National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

EE 433: Digital Image Processing

Amal Saqib (282496)

Muhammad Hadi (305774)

Class: BSCS 9C

Lab 11: Segmentation of Retinal Blood Vessels

Date: 6th December 2021

Time: 2.00Pm to 5.00Pm

Instructor: Dr. Imran Malik

In [95]:
```
import cv2 as cv
import numpy as np
from PIL import Image
from scipy.ndimage import rotate
from scipy import stats
```

Pre processing

In [96]:
```
#--------------------PreProcessing-----------------------------------------#

def calculateBGValue(input_arr,filter_arr):
    multiplication_res=input_arr*filter_arr
    return multiplication_res.sum()


def performbgNormalization(img_array,filter_arr):
    padding=(filter_arr[0].size-1)//2
    padded_array=np.pad(img_array,padding) #apply zero padding
    (height,width)=padded_array.shape
    output_array=padded_array.copy()
    for y in range(padding,height-padding):
        for x in range(padding,width-padding):
            #array responsible for making subset array
            array=padded_array[y-padding:y+padding+1,x-padding:x+padding+1]
            output_array[y][x]=calculateBGValue(array, filter_arr)

     #remove padding
    output_array=output_array[padding:height-padding,padding:width-padding]
    return output_array

def performPreProcessing(image,size):
    #convert image to gray scale
    #grayImage=cv.cvtColor(image,cv.COLOR_BGR2GRAY)
    grayImage=image
    #perform backgroundNormalization
    largeKernel=np.ones(shape=(size,size))/(size*size) #making box filter kernel
    bgNormalizedImage=performbgNormalization(grayImage, largeKernel)

    image=image.astype(int)
    bgNormalizedImage=bgNormalizedImage.astype(int)
    res=abs(image-bgNormalizedImage)
    res=res.astype(np.uint8)
    return res
```

Thin Vessel Enhancement

In [97]:
```
#----------------------Thin Vessel Enchancement----------------------#

def getDetectionFilters():
    return [
        [
            [-1/6,-1/6,-1/6],
```

```
            [2/6,2/6,2/6],
            [-1/6,-1/6,-1/6]
        ],
        [
            [-1/6,-1/6,2/6],
            [-1/6,2/6,-1/6],
            [2/6,-1/6,-1/6]
        ],
        [
            [-1/6,2/6,-1/6],
            [-1/6,2/6,-1/6],
            [-1/6,2/6,-1/6]
        ],
        [
            [2/6,-1/6,-1/6],
            [-1/6,2/6,-1/6],
            [-1/6,-1/6,2/6]
        ],
    ]

def getHighestFilterResponse(arr,filters):
    max_res=0
    for f in filters:
        mul=arr*f
        mul_sum=mul.sum()
        max_res=max(max_res,mul_sum)
    return max_res

def performThinVesselEnhancement(img_array):
    detectionFilters= getDetectionFilters()
    padding=1
    padded_array=np.pad(img_array,padding).astype(int) #apply zero padding
    (height,width)=padded_array.shape
    output_array=padded_array.copy()
    for y in range(padding,height-padding):
        for x in range(padding,width-padding):
            #array responsible for making subset array
            array=padded_array[y-padding:y+padding+1,x-padding:x+padding+1]
            output_array[y][x]+=getHighestFilterResponse(array, detectionFilters)

     #remove padding
    output_array=output_array[padding:height-padding,padding:width-padding]
    output_array=np.where(output_array>255,255,output_array)
    return output_array.astype(np.uint8)
```

Candidate Detection

In [98]:

```
def applyGaussianFilterThingy(array,filter_arr):
    array=array.astype(int)
    filter_arr=filter_arr.astype(int)
    height=len(filter_arr)
    width=len(filter_arr[0])
    h=(height-1)//2
    w=(width-1)//2
    padding=max(h,w)
    padded_array=np.pad(array,padding) #apply zero padding
    (height,width)=padded_array.shape
```

```
    output_array=padded_array.copy()

    for y in range(padding,height-padding):
        for x in range(padding,width-padding):
            #array responsible for making subset array
            array=padded_array[y-h:y+h+1,x-w:x+w+1]
            mul=array*filter_arr
            mul_sum=mul.sum()
            output_array[y][x]=mul_sum

     #remove padding
    output_array=output_array[padding:height-padding,padding:width-padding]
    return output_array


def horizontalParse(array,direction):
    (height,width)=array.shape
    final_arr=np.zeros(shape=(height,width))
    startX=0
    startY=0
    endX=0
    endY=0
    if(direction==1):
        startX=0
        startY=0
        endX=width-3
        endY=height
    elif(direction==2):
        startX=0
        startY=0
        endX=width-3
        endY=height-3
    elif(direction==3):
        startX=0
        startY=0
        endX=width
        endY=height-3
    elif(direction==4):
        startX=3
        startY=3
        endX=width
        endY=height-3
    for y in range(startY,endY):
        for x in range(startX,endX):
            subset=[]
            if(direction==1):
                subset=[array[y][x],array[y][x+1],array[y][x+2],array[y][x+3]]
            elif(direction==2):
                subset=[array[y][x],array[y+1][x+1],array[y+2][x+2],array[y+3][x+3]]
            elif(direction==3):
                subset=[array[y][x],array[y+1][x],array[y+2][x],array[y+3][x]]
            elif(direction==4):
                subset=[array[y][x],array[y-1][x-1],array[y-2][x-2],array[y-3][x-3]]

            avg=sum(subset)/4    #Calculate AVG
            match_found=False
            max_val=max([subset[0],subset[1],subset[2],subset[3]])
            min_val=min([subset[0],subset[1],subset[2],subset[3]])
```

```
            #Evaluating conditions
            if(avg<0 and subset[1]>0 and subset[2]<0 and subset[3]<0):
                match_found=True
            elif(subset[0]>0 and subset[1]>0 and subset[2]<0 and subset[3]<0):
                match_found=True
            elif(avg>0 and subset[0]>0 and subset[1]>0 and subset[2]<0):
                match_found=True
            elif(subset[0]>0 and subset[1]==0 and subset[2]<0):
                max_val=max([subset[0],subset[1],subset[2]])
                min_val=min([subset[0],subset[1],subset[2]])
                match_found=True


            #Assigning max+abs(lowest) to the highest intensity value
            if(match_found):
                if(max_val==subset[0]):
                    final_arr[y][x]=max_val+abs(min_val)
                if(max_val==subset[1]):
                    if(direction==1):
                        final_arr[y][x+1]=max_val+abs(min_val)
                    elif(direction==2):
                        final_arr[y+1][x+1]=max_val+abs(min_val)
                    elif(direction==3):
                        final_arr[y+1][x]=max_val+abs(min_val)
                    elif(direction==4):
                        final_arr[y-1][x-1]=max_val+abs(min_val)
                if(max_val==subset[2]):
                    if(direction==1):
                        final_arr[y][x+2]=max_val+abs(min_val)
                    elif(direction==2):
                        final_arr[y+2][x+2]=max_val+abs(min_val)
                    elif(direction==3):
                        final_arr[y+2][x]=max_val+abs(min_val)
                    elif(direction==4):
                        final_arr[y-2][x-2]=max_val+abs(min_val)
                if(max_val==subset[3]):
                    if(direction==1):
                        final_arr[y][x+3]=max_val+abs(min_val)
                    elif(direction==2):
                        final_arr[y+3][x+3]=max_val+abs(min_val)
                    elif(direction==3):
                        final_arr[y+3][x]=max_val+abs(min_val)
                    elif(direction==4):
                        final_arr[y-3][x-3]=max_val+abs(min_val)

    return final_arr

def candidateSelection(enhancement):
    initialFilter=np.array([
        [-1,-2,+0,+2,+1],
        [-2,-4,+0,+4,+2],
        [-1,-2,+0,+2,+1],
    ])

    initial_res=applyGaussianFilterThingy(enhancement,initialFilter)
    res=horizontalParse(initial_res,1)
    res=res.astype(np.uint8)

    quarter_rotate=rotate(initialFilter,angle=45,reshape=False)    #Rotate kernel by 45
    quarter_res=applyGaussianFilterThingy(enhancement,quarter_rotate)
```

```
    quarter_res=horizontalParse(quarter_res,2)
    quarter_res=quarter_res.astype(np.uint8)                    #Convert back to int form

    half_rotate=rotate(initialFilter,angle=90,reshape=False)   #Rotate kernel by 90
    half_res=applyGaussianFilterThingy(enhancement,half_rotate)
    half_res=horizontalParse(half_res,3)
    half_res=half_res.astype(np.uint8)

    almost_rotate=rotate(initialFilter,angle=135,reshape=False)    #Rotate kernel by 13
    almost_res=applyGaussianFilterThingy(enhancement,almost_rotate)
    almost_res=horizontalParse(almost_res,4)
    almost_res=almost_res.astype(np.uint8)

    return [res,quarter_res,half_res,almost_res]
```

