

# Tópicos em genética e melhoramento de plantas

PORCENTAGEM DA ÁREA AFETADA POR ÁREA FOLIAR

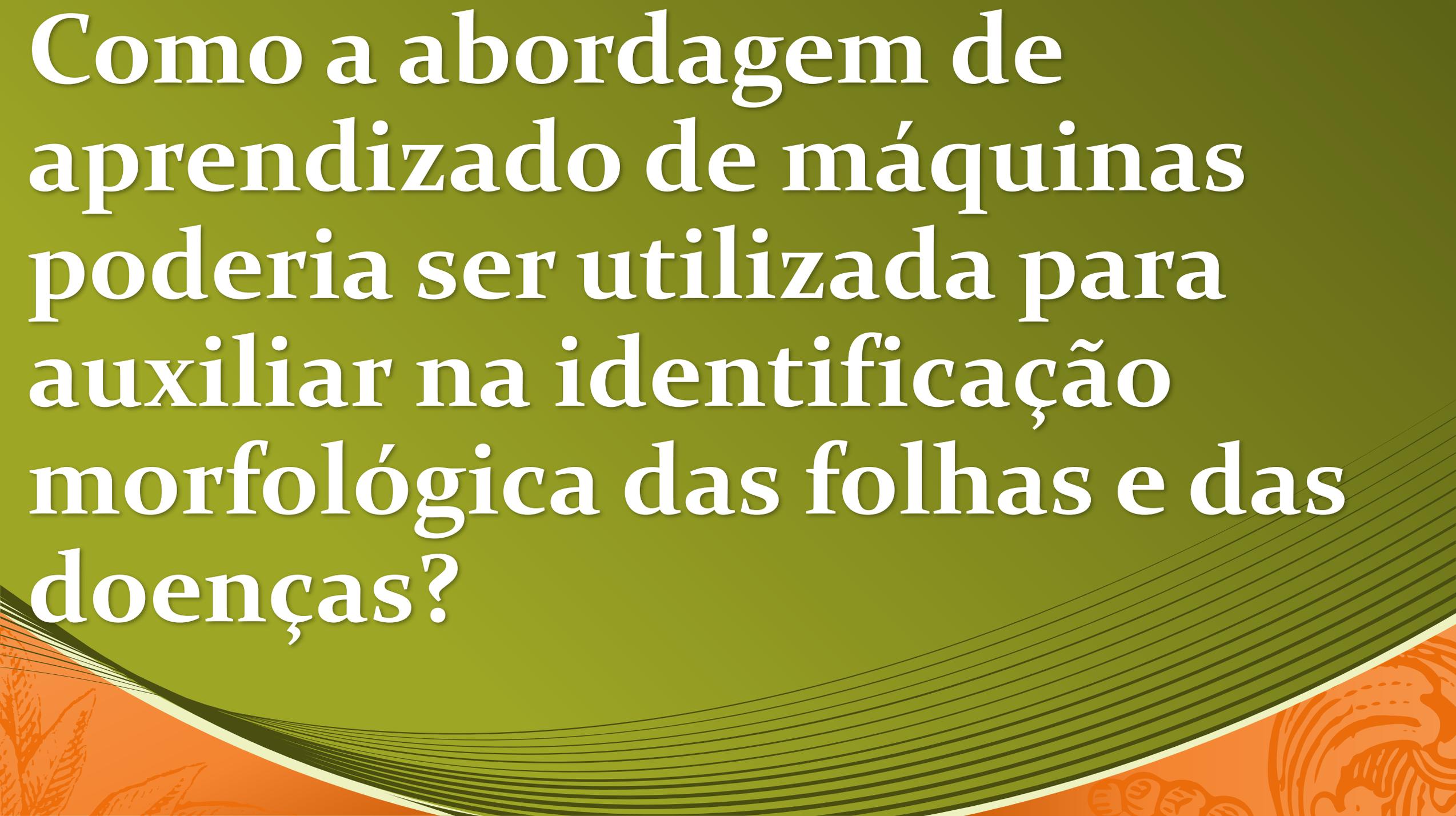
ALUNAS: MELINA PRADO

BRUNA ORSI

PROFESSORES: JOSÉ BALDIN PINHEIRO

FERNANDO ANGELO PIOTO

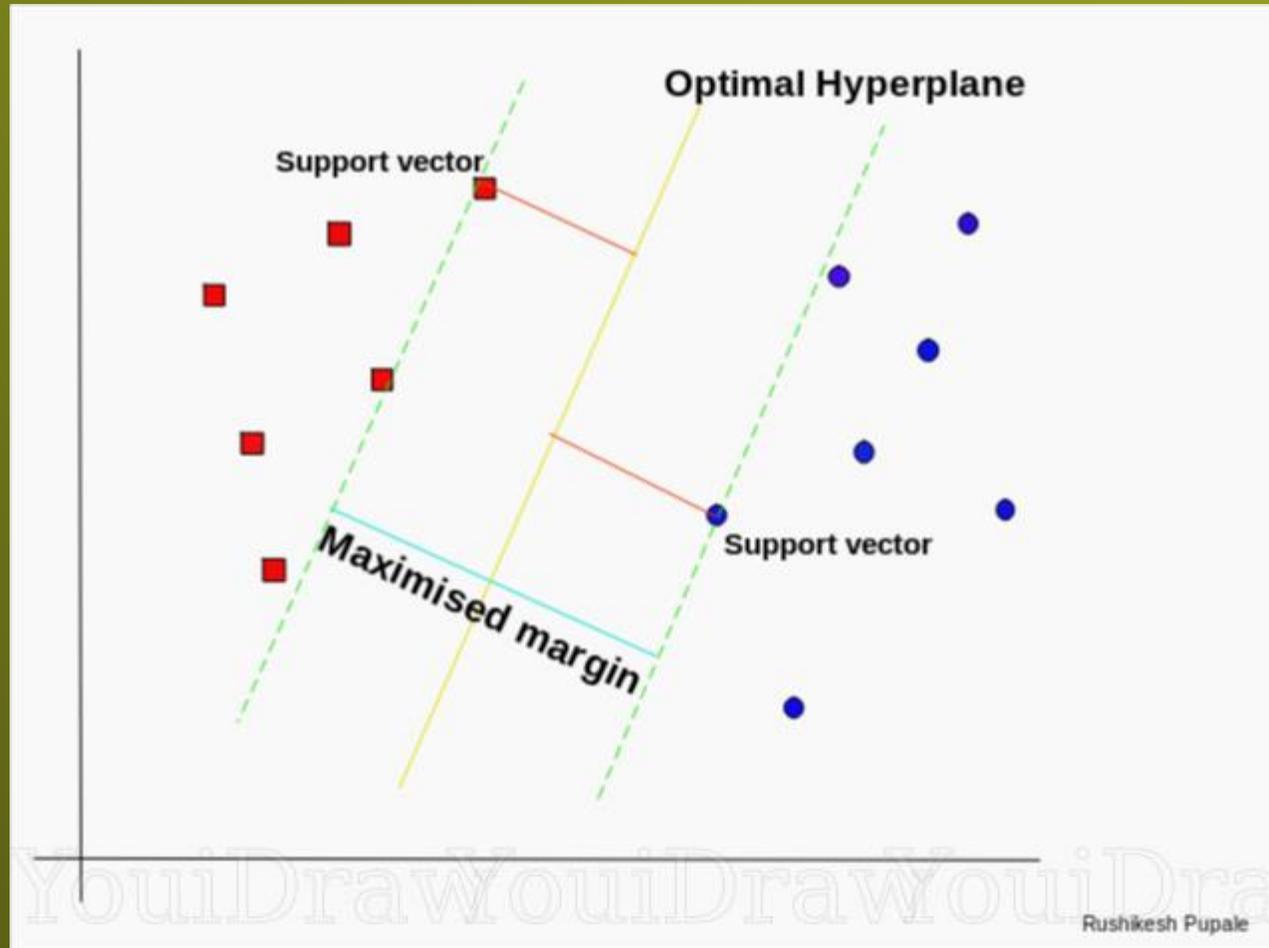
Como a abordagem de aprendizado de máquinas poderia ser utilizada para auxiliar na identificação morfológica das folhas e das doenças?



