP}{NB_{BI}}$; NBGT and

NBBI are the number of black pixels in ground truth and in binarized image respectively. Here Ivalue signifies the information preserved in the binarized image and Nvalue represents the noise in the binarized image. The value of IND ranges between -1 to +1 where +1 means binarized image is the exact copy of ground truth while -1 signifies that binarized image is the invert of ground truth.

7. RESULTS AND DISCUSSION

For the testing of Otsu's approach of local binarization, we collected a data set of handwritten as well as printed documents from newspapers, old books and from different writers. The collected documents are scanned using a scanner at 300 dpi and tested on the computer and GPU specifications shown in content 5. To make faster the method, we parallelized it on CUDA and achieved an average speedup of 1.6x over the serial implementation when running on a GPU. The comparison of execution time of serial implementation over parallel is shown in table 1. Since Otsu's method is global binarization method, the execution time of algorithm depends on the window size and image size in megapixels as shown in fig.3.

The results of Otsu's binarization approach are shown in fig. 4 that demonstrates the efficiency of this approach. The algorithm of global binarization approach is also evaluated using PSNR, F-measure, NRM and IND measures. The experimental results of evaluation measure are shown in table 2.

Table 1: comparison of execution time of serial implementation over parallel

Window Size	Megapixels	Serial	Parallel	Speed-Up	Speed-Up Average
7	1	0.0169	0.0106	1.59	1.6x
	2	0.0335	0.0209	1.60	
	4	0.067	0.0403	1.66	
	8	0.1339	0.0804	1.66	
	16	0.2679	0.1597	1.67	

Table 2: Evaluation Measures

Image	F- measure (%)	PSNR (db)	NRM (10-2)	IND
1	93.88	70.36	1.59	.171
2	92.23	65.06	1.99	.198
3	90.35	63.12	2.01	.201
4	72.76	50.31	2.87	.278

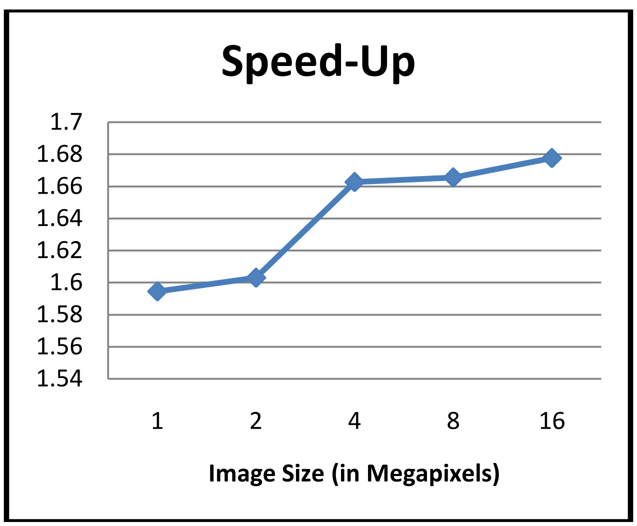


Figure 3: Speed-up Vs. Image size

8. CONCLUSION

In this research work, a well known Otsu's global binarization algorithm has been parallelized and analyzed with evaluation measures. The method is evaluated using PSNR, F-measure, NRM, and IND evaluation measures. The implementation of binarization algorithm on the graphics device is promising. However, proposed method is not so much speed-up due to its global properties of binarization.

Fast and accurate algorithms are necessary for fast Optical Character Recognition (OCR) systems to perform operations on large size document images. To speedup the processing, parallel implementation of algorithms on Graphics Processing Unit (GPU) makes it attractive. GPU itself has been shown to be an excellent hardware device to accelerate computational speed-up in the development of fast OCR systems.

9. REFERENCES

[1] Fernando, R and Kilgard, M. J. 2003. The Cg tutorial the definitive guide to programmable real-time graphics. Addison-Wesley.

[2] Moravanszky, A. 2003. Linear algebra on the GPU, in: W.F. Engel (Ed.), Shader X 2, Wordware Publishing, Texas.

[3] Manocha, D. 2003. Interactive geometric & scientific computations using graphics hardware, SIGGRAPH 2003 Tutorial Course #11.

[4] Moreland, K. and Angel E. 2003. The FFT on a GPU. In Proceedings of SIGGRAPH Conference on Graphics Hardware, 112-119.

[5] Mairal, J., Keriven, R. and Chariot, A. 2006. Fast and efficient dense variational Stereo on GPU. In Proceedings of International Symposium on 3D Data Processing, Visualization, and Transmission, 97-704.

[6] Yang, R. and Welch, G. 2002. Fast image segmentation and smoothing using commodity graphics hardware. Journal of Graphics Tools, Vol. 17, (4), 91-100.

[7] Fung, J. and Man, S. 2005. OpenVIDIA: Parallel GPU computer vision. In Proceedings of ACM International Conference on Multimedia, 849-852.

[8] Jang, H., Park, A. and Jung, K. 2008. Neural network implementation using CUDA and OpenMP. In Proceeding of Computing: Techniques and Applications, (DICTA), IEEE, 155 – 161.

