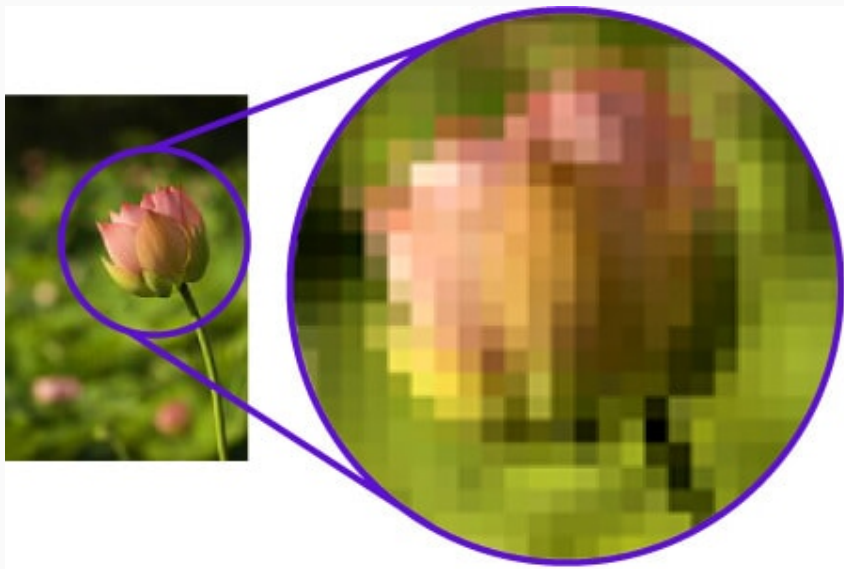


Análise de imagem no Rstudio

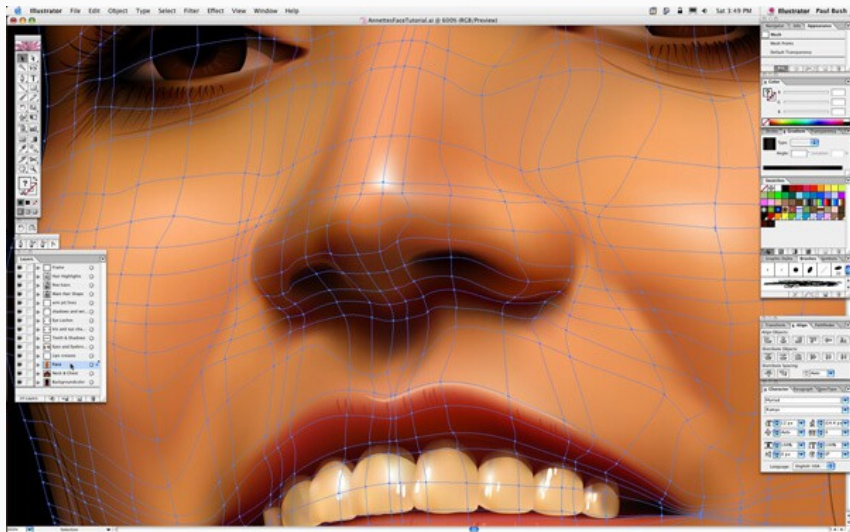
Ana Tércia, João, Laura Reis, Leonardo e Paulo

20 de novembro de 2019

Tipos de imagens (Bitmap)



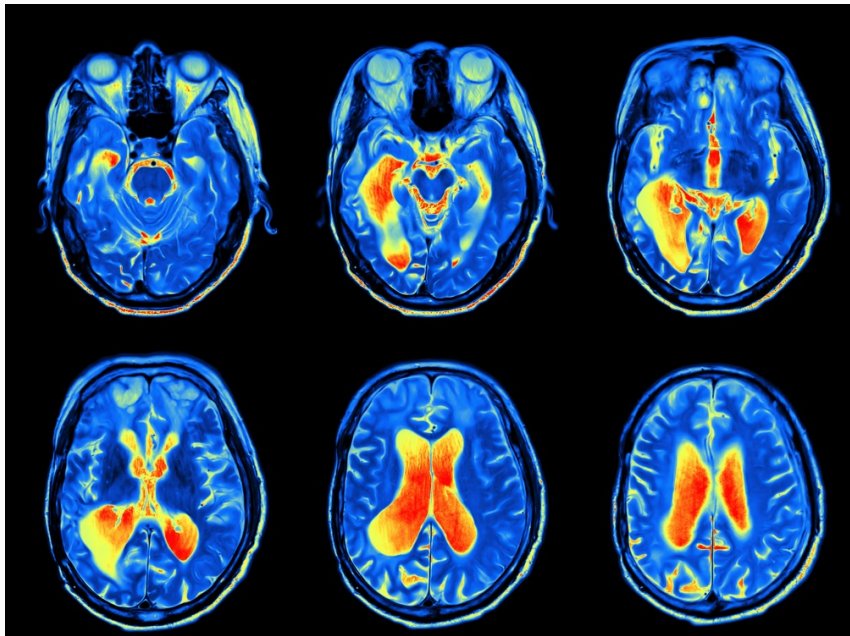
Tipos de imagens (Vetorial)



Formatos de imagens

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

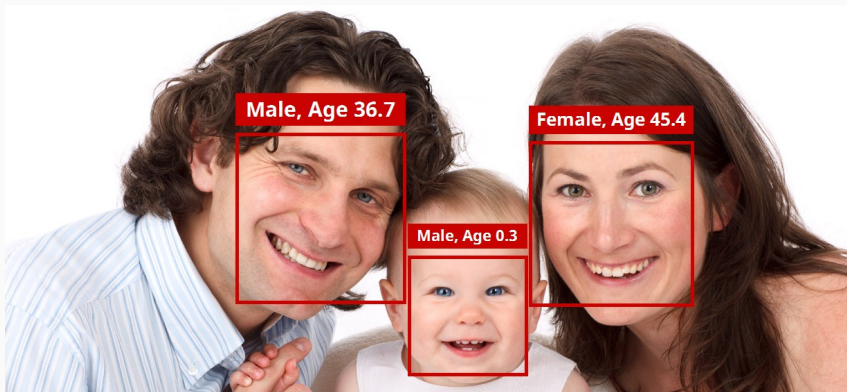
A importância de análise de imagens



A importância de análise de imagens



A importância de análise de imagens



Pacotes

Oos principais pacotes para manipulação de imagem são:

```
require("BiocManager")
```

```
require("EBImage")
```

```
require("imager")
```

```
require("magick")
```


Como os pacotes lêem as imagens?

```
• ima          Formal class Image
  ..@ .Data : num [1:1920, 1:1080, 1:3] 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 ...
  ..@ colormode: int 2
• ima_1        Large cimg (6220800 elements, 47.5 Mb)
  'cimg' num [1:1920, 1:1080, 1, 1:3] 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 0.0157 ...
  ima_2        External pointer of class 'magick-image'
```

Importação e visualização de imagens:

- EImage:

```
.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)
```

Mudar dimensões

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

QUEREMOS!

Mudar dimensões

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

QUEREMOS!

Mudar dimensões

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

```
##    format width height colorspace matte filesize density  
## 1    PNG   210    41      sRGB  TRUE         0   72x72
```

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

```
##    format width height colorspace matte filesize density  
## 1    PNG   207    40      sRGB  TRUE         0   72x72
```

Converter ou salvar em formatos desejados

```
tigre_convertido <- image_convert(queremos, "jpeg")  
image_info(tigre_convertido)
```

```
##    format width height colorspace matte filesize density  
## 1    JPEG   280    54         sRGB  TRUE         0    72x72
```

```
image_write(queremos, path = "queremos.png", format = "png")
```

Imagens para manipulação

```
patrik <- image_read("IMAGENS/patrik.png")
```



Imagens para manipulação

```
bigdata <- image_read('IMAGENS/bigdata.jpg')
```



Imagens para manipulação

```
logo <- image_read('IMAGENS/Rlogo.png')
```



Girar e modificar

`image_flop(patrik)`



`image_flip(patrik)`



`image_rotate(patrik, 45)`



`image_crop(patrik, "123x170+80")`

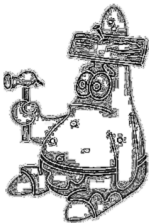


Alguns tipos de filtros

```
image_blur(patrik, 10, 5)
```



```
image_charcoal(patrik)
```



```
image_oilpaint(patrik)
```



```
image_negate(patrik)
```



```
image_modulate(patrik, brightness = 80, saturation = 120, hue = 90)
```



```
image_fill(patrik, "black", point = "+30+105", fuzz = 20)
```



Imagens sobrepostas

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

##	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

Imagens sobrepostas

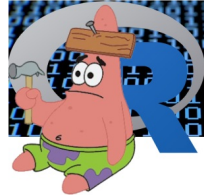
```
image_append(image_scale(img,"100"), stack = TRUE)
```



```
image_append(image_scale(img,"x200"))
```



```
image_mosaic(img)
```



```
image_flatten(img)
```



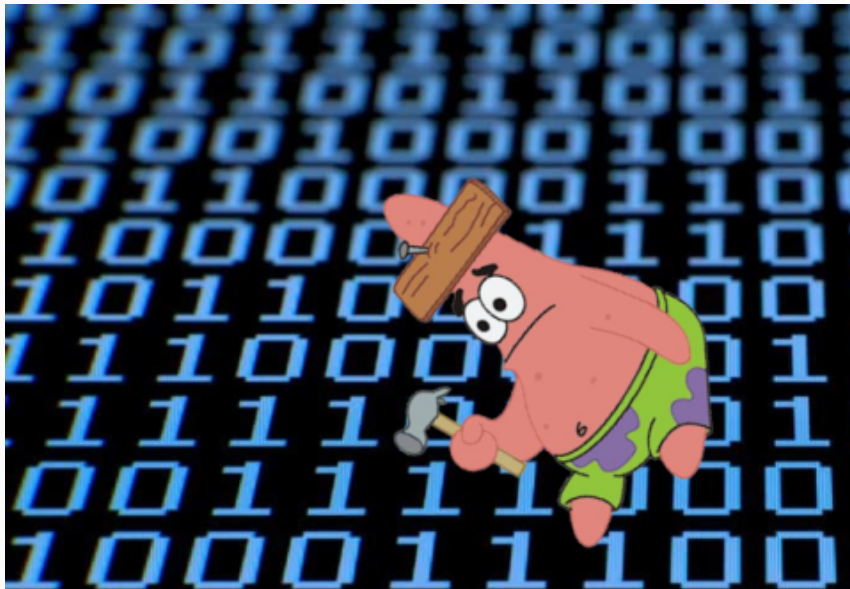
```
image_flatten(img, "Minus")
```



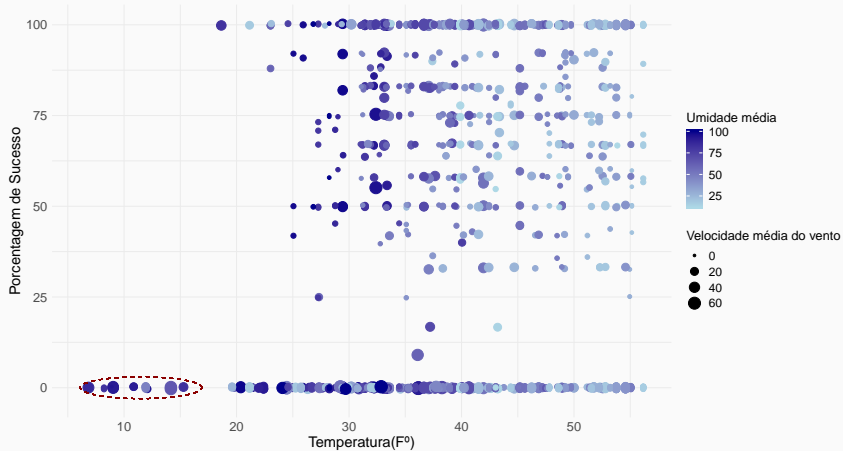
Imagens sobrepostas

```
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")
```

Imagens sobrepostas



Utilidade em gráficos



Utilidade para sobreposição de imagens

```
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",
```

Utilidade para sobreposição de imagens



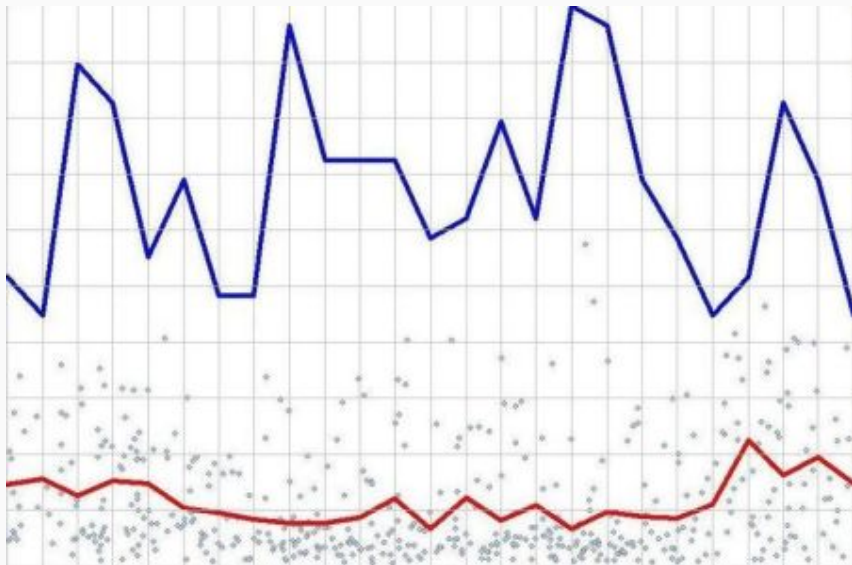
Anotações em imagens

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

Anotações em imagens



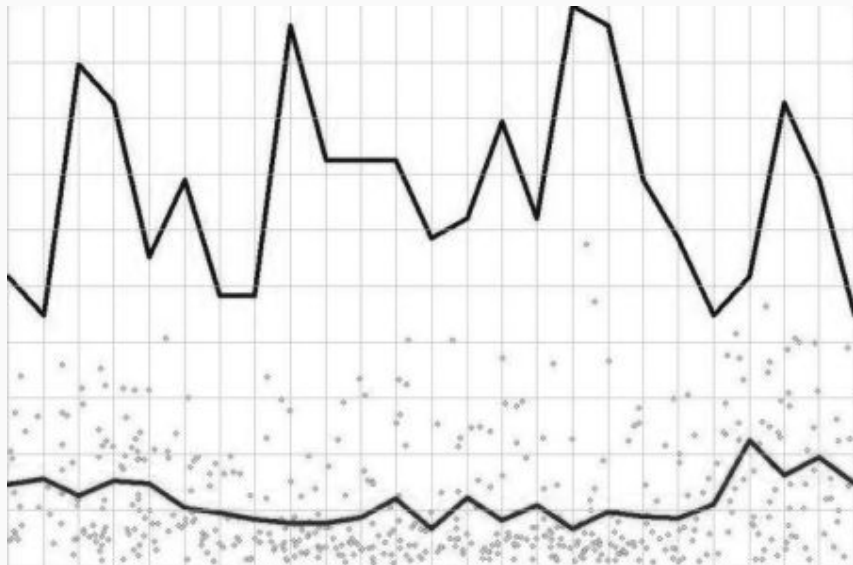
Como retirar pontos de um gráfico sem seu código?



Como retirar pontos de um gráfico sem seu código?

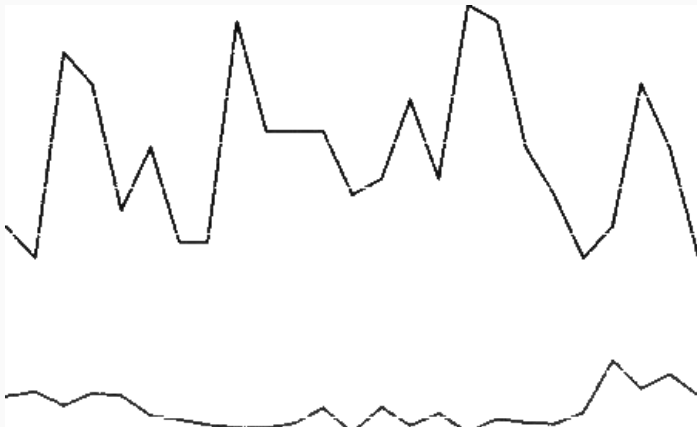
```
library(tidyverse)
im <- image_read("IMAGENS/grafico_ponto.jpg")
im_proc <- im %>%
  image_channel("saturation")
image_write(im_proc, path = "IMAGENS/grafico_ponto1.png",
            format = "png")
```

Como retirar pontos de um gráfico sem seu código?



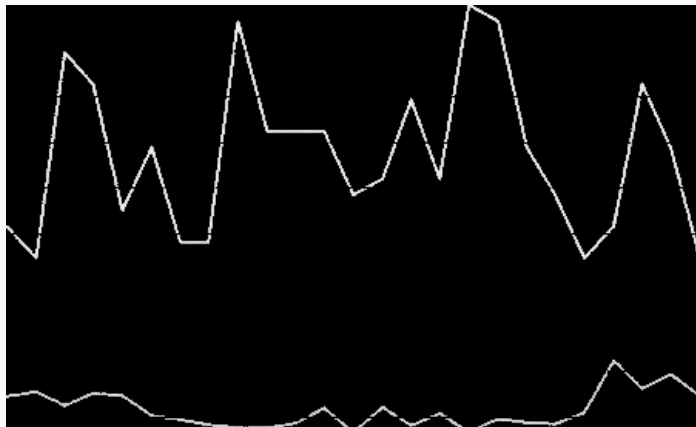
Como retirar pontos de um gráfico sem seu código?

```
im_proc2 <- im_proc %>%  
  image_threshold("white", "30%")  
image_write(im_proc2, path = "IMAGENS/grafico_ponto2.png",  
            format = "png")
```



Como retirar pontos de um gráfico sem seu código?

```
im_proc3 <- im_proc2 %>%  
  image_negate()  
image_write(im_proc3, path = "IMAGENS/grafico_ponto3.png",  
            format = "png")
```



Como retirar pontos de um gráfico sem seu código?

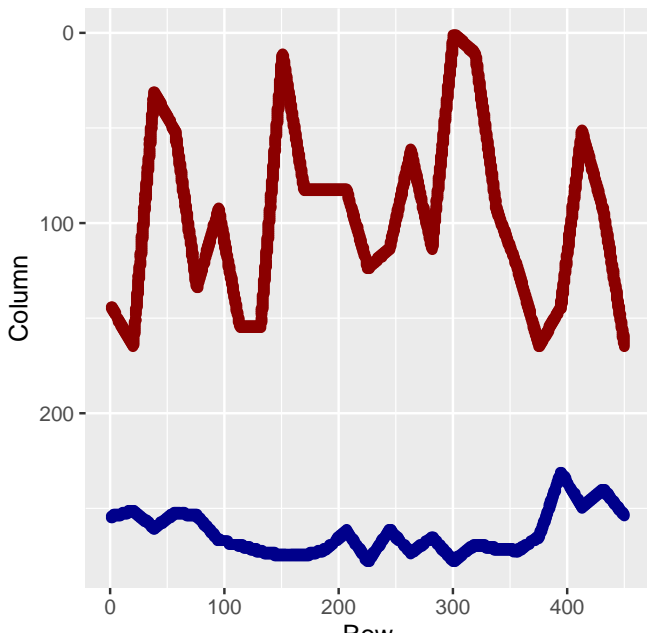
```
require(tidyverse)

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))
```

Como retirar pontos de um gráfico sem seu código?

```
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)
```

Como retirar pontos de um gráfico sem seu código?



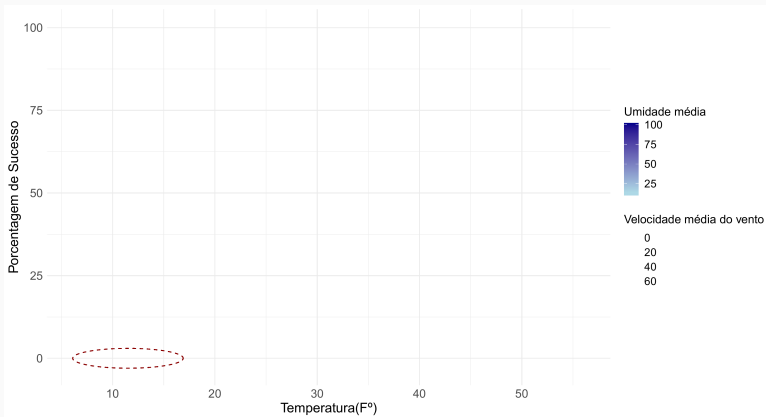
Fraqueza na leitura de PDF

```
require(pdftools)

tempo <- image_read_pdf("IMAGENS/temp.pdf")

image_write(tempo, path = "tempo.pdf", format = "pdf")
```

Fraqueza na leitura de PDF



```
earth <- image_read(  
  "https://jeroen.github.io/images/earth.gif"  
) %>%  
  image_scale("250x") %>%  
  image_quantize()  
  
length(earth)
```

```
## [1] 44
```

Como montar um GIF

1º - Importe as imagens:

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

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

2º - Junte as imagens e redimensione:

```
⇒ img <- c(im_1, ... , im_n)
```

```
⇒ img <- image_scale(img, "300x300")
```

3º - Argumentos:

```
⇒ image_animate(img)
```


Exemplo de GIF (Mês por frame com anotações)

```
# Importando os gráficos do computador
im_1 <- image_read("IMAGENS/GIF/a1.png")
...
im_11 <- image_read("IMAGENS/GIF/a11.png")
# Adicionando uma frase com o mês correspondente
im_1 <- image_annotate(im_1, "Janeiro", size = 27,
                        color = "darkblue",
                        location = "+630+360")
...
im_11 <- image_annotate(im_11, "Novembro", size = 27,
                        color = "darkblue",
                        location = "+630+360")
```

Exemplo de GIF (Mês por frame com anotações)

```
img <- c(im_1,im_2,im_3,im_4,im_5,im_6,im_7,im_8,im_9,im_10)
img <- image_scale(img, "600x600")
img <- image_animate(img, fps = 1)
```

```
library(gifski)
```

```
image_write_gif(img, path = "IMAGENS/GIF/GIF.gif")
```

```
# É possível adicionar um delay
```

```
image_write_gif(img, path = "IMAGENS/GIF/GIF_delay.gif",
                 delay = 1/2)
```

GIF e filtro Voronoi

```
# Download da imagem  
file="http://ereaderbackgrounds.com/movies/bw/Imagem.jpg"  
download.file(file, destfile = "imagem.jpg", mode = 'wb')  
# Lê e converte para a escala de cinza  
load.image("C:/Users/nick_/Downloads/Imagem.jpg") %>%  
  grayscale() -> x
```

GIF e filtro Voronoi

```
# Isso é apenas para 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
```

GIF e filtro Voronoi

```
# Filtra a imagem e converte para preto e branco  
x %>%  
  threshold("45%") %>%  
  as.cimg() %>%  
  as.data.frame() -> df
```

GIF e filtro Voronoi

```
# Função para calcular e plotar o diagrama de Voronoi,  
# depende 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
```

GIF e filtro Voronoi

```
# 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
```

GIF e filtro Voronoi

```
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)}
```


GIF e filtro Voronoi

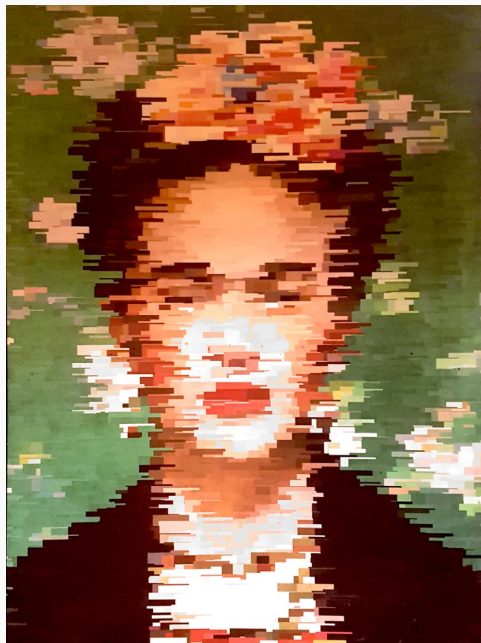
```
# Eu chamei a função anterior e salvei o resultado do plot em  
# formato jpeg  
i= 500  
name=paste0("imagem",i,".jpeg")  
jpeg(name, width = 600, height = 800, units = "px",  
      quality = 100)  
doPlot(i)  
dev.off()
```

GIF e filtro Voronoi

Assim que todas as imagens são salvas, eu posso 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)
```

Filtro geométrico



Filtro geométrico

Por meio do pacote “imager” é possível estilizar uma imagem e deixá-la pixelizada:

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

Filtro geométrico

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





Keras



TensorFlow

Anaconda



Reconhecimento de imagem - Aviões e carros



Reconhecimento de imagem - Pacotes

```
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()
```

Reconhecimento de imagem - Leitura

```
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])}
```

Reconhecimento de imagem - Análise manual

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


Reconhecimneto de imagem - Remodelagem

```
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)
```

Reconhecimneto de imagem - Criação de labels

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

Reconhecimento de imagem - 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)
```

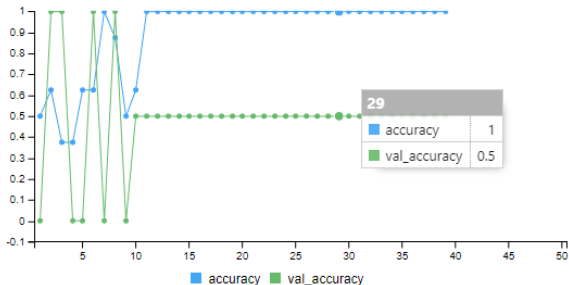
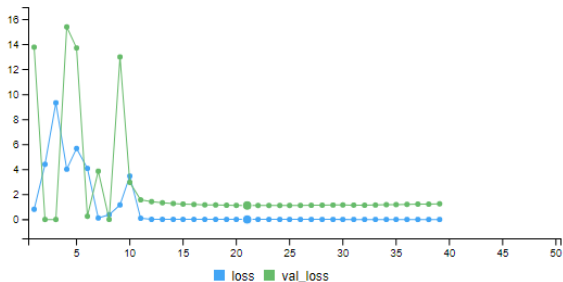

Reconhecimento de imagem - Compilação

```
model %>%  
  compile(loss = 'binary_crossentropy',  
          optimizer = optimizer_rmsprop(),  
          metrics = 'accuracy')
```

Reconhecimento de imagem - Ajustando/ Treinando o modelo

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

Reconhecimento de imagem - Ajustando/ Treinando o modelo



Reconhecimento de imagem - Avaliação e previsão

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