Connection Of Candidate Points

In [99]:

```
def computeThresholdValue(img):
    #Filtering out background from image array
    candidate_arr=img[img != 0]
    mean=np.mean(candidate_arr,dtype=np.uint8)
    std=np.std(candidate_arr,dtype=np.uint8)
    mode = stats.mode(candidate_arr)
    thresh=mean-(std*mode[0])
    return thresh

def connectionOfCandidatePoints(img):
    thresh=computeThresholdValue(img)
    seed_arr=np.where(img<thresh,0,img)
    return seed_arr
```

Main

In [100…

```
image=cv.imread('a.png',cv.IMREAD_GRAYSCALE)
normalizedImage=performPreProcessing(image,25)
enhancement=performThinVesselEnhancement(normalizedImage)
```

In [101…

```
selection=candidateSelection(enhancement)
```

In [102…

```
Image.fromarray(connectionOfCandidatePoints(selection[0]))
```

Out[102…

In [103…

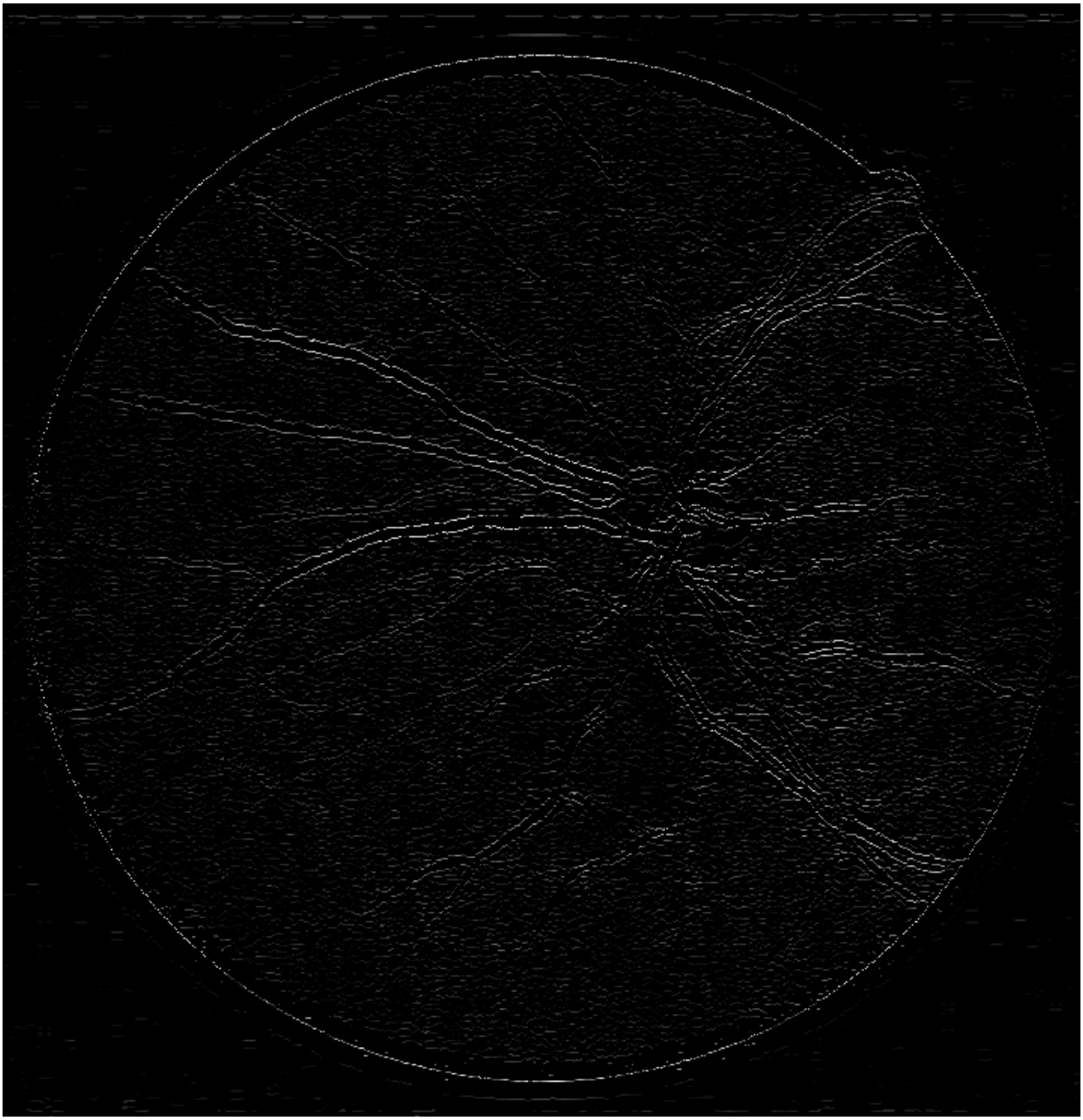
```
Image.fromarray(connectionOfCandidatePoints(selection[1]))
```