[9] Otsu, N. 1979. A threshold selection method from gray level histograms. IEEE Trans. on Systems, Man and Cybernetics, Vol. 9, 62-66.

[10] Yu, B., Jain, A. and Mohiuddin, M. 1997. Address block location on complex mail Pieces," In Proceeding of International Conference of Document Analysis and Recognition, IEEE, 897-901.

[11] Rosenfeld, A. and Kak, A.C. 1982. Digital picture processing, second ed., Academic Press, New York.

[12] Kittler J. and Illingworth J. 1985. On threshold selection using clustering criteria. IEEE Trans. Systems Man Cybernetics, Vol. 15, 652–655.

[13] Brink, A.D. 1992. Thresholding of digital images using two-dimensional entropies. Pattern Recognition, Vol. 25, 803–808.

[14] Yan, H. 1996. Unified formulation of a class of image thresholding techniques. Pattern Recognition, Vol. 29, 2025–2032.

[15] Bernsen, J. 1986. Dynamic thresholding of grey-level images. In Proceeding of International Conference of Pattern Recognition, 1251-1255.

[16] Niblack, W. 1986. An Introduction to digital image processing, Prentice-Hall, Englewood Cliffs, NJ, 115–116.

[17] Sauvola, J. and Pietikainen, M. 2000. Adaptive document image binarization. Pattern Recognition, Vol. 33, 225–236.

[18] Kim, I.K., Jung, D.W. and Park, R.H. 2002. Document image binarization based on topographic analysis using a water flow model. Pattern Recognition, Vol. 35, 265–277.

[19] Gatos, B., Pratikakis, I. and Perantonis, S. J. 2006. Adaptive degraded document image binarization. Pattern Recognition, Vol. 39, 317–327.

[20] Chang, Y.F., Pai, Y.T. and Ruan, S.J. 2008. An efficient thresholding algorithm for degraded document images based on intelligent block detection. In Proceeding of IEEE International Conference on Systems, Man, and Cybernetics,667-672.

[21] Valizadeh, M., Komeili, M., Armanfard, N. and Kabir, E. 2009. Degraded document image binarization based on combination of two complementary algorithms. In Proceeding of International Conference of Advances in Computational Tools for Engineering Applications, IEEE, 595-599.

[22] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A. E. and Purcell, T. J. 2005. A survey of general-purpose computation on graphics hardware. In proceeding of Eurographics, State of the Art Reports, 21–51.

[23] Larsen, E. S., McAllister, D. 2001. Fast Matrix Multiplies using Graphics Hardware. In Proceeding of International Conference for High Performance Computing and Communications, 159-168.

[24] Trendall C. and Stewart, A. J. 2000. General calculations using graphics hardware with applications to interactive caustics. Rendering Techniques 2000: 11th Eurographics Workshop on Rendering, 287-298.

[25] Li, Wei, Wei, Xiaoming, A. and Kaufman, 2001. Implementing lattice boltzmann computation on graphics hardware. In proceeding of the International Conference for High Performance Computing and Communications.

[26] Mizukami, Y., Koga, K. and Torioka, T. 1994. A handwritten character recognition system using hierarchical extraction of displacement. IEICE, J77-D-II(12):2390–2393.

[27] Kruger, J. and Westermann, R. 2003. Linear operators for GPU implementation of numerical algorithms. In Proceedings of SIGGRAPH, San Diego, 908- 916.

[28] Steinkraus, D., Buck, I., and Simard, P. Y. 2005. GPUs for machine learning algorithms. In proceeding of International Conference of Document Analysis and Recognition, 1115-1120.

[29] Mizukami, Y. and Koga, K. 1996. A handwritten character recognition system using hierarchical displacement extraction algorithm. In Proceeding of International Conference of Pattern Recognition, volume 3,160–164.

[30] Ilie, A. Optical character recognition on graphics hardware. Downloaded from www.cs.unc.edu/~adyilie/IP/Final.pdf

[31] Oh, K.S. and Jung, K. 2004. GPU implementation of neural networks. Pattern Recognition, Elsevier, 1311-1314.

[32] Jung, K. 2001. Neural Network-based text localization in color images. Pattern Recognition Letters, Vol. 22, (4), 1503- 1515.

[33] Singh, B.M., Mittal A. and Ghosh, D. 2011. Parallel implementation of Devanagari text line and word segmentation approach on GPU. International Journal of Computer Applications 24(9):7–14.

[34] NVIDIA CUDA Programming Guide Version 2.0, available at www.nvidia.com/object/cuda_develop.html.

