Image Analysis for Quantifying Sickle Cell Adherence in Microfluidic Channel: Final Report

Muhammad Noman Hasan

Submitted: December 20th

Contents

1.1 Introduction

The term sickle cell disease (SCD) describes a group of inherited red blood cell disorders. People with SCD have abnormal hemoglobin, called hemoglobin S or sickle hemoglobin, in their red blood cells.

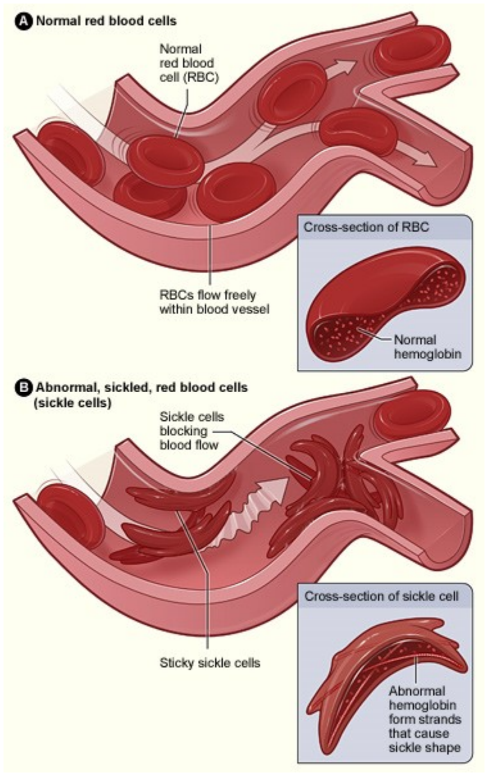
A Normal red blood cells
Normal red blood cell (RBC)
RBCs flow freely within blood vessel
Cross-section of RBC
Normal hemoglobin
B Abnormal, sickled, red blood cells (sickle cells)
Sickle cells blocking blood flow
Sticky sickle cells
Cross-section of sickle cell
Abnormal hemoglobin form strands that cause sickle shape

Figure 1: Figure A shows normal red blood cells flowing freely in a blood vessel. The inset image shows a cross-section of a normal red blood cell with normal hemoglobin. Figure B shows abnormal, sickled red blood cells blocking blood flow in a blood vessel. The inset image shows a cross-section of a sickle cell with abnormal (sickle) hemoglobin forming abnormal stiff rods.

Source: https://www.nhlbi.nih.gov/health/health-topics/topics/sca

We have developed microfluidic channel that can mimic capture the red blood cell (RBC) with the help of surface functionalization. The experiment is performed under microscope for visual inspection. After the completion of the experiment the microfluidic channel is scanned and imaged by "image stitching". After aqcuring the large image by "image stitching", we need to manually quantify the percentage of sickled and healthy RBCs. This is a very painstacking and laborous task. So the purpose of this image analysis project is to develop a code that can quantify the healthy and sickled RBC with a reasonable degree of accuracy. The project can be devide into the following sus-aims -

- To open and read a very large size image (approximately, 35000 x 14000).
- To split the large image file into smaller segments.
- To develope a library of healthy and sickled RBCs.
- To train the code to identify healthy and sickled RBCs with a reasonable accuracy.
- Identify and quantify the RBCs from the segmenteg images (section 2).

1.2 Project workflow

The objective of this work, as mentioned above, is to identify, differentiate and quantify healthy and unhealthy RBCs from images obtained from experiments of RBC adherence in microfluidic channel. The resulting images are very large (~ 35000x14000). To successfully cary out the objective of this work, the project has been devided into 5 sub-aims (see, Introduction). The figure below illustrates the workflow for the "Image Analysis" for this project.

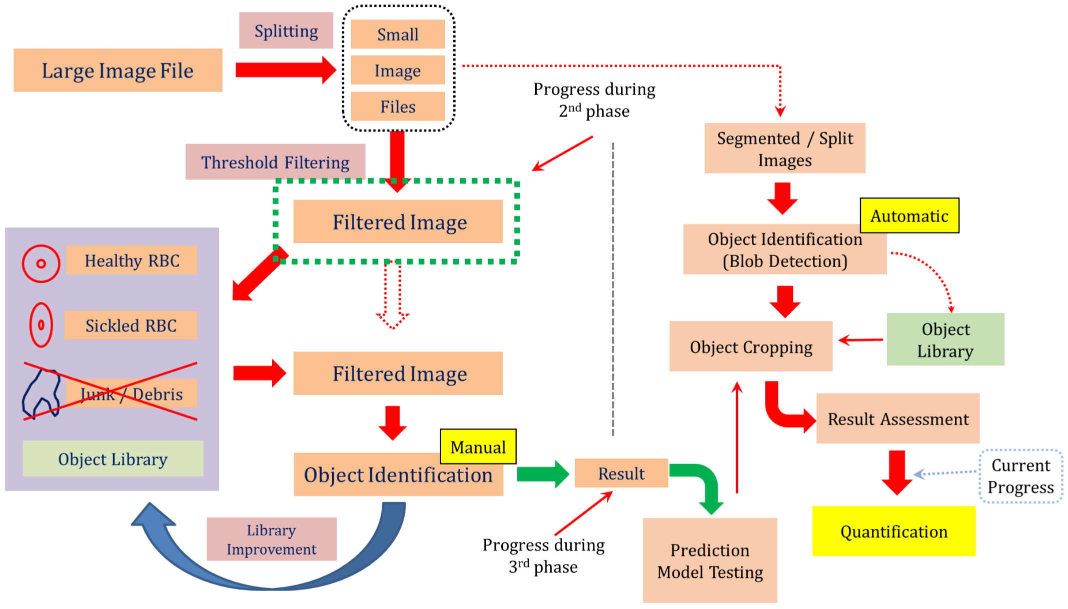

Figure 2: A semi-supervised learning approach designed for this image analysis

The complete workflow is performed through a number of seperatly developped codes. The development of code was started using R programming Language, initially. But the code development was facing huge obstructions due to the sheer size of the image files. Due to the setback faces, while using R as the code development platform, the code development was shifted to Python. The List of codes and their functiones will be provided in later section of the report. The breakdown of the steps in the workflow is provided below -