# Etapas:

- Pré-processamento
  - Extração de características
  - Classificação
  - Validação
- SVM
  - k-nearest neighbour
  - Árvores de decisão
  - CNN
- 

# Support Vector Machine

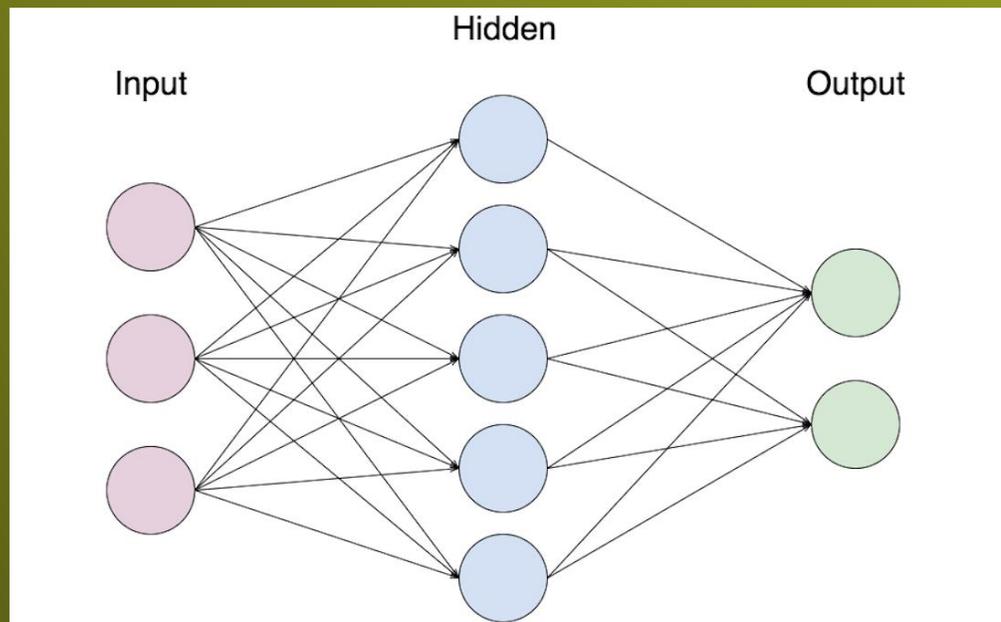


- Associação com funções kernel

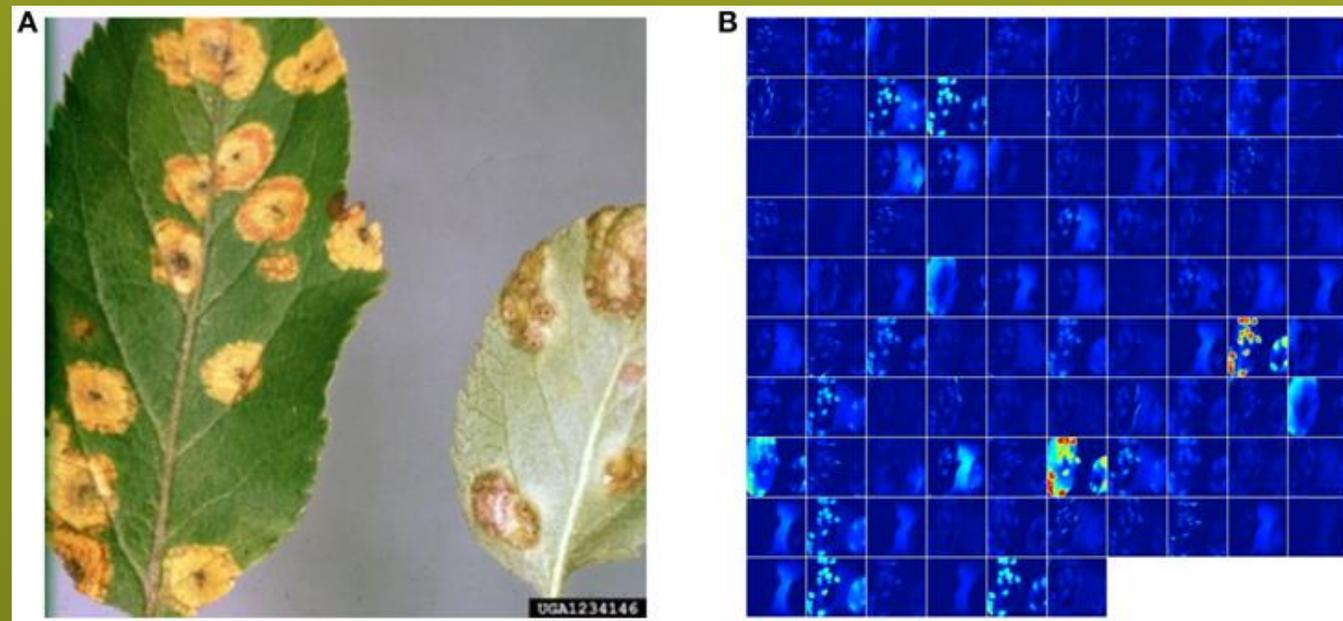
# Redes Neurais artificiais

- Redes Neurais convolucionais
- Detecção de padrões

# Redes Neurais convolucionais



Lee (2018)



Mohanty, Hughes e Salathé (2016)

Arquiteturas pré-definidas: AlexNet, GoogleNet, Overfeat, VGG...

Utilizando machine learning  
para prever a porcentagem  
da área afetada por área  
foliar:

# Pré-processamento - R

Pacotes:

- **Magick**: Processamento das imagens (contraste, brilho, saturação)

Source

Files Plots Packages Help Viewer



Console Terminal x

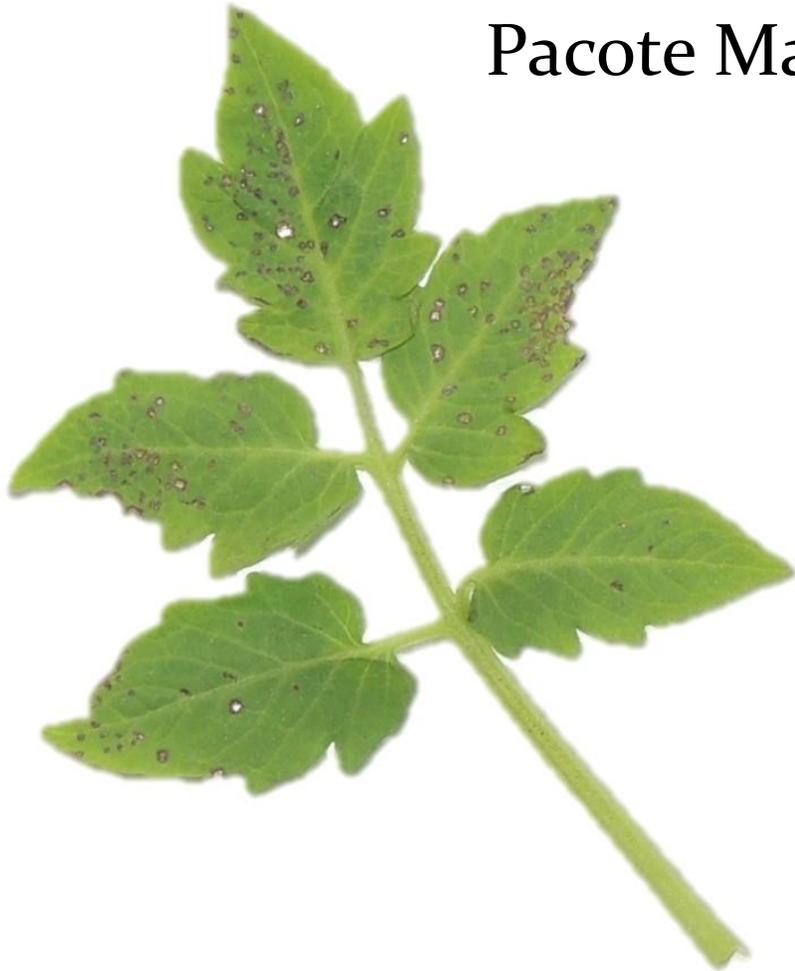
```
> rm(list = ls(all=TRUE))
> library(magick)
> imagem2<- image_read(("C:/Planilha_R/L2B1.jpg"))
> #saturação, brilho, angulo hue
> imagem3<-image_modulate(imagem2, brightness = 100, saturation = 120,
hue=90)
> #contraste
> imagem4<-image_contrast(imagem2, sharpen = 50)
> #Maior contraste
> imagem5<- image_normalize(imagem2)
> print(imagem2)
  format width height colorspace matte filesize density
1  JPEG  4608   3456          sRGB FALSE  2250402 300x300
> |
```

