Segmentation of Skin Lesions using Combined Adaptive Thresholding and Connected Component Analysis

A Project by: Group #12

Johnathan Maynard (jam945)

Aswin Raghav Nirmaleswaran (an539)

Stephen Schneider (sms675)

Table of Contents

1. Methods

The first approach was to preprocess the image followed by an algorithm which implements a form of adaptive snake wherein distinct intensity difference in pixels corresponding to the regions between skin and the lesions will be used. These edges will be marked to create a binary image. The inter pixel variations in the images will be used to identify any connected pixels which are of similar intensity. Strong edges will be used as weights to identify the direction of propagation of the boundary for consecutive iterations. The algorithm will be iterated multiple times till all the weight components at different edge points reach a particular value of minima.

In the second approach, an attempt to use the CIELAB color scheme was made to extract out an intensity invariant channel. However, implementation of this approach in VisionX was not done. An attempt to convert RGB channels to hexadecimal scale was also tried. This actually gave room for clustering objects by Otsu's using the Hexadecimal color plot of the image. The computation complexity in this method was too high that it wasn't possible for us to compute the segmentation of the images to build the algorithm. So, it was decided to go with extraction of the channel which has the least noise for the training dataset.

In the final approach, each color channel was separately processed and segmented using Otsu's method. The three color channels were then merged using logic functions to create a single representative mask of the skin lesion. This method of adaptive segmentation yielded the maximum Jaccard index. Hence, all the advantages which were observed in these approaches were combined to yield us the final algorithm which is discussed further.

1.1 Final Algorithm

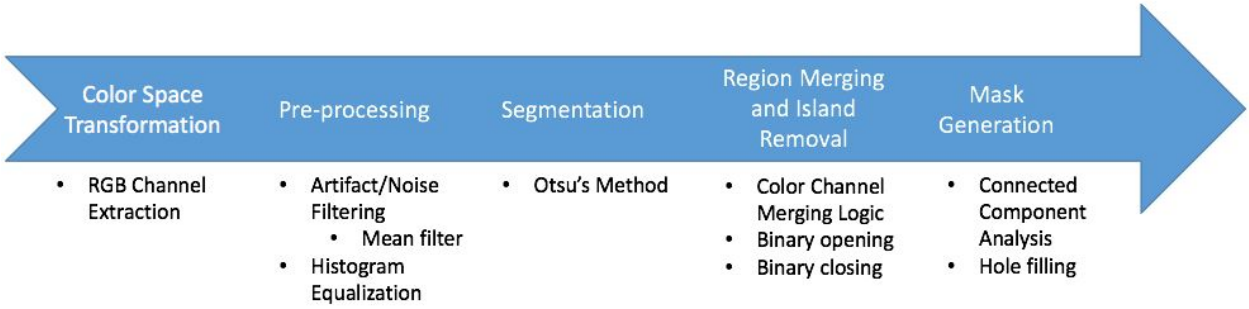

Figure 1: Algorithm process flow

The team developed an algorithm that takes advantage of the contrast between normal skin and the skin lesion of interest in the dermoscopic images. The algorithm consists of 5

primary steps: color space transformation, pre-processing, segmentation, region merging and island removal, and mask generation, see Figure 1.

In order to perform pixel value operations on the images, a color space transform was used to separate the three color channels (red, blue, and green) into three grayscale images. Because of differences in light intensity, noise levels, and the presence of artifacts, pre-processing of these images is necessary. Low pass filtering of the image using a mean filter is an effective way to linearly filter an image with great effect. Histogram equalization, linearization of the cumulative distribution function, is used to maximize the contrast of the image.

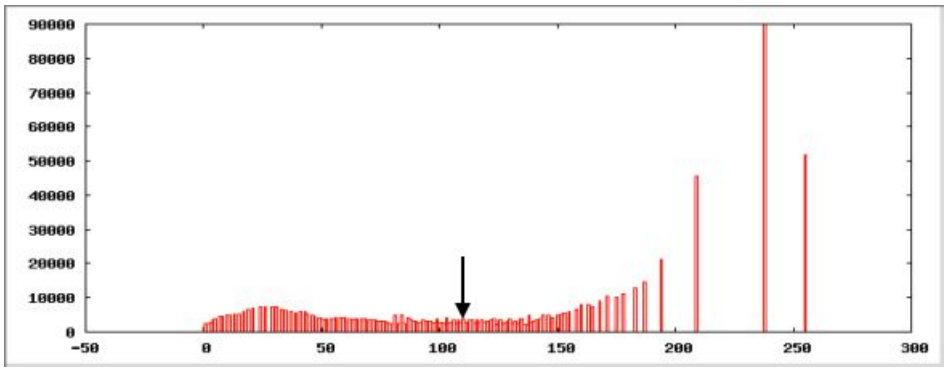

Figure 2: Pre-processed histogram.The arrow indicates the threshold found via Otsu's Method.

After pre-processing to remove artifacts and noise, the histogram of the dermoscopic image can be approximated as a bimodal gaussian distribution, see Figure 2. Thus, thresholding of the image is an effective method to segment the image into two regions. Skin lesions are darker in tone than the surrounding skin especially in the blue color channel. Therefore, skin lesions can be reasonably be expected to be below an implemented threshold, see Figure 3. Otsu's method is used to define the image threshold. Otsu's method optimizes the intra-class variance ($\sigma^2_{intra} = p_1(T)\sigma^2_1(T) + p_2(T)\sigma^2_2(T)$) of the pixel values above and below the image threshold allowing for optimal separation of the bimodal distribution, see Figure 2.