- Large images (obtained from experiments) are imported into
- Splitting up the large image into smaller image segments
- Thresholding the split images (to mask out the unwanted background as well as debris)
- Object identification from the filtered images
    - Manual cropping of cell images from the split images
    - Automatic identification (blob detection) and image cropping
- Building up object library

    – Using images from manually cropped images

    – Using automatically cropped images

  – Prediction model performance testing using object libraby (manually cropped)

  – Application of the prediction models with object library (automatically cropped)

  – Result assessment

  – Quantification

The later sections of the report will provide a walkthrough of how each sub-aims has been achieved

1.3 Sub-aim 1 & 2: Open, read and split large image files -

This following section of the code splits the large “image stitched” files into smaller sections and saves in the disk. The large image is splitted into 25 vetrical and 16 horizontal slices, yelding 400 split images with a resolution of 1500 x 1000 pixels each (approximately).

```
#================================================================================
#
# This following is the definition of the fuction that splts the images
#
#================================================================================

def crop(infile,outfile,vertical_count, horizontal_count):
    im = Image.open(infile)
    img_width, img_height = im.size

    height = img_height/vertical_count
    width = img_width/horizontal_count

    # The followings are index for crop box determination
    i = 0
    j = 0

    # this is an index for saving cropped images
    k = 0


### Beginning of the loop

    for p in range(0,vertical_count, 1):
        for q in range(0,horizontal_count, 1):
            crop_box = (j, i, j+width, i+height)
            img_obj = im.crop(crop_box)
            img_obj.save(os.path.join(outfile,"IMG-%s.png" % k))

            k += 1
            # initilizes the y-coordinate for the next box of the same row
            j += width

        # initilizes the x-coordinate for the next rows of images
        # putting "j = 0" initializes the position to left most image crop
        i += height
        j = 0

### End of the loop

#==============================================================================

# The image file path is

InputFilePath = './figs/2016-09-21-Laminin-55-LN-10-mul-1.jpg'
OutputDir = './figs/WorkingImageSlices/'

#==============================================================================
#
# The splitting parameters are preset at this point
#
#==============================================================================

vertical_count = 16
horizontal_count = 25

crop(InputFilePath,OutputDir,vertical_count,horizontal_count)
```

1.4 Sub-aim 3: To develope a library of healthy and sickled RBCs

1.4.1 Step 1: Thresholding and elimination of non-celullar objects -

This following of the code reads all the saved image (saved after splitting) from the directory and applyies filter on them.

- Otsu's method has been used for the thresholding
- The algorithm reduces a graylevel image to a binary image.
- it calculates the optimum threshold separating the two classes, the foreground pixels and the background pixels.

```

#==============================================================================
#
# Otsu Thresholding for Image Analysis
#
# Files will be read, processed and saved in loop
#
#==============================================================================

# Loop for file list in a directory


# files = [f for f in os.listdir(dirToScreens) if path.isfile(f)]

current_dir = ("C:/Noman/Case Western Reserve University/Courses/2016-3-Fall/"
               "DSCI-451/Git/16f-dsci351-451-mnh38/assignments/"
               "451-semester-projects")
dirToScreens = current_dir + '/figs/WorkingImageSlices/'
dirToSave = current_dir + '/figs/SlicedImageThresholdOtsu/'

file_list = os.listdir(dirToScreens)

#==============================================================================
# "thresh" is a vector which contains the threshold values. The following
# "for loop" runs throught the slices image file and stores the otsu threshold
# values in the "thresh" list.
#
# This list is then plotted to see the distribution of threshold values for
# all the sliced images. Then the median of that distribution is used for
# further processing
#==============================================================================

thresh_list = []

for i in file_list:
    img_abs_file_name = dirToScreens + i
    imgPath4otsu = img_abs_file_name
    imgRead4Thesh = io.imread(imgPath4otsu)
    thresh_list.append(threshold_otsu(imgRead4Thesh))

### End of for loop

sns_plot = sns.distplot(thresh_list)
plt.savefig("./figs/split_image_threshold_density.png")

thresh = np.median(np.array(thresh_list))

#==============================================================================
# The following "for loop" applies the otsu filter on the sliced images
#==============================================================================

for i in file_list:
    img_abs_file_name = dirToScreens + i
    imgPath4otsu = img_abs_file_name
    imgRead4Thesh = io.imread(imgPath4otsu)

    binary = imgRead4Thesh > thresh
    img_abs_file_name_2_save = dirToSave + 'Otsu-Thresh-'+i
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,binary)

### End of for loop
```

The thresholding using Otsu's method has a picture-to-picture varation. Some of the pictures are very well filtered and some other are not acceptable for further processing. Below is an example for both -