Environment

History

Connections

# Pacote Magick





# Extração de características - R

- **Colordistance:** Visualizar e determinar o valor dos pixels da imagem
- **Countcolors:** Contagem da porcentagem de pixels de um valor médio de RGB

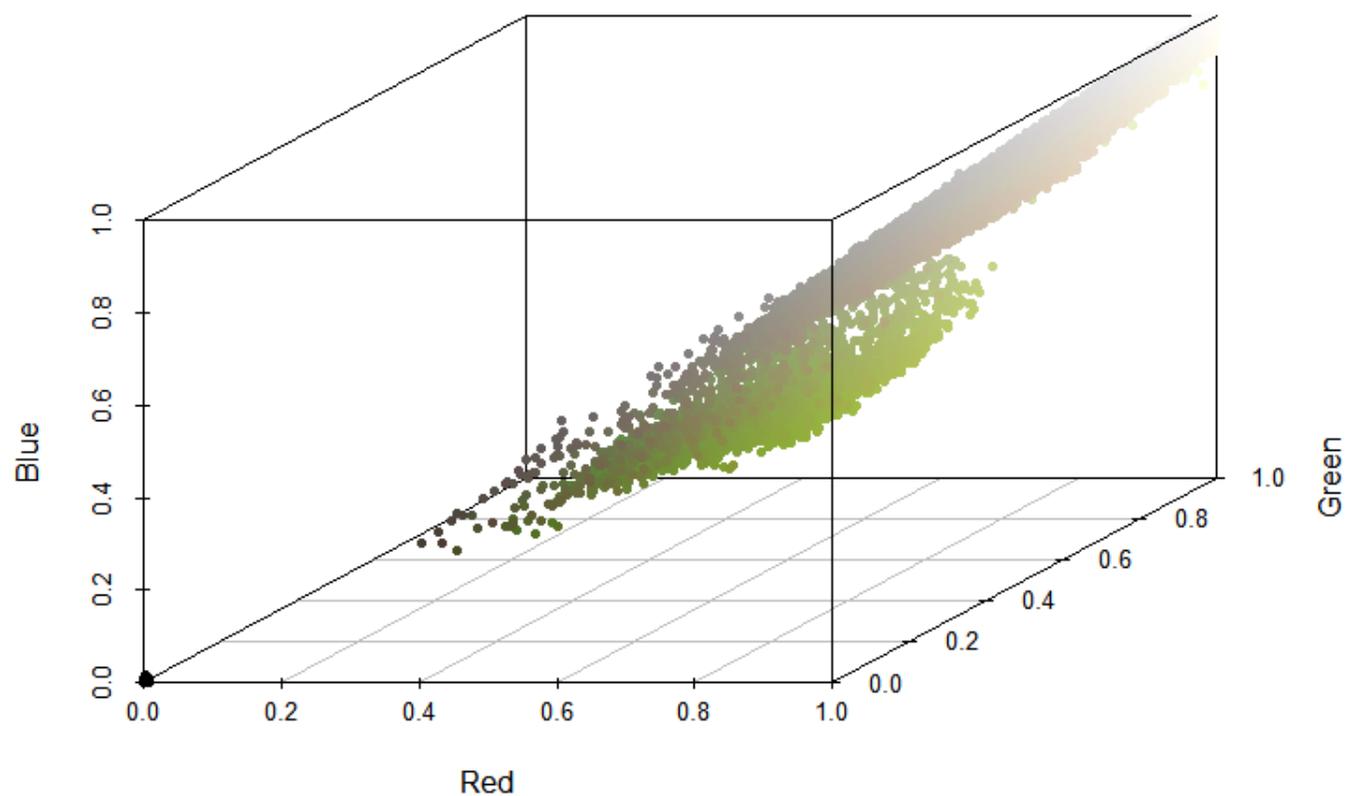
Source

Files Plots Packages Help Viewer

Zoom Export

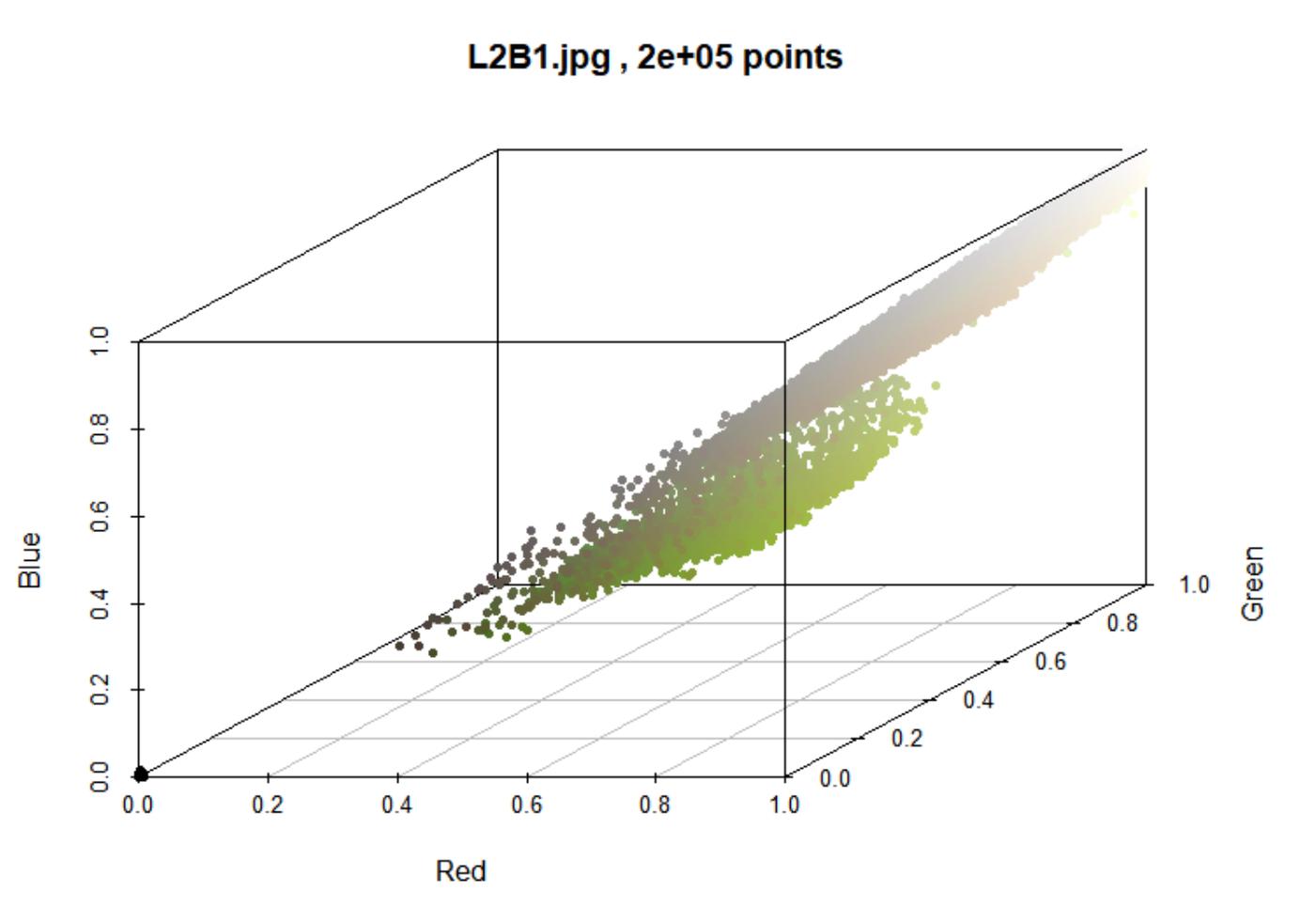
Publish

L2B1.jpg , 2e+05 points



Console Terminal x

```
~/ > rm(list = ls(all=TRUE))
> library(magick)
> imagem2<- image_read(("C:/Planilha_R/L2B1.jpg"))
> #saturação, brilho, angulo hue
> imagem3<-image_modulate(imagem2, brightness = 100, saturation = 120,
hue=90)
> #contraste
> imagem4<-image_contrast(imagem2, sharpen = 50)
> #Maior contraste
> imagem5<- image_normalize(imagem2)
> print(imagem2)
format width height colorspace matte filesize density
1 JPEG 4608 3456 sRGB FALSE 2250402 300x300
> library(colordistance)
> #Plotando gráfico com os pontos RGB
> colordistance::plotPixels("C:/Planilha_R/L2B1.jpg", lower = NULL, upper = NULL, n = 200000)
> |
```



```
> rm(list = ls(all=TRUE))
> library(magick)
> imagem2<- image_read(("C:/Planilha_R/L2B1.jpg"))
> #saturação, brilho, angulo hue
> imagem3<-image_modulate(imagem2, brightness = 100, saturation = 120,
hue=90)
> #contraste
> imagem4<-image_contrast(imagem2, sharpen = 50)
> #Maior contraste
> imagem5<- image_normalize(imagem2)
> print(imagem2)
format width height colorspace matte filesize density
