Appendix

```
%Training
function trainingSet = loadData()
    baseFolder = cd("training");

    %Training matrix- hard-coded Training and Class
    %L:Lymphocyte, M:Monocyte, N:Neutrophil
    trainingSet = cell(4, 60);
    nameStorage = cell(3, 60);

    for i = 1:60
        if i <= 20
            trainingSet{1, i} = 'L';
        elseif i <= 40
            trainingSet{1, i} = 'M';
        elseif i <= 60
            trainingSet{1, i} = 'N';
        end
    end

    IT_Folder_Index = 2; %2 for image, 3 for truth
    for imageTruth = 3:length(dir)     %Image or Truth (IT)
        %Change Folder to sub Training folder and store Training directory
        IT_Directory = dir;
        trainFolder = cd(IT_Directory(imageTruth).name);
        Image_Index = 1;
        for LMN = 3:length(dir)        %Lymphocyte, Monocyte, Neutrophil (LMN)
            %Change Folder to L, M, or N and store Image or Truth Dir
            LMN_Directory = dir;
            baseImageTruth = cd(LMN_Directory(LMN).name);

            %Storing Images
            for image = 3:length(dir)
                image_Directory = dir;
                if IT_Folder_Index == 2
                    currentIndex = Image_Index;
                    periodPosition = strfind(image_Directory(image).name, '.');
                    fileName = image_Directory(image).name(1:periodPosition -
1);
                    trainingSet{4, currentIndex} = strcat(fileName,
trainingSet{1, currentIndex});
                else
```

```
                    underscorePosition = strfind(image_Directory(image).name,
'_');
                    fileName = image_Directory(image).name(1:underscorePosition
- 1);
                    fileName = strcat(fileName, trainingSet{1, Image_Index});
                    nameSearch = strfind({trainingSet{4, :}}, fileName);
                    currentIndex = find(~cellfun('isempty', nameSearch));
                end
                trainingSet{IT_Folder_Index, currentIndex} =
imread(image_Directory(image).name);
                nameStorage{IT_Folder_Index, currentIndex} =
image_Directory(image).name;
                Image_Index = Image_Index + 1;
            end
            cd(baseImageTruth);
        end
        cd(trainFolder);
        IT_Folder_Index = 3;
    end
    cd(baseFolder);

    for i = 1:60
        if i <= 20
            trainingSet{1, i} = 0;
        elseif i <= 40
            trainingSet{1, i} = 1;
        elseif i <= 60
            trainingSet{1, i} = 2;
        end
    end
end

------------------------------------------

% Segmenting the nucleus
function nuclearData = nuclear(data)
    % Initialize empty cell for segmented nucleus
    cols = size(data, 2);
    nuclearData = cell(1, cols);

    % Process each training image
    for i = 1:cols
        image = data{2, i};
        % Convert image to lab space and normalize
        imagelab = rgb2lab(image);
        lab2norm = imagelab(:, :, 2) / 100;

        % Perform thresholding
        level = multithresh(lab2norm, 3);
        lab2multi = imquantize(lab2norm, level);

        % Set all values that are not maximum to 0
        maximum = max(max(lab2multi));
        lab2multi(lab2multi < maximum) = 0;
        lab2multi(lab2multi == maximum) = 1;


        % Denoise image