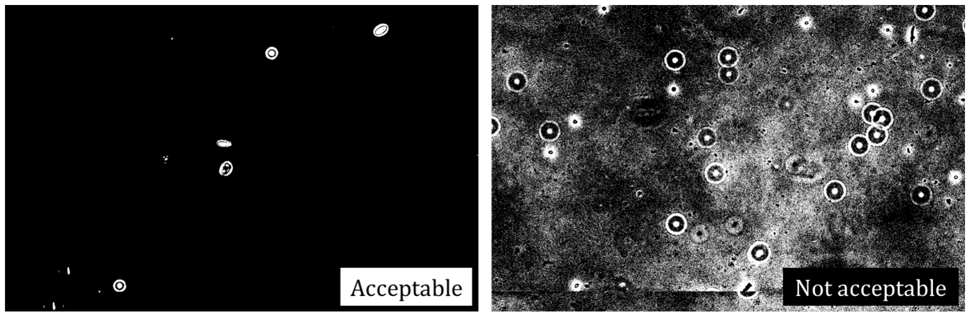

Figure 3: An example of picture-to-picture variation of thresholding using Otsu's Method. A comparison of thresholding between an acceptable result and an unacceptable result for Otsu's method.

1.4.2 Step 2: Elimination of image-to-image variation of thresholding (Otsu's method)

To eliminate the discrepency of threshholding and leveling the picture to picture variation, The distribution of the optimum threshold value computed by the Otsu's algorithm has been plotted. Below is the density plot of the threshold values -

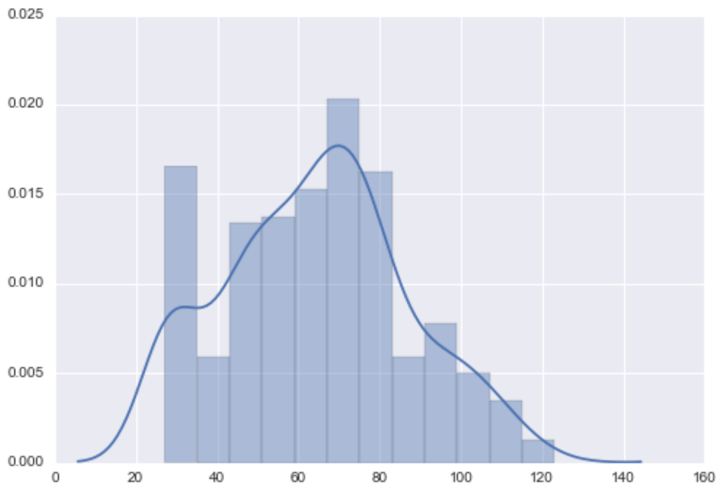

Figure 4: The distribution of the optimum threshold value computed by the Otsu's algorithm for all the split images

The distribution shows the picture-to-picture variation of the computed optimum threshold value fr all the images. After calculating the optimum threshold value for individual pictures, the median of their distributaion is selected as a global thresholging parameter for all the pictures. Thus, the picture to picture variation of the thresholding is eliminated

