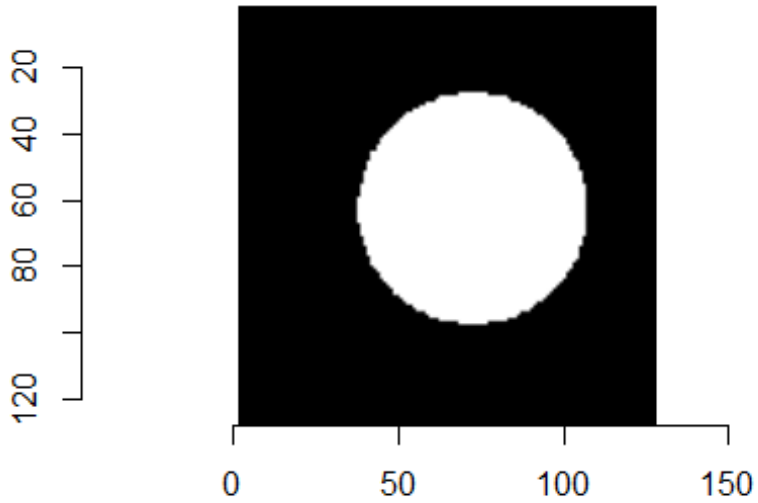
Project4_g29

Hannah Li, Akankshi Mody, Sebastian Osorio

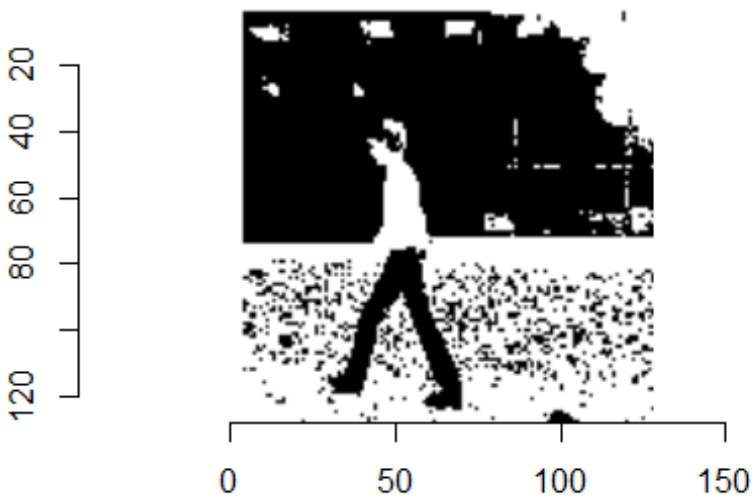
3/23/2020

Experimenting with the Auto threshold option in the imager library

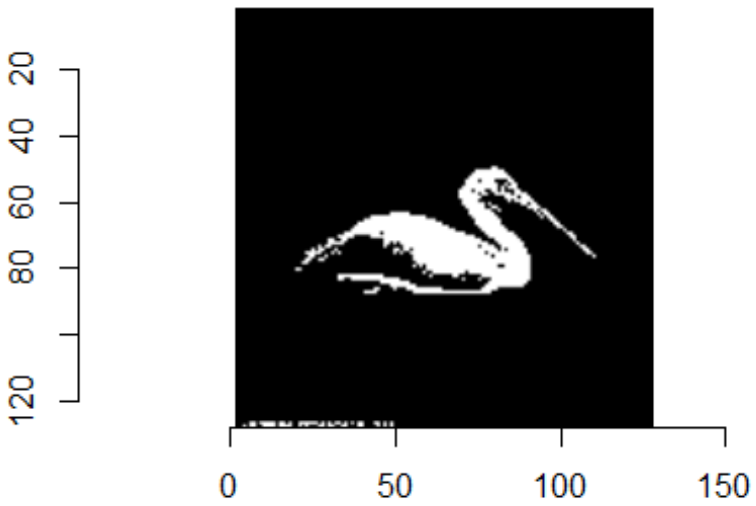
```
seg_thresh_auto = function(image){
  image1 = load.image(image)
  image1 = resize(image1,size_x = 128,size_y = 128)
  #plot(image1)
  #hist(image1)
  transformed_image1 = threshold(image1, thr = "auto", approx = TRUE, adjust
= 1)
  return(plot(transformed_image1))
}

seg_thresh_auto("Pic1.jpg")
```



