OPEN ACCESS

sensors

ISSN 1424-8220

www.mdpi.com/journal/sensors

Supplementary Information

Optimization of a Cell Counting Algorithm for Mobile Point-of-Care Testing Platforms. *Sensors* 2014, *14*, 15244-15261

DaeHan Ahn [1], Nam Sung Kim [2], SangJun Moon [3,†], Taejoon Park [1,†,*] and Sang Hyuk Son [1]

[1] Real-Time Cyber-Physical Systems Laboratory, Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu 711-873, Korea; E-Mails: dahan@dgist.ac.kr (D.H.A.); son@dgist.ac.kr (S.H.S.)

[2] Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706, USA; E-Mail: nskim3@wisc.edu

[3] Cybernetics Laboratory, Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu 711-873, Korea; E-Mail: nanobiomems@dgist.ac.kr

† These authors contributed equally to this work.

* Author to whom correspondence should be addressed; E-Mail: tjpark@dgist.ac.kr; Tel.: +82-53-785-6313.

Abstract: In our optimized cell counting algorithm, a cell positioning and counting results are computed by applying an optimized cell image library and an approximation technique for the evaluation of normalized cross-correlation (NCC) values, both of which employ an appropriate threshold value to identify cells in a blood sample image. The runtime (and energy-efficiency) of our algorithm is demonstrated using two programs, each running on Matlab for Windows and an Android™ platform, to process six in-line holographic images.

Keywords: Mobile Point-Of-Care Testing; Optimized Cell Counting; Normalized Cross-Correlation (NCC); In-Line Holographic Image

1. Introduction

Our optimized algorithm (i) computes NCC values for the entire blood sample image in a zig-zag manner by applying cell images in the optimized library; (ii) by using our approximation approach, marks and counts a point as a cell if the maximum of NCC values for the point exceeds a threshold; and (iii) reports the number of marked points as the count of cells.

The optimized library is built from a randomly-generated library by eliminating duplicated or similar cell images that incur very small loss in counting accuracy if removed. Moreover, the threshold value is lowered to compensate the loss in accuracy. This optimized library is generally applicable to any blood sample images.

NCC values are computed between two images, i.e., a blood sample image and one of the cell images in the optimized library. The computed NCC value ranges from −1.0 to +1.0; the closer the NCC value is to −1.0 or +1.0, the stronger the two images are correlated. Since a cell usually appears as a cluster of points (having a maximum size of 3 × 3), our algorithm skips NCC evaluations for neighboring points according to a heuristic pattern (presented in Table 1 of the paper).

This joint optimization of the cell image library and the NCC-based counting process leads to much shorter runtime with negligible impact on counting accuracy. Six in-line holographic images are used to evaluate the runtime (and energy-efficiency) of our optimized algorithm. These images are tested according to a procedure, "How to run the software" that explains how to execute the two programs implementing our algorithm on two different platforms, Windows and Android™. Each program is explained in detail using annotations.

2. Legends for The Source Code, Cell Images in The Library, and Blood Sample Images

- Matlab code files, **.m**: **CellCount.m** containing the Matlab source code for optimized cell counting based on the NCC method.
- Android code files, **.java**: **MainActivity.java** containing the Java source code for optimized cell counting that runs on an Android™ platform.
- 20 cell images in the optimized library: **CellType-1.tif**, **CellType-2.tif**, **CellType-3.tif**, …, **CellType-20.tif.**
- 6 blood sample images: **image1.tif**, **image2.tif**, **image3.tif**, **image4.tif**, **image5.tif**, and **image6.tif.**