Out[103…

In [104…

```
Image.fromarray(connectionOfCandidatePoints(selection[2]))
```

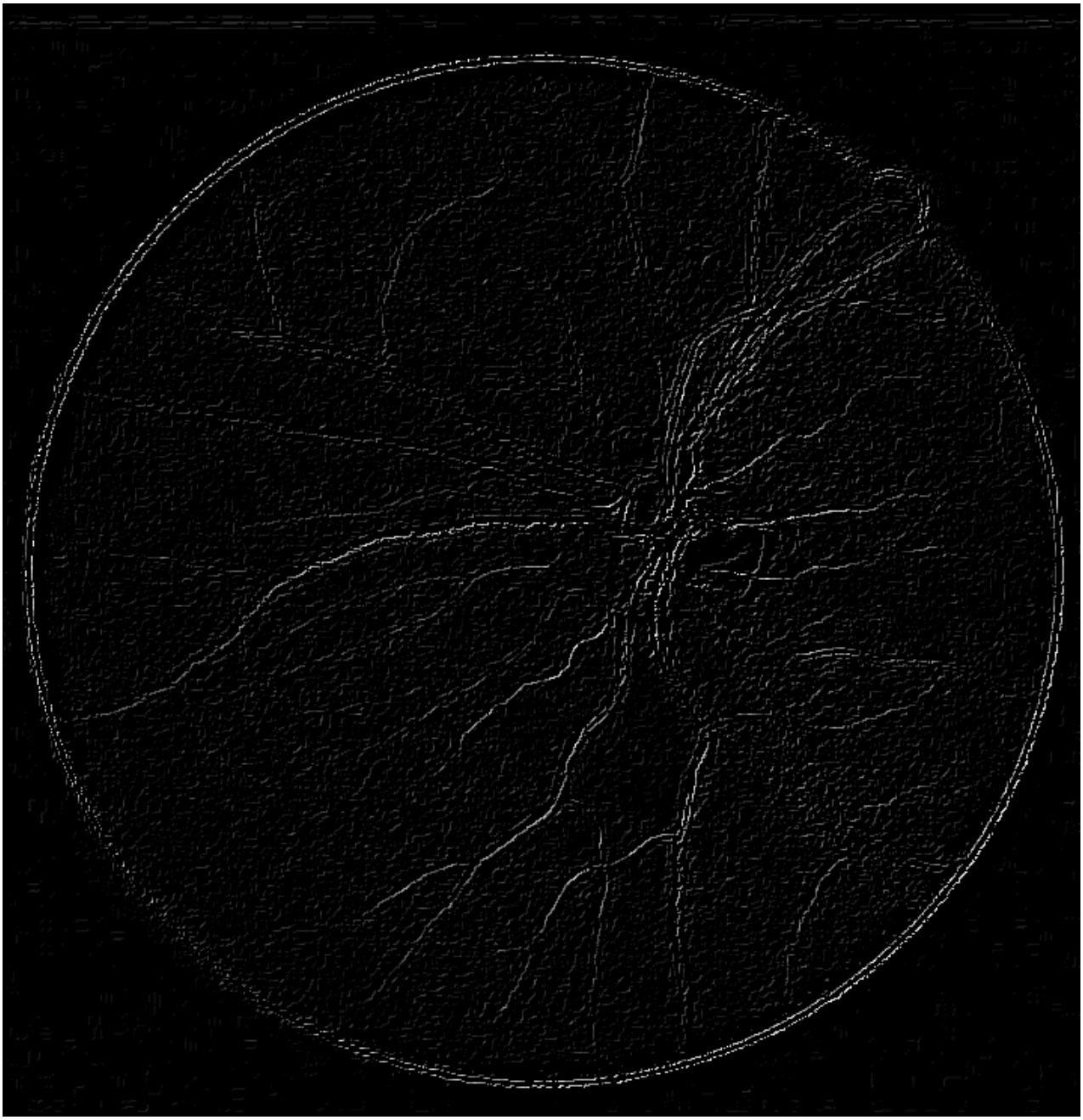
Out[104…



In [105…

```
Image.fromarray(connectionOfCandidatePoints(selection[3]))
```

Out[105…

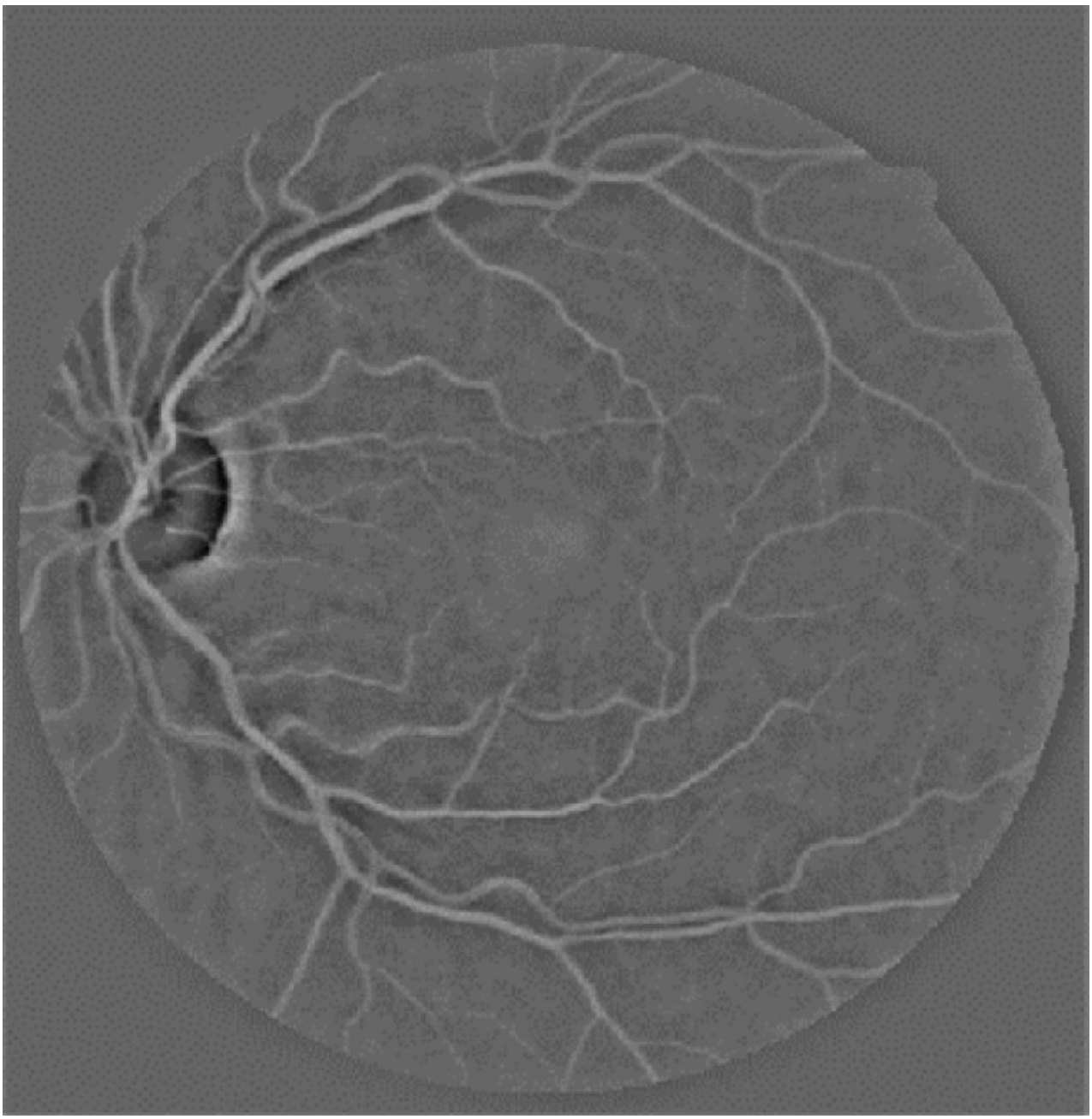


Task 2

In [1]:
```
from PIL import Image
import numpy as np
```

In [2]:
```
normalizedImage = Image.open('normalizedImage.jpg')
normalizedImage
```

Out[2]:



In [3]:
```
normalizedArray = np.asarray(normalizedImage)
normalizedArray
```

Out[3]:
```
array([[[121, 121, 121],
        [117, 117, 117],
```

```
        [110, 110, 110],
        ...,
        [117, 117, 117],
        [120, 120, 120],
        [123, 123, 123]],

       [[115, 115, 115],
        [111, 111, 111],
        [106, 106, 106],
        ...,
        [116, 116, 116],
        [120, 120, 120],
        [122, 122, 122]],

       [[113, 113, 113],
        [109, 109, 109],
        [104, 104, 104],
        ...,
        [113, 113, 113],
        [119, 119, 119],
        [122, 122, 122]],

       ...,

       [[108, 108, 108],
        [105, 105, 105],
        [103, 103, 103],
        ...,
        [117, 117, 117],
        [120, 120, 120],
        [125, 125, 125]],

       [[131, 131, 131],
        [125, 125, 125],
        [116, 116, 116],
        ...,
        [126, 126, 126],
        [132, 132, 132],
        [136, 136, 136]],

       [[139, 139, 139],
        [133, 133, 133],
        [125, 125, 125],
        ...,
        [138, 138, 138],
        [147, 147, 147],
        [149, 149, 149]]], dtype=uint8)
```