1.4.3 Step 3: Library buildup: Image acquisition and library proliferation

After achieving an acceptable degree of thresholding, images of healthy and sickled RBCs has been cropped manually. A set of 100 sickled RBCs and a set of 50 healthy RBCs has been build manually. After building the base library, the images has been rotated at 90, 180, and 270 degrees and saved to proliferate the image libraby. Below is the code for image rotation -

```
#==============================================================================
#
# This section of the code will read the files of sickled image libraary and
# rotate and save them to increase the librarby size
#
#
#==============================================================================


current_dir  = ("C:/Noman/Case Western Reserve University/Courses/2016-3-Fall/"
               "DSCI-451/Git/16f-dsci351-451-mnh38/assignments/"
               "451-semester-projects")

dirToScreens_sickled = current_dir + '/figs/ImageLibrary/SickledRBC/'
dirToSave_sickled = current_dir + '/figs/ImageLibrary/SickledRBC_Processed/'

dirToScreens_healthy = current_dir + '/figs/ImageLibrary/HealthyRBC/'
dirToSave_healthy = current_dir + '/figs/ImageLibrary/HealthyRBC_Processed/'


file_list_sickled = os.listdir(dirToScreens_sickled)

#==============================================================================
# This following section will appaend 'a' at the end of all the base library
# images
#==============================================================================

for i in file_list_sickled:
    img_abs_file_name = dirToScreens_sickled + i
    imgPath4Append_a = img_abs_file_name
    imgRead4Append_a = io.imread(imgPath4Append_a)

    img_abs_file_name_2_save = dirToSave_sickled + i.replace('.png', '') + 'a' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgRead4Append_a)

#==============================================================================

#==============================================================================
# This following section will rotate all the base library images to 90 degree
#==============================================================================

for i in file_list_sickled:
    img_abs_file_name = dirToScreens_sickled + i
    imgPath4rotation = img_abs_file_name
    imgRead4rotation = io.imread(imgPath4rotation)

    imgTransform_rotation = skimage.transform.rotate(imgRead4rotation, 90)

    img_abs_file_name_2_save = dirToSave_sickled + i.replace('.png', '') + 'b' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgTransform_rotation)

#==============================================================================

#==============================================================================
# This following section will rotate all the base library images to 180 degree
#==============================================================================

for i in file_list_sickled:
    img_abs_file_name = dirToScreens_sickled + i
    imgPath4rotation = img_abs_file_name
    imgRead4rotation = io.imread(imgPath4rotation)

    imgTransform_rotation = skimage.transform.rotate(imgRead4rotation, 180)

    img_abs_file_name_2_save = dirToSave_sickled + i.replace('.png', '') + 'c' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgTransform_rotation)

#==============================================================================

#==============================================================================
# This following section will rotate all the base library images to 270 degree
#==============================================================================

for i in file_list_sickled:
    img_abs_file_name = dirToScreens_sickled + i
    imgPath4rotation = img_abs_file_name
    imgRead4rotation = io.imread(imgPath4rotation)

    imgTransform_rotation = skimage.transform.rotate(imgRead4rotation, 270)

    img_abs_file_name_2_save = dirToSave_sickled + i.replace('.png', '') + 'd' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgTransform_rotation)

#==============================================================================

#==============================================================================
#
# This section of the code will read the files of healthy image libraary and
# rotate and save them to increase the librarby size
#
#
#==============================================================================
file_list_healthy = os.listdir(dirToScreens_healthy)

#==============================================================================
# This following section will appaend 'a' at the end of all the base library
# images
#==============================================================================

for j in file_list_healthy:
    img_abs_file_name = dirToScreens_healthy + j
    imgPath4Append_a = img_abs_file_name
    imgRead4Append_a = io.imread(imgPath4Append_a)

    img_abs_file_name_2_save = dirToSave_healthy + j.replace('.png', '') + 'a' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgRead4Append_a)

#==============================================================================

#==============================================================================
# This following section will rotate all the base library images to 90 degree
#==============================================================================

for j in file_list_healthy:
    img_abs_file_name = dirToScreens_healthy + j
    imgPath4rotation = img_abs_file_name
    imgRead4rotation = io.imread(imgPath4rotation)

    imgTransform_rotation = skimage.transform.rotate(imgRead4rotation, 90)

    img_abs_file_name_2_save = dirToSave_healthy + j.replace('.png', '') + 'b' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgTransform_rotation)

#==============================================================================

#==============================================================================
# This following section will rotate all the base library images to 180 degree
#==============================================================================

for j in file_list_healthy:
    img_abs_file_name = dirToScreens_healthy + j
    imgPath4rotation = img_abs_file_name
    imgRead4rotation = io.imread(imgPath4rotation)

    imgTransform_rotation = skimage.transform.rotate(imgRead4rotation, 180)

    img_abs_file_name_2_save = dirToSave_healthy + j.replace('.png', '') + 'c' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgTransform_rotation)

#==============================================================================

#==============================================================================
# This following section will rotate all the base library images to 270 degree
#==============================================================================

for j in file_list_healthy:
    img_abs_file_name = dirToScreens_healthy + j
    imgPath4rotation = img_abs_file_name
    imgRead4rotation = io.imread(imgPath4rotation)

    imgTransform_rotation = skimage.transform.rotate(imgRead4rotation, 270)

    img_abs_file_name_2_save = dirToSave_healthy + j.replace('.png', '') + 'd' + '.png'
    #plt.plot(binary)
    mpimg.imsave(img_abs_file_name_2_save,imgTransform_rotation)

#==============================================================================
```

Below is an illustration of the library images for both healthy and sickled RBC

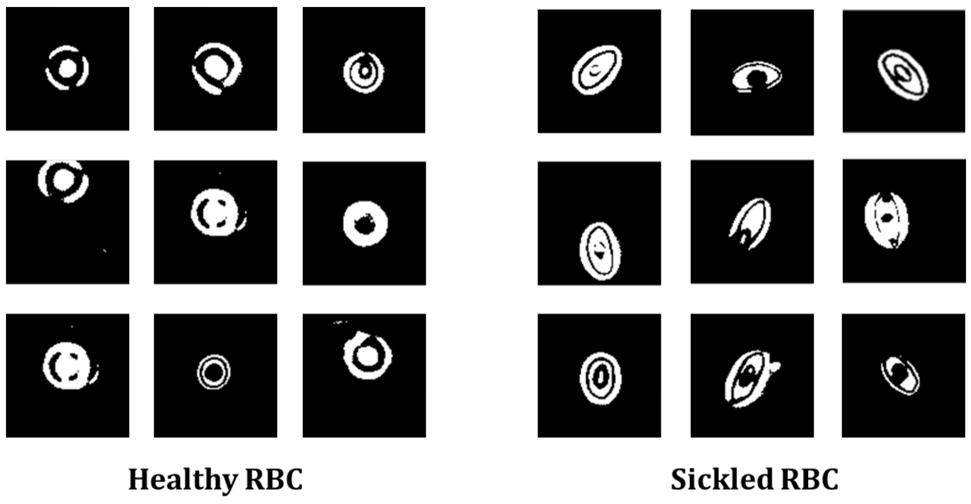

Figure 5: Example from the image libraby for both healthy and sickled RBCs (100x100, manually cropped)

