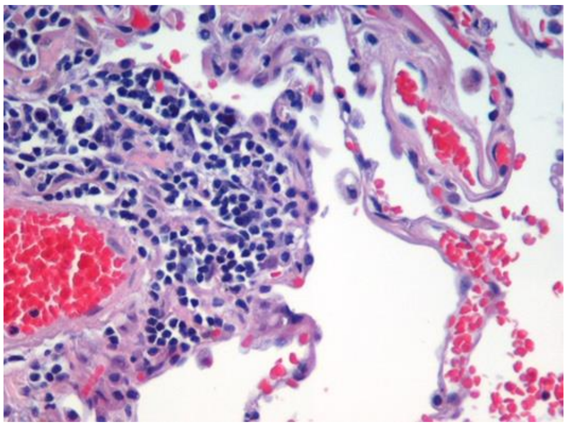
Image filter lab

Below you will find instructions for creating an image filter for Lab I. You can use any image you want, but make sure it has the proper file type for the image_read() function, i.e., jpeg (depending on your software other file types can work). You can also use the example file:



IMPORTANT! Make sure to try your code often to see if the last change worked. Also, use the help/documentation of functions as much as possible to understand the class, structure and which arguments the functions expect.

To separate the code, we want to create at least two function types, i) functions that takes an RGB (Red, Green, Blue) hexadecimal value and manipulates it in different ways, allowing the conversion to grey, red, green or blue, and ii) a function that goes pixel by pixel through a bitmap (image) and runs the previously created conversion function on that specific pixel to convert its colour. For this lab we will create a grey scale image, a red image, a green image and a blue image.

RGB – Colours are commonly represented by a three-element vector composed of the red, green and blue value. Each colour usually ranges from 0 to 255 (00 to ff in hexadecimal), with 0 being the lowest value of that colour and 255 being the highest.

Hexadecimal - Hexadecimal values use 0-15 instead of 0-9 vales (decimal system) and is most often used in computer science to represent a byte (256 possible values). Practically this results in an array of 0-0, 1-1, 2-2, 3-3, 4-4, 5-5-, 6-6, 7-7, 8-8, 9-9, 10-A, 11-B, 12-C, 13-D, 14-E, 15-F, making for example the value 255 = FF.

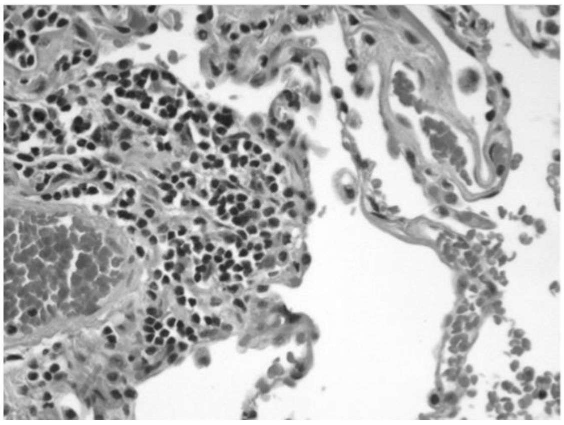
Important to note is that the object returned from the image_read() function provides the RGB as a raw() class of the first dimension of a 3-dimensional object, the other two dimensions are the x and y axis of the image.

1. To make it easier to import an image to R we use the package “magick”. Make sure to start the code by installing the package and loading the library.
2. We then use the image_read() function with the appropriate file path to load an image object to the environment. We need to convert this object to a bitmap image to be able to access the raw pixel values. From the help we find that one way of doing this is by subsetting the first element of the object we assigned from the image_read() function, e.g., my_bitmap <- my_object[[1]].
3. We can then display the image in the R plot window by using the image_read() function, to know we have succeeded.

4. To create separation in our code, we then create filter functions, one for the grey scale and one for colour cut-offs. The code should take a single raw() vector as input and return the greyscale value or cut-off value.
    - For the grey scale the ratio of red, green and blue that they eye interprets as grey is around: R * 0.299, G * 0.587 and B * 0.114. A colour of R = 50, G = 120 and B = 10, on a scale of 0-255 would therefore give 87 (14.95 + 70.44 + 1.14 ~ 87, or 57 in hexadecimals) for each of R, G and B.
    - For the cut-off we use the greyscale values and then choose any either hexadecimal or ratio as cut-off. If the input is below the value, the pixel is given one colour and if the input is above the value, it gets a different colour. It is recommended to set the default cut-off for the function argument to 50%, or 127 of 255.
5. We then create a function that takes the bitmap image and the filter function as arguments. This function should go pixel by pixel in the two-dimensional object and change the colour in accordance too the filter provided.
6. Use the image_read() function again to make sure you have successfully changes the look of the image.

When you are finished you should be able to create two different images, depending on what filter function you provide as argument to the change bitmap function.

The greyscale should look like this:



If you want to continue experimenting with the code you can create a function that applies a cut-off, for example, if the greyscale value is < 128 it is replaced by “#000000” and if it is higher it is replaced by “#FF00FF”, result below: