Tutorial

Ana Tércia, João, Laura Reis, Leonardo e Paulo

19 de novembro de 2019

0. AVISO

Todas as imagens se encontram nos slides.

1. INTRODUÇÃO

1.1. A importância da análise de imagens

A análise e manipulação de dados no R é importante pois de outra maneira seria muito difícil fazer determinados trabalhos ou tirar certas conclusões como por exemplo:

*Análise de imagens do solo feitas por satelites que pilotos e drones usam para saber se podem pousar onde desejam;

*Análise de rostos para saber a idade de pessoas, assim a identificação de menores em situações de exposição na internet é mais rápida e precisa;

*Análise de imagens dos pacientes por ressonância magnética (functional magnetic resonance imaging), pois dessa forma a imagen é nítida e o método é não invasivo.

1.2. Formatos de imagens

Existem dois tipos de imagens, sendo elas vetoriais e bitmap. A primeira se trata de imagens feitas por vetores, que recarregam todo o tempo. Um pdf de livro é um bom exemplo, que quando dado zoom demora alguns segundos e fica nítido de novo. Já o segundo tipo são imagens compostas por pixels (na tradução literal é mapa de bites). Existem 8 formatos de imagens, e todos o R é capaz de processar (com ressalvas para PDF). São eles:

- TIFF
- JPEG
- PNG
- SVG
- GIF
- BMP
- PDF
- EPS

2. Pacotes

Os principais pacotes para a manipulação de imagens são BiocManager, EBImage (para formatos JPEG, PNG e TIFF), Imager (JPEG, PNG e BMP) e Magick.

```
install.packages ('BiocManager')
install.packages("EBImage")
install.packages ('imager')
install.packages ('magick')
```

```
require ('BiocManager')
require("EBImage")
require('imager')
require('magick')
```

3. IMPORTAÇÃO E VISUALIZAÇÃO DE IMAGENS

*EbImage:

```
ima <- readImage("C:/Users/nick_/Downloads/897207.jpg")
display (ima)
```

*Imager:

```
ima_1 <- load.image('C:/Users/nick_/Downloads/897207.jpg')
plot(ima_1)
```

*Magick:

```
ima_2 <- image_read("C:/Users/nick_/Downloads/897207.jpg")
print(ima_2)
```

4. MANIPULAÇÃO DE IMAGENS

4.1 Alterar dimensionamento e formato

Para mudar a dimensão de uma imagem usamos a função width que, se for usada conjuntamente com height, a primeira altera a largura e a segunda o comprimento. Se for usada sozinha, altera ambas sem mudar as proporções da imagem.

```
library(rsvg)
queremos <- image_read_svg(
  'https://s3.amazonaws.com/wd-static/static_v1/pt/logo.svg')
queremos
```

Imagem no slide 12

```
queremos2 <- image_read_svg(
  'https://s3.amazonaws.com/wd-static/static_v1/pt/logo.svg',
  width = 210) # 220 = width,
               # 220x = height
queremos2
```

Imagem no slide 13

Quando queremos mudar as dimensões da imagem o código abaixo pode ser utilizado

```
queremos_redimensionado1 <- image_scale(queremos, "210x42")
image_info(queremos_redimensionado1)

queremos_redimensionado2 <- image_scale(queremos, "210x40")
image_info(queremos_redimensionado2)
```

Para converter a imagem e salvar em formato desejado, basta usar a função image_convert. Usando image_info é possível ver as informações e para salvar, usa-se image_write.

```
queremos_convertido <- image_convert (queremos, 'jpeg')
image_info (queremos_convertido)
image_write (queremos, path = 'queremos.png', format = 'png')
```

4.2 Modificar

```
patrik <- image_read("IMAGENS/patrik.png")
bigdata <- image_read('IMAGENS/bigdata.jpg')
\includegraphics[width=3.4in]{IMAGENS/Rlogo}
```

Imagens Patrick, Bigdata e Logo nos slides 16,17,18

Para modificar a imagem (girar, espelhar, etc) os seguintes códigos podem ser usados:

```
image_flop (bob) #espelhar
image_flip (bob) #inverter
image_rotate (bob, 45) #rotacionar
image_crop (bob, '100x150+50') #cortar
```

Imagens no slide 18

4.3 Aplicar filtros

```
image_blur(bob, 10,5) #embaçar
image_charcoal(bob) #gravura
image_oilpaint(bob) #pintura à óleo
image_negate(bob) #negativo
image_modulate(bob, brightness = 80, saturation = 120, hue = 90) #saturada
image_fill(bob, 'orange', point = '+100+200', fuzz = 20) #preenche os contornos com a cor da imagem
```

