

Image Processing Functions of Pathak– the Bangla OCR

Author: Almagir Mohammed

Assistant Professor, Dept of CSE, SUST.

Email: alamgir99@gmail.com Web: www.apona-bd.com

Last update: 17/06/06

About the document:

This document details the image processing functions of the Bangla OCR software Apona Pathak that I have developed and now maintain. This is being released in public with the hope that it may help other OCR developers. But no responsibility is assumed. The information contained in this document have been found by me unless where clearly specified. You may use this document for any kind of use, but you may NOT modify, edit or make it part of another document. If you do need to any of these task, please contact me.

Introduction:

Like any other OCR software Pathak has some image processing functions implemented within. Most of these processing are NOT exactly reported image processing literature because of the unusual nature of text image. For example, a text page has a quite different histogram than an ordinary photograph. Pathak has functions like, determining whether an image is a text page or photograph, finding and estimating the skew of text, converting gray image to binary etc.

Text Image:


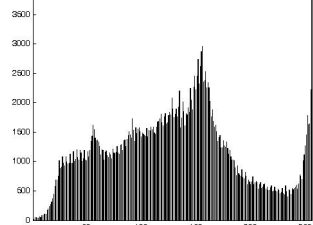
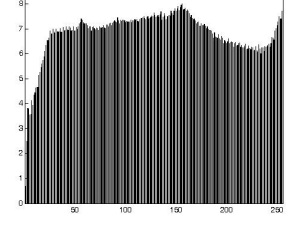
Text images have quite different statistics as compared to an ordinary photograph of nature or human face etc. As a result, image processing techniques which are applicable to photographs are not quite suitable for text image. In general, less than 10% area of a page is dark, where as most photographs have continuous coverage over the entire page. Looking straight at the histogram of an image can reveal whether an image is a text page or not. But in programming a subjective threshold measure is required to make the decision.

Pathak follows a very simple procedure for identifying text pages from photographs. The algorithm is:

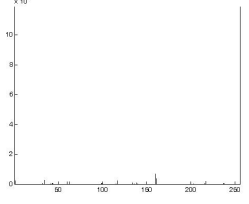
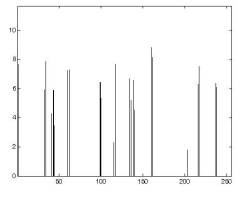
1. If the image is RGB, then convert to gray scale image.
2. Let h be the histogram of the gray image.
3. Set all zero values in h to 1 (to avoid log of zero).
4. Set $hl = \log(h)$ [compute the logarithm]
5. Find mx the maximum and avg the average value of hl .
6. if $avg < mx/2$ then the image is likely to be a text page
7. Otherwise it is a photograph.

The algorithm in action:

Example 1:

Image	Histogram	Log of histogram	stat	result
			avg=6.8 mx=8.2	photo

Example 2:

Image	Histogram	Log of histogram	stat	result
সম্পর্ক জলের সাথে জলাঙ্গী জীবনে জলের আঁশটে গন্ধ ভালোবাসি খুব মৎসভূক পাখি নই তবু দিই ডুব যেন এক কর্মযোগী জলের চরণে। শুধু শকুন্তলা, মুহম্মদ নুরুল হুদা।			avg=0.6 mx = 11.7	text

Skew detection:

Though documents are scanned with care, there is always some skew in scanned image. If an image is surely a text page then there are a number of ways to find the skew in the image. OCR of Latin based languages first apply an averaging over the image to blur the text and applies some algorithm to detect the presence of line. Luckily Bangla texts have always matra line and that is the best thing to detect the skew in an image. For detecting the presence of any line and determining its slope Hough transform is known to be best algorithm. Unfortunately, it can not be used without paying royalty or infringing the patent held by its inventor. There are a few other ways. The method Pathak uses is based on a very simple technique. The only limitation of this method is that it works only when the skew angle is less than 5 degrees or so. Fortunately most documents do not have skews more than 5 degrees.

First, we slice the image into a number of vertical slices and calculate the horizontal projection of each slice. Four to six slices are enough. The projection profiles are then correlated with each others in pair to find the best correlation delay. The correlation delay and horizontal distance of the slices corresponding to the profiles give the skew angle.

