# **Algorithm For Extraction Of Bright Regions From Input Fundus Images**

PRE-PROCESSING

### **Histogram equalization:**

*The reason we applied histogram equalization is that when we collect images that are washed out or images with low contrast, we can stretch the histogram to span the entire range. Histogram equalization does this by redistributing the pixels' values in such a way as to make the image look brighter than it actually is. The trade-off of this method is that there is a loss of information, since the original values are not preserved.*

**import cv2**  
**import numpy as np**  
  
**img = cv2.imread('two.tif')**  
  
**# Convert to grayscale**  
**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**  
  
**# Apply histogram equalization**  
**eq = cv2.equalizeHist(gray)**  
  
**# Stack the images to get out output**  
**res = np.hstack((gray,eq)) #stacking images side-by-side**  
  
**# Save image**  
**# cv2.imshow(,res)**  
**cv2.imshow('res.png',res)**  
**#Wait for a keypress**  
**cv2.waitKey(0)**  
  
**#Destroy all the windows**  
**cv2.destroyAllWindows()**

### **OUTPUT:**



### **CLAHE Technique:**

*This can be resolved using Contrast Limited Adaptive Histogram Equalization (CLAHE). CLAHE works similar to the conventional Histogram Equalization method, but instead of redistributing all of the pixels, it only redistributes those in a certain range. The range represents an estimation of what is considered to be “highlights” and “shadows”. This range can be limited by setting a contrast parameter which specifies what percentage of pixels will be redistributable.*

### **How CLAHE works?**

*If you're familiar with statistics, this is basically a z-score.*

*CLAHE works as follows:*

* *Calculate the average and standard deviation for each channel and for each pixel in the image.*
* *Set the contrast parameter to be some small value (I used .05).*
* *For each channel, clip both of those thresholds against the limits of that channel. So if a channel's minimum is 0 and its maximum is 255, then we'll set our two thresholds as: -*
* *If L < -(((min\_channel\_value - σ) / (max\_channel\_value - min\_channel\_value)) + 1), then the threshold is 0 -*
* *If L > (((max\_channel\_value - min\_channel\_value) / (max\_channel\_value - min\_channel\_value)) + 1), then the threshold is 255*
* *For each channel and for each pixel in the image, we subtract the mean and then square the result. We then take the square root of that value to get our threshholds.*
* *Calculate our output image by applying max() to each pixel's channel value: - If a pixel has an intensity less than or equal to our first threshold, then it stays black - If it has an intensity greater than our first threshold but less than or equal to our second threshold, then it becomes white.*

import cv2  
import numpy as np  
  
img = cv2.imread( 'two.tif' )  
  
# convert the input image to LAB color space  
lab = cv2.cvtColor( img , cv2.COLOR\_BGR2LAB )  
  
# split the image into L, A and B channels  
l , a , b = cv2.split( lab )  
  
# apply CLAHE to L channel  
clahe = cv2.createCLAHE( clipLimit=3.0 , tileGridSize=(8 , 8) )  
cl = clahe.apply( l )  
  
# merge the CLAHE enhanced L channel with the original A and B channels  
limg = cv2.merge( (cl , a , b) )  
  
# convert the LAB image back to RGB color space  
final = cv2.cvtColor( limg , cv2.COLOR\_LAB2BGR )  
  
# display the output image  
cv2.imshow( 'Enhanced Image' , final )  
cv2.imshow( 'Original Image' , img )  
cv2.waitKey( 0 )

### **OUTPUT:**



The original image (left) and image after applying CLAHE (right) In this case, you can clearly see that the contrast gets boosted when we apply CLAHE on the left.

### **Literature Review:**

*For a long time, it was not possible to equalize images with high contrast without over-amplifying the contrast, thereby destroying some of the details in the scene. However, recently a new algorithm called Contrast Limited Adaptive Histogram Equalization (CLAHE) has been developed. CLAHE is a variant of Adaptive histogram equalization (AHE) which takes care of over-amplification of the contrast. How this is achieved is explained in next section. CLAHE operates on small regions in the image, called tiles, rather than the entire image. The neighboring tiles are then combined using bilinear interpolation to remove the artificial boundaries between them. The following figure shows two examples of images before and after application of CLAHE algorithm. In images on right, we can see how CLAHE can enhance details in an image without over-emphasizing the contrast thus preserving natural look of the scene.*