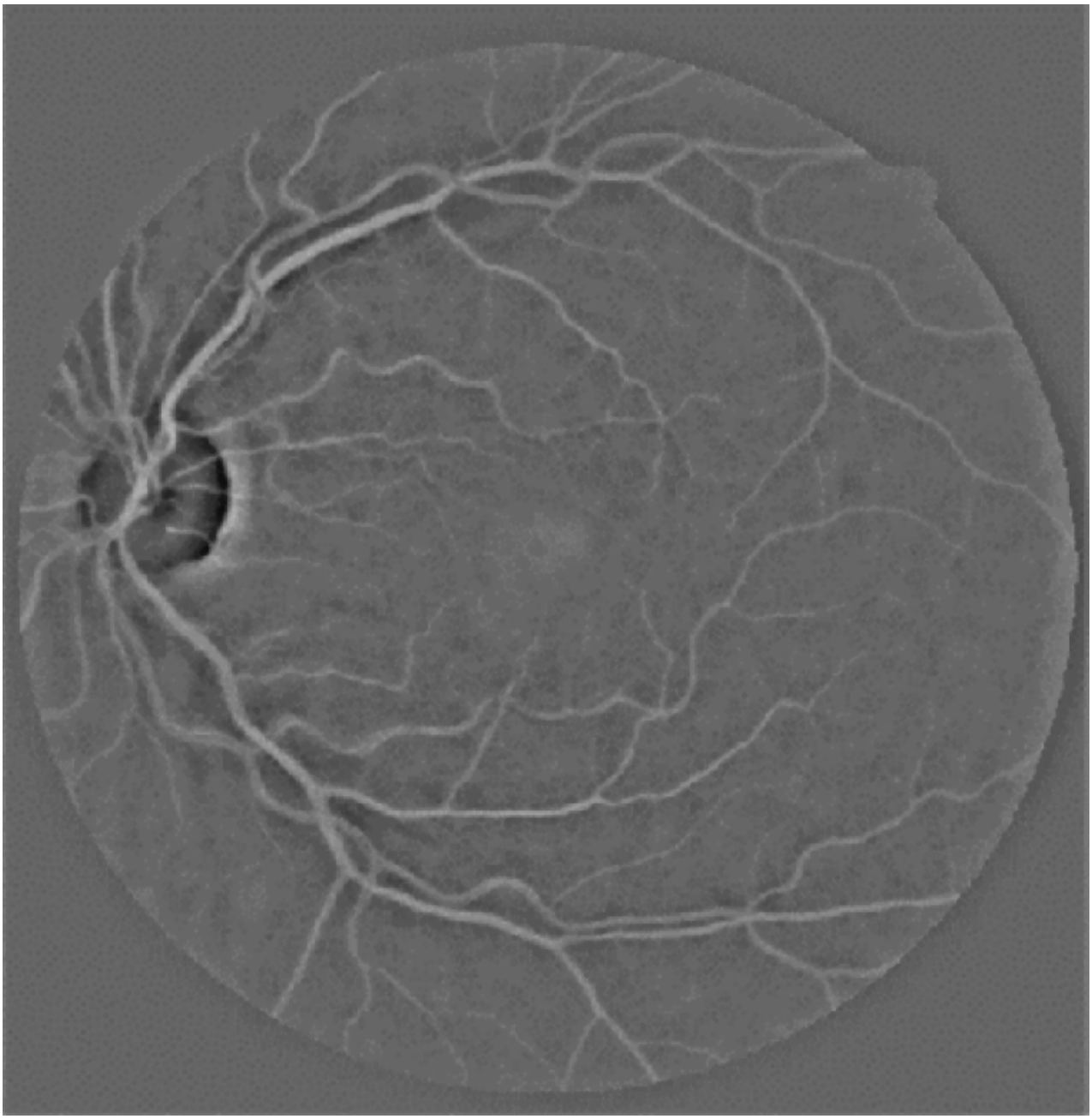
Multiscale Morphological Enhancement

In [4]:

```
import skimage.morphology
SE1 = skimage.morphology.disk(1)
SE2 = skimage.morphology.disk(2)
SE3 = skimage.morphology.disk(3)
SE4 = skimage.morphology.disk(4)
SE5 = skimage.morphology.disk(5)
SE6 = skimage.morphology.disk(6)
SE7 = skimage.morphology.disk(7)
SE8 = skimage.morphology.disk(8)
```

In [5]:
```
import cv2 as cv
closed = cv.morphologyEx(normalizedArray, cv.MORPH_CLOSE, SE1)
closedImage = Image.fromarray(closed)
closedImage
```

Out[5]:

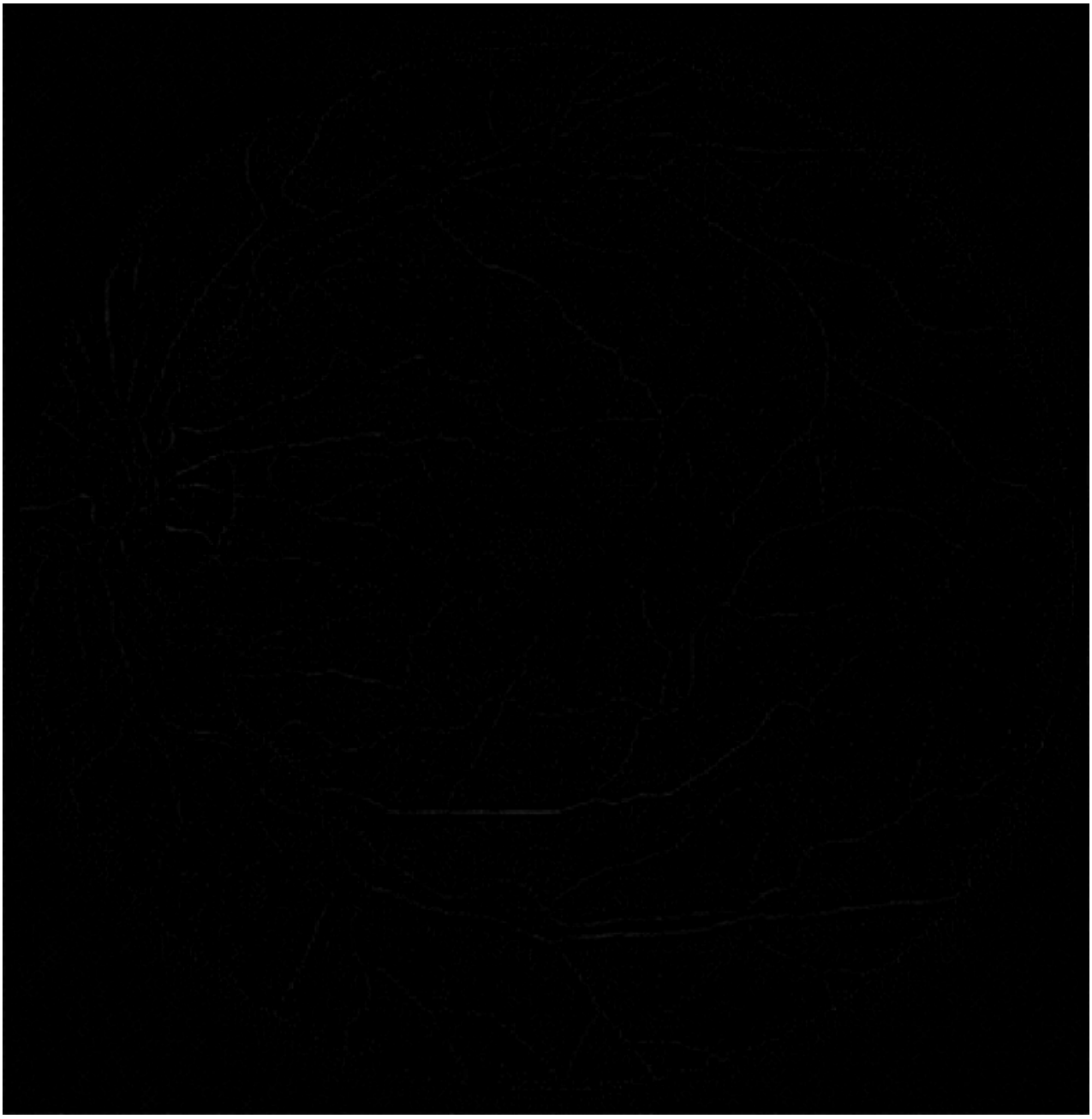


In [6]:
```
open1 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE1)
open2 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE2)
open3 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE3)
open4 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE4)
open5 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE5)
open6 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE6)
open7 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE7)
open8 = cv.morphologyEx(closed, cv.MORPH_OPEN, SE8)
```

In [7]:
```
topHat1 = normalizedArray - np.minimum(open1, normalizedArray)
topHat2 = normalizedArray - np.minimum(open2, normalizedArray)
topHat12 = (np.add(topHat1, topHat2) / 2).astype('uint8')
```

```
topHat12Image = Image.fromarray(topHat12)
topHat12Image
```