1.5 Sub-aim - 4: To train the code to identify healthy and sickled RBCs with a reasonable accuracy

1.5.1 Step 1: Image check and resize

The following section of the code -

- Reads all the images from the disk, saved as library image
- Checks the size of the images (whether they are 100 x 100 pixel or not)
- If the image is not of the required size, it resizes them.

```
#==============================================================================
# This following section reads all the image files in the library and checks
# their size. If the fize size is not 100 x 100, itresizes the files to
# 100 x 100.
#==============================================================================

check_folder = ("C:/Noman/Case Western Reserve University/Courses/2016-3-Fall/"
                "DSCI-451/Git/16f-dsci351-451-mnh38/assignments/"
                "451-semester-projects/figs/ImageLibrary/HealthyRBC_Processed/")
files = fnmatch.filter(os.listdir(check_folder), '*.png')

### Function to change RGB files to greyscale

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

sz = (100,100)  # Target Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

### Function to resize images

def process_image(img_file):
    img = cv2.imread(img_file)
    img = rgb2gray(img)
    #r = 100.0 / img.shape[1]
    #dim = (100, int(img.shape[0] * r))
    img = cv2.resize(img, sz).astype('float32')
    return img

im = process_image(check_folder + 'hrbc-005a.png')
im.shape

image_size = 100  # Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

### Main fuction to resize the imagaes

def load_pic(folder, min_num_images,n):
  """Load the data for a single type."""
  image_files = os.listdir(folder)
  dataset = np.ndarray(shape=(n, image_size, image_size),
                       dtype=np.float32)
  print(folder)

  num_images = 0
  for image in image_files:
    image_file = os.path.join(folder, image)
    img_data= process_image(image_file)
    try:
      image_data =  (img_data - pixel_depth / 2) / pixel_depth
      if image_data.shape != (image_size, image_size):
        raise Exception('Unexpected image shape: %s' % str(image_data.shape))
      dataset[num_images, :, :] = image_data
      num_images = num_images + 1
    except IOError as e:
      print('Could not read:', image_file, ':', e, '- it\'s ok, skipping.')

  dataset = dataset[0:num_images, :, :]
  if num_images < min_num_images:
    raise Exception('Many fewer images than expected: %d < %d' %
                    (num_images, min_num_images))

  print('Full dataset tensor:', dataset.shape)
  print('Mean:', np.mean(dataset))
  print('Standard deviation:', np.std(dataset))
  return dataset

### End of "load_pic" function
```

1.5.2 Step 2: Data labeling, preparation of training and test dataset

The following section of the code -

- Creates label for both healthy and sickled RBC images. ("1" for healthy, "0" for sickled
- assigns the labels to the loaded images
- Merges the healthy and sickled image data
- The merged data is then splitted into “a training” and “a test” data set.

```
#==============================================================================
# In the following section, two arrays are created, one for the healthy and
# another for the sickled rbs image.
#
# Healthy cell images have been labelled as "1" and the sickled cell images
# have been as "0".
#==============================================================================

label_healthy = np.ones(200)
label_sickle = np.zeros(416)

data_merge = np.concatenate((train_datasets_h,train_datasets_s),axis=0)
label = np.concatenate((label_healthy,label_sickle),axis=0)

#==============================================================================
# The following section randomly splits the combined dataset into a test and
# a train dataset.
#==============================================================================

### test_size=0.3 means, 30% of the data has been used to creat the test
### dataset and 70% data has been used to create teh training dataset

X_train, X_test, y_train, y_test = train_test_split(data_merge, label, test_size=0.3, random_state=42)

#==============================================================================
# ## no need
# def randomize(dataset, labels):
```

```
#   permutation = np.random.permutation(labels.shape[0])
#   shuffled_dataset = dataset[permutation,:,:]
#   shuffled_labels = labels[permutation]
#   return shuffled_dataset, shuffled_labels
#
# X_train, y_train = randomize(X_train,y_train)
# X_test, y_test = randomize(X_test,y_test)
#==============================================================================

X_train = X_train.reshape(431,10000)
X_test = X_test.reshape(185,10000)
```

1.5.3 Step 3: Prediction modeling

The following section of the code -

- Prediction models are implemented.
- For each model, the accuracy of the model is evaluated.

```
### "clf"  in the name for the imported model, .LogisticRegression.
### This section is the model
### fit section

clf = linear_model.LogisticRegression(C=1e-3, max_iter=10000,n_jobs=-1)
clf.fit(X_train,y_train)

#X_test = test_dataset.reshape(len(test_dataset),len(test_dataset[0])*len(test_dataset[0]))

### Model predict section

pred_train= [(clf.predict(row.reshape(1,-1)))[0] for row in X_train]
pred_test = [(clf.predict(row.reshape(1,-1)))[0] for row in X_test]
print ('accuracy is : ',accuracy_score(y_test,pred_test))
print ('accuracy is : ',accuracy_score(y_train,pred_train))

### Checking
# y_test
### zip section
a = zip(y_test,pred_test)

np.sum(abs(y_test-pred_test))

### SVM = Support vector Machine classification
### SVC = Support vector classification

clf = SVC(C=15,random_state=42)
clf.fit(X_train, y_train)
pred_train= [(clf.predict(row.reshape(1,-1)))[0] for row in X_train]
pred_test = [(clf.predict(row.reshape(1,-1)))[0] for row in X_test]
print ('accuracy is : ',accuracy_score(y_test,pred_test))
print ('accuracy is : ',accuracy_score(y_train,pred_train))

### Chceking
```

```
# clf


### Random Forrest Classification model

clf = RandomForestClassifier(n_estimators=200,n_jobs=-1)
clf.fit(X_train, y_train)
pred_train= [(clf.predict(row.reshape(1,-1)))[0] for row in X_train]
pred_test = [(clf.predict(row.reshape(1,-1)))[0] for row in X_test]
print ('accuracy is : ',accuracy_score(y_test,pred_test))
print ('accuracy is : ',accuracy_score(y_train,pred_train))
```