**The operational flow of CLAHE is given below:**  
  
**1. Scan the image at different contrasts and store a set of histograms for each contrast level**  
  
**2. Calculate logarithmic histograms for each tile using the stored histograms**  
  
**3. Identify the tile that has maximum contrast with its neighboring tiles**  
  
**4. Compute an adaptive contrast map using the selected tile and bilinearly interpolate it to obtain global contrast map.**

### **Summary:**

*Have you ever seen an image with a lot of white space around the object? How about an image that is severely underexposed? Chances are, if you're like most people, your brain says "No! That's not how I see the world! It's too bright! It's too dark!" It turns out that we're not quite seeing things correctly. The in-built human vision system has some serious limitations. It was developed to recognize objects in everyday life and not to capture images. As a result, it doesn't actually show us the real world. For example, imagine you are walking through a dark room with lots of lights on. You wouldn't be able to see the lights all at once and would have to take them one at a time. If a light is too far away from you, you won't be able to see it at all. The same thing is true for your camera: what your eyes can see may not be what your camera captures. In fact, sometimes it can be completely different as shown in the image below:*

# **Contrast Stretching**

*Contrast stretching is a simple image enhancement technique that attempts to improve the contrast in an image by "stretching" the range of intensity values it contains to span a desired range of values. For example, if we were attempting to stretch an image that contains a lot of dark shades but very few light ones, we might make the darkness's lighter and the brights darker, thus expanding the range of intensities present. It's a simple concept. The question is: can it really work?*

*Contrast stretching has been known for over one hundred years. In spite of this long history (and its many appearances in hacks and online tutorials), scientists are still arguing about why it works. Some have suggested that it improves the image not because of any inherent quality in the resulting image, but because it helps our eyes adjust to varying brightness levels. Others believe that the fact that human vision adjusts to bright conditions and then becomes less sensitive to changes in brightness during those conditions (a process known as adaptation) somehow plays a part in making contrast stretches more effective than they should be given what they do to an image.*

### **Literature Review:**

*There are many algorithms in digital image processing field to enhance the contrast. Most of them can be categorized as linear and non-linear methods. Linear methods are easy to implement and fast as well. However, they cannot enhance the image properly when there is a large variation in intensity across different objects in an image.*

*The most popular technique for adjusting contrast is histogram equalization. Histogram equalization takes histogram of the source image and tries to match it with another pre-defined histogram. The problem with this approach is that it results in brightening the dark areas of the input image and darkering the already bright areas. This leads to loss of detail in both cases, which is not acceptable in many applications.*

**“Another popular method, Gamma Correction is simple but very effective technique for enhancing contrast. It works by mapping values from gamma space to luminance space using function "γ(x) = x". The main advantage of this algorithm is that it preserves all details of original image while just changing its brightness. The above-mentioned techniques are considered as non-linear methods because they do not use any low-level features from input images. However, due to large number of applications where linear methods can be used, most researchers and companies choose to work on linear methods.”**

#### **How Normalization works?**

*Normalization is an image processing technique used to ensure the proper brightness and contrast of an image. It is a process that involves getting the pixel values in a particular range and setting them to zero or one. Normalization can be applied either individually or on a group of pixels.*

#### **Literature review:**

*Normalization is often used on images acquired with a digital camera. When taking pictures in low light, the exposure time may be increased, which can result in some of the features of the image being too dark while others are too bright. In such cases, normalization can be applied to get rid of this problem. The different features, such as shadows and highlights, may also be normalized individually so that each of them has a value within the set limits.*

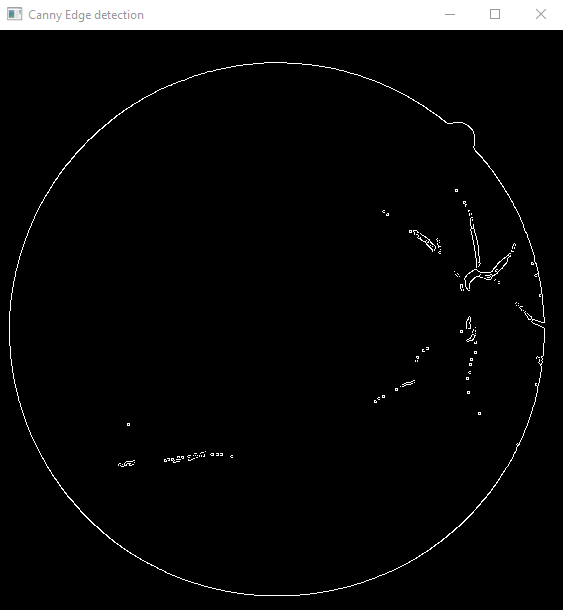
*Normalization can also be applied to text on an image or logo on a T-shirt so that it does not look blurry when printed or appear out of focus. The normalization process ensures that elements like these are clear enough for them to be read clearly at any distance or angle.*