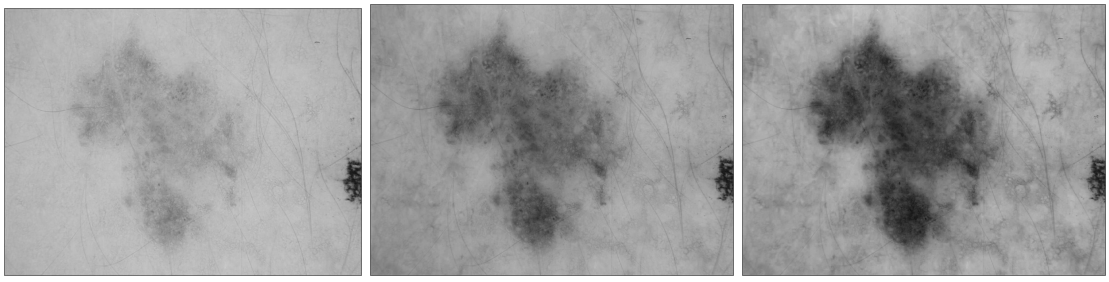

Figure 3: Grayscale images of each color channel. Left to right: red, green, and blue color channels.

The regions identified in each color channel need to be merged to develop a single region that best captures information from all three color channels. A boolean logic function is used to

merge the color channels as detailed in Figure 4. This logic function provides the most information about the region of interest, while minimizing the impact of remaining artifacts. Binary opening (erosion-dilation) followed by binary closing (dilation-opening) is then used to smooth region boundaries and remove non-connected islands, Figure 5. Finally, a mask of the isolated lesion was created using connected component analysis to identify the largest connected region and hole filling to fill fully enclosed gaps in the region.

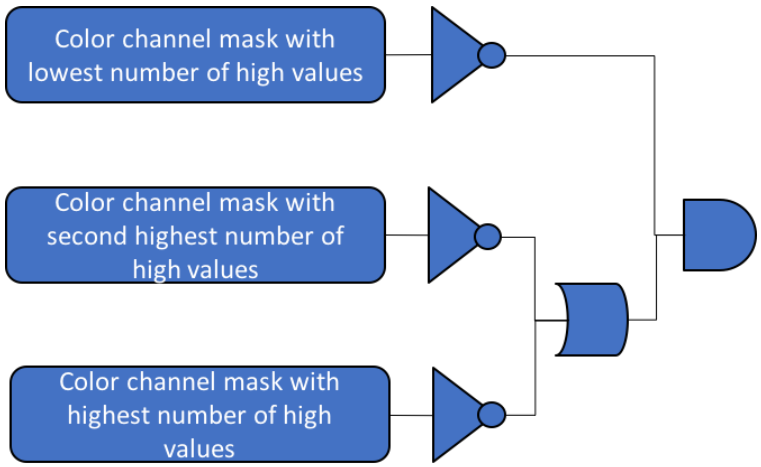

Figure 4**:** Color channel merging logic, $(\neg Highest \vee \neg 2nd\ Highest) \wedge \neg Lowest$.

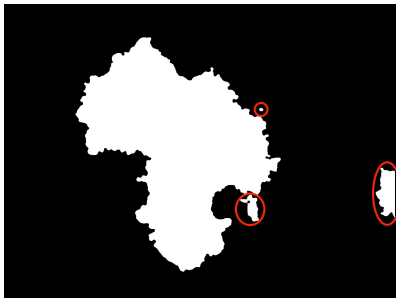

Figure 5: Image with islands to be removed with opening and region borders to be smoothed with closing. Red circles indicate islands that are not connected to the region of interest.

1.2 Experiment

Training of the algorithm began with the use of small, synthetic images (20x20 pixels) created in Gimp to verify that the algorithm was producing expected results. Further training was performed utilizing a set of color dermoscopic images previously used in the Skin Lesion Analysis Towards Melanoma Detection Challenge for ISBI 2016. The competition provided a comprehensive data set consisting of 900 images and the corresponding skin lesion ground truths identified by dermatologists.

1.2.1 Hypothesis and Evaluation Function

During the ISBI 2016 challenge, participants were provided training data from the same data set as used to train the proposed algorithm. Challenge rankings for the image segmentation task was determined using the Jaccard index. The Jaccard Index is defined as: $\frac{\#\ of\ true\ positives}{\#\ of\ true\ positives + \#\ of\ false\ negatives + \#\ of\ false\ positives}$, with 1 being an exact match and 0 being no match between the automated mask and the ground truth. The highest ranked team achieved an average Jaccard index of 0.843 over a test image set. [1]

The team hypothesizes that the algorithm defined in this paper will achieve an average Jaccard index approaching or exceeding 0.843.

1.2.2 Documented Data Set

The ISBI 2016 challenge data set consists of 900 dermoscopic images and the corresponding ground truth, manually marked by dermatologists. [1] The dermoscopic images are full color jpeg files of varying resolution and image size. The ground truth masks are binary png files of corresponding resolution and image size.

1.2.3 Experiment procedure

For training, 25 images and corresponding ground truths were randomly selected from the dataset. Optimization of parameters for both the snake and adaptive thresholding was performed by maximizing the average Jaccard index over the training set. The critical parameters that were involved in this final algorithm include:

1. Filter type (Mean, Gaussian, Median)
2. Kernel size (x-dimension, y-dimension)
3. Opening kernel (radius)
4. Closing kernel (radius)
5. Merging method ($(\neg Highest \vee \neg 2nd\ Highest) \wedge \neg Lowest$, $\neg Blue\ Channel$, $\neg(Lowest \vee Highest)$)

For testing, an additional 100 different images and corresponding ground truths were randomly selected from the data set. For these test images, the optimized algorithm was used to segment the images and the average Jaccard index over the entire test set was recorded.

2. Results:

2.1 Training

2.1.1 Small synthetic test image

Table 1: Small synthetic test image outputs.

Algorithm	Jaccard Index
Adaptive Thresholding	0.832
Snake	0.899

2.1.2 Parameter optimization

Table 2: Selected average Jaccard index values across the training set using various parameters.

