Making a cross-stitch pattern for an image

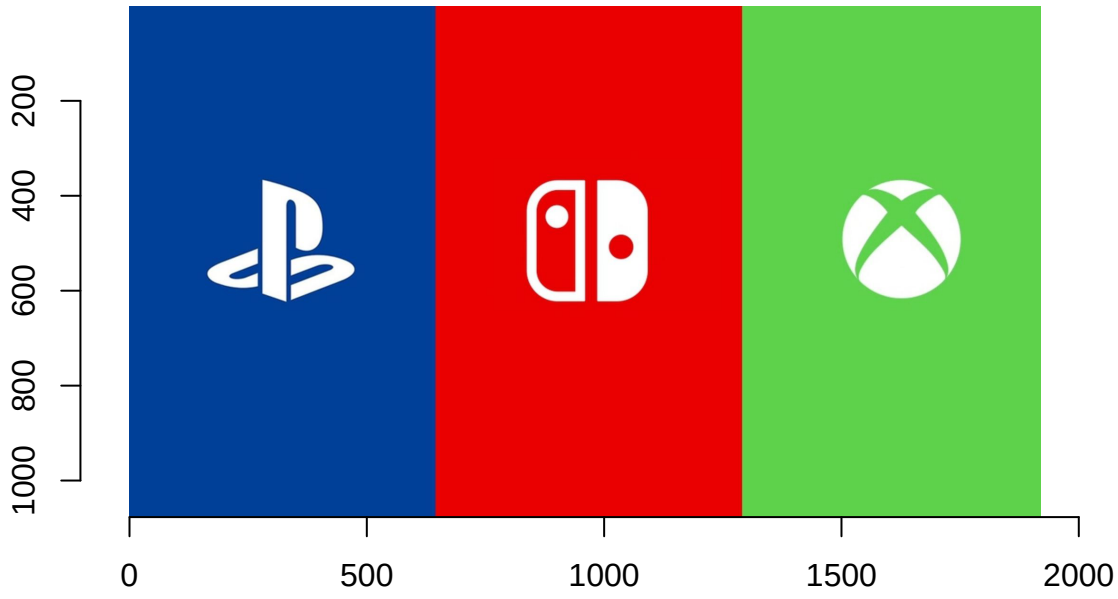
Shu Liu

Introduction

This vignette will provide a guideline to the functions: process_image(), color_strips(), scree_plot() and make_pattern(). These functions will help to make a cross-stitch pattern for a picture.

These functions will be explained in the following 4 sections, including their basic use and the explanations of the parameters. In this case, I will use a sample picture as an example to further illustrate. The picture is the collection of the logos of Playstation, Nintendo Switch and Xbox.

```
library(imager)

im <- 'C:/Users/12518/Desktop/logo.jpg'
image<-load.image(im)
plot(image)
```



Process_image

Basic usage

The process_image() function will process the image and generate a large list of the image informations.

Here we have a image, named 'logo.jpg'.

```
im <- 'C:/Users/12518/Desktop/logo.jpg'
```

Then we set a k list, it represents the numbers of centres in the clustering. Different k values will generate various clustering results.

Finally, process_image() is used to process information.

```
kl<-c(2:10)
cluster_info<-process_image(im,kl)
```

The output of process_info is a large list that contains two lists.

```
glimpse(cluster_info[[1]])
```

```
## Rows: 2,067,840
## Columns: 5
## $ x <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19...
## $ y <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ R <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ G <dbl> 0.2509804, 0.2509804, 0.2509804, 0.2509804, 0.2509804, 0.2509804,...
## $ B <dbl> 0.5960784, 0.5960784, 0.5960784, 0.5960784, 0.5960784, 0.5960784,...
```

The first list is the color informations of every point in the image. It consists of the coordinate as well as the RGB color mode of each point.

```
glimpse(cluster_info[[2]])
```

```
## Rows: 9
## Columns: 7
## $ k_list       <int> 2, 3, 4, 5, 6, 7, 8, 9, 10
## $ kclust       <list> [<2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ totss        <dbl> 737042.8, 737042.8, 737042.8, 737042.8, 737042.8, 7370...
## $ tot.withinss <dbl> 451876.3548, 80397.0884, 3839.3499, 2650.4372, 1936.96...
## $ betweenss    <dbl> 285166.5, 656645.8, 733203.5, 734392.4, 735105.9, 7354...
## $ iter         <int> 1, 2, 3, 3, 6, 4, 6, 5, 4
## $ tdata        <list> [<tbl_df[2 x 6]>, <tbl_df[3 x 6]>, <tbl_df[4 x 6]>, <...
```

The second list is the results of clustering at different k values. As previously setted, k list is from 2 to 10, then the list contains the clustering through 2 to 10, including their withinss, betweenss.