1.5.4 Step 4: Interpret results

In the Statistical prediction/modeling section, three different classification model has been implemented to evaluate the training data set in terms of the accuracy of the "test" data. The models which were implemented are -

- Logistic Regression - test data accuracy: 85.41%
- Support Vector Classification - test data accuracy: 90.81%
- Random Forrest Classification - test data accuracy: 92.43%

1.5.5 Step 5: Challenge results

The test result can be challanged comparing with the accuracy on the train data and checking for overshoot. The accuracy of the train data is presented below -

- Logistic Regression - test data accuracy: 86.31%
- Support Vector Classification - test data accuracy: 94.43%
- Random Forrest Classification - test data accuracy: 100%

Based on the comparison between test data and train data accuracy, we can conclude that the prediction data is very much reasonable.

1.6 Sub-aim 5: Identify and quantify the RBCs from the segmenteg images (section 2)

In the previous section, we have manually identified objects from slip images and it was followed by library preparation and proliferation. But, these images were manually cropped. But, the objective of this project is to identify objects automatically from the split images. For, this purpose, we have implemented blob detection algorithm to identify the objects from the thresholded split images. We tested tree different algorithms for blob detection. They are -

- Laplacian of Gaussian
- Difference of Gaussian
- Determinant of Hessian

The detailes of the blob detection has been presented in the following section.

1.6.1 Step 1: Blob detection

As mentioned above, three different algoritms for blob detection have been implemented. The following code detects, crops and saves objects (healthy and sickled cell) from the thresholded split images. It also provides the location and radious of the blob. After carefull testing and validating them againts the original images, the Laplacian of Gaussian has been selected to use for the rest of the work. The code identifies the blos and calculated their centroid and generates an image for each identified object with a size of 60x60. The code is capable of extracting an object and generate image even from the very edge of the image by taking account of the relative position of the nearby edges and the centroid of the detected blobs.

```
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

def ImageBoxSizeCheck(x1, y1, x2, y2, img_height, img_width):
    if x1 < 0:
        error = -x1
        x1 = 0
        x2 = x2 + error
    if y1 < 0:
        error = -y1
        y1 = 0
        y2 = y2 + error
    if x2 > img_width:
        error = x2 - img_width
        x2 = img_width
        x1 = x1 - error
    if y2 > img_height:
        error = y2 - img_height
        y2 = img_height
        y1 = y1 - error

    return x1, y1, x2, y2

dirToSearch = './figs/SlicedImageThresholdOtsu/'

file_list = [None] * 400

for i in range(0,len(file_list)):
    file_list[i] = 'Otsu-Thresh-IMG-' + str(i) + '.png'

#===============================================================================

### number of the image we want to check
image_number = 25

### file_list = os.listdir(dirToSearch)

img1 = ndimage.imread(dirToSearch + file_list[image_number])
img1 = rgb2gray(img1)

img_height, img_width  = img1.shape
```

```
### "log" stands for "Laplacian of Gaussian"

blobs_log = blob_log(img1, max_sigma=30, num_sigma=10, threshold=.1)
blobs_log[:, 2] = blobs_log[:, 2] * math.sqrt(2)

### "dog" stands for "Difference of Gaussian"

blobs_dog = blob_dog(img1, max_sigma=30, threshold=.1)
blobs_dog[:, 2] = blobs_dog[:, 2] * math.sqrt(2)

### "doh" stands for "Determinant of Hessian"

blobs_doh = blob_doh(img1, max_sigma=30, threshold=.01)


for y,x,r in blobs_dog:
    print x,y,r


# fig, axs = plt.subplots(2,3, figsize=(9, 6))
# fig, axs = plt.subplots(len(blobs_dog),1, figsize=(9, 6))
# fig.subplots_adjust(hspace = .5, wspace=.001)
# axs = axs.ravel()


i=0


#for i in range(6):
for y,x,r in blobs_dog:
    if r > 15 and r < 30:
        y1 = y - 50
        y2 = y + 50
        x1 = x - 50
        x2 = x + 50

        x1, y1, x2, y2 = ImageBoxSizeCheck(x1, y1, x2, y2, img_height, img_width)

            #print y1,y2,x1,x2,r
        cropped = img1[y1:y2,x1:x2]
        print cropped.shape
        # cropped = img_as_float(cropped)
        # axs[i].imshow(cropped,cmap='gray')
        name = './figs/ImageLibrary/Blob_Dump/'+'IMG-'+ str(image_number) + '-' +str(i)+'.png'
        imsave(name, cropped)
        i=i+1
        #j=j+1

### End of the big "i" loop
#==============================================================================
```

This code generated images from the detected blobs with the size of 60x60. This code is looped for all the thresholted split images to detect objects from all of them. After generating the images, these images have been used to prepare another object library following the same procedure mentioned at section: 1.4.3. This was followed by data labeling and preparation of training, test dataset (section: 1.5.2) which were then subjection to prediction modeling (section: 1.5.3) .