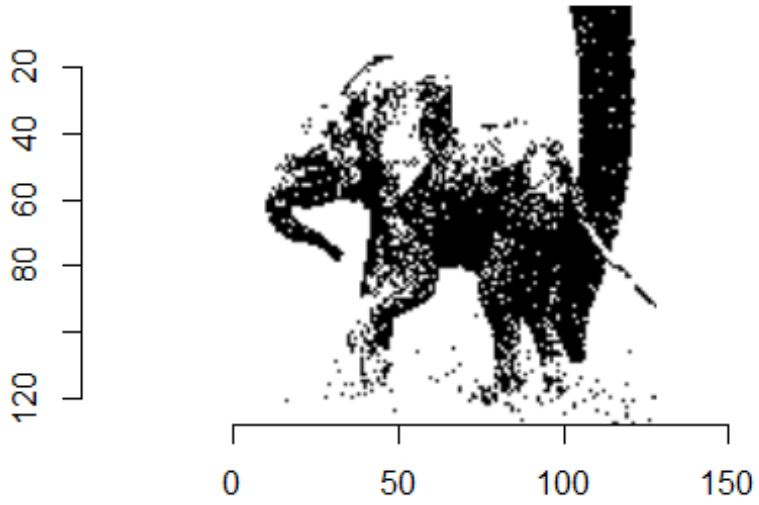

```
seg_thresh_auto("Pic2.jpg")
```




```
seg_thresh_auto("Pic3.jpg")
```