In this case, if you set a different k_list, the output will be varied.

```
k_list <- c(2:4)

ci<-process_image(im,k_list)

ci[[2]]
```

```
## # A tibble: 3 x 7
##   k_list kclust     totss tot.withinss betweenss  iter tdata
##    <int> <list>     <dbl>        <dbl>     <dbl> <int> <list>
## 1      2 <kmeans> 737043.      451876.   285166.     1 <tibble [2 x 6]>
## 2      3 <kmeans> 737043.      183906.   553137.     3 <tibble [3 x 6]>
## 3      4 <kmeans> 737043.        3839.   733203.     3 <tibble [4 x 6]>
```

Here we set k list to be 2-4, the output list will only contain the clustering information from 2 to 4.

Scree_plot

Basic usage

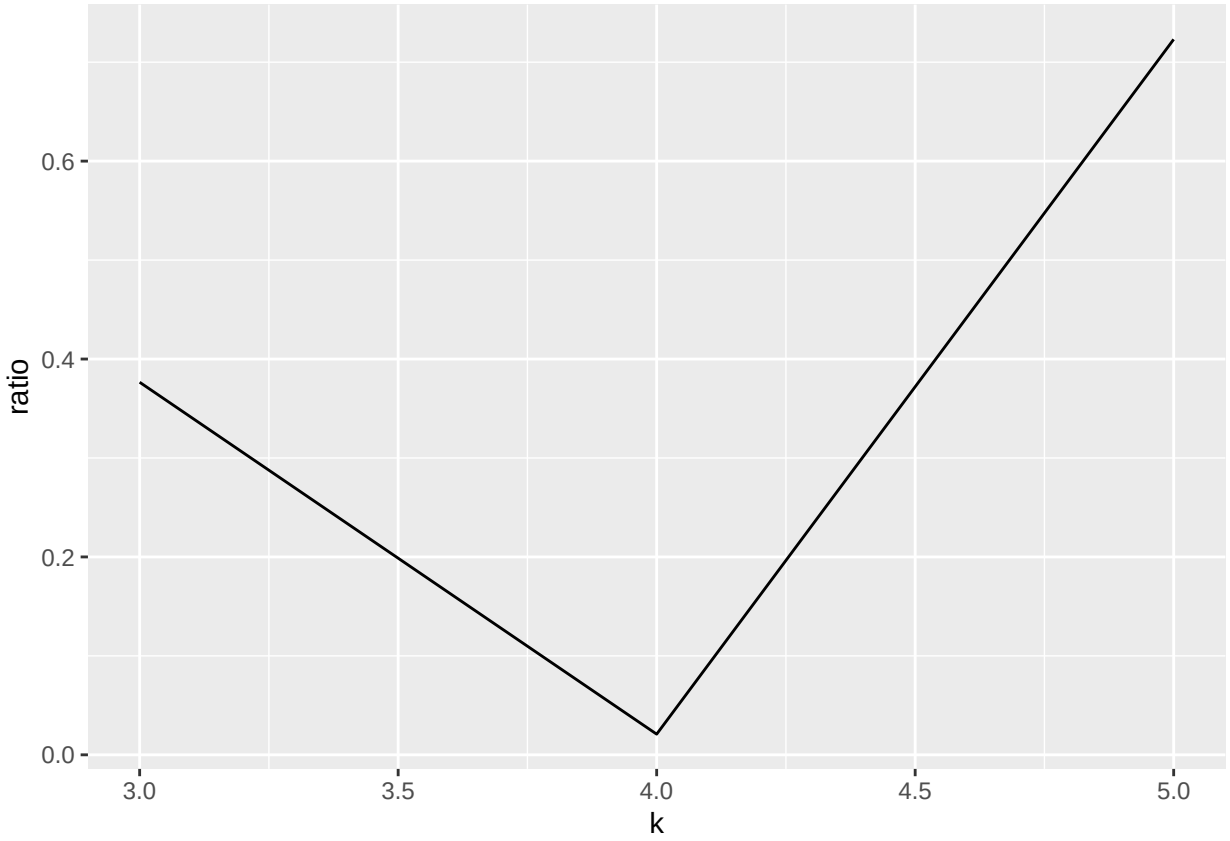
The scree_plot() function produces screeplot based on the output from process_image(). It will show the relationship between the ratio of total loss and k values. The goal of this function is to help finding the appropriate number of centers in the clustering.