1.6.2 Step 2: Interpret results

For the prediction/modeling, the three different classification models which have been mentioned in section: 1.5.4 have been implemented on the new test / train dataset. The accuracy of the implemented predeiction models are -

- Logistic Regression - test data accuracy: 97.72%
- Support Vector Classification - test data accuracy: 97.15%
- Random Forrest Classification - test data accuracy: 97.72%

1.6.3 Step 3: Challenge results

again for the newly developed image library, the test result has been challanged comparing with the accuracy on the train data. The accuracy of the train data is presented below -

- Logistic Regression - test data accuracy: 96.21%
- Support Vector Classification - test data accuracy: 99.75%
- Random Forrest Classification - test data accuracy: 100%

In this section, when we created the object library using blod detection, The accuracy for test data seems to have increased compared to the accuracy for the previous method. Based on the comparison between test data and train data accuracy, we can conclude that the prediction data is very much reasonable.

1.7 Validation

The developed code has been validated by applying the developed library. For the validation purpose, some other images has been used to generate image after object identification. Then the same the prediction model has been implemented to test and evaluate the performance of the code. Below is the code that does this -

```
#==============================================================================
#
# This following section of the code will work on a validation image to
# generate the images for applying the training data and prediction model
#
#==============================================================================


def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

def ImageBoxSizeCheck(x1, y1, x2, y2, img_height, img_width):
    if x1 < 0:
        error = -x1
        x1 = 0
        x2 = x2 + error
    if y1 < 0:
        error = -y1
        y1 = 0
        y2 = y2 + error
    if x2 > img_width:
        error = x2 - img_width
        x2 = img_width
        x1 = x1 - error
    if y2 > img_height:
        error = y2 - img_height
        y2 = img_height
        y1 = y1 - error

    return x1, y1, x2, y2

dirToSearch = './figs/SlicedImageThresholdOtsu/'

file_list = [None] * 400

for i in range(0,len(file_list)):
    file_list[i] = 'Otsu-Thresh-IMG-' + str(i) + '.png'

#===============================================================================

# counter = 0

#===============================================================================
#
# ### Beginning of the "big_i" loop
#
#===============================================================================
for big_i in range(25,150):
    img1 = ndimage.imread(dirToSearch + file_list[big_i])
    img1 = rgb2gray(img1)

    img_height, img_width  = img1.shape

    print "\n\n" +  'Current working file: ' + 'IMG-'+str(big_i) + "\n"


#===============================================================================
#     ### "log" stands for "Laplacian of Gaussian"
#
#     blobs_log = blob_log(img1, max_sigma=30, num_sigma=10, threshold=.1)
#     blobs_log[:, 2] = blobs_log[:, 2] * math.sqrt(2)
#===============================================================================

    ### "dog" stands for "Difference of Gaussian"

    blobs_dog = blob_dog(img1, max_sigma=30, threshold=.1)
    blobs_dog[:, 2] = blobs_dog[:, 2] * math.sqrt(2)

#===============================================================================
#     ### "doh" stands for "Determinant of Hessian"
#
#     blobs_doh = blob_doh(img1, max_sigma=30, threshold=.01)
#==============================================================================

    for y,x,r in blobs_dog:
        print x,y,r

    i=0

    for y,x,r in blobs_dog:
        if r > 10 and r < 30:
            y1 = y - 30
            y2 = y + 30
            x1 = x - 30
            x2 = x + 30

            x1, y1, x2, y2 = ImageBoxSizeCheck(x1, y1, x2, y2, img_height, img_width)

            cropped = img1[y1:y2,x1:x2]
            print cropped.shape

            name = './figs/Validation/'+'IMG-'+str(big_i)+'-'+str(i)+'.png'
            imsave(name, cropped)
            i=i+1
            #j=j+1

    # print counter

    # counter = counter + 1

### End of the big "i" loop
#==============================================================================

### Function to change RGB files to greyscale

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

sz = (60,60)  # Target Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

### Function to resize images

def process_image(img_file):
    img = cv2.imread(img_file)
    img = rgb2gray(img)
    #r = 100.0 / img.shape[1]
    #dim = (100, int(img.shape[0] * r))
    img = cv2.resize(img, sz).astype('float32')
    return img

#==============================================================================
# im = process_image(check_folder + 'hrbc-005a.png')
# im.shape
#
#===============================================================================
image_size = 60  # Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

### Main fuction to resize the imagaes

def load_pic(folder, min_num_images,n):
  """Load the data for a single type."""
  image_files = os.listdir(folder)
  dataset = np.ndarray(shape=(n, image_size, image_size),
                         dtype=np.float32)
  print(folder)

  num_images = 0
  for image in image_files:
    image_file = os.path.join(folder, image)
    img_data= process_image(image_file)
    try:
      image_data =  (img_data - pixel_depth / 2) / pixel_depth
      if image_data.shape != (image_size, image_size):
        raise Exception('Unexpected image shape: %s' % str(image_data.shape))
      dataset[num_images, :, :] = image_data
      num_images = num_images + 1
    except IOError as e:
      print('Could not read:', image_file, ':', e, '- it\'s ok, skipping.')

  dataset = dataset[0:num_images, :, :]
  if num_images < min_num_images:
    raise Exception('Many fewer images than expected: %d < %d' %
                    (num_images, min_num_images))

  print('Full dataset tensor:', dataset.shape)
  print('Mean:', np.mean(dataset))
  print('Standard deviation:', np.std(dataset))
  return dataset

### End of "load_pic" function

#===============================================================================
# The  following section loads and resizes the files and creates python
# objects for both "healthy" and "sickled" blood cell images and saves in
# the disk
#===============================================================================

validation_dir = ("C:/Noman/Case Western Reserve University/Courses/"
                    "2016-3-Fall/DSCI-451/Git/16f-dsci351-451-mnh38/"
                    "assignments/451-semester-projects/figs/Validation")

### "NVI" stands for no_of_validation_image
```

```
validation_files = os.listdir(validation_dir)
NVI = len(validation_files)

validation_dataset = load_pic(validation_dir, NVI,NVI)

#==============================================================================
#
# This section of the code loads the previously saved train data
#
#==============================================================================
with open('./python_objects/x_train.p', 'rb') as f:
          X_train = pickle.load(f)

with open('./python_objects/y_train.p', 'rb') as f:
          y_train = pickle.load(f)

### The X_train.reshape() and the X_test.reshape() functions have two
### parameters. The first number indicates the "test_size *  data_merge" and
### "test_size *  data_merge" for train and test, respectivey.

### The second parameter = image size * image size (such as 60)

X_train = X_train.reshape(819,3600)
validation_dataset = validation_dataset.reshape(NVI,3600)

# print(y_test.shape, y_train.shape, X_train.shape,X_test.shape)

### "clf"  in the name for the imported model, .LogisticRegression.
### This section is the model
### fit section

clf = linear_model.LogisticRegression(C=1e-3, max_iter=10000,n_jobs=-1)
clf.fit(X_train,y_train)

#X_test = test_dataset.reshape(len(test_dataset),len(test_dataset[0])*len(test_dataset[0]))

### Model predict section

pred_train= [(clf.predict(row.reshape(1,-1)))[0] for row in X_train]
pred_test = [(clf.predict(row.reshape(1,-1)))[0] for row in validation_dataset]
# print ('accuracy is : ',accuracy_score(y_test,pred_test))
print ('accuracy is : ',accuracy_score(y_train,pred_train))

for i in range(0, NVI-1):
    if pred_test[i] == 1:
        print validation_files[i]

### Checking
# y_test
### zip section
a = zip(y_test,pred_test)

np.sum(abs(y_test-pred_test))
```

```
### SVM = Support vector Machine classification
### SVC = Support vector classification

clf = SVC(C=15,random_state=42)
clf.fit(X_train, y_train)
pred_train= [(clf.predict(row.reshape(1,-1)))[0] for row in X_train]
pred_test = [(clf.predict(row.reshape(1,-1)))[0] for row in validation_dataset]
# print ('accuracy is : ',accuracy_score(y_test,pred_test))
print ('accuracy is : ',accuracy_score(y_train,pred_train))

for i in range(0, NVI-1):
    if pred_test[i] == 1:
        print validation_files[i]


### Chceking
# clf


### Random Forrest Classification model

clf = RandomForestClassifier(n_estimators=200,n_jobs=-1)
clf.fit(X_train, y_train)
pred_train= [(clf.predict(row.reshape(1,-1)))[0] for row in X_train]
pred_test = [(clf.predict(row.reshape(1,-1)))[0] for row in X_test]
# print ('accuracy is : ',accuracy_score(y_test,pred_test))
print ('accuracy is : ',accuracy_score(y_train,pred_train))

for i in range(0, NVI-1):
    if pred_test[i] == 1:
        print validation_files[i]

#==============================================================================
```