Low-pass filter	Filter kernel (x,y)	Channel Merging Logic	Opening kernel	Closing kernel	Average Jaccard index
Mean	(7,7)	$(\neg Highest \vee \neg 2nd\ Highest) \wedge \neg Lowest$	30	70	0.655
Mean	(7,7)	$\neg Blue\ Channel$	30	70	0.633
Mean	(7,7)	$\neg(Lowest \vee Highest)$	30	70	0.530

Table 2.1: Selected average Jaccard index values across the training set for the Adaptive Snake method using various degrees of smoothing.

Gaussian kernel size	Iteration #	Initial mask	Smoothing (alpha)	Average Jaccard Index
10	330	Centered Square	.1	0.7953
10	330	Centered Square	.2	0.7950
10	330	Centered Square	.3	0.7945

2.2 Testing

Table 3: Average Jaccard index across the test set using optimized parameters.

Low-pass filter	Filter kernel (x,y)	Channel Merging Logic	Opening kernel	Closing kernel	Average Jaccard index
Mean	(7,7)	$(\neg Highest \vee \neg 2nd\ Highest) \wedge \neg Lowest$	30	70	0.497

Testing was not able to be completed for the adaptive snake segmentation due to the large computational time necessary to experiment on 100 images.

3. Discussion

3.1 Training

3.1.1 Small synthetic test image

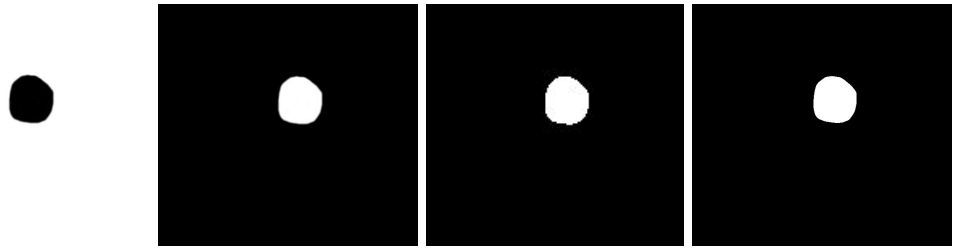

Figure 6: Small synthetic test image output. Left to right: Original test image, ground truth, snake output, adaptive threshold output.

The snake algorithm produced results closer to that of the ground truth for the synthetic image, see Table 2.1. This is likely because the snake utilizes a gaussian filter rather than a mean filter. The mean filter has a larger impact on the values along the boundary of the region of interest due to both the large kernel size and how the average is computed compared to the gaussian function.

3.1.2 Parameter optimization

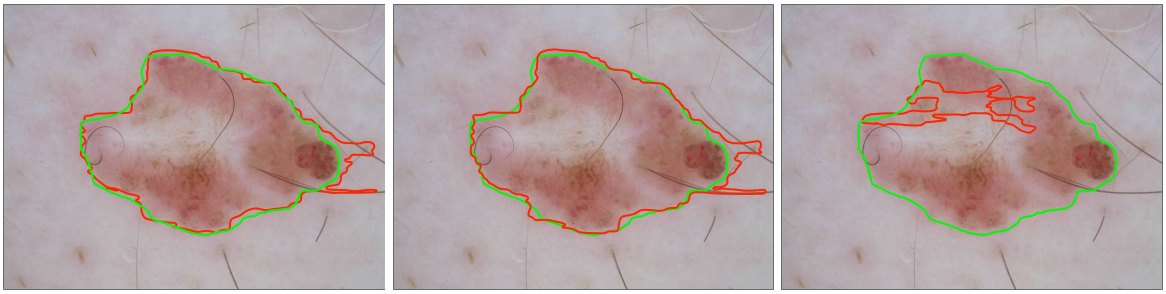

Figure 7: An image segmented by adaptive thresholding. Left to right: $(\neg Highest \vee \neg 2nd\ Highest) \wedge \neg Lowest$, $\neg Blue\ Channel$, and $\neg(Lowest \vee Highest)$. Green is the boundary of the ground truth mask. Red is the boundary of the algorithm output. Processing with a mean filter of (7,7), an opening kernel of 30, and a closing kernel of 70.

The logic function, $(\neg Highest \vee \neg 2nd\ Highest) \wedge \neg Lowest$, resulted in the highest average Jaccard index over the training set. Figure 7 shows the amount of overlap between the ground truth and the automatic segmentation. In the figure, the first two logic functions result in relatively good regions, however both results have a pair of protrusions that extend beyond the skin lesion. This is likely due to the hair artifact present within each protrusion being included as

part of the region. It may be possible to reduce the impact of hair artifacts by further low-pass filtering or using removal techniques such as shape matching or the linear Hough transform. The third logic function fails to segment the lesion effectively and can be seen to be bounding the lighter interior of the lesion. This highlights the detrimental impact that the high amount of internal variation within the skin lesions has on the ability to effectively segment the lesion from the background. In this case, the interior of the lesion is very similar in color to the surrounding skin.

3.2 Testing

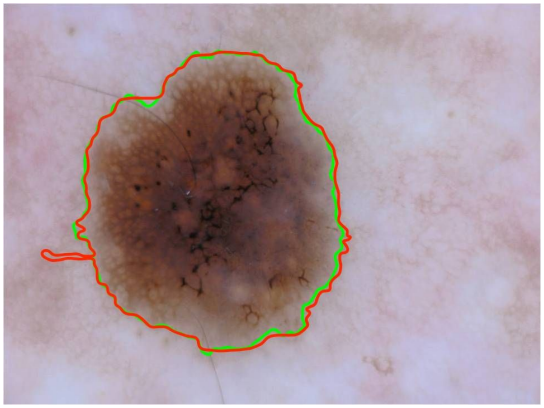

Figure 8: An image well segmented by adaptive thresholding. Green is the boundary of the ground truth mask. Red is the boundary of the algorithm output.