Here we have an output from process_image(), named cluster_info

```
kl<-c(2:5)
cluster_info<-process_image(im,kl)
```

Then we use scree_plot() to generate a scree plot.

```
scree_plot(cluster_info)
```



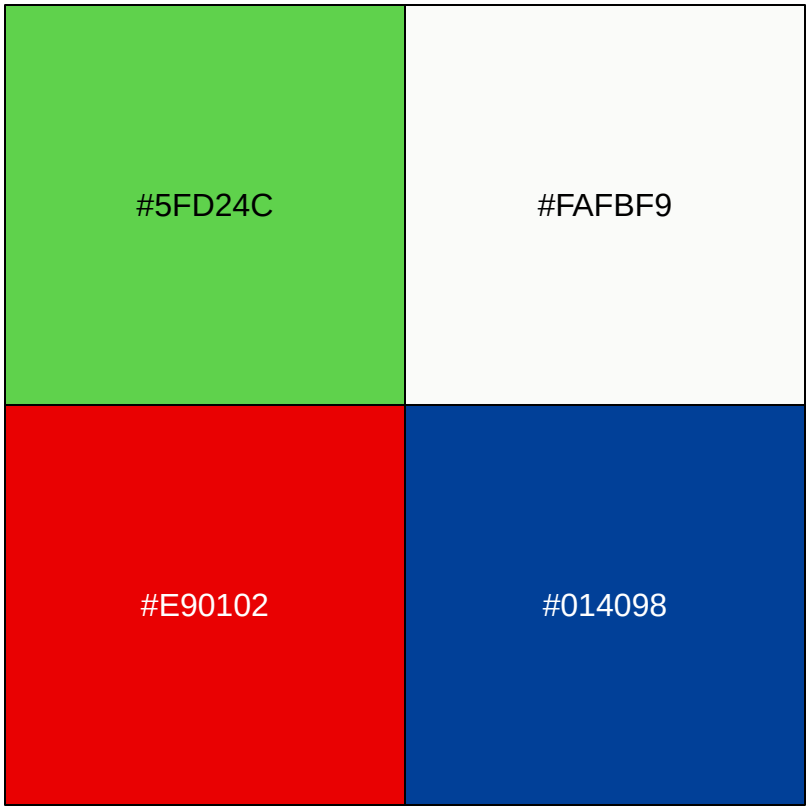
According to the scree plot, k=4 is an inflexion point. Hence in this example, we choose 4 as the number of cluster centers.

Color_strips

Basic usage

When k is determined, color_strips() can be used to obtain the RGB color information of centres. It will generate a strip with colors that represent each cluster centre and labelled by their hex number. In this case, as previously noticed, k=4.

```
k=4

color_strips(cluster_info,k)
```




```
## # A tibble: 4 x 7
##         R       G       B   size withinss cluster col
##     <dbl>   <dbl>   <dbl>  <int>    <dbl> <fct>   <chr>
## 1 0.371   0.822   0.296   642965     894. 1       #5FD24C
## 2 0.981   0.983   0.976   103379     959. 2       #FAFBF9
## 3 0.913   0.00379 0.00712 657615    1515. 3       #E90102
## 4 0.00243 0.251   0.595   663881     472. 4       #014098
```

There are two outputs, the first is a list that gathers the RGB color information of each cluster centre. The second is a color strip with k colors.

In this example we have k=4 colors with their hex numbers(e.g #E90102).