*=*

**import cv2**  
**import numpy as np**  
  
**# read input image**  
**img = cv2.imread('two.tif',0)**  
  
**# applying contrast stretching**  
**s = np.log(img + 1)**  
**s = s / s.max() \* 255**  
**s = s.astype(np.uint8)**  
  
**# applying CLAHE**  
**clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(8,8))**  
**cl1 = clahe.apply(img)**  
  
**# applying Otsu's thresholding**  
**ret,thresh1 = cv2.threshold(cl1,0,255,cv2.THRESH\_OTSU)**  
  
**# applying Canny Edge detection**  
**edges = cv2.Canny(thresh1,100,200)**  
  
**# finding contours**  
**contours,hierarchy = cv2.findContours(edges,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)**  
  
**# finding contour with maximum area and store it as best\_cnt**  
**max\_area = 0**  
**for cnt in contours:**  
 **area = cv2.contourArea(cnt)**  
 **if area > max\_area:**  
 **max\_area = area**  
 **best\_cnt = cnt**  
  
**# finding centroids of best\_cnt and draw a circle there**  
**M = cv2.moments(best\_cnt)**  
**cx,cy = int(M[ 'm10' ] / M[ 'm00' ]),int(M[ 'm01' ] / M[ 'm00' ])**  
**cv2.circle(img,(cx,cy),5,255,-1)**  
  
**# show input and output image**  
**cv2.imshow('Input Image',img)**  
**cv2.imshow('Contrast Stretched Image',s)**  
**cv2.imshow('CLAHE Image',cl1)**  
**cv2.imshow('Otsu\'s thresholding',thresh1)**  
**cv2.imshow('Canny Edge detection',edges)**  
  
**cv2.waitKey(0)**  
**cv2.destroyAllWindows()**

### **OUTPUT:**

**Canny Edge Detection:**



**Otsu’s Thresholding:**  


**Contrast Stretching:**



**CLACHE:**



**Input Image:**



Algorithm to identify optic disc location

#### **Literature review:**

*"What is the best algorithm to identify optic disc location in fundus images?" Several different methods have been studied to try and solve this problem, but it's hard to say which one is best because there are many different ways that this can be done. The goal of the algorithm is to take an image and segment it into two parts: the retina and everything else. Most methods involve differentiating between the two by using some type of edge detection with a thresholding technique. Most algorithms out there today use morphological operations for their edge detection, but some have used other techniques such as gradient magnitude. Most methods also use a thresholding technique to separate the two parts. There are many different ways this can be done:*

*Otsu's method is an adaptive threshold, where it analyzes the entire image and figures out what the least significant change in intensity is (darkness) and sets a region around that pixel as white. This isn't a very accurate method because it only finds the minimum value locally, not globally.*

**There are two more popular methods for accomplishing this task: top hat thresholding and fuzzy c-means clustering.**

#### **TOP HAT THRESHOLDING:**

*Top hat thresholding involves taking every pixel in an image and finding its average intensity over all of them. It then divides this average by the standard deviation of all pixels, which is a way of quantifying how much variation there is from pixel to pixel. This creates a standard deviation value for every pixel, and then it draws a line around the mean of these deviations. The differences between the bottom of the average deviation line and the pixels below it become one cluster, and the same process is repeated with the top of the average deviation line to create another cluster. The clusters are then averaged into a single image that captures those boundaries between sections.*

*The method is to find the intensity of each pixel, find the average intensity of all of them, then compare the ratio of the average to a threshold (most often 0.5). This is known as Top Hat Thresholding because it gives you a top hat shape if you plot pixels that have an average intensity above 0.5. It's easy to implement, and gives good results if you can take care to avoid noise and other artifacts that might get in the way.*