1.8 Quantification

For the validation purpose, the code generated images from 127 identified objects. After performing the prediction modeling, the performance of the code has been assessed -

- Logistic Regression - 2 out of 51 healthy RBC detected, accuracy 3.92%
- Support Vector Classification - 9 out of 51 healthy RBC detected, accuracy 17.64%
- Random Forrest Classification - 2 out of 51 healthy RBC detected, accuracy 3.92%

Rest of the RBCs has been detected as sickled RBCs. This is a huge erroe by margin. The validation step shows that, the code is not robust enough to achieve a reasonable degree of accuracy while applied on blind data. The code needs to be improved to achieve higher accuracy.

1.9 Conclusion

In conclusion, the project has reached it's landmark within the limited time available, that is, all the sub-aims have been achieved within this limited amount of time. However, the developed code is not robust enough while being tested against blind data set.

1.10 Future direction

The most important findings of this project is that performance of the developed code on a blind dataset. It shows that the developed code lacks the robustness to provide a reasonable accuracy while tested on blind dataset. So, the future direction for this project is to improve the robustness to schieve the desired accuracy on blind dataset.

1.11 Acknowledgement

I would like to thank

– Course Instruction: Dr. Roger French

– Theaching Assistant: Yu Wang

– Mohammad Akram Hossain, Solar Durability and Lifetime Extension (SDLE) Center

– Justin Fada, Solar Durability and Lifetime Extension (SDLE) Center

for their valuable assistance and guidance in this project.

2 Appendix A

List of Codes

The list of python codes attached with theis final report is given below -

- 451-semester-project-split-threshold-combined-code.py
  - Used for image reading, splitting, thresholding / filtering
- LibraryImageRotate-Manually-Cropped.py
  - Used for library proliferation, by rotating the library images
- Cell_IdentificationCode-Short-Manually-Cropped.py
  - Used for training, testing and predition
- MultiCellDetection_Library_BuildUp_Short.py
  - Used for blob detection and library preparation
- LibraryImageRotate-Automatically-Cropped.py
  - Used for proliferation of images library after blob detection
- Cell_IdentificationCode-Short-Automatically-Cropped.py
  - Used for training, testing and predition on image generated by blob detection
- MultiCellDetection_Prediction_Validation.py
  - Used for validating the code, by testing it on blind dataset.