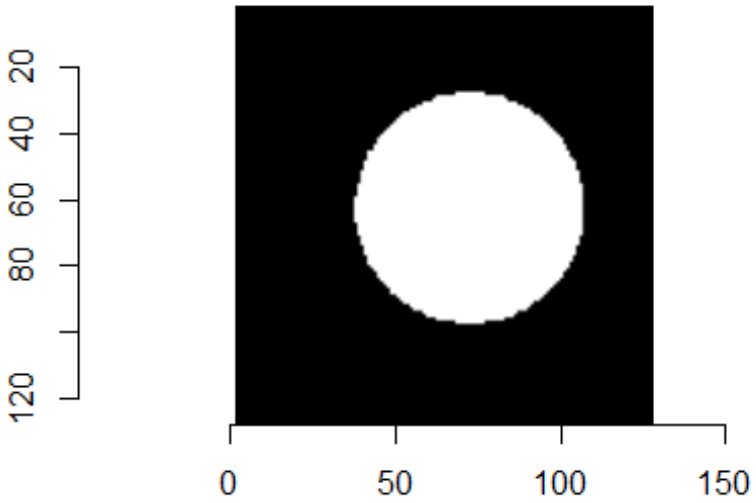



```
seg_thresh_auto("Pic4.jpg")
```

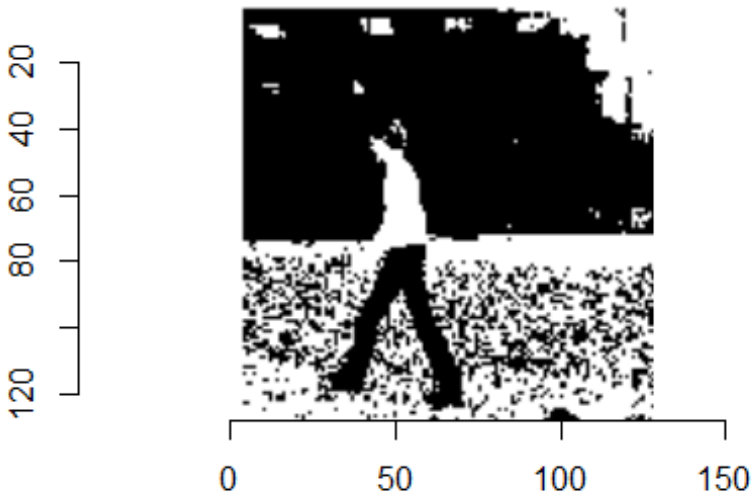


Using custom threshold: (Range/2) to get a center point in the histogram

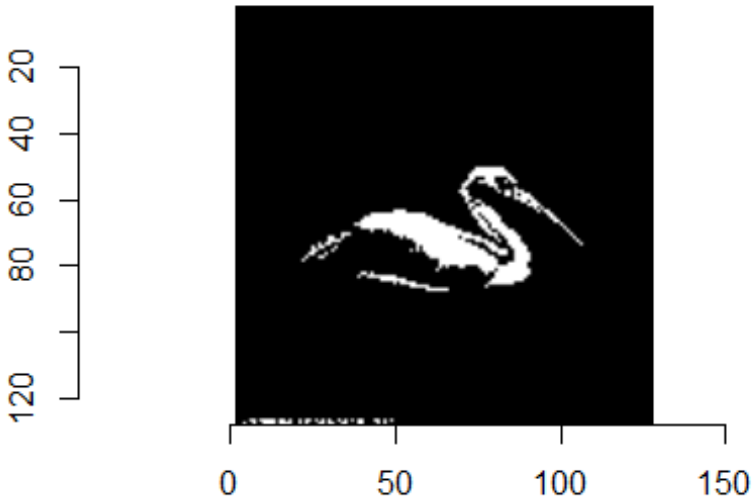
```
seg_thresh_custom = function(image){
  image1 = load.image(image)
  image1 = resize(image1,size_x = 128,size_y = 128)
  threshold_val = ((max(image1) - min(image1))/2) + min(image1)
  transformed_image1 = threshold(image1, thr = threshold_val, approx = TRUE,
adjust = 1)
  return(plot(transformed_image1))
}

seg_thresh_custom("Pic1.jpg")
```




```
seg_thresh_custom("Pic2.jpg")
```




```
seg_thresh_custom("Pic3.jpg")
```




```
seg_thresh_custom("Pic4.jpg")
```

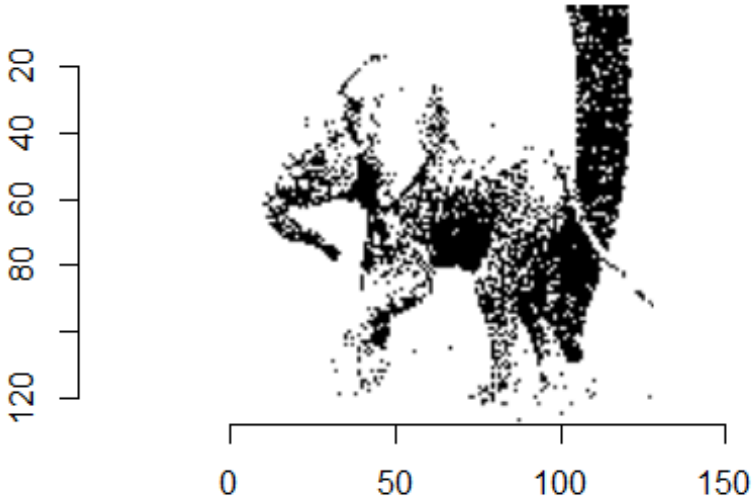


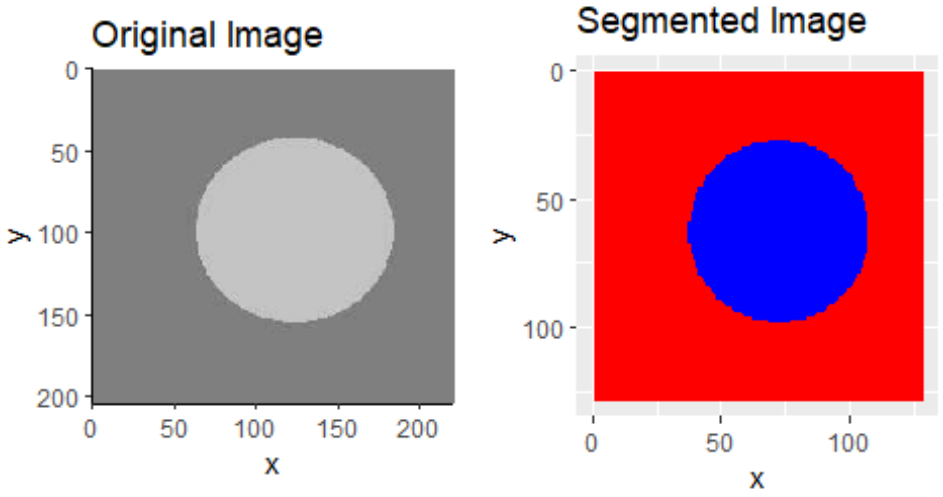
Image Segmentation with Min-cut/Max-flow

```
#library(cowplot)
imgSegMaxflow <- function(inputImage) {
  img = load.image(inputImage)

  img = resize(img,size_x = 128,size_y = 128)
  nrows = nrow(img)
  ncols = ncol(img)
  img.df = as.data.frame(img)
  df.nrows = nrow(img.df)

  x_bound = c(round(nrows/4), 3*round(nrows/4))
  y_bound = c(round(ncols/4), 3*round(ncols/4))

  #Creating temporary background and foreground
  img.df = mutate(img.df, isinFG = as.integer((x>x_bound[1]&x<x_bound[2]) & (
y>y_bound[1]&y<y_bound[2])))
  #unique(img.df['isinFG'])

  #Let pf be the average intensity of the pixels in the temporary foreground
and pb be the average intensity of the pixels in the temporary background
  pf = mean(filter(img.df,img.df$isinFG == 1)$value)
  pb = mean(filter(img.df,img.df$isinFG == 0)$value)

  K = 0.01
  sigma = 1
  img.df$a = -log(abs(img.df$value-pf)/(abs(img.df$value-pf)+abs(img.df$value
-pb)))
  img.df$b = -log(abs(img.df$value-pb)/(abs(img.df$value-pf)+abs(img.df$value
-pb)))

  from_all = c() #c=[], c.append(1), c(c,1) = [1]
  to_all = c()
  weight_all = c()

  for (index in 1:df.nrows){
    from = c()
    to = c()
    weight = c()
    if (index-1 > 0){
      adj_index = index-1
      from = c(from, index)
      to = c(to, adj_index)
      weight = c(weight, K*exp(-(img.df$value[index] - img.df$value[adj_index
])^2/sigma^2))
    }
    if (index+1 <= df.nrows){
      adj_index = index+1
      from = c(from, index)
      to = c(to, adj_index)
```