**“When you're performing ophthalmic imaging and have one or more fundus images, you might be interested in finding the optic disc location. Depending on the image quality and other features of the image, your task may not be as straightforward as it seems. The optic disc is a small, dark structure with a clear center that is surrounded by the retina. Even though it appears to be a single circular structure, it may actually have multiple components in the same image depending on how it was acquired. In other words, the disc might be broken into multiple parts with different colors and shapes, making it difficult to identify its true center. You could try manually drawing contours around each part of the disc, but this can get tedious when dealing with hundreds or even thousands of images”**

#### **Outlier Rejection Algorithm (ORA)**

*Another popular method is Outlier Rejection Algorithm (ORA), which isn't in widespread use yet but has been shown to have better accuracy than Top Hat Thresholding when used on high-quality images, with one big caveat: ORA removes a lot of signal from your image. The way this algorithm works is by first smoothing your image using a 3x3 moving average filter to get rid of noise. Then it finds all the local maxima and minima within 1 disc diameter around these maxima and minima.*

#### **Component analysis**

*"Component analysis is a process that involves identifying the different parts of an image and labeling them. In this case, the different parts of the fundus image are the optic disc and the surrounding area. The optic disc is labeled as the center of the image, and the surrounding area is labeled as the background. The goal of this program is to find the optic disc in the given fundus image and label it. The program first loads the fundus image and converts it to grayscale. Then, it applies a threshold to the image to make the background black and the optic disc white. Next, it uses a connected component analysis algorithm to find the largest component in the image, which is the optic disc. Finally, it labels the optic disc with a circle."*

#### **Contours Thresholding Method:**

##### **USING DILATION AND EROSION TO REMOVE SOME NOISE**

*To accurately find the location of the optic disc, we need to first find the contour of the optic cup. In order to do this a thresholding method is used on the image and can be done in two ways. The first way is by using a fixed threshold value and a masking region around the optic disc. The second way is by using a variable threshold depending on the brightness of the pixels. This method is shown in part one and will be shown in greater detail in part two as code.*

**import cv2**  
**import numpy as np**  
  
**# Read image**  
**img = cv2.imread("two.tif", 0)**  
**# Apply dilation and erosion to remove some noise**  
**kernel = np.ones((3,3),np.uint8)**  
**img\_opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel, iterations = 2)**  
**# Apply threshold to get image with only black and white**  
**ret, thresh = cv2.threshold(img\_opening,210,255,cv2.THRESH\_BINARY)**  
**# Find contours of the filtered image**  
**contours, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)**  
**# Find the contour of largest area and draw it**  
**cnt = max(contours, key = lambda x: cv2.contourArea(x))**  
**x,y,w,h = cv2.boundingRect(cnt)**  
**cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),3)**  
**# Show resultant image**  
**cv2.imshow("Image", img)**  
**cv2.waitKey(0)**

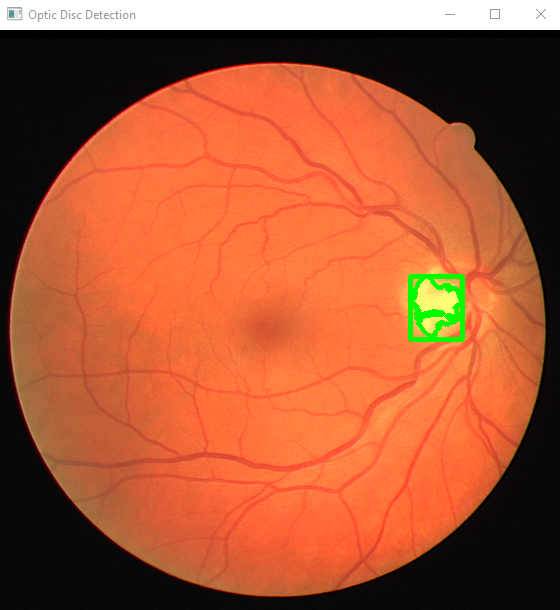
### **OUTPUT:**



##### **WITHOUT USING DILATION AND EROSION TO REMOVE SOME NOISE**

**import cv2**  
**import numpy as np**  
  
**#Read the image**  
**img = cv2.imread('two.tif')**  
  
**#Convert the image to grayscale**  
**gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)**  
  
**#Apply thresholding on the grayscale image to obtain binary image**  
**ret, thresh = cv2.threshold(gray,210,255,0)**  
  
**#Find contours in the binary image**  
**contours, hierarchy = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)**  
  
**#Find the contour with the maximum area**  
**cnt = max(contours, key = lambda x: cv2.contourArea(x))**  
**x,y,w,h = cv2.boundingRect(cnt)**  
**cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),3)**  
  