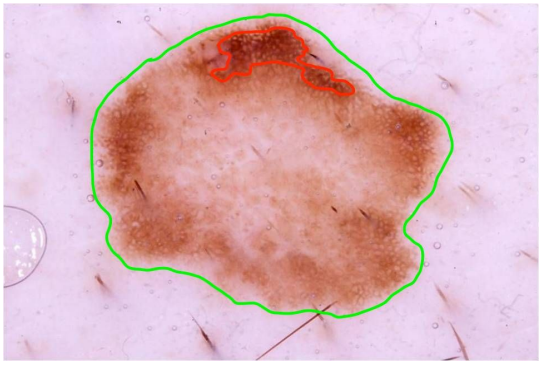

Figure 9: A image poorly segmented by adaptive thresholding. Green is the boundary of the ground truth mask. Red is the boundary of the algorithm output.

The adaptive thresholding algorithm did not produce accurate segmentation overall for the test image set. A high amount of variation in the individual Jaccard indexes was noted as some images such as that in Figure 8 were better segmented while images such as that in Figure 9 were rather poorly segmented.

The good segmentation of the image in Figure 8 is largely due to the lower variation of color in the skin lesion, the lack of gaps along the edges of the lesion, and the high contrast between the skin lesion and the surrounding skin. The image in Figure 9 exhibits characteristics similar to that of the image in Figure 7. There is a very high amount of variation between regions

within the skin lesion. In particular, the region selected by the adaptive threshold in Figure 9 is darker than the surrounding lesion and skin and is disconnected from nearby dark regions of the lesion. This makes robust segmentation of this image difficult as there are many potential regions that may be selected by Otsu's method. Additional filtering may help to reduce the variation and produce an image histogram closer to biomodal, improving the automatic threshold accuracy. Another possibility is the introduction of a euclidean distance transform to merge nearby regions into single larger regions before performing connected component analysis.

Additionally, high resolution images posed challenges in the robustness of the algorithm. Aside from requiring extensive computational resources, the optimized parameters were not scaled to the size of the image and as such a drop in the accuracy of lesion segmentation was noted for larger images. To address this, future implementations of the algorithm will utilize a scaled kernel for filtering to ensure that the amount of pre and post processing of the image and binary mask are proportional to the image size.

The snake algorithm yielded a fairly better output for low resolution images. However, it failed when presented with high resolution images or images with dark hair which resembled contours. Moreover, it strictly required a low pass filtering. Unfortunately, we were unable to convert the MATLAB script needed to generate this code into C in the given time frame. Therefore, we decided to explore other options for our evaluation. Despite this, some outputs can be seen below, Figure 10.

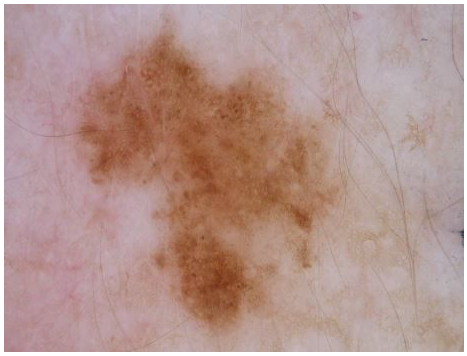

Figure 10A: Original Image.

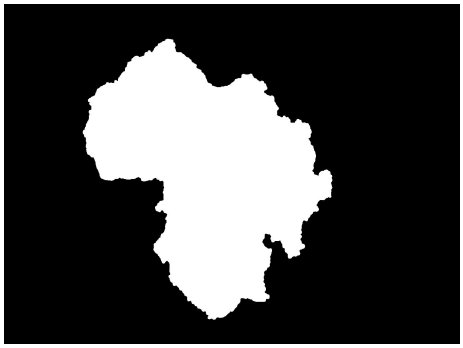

Figure 10B: Ground truth.

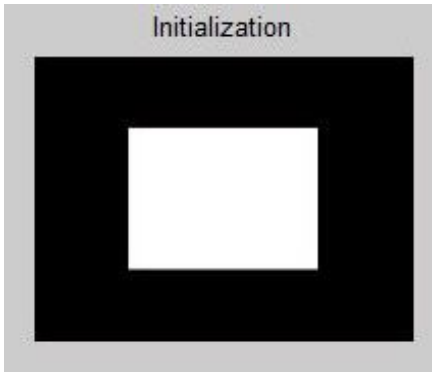

Figure 10C: Adaptive Snake Initialization (Centered Rectangle chosen at random)**.**

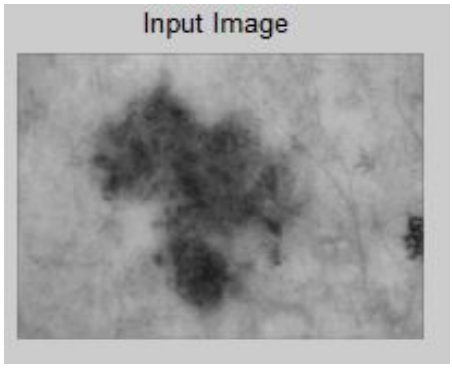

Figure 10D: Pre-processed image (Blue channel with Gaussian Filter of sigma 6 and size 10).

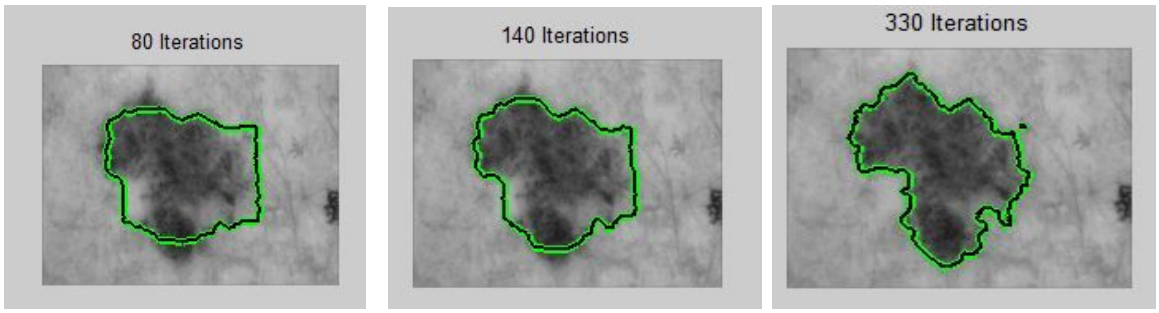

Figure 10E: Adaptive snake output after 80, 140 and 330 iterations.