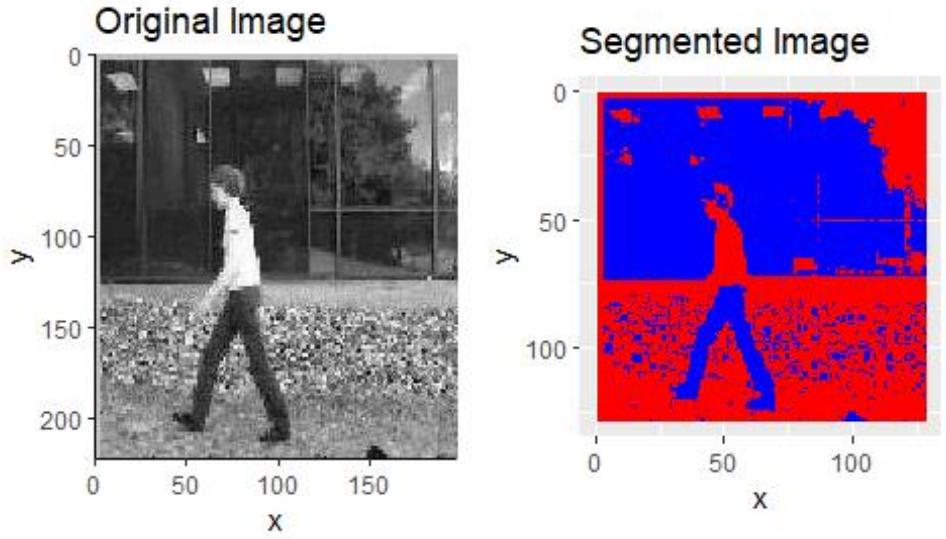
```
      weight = c(weight, K*exp(-(img.df$value[index] - img.df$value[adj_index
])^2/sigma^2))
    }
    if (index-nrows > 0){
      adj_index = index-nrows
      from = c(from, index)
      to = c(to, adj_index)
      weight = c(weight, K*exp(-(img.df$value[index] - img.df$value[adj_index
])^2/sigma^2))
    }
    if (index+nrows <= df.nrows){
      adj_index = index+nrows
      from = c(from, index)
      to = c(to, adj_index)
      weight = c(weight, K*exp(-(img.df$value[index] - img.df$value[adj_index
])^2/sigma^2))
    }
    from_all = c(from_all, from)
    to_all = c(to_all, to)
    weight_all = c(weight_all, weight)
  }

  #Add source
  #from_all = c(from_all, rep("s",df.nrows))
  #to_all = c(to_all, c(1:df.nrows))
  #weight_all = c(weight_all, img.df$a)

  #Add sink
  from_all = c(from_all, rep("s",df.nrows), c(1:df.nrows))
  to_all = c(to_all, c(1:df.nrows), rep("t",df.nrows))
  weight_all = c(weight_all, img.df$a, img.df$b)
  graph.df = data.frame(from_all,to_all,weight_all)
  network.graph = graph.data.frame(graph.df)

  solution = max_flow(network.graph, source = 's', target = 't', capacity = w
eight_all)
  pic.fore = as.vector(solution$partition1[solution$partition1 < df.nrows + 1
])
  pic.back = as.vector(solution$partition2[solution$partition2 < df.nrows + 1
])

  ##foreground with blue
  ##background with red
  img.df$rgb.val = 0
  img.df[pic.fore,]$rgb.val = '#0000FF'
  img.df[pic.back,]$rgb.val = '#FF0000'

  #Plotted solution and then reverse it to get the segmentation output of the
original image
```

```
  p <- ggplot(img.df,aes(x,y))+geom_raster(aes(fill=rgb.val))+scale_fill_iden
tity()+theme(aspect.ratio=1)+ggtitle("Segmented Image")
  p = p+scale_y_reverse()

  library(ggpubr)
  library(magick)
  origim = image_read(inputImage)
  origim = image_ggplot(origim)+theme_classic()+ggtitle("Original Image")
  return(ggarrange(origim, p))
}

imgSegMaxflow('Pic1.jpg')
```