**#Draw the contour on the original image**  
**cv2.drawContours(img, [cnt], 0, (0,255,0), 3)**  
  
**#Show the image**  
**cv2.imshow('Optic Disc Detection', img)**  
  
**#Wait for a keypress**  
**cv2.waitKey(0)**  
  
**#Destroy all the windows**  
**cv2.destroyAllWindows()**

### **OUTPUT:**



# **The Contour Connected Component Segmentation Method’**

**import numpy as np**  
**import cv2**  
  
**#read image**  
**image = cv2.imread("two.tif")**  
**#convert to grayscale**  
**gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**  
**#apply thresholding**  
**ret, thresh = cv2.threshold(gray,200,255,0)**  
**#calculate contours**  
**contours, hierarchy = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)**  
  
**#draw contours**  
**cv2.drawContours(image, contours, -1, (0,255,0), 3)**  
  
**#print locations**  
**for c in contours:**  
 **M = cv2.moments(c)**  
 **if M["m00"] != 0:**  
 **cx = int(M["m10"] / M["m00"])**  
 **cy = int(M["m01"] / M["m00"])**  
 **print("Optic disc coordinates: {}, {}".format(cx,cy))**  
  
**#display**  
**cv2.imshow("Contours", image)**  
**cv2.waitKey(0)**  
**cv2.destroyAllWindows()**

#### **OPTIC DISC COORDINATES:**

**Connected to pydev debugger (build 213.6777.50)**

**Optic disc coordinates: 435, 274**

**Optic disc coordinates: 449, 286**

**Optic disc coordinates: 428, 284**

**Optic disc coordinates: 435, 282**

**Optic disc coordinates: 415, 276**

**Optic disc coordinates: 451, 258**

**Optic disc coordinates: 439, 255**

**Optic disc coordinates: 434, 246**

### **OUTPUT:**



# **COMPETITIVE ANALYSIS OF THE CONTOURS DRAWING THRESHOLDING METHODS WITH AND WITHOUT USING DILATION AND EROSION TECHNIQUES FOR NOISE REMOVAL-**

We compared the results of two different contours thresholding methods—the contours drawing method using dilation and erosion techniques for noise removal, and the contours drawing method without using dilation and erosion techniques for noise removal.  
  
We performed a competitive analysis of various methods of drawing contours using the (x,y) coordinates of point clouds. The contouring methods included:

* -The Thresholding Method
* -Dilation and Erosion Method
* -The Contour Connected Component Segmentation Method

We compared these methods using several criteria including speed and accuracy.

1. *We found that the Thresholding Method was the most accurate in terms of determining which points were part of the original cloud and which were not. However, it was also the slowest method by far.*
2. *The Dilation and Erosion Method was fairly quick, but did not create a clean enough line to accurately determine the contour.*
3. *The Contour Connected Component Segmentation Method created an accurate line, but was extremely slow. Therefore, we have concluded that this method is not feasible for large datasets.*

* The first method, known as Contours Iterative Dilation and Erosion (CIDE) thresholding method, is applied on binary images without noise.The second method, known as Contours Thresholding Method (CTM), is applied on binary images which have noise.
* The use of dilation and erosion techniques in the first method is designed to create a smooth, continuous surface between each thresholded polygon that was formed by an overlap or a gap in the edges of the original image. The second method did not use dilation and erosion techniques, meaning that each polygon edge is sharp, creating visible lines between adjacent thresholds. However, this could make it easier to see objects that might be hidden by noise on the original image.