3. How to Run The Software

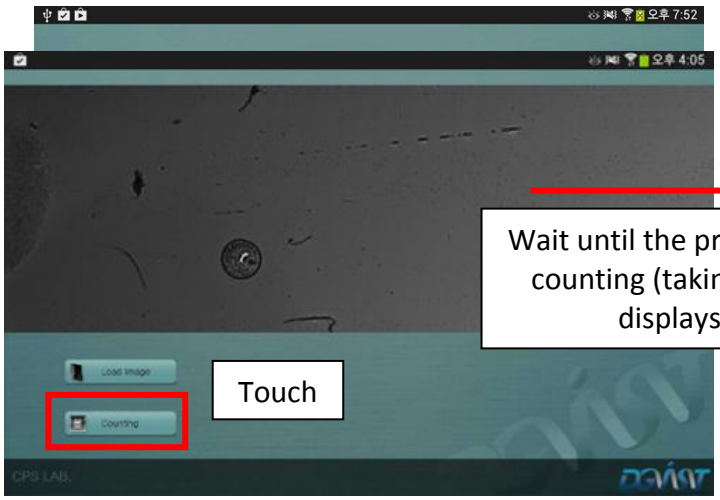
3.1. Matlab Code

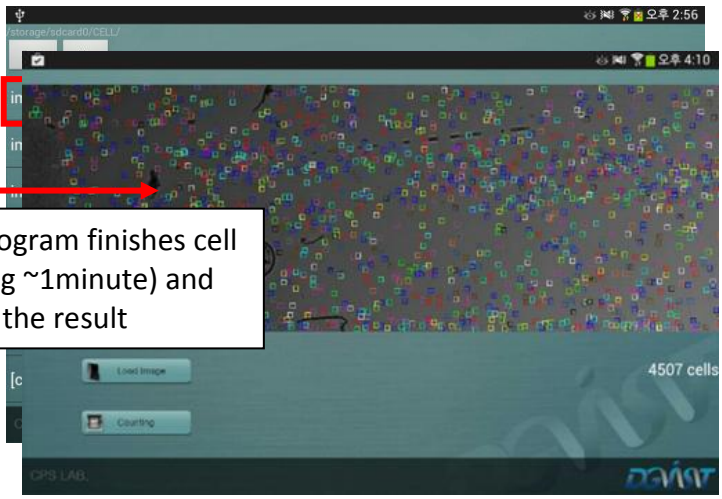
- In Matlab, type: **CellCount**.
- A file selector window appears to ask the user to select a **.tif** blood sample image.
- Select one of the blood sample images.
- The program counts the number of cells and display the counting result.

3.2. Android Code

- In an Android™ device, create two new folders, **root\CELL** and **root\CELL\cellLib**.
- Download from the Google store, and Install, the *OpenCV Manager* application.
- Install the cell counting application, **cell.apk** (provided) using the file manager.
- Copy blood sample images to **root\CELL** and cell images to **root\CELL\cellLib**.
- Execute the cell counting application.

- Touch the <Load Image> button and select a blood sample image.





- Touch the <Counting> button and wait until the program finishes cell counting and displays the result.