        lab2multinone = bwareaopen(lab2multi, 250);

        % Add images
        nuclearData{1, i} = lab2multinone;
    end
end

------------------------------------------

% Segmenting the cytoplasm
function cytoNucSeg = cytoplasm(data, nuclearData)
    % Initialize empty cell for segmented cytoplasm
    cols = size(data, 2);
    cytoNucSeg = cell(1, cols);

    % Looping through all images
    for image = 1:size(data, 2)
        % Obtaining image size
        [x, y, z] = size(data{2, image});

        % Initializing Contour space by creating a box space around
        % nucleas
        [i, j, v] = find(nuclearData{1, image});
        minI = max(min(i)+5, 1);
        minJ = max(min(j)+5, 1);
        maxI = min(max(i)-5, x);
        maxJ = min(max(j)-5, y);
```

```
% Creating mask or initial contour space
mask = zeros(x, y);
mask(minI:maxI, minJ:maxJ) = 1;

% Converting Original Image to HSV or HSI to extract S values
imageHSV = rgb2hsv(data{2, image});
imageS = imageHSV(:,:,2);


% Smoothing equalized image with anisotropic diffusion
imageDiff = imdiffusefilt(imageS);

% Performing Histogram Equalization on Saturation values of image
imageSDiff = histeq(imageDiff);



% Performing active contour of smoothed image and mask
bw = activecontour(imageSDiff  + edge(imageSDiff, 'sobel'), mask);

% Isolating largest connected component
connectedBW = bwconncomp(bw); %struct.PixelIdxList
cleanBW = zeros(x, y);
if(connectedBW.NumObjects > 1)
    % There are more than one connected components
    % Using cellfun to get size of components
    largestCompSize = cellfun('size',connectedBW.PixelIdxList,1);

    % Getting size of largest component (Assuming cell is largest)
    largestCompIndex = find(largestCompSize == max(largestCompSize));
    largestComp = connectedBW.PixelIdxList{largestCompIndex};

    % Setting matrix at indices of largest component to 1
    cleanBW(largestComp) = 1;
else
    % There is only one connected component
    cleanBW = bw;
end

% Overlaying active contour with nucleus segment to produce
% total segmented image
cytoNucSeg{image} = cleanBW + nuclearData{1,image};
```

```
    end
end

------------------------------------------

% Create feature matrix
function f = features(originalData, segNucleus, segCytoplasm)
    rows = size(originalData, 2);
    fMatrix = zeros(rows, 16);

    for i = 1:rows
        % Feature column
        fMatrix(i, 1) = originalData{1, i};

        % Perimeters
        fMatrix(i, 2) = perimeter(segNucleus{1, i});
        fMatrix(i, 3) = perimeter(segCytoplasm{1, i});

        % Area
        fMatrix(i, 4) = area(segNucleus{1, i});
        fMatrix(i, 5) = area(segCytoplasm{1, i});

        % Roundness
        fMatrix(i, 6) = roundness(segNucleus{1, i});
        fMatrix(i, 7) = roundness(segCytoplasm{1, i});

        % Ratio
        fMatrix(i, 8) = ratio(segNucleus{1, i}, segCytoplasm{1, i});

        % Number of nucleus parts
        fMatrix(i, 9) = number(segNucleus{1, i});

        % Color
        color = avgColor(originalData{2, i}, segCytoplasm{1, i});
        fMatrix(i, 10:12) = color(1:3);

        % Texture features
        tex = texture(segCytoplasm{1, i}, originalData{2, i});
        fMatrix(i, 13:16) = [tex.Contrast, tex.Correlation, tex.Energy, ...