1 JPEG 4608 3456 SRGB FALSE 2250402 300x300
> library(colordistance)
> #Plotando gráfico com os pontos RGB
> colordistance::plotPixels("C:/Planilha_R/L2B1.jpg", lower = NULL, upper = NULL, n = 200000)
> #Extraindo o valor dos pontos
> kmeans.clusters <- colordistance::getKMeanColors("C:/Planilha_R/L2B1.jpg", n = 10, plotting = FALSE)
> colordistance::extractClusters(kmeans.clusters)
```

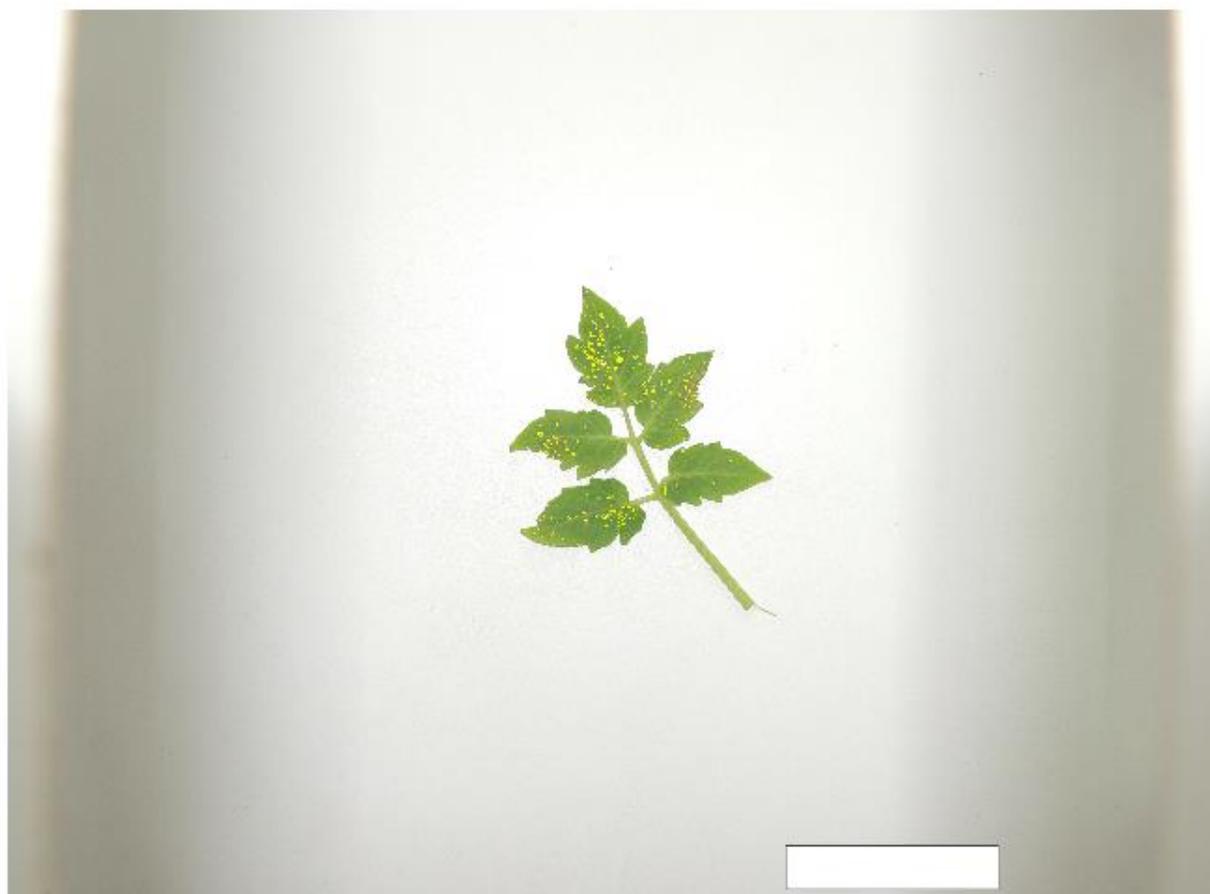
	R	G	B	Pct
1	0.61870256	0.60864564	0.54384883	0.04960
2	0.71648133	0.71626823	0.66329153	0.10765
3	0.51010230	0.62132992	0.29514919	0.02300
4	0.93673664	0.94061382	0.92017441	0.15020
5	0.83129778	0.83168470	0.80574029	0.10490
6	0.77002411	0.77152116	0.73215053	0.08985
7	0.88335594	0.88572074	0.86553936	0.13880
8	0.03431373	0.03063725	0.02377451	0.00080
9	0.67224883	0.66650699	0.60665730	0.09210
10	0.98639630	0.98883700	0.97597696	0.24310

Source

Files Plots Packages Help Viewer

Zoom Export

Publish



Console Terminal

```
>
>
> #Definindo o ponto que será usado
>
> center.spherical <- c(0.42, 0.18, 0.19)
> library(countcolors)
> library(jpeg)
> fol <- jpeg::readJPEG("C:/Planilha_R/L2B1.jpg")
> folha <- countcolors::sphericalRange(fol, center = center.spherical,
radius = 0.21, color.pixels = FALSE, plotting = FALSE); names(folha)
[1] "pixel.idx" "pixel.count" "img.fraction" "original.img"
> folha$img.fraction
[1] 0.001346227
>
> #visualizando os resultados
> countcolors::changePixelColor(fol, folha$pixel.idx, target.color="yellow")
> |
```