In the second approach of using individual color channels for segmentation, the individual channels were first converted to grayscale as shown below:

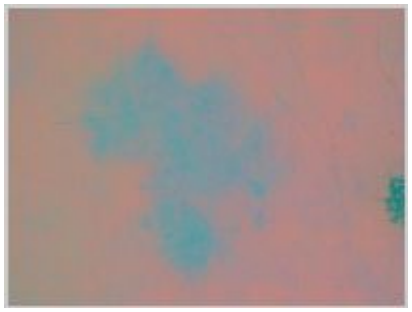

Figure 11A: Image converted to l*a*b color space.

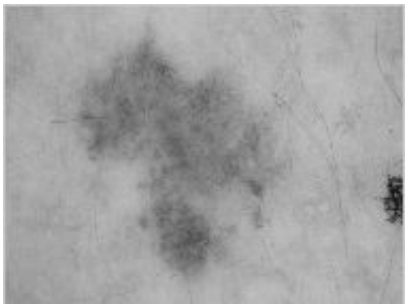

Figure 11B: Channel 1 from l*a*b.

Figure 11C: Channel 2 from l*a*b.

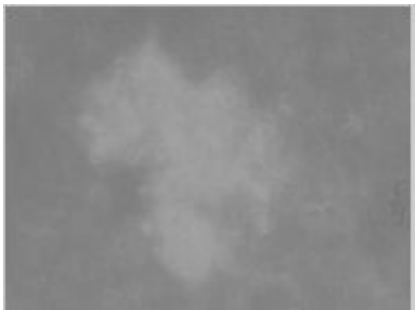

Figure 11D: Channel 3 from l*a*b.

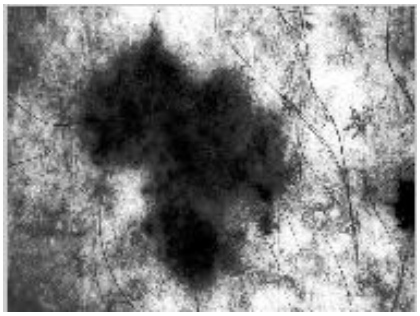

Figure 11E: Histogram-equalized Ch1.

During this second approach this observation of enhanced features in l*a*b color space was observed to be useful. Thus, these factors which were observed was considered to be useful. Our observations that were made in the final algorithm will be discussed during the presentation.

Conclusion

The proposed combined adaptive thresholding and connected component analysis algorithm achieved a low average Jaccard index of 0.497 over the selected 100 image test set. This is substantially less than the hypothesized index of 0.843 and as such the team fails to accept the hypothesis. Issues with computational resources and parameter variation when using higher resolution images also plagued the algorithm.

The adaptive snake is a powerful algorithm but its performance was too poor for high resolution images. It scored an average Jaccard Index of 0.795 with the training data. The individual channels of l*a*b color space was intensity invariant and is powerful in handling the dermoscopic images which were taken with improper illumination. They also were useful in identifying the texture of tumor. This proves its usefulness in automated disease classification or Disease Diagnosis. By including all these merits of the individual algorithms into our final algorithm would enhance the capabilities of our current algorithm. Doing that while still maintaining a high Jaccard value will be left as a future scope of this project.

References

1. Margarida Silveira, Jacinto C. Nascimento, Jorge S. Marques, André R. S. Marçal, Teresa Mendonça, Syogo Yamauchi, Junji Maeda, and Jorge Rozeira "Comparison of Segmentation Methods for Melanoma Diagnosis in Dermoscopy Images", IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING, VOL. 3, NO. 1, FEBRUARY 2009.
2. Alexander Wong, Jacob Scharcanski and Paul Fieguth, "Automatic Skin Lesion Segmentation via Iterative Stochastic Region Merging", IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 15, NO. 6, NOVEMBER 2011.
3. Zhen Ma and Joao Manuel R. S. Tavares, "A Novel Approach to Segment Skin Lesions in Dermoscopic Images Based on a Deformable Model", IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, VOL. 20, NO. 2, MARCH 2016.
4. H. Ganster, P. Pinz, R. Rohrer, E. Wildling, M. Binder, and H. Kittler, "Automated melanoma recognition," IEEE Trans. Med. Imag., vol. 20, pp. 233–239, Mar. 2001
5. G. A. Hance, S. E. Umbaugh, R. H. Moss, and W. V. Stoecker, "Unsupervised color image segmentation," IEEE Eng. Med. Biol. Mag., vol. 15, no. 1, pp. 104–111, Jan./Feb. 1996
6. P. Rubegni, A. Ferrari, G. Cevenini, D. Piccolo, M. Burroni, R. Perotti, K. Peris, P. Taddeucci, M. Biagioli, G. Dell'Eva, S. Chimenti, and L. Andreassi, "Differentiation between pigmented spitz naevus and melanoma by digital dermoscopy and stepwise logistic discriminant analysis," Melanoma Res., vol. 11, no. 1, pp. 37–44, 2001.
7. J. Nascimento and J. S. Marques, "Adaptive snakes using the em algorithm," IEEE Trans. Image Process., vol. 14, pp. 1678–1686, 2005.
8. Cancer.org- Key Statistics for melanoma skin cancer: http://www.cancer.org/cancer/skincancer-melanoma/detailedguide/melanoma-skin-cancer-key-statistics, 20th May, 2016
9. Gutman, D., Codella, N., Celebi, E., Helba, B., Marchetti, M., Mishra, N., & Halpern, A. (2016). Skin Lesion Analysis toward Melanoma Detection: A Challenge at the International Symposium on Biomedical Imaging (ISBI) 2016, hosted by the International Skin Imaging Collaboration (ISIC). *Arxiv*. Retrieved from http://arxiv.org

Program Documentation

1. Jaccard Index:

NAME:

jacind- calculates the Jaccard Index between two binary images by getting two individual images as input.