4. Source Code

4.1. CellCount.m

```
%%  Initializing Testing Image with environment.
% Close/Clear Working Environment.
close all;
clear all;

% Selecting testing image file and save file name and path.
[fileName,pathName] = uigetfile('*.tif','Select Testing TIF image file');

% Clock started to observe runtime, this is in seconds.
tic;

% Display message to user.
display 'Initializing sample library images and testing image......';

% Colors for marking cells on result image.
color = ['r','g','b','c','m','y','r','g','b','c','m','y','r','g','b','c','m','y','r','g'];
% shape for marking different type of cells.
shape = ['*','*','*','*','*','*','o','o','o','o','o','o','*','*','*','*','*','*','o','o'];

%% Preparing image to be processed.

% Getting image to memory.
Filter_image = imread(fileName);

% Getting size of the image.
[numRow numCol]= size(Filter_image);


%%  Initializing Sample Library Image with environment.
% Path to cell library.
```

```
LibPath = strcat(pathName,'CellLib\');

% size of the sample library image(it's 9 by 9 sample image, but we kept 8,
% because 0 to 8.
LibSize = size(imread(strcat(LibPath,'CellType-1.tif')))-1;

% One side of sample image is taken.
LibSize = LibSize(1,1);

%% Tests how many cell types are in the cell library.
numbOfTypes = 0;
IsExist = 1;
while(IsExist)
    if(exist(strcat(LibPath,['CellType' '-' num2str(IsExist) '.tif']),'file') == 0)
        numbOfTypes = IsExist - 1;
        IsExist = 0;
    else
        IsExist = IsExist + 1;
    end
end
clear IsExist;

%% Initializing Graphical Window For Result.
set(0,'Units','pixels')
scrsz = get(0,'ScreenSize');
figure('Position', [scrsz(1) scrsz(2) scrsz(3) scrsz(4)]);
imshow(Filter_image,'InitialMagnification','fit');
title('Counting image');
hold on;

%% Preparing Counting / Locating process.

% Specified Cell Radius is 2.
radius = 2;

% Preparing Position Data map that holds onto location result.
% In order to analyze the edges without errors, added sample size.
posData = zeros([numRow numCol] + LibSize);

cellTracker = zeros([numRow numCol]);

% finding center of the cell
matrixShift = 4;

% Correlation threshold number, used in testing similarity.
threshold = 0.616;

corValues = zeros(numbOfTypes,1);
%% Preparing Normalized 2D Cross Correlation Map.

% Reserving space to hold onto the Normalized 2D Cross Correlaiton Map.
% Number of maps varies based on number of cell types.
Z = zeros((numRow+LibSize),(numCol+LibSize),numbOfTypes);

totalTime = toc;
disp( strcat('setting time: ', num2str(totalTime), ' seconds.'));
cellcolor=zeros(numbOfTypes);
cellcolor=double(cellcolor);
%  Generates the Normalized 2D Cross Correlaiton Map.
for i = 1:1:numbOfTypes
    Libimage = imread(strcat(LibPath,['CellType' '-' num2str(i) '.tif']));
    Z(:,:,i) = normxcorr2(Libimage,Filter_image); % CorNumb: min -1 ~ max 1
end
%clear Libimage;

totalTime = toc;
disp( strcat('norm1 time: ', num2str(totalTime), ' seconds.'));

%% Starting actual counting and cell locating process started.
display 'Counting in progress. Please Wait......';
```

```
%% Starting actual counting and cell locating process started.
