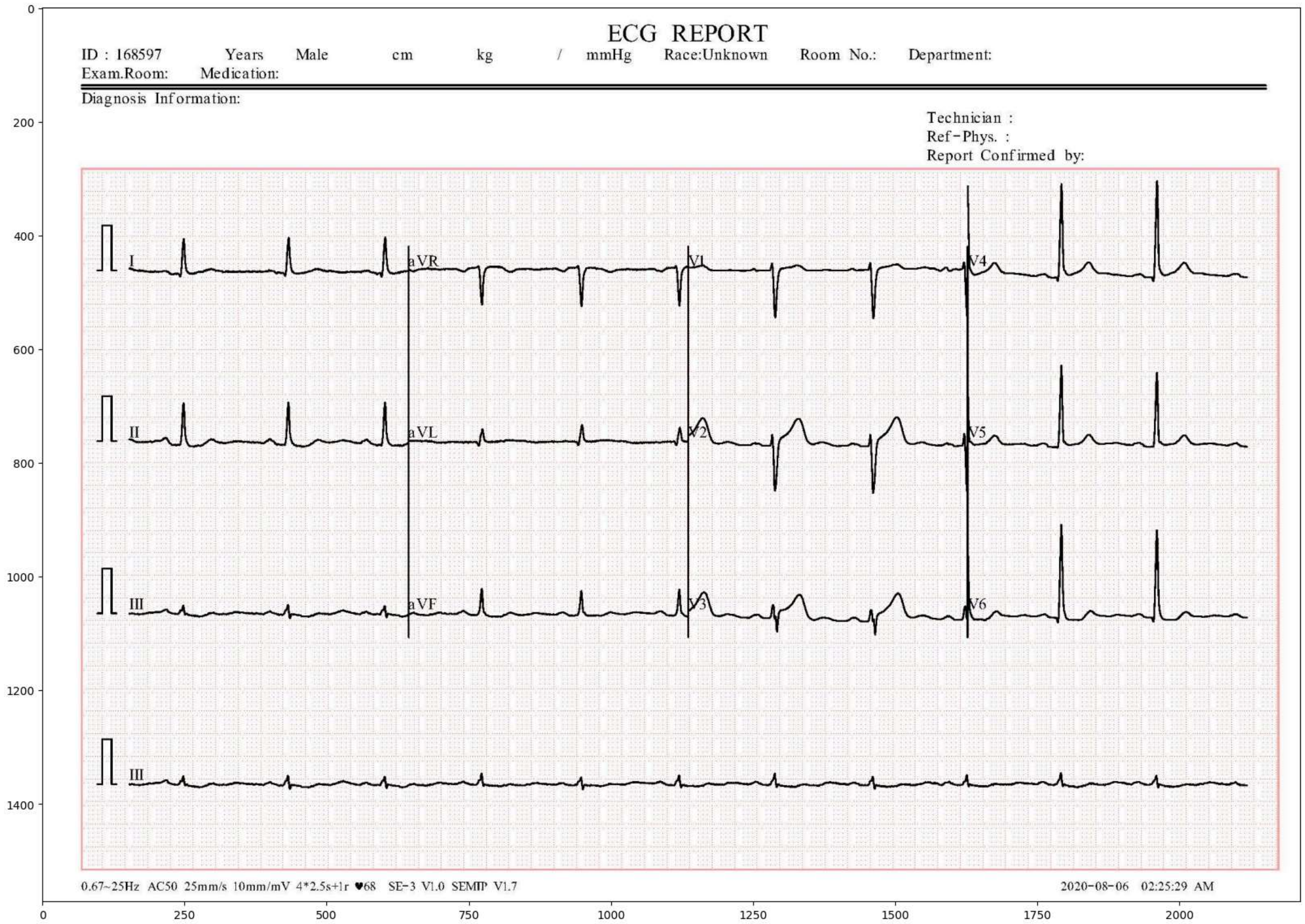
```
from google.colab import drive
import zipfile
import os

drive.mount('/content/drive')

zip_file_path = '/content/drive/MyDrive/HEART.zip'

extraction_path = '/content/extracted_files'

if not os.path.exists(extraction_path):
    os.makedirs(extraction_path)

try:
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extraction_path)
    print(f"Successfully extracted '{zip_file_path}' to '{extraction_path}'")
except FileNotFoundError:
    print(f"Error: Zip file not found at '{zip_file_path}'")
except zipfile.BadZipFile:
    print(f"Error: Invalid zip file at '{zip_file_path}'")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Successfully extracted '/content/drive/MyDrive/HEART.zip' to '/content/extracted_files'
```

```
from skimage.io import imread
from skimage import color
import matplotlib.pyplot as plt

fig0 , ax0 = plt.subplots()

fig0.set_size_inches(20, 20)

image=imread('/content/drive/MyDrive/Normal(2).jpg')

ax0.imshow(image)
plt.show()
```