Skew detection algorithm:

1. Let $imwd$ = the width of the given image
2. Set $profwd = imwd/5$ [the width of a slice]
3. Set $startpoint1 = profwd/2$ [where the first slice begins]
4. Set $startpoint2 = profwd*2.5$ [where the second slice begins]
5. Compute profile1 array by adding all pixels from columns $startpoint1$ to $startpoint1+profwd$, for each row of the image

6. Compute profile2 array by adding all pixels from columns startpoint2 to startpoint1+profwd, for each row of the image
7. Set $L = (\text{startpoint2} - \text{startpoint1}) * \tan(2 * \pi * \text{thetamax} / 360)$ [take thetamax = 5]
8. Set acc, an array of $2L+1$ elements to zeros.
9. For each delay l in $(-L$ to $+L)$ compute the cross-correlation of profile1 and profile2 [this is basically multiplying elements of profile1 with those of profile2 but delayed by l , and then adding them up the products.]
10. The maximum value in acc denotes the strongest correlation and so corresponding l denotes the skew angle.
11. The exact skew angle is $\text{skew} = (l_{mx} - L) / (\text{startpoint2} - \text{startpoint1})$, where l_{mx} is the l corresponding to maximum value in acc. The angle is in radians.

Correction for skew:

Once the skew angle is known, the image can be rotated in opposite angle in order to compensate for the skew. Textbook algorithms for rotation are accurate but relies on (double precision) floating point arithmetic. Often the text image is large and rotating such an image with textbook algorithm takes a long time. The alternative is a fixed-point implementation. Pathak uses a 32-bit fixed-point fast rotation algorithm based on incremental design.

The algorithm:

let theta = the angle of rotation in radians

set Cos = (int) (8192*cos(theta)); // multiply by 8192 and take integer part

set Sin = (int) (8192*sin(theta));

Center of rotation,

$h = \text{Width}/2$

$k = \text{Height}/2$;

[rotate the first pixel]

Set $x = 0$; $y = 0$; [first pixel]

$xval = x * \text{Cos} + y * \text{Sin} - h * \text{Cos} - k * \text{Sin} + h * 8192$; //where would it go in x

$yval = -x * \text{Sin} + y * \text{Cos} + h * \text{Sin} - k * \text{Cos} + k * 8192$; // where would it go in y

for $y=0$ to $\text{Height}-1$ // for each row

set oldx = xval;

set oldy = yval;

for $x=0$ to $\text{Width}-1$ // for each column in the row

set outx = $xval \gg 13$; // divide by 8192

set outy = $yval \gg 13$;

if ((outx ≥ 0) && (outx $< \text{Width}$)) //within width

if((outy ≥ 0) && (outy $< \text{Height}$)) //within height

img_rotated(x,y) = GetPix at (outx, outy);

```

        xval += Cos;
        yval -= Sin;
end for
// new row

xval = oldx + Sin;
yval = oldy + Cos;
enf for

```

Converting to Binary:

OCR algorithms are generally design to work on binary or black and white images. Gray images are needed to be converted to binary for this reason. The process is very easy. Given a threshold, any pixel higher than that gets 0 (a higher gray level usually means a white pixel, but this could be different) and those less than the threshold get 1. The biggest problem is finding a good threshold. This is one of big challenges among the image processing things in an OCR design. A higher value may make the image too skinny, a lower value may blur the characters or allow overlapping of characters. A number of algorithm have been reported in the literature for finding best threshold. Some algorithms use an adaptive (and variable) threshold for binirization. Some algorithms work in presence of image in texts. Pathak uses a very simple algorithm based on image histogram. It is not the best, but works fine most times for current purpose.

The threshold finding algorithm:

1. Let hst is the histogram of the gray image
2. Replace any zeros in hst by a value of previous index. [set index 1 value to 1]
3. Apply a 3 point averaging filter. [add current and next 3 index values]
4. Now, if still any zeros are there, set them to 1 (to avoid log of zero)
5. Let it be hstnew
6. Take logarithom of hstnew, $hlog = \log(hstnew)$
7. Now, partition hlog into two parts (1 to 127, 128 to 256) and find the peak and its index in each partition.
8. Let them be $mx1$, $mx11$, $mx2$, $mx21$ [i-'s are the indices]
9. The optimum threshold is $hopt = (mx1*mx11 + mx2*mx21)/(mx1 + mx2)$;

Otsu's method

MATLAB 7 uses this method for finding threshold. The details can be found in:

N. Otsu. *A threshold selection method from gray-level histograms*. IEEE Trans. on System, Man and Cybernetics, 9(1):62--66, 1979.