University of Southeastern Philippines
College of Information and Computing

ICE 415 - Digital Image Processing
Module 2 Image Preprocessing and Analysis
Module 2 Assessment Sheet

Name: Ronmar John S. Sabado Year and Section: 4B-BTM Date: 11/10/2023

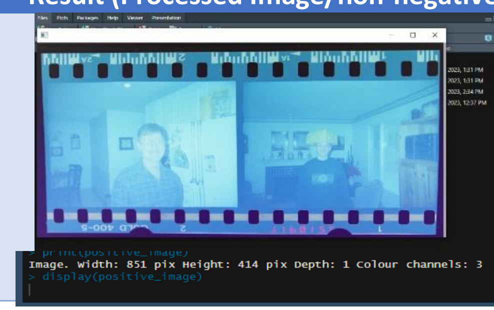
==

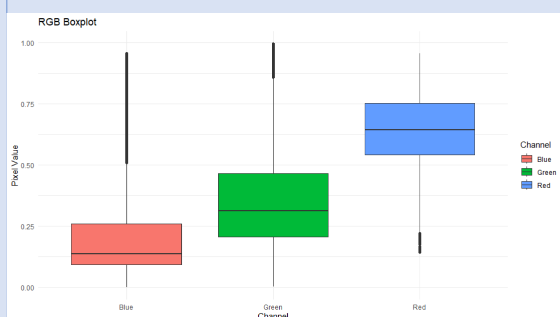
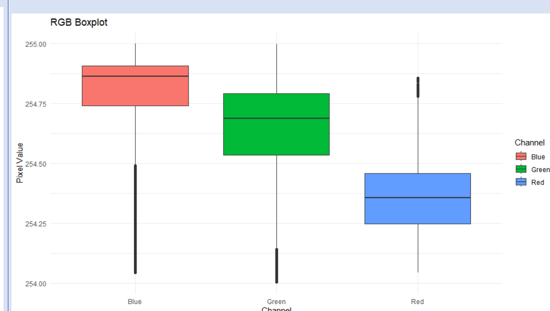
Actual image can be accessed thru this link: https://drive.google.com/drive/folders/1cptF6HzkrdKGTzDhMAHtBqrCothJT28P?usp=share_link

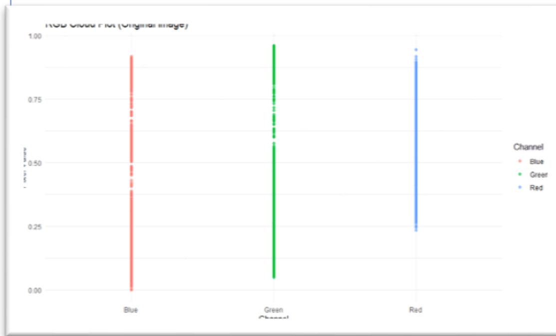
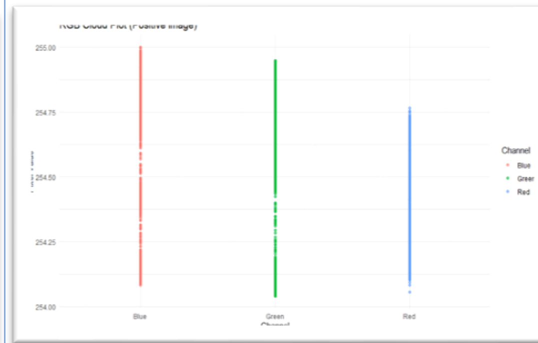
A. Image color channel statistics

You will be given a typical negative image which is sourced from an analog camera, and is digitized through scanning. Obtain the non-negative image by using the image_negate() in magick library or by using the 1-image in openImageR or imager library. Save the processed image by exporting or by using the write image command(as discussed in Module 1 Lesson 3) using the filename format **LASTNAME_ProcessedImageA**. It is preferred to use the write image command in magick library**: image_write(neg_image,path="LASTNAME_ProcessedImageA1.jpg",format="JPG")** because this will save the image excluding the unnecessary background.