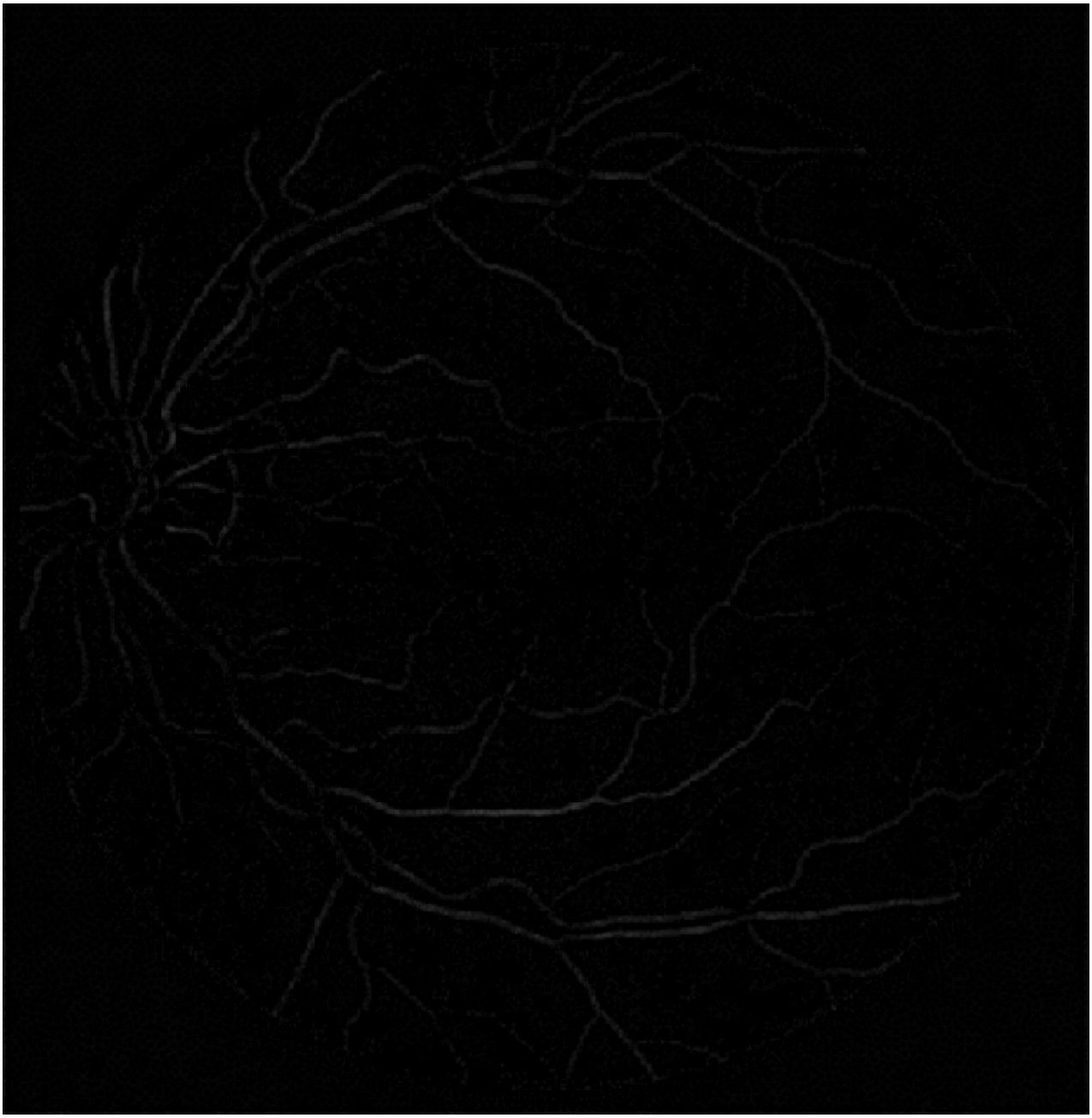
Out[7]:



In [8]:

```
topHat3 = normalizedArray - np.minimum(open3, normalizedArray)
topHat4 = normalizedArray - np.minimum(open4, normalizedArray)
topHat34 = (np.add(topHat3, topHat4) / 2).astype('uint8')
topHat34Image = Image.fromarray(topHat34)
topHat34Image
```

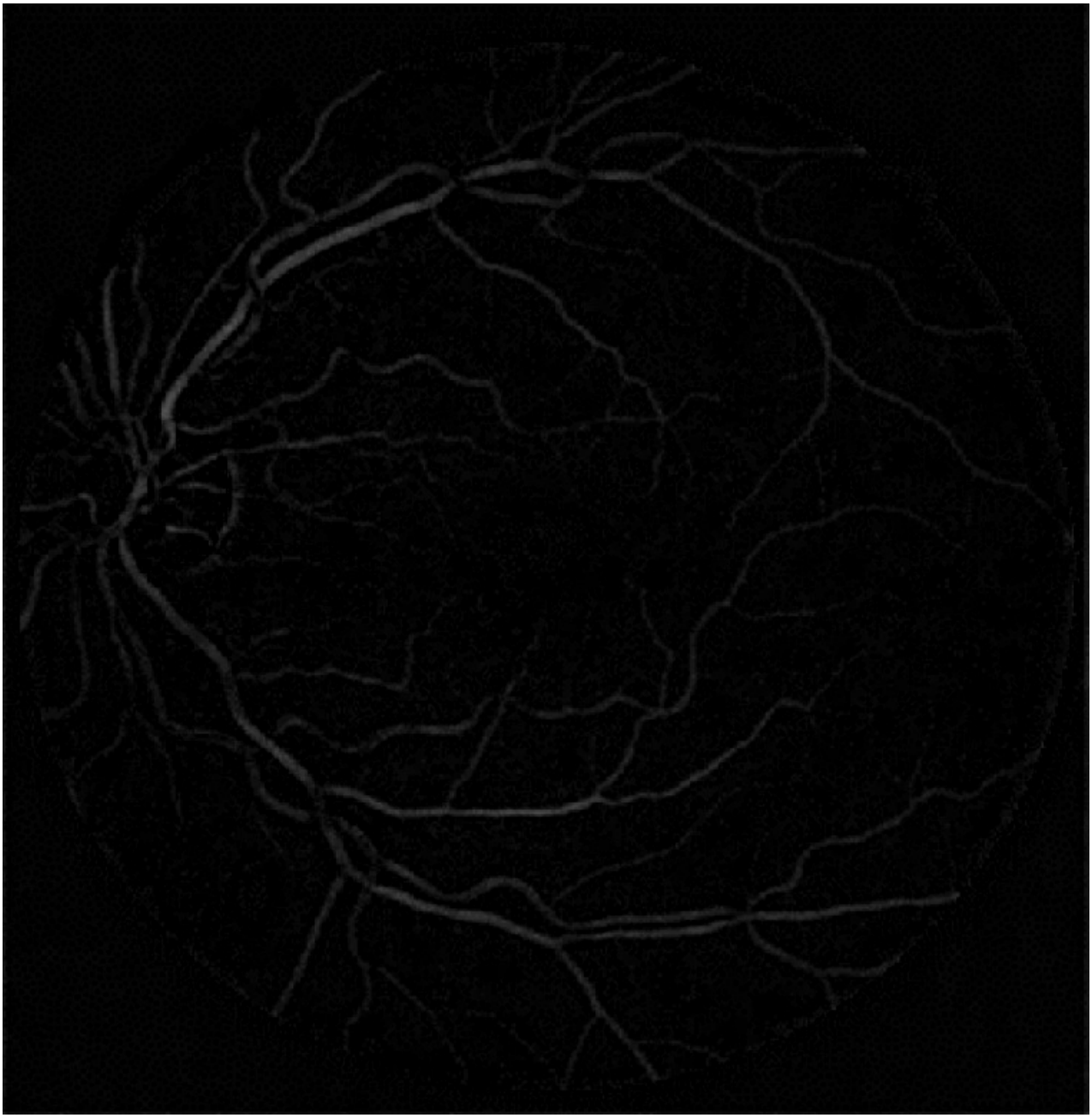
Out[8]:



In [9]:

```
topHat5 = normalizedArray - np.minimum(open5, normalizedArray)
topHat6 = normalizedArray - np.minimum(open6, normalizedArray)
topHat56 = (np.add(topHat5, topHat6) / 2).astype('uint8')
topHat56Image = Image.fromarray(topHat56)
topHat56Image
```

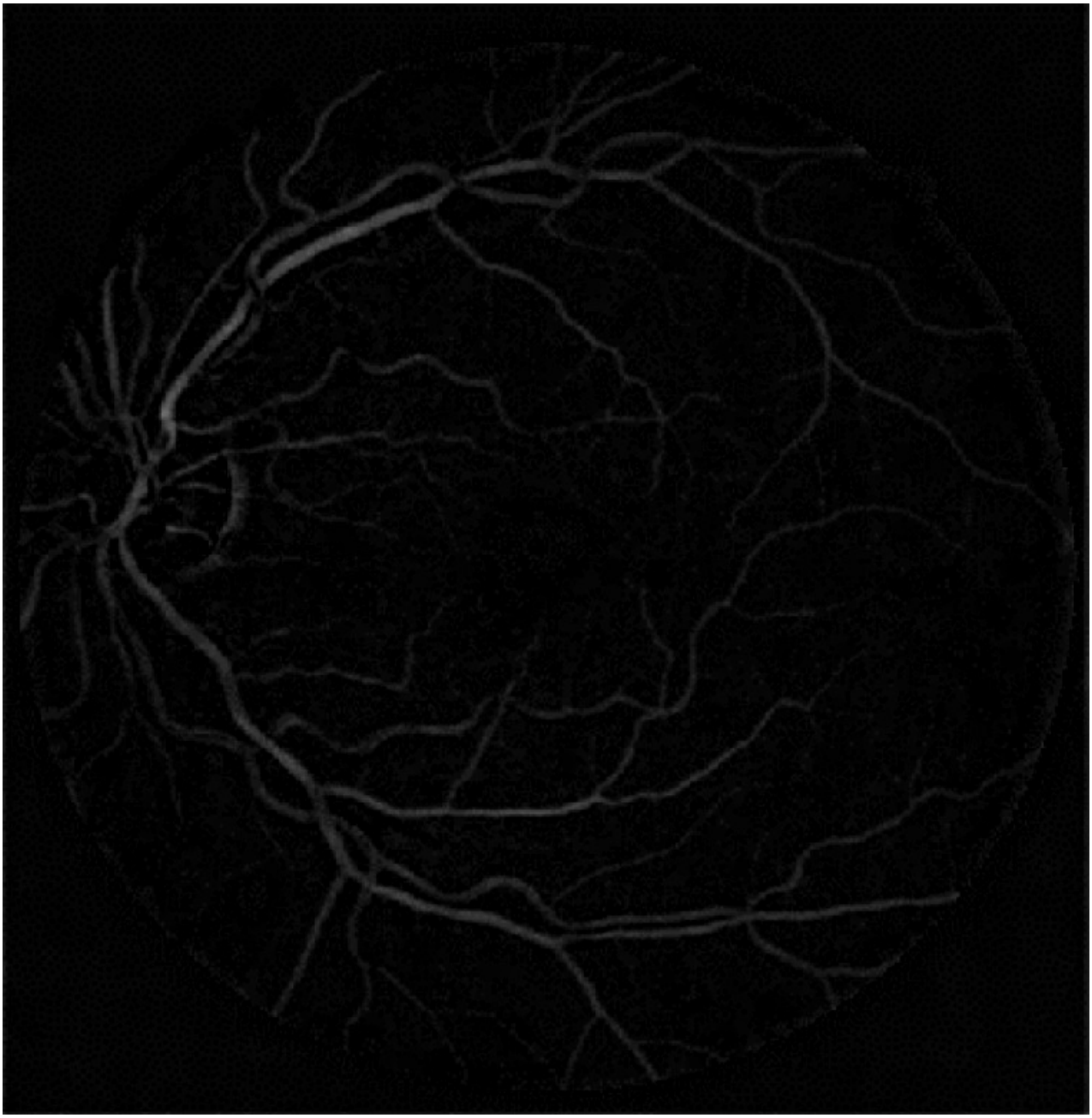
Out[9]:



In [10]:

```
topHat7 = normalizedArray - np.minimum(open7, normalizedArray)
topHat8 = normalizedArray - np.minimum(open8, normalizedArray)
topHat78 = (np.add(topHat7, topHat8) / 2).astype('uint8')
topHat78Image = Image.fromarray(topHat78)
topHat78Image
```

Out[10]:



Multiscale Reconstruction

In [11]:
```
nonNULL12 = topHat12[topHat12 != 0]
nonNULL12
```

Out[11]:
```
array([ 7,  7,  7, ..., 18, 18, 18], dtype=uint8)
```

In [12]:
```
nonNULL34 = topHat34[topHat34 != 0]
nonNULL34
```

Out[12]:
```
array([18, 18, 18, ..., 31, 31, 31], dtype=uint8)
```

In [13]:
```
nonNULL56 = topHat56[topHat56 != 0]
nonNULL56
```

Out[13]:
```
array([19, 19, 19, ..., 47, 47, 47], dtype=uint8)
```

In [14]:
```
nonNULL78 = topHat78[topHat78 != 0]
nonNULL78
```

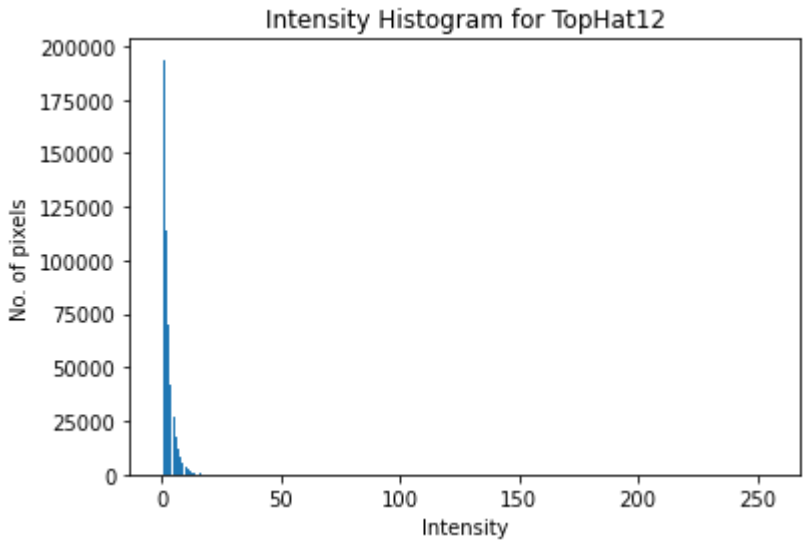
Out[14]:
```
array([22, 22, 22, ..., 51, 51, 51], dtype=uint8)
```

In [15]:
```
from matplotlib import pyplot as plt
```

In [16]:
```
count12 = np.zeros(shape = (256))
for x in range(len(nonNULL12)):
    count12[nonNULL12[x]] += 1
```

In [17]:
```
plt.title("Intensity Histogram for TopHat12")
plt.xlabel("Intensity")
plt.ylabel("No. of pixels")
plt.bar([i for i in range(256)], count12)
```

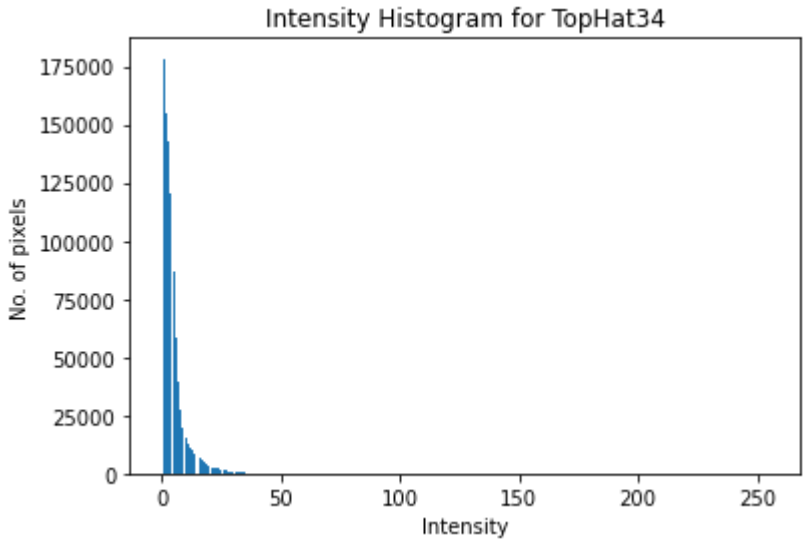
Out[17]:
```
<BarContainer object of 256 artists>
```



In [18]:
```
count34 = np.zeros(shape = (256))
for x in range(len(nonNULL34)):
    count34[nonNULL34[x]] += 1
```

In [19]:
```
plt.title("Intensity Histogram for TopHat34")
plt.xlabel("Intensity")
plt.ylabel("No. of pixels")
plt.bar([i for i in range(256)], count34)
```

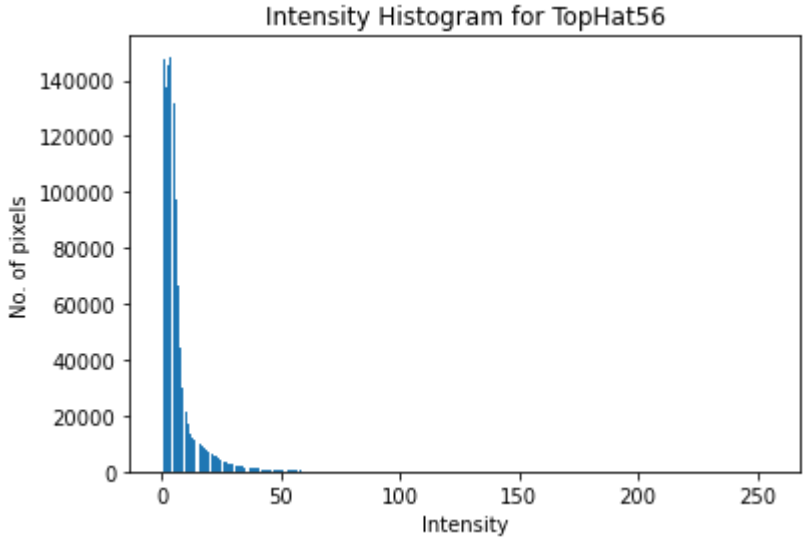
Out[19]:
```
<BarContainer object of 256 artists>
```



In [20]:
```
count56 = np.zeros(shape = (256))
for x in range(len(nonNULL56)):
    count56[nonNULL56[x]] += 1
```

In [21]:
```
plt.title("Intensity Histogram for TopHat56")
plt.xlabel("Intensity")
plt.ylabel("No. of pixels")
plt.bar([i for i in range(256)], count56)
```

Out[21]: <BarContainer object of 256 artists>



In [22]:
```
count78 = np.zeros(shape = (256))
for x in range(len(nonNULL78)):
    count78[nonNULL78[x]] += 1
```

In [23]:
```
plt.title("Intensity Histogram for TopHat78")
plt.xlabel("Intensity")
plt.ylabel("No. of pixels")
plt.bar([i for i in range(256)], count78)
```

Out[23]: <BarContainer object of 256 artists>