ECG REPORT

ID : 168597 Years Male cm kg / mmHg Race:Unknown Room No.: Department:
Exam.Room: Medication:

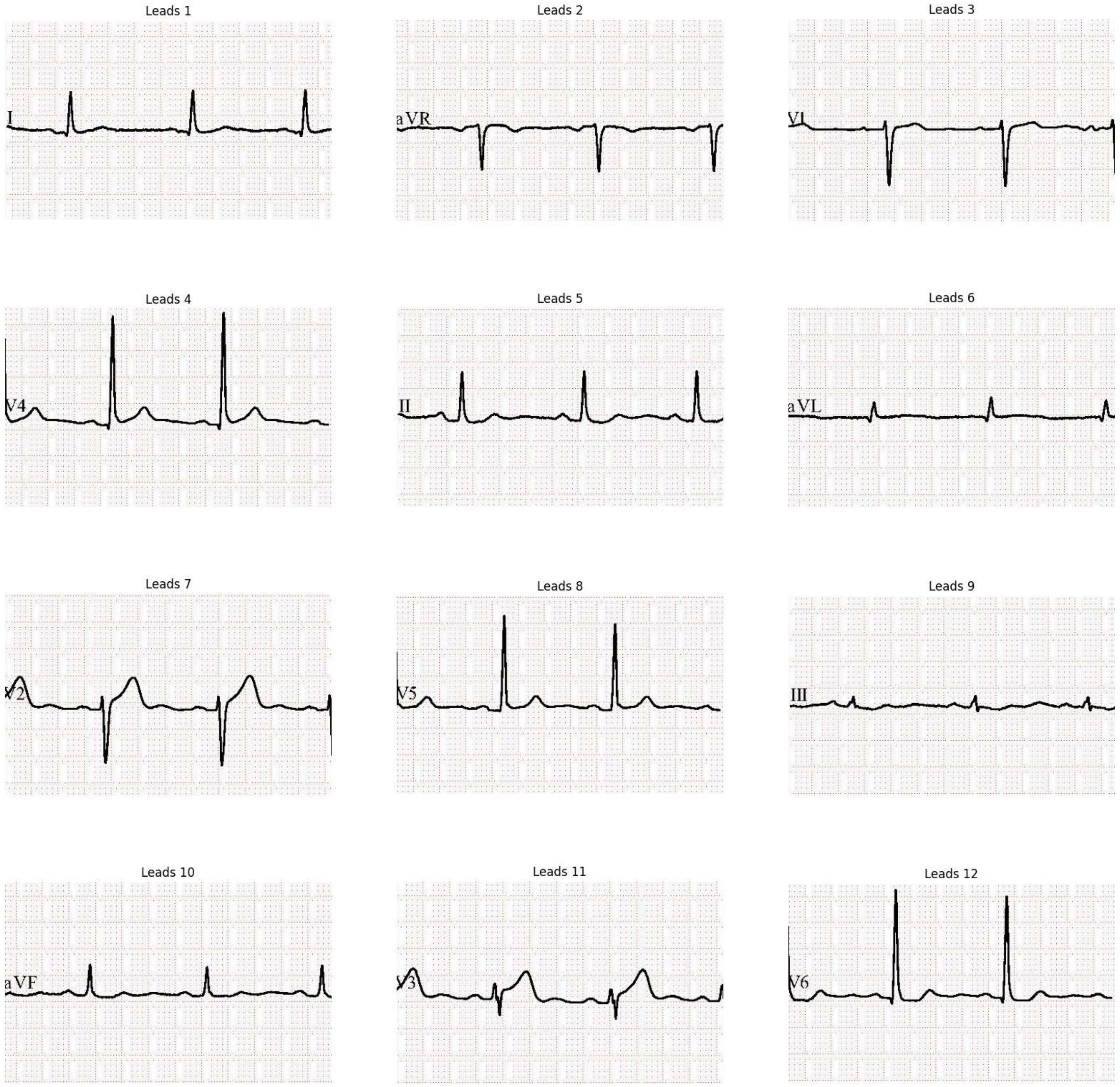
Diagnosis Information:

Technician :
Ref-Phys. :
Report Confirmed by:

0.67~25Hz AC50 25mm/s 10mm/mV 4*2.5s+1r ♥68 SE-3 V1.0 SEMIP V1.7 2020-08-06 02:25:29 AM