As what is covered in this module, obtain the details of the original image (negative image) and processed image (non-negative equivalent) based on the given matrix: (Note: capture/screenshot the result)

Required	Command that you may use	Result (Original Image/negative)	Result (Processed Image/non-negative)
Image dimension, depth, and color channels	print() in imager and gglot2 library Note: do the print() command after reading the image. e.g. library(imager) library(ggplot2)		

	img <- load.image(file.choose()) print(img)		
Red, Green, and Blue statistics	summary()	Min. 1st Qu. Median Mean 3rd Qu. Max. 0.0000 0.1608 0.3882 0.4175 0.6431 0.9961	Min. 1st Qu. Median Mean 3rd Qu. Max. 254.0 254.4 254.6 254.6 254.8 255.0
Red, Green, and Blue statistics – visual/graphic	boxplot()		

Red, Green, and Blue statistics – visual/graphic	cloud()		

What is your general impression on the differences of both images?

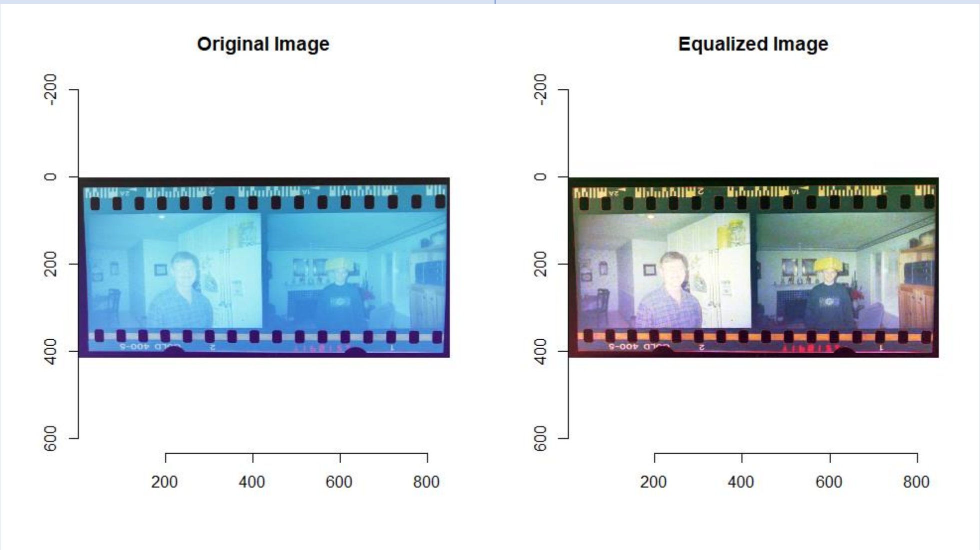
Based on my observation, it appears that the original picture was negative, which is why we applied an inversion to make it appear positive or normal. This inversion was intended to draw attention to the differences between the two pictures. We used a variety of visualization techniques, including summaries, box plots, and cloud representations, to effectively convey the differences. These tools provide a more thorough examination and understanding of the transformation by providing a complete and detailed picture of the changes between the original and inverted photos.

B. Histogram Equalization

Using the processed image in part A, equalize the histogram and save it using the filename format **LASTNAME_ProcessedImageB**. It is preferred you will save the processed image using the command save.image in imager library:

save.image(img1,"LASTNAME_ProcessedImageB.jpg")

Complete the matrix:

Required	Processed Image A	Processed Image B
Histogram		

Output Image		
What are your general impressions of both images? Which is better? **I prefer the equalized image as it effectively balances the RGB components, resulting in a clearer and more vibrant representation.**		

University of Southeastern Philippines
College of Information and Computing

C. Image Hashing (https://cran.r-project.org/web/packages/OpenImageR/vignettes/The_OpenImageR_package.html)

The image hashing functions (*average_hash*, *dhash*, *phash*, *invariant_hash*, *hash_apply*) of the OpenImageR package are implemented in the way **perceptual hashing** works. Perceptual hashing is the use of an algorithm that produces a fingerprint of images (in OpenImageR those fingerprints are *binary features* or *hexadecimal hashes*). The difference between cryptographic and image hashing is that the latter tries to find similar and not exact matches. In cryptographic hashing small differences of the hashes lead to entirely different output, which is not the case for perceptual hashing. A practical application of image hashing would be to compare a database of already created image hashes with a new hash (image) to find similar images in the database.

Refer to the documentation provided in the link and follow the steps in processing image hashing using the processed images in part A and part B (LASTNAME_ProcessedImageA and LASTNAME_ProcessedImageB).

Complete the matrix: (Average_hash is already done.)