SYNOPSIS:

jacind [if=<image file>] [tf=<ground truth file>]

DESCRIPTION:

jacind calculates the Jaccard Index of two binary images.

Individual pixels of the input images are scanned and counted up using a counter for every high intensity value '1'. The low intensity values '0' are neglected. Individual pixels of the ground truth image are scanned and counted up for every high value.
The pixel positions wherein both images have high values are counted using a counter. These three counted values are computed for the formula shown below to calculate Jaccard index:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

OPTIONS:

if=	binary image file
tf=	binary image file of ground truth
-v	prints immediate information

AUTHORS:

Johnathan Maynard, Aswin Raghav Nirmaleswaran, Stephen Schneider

2. Finding the size of the mask:

NAME:

nmask- calculates the size of the mask.

SYNOPSIS:

nmask [if=<image file>]

DESCRIPTION:

nmask calculates the the size of mask and prints it.

Individual pixels of the input images are scanned and counted up using a counter for every non zero intensity value ‘’.

OPTIONS:

if=	image file in short format

AUTHORS:

Johnathan Maynard, Aswin Raghav Nirmaleswaran, Stephen Schneider
Utilizes assets from VisionX v4 provided by Dr. Anthony Reeves and Cornell University.

3. Shell Parameter Refinement Algorithm:

NAME:

LogicTrain- creates a mask of a tissue lesion from dermoscopic images according to inputted parameters. Outputs the Jaccard Index to the command line.

SYNOPSIS:

LogicTrain [fname=<filename>] [ftype=<filter type>] [xkernel=<x-kernel dimension>] [ykernel=<y-kernel dimension>] [okernel=<opening kernel radius>] [ckernel=<closing kernel radius>] [fmethod=<fill method>] [mergemethod=<merging method>]

DESCRIPTION:

LogicTrain creates a mask of a tissue lesion from dermoscopic images according to inputted parameters

Placeholder

OPTIONS:

fname=	color .jpg input image filename
ftype=	filter type (Mean, Gaussian, Median)
xkernel=	x-kernel dimension/standard deviation for filter
ykernel=	y-kernel dimension/standard deviation for filter
okernel=	opening kernel radius
ckernel=	closing kernel radius
fmethod=	fill method (Single, Multiple)

mergemethod= color channel merging logic (NotHighOrLow, LowUHigh, BlueOnly, HighUHighNLow)

AUTHORS:

Johnathan Maynard, Aswin Raghav Nirmaleswaran, Stephen Schneider
Utilizes assets from VisionX v4 provided by Dr. Anthony Reeves and Cornell University.