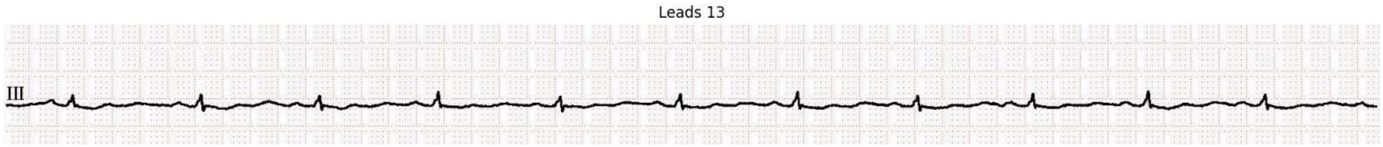
```
Lead_1 = image[300:600, 150:643]
Lead_2 = image[300:600, 646:1135]
Lead_3 = image[300:600, 1140:1625]
Lead_4 = image[300:600, 1630:2125]
Lead_5 = image[600:900, 150:643]
Lead_6 = image[600:900, 646:1135]
Lead_7 = image[600:900, 1140:1625]
Lead_8 = image[600:900, 1630:2125]
Lead_9 = image[900:1200, 150:643]
Lead_10 = image[900:1200, 646:1135]
Lead_11 = image[900:1200, 1140:1625]
Lead_12 = image[900:1200, 1630:2125]
Lead_13 = image[1250:1480, 150:2125]

Leads=[Lead_1,Lead_2,Lead_3,Lead_4,Lead_5,Lead_6,Lead_7,Lead_8,Lead_9,Lead_10,Lead_11,Lead_12,Lead_13]


from skimage.segmentation import slic
from skimage.color import label2rgb

#plotting lead 1-12
fig , ax = plt.subplots(4,3)
```

```
fig.set_size_inches(20, 20)

x_counter=0
y_counter=0


for x,y in enumerate(Leads[:len(Leads)-1]):
  if (x+1)%3==0:
    ax[x_counter][y_counter].imshow(y)
    ax[x_counter][y_counter].axis('off')
    ax[x_counter][y_counter].set_title("Leads {}".format(x+1))
    x_counter+=1
    y_counter=0
  else:
    ax[x_counter][y_counter].imshow(y)
    ax[x_counter][y_counter].axis('off')
    ax[x_counter][y_counter].set_title("Leads {}".format(x+1))
    y_counter+=1

#plot the image
plt.show()
```

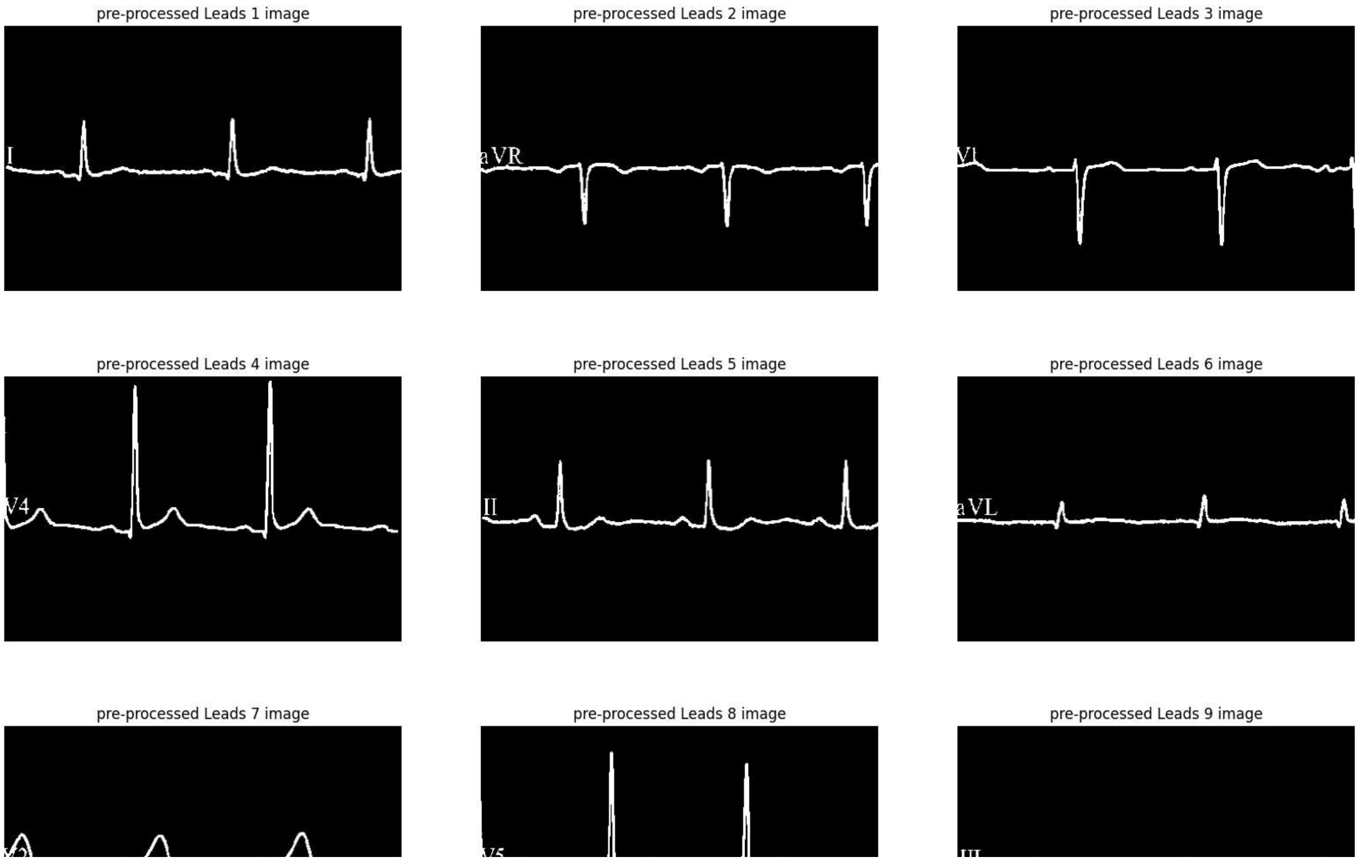


```
fig1 , ax1 = plt.subplots()
fig1.set_size_inches(20, 20)
```

```
ax1.imshow(Lead_13)
ax1.set_title("Leads 13")
ax1.axis('off')
plt.show()
```