Method	Processed Image A	Processed Image B	Do they have the same value? If no, highlight the different bits/literals and count.
aveg_hash	0177777777477e00	0077777747477e00	0077777747477e00; 2 literals
aveg_bin	[1] 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 [33] 1 1 1 0 1 1 1 0 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0	[1] 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 [33] 1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0	[1] 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 [33] 1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 3 bits
ph_hash	"a3cb9c329873b40f"	"a349dc329873b58b"	"a349dc329873b58b"
ph_bin	[1] 1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1 [52] 0 1 1 0 1 1 1 1 1 0 0 0 0	[1] 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1 [52] 0 1 1 0 1 1 1 0 1 0 0 0 1	[1] 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1 [52] 0 1 1 0 1 1 1 0 1 0 0 0 1
Dhash method – do not use the gamma correction instead use the two processed images and compare them directly			
dh_hash	"46357717676566e8"	"46357717676566e0"	46357717676566e0

dh_bin	0 1 1 0 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 [52] 0 0 1 1 0 0 0 0 1 0 1 1 1	0 1 1 0 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 [52] 0 0 1 1 0 0 0 0 0 0 1 1 1	0 1 1 0 0 0 1 0 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 [52] 0 0 1 1 0 0 0 0 0 0 1 1 1

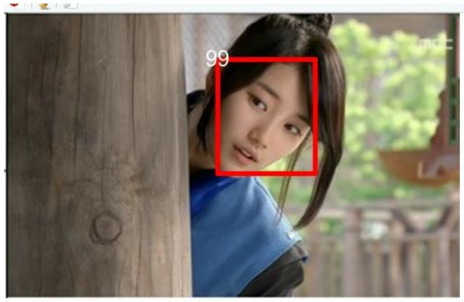
Invariant_hash – using the two processed images, identify the minimum and maximum values for the hamming or the levenshtein distance		
	inv_hash	inv_bin
Min	0.15625	9
Max	0.625	16

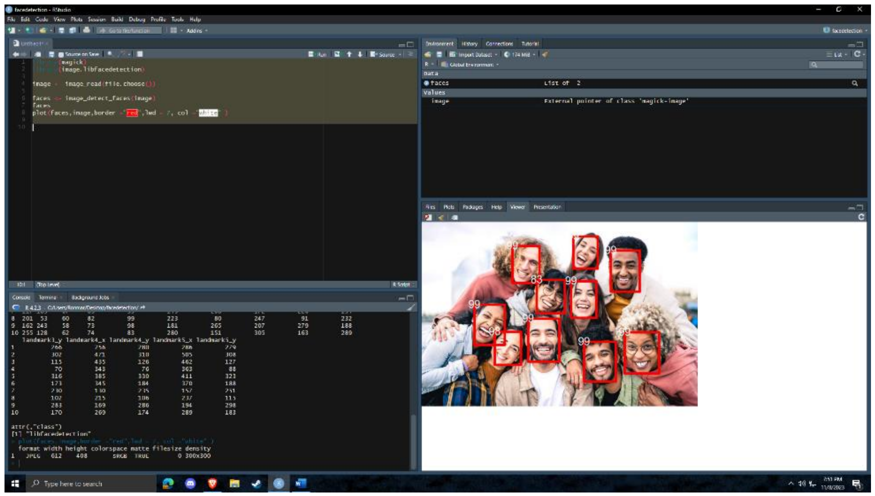
Which of the image hashing methods you think better can identify similarities of images? And why?
I think it's the PH hash and the Ph bin because they have a lot of different bits.

D. Face Detection

To explore face detection in R, you need to install the image.libfacedetection package which can be found in https://cran.r-project.org/web/packages/image.libfacedetection/index.html , select the appropriate download source based on your operating system. Try the given code using your preferred image with more than one face and save the image with detected faces as **LASTNAME_FaceDetection.**

```
#https://cran.r-project.org/web/packages/image.libfacedetection/index.html
library(magick)
library(image.libfacedetection)
image <- image_read(file.choose())
faces <- image_detect_faces(image)
faces
plot(faces, image, border = "red", lwd = 7, col = "white")
```





Deliverables:

- Module 2 Assessment Sheet with answer
- Miscellaneous files (code/s in r and processed images)