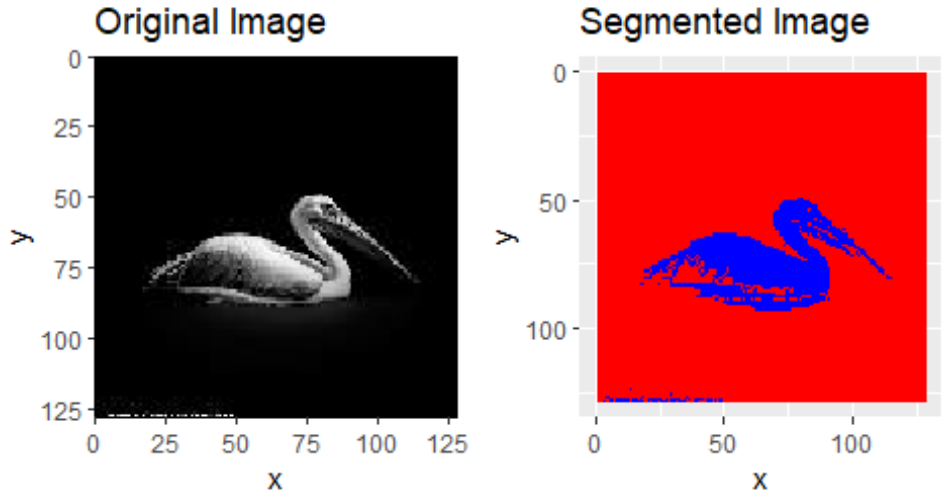
```
## Warning: package 'ggpubr' was built under R version 3.6.3

## Warning: package 'magick' was built under R version 3.6.3

## Linking to ImageMagick 6.9.9.14
## Enabled features: cairo, freetype, fftw, ghostscript, lcms, pango, rsvg, w
ebp
## Disabled features: fontconfig, x11
```



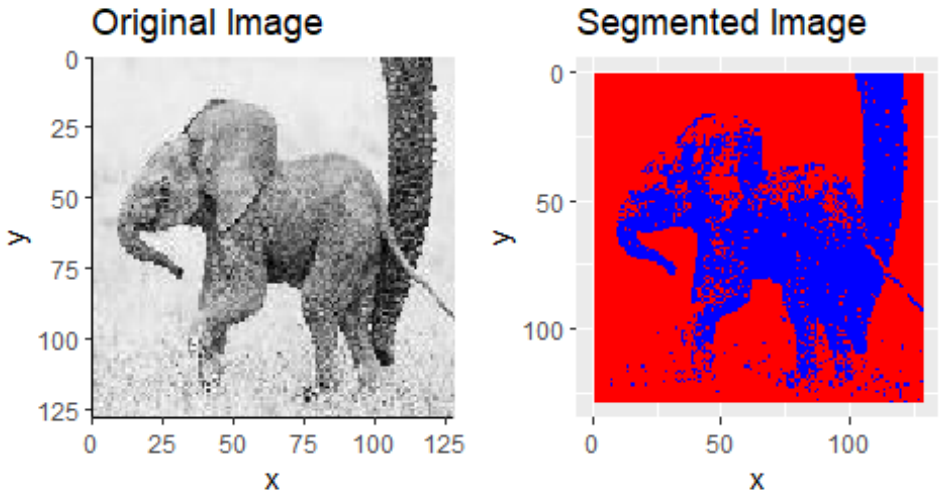

```
imgSegMaxflow('Pic2.jpg')
```




```
imgSegMaxflow('Pic3.jpg')
```




```
imgSegMaxflow('Pic4.jpg')
```


Original Image
Segmented Image
y
x
0
25
50
75
100
125