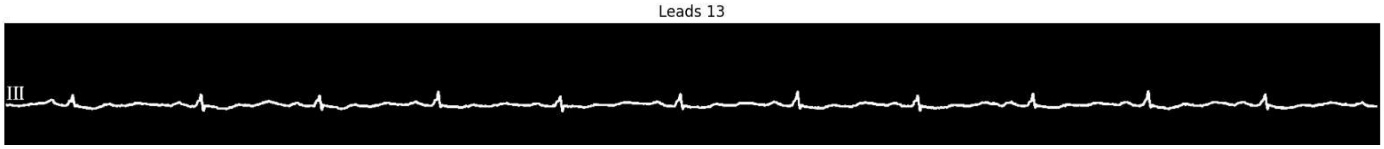


```
#importing gaussian filter and otsu threshold
from skimage.filters import threshold_otsu,gaussian
from skimage.transform import resize
from numpy import asarray

#creating subplot of size(4,3) 4 rows and 3 columns
fig2 , ax2 = plt.subplots(4,3)

fig2.set_size_inches(20, 20)

#setting counter for plotting based on value
x_counter=0
y_counter=0

#looping through image list containg all leads from 1-12
for x,y in enumerate(Leads[:len(Leads)-1]):
  #converting to gray scale
  grayscale = color.rgb2gray(y)
  #smoothing image
  blurred_image = gaussian(grayscale, sigma=0.7)
  #thresholding to distinguish foreground and background
  #using otsu thresholding for getting threshold value
  global_thresh = threshold_otsu(blurred_image)

  #creating binary image based on threshold
  binary_global = blurred_image < global_thresh
  #resize image
  binary_global = resize(binary_global, (300, 450))

  if (x+1)%3==0:
    ax2[x_counter][y_counter].imshow(binary_global,cmap="gray")
    ax2[x_counter][y_counter].axis('off')
    ax2[x_counter][y_counter].set_title("pre-processed Leads {} image".format(x+1))
    x_counter+=1
    y_counter=0
  else:
    ax2[x_counter][y_counter].imshow(binary_global,cmap="gray")
    ax2[x_counter][y_counter].axis('off')
    ax2[x_counter][y_counter].set_title("pre-processed Leads {} image".format(x+1))
    y_counter+=1

#plot the image
plt.show()
```


pre-processed Leads 1 image
I
pre-processed Leads 2 image
aVR
pre-processed Leads 3 image
V1
pre-processed Leads 4 image
V4
pre-processed Leads 5 image
II
pre-processed Leads 6 image
aVL
pre-processed Leads 7 image
pre-processed Leads 8 image
pre-processed Leads 9 image

```
#plotting lead 13
fig3 , ax3 = plt.subplots()
fig3.set_size_inches(20, 20)

#converting to gray scale
grayscale = color.rgb2gray(Lead_13)
#smoothing image
blurred_image = gaussian(grayscale, sigma=0.7)
#thresholding to distinguish foreground and background
#using otsu thresholding for getting threshold value
global_thresh = threshold_otsu(blurred_image)
print(global_thresh)

#creating binary image based on threshold
binary_global = blurred_image < global_thresh
ax3.imshow(binary_global,cmap='gray')
ax3.set_title("Leads 13")
ax3.axis('off')
```