SOURCE SCRIPT:

```
#!/bin/sh
pname=LogicTrain
# this script outputs the jaccard index using various parameters
# utilize an adaptive proportional kernel for filtering

# syntax: LogicTrain <fname> <ftype> <xkernel> <ykernel> <okernel> <ckernel> <fmethod>
<mergemethod>

fname=$1 # Image filename
ftype=$2 # Filter type (Mean, Gaussian, or Median)
xkernel=$3 # x-kernel proportion (1 for no filtering) Note: kernel=xdim/xkernel
ykernel=$4 # y-kernel proportion (1 for no filtering)
okernel=$5 # Region opening kernel (1 for no opening)
ckernel=$6 # Region closing kernel (1 for no closing)
fmethod=$7 # Fill method (Single or Multiple)
mergemethod=$8 # Merging method (NotHighorLow, LowUHigh, BlueOnly, HighUHighNLow)

gaussx=$xkernel # Gaussian x-standard deviation
gaussy=$ykernel # Gaussian y-standard deviation

r=0
g=0
b=0
jac=0

## file conversion ##
# convert jpeg color image to .vx byte image
vfmt if="$fname.jpg" of="$fname.vx" -jpeg

## color channel extraction ##
# red
vchan if="$fname.vx" of="r$fname.vx" c=1
# green
vchan if="$fname.vx" of="g$fname.vx" c=2
# blue
```

```
vchan if="$fname.vx" of="b$fname.vx" c=3

## get x and y dimension ##
vcc xDim.c -o xDim
vcc yDim.c -o yDim
xdim=$(xDim if="r$fname.vx")
ydim=$(yDim if="r$fname.vx")

## create proportional kernels ##
# find proportional x-kernel
xkernel=$(expr $xdim / $xkernel)
# verify that the resulting kernel is odd
rem=$(( $xkernel % 2 ))
if [ $rem -eq 0 ]
then
xkernel=$xkernel - 1
fi
# find proportional y-kernel
ykernel=$(expr $ydim / $ykernel)
# verify that the resulting kernel is odd
rem=$(( $ykernel % 2 ))
if [ $rem -eq 0 ]
then
ykernel=$ykernel - 1
fi
# assign kernels
meanx=$xkernel # Mean x-kernel dimension
meany=$ykernel # Mean y-kernel dimension
medianx=$xkernel # Median x-kernel dimension
mediany=$ykernel # Median y-kernel dimension

case $ftype in
Mean)
## perform mean filtering ##
# red
cp "r$fname.vx" temp.vx
vmean if=temp.vx of="r$fname.vx" xs=$meanx xy=$meany
# green
cp "g$fname.vx" temp.vx
vmean if=temp.vx of="g$fname.vx" xs=$meanx xy=$meany
# blue
cp "b$fname.vx" temp.vx
vmean if=temp.vx of="b$fname.vx" xs=$meanx xy=$meany;;
```

```
Gaussian)
## perform gaussian filtering ##
# red
cp "r$fname.vx" temp.vx
vgfilt if=temp.vx of="r$fname.vx" xs=$gaussx ys=$gaussy zs=0
# green
cp "g$fname.vx" temp.vx
vgfilt if=temp.vx of="g$fname.vx" xs=$gaussx ys=$gaussy zs=0
# blue
cp "b$fname.vx" temp.vx
vgfilt if=temp.vx of="b$fname.vx" xs=$gaussx ys=$gaussy zs=0;;
Median)
## perform median filtering ##
# red
cp "r$fname.vx" temp.vx
vmedian if=temp.vx of="r$fname.vx" xs=$medianx xy=$mediany
# green
cp "g$fname.vx" temp.vx
vmedian if=temp.vx of="g$fname.vx" xs=$medianx xy=$mediany
# blue
cp "b$fname.vx" temp.vx
vmedian if=temp.vx of="b$fname.vx" xs=$medianx xy=$mediany;;
esac

## histogram equalization ##
# red
cp "r$fname.vx" temp.vx
vhisteq if=temp.vx of="r$fname.vx"
# green
cp "g$fname.vx" temp.vx
vhisteq if=temp.vx of="g$fname.vx"
# blue
cp "b$fname.vx" temp.vx
vhisteq if=temp.vx of="b$fname.vx"

## perform otsu's method thresholding ##
# red
vthresh if="r$fname.vx" of="or$fname.vx"
# green
vthresh if="g$fname.vx" of="og$fname.vx"
# blue
vthresh if="b$fname.vx" of="ob$fname.vx"
```

```
## color mask merging ##
case $mergemethod in
NotHighOrLow)
# merge the mask with the lowest value with the inverse of the mask with the highest value
vcc nMask.c -o nMask
r=$(nMask if="or$fname.vx")
b=$(nMask if="ob$fname.vx")
g=$(nMask if="og$fname.vx")
if [ $r -ge $b -a $r -ge $g ]
then
   # inverse r
   vpix -neg if="or$fname.vx" of=temp.vx
   if [ $g -ge $b ]
   then
         echo not r or b
         # find union
         vembed if=temp.vx ig="ob$fname.vx" of="lo$fname.vx" -merge
   else
         echo not r or g
         # find union
         vembed if=temp.vx ig="og$fname.vx" of="lo$fname.vx" -merge
   fi
elif [ $b -ge $r -a $b -ge $g ]
then
   # inverse b
   vpix -neg if="ob$fname.vx" of=temp.vx
   if [ $g -ge $r ]
   then
         echo not b or r
         # find union
         vembed if=temp.vx ig="or$fname.vx" of="lo$fname.vx" -merge
   else
         echo not b or g
         # find union
         vembed if=temp.vx ig="og$fname.vx" of="lo$fname.vx" -merge
   fi
else
  # inverse g
   vpix -neg if="og$fname.vx" of=temp.vx
   if [ $b -ge $r ]
   then
         echo not g or r
         # find union
```

```
        vembed if=temp.vx ig="or$fname.vx" of="lo$fname.vx" -merge
    else
        echo not g or b
        # find union
        vembed if=temp.vx ig="ob$fname.vx" of="lo$fname.vx" -merge
    fi
fi
## inverse result ##
cp "lo$fname.vx" temp.vx
vpix -neg if=temp.vx of="lo$fname.vx";;
LowUHigh)
# interesect the mask with the highest value with the union of the two masks with the lowest values
vcc nMask.c -o nMask
r=$(nMask if="or$fname.vx")
b=$(nMask if="ob$fname.vx")
g=$(nMask if="og$fname.vx")
if [ $r -ge $b -a $r -ge $g ]
then
echo b or g and r
# find union of b and g
vembed if="ob$fname.vx" ig="og$fname.vx" of=temp2.vx -merge
# take negative
vpix -neg if=temp2.vx of=temp.vx
# find intersection with r
vembed if=temp.vx ig="or$fname.vx" of="lo$fname.vx" -and
elif [ $b -ge $r -a $b -ge $g ]
then
echo r union g or b
# find union of r and g
vembed if="or$fname.vx" ig="og$fname.vx" of=temp2.vx -merge
# take negative
vpix -neg if=temp2.vx of=temp.vx
# find intersection with b
vembed if=temp.vx ig="ob$fname.vx" of="lo$fname.vx" -and
else
echo rUbNg
# find union of r and b
vembed if="or$fname.vx" ig="ob$fname.vx" of=temp2.vx -merge
# take negative
vpix -neg if=temp2.vx of=temp.vx
# find intersection with g
vembed if=temp.vx ig="og$fname.vx" of="lo$fname.vx" -and
fi;;
```

```
BlueOnly)
# utilize only the blue channel for analysis
echo blue channel isolation
cp "ob$fname.vx" temp.vx
# inverse channel
vpix -neg if=temp.vx of="lo$fname.vx";;
HighUHighNLow)
# negative union of the two highest channels intersected with the inverse lowest channel
vcc nMask.c -o nMask
r=$(nMask if="or$fname.vx")
b=$(nMask if="ob$fname.vx")
g=$(nMask if="og$fname.vx")
if [ $r -le $b -a $r -le $g ]
then