display 'Counting in progress. Please Wait......';
% Row loop moving in width.
   for i = 1:1:numRow;

        % Column loop moving in height.
        for j = 1:2:numCol;

            %center is at half of the cell (x coordinate)
            cellCenterY = j+matrixShift;  % Z matrix position, in order to get correlated answer
            aroundCellLowerX = j-radius; % in order to see if any of around pixel is filled.
            aroundCellUpperX = j+radius; % in order to see if any of around pixel is filled.
            if (aroundCellLowerX < 1)
               aroundCellLowerX = 1;
            end

            cellCenterMatchY = aroundCellLowerX:aroundCellUpperX; % Filling matrix, this matrix is used to
mark/check
                                %if the pixel is filled or not

            %center is at half of the cell (y coordinate)
            cellCenterX = i+matrixShift;
            aroundCellLowerY = i-radius;
            aroundCellUpperY = i+radius;
            if (aroundCellLowerY < 1)
                aroundCellLowerY = 1;
            end

            %Filling matrix, row
            cellCenterMatchX = aroundCellLowerY:aroundCellUpperY;

            % Finding best match in cell types.
            corValues = Z(cellCenterX,cellCenterY,(1:numbOfTypes));

            % Best match in type with it's type number.
            [corMax cellType] = max(corValues);


            %else
                if ((corMax > threshold) && (posData(i,j) == 0))
                posData(cellCenterMatchX,cellCenterMatchY) = 1;
                cellTracker(i,j) = 1;
                plot(j,i,[color(cellType) ':' shape(cellType)]);
            end
        end

    end

totalTime = toc;
disp( strcat('match2 time: ', num2str(totalTime), ' seconds.'));
disp( strcat('end'));


%% Preparing Normalized 2D Cross Correlation Map.

% Reserving space to hold onto the Normalized 2D Cross Correlaiton Map.
% Number of maps varies based on number of cell types.

%  Generates the Normalized 2D Cross Correlaiton Map.
%% Graphing & Marking Started.

graphGap = 1.0; % mm
pixelSize = [9 5.2]; % Sony CCD 9um/pixel, Small CCD 5.2um
sectionPixel = round(graphGap*1000/pixelSize(1)); % 111
sNumber = round(numCol/sectionPixel); % 30

rowSum = sum(cellTracker,1); % 1 x numCol
%clear cellTracker;
```

```
graphGap_Sum = zeros(1,sNumber);
for j = 1:1:(sNumber-1);
    graphGap_Sum(1,j) = sum( rowSum(1,((j-1)*sectionPixel+1):(j*sectionPixel) ));
end
graphGap_Sum(1,sNumber) = sum( rowSum(1,((sNumber-1)*sectionPixel+1):numCol ));

totalCell = sum(graphGap_Sum);
distgap = 1:1:sNumber;
figure,
plot(distgap, graphGap_Sum', 'r:*');

% Use following line to see auto count result.
title(['Total Cell count (6ul) = ', num2str(totalCell), ', (',num2str(totalCell/6),'cells/ul)']);

set(gca,'XTick',0:graphGap:sNumber);
set(gca,'XTickLabel',0:graphGap:sNumber);
xlabel('Distance (mm)');

% Clock ends.
totalTime = toc;

%% Display Result.
disp( strcat('Runtime: ', num2str(totalTime), ' seconds.'));
disp( strcat('Number of Cells counted: ', num2str(totalCell), ' cells'));
disp('Counting Finished.');
```

4.2. MainActivity.java

```
package com.example.cell;

import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.Drawable;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.HorizontalScrollView;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.Core;
import org.opencv.core.Core.MinMaxLocResult;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.highgui.Highgui;
import org.opencv.imgproc.Imgproc;
```

```
@SuppressWarnings("unused")
public class MainActivity extends Activity {
String filePath;
int col=0,row=0;
int[][] resul;
TextView mText;
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // load main layout (activity_main.xml)
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //set button and listening
        Button btn1=(Button)findViewById(R.id.button1);
        Button btn2=(Button)findViewById(R.id.button2);
        btn2.setBackgroundResource(R.drawable.btnbg22);
        btn2.setEnabled(false);
        btn1.setOnClickListener(mClickListener);
        btn2.setOnClickListener(mClickListener);

        mText=(TextView)findViewById(R.id.TV);

        //call openCV library
        if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_2, this, mOpenCVCallBack))
        {
            Log.e("TEST", "Cannot connect to OpenCV Manager");
        }
    }

    //callback openCV library
    private BaseLoaderCallback  mOpenCVCallBack = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                    case LoaderCallbackInterface.SUCCESS:
                    {} break;
                    default:
                    {
                        super.onManagerConnected(status);
                    } break;
                }
        }
        };

    //set action bar
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    //button click listener
    Button.OnClickListener mClickListener = new Button.OnClickListener(){
        public void onClick(View v){
            switch(v.getId()){
            case R.id.button1:

                Intent intent =new Intent(MainActivity.this,fileview.class); //call file view
                //intent listener from file view
                startActivityForResult(intent,0);

                //button2 enable
                Button btn2=(Button)findViewById(R.id.button2);
                btn2.setBackgroundResource(R.drawable.btnbg2);
                btn2.setEnabled(true);
                break;

            case R.id.button2:
```

```
            TextView tv=(TextView)findViewById(R.id.textView1);
            tv.setText("Couning...Plase wait");
            norcorr(filePath);
            break;

        }
    }

};
//intent listener
//when a file is clicked in fileview intent, it get the message (requestcode, resultcode and filepath)
protected void onActivityResult(int requestCode,int resultCode, Intent data){
    switch(requestCode){
    case 0 :
        if(resultCode == RESULT_OK){
            filePath=data.getStringExtra("text_out");
            //show file location
            Toast.makeText(MainActivity.this, filePath, Toast.LENGTH_SHORT).show();
            //call img function which show the file and null graph
            img();
        }
        break;
    }
}

//load and show source1 image and null graph
public void img(){
      //convert file name to source1.jpg
      Mat img=Highgui.imread(filePath);
     Highgui.imwrite(Environment.getExternalStorageDirectory().getAbsolutePath()+"/CELL/cellLib/source1.jpg",
img);
      //load source image
      ImageView image = (ImageView)findViewById(R.id.imgView);
      Bitmap bMap=BitmapFactory.decodeFile(Environment.getExternalStorageDirectory().getAbsolutePath()
      +"/CELL/cellLib/source1.jpg");
      image.setImageBitmap(bMap);
}

//check neighbor point
//if this point and neighbor point are not marked, it return 0
public int summ(int[][] res,int i,int j){
     if(i<2){i=2;} if(j<2){j=2;}
     if(i>row){i=row;} if(j>col){j=col;}
     int ret=res[i-2][j-2]+res[i-2][j-1]+res[i-2][j]+res[i-2][j+1]+res[i-2][j+2]+res[i-1][j-2]
     +res[i-1][j-1]+res[i-1][j]+res[i-1][j+1]+res[i-1][j+2]+res[i][j-2]+res[i][j-1]+res[i][j+1]
     +res[i][j+2]+res[i+1][j-2]+res[i+1][j-1]+res[i+1][j]+res[i+1][j+1]+res[i+1][j+2]+res[i+2][j-2]
     +res[i+2][j-1]+res[i+2][j]+res[i+2][j+1]+res[i+2][j+2]+res[i][j];
     return ret;
}


public void norcorr(String source1){
      TextView tv=(TextView)findViewById(R.id.textView1);
      //in order to know, the least one cell is needed.
      String source=Environment.getExternalStorageDirectory().getAbsolutePath()
      +"/CELL/cellLib/CellType-1.tif";
      //load source1 and source image
      Mat img=Highgui.imread(source1);
      Mat temp= Highgui.imread(source);
      //calculate size
      int width = img.width() - temp.width() + 1;
      int height = img.height() - temp.height() + 1;
      row=height-3;
      col=width-3;
      //make result matrix
      Mat result = new Mat(width,height,CvType.CV_32FC1);
      //select calculate type = normalized cross correlation
      int mathod = Imgproc.TM_CCOEFF_NORMED;
      //res save the NCC result, max is the max value on a point, shape set cell color
      int[][] res=new int[result.width()][result.height()];
```

```
        resul=new int[result.width()][result.height()];
        double[][] _max=new double[result.width()][result.height()];
        int[][] shape=new int[result.width()][result.height()];

        ////////////////////////////////////////////////////
        //setting threshold value
         double threshold=0.615;
        ////load library (20 cell images)
        for(int k=1;k<=20;k++){
           Mat
temp2=Highgui.imread(Environment.getExternalStorageDirectory().getAbsolutePath()
            +"/CELL/cellLib/CellType-"+k+".tif");
           //NCC computation
           Imgproc.matchTemplate(img, temp2, result, mathod);
           //heuristic pattern 1 (i=1, j=j+2)
           for (int i=0;i<result.height();i++){
               for(int j=0;j<result.width();j=j+2){
                      //get NCC value
                      double a[]=result.get(i,j);
            //mark the point if the value is greater than threshold and if the point is not
marked
                      if(a[0]>threshold){
                             //identifying NCC value
                             int trac=summ(res,i,j);
                             if(trac==0){
                                    //marking the point (i,j)
                                    res[i][j]=1;
                             }
                             //define the cell shape
                             if(_max[i][j]<a[0]){
                                    _max[i][j]=a[0];
                                    shape[i][j]=k;
                             }
                      }

              }
           }
        }

         //draw rectangle on the marked point
         for (int i=0;i<result.height();i=i+1){
            for(int j=0;j<result.width();j=j+1){
               if(res[i][j]>0){
                  switch(shape[i][j]){
                     case 1:
                       Core.rectangle(img, new Point(j,i),
                       new Point(j + temp.cols(),i + temp.rows()), new Scalar(0, 255, 0));
                     break;
                     case 2:
                       Core.rectangle(img, new Point(j,i),
                       new Point(j + temp.cols(),i + temp.rows()), new Scalar(255, 0, 0));
                     break;
                     case 3:
                       Core.rectangle(img, new Point(j,i),
                       new Point(j + temp.cols(),i + temp.rows()), new Scalar(0, 0, 255));
                     break;
                     case 4:
                       Core.rectangle(img, new Point(j,i),
                     new Point(j + temp.cols(),i + temp.rows()), new Scalar(255, 255, 0));
                     break;
                     case 5:
                       Core.rectangle(img, new Point(j,i),
                       new Point(j + temp.cols(),i + temp.rows()), new Scalar(0, 255, 255));
                     break;
                     case 6:
                       Core.rectangle(img, new Point(j,i),
                       new Point(j + temp.cols(),i + temp.rows()), new Scalar(255, 0, 255));
                     break;
                     case 7:
                       Core.rectangle(img, new Point(j,i),
```

```
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(150, 255, 0));
                  break;
                  case 8:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(0, 255, 150));
                  break;
                  case 9:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(150, 150, 255));
                  break;
                  case 10:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(255, 255, 255));
                  break;
                  case 11:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(163, 53, 200));
                  break;
                  case 12:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(178, 72, 12));
                  break;
                  case 13:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(255, 50, 100));
                  break;
                  case 14:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(50, 255, 100));
                  break;
                  case 15:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(50, 100, 255));
                  break;
                  case 16:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(100, 255, 50));
                  break;
                  case 17:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(0, 50, 100));
                  break;
                  case 18:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(0, 0, 0));
                  break;
                  case 19:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(100, 100, 100));
                  break;
                  case 20:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(83, 60, 23));
                  break;
                  default:
                    Core.rectangle(img, new Point(j,i),
                    new Point(j + temp.cols(),i + temp.rows()), new Scalar(255, 255, 255));
                  break;
                }
              }
            }
          }
        //save the marked source image
        Highgui.imwrite(Environment.getExternalStorageDirectory().getAbsolutePath()+"/CELL/cellLib/Result.jpg",
img);

        //call the result show function
        resul=res;
        results(resul);
}
```

```
//show the result
void results(int[][] res){
       //display the result image
       ImageView image = (ImageView)findViewById(R.id.imgView);
       Bitmap bMap=BitmapFactory.decodeFile(Environment.getExternalStorageDirectory().getAbsolutePath()
       +"/CELL/cellLib/Result.jpg");
       image.setImageBitmap(bMap);

       int sum=0;
       Log.v("res",""+sum);
       //count how many cells
       for (int i=0;i<row;i++){
          for (int j=0;j<col;j++){
             if (res[i][j]==1){
                sum++;
             }
          }
       }
       //print count result
       TextView tv=(TextView)findViewById(R.id.textView1);
       tv.setText(sum+" cells");
       Log.v("res",""+sum);

    }

}
```