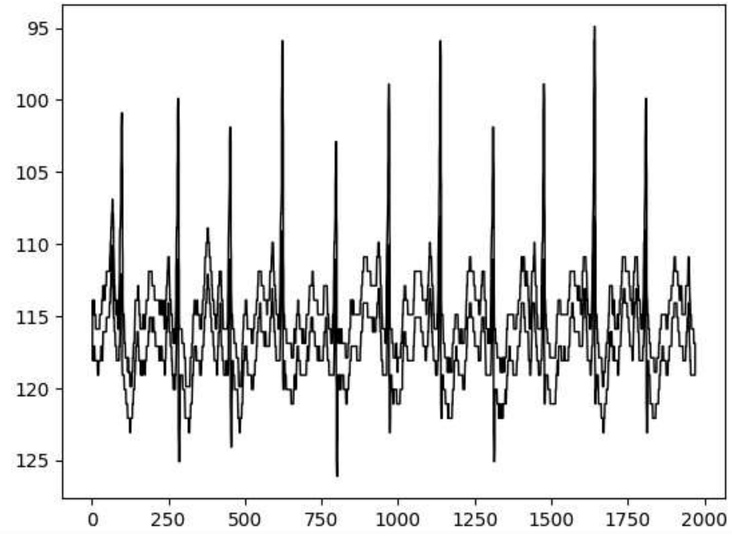
```
0.5551925786352789
(-0.5, 1974.5, 229.5, -0.5)
```



```
#import measure
from skimage import measure
import scipy.ndimage as ndimage

#finding contour
contours = measure.find_contours(binary_global,0.9)

# Shows the image with contours found
fig4, ax4 = plt.subplots()

plt.gca().invert_yaxis()

contours_shape = sorted([x.shape for x in contours])[::-1][0:1]
print(contours_shape)
```

```
for contour in contours:
  if contour.shape in contours_shape:
    test = resize(contour, (255, 2))
    ax4.plot(contour[:, 1], contour[:, 0],linewidth=1,color='black')
ax1.axis('image')
ax1.set_title("Sample pre-processed Leads 13 image")
```

```
[(5365, 2)]
Text(0.5, 1.0, 'Sample pre-processed Leads 13 image')
```



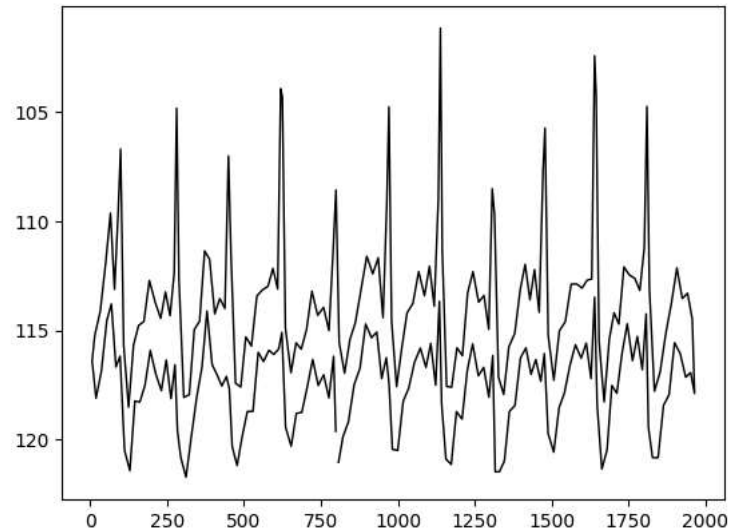
```
contours_shape = sorted([x.shape for x in contours])[::-1][0:3]
contours_shape
```

```
[(5365, 2), (61, 2), (61, 2)]
```

```
test.shape
```

```
(255, 2)
```

```
import pandas as pd

#convert contour to dataframe
df = pd.DataFrame(test, columns = ['X','Y'])
fig5, ax5 = plt.subplots()

plt.gca().invert_yaxis()

#plot the image
ax5.plot(df['Y'],df['X'],linewidth=1,color='black',linestyle='solid')

#save the image
fig5.savefig('Lead13_Signal.png')
```



```
df.to_csv('data.csv',index=False)

#View CSV data for verification
test_df=pd.read_csv('data.csv')
test_df
```

	X	Y
0	119.623116	798.134468
1	116.191637	791.215063
2	118.095667	775.828611
3	117.037716	758.959849
4	117.523500	741.169638
...	...	...
250	116.736264	876.890061
251	117.480001	858.051301
252	119.191059	839.351112
253	119.863793	821.770199
254	121.042986	807.123784

255 rows × 2 columns

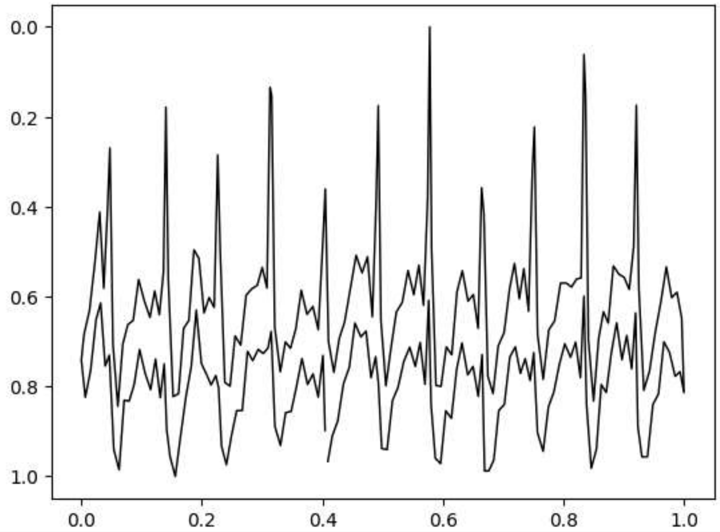
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

fit_transform_data = scaler.fit_transform(df)
Normalized_Scaled=pd.DataFrame(fit_transform_data, columns = ['X','Y'])
Normalized_Scaled
```

	X	Y
0	0.897685	0.404424
1	0.730761	0.400889
2	0.823382	0.393028
3	0.771919	0.384410
4	0.795549	0.375321
...	...	...
250	0.757254	0.444661
251	0.793433	0.435036
252	0.876667	0.425482
253	0.909392	0.416500
254	0.966754	0.409017

255 rows × 2 columns

```
import pandas as pd

df = pd.DataFrame(Normalized_Scaled, columns = ['X','Y'])

fig6, ax6 = plt.subplots()

plt.gca().invert_yaxis()

ax6.plot(Normalized_Scaled['Y'],Normalized_Scaled['X'],linewidth=1,color='black',linestyle='solid')
```

```
[<matplotlib.lines.Line2D at 0x7c7891448290>]
```



```
Normalized_Scaled.to_csv('scaled_data.csv',index=False)
#reading CSV to test
test_scaled_df=pd.read_csv('scaled_data.csv')
test_scaled_df
```

	X	Y
0	0.897685	0.404424
1	0.730761	0.400889
2	0.823382	0.393028
3	0.771919	0.384410
4	0.795549	0.375321
...	...	...
250	0.757254	0.444661
251	0.793433	0.435036
252	0.876667	0.425482
253	0.909392	0.416500
254	0.966754	0.409017

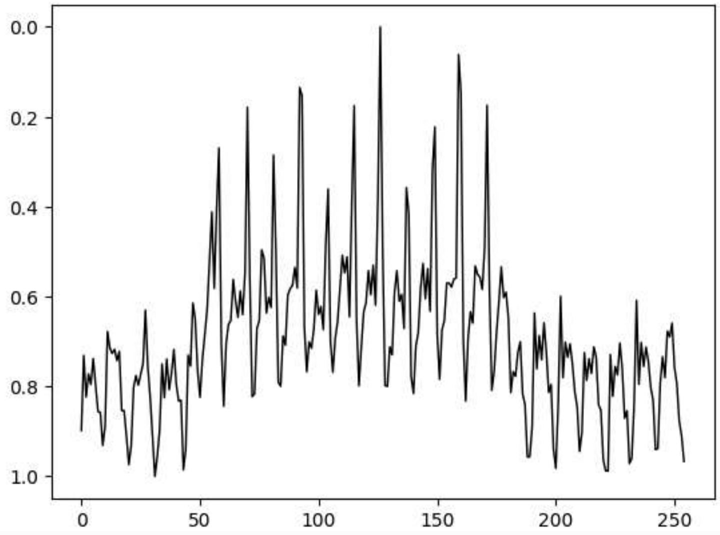
255 rows × 2 columns

```
# For now save the X axis as a seperate csv file (1D) as it seems to corresponds the high and low points and y axis corresponds to curve/sha
#scaled_data to CSV
Normalized_Scaled['X'].to_csv('scaled_data_X.csv',index=False)
#reading CSV to test
test_scaled_df_X=pd.read_csv('scaled_data_X.csv')
test_scaled_df_X.shape
```