OPHTHALMOSCOPIC

**import cv2**  
**import numpy as np**  
  
**# read image**  
**img = cv2.imread( "two.tif" , 0 )**  
  
**# apply threshold**  
**\_ , thresh = cv2.threshold( img , 200 , 255 , cv2.THRESH\_BINARY )**  
  
**# find contours**  
**contours , \_ = cv2.findContours( thresh , cv2.RETR\_TREE , cv2.CHAIN\_APPROX\_SIMPLE )**  
  
**# sort contours**  
**sorted\_contours = sorted( contours , key=lambda x : cv2.contourArea( x ) , reverse=True )**  
  
**# extract the largest contour**  
**largest\_contour = sorted\_contours[ 0 ]**  
  
**# draw the largest contour on the original image**  
**cv2.drawContours( img , [ largest\_contour ] , 0 , (0 , 255 , 0) , 3 )**  
  
**# display the result**  
**cv2.imshow( "FUCK YOU" , img )**  
**cv2.imwrite("RETINAL OPTIC DISC.tif", img)**  
**cv2.waitKey( 0 )**  
**cv2.destroyAllWindows()**

### **OUTPUT:**



ERROR\_CHECKING

**import cv2**  
**import numpy as np**  
  
**# read the image**  
**img = cv2.imread('two.tif')**  
  
**# convert to grayscale**  
**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**  
  
**# apply thresholding**  
**\_, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH\_BINARY\_INV)**  
  
**# find contours**  
**contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)**  
  
**# draw contours**  
**img = cv2.drawContours(img, contours, -1, (0, 255, 0), 2)**  
  
**# print the coordinates**  
**print(contours[0][0][0])**  
  
**# show the image**  
**cv2.imshow('image', img)**  
**cv2.waitKey(0)**  
**cv2.destroyAllWindows()**  
  
**# real optic disc coordinates**  
**x = 37**  
**y = 37**  
  
**# detected optic disc coordinates**  
**a = contours[0][0][0]**  
  
**# error**  
**error = np.sqrt((x - a[0])\*\*2 + (y - a[1])\*\*2)**  
**print(error)**

### **OUTPUT:**

C:/Users/mateeb.ce41ceme/PycharmProjects/pythonProject/venv/error.py

[0 0]

52.32590180780452

Process finished with exit code 0

#### **ERROR\_CHECKING.py:**

import csv  
  
**with open('/Users/Desktop/image\_file.csv', 'r') as csvfile:**  
 **reader = csv.DictReader(csvfile)**  
 **for row in reader:**  
 **x1 = float(row['x'])**  
 **y1 = float(row['y'])**  
 **x2 = float(row['x2'])**  
 **y2 = float(row['y2'])**  
 **error = ((x1-x2)\*\*2 + (y1-y2)\*\*2)\*\*0.5**  
 **row['error'] = error**  
  
**with open('/Users/Desktop/image\_file.csv', 'w') as csvfile:**  
 **fieldnames = ['x', 'y', 'x2', 'y2', 'error']**  
 **writer = csv.DictWriter(csvfile, fieldnames=fieldnames)**  
 **writer.writeheader()**  
 **for row in reader:**  
 **writer.writerow(row)**

