MTH208: Worksheet 4

Images in R

Images are broken down into small square boxes, called pixels. Pixels in color images are made from the three colors: red, green, blue (rgb). Every pixel in the image can written as a combination of these three colors. Images in greyscale are **not** saved as rgb and rather as just one numeric value (indicating how much white there is).

We will use a package in `R` to study images. `R` packages are a collection of functions written to perform tasks. To install any package, the command is

```
install.packages("name of package in quotations")
```

We will use package `imager`. This package has already been installed in all your machine. Packages only needs to be installed once on a computer. After that, for any `R` session you only need to load the package using

```
library(imager) #now without quotes
```

Images in package `imager` are represented as 4D numeric arrays. The four dimensions are labelled x, y, z, c. The first two are the usual spatial dimensions, the third one will usually corresponds to depth or time (for gifs), and the fourth one is colour for each channel: red, blue, and green. Arrays are structured similar to matrices and follow the same rules of subsetting.

Let's first load, plot, and check dimension of an image.

First, make sure you download the all the images in the GitHub repository in your local machine, and change your working directory to the location of the images.

```
dog <- load.image("dog.jpeg")
dim(dog) # stored as RGB
plot(dog)  # plot image
```

You will see that the dimension of the image is $640 \times 635 \times 1 \times 3$. That means, there are 640×635 pixels in the image (640 columns and 635 rows), only one time (so it is not a gif) and 3 color channels, one for each primary color. We can also obtain the grayscale version of the image.

```
graydog <- grayscale(dog)
plot(graydog)
dim(graydog)
```

To extract the raw matrix or array of the image, we may do the following

```
# Extract the black and white image as matrix
gray.mat <- as.matrix(graydog[,,1,1])
dim(gray.mat)

# Extracts the array will all three rgb channels
col.mat <- as.array(dog[, ,1, ])
dim(col.mat)
```

We may now do manipulations on the arrays and then save the updated arrays as image

```
# Vertical cropping
cropped.mat <- col.mat[1:300, , ]
crop.dog <- as.cimg(cropped.mat)
plot(crop.dog)
```

Using all this information, and the basics of the `R` that you know, write code for the following tasks:

1. Find the "purest green" part of the image and mark that with a red point on the dog image. You may have to use the `which( …, arr.ind = TRUE)` command and the `points()` function.

2. Repeat the previous part for the "purest red" and "purest blue".

3. Images "col1.png", "col2.png", "col3.png" are images of the primary colors. Without opening the files on the computer, write an R program that guesses which file is which color.

4. Write an `R` function that takes a image as an input and outputs a prediction on whether the image has a lot of snow or not. Test this function on "land1.jpeg" and "land2.jpeg".

5. Write an `R` function that takes an `imager` image as input and outputs the `imager` image rotated by 180 degrees.

6. Write an `R` function that takes an `imager` image as input and outputs the `imager` image rotated clockwise by 90 degrees.

7. Write an `R` function that takes an `imager` image as input and outputs the `imager` image rotated anti-clockwise by 90 degrees.

8. Crop the image of the dog to a 600×600 pixel file. Write an algorithm to convert the image to a 300×300 pixel `imager` image. The reduced image should still have the complete dog. Save the `imager` image in a `jpeg` file using command `save.image()`. What is the size of this new file?

9. Repeat the above for making a 60×60 `imager` image of the dog.

10. The above is an example of a simple image compression. Can you think of other ways of compressing the image, using tools from linear algebra?