```
(255, 1)
```

```
import pandas as pd

test_plot_df = pd.DataFrame(test_scaled_df_X, columns = ['X'])
fig6, ax6 = plt.subplots()

plt.gca().invert_yaxis()

ax6.plot(test_plot_df,linewidth=1,color='black',linestyle='solid')
```

```
[<matplotlib.lines.Line2D at 0x7c788fc56050>]
```



```
test_transpose = test_scaled_df_X.T
test_transpose
```

	0	1	2	3	4	5	6	7	8	9	...	245	246	247	2
X	0.897685	0.730761	0.823382	0.771919	0.795549	0.73776	0.797518	0.855601	0.857979	0.930991	...	0.733781	0.780223	0.676803	0.6895

1 rows × 255 columns

```
def Convert_Image_Lead(image_file,parent_folder):
    #read the image
    image=imread('{parent}/{image_file}'.format(parent=str(parent_folder),image_file=str(image_file)),plugin='matplotlib')
    #dividing the ECG leads from 1-13 from the above image
    Lead_1 = image[300:600, 150:643]
    Lead_2 = image[300:600, 646:1135]
    Lead_3 = image[300:600, 1140:1626]
    Lead_4 = image[300:600, 1630:2125]
    Lead_5 = image[600:900, 150:643]
    Lead_6 = image[600:900, 646:1135]
    Lead_7 = image[600:900, 1140:1626]
    Lead_8 = image[600:900, 1630:2125]
    Lead_9 = image[900:1200, 150:643]
    Lead_10 = image[900:1200, 646:1135]
    Lead_11 = image[900:1200, 1140:1626]
    Lead_12 = image[900:1200, 1630:2125]
    Lead_13 = image[1250:1480, 150:2125]

    #list of leads
    Leads=[Lead_1,Lead_2,Lead_3,Lead_4,Lead_5,Lead_6,Lead_7,Lead_8,Lead_9,Lead_10,Lead_11,Lead_12,Lead_13]

    #folder_name to store lead_images
    folder_name= re.sub('.jpg', '',image_file)

    #loop through leads and create seperate images
    for x,y in enumerate(Leads):
      fig , ax = plt.subplots()
      #fig.set_size_inches(20, 20)
      ax.imshow(y)
      ax.axis('off')
      ax.set_title("Leads {0}".format(x+1))
      if (os.path.exists(parent_folder+'/'+folder_name)):
        pass
      else:
        os.makedirs(parent_folder+'/'+folder_name)

      #save the image
      plt.close('all')
      plt.ioff()
      fig.savefig('{parent}/{folder_name}/Lead_{x}_Signal.png'.format(folder_name=folder_name,x=x+1,parent=parent_folder))

    extract_signal_leads(Leads,folder_name,parent_folder)



def extract_signal_leads(Leads,folder_name,parent):
  #looping through image list containg all leads from 1-13
  for x,y in enumerate(Leads):
    #creating subplot
    fig1 , ax1 = plt.subplots()

    #set fig size
    #fig1.set_size_inches(20, 20)

    #converting to gray scale
    grayscale = color.rgb2gray(y)
    #smoothing image
    blurred_image = gaussian(grayscale,sigma=0.7)
    #thresholding to distinguish foreground and background
    #using otsu thresholding for getting threshold value
    global_thresh = threshold_otsu(blurred_image)

    #creating binary image based on threshold
    binary_global = blurred_image < global_thresh

    #resize image
    if x!=12:
      binary_global = resize(binary_global, (300, 450))

    ax1.imshow(binary_global,cmap="gray")
    ax1.axis('off')
    ax1.set_title("pre-processed Leads {} image".format(x+1))
    plt.close('all')
    plt.ioff()
    #save the image
    fig1.savefig('{parent}/{folder_name}/Lead_{x}_preprocessed_Signal.png'.format(folder_name=folder_name,x=x+1,parent=parent))
    fig7 , ax7 = plt.subplots()
    plt.gca().invert_yaxis()