[35] NVIDIA Corporation: NVIDIA CUDA programming guide. Jan 2007, available at http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf

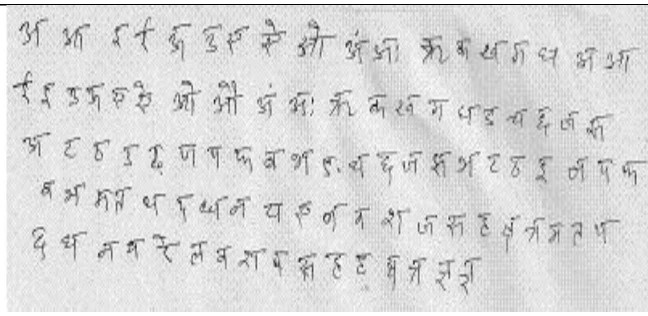
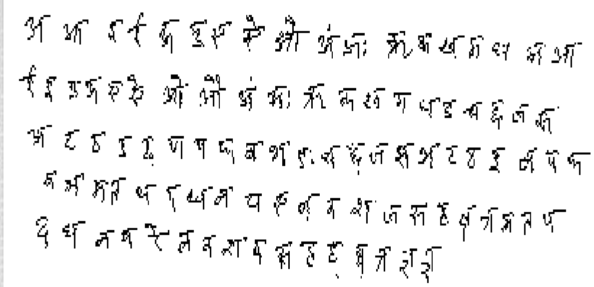
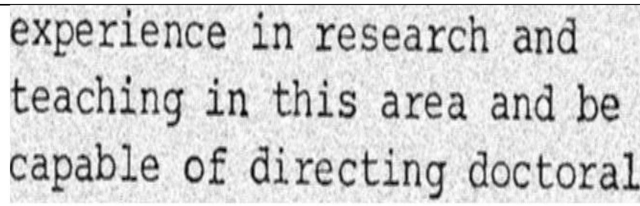
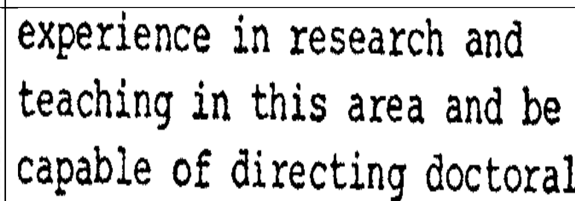
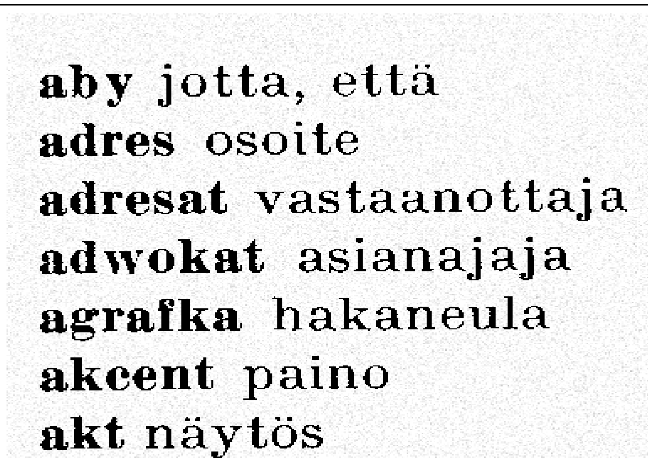
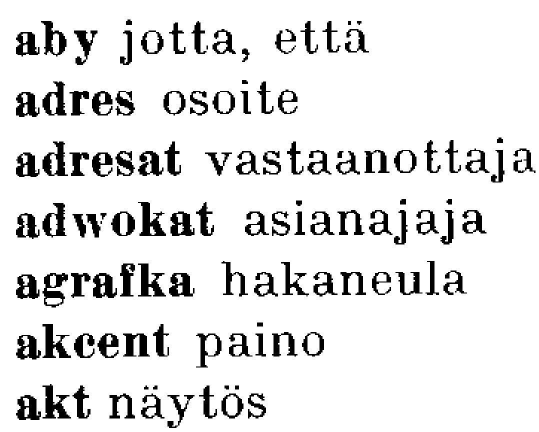
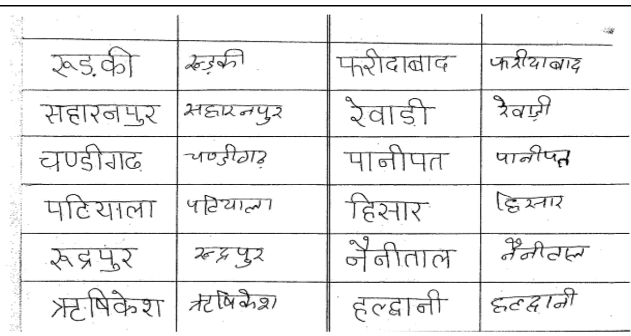
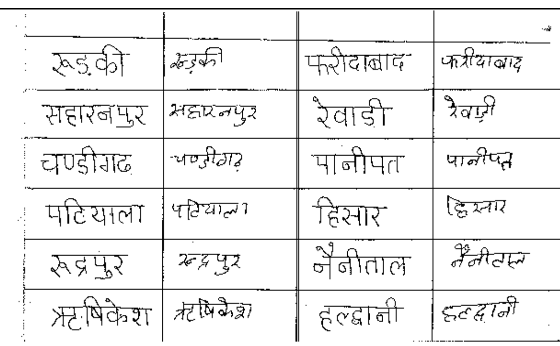
Image	Input Image	Output Image
1		
2		
3		
4		

Figure 4: Otsu’s output of binarization