# take negative of b and g
vpix -neg if="ob$fname.vx" of=temp.vx
vpix -neg if="og$fname.vx" of="lo$fname.vx"
# find union of not b and not g
vembed if=temp.vx ig="lo$fname.vx" of=temp2.vx -merge
# find intersection with not r
vpix -neg if="or$fname.vx" of=temp.vx
vembed if=temp2.vx ig=temp.vx of="lo$fname.vx" -and
elif [ $b -le $r -a $b -le $g ]
then
# take negative r and g
vpix -neg if="or$fname.vx" of=temp.vx
vpix -neg if="og$fname.vx" of="lo$fname.vx"
# find union of not r and not g
vembed if=temp.vx ig="lo$fname.vx" of=temp2.vx -merge
# find intersection with not b
vpix -neg if="ob$fname.vx" of=temp.vx
vembed if=temp2.vx ig=temp.vx of="lo$fname.vx" -and
else
# take negative r and b
vpix -neg if="or$fname.vx" of=temp.vx
vpix -neg if="ob$fname.vx" of="lo$fname.vx"
# find union of not r and not b
vembed if=temp.vx ig="lo$fname.vx" of=temp2.vx -merge
# find intersection with not g
vpix -neg if="og$fname.vx" of=temp.vx
vembed if=temp2.vx ig=temp.vx of="lo$fname.vx" -and
fi;;
esac
```

```
## perform opening ##
# create kernel
vgenim of=temp.vx c= x=$okernel
# erode
vsf if="lo$fname.vx" of=temp2.vx k=temp.vx -e
# create kernel
vgenim of=temp.vx c= x=$okernel
# dilate
vsf if=temp2.vx of="lo$fname.vx" k=temp.vx -d

## perform closing ##
# create kernel
vgenim of=temp.vx c= x=$ckernel
# dilate
vsf if="lo$fname.vx" of=temp2.vx k=temp.vx -d
# create kernel
vgenim of=temp.vx c= x=$ckernel
# erode
vsf if=temp2.vx of="lo$fname.vx" k=temp.vx -e

## fill enclosed voids ##
cp "lo$fname.vx" temp.vx
case $fmethod in
Single)
vbfilt if=temp.vx of="lo$fname.vx" -d;;
Multiple)
v3bfill if=temp.vx of="lo$fname.vx" -2d -d;;
esac

## display parameters ##
echo "Image file = $fname"
echo "Filter type = $ftype"
case $ftype in
Mean)
echo "Mean x-kernel dimmension = $meanx"
echo "Mean y-kernel dimmension = $meany";;
Median)
echo "Median x-kernel dimmension = $medianx"
echo "Median y-kernel dimmension = $mediany";;
Gaussian)
echo "Gaussian x-standard deviation = $gaussx"
echo "Gaussian y-standard deviation = $gaussy";;
```

```
esac
echo "Region opening kernel = $okernel"
echo "Region closing kernel = $ckernel"
echo "Fill method = $fmethod"
echo "Merge method = $mergemethod"

## find Jaccard index ##
vcc jacind.c -o jacind
vfmt if="$fname""_Segmentation.png" of="tru$fname.vx" -png
jacind if="lo$fname.vx" tf="tru$fname.vx"
jac=$(jacind if="lo$fname.vx" tf="tru$fname.vx")

echo "$jac" >> output_file.txt

## convert final image to gif ##
vxto gif "lo$fname.vx"

/*****************************************************************/
/* jacind:     Calculate the Jaccard Index between two image masks   */
/*****************************************************************/

#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */

VXparam_t par[] =            /* command line structure               */
{
{  "if=",   0,   " input file, jacind: calculate Jaccard index"},
{  "tf=",   0,   " true file, jacind: calculate Jaccard index"},
{  "-v",    0,   " (verbose) print intermediate information"},
{   0,      0,   0} /* list termination */
};
#define  IVAL   par[0].val
#define  TVAL   par[1].val
#define  VFLAG  par[2].val

main(argc, argv)
int argc;
char *argv[];
{

    Vfstruct (im);                  /* input image structure          */
    Vfstruct (tm);              /* true image structure       */
    int y,x;                        /* index counters                 */
    int i;
```

```
    int insct = 0;              /* intersection of input and true */
    int inp = 0;          /* input values                */
    int tru = 0;          /* true values                 */
    float jac = 0;              /* Jaccard index               */

    VXparse(&argc, &argv, par);    /* parse the command line         */

      Vfread( &im, IVAL);
      Vfread( &tm, TVAL);
      if ( im.ylo != tm.ylo || im.xlo != tm.xlo || im.yhi != tm.yhi ||
im.xhi != tm.xhi ) {
            fprintf (stderr, "error: input and true image are not of the
same size\n");
            exit (1);
      }
      if ( im.type != VX_PBYTE ) {
               fprintf (stderr, "error: input image not byte type\n");
               exit (1);
        }
            if ( tm.type != VX_PBYTE ) {
               fprintf (stderr, "error: true image not byte type\n");
               exit (1);
        }

        /* compute the number of values */
        for (y = im.ylo; y <= im.yhi; y++) {
            for (x = im.xlo; x <= im.xhi; x++) {
                 if (im.u[y][x] >= 1) {
                inp++;
             }
             if (tm.u[y][x] >= 1) {
                 tru++;
             }
             if (im.u[y][x] >= 1 && tm.u[y][x] >= 1) {
                 insct++;
             }
          }
      }
jac = tru+inp-insct;
jac = insct/jac;

if(VFLAG) {
fprintf (stderr, "inp = %d insct = %d tru = %d\n",inp,insct,tru);
}
```

```
fprintf(stderr, "true positive = %d false negative = %d false positive =
%d\n             Jaccard index = %f\n",insct, (inp-insct), (tru-insct),
jac);

    exit(0);
}

/*****************************************************************/
/* nMask:     Find size of mask                                  */
/*****************************************************************/

#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */

VXparam_t par[] =            /* command line structure               */
{
{  "if=",   0,   " input file, nMask: find size of mask"},
{   0,      0,   0} /* list termination */
};
#define  IVAL   par[0].val

main(argc, argv)
int argc;
char *argv[];
{

    Vfstruct (im);                /* input image structure          */
    int y,x,i=0;                  /* index counters                 */

    VXparse(&argc, &argv, par);   /* parse the command line         */

    Vfread( &im, IVAL);
    if ( im.type != VX_PBYTE ) {
      fprintf (stderr, "error: input image not byte type\n");
      exit (1);
    }

    for (y = im.ylo; y <= im.yhi; y++) {
        for (x = im.xlo; x <= im.xhi; x++) {
          if (im.u[y][x] != 0){
            i++;
          }
      }
    }
```

```
    fprintf(stdout,"%d",i);

    exit(0);
}
```