# Pacote Countcolors











# Programa Python

```
#Importando a biblioteca necessária
import cv2 #Importando a biblioteca cv2

#Recortando imagem original
imagem = cv2.imread("L2B1.jpg") #Lendo a imagem e a colocando em uma variável
fatia = imagem[200:2600, 1550:3200] #Fatiando a imagem
cv2.imwrite("folharecortada.png", fatia) #Salvando a imagem resultante como folharecortada

#Deixando a imagem suave
img = cv2.imread('folharecortada.png') ##Lendo a imagem e a colocando em uma variável
suave = cv2.GaussianBlur(img, (11, 11), 0) #Desfocando e suavizando a imagem
cv2.imwrite("suave.png", suave) #Salvando a imagem como suave
cv2.waitKey(0)
```



# Programa Python

```
#Importando a biblioteca necessária
import cv2 #Importando a biblioteca cv2

#Recortando imagem original
imagem = cv2.imread("L2B1.jpg") #Lendo a imagem e a colocando em uma variável
fatia = imagem[200:2600, 1550:3200] #Fatiando a imagem
cv2.imwrite("folharecortada.png", fatia) #Salvando a imagem resultante como folharecortada

#Deixando a imagem suave
img = cv2.imread('folharecortada.png') ##Lendo a imagem e a colocando em uma variável
suave = cv2.GaussianBlur(img, (11, 11), 0) #Desfocando e suavizando a imagem
cv2.imwrite("suave.png", suave) #Salvando a imagem como suave
cv2.waitKey(0)
```

## 2. Gaussian Blurring

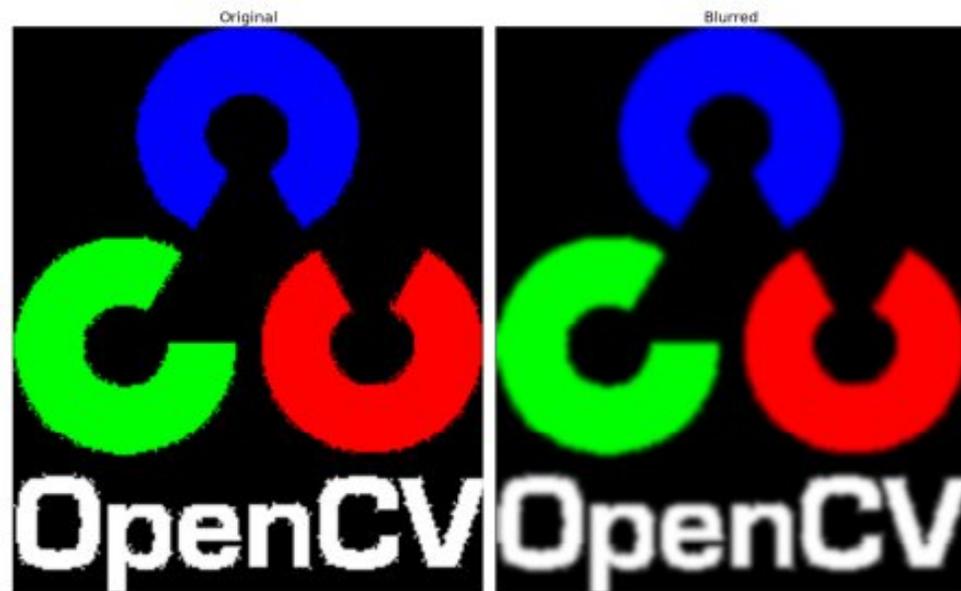
In this, instead of box filter, gaussian kernel is used. It is done with the function, `cv2.GaussianBlur()`. We should specify the width and height of kernel which should be positive and odd. We also should specify the standard deviation in X and Y direction, `sigmaX` and `sigmaY` respectively. If only `sigmaX` is specified, `sigmaY` is taken as same as `sigmaX`. If both are given as zeros, they are calculated from kernel size. Gaussian blurring is highly effective in removing gaussian noise from the image.

If you want, you can create a Gaussian kernel with the function, `cv2.getGaussianKernel()`.

The above code can be modified for Gaussian blurring:

```
1 blur = cv2.GaussianBlur(img,(5,5),0)
```

Result:



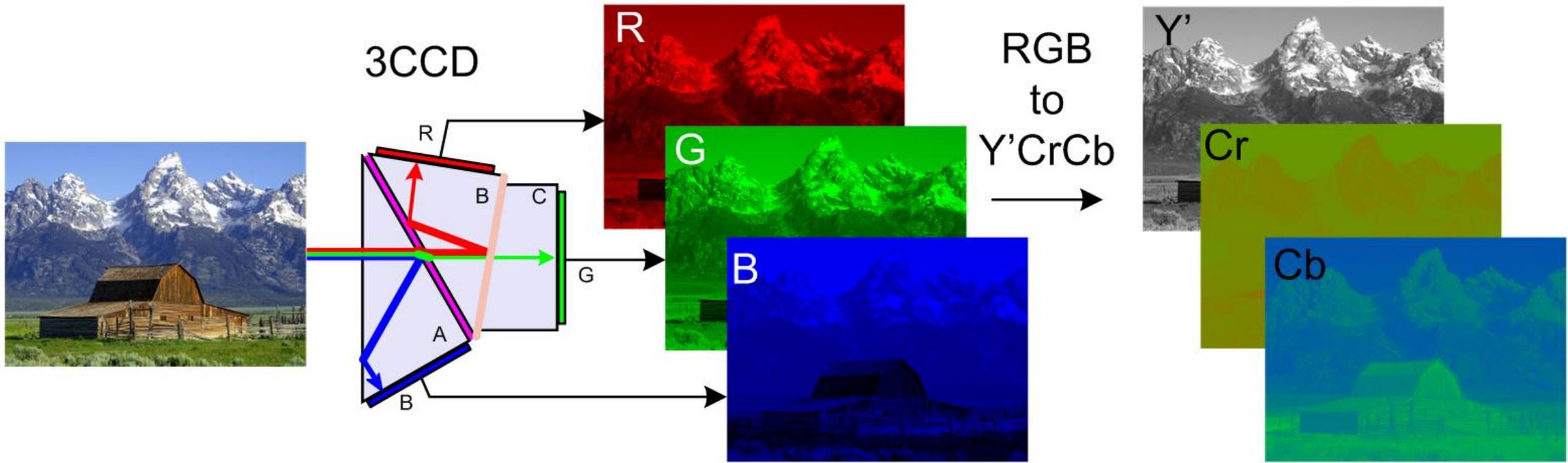


```
}#Deixando a imagem com alto contraste
img2 = cv2.imread('folharecortada.png') #Variável recebe e lê a imagem
contraste = cv2.cvtColor(img2, cv2.COLOR_BGR2YUV) #Variável contraste recebe imagem RGB com novo formato, YUV
contraste[:, :, 0] = cv2.equalizeHist(contraste[:, :, 0]) #Variável recebe imagem realçada por equalização
realce_equalizacao = cv2.cvtColor(contraste, cv2.COLOR_YUV2BGR) #Variável recebe a imagem YUV equalizada com novo formato, RGB
cv2.imwrite('contraste.jpg', realce_equalizacao) #Salvando a imagem como contraste

#Recebendo as imagens e verificando os tamanhos
imgfolha = cv2.imread("suave.png", cv2.IMREAD_GRAYSCALE) #Variável recebendo a imagem suavizada em escala de cinza
imgdoenca = cv2.imread("contraste.jpg", cv2.IMREAD_GRAYSCALE) #Variável recebendo a imagem com contraste em escala de cinza
x, y = imgdoenca.shape #Verificando o tamanho da imagem
totalpixels = x*y #Verificando o tamanho total da imagem em pixels
print("Quantidade total de pixels: %d" % (totalpixels)) #Imprime o total de pixels
```

# Realce por contraste

- Melhorar a qualidade das imagens sob os critérios subjetivos do olho humano.
- Aumenta a discriminação visual entre os objetos presentes na imagem. Realiza-se a operação ponto a ponto, independentemente da vizinhança.
- O contraste entre dois objetos pode ser definido como a razão entre os seus níveis de cinza médios.



Qiita(2008)

## Histograms Equalization in OpenCV

OpenCV has a function to do this, `cv2.equalizeHist()`. Its input is just grayscale image and output is our histogram equalized image.

Below is a simple code snippet showing its usage for same image we used :

```
1 img = cv2.imread('wiki.jpg',0)
2 equ = cv2.equalizeHist(img)
3 res = np.hstack((img,equ)) #stacking images side-by-side
4 cv2.imwrite('res.png',res)
```



image

So now you can take different images with different light conditions, equalize it and check the results.

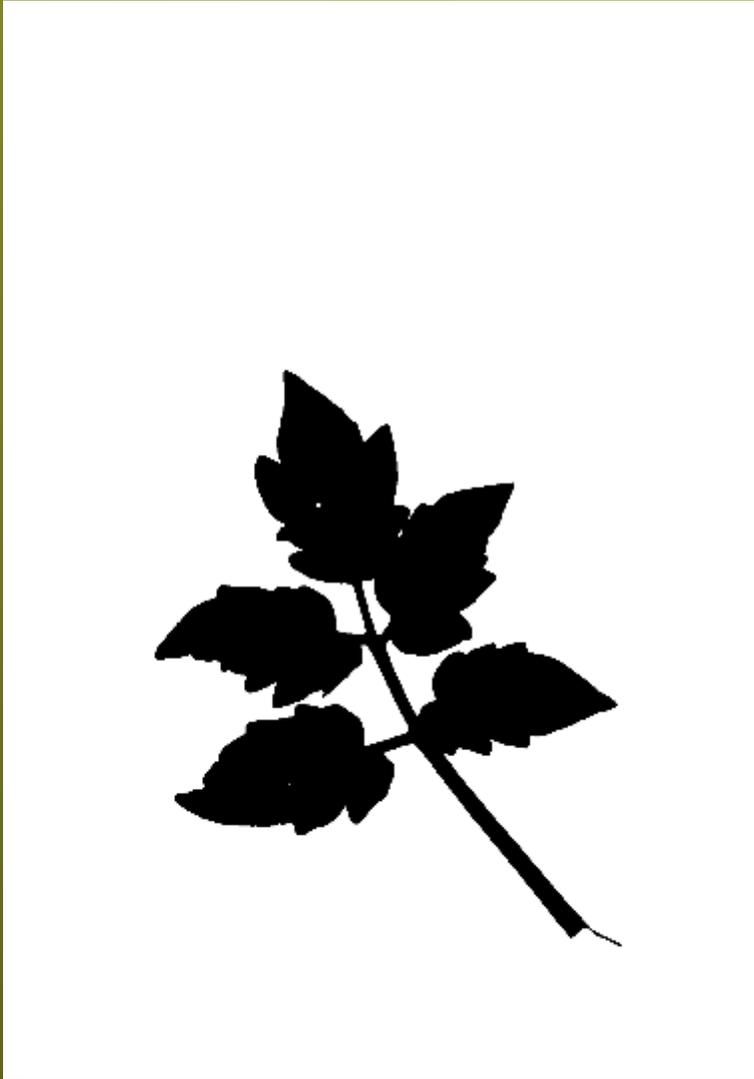
Histogram equalization is good when histogram of the image is confined to a particular region. It won't work good in places where there is large intensity variations where histogram covers a large region, ie both bright and dark pixels are present. Please check the SOF links in Additional Resources.



```
#Transformação para imagem binária
_, folhabinaria = cv2.threshold(imgfolha, 210, 300, cv2.THRESH_BINARY) #Variável recebe a imagem para a folha binária
_, doencabinaria = cv2.threshold(imgdoenca, 10, 300, cv2.THRESH_BINARY) #Variável recebe a imagem para doença binária
cv2.imwrite("Folhabinaria.png", folhabinaria) #Salva a imagem
cv2.imwrite("Doencabinaria.png", doencabinaria) #Salva a imagem

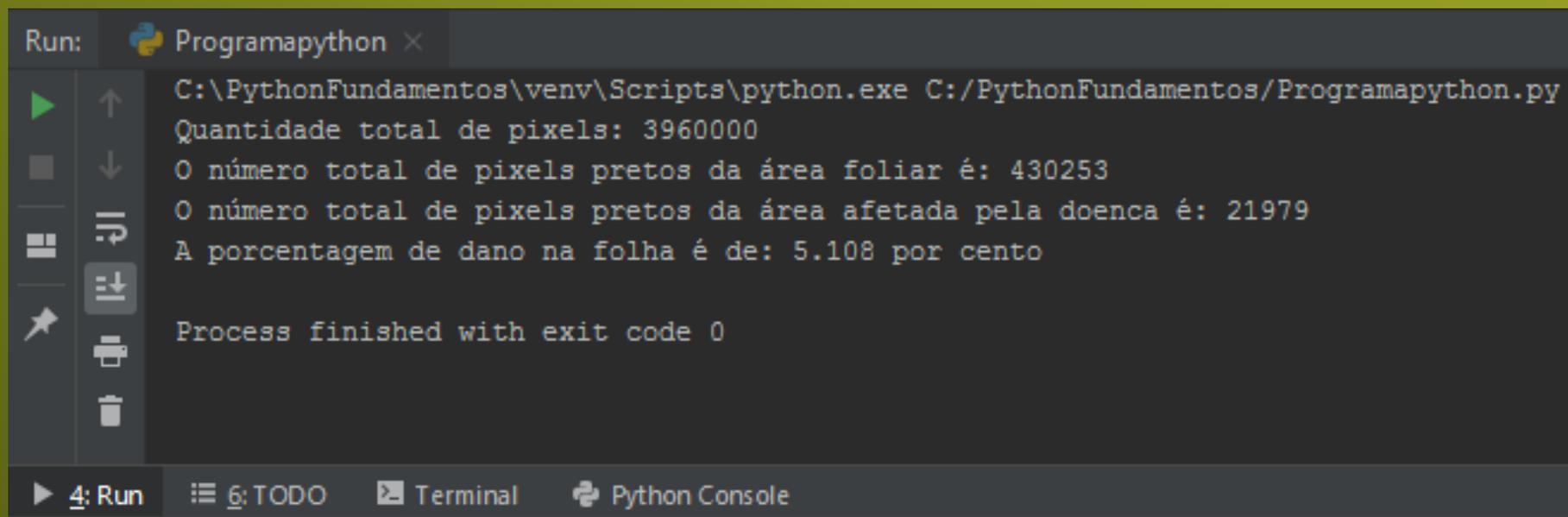
#Looping para folha
matriz = folhabinaria #Variável recebe a imagem binária da folha
total=0 #Variável recebe zero
for l in range(0,2400): #Estrutura de repetição para todas as 2400 linhas
    for c in range(0,1650): #Estrutura de repetição para todas as 1650 colunas
        if matriz[l][c]==0: #Estrutura condicional, se o pixel tiver um valor de 0 (preto) segue
            total+=1 #Variável soma um ao valor total
        if matriz[l][c]==255: #Estrutura condicional, se o pixel tiver um valor de 255 (branco) segue
            total+=0 #Variável soma zero ao valor total
print("O número total de pixels pretos da área foliar é: %r" % (total)) #Imprime o número total de pixels pretos da área foliar

#Looping para doença
matriz2 = doencabinaria #Variável recebe a imagem binária da doença
total2=0 #Variável recebe zero
for l in range(0,2400): #Estrutura de repetição para todas as 2400 linhas
    for c in range(0,1650): #Estrutura de repetição para todas as 1650 colunas
        if matriz2[l][c]==0: #Estrutura condicional, se o pixel tiver um valor de 0 (preto) segue
            total2+=1 #Variável soma um ao valor total
        if matriz2[l][c]==255: #Estrutura condicional, se o pixel tiver um valor de 255 (branco) segue
            total2+=0 #Variável soma zero ao valor total
print("O número total de pixels pretos da área afetada pela doença é: %r" % (total2)) #Imprime o número total de pixels pretos da área da doença
```



```
#Calcula a porcentagem de doença pela área foliar
pctgdano = (total2/total)*100 #Variável recebe o valor do cálculo da porcentagem do dano por área foliar
arredonda = round(pctgdano,3) #Variável recebe função para limitar o número de casas depois da vírgula
print(f"A porcentagem de dano na folha é de: {r por cento" % (arredonda)) #Imprime a porcentagem do dano na folha

cv2.waitKey(0)
cv2.destroyAllWindows()
```



```
Run: Programapython x
C:\PythonFundamentos\venv\Scripts\python.exe C:/PythonFundamentos/Programapython.py
Quantidade total de pixels: 3960000
O número total de pixels pretos da área foliar é: 430253
O número total de pixels pretos da área afetada pela doença é: 21979
A porcentagem de dano na folha é de: 5.108 por cento

Process finished with exit code 0
```

4: Run | 6: TODO | Terminal | Python Console

# Classificação - ML

- Dados:

- Área lesionada:

% pixels das áreas lesionadas

- Área lesionada por folha:

% pixels das áreas lesionadas

---

% área foliar total

# Classificação - ML

- Definição das categorias:
- Suscetível:  $\geq 2\%$
- Resistente:  $< 2\%$

Linagem 1:



Bloco 1  
Resistente



Bloco 2  
Resistente



Bloco 3  
Suscetível

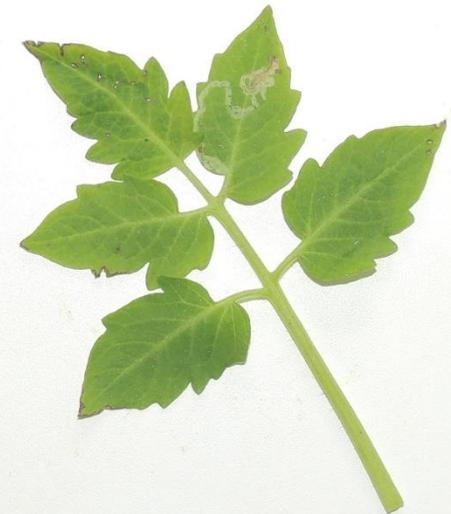
Linhagem 2:



Bloco 1  
Suscetível

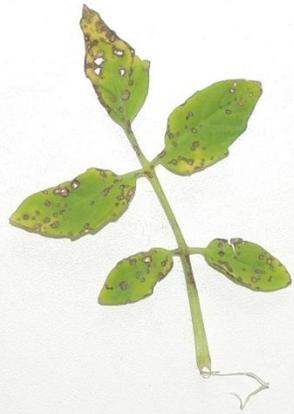


Bloco 2  
Suscetível



Bloco 3  
Resistente

Linagem 3:



Bloco 1  
Suscetível



Bloco 2  
Suscetível

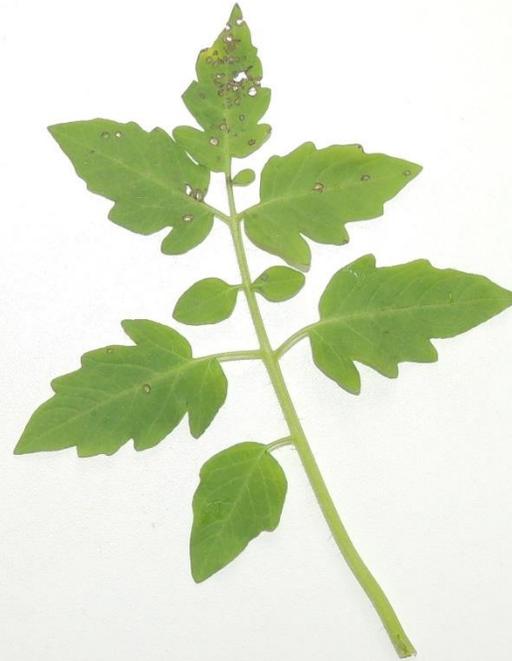


Bloco 3  
Suscetível

Linagem 4:



Bloco 1  
Resistente



Bloco 2  
Resistente



Bloco 3  
Suscetível

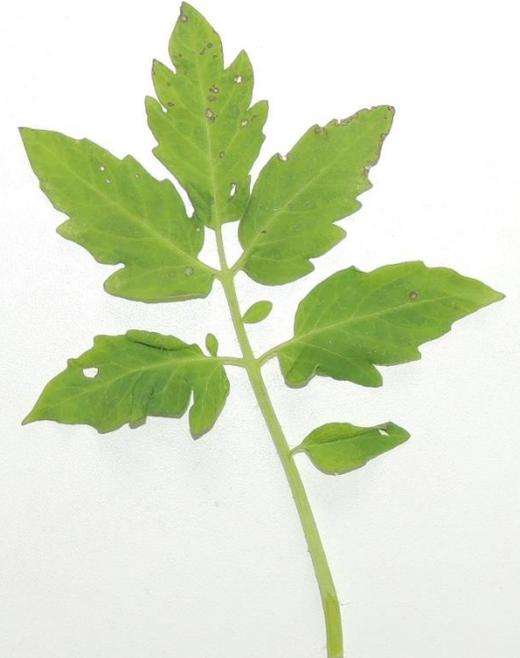
Linagem 5:



Bloco 1  
Resistente



Bloco 2  
Resistente



Bloco 3  
Resistente

```
> rm(list = ls(all=TRUE))
> getwd()
[1] "C:/Users/bruna/Documents"
> setwd("C:/Planilha_R")
> folha<-read.csv2(("C:/Planilha_R/teste_folhas.csv"))
> folha
  linhagem      lesao  area  classe
1      11b1 0.06626584  1.60 resistente
2      11b2 0.01525900  0.52 resistente
3      11b3 0.08998918  2.20 suscetivel
4      12b1 0.09587900  5.79 suscetivel
5      12b2 0.03688483  2.32 suscetivel
6      12b3 0.05189244  1.67 resistente
7      13b1 0.45029750 34.57 suscetivel
8      13b2 0.40575820 35.66 suscetivel
9      13b3 0.55773070 43.56 suscetivel
10     14b1 0.05738686  1.58 resistente
11     14b2 0.06774777  1.99 resistente
12     14b3 0.16721670  9.52 suscetivel
13     15b1 0.06446367  1.46 resistente
14     15b2 0.03945307  1.17 resistente
15     15b3 0.04980142  1.41 resistente
>
> #etapas de preparação dos dados para o treinamento
> set.seed(123)
> peso <- sample(2, nrow(folha), replace=TRUE, prob=c(0.67, 0.33))
>
> #composição dos training set
> trainingf<-folha[peso==1, 2:3]
>
> #composição do test set
> testf<-folha[peso==2, 2:3]
>
```

Determinando  
parâmetros a serem  
utilizados no ML

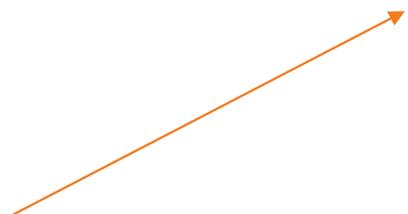


```
> list(trainingf)
[[1]]
      lesao  area
1  0.06626584  1.60
3  0.08998918  2.20
6  0.05189244  1.67
7  0.45029750 34.57
9  0.55773070 43.56
10 0.05738686  1.58
12 0.16721670  9.52
14 0.03945307  1.17
15 0.04980142  1.41
```

```
> list(testf)
[[1]]
      lesao  area
2  0.01525900  0.52
4  0.09587900  5.79
5  0.03688483  2.32
8  0.40575820 35.66
11 0.06774777  1.99
13 0.06446367  1.46
```

```
> #composição dos alvos de treinamento
> trainlabels<-folha[peso==1,4]
>
> #composição dos alvos de teste
> testlabels<-folha[peso==2,4]
> #Verificação dos resultados
> print(trainlabels)
[1] resistente suscetivel resistente suscetivel suscetivel resistente suscetivel resistente resistente
Levels: resistente suscetivel
> print(testlabels)
[1] resistente suscetivel suscetivel suscetivel resistente resistente
Levels: resistente suscetivel
>
```

Colunas sorteadas para o teste e treino



# Pacote Class

```
>  
> #treinamento  
> library(class)  
> pred <- knn(train = trainingf, test = testf, cl = trainlabels, k=2) → k-nearest neighbors  
>
```

Utilizando o mesmo Set.seed...

## Resultado dados R

```
Classes preditas Classes Observadas  
1      resistente      resistente  
2      suscetivel      suscetivel  
3      suscetivel      suscetivel  
4      suscetivel      suscetivel  
5      suscetivel      resistente  
6      resistente      resistente  
> #inspecionando a acurácia do teste  
> acc <- 100 * sum(testlabels == pred)/NROW(testlabels)  
> acc  
[1] 83.33333
```

L4B2

## Resultado dados Python

```
Classes preditas Classes Observadas  
1      resistente      resistente  
2      resistente      suscetivel  
3      suscetivel      suscetivel  
4      suscetivel      suscetivel  
5      resistente      resistente  
6      resistente      resistente  
> #inspecionando a acurácia do teste  
> acc <- 100 * sum(testlabels == pred)/NROW(testlabels)  
> acc  
[1] 83.33333
```

L2B1

# Pacote Random Forest

```
> modelo1 <- randomForest(classe ~ ., data = trainset, importance = TRUE)
> modelo1

Call:
randomForest(formula = classe ~ ., data = trainset, importance = TRUE)
  Type of random forest: classification
    Number of trees: 500
No. of variables tried at each split: 1

      OOB estimate of  error rate: 10%
Confusion matrix:
      resistente suscetivel class.error
resistente      5         0         0.0
suscetivel      1         4         0.2
>
> modelo2 <- randomForest(classe ~ ., data = trainset, ntree = 500, mtry = 2, importance = TRUE)
> modelo2

Call:
randomForest(formula = classe ~ ., data = trainset, ntree = 500, mtry = 2, importance = TRUE)
  Type of random forest: classification
    Number of trees: 500
No. of variables tried at each split: 2

      OOB estimate of  error rate: 20%
Confusion matrix:
      resistente suscetivel class.error
resistente      4         1         0.2
suscetivel      1         4         0.2
>
```

Random Forest

# Pacote Random Forest

Utilizando o mesmo Set.seed...

Resultado dados R

```
> predtest <- predict(modelo1, testset, type = "class")
> mean(predtest == testset$classe)
[1] 1
> table(predtest, testset$classe)

predtest      resistente suscetivel
resistente      3          0
suscetivel      0          2
> importance(modelo1)
      resistente suscetivel MeanDecreaseAccuracy MeanDecreaseGini
lesao  9.851763   7.954141      10.05547          1.963095
area  13.920594  11.360860      14.13552          2.574905
> |
>
```

Resultado dados Python

```
> predtest <- predict(modelo1, testset, type = "class")
> mean(predtest == testset$classe)
[1] 1
> table(predtest, testset$classe)

predtest      resistente suscetivel
resistente      3          0
suscetivel      0          2
> importance(modelo1)
      resistente suscetivel MeanDecreaseAccuracy MeanDecreaseGini
lesao  9.542615   7.595565      9.51549          1.916644
area  14.381672  12.474190      14.85254          2.574956
> |
>
```

Quais seriam as alternativas para otimizar a avaliação da planta como um todo, utilizando fenotipagem de alto rendimento?

# Plant Disease Detection by Imaging Sensors – Parallels and Specific Demands for Precision Agriculture and Plant Phenotyping

Article in *Plant Disease* - September 2015

DOI: 10.1094/PDIS-03-15-0340-FE

CITATIONS

148

READS

3,949

1 author:



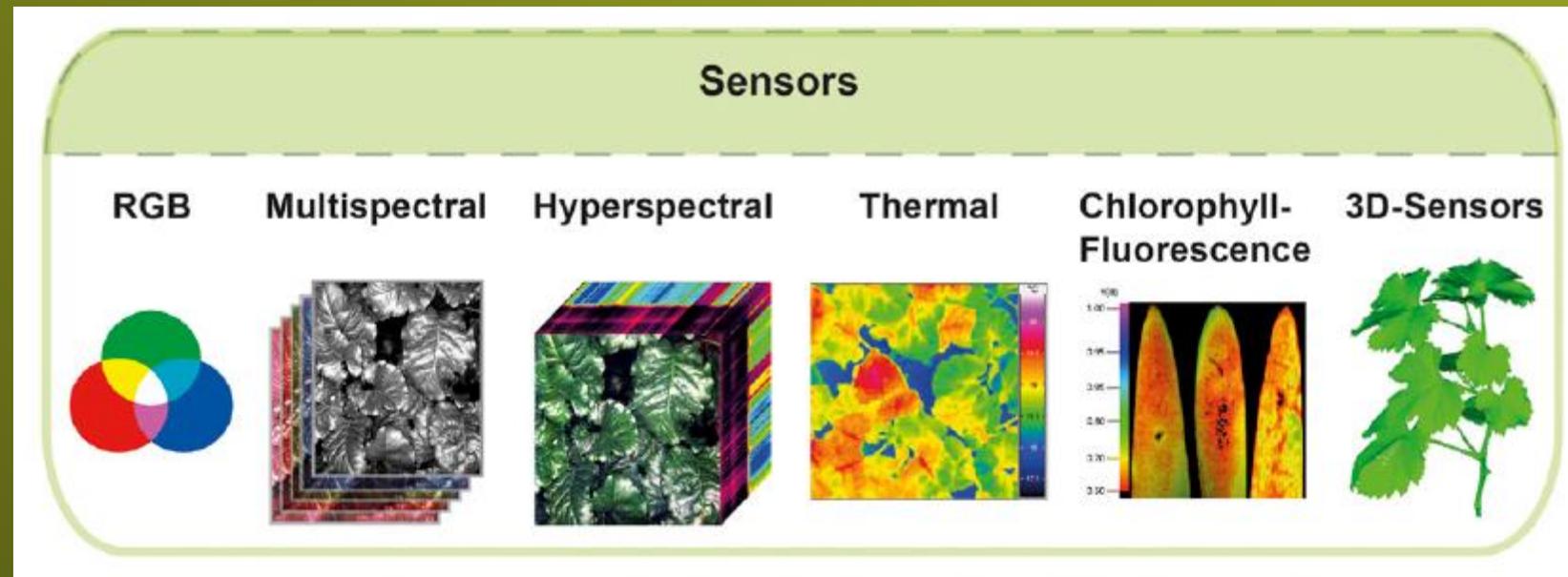
Anne-Katrin Mahlein

Institute of Sugar Beet Research (IfZ) at the University Göttingen

70 PUBLICATIONS 1,874 CITATIONS

[SEE PROFILE](#)

Mahlein(2015)



Mahlein(2015)

# Referência

- Mohanty, S. P.; Hughes, D. P.; Salathé, M. Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, v. 7, 2016.
- Pupale, R. Support Vector Machines(SVM) — An Overview. Disponível em: <https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>.
- Lee, Y. X. Converting a Simple Deep Learning Model from PyTorch to TensorFlow. Disponível em: <https://towardsdatascience.com/converting-a-simple-deep-learning-model-from-pytorch-to-tensorflow-b6b353351f5d>.
- Mahlein, A. K. Plant Disease Detection by Imaging Sensors – Parallels and Specific Demands for Precision Agriculture and Plant Phenotyping. Local de publicação: Editora, ano de publicação. Alemanha: Plant Disease, 2015.

# Referência

- Divisão de Processamento de Imagens. Pagar para fazer trabalho acadêmico é ilegal. Disponível em: <http://www.dpi.inpe.br/DPI/>. Acesso em: 07 de junho de 2019
- Quiita. Pagar para fazer trabalho acadêmico é ilegal. Disponível em: <https://qiita.com/yb8jo/items/22287f341aefa395e789>. Acesso em: 07 de junho de 2019