            tex.Homogeneity];

    end
    % Include the header in matrix for ease of reading
    f = fMatrix;
end

------------------------------------------

% Parameter is one segmented image
% Calculate the perimeter of nucleus or cytoplasm
function per = perimeter(cell)
    [x, y, z] = size(cell);
    [i, j, v] = find(cell);
    minI = max(min(i) + 5, 1);
    minJ = max(min(j) + 5, 1);
    maxI = min(max(i) - 5, x);
    maxJ = min(max(j) -5, y);

    xdistance = maxJ - minJ;
    ydistance = maxI - minI;
    per = 2 * xdistance + 2 * ydistance;
end

------------------------------------------

% Parameter is one segmented image
% Calculate the area of nucleus or cytoplasm
function a = area(cell)
    x = nonzeros(cell);
    a = sum(x);
end

------------------------------------------

% Parameter is one segmented image
% Calculate the roundness of nucleus
function round = roundness(cell)
    p = perimeter(cell);
    a = area(cell);
    round = (p ^ 2) / ( 4 * pi * a);
end

------------------------------------------
```

```
% Parameter is one segmented image
% Calculate the number of parts of the nucleus
function n = number(nucleus)
    n = bwconncomp(nucleus);
    n = n.NumObjects;
end

------------------------------------------

% Parameters are the segmented cytoplasm and nucleus images
% Calculate the ratio of nucleus to cytoplasm
function rat = ratio(nucleus, cytoplasm)
    cArea = area(cytoplasm);
    nArea = area(nucleus);
    rat = nArea/ cArea;
end

------------------------------------------

% Calculate the average color of the cytoplasm
% The parameters are the original image and the segmented cytoplasm
function color = avgColor (original, segCytoplasm)
    [cytoRow, cytoCol] = find(segCytoplasm == 1);
    color = zeros(1,3, 'uint32');
    % Add up all of the RGB values
    for i=1:size(cytoRow)
        color = color + uint32(reshape(original(cytoRow(i),cytoCol(i),:), 1,
3));
    end
    % Find the average color
    color = color/size(cytoRow,1);
    % Convert back to uint8
    color = uint8(color);
end

------------------------------------------


function tex = texture(cytoplasm, original)
    % Convert to grayscale
    gray = rgb2gray(original);
    [x, y] = size(gray);
    % Find linear indices of everything but the cytoplasm
    notCyto = (cytoplasm ~= 1);
    k = find(notCyto);

    % Set all colors that are not in the cytoplasm to black
    gray(k) = 0;

    % Reduce image size
    [i, j] = find(gray);
    minI = max(min(i), 1);
    minJ = max(min(j), 1);
    maxI = min(max(i), x);
    maxJ = min(max(j), y);
    reduced = gray(minI:maxI, minJ:maxJ);

    % Find lowest grey level value
    u = unique(reduced);
    u = u(2) - 1;

    % Get rid of space between 0 (background) and first grey level
    reduced2 = reduced - u;
    reduced(reduced ~= 0) = reduced2(reduced2 ~=0);

    % Reduce number of grey levels to 64
    last = unique(reduced);
    last = last(end);
    denominator = ceil(last/64);
    final = floor(reduced/denominator);


    % Using a distance of 1, calculate 4 co-occurence matrices of
    % different angles and then average them
    d = 1;
    glcms0= graycomatrix(final,'NumLevels', 64, 'Offset', [0 d]);
    glcms45= graycomatrix(final,'NumLevels', 64, 'Offset', [-d d]);
    glcms90= graycomatrix(final,'NumLevels', 64, 'Offset', [-d 0]);
    glcms135= graycomatrix(final,'NumLevels', 64, 'Offset', [-d d]);
    glcms = (glcms0 + glcms45 + glcms90 + glcms135) / 4;
    glcms = uint8(glcms);

    props = graycoprops(glcms);
    tex = props;
end
```

```
------------------------------------------

% Loading the test data
% Going from testing folder to a matrix
function testSet = loadTest()

    baseFolder = cd("testing"); %Before it changes to the testing folder, it
tells you where it originated, cd changes the directory

    testSet = cell(2,10); %cells store multiple types of data, unlike arrays
that only store the same type of data


    for testImage = 3:length(dir)
        testSet{1, testImage-2} = -1;
        currentDirector = dir;
        testSet{2,testImage-2} = imread(currentDirector(testImage).name);
    end
    cd(baseFolder);
end

------------------------------------------


% Script to run the program
[trainData, testData] = loadData();
trainNuc = nuclear(trainData);
testNuc = nuclear(testData);

trainCyto = cytoplasm(trainData, trainNuc);
testCyto = cytoplasm(testData, testNuc);

trainFeat = features(trainData, trainNuc, trainCyto);
testFeat = features(testData, testNuc, testCyto);
```