    #find contour and get only the necessary signal contour
    contours = measure.find_contours(binary_global,0.8)
    contours_shape = sorted([x.shape for x in contours])[::-1][0:1]
    for contour in contours:
      if contour.shape in contours_shape:
        test = resize(contour, (255, 2))
        ax7.plot(test[:, 1], test[:, 0],linewidth=1,color='black')
    ax7.axis('image')
    ax7.set_title("Contour {} image".format(x+1))
    plt.close('all')
    plt.ioff()
    #save the image
    fig7.savefig('{parent}/{folder_name}/Lead_{x}_Contour_Signal.png'.format(folder_name=folder_name,x=x+1,parent=parent))
    lead_no=x
    #convert_csv(test,lead_no,folder_name,parent)
    #scale_csv(test,lead_no,folder_name,parent)
    scale_csv_1D(test,lead_no,folder_name,parent)


def convert_csv(test,lead_no,folder_name,parent):
#convert contour to dataframe
  target=folder_name[0:2]
  df = pd.DataFrame(test, columns = ['X','Y'])
  df['Target']=target
  #x_axis= 'Lead_{lead_no}_X'.format(lead_no=lead_no)
  #y_axis= 'Lead_{lead_no}_Y'.format(lead_no=lead_no)
  fig5, ax5 = plt.subplots()
  #convert to CSV
  df.to_csv('{parent}/{folder_name}/{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent,folder_name=folder_name),index=False)

def scale_csv(test,lead_no,folder_name,parent):
  #scaling the data and testing
  target=folder_name[0:2]
  scaler = MinMaxScaler()
  fit_transform_data = scaler.fit_transform(test)
  Normalized_Scaled=pd.DataFrame(fit_transform_data, columns = ['X','Y'])
  Normalized_Scaled=Normalized_Scaled.T
  Normalized_Scaled['Target']=target
  #scaled_data to CSV
  if (os.path.isfile('{parent}/Scaled_{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent))):
    Normalized_Scaled.to_csv('{parent}/Scaled_{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent), mode='a', header=False,index=False)
  else:
    Normalized_Scaled.to_csv('{parent}/Scaled_{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent,folder_name=folder_name),index=False)

def scale_csv_1D(test,lead_no,folder_name,parent):
  target=folder_name[0:2]
  #scaling the data and testing
  scaler = MinMaxScaler()
  fit_transform_data = scaler.fit_transform(test)
  Normalized_Scaled=pd.DataFrame(fit_transform_data[:,0], columns = ['X'])
  fig6, ax6 = plt.subplots()
  plt.gca().invert_yaxis()
  ax6.plot(Normalized_Scaled,linewidth=1,color='black',linestyle='solid')
  plt.close('all')
  plt.ioff()
  fig6.savefig('{parent}/{folder_name}/ID_Lead_{lead_no}_Signal.png'.format(folder_name=folder_name,lead_no=lead_no+1,parent=parent))
  Normalized_Scaled=Normalized_Scaled.T
  Normalized_Scaled['Target']=target
  #scaled_data to CSV
  if (os.path.isfile('{parent}/scaled_data_1D_{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent))):
    Normalized_Scaled.to_csv('{parent}/scaled_data_1D_{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent), mode='a', header=False,index=F
  else:
    Normalized_Scaled.to_csv('{parent}/scaled_data_1D_{lead_no}.csv'.format(lead_no=lead_no+1,parent=parent,folder_name=folder_name),index=F
```

```
import os
import re
from skimage.filters import threshold_otsu,gaussian
from skimage import measure
import pandas as pd
import numpy as nm
from sklearn.preprocessing import MinMaxScaler
from skimage.io import imread
from skimage import color
from skimage.transform import resize
from numpy import asarray
import matplotlib.pyplot as plt

"""#### **NOW WE HAVE BOTH CSV FILES AND CROPPED LEAD IMAGES(1-13) TO WORK ON. WE CAN PERFROM CNN on 1D images & 2D images and perform diffe

### **NOW PERFORM DATA PREPROCESSING/FEATURE EXTRACTION ON  ALL THE FILES IN THE ECG_IMAGES FOLDER**

#### **FUNCTION TO EXTRACT IMAGE LEADS(1-13) (FEATURE EXTRACTION)**
"""

"""####**FUNCTIONS FOR CSV CONVERSION AND SCALING**"""

#extract_only signal from images
def extract_signal_leads(Leads,folder_name,parent):
  #looping through image list containg all leads from 1-13
  for x,y in enumerate(Leads):
    #creating subplot
    fig1 , ax1 = plt.subplots()

    #set fig size
    #fig1.set_size_inches(20, 20)

    #converting to gray scale
    grayscale = color.rgb2gray(y)
    #smoothing image
    blurred_image = gaussian(grayscale,sigma=0.7)
    #thresholding to distinguish foreground and background
    #using otsu thresholding for getting threshold value
    global_thresh = threshold_otsu(blurred_image)

    #creating binary image based on threshold
    binary_global = blurred_image < global_thresh

    #resize image
    if x!=12:
      binary_global = resize(binary_global, (300, 450))

    ax1.imshow(binary_global,cmap="gray")
    ax1.axis('off')
    ax1.set_title("pre-processed Leads {} image".format(x+1))
    plt.close('all')
    plt.ioff()
```