Imagens no slide 19

4.5 Sobrepor imagens

Para sobrepor imagens primeiro fazemos uma outra imagem colocando as três juntas. Depois é possível fazer algumas modificações (todas as modificações são feitas na ordem que as imagens foram escitas para compor a sobreposta).

```
img <- c(bigdata, logo, patrik)
img <- image_scale(img, "300x300")
image_info(img)
```

Saída:

format width height colorspace matte filesize density

1 JPEG 300 207 sRGB FALSE 0 96x96

2 PNG 300 232 sRGB TRUE 0 72x72

3 PNG 203 300 sRGB TRUE 0 72x72

```
image_append(image_scale(img, 'x200')) #coloca as imagens ao lado umas das outras
image_append (image_scale(img, '100'), stack = TRUE) #coloca as imagens
#com quebra de linha
image_mosaic(img) #sobrepõe as imagens
image_flatten (img) #sobrepõe as imagens em uma unica
```

```
#com o tamanho da primeira
image_flatten(img, 'Minus') #deixa as cores mais
#escuras sobrepostas às outras
```

Imagem no slide 22

```
bigdatapatrik <- image_scale(image_rotate(
  image_background(patrik, "none"), 300), "x260")
juntos <-image_composite(image_scale(
  bigdata, "x330"), bigdatapatrik, offset = "+150+70")
image_write(juntos, path = "juntos.png", format = "png")
```

Imagem no slide 24

4.5.1 Utilidade em gráficos

É útil sobrepor imagens em gráficos para deixar uma parte do gráfico mais explicita ou autoexplicativa.

```
graph <- image_read("IMAGENS/Rplot1.png")
temp <- image_read("IMAGENS/low_temp.png")
temp_graph <- image_scale(image_rotate
                          (image_background(
  temp, "none"), 340), "x50")
temp_graph

juntos_2 <-image_composite(image_scale(
  graph, "x600"), temp_graph, offset = "+150+440")
image_write(juntos_2, path = "juntos2.pdf", format = 'pdf')
```

Imagem no slide 27

4.6 Anotações em imagens

Para escrever nas imagens, usa-se a função image_annotate da seguinte maneira:

```
patrik_anot <- image_annotate(patrik, "Aqui", size = 21,
                              color = "red",
                              boxcolor = "black",
                              degrees = 10,
                              location = "+120+50")
patrik_anot <- image_scale(patrik_anot, "x350")
image_write(patrik_anot, path = "patrik_anot.png",
            format = "png")
```

Imagem no slide 29

5. MANIPULAÇÃO DE GRÁFICOS

É útil saber manipular gráficos sem ter seu código, pois poupa o trabalho de procurar o local exato no código ou situações similares.

Para retirar pontos de um gráfico, o primeiro passo é ler a imagem do gráfico e alterar a sua saturação.

```
library (tidyverse)
im <- image_read("IMAGENS/grafico_ponto.jpg")
im_proc <- im %>%
    image_channel("saturation")
```

```
image_write(im_proc, path = "IMAGENS/grafico_ponto1.png",
            format = "png")
```

Imagem no slide 32

Após isso, é necessário deixar o fundo branco e depois negativo.

```
im_proc2 <- im_proc %>%
  image_threshold('white', '30%')
image_write(im_proc2, path= 'Imagens/graficop2.png', format = 'png')
```

Imagens no slide 33

```
im_proc3 <- im_proc2 %>%
  image_negate()
image_write (im_proc3, path = 'Imagens/graficop3.png', format = 'png')
```

Imagem no slide 34

Para finalizar:

```
require(tidyverse)
dat <- image_data(im_proc3)[1,,] %>%
  as.data.frame() %>%
  mutate(Row = 1:nrow(.)) %>%
  select(Row, everything()) %>%
  mutate_all(as.character) %>%
  gather(key = Column, value = value, 2:ncol(.)) %>%
  mutate(Column = as.numeric(gsub("V", "", Column)),
    Row = as.numeric(Row),
    value = ifelse(value == "00", NA, 1)) %>%
  filter(!is.na(value))
dat

require(ggplot2)
grafico_final <-ggplot(data = dat,
                       aes(x = Row,
                           y = Column,
                           colour = (Column < 200))) +
  geom_point() +
  scale_y_continuous(trans = "reverse") +
  scale_colour_manual(values = c( "blue4","red4")) +
  theme(legend.position = "off")+
ggsave("grafico_final.pdf", width = 4, height = 4)
```

Imagem no slide 37

6. FAZER UM GIF

Primeiro passo para montar um GIF, importe as imagens que você deseja usar:

```
im_1 <- image_read("C:/Users/nick_/Downloads/im_1.jpg")
im_n <- image_read("C:/Users/nick_/Downloads/im_n.jpg")
```

Segundo passo é juntar as imagens e redimensioná-las:

```
imag <- c(im_1, ..., im_n)
imag <- image_scale(imag, '300x300')
```

Depois basta usar a função de animar imagens.

```
image_animate(imag)
```

Para salvar o GIF:

```
library(gifski)

image_write_gif(img, path = 'grafico.gif')
#é possível adicionar um delay que vai de null a 1/10
image_write_gif(img, path = 'grafico_delay.gif', delay = 1/6)
```

7. GIF e filtro Voronoi

Primeira coisa a ser feita é o download da imagem que será usada e depois a conversão dela para a escala de cinza.

```
file="http://ereaderbackgrounds.com/movies/bw/Imagem.jpg"
download.file(file, destfile = "imagem.jpg", mode = 'wb')

load.image("C:/Users/nick_/Downloads/Imagem.jpg") %>%
grayscale() -> x
```

Após isso é necessário definir os limites dos frames.

```
x %>%
  as.data.frame() %>%
  group_by() %>%
  summarize(xmin=min(x), xmax=max(x),
            ymin=min(y), ymax=max(y)) %>%
  as.vector() -> rw
```

O terceiro passo é filtrar a imagem e converter para preto e branco.

```
x %>%
  threshold("45%") %>%
  as.cimg() %>%
  as.data.frame() -> df
```

Depois deve ser calculado e plotado o diagrama de voroni com uma função. É importante lembrar que essa função vai depender do tamanho da amostra.

```
doPlot = function (n)

{
  # Diagrama de Voronoi
  df %>%
    sample_n(n, weight=(1-value)) %>%
    select(x,y) %>%
    deldir(rw=rw, sort=TRUE) %>%
    .$dirsgs -> data

# Isso é apenas para adicionar alguns alfas nas linhas,
# depende da longitude
  data %>%
    mutate(long=sqrt((x1-x2)^2+(y1-y2)^2),
      alpha=findInterval(
      long,
```

```
      quantile (long,
                robs = seq (0,
                            1,
                            length.out = 20)
                                  )
                       )/21 -> data

data %>%
  ggplot(aes(alpha=(1-alpha))) +
  geom_segment(aes(x = x1, y = y1, xend = x2, yend y2),
                color="black", lwd=1) +
  scale_x_continuous(expand=c(0,0))+
  scale_y_continuous(expand=c(0,0), trans=reverse_trans())+
  theme(legend.position = "none",
        panel.background = element_rect(fill="white"),
        axis.ticks = element_blank(),
        panel.grid = element_blank(),
        axis.title = element_blank(),
        axis.text = element_blank())->plot
  return(plot)}
```

Agora é necessário chamar a função anterior e salvar o resultado do plot em jpeg.

```
i= 500
name=paste0("imagem",i,".jpeg")
jpeg(name, width = 600, height = 800, units = "px",
      quality = 100)
doPlot(i)
dev.off()
```

Assim que as imagens são salvas, é possível criar o gif.

```
library(magick)
frames=c()
images=list.files(pattern="jpeg")
for (i in length(images):1)
{
  x=image_read(images[i])
  x=image_scale(x, "300")
  c(x, frames) -> frames
}
animation=image_animate(frames, fps = 2)
image_write(animation, "Imagem.gif")
print(animation)
```

8. FAZER UM FILTRO

Para fazer um filtro geométrico primitivo e deixar a imagem pixelada, é necessário usar o pacote Imager.

```
library(imager)
foto <- load.image("C:/Users/nick_/Downloads/foto.jpg")

foto2<- foto %>% resize(size_x = 80, size_y = 80,
                        interpolation_type = 1L)
suppressMessages(suppressWarnings(library(imager)))
```

```r
foto2 <- rowMeans(foto2, dims = 2)

foto2 %>%
  apply(1, rev) %>%
  t() %>%
  image(col = grey.colors(256), axes = FALSE)
```

Imagem no slide 54

Já para um filtro colorido:

```r
library(imager)
library(purrr)
setwd(diretorio aqui)
im <- load.iamge('frida') %>% imresize(.5)
qsplit <- function(im)
{
  imsplit(im,"x",2) %>% map(~ imsplit(.,"y",2)) %>%
  flatten
}
qsplit(im) %>% as.imlist %>% plot
```

Imagem no slide 56

Após isso:

```r
qunsplit <- function(l)
{
  list(l[1:2],l[3:4]) %>% map(~ imappend(.,"y")) %>%
    imappend("x")
}
qsplit(im) %>% qunsplit %>% plot
imsd <- function(im)
{
  imsplit(im,"c") %>% map_dbl(sd) %>% max
}

refine <- function(l)
{if (is.cimg(l)) # Nós temos uma folha
  {qs <- qsplit(l) # Separa
    if (any(dim(l)[1:2] <= 4)) # Quadrantes são muito pequenos
    {qs$sds <- rep(0,4) # Impede refinamentos adicionais
    }else
    {qs$sds <- map_dbl(qs,imsd)
      }qs}
  else # Não é uma folha, explora mais adiante
  {indm <- which.max(l$sds)
    l[[indm]] <- refine(l[[indm]]) # Refina
    l$sds[indm] <- max(l[[indm]]$sds)
    l}}

refine <- function(l)
{if (is.cimg(l)) # Nós temos uma folha
  {qs <- qsplit(l) # Separa
    if (any(dim(l)[1:2] <= 4)) # Quadrantes são muito pequenos
    {qs$sds <- rep(0,4) # Impede refinamentos adicionais
```

```
    }else
    {qs$sds <- map_dbl(qs,imsd)
      }qs}
  else # Não é uma folha, explora mais adiante
  {indm <- which.max(l$sds)
    l[[indm]] <- refine(l[[indm]]) # Refina
    l$sds[indm] <- max(l[[indm]]$sds)
    l}}
```

Imagem no slide 61 ##9. KERAS, TENSORFLOW E ANACONDA

9.1 Bibliotecas e pacotes

Keras é um pacote do R feito para ajustar modelos de redes neurais profundas. Foi desenvolvida para facilitar experimentações rápidas. Tensorflow é uma biblioteca de software de código aberto para computação numérica usando grafos computacionais. É utilizado para implementação de inteligência artificial. Anaconda é um gerenciador de pacotes que permite gerir distribuições, ambientes de trabalhos e módulos. É uma distribuição de dados científicos do R e do Python.

9.2 Reconhecimento de imagens

Para fazer o R reconhecer imagens e diferencia-las, os seguintes pacotes são necessários:

```
install.packages("tfestimators")
install_tensorflow()
devtools::install_github("rstudio/keras")
devtools::install_github("rstudio/tensorflow")
devtools::install_github("rstudio/keras")
reticulate::py_discover_config()
reticulate::use_condaenv("r-tensorflow")
reticulate::py_config()
```

O próximo passo são as leituras das imagens:

```
library(EBImage)
library(keras)
library(kerasR)
library(kerasformula)
setwd('C:\\Users\\diretorio R')
pics <- c('p1.jpg', 'p2.jpg', 'p3.jpg', 'p4.jpg', 'p5.jpg',
          'p6.jpg','c1.jpg', 'c2.jpg', 'c3.jpg', 'c4.jpg',
          'c5.jpg','c6.jpg')
mypic <- list()
for (i in 1:12) {mypic[[i]] <- readImage(pics[i])}
```

Após isso, é feita uma análise manual das imagens:

```
print(mypic[[1]])
```

Saída:

Image

colorMode : Color

storage.mode : double

dim : 281 180 3

frames.total : 3

frames.render: 1

imageData(object)[1:5,1:6,1]

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 1 1 1 1 1 1

[2,] 1 1 1 1 1 1

[3,] 1 1 1 1 1 1

[4,] 1 1 1 1 1 1

[5,] 1 1 1 1 1 1

```
display(mypic[[12]])
summary(mypic[[1]])
hist(mypic[[2]])
```

Saída:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.0000 0.9529 1.0000 0.9283 1.0000 1.0000

Depois as imagens são redimensionadas:

```
for (i in 1:12) {mypic[[i]] <- resize(mypic[[i]],28, 28)}
for (i in 1:12) {mypic[[i]] <- array_reshape(mypic[[i]],
                                             c(28,28,3))}
```

Agora uma remodelagem das imagens:

```
trainx <- NULL
for (i in 1:5) {trainx <- rbind(trainx, mypic[[i]])}
for (i in 7:11) {trainx <- rbind(trainx, mypic[[i]])}
str(trainx)
testx <- rbind(mypic[[6]], mypic[[12]])
trainy <- c(0,0,0,0,0,1,1,1,1,1)
testy <- c(0,1)
```

Uma criacão de labels:

```
trainLabels <- to_categorical(trainy)
testLabels <- to_categorical (testy)
```

É feito um modelo:

```
model <- keras_model_sequential()
model %>%
        layer_dense(units = 256, activation = 'relu',
        input_shape = c(2352)) %>%
        layer_dense(units = 128, activation = 'relu') %>%
        layer_dense(units = 2, activation = 'softmax')
summary(model)
```

Compilar as imagens:

```
model %>%
         compile(loss = 'binary_crossentropy',
```

```
                  optimizer = optimizer_rmsprop(),
                  metrics = 'accuracy')
```

Agora é ajustar e treinar o modelo:

```
history <- model %>%
        fit(trainx,
            trainLabels,
            epochs = 500,
            batch_size = 32,
            validation_split = 0.2)
```

Para finalizar é feita a avaliação e previsão

```
model %>% evaluate(testx, testLabels)
pred <- model %>% predict_classes(trainx)
table(Predicted = pred, Actual = trainy)
prob <- model %>% predict_proba(trainx)
cbind(prob, Prected = pred, Actual= trainy)
```

Saída: *Imagem no slide 76*