Cell Images in The Optimized Library and Blood Sample Images:

Following images are also supplied as separate files.

CellType-1.tif ~ CellType-20.tif.

image1.tif

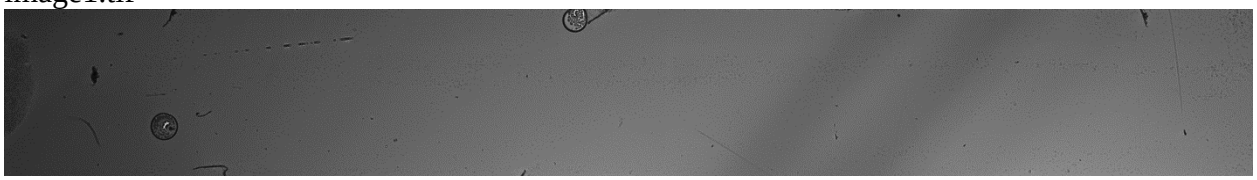


image2.tif

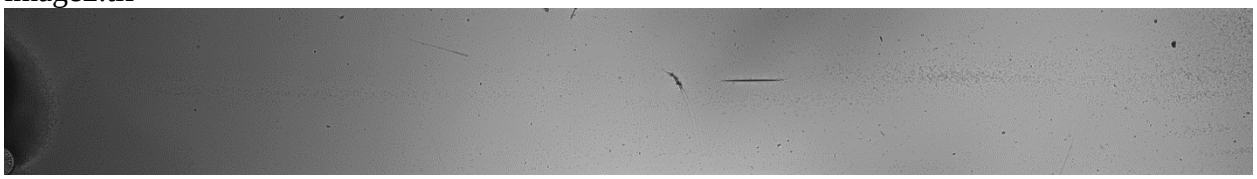


image3.tif

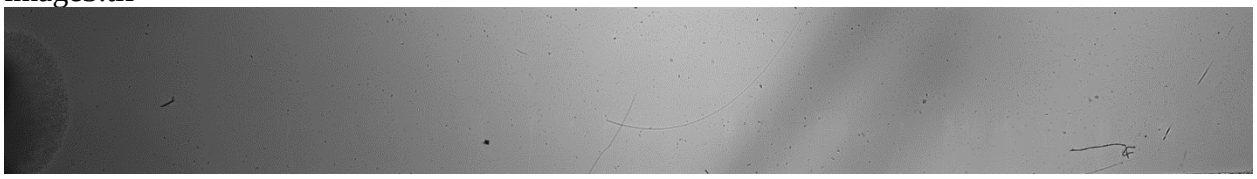


image4.tif

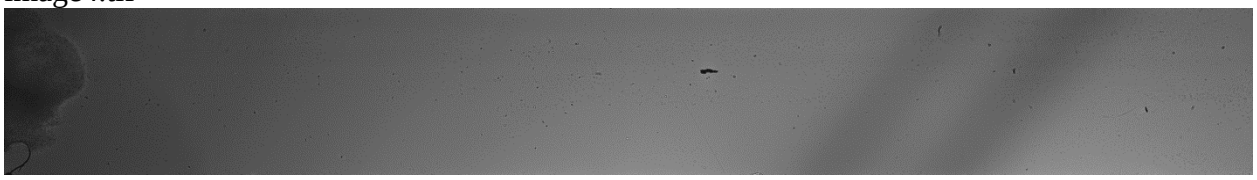


image5.tif

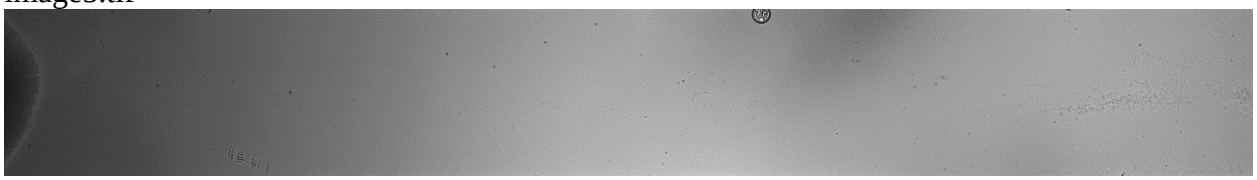


image6.tif