A.

```r
library(imager)
library(OpenImageR)
library(ggplot2)

img <- load.image(file.choose())
print(img)

positive_image <- 255 - img

save.image(positive_image, "SABADO_ProcessedImageA1.jpg.jpg")

summary(img)
summary(positive_image)

red_channel_pos <- positive_image[,,1]
green_channel_pos <- positive_image[,,2]
blue_channel_pos <- positive_image[,,3]


cloud_df_pos <- data.frame(
  Channel = rep(c("Red", "Green", "Blue"), each = prod(dim(positive_image)[1:2])),
  Value = c(as.vector(red_channel_pos), as.vector(green_channel_pos), as.vector(blue_channel_pos))
)


set.seed(42)
sample_size <- 10000
sampled_data <- cloud_df_pos[sample(nrow(cloud_df_pos), sample_size), ]


cloud_plot_pos <- ggplot(sampled_data, aes(x = Channel, y = Value, color = Channel)) +
  geom_point(alpha = 0.6) +
  labs(title = "RGB Cloud Plot (Positive Image)",
       x = "Channel",
       y = "Pixel Value") +
  theme_minimal()

print(cloud_plot_pos)

red_channel <- positive_image[,,1]
green_channel <- positive_image[,,2]
blue_channel <- positive_image[,,3]


boxplot_df <- data.frame(
  Channel = rep(c("Red", "Green", "Blue"), each = prod(dim(img)[1:2])),
  Value = c(as.vector(red_channel), as.vector(green_channel), as.vector(blue_channel))
)


plot <- ggplot(boxplot_df, aes(x = Channel, y = Value, fill = Channel)) +
  geom_boxplot() +
  labs(title = "RGB Boxplot",
       x = "Channel",
       y = "Pixel Value") +
  theme_minimal()

print(plot)
```

B.

```r
library(imager)
library(ggplot2)
library(gridExtra)
library(magick)

imgEq <- load.image(file.choose())

imgH.eq <- function(im)
  as.cimg(ecdf(im)(im),
          dim=dim(im))


imgHist <- iiply(imgEq,"c", imgH.eq)

layout(t(1:2))
plot(imgEq)
title(main = "Original Image")
plot(imgHist)
title(main = "Equalized Image")

img.plot <- as.data.frame(imgHist)
imgHH <- as.data.frame(imgEq)


ggplot_imgHH <- ggplot(imgHH, aes(value)) +
  geom_histogram(bins = 30) +
  labs(title = "Original Image Histogram") +
  facet_wrap(~ cc)

ggplot_img.plot <- ggplot(img.plot, aes(value)) +
  geom_histogram(bins = 30) +
  labs(title = "Histogram Equalized Image") +
  facet_wrap(~ cc)

grid.arrange(ggplot_imgHH, ggplot_img.plot, ncol = 2)

save.image(imgHist, "SABADO_ProcessedEqualizedImageB.jpg")
```

C.

```r
imgA <- readImage(file.choose())
imgB <- readImage(file.choose())

imgA1 = rgb_2gray(imgA)
imgB1 = rgb_2gray(imgB)

# PH HASH
ph_hashA01 = phash(imgA1, hash_size = 8, highfreq_factor = 4, MODE = 'hash', resize = "bilinear")
ph_hashB01 = phash(imgB1, hash_size = 8, highfreq_factor = 4, MODE = 'hash', resize = "bilinear")

ph_hashA01
ph_hashB01

# PH BIN
ph_binA01 = phash(imgA1, hash_size = 8, highfreq_factor = 4, MODE = 'binary', resize = "bilinear")
ph_binB01 = phash(imgB1, hash_size = 8, highfreq_factor = 4, MODE = 'binary', resize = "bilinear")

as.vector(ph_binA01)
as.vector(ph_binB01)

# DH HASH
dh_hashA01 = dhash(imgA1, hash_size = 8, MODE = 'hash', resize = "bilinear")
dh_hashB01 = dhash(imgB1, hash_size = 8, MODE = 'hash', resize = "bilinear")

dh_hashA01
dh_hashB01

# DH BIN
dh_binA01 = dhash(imgA1, hash_size = 8, MODE = 'binary', resize = "bilinear")
dh_binB01 = dhash(imgB1, hash_size = 8, MODE = 'binary', resize = "bilinear")

as.vector(dh_binA01)
as.vector(dh_binB01)

# INV HASH (MAX & MIN)
inv_hashAB01 = invariant_hash(imgA1, imgB1, mode = 'binary', flip = T, rotate = T,
                              angle_bidirectional = 10, crop = T)

inv_hashAB01

# INV BIN (MAX & MIN)
inv_binAB01 = invariant_hash(imgA1, imgB1, mode = 'hash', flip = T, rotate = T,
                             angle_bidirectional = 10, crop = T)
inv_binAB01
```

D.

```
library(image.libfacedetection)
image <- image_read(file.choose())
faces <- image_detect_faces(image)
faces
plot(faces,image,border ="red",lwd = 7, col ="white" )
```