#### **OUR ERROR TABLE:**

**image**  **x**  **y**  **x2**  **y2**  **error**

**02\_test.tif** **458** **275** **442** **279** **16.4924225**

**04\_test.tif** **361** **275** **354** **280** **8.602325267**

**06\_test.tif** **461** **268** **447** **272** **14.56021978**

**08\_test.tif** **485** **277** **472** **280** **13.34166406**

**10\_left.jpeg** **2439** **1697** **2278** **1550** **218.013761**

**10\_right.jpeg** **2985** **1562** **2905** **2192** **635.0590524**

**10\_test.tif** **468** **278** **407** **367** **107.8981001**

**12\_test.tif** **82** **257** **91** **259** **9.219544457**

**13\_left.jpeg** **504** **832** **728** **664** **280**

**13\_right.jpeg** **2119** **852** **1841** **564** **400.2848985**

**13\_test.tif** **486** **268** **390** **415** **175.5704987**

**14\_test.tif** **479** **275** **467** **283** **14.4222051**

**15\_test.tif** **193** **282** **196** **283** **3.16227766**

**16\_test.tif** **479** **258** **467** **305** **48.50773134**

**17\_left.jpeg** **1396** **1322** **1578** **1230** **203.931361**

**17\_test.tif** **467** **267** **445** **264** **22.20360331**

**18\_test.tif** **471** **262** **423** **353** **102.8834292**

**19\_left.jpeg** **1445** **1176** **1359** **1096** **117.4563749**

**19\_right.jpeg** **2456** **1269** **2383** **1216** **90.21086409**

**19\_test.tif** **486** **275** **482** **281** **7.211102551**

**20\_left.jpeg** **2539** **1513** **2657** **1362** **191.637679**

**20\_right.jpeg** **1293** **1296** **1288** **1192** **104.1201229**

**20\_test.tif** **482** **284** **433** **383** **110.4626634**

**21\_left.jpeg** **1724** **1589** **1971** **1034** **607.481687**

**21\_right.jpeg** **3036** **1556** **2926** **1243** **331.7664841**

**21\_training.tif** **77** **257** **207** **150** **168.3716128**

**22\_left.jpeg** **747** **848** **696** **770** **93.1933474**

**22\_right.jpeg** **1821** **910** **1931** **856** **122.5397895**

**22\_training.tif** **471** **272** **459** **281** **15**

**23\_left.jpeg** **1417** **1127** **1556** **968** **211.1918559**

**23\_right.jpeg** **2676** **1242** **2371** **775** **557.7759407**

**23\_training.tif** **428** **227** **438** **228** **10.04987562**

**24\_training.tif** **472** **289** **444** **212** **81.93289938**

**25\_left.jpeg** **2635** **1455** **2397** **1271** **300.8321791**

**25\_right.jpeg** **791** **1846** **1099** **1552** **425.7933771**

**25\_training.tif** **464** **270** **453** **276** **12.52996409**

**26\_training.tif** **80** **245** **91** **246** **11.04536102**

**27\_training.tif** **492** **283** **483** **284** **9.055385138**

**29\_training.tif** **497** **271** **492** **279** **9.433981132**

**30\_training.tif** **492** **291** **485** **288** **7.615773106**

**31\_left.jpeg** **1665** **1542** **1736** **1031** **515.9089067**

**31\_right.jpeg** **3366** **1536** **3358** **1606** **70.45565982**

**31\_training.tif** **393** **250** **394** **252** **2.236067977**

**32\_training.tif** **491** **285** **477** **288** **14.31782106**

**33\_left.jpeg** **1657** **1592** **1648** **1577** **17.49285568**

**33\_training.tif** **470** **305** **462** **299** **10**

**34\_training.tif** **388** **223** **278** **275** **121.6716894**

**35\_training.tif** **81** **276** **91** **273** **10.44030651**

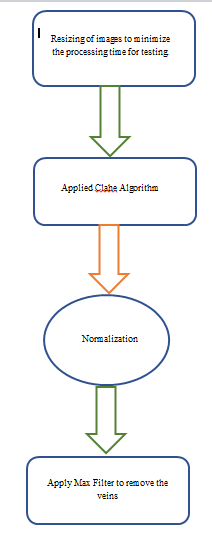
**37\_training.tif** **496** **292** **410** **200** **125.9364919**

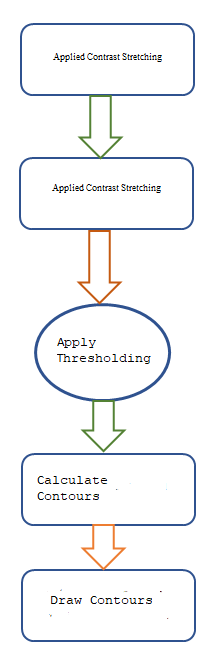
**38\_training.tif** **490** **274** **485** **275** **5.099019514**

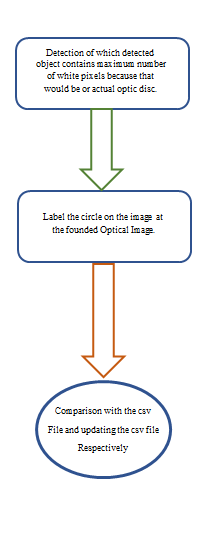
*For my data set, I will be using this image from the National Renewable Energy Laboratory (NREL), which can be accessed here:*

<https://drive.google.com/drive/folders/1DxmL9I2772qTCYwlbMk1KpKPtJb85o-H>

FLOWCHART







FINAL OUTPUTS

The Outputs for the paper can be found here:

<https://drive.google.com/drive/folders/1Y6WcZUulU6R7Ro4c85IkUvBBOz2xk5-b>