Make_pattern

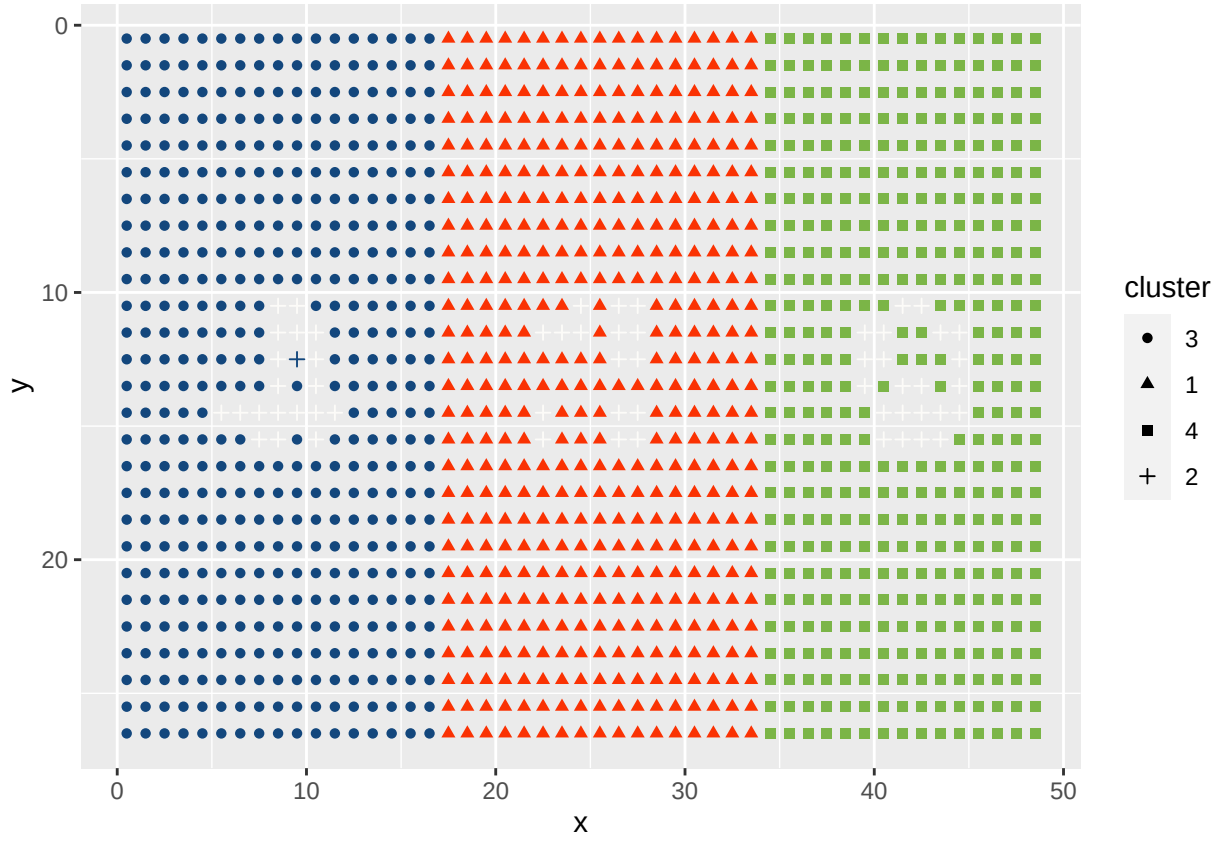
Basic usage

After choosing an appropriate k value, make_pattern() will produce the final cross-stitch pattern of the image. We should use the output from process_image(), determine k value, and choose an appropriate size of horizontal stitches.

As previously mentioned, k=4, cluster_info is the output from process_image(). As the image is large, so we firstly determine x_size to be 50 to see if it's proper.

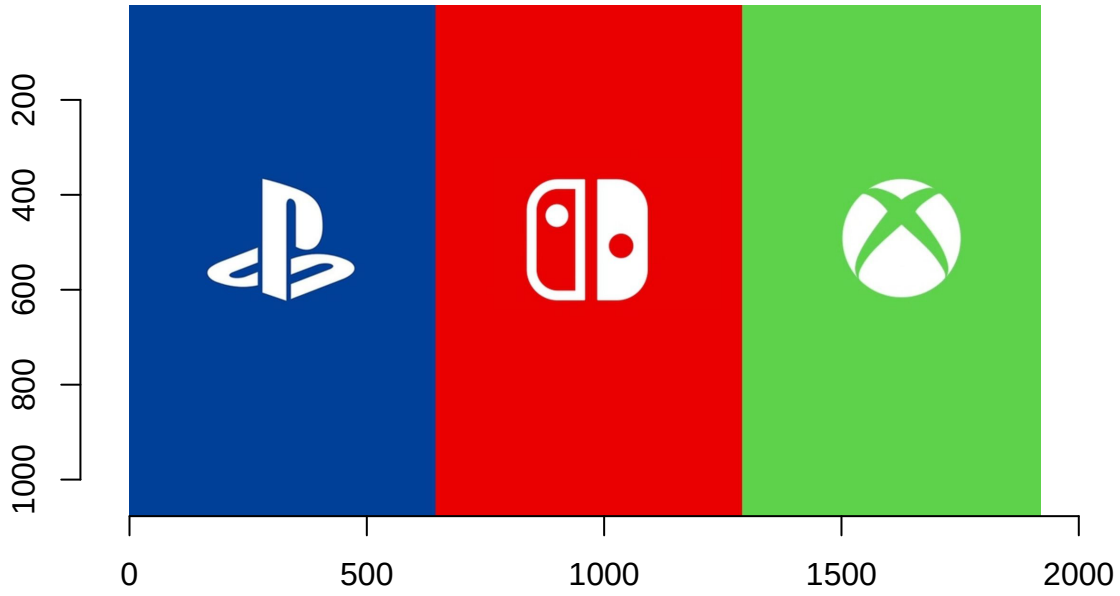
```
k=4

make_pattern(cluster_info,k,50,black_white = FALSE,background_colour = NULL)
```

```
## Warning: The ‘x‘ argument of ‘as_tibble.matrix()‘ must have unique column names if ‘.name_repair‘ is
## Using compatibility ‘.name_repair‘.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_warnings()‘ to see where this warning was generated.
```



Then we compare the cross-stitch pattern to its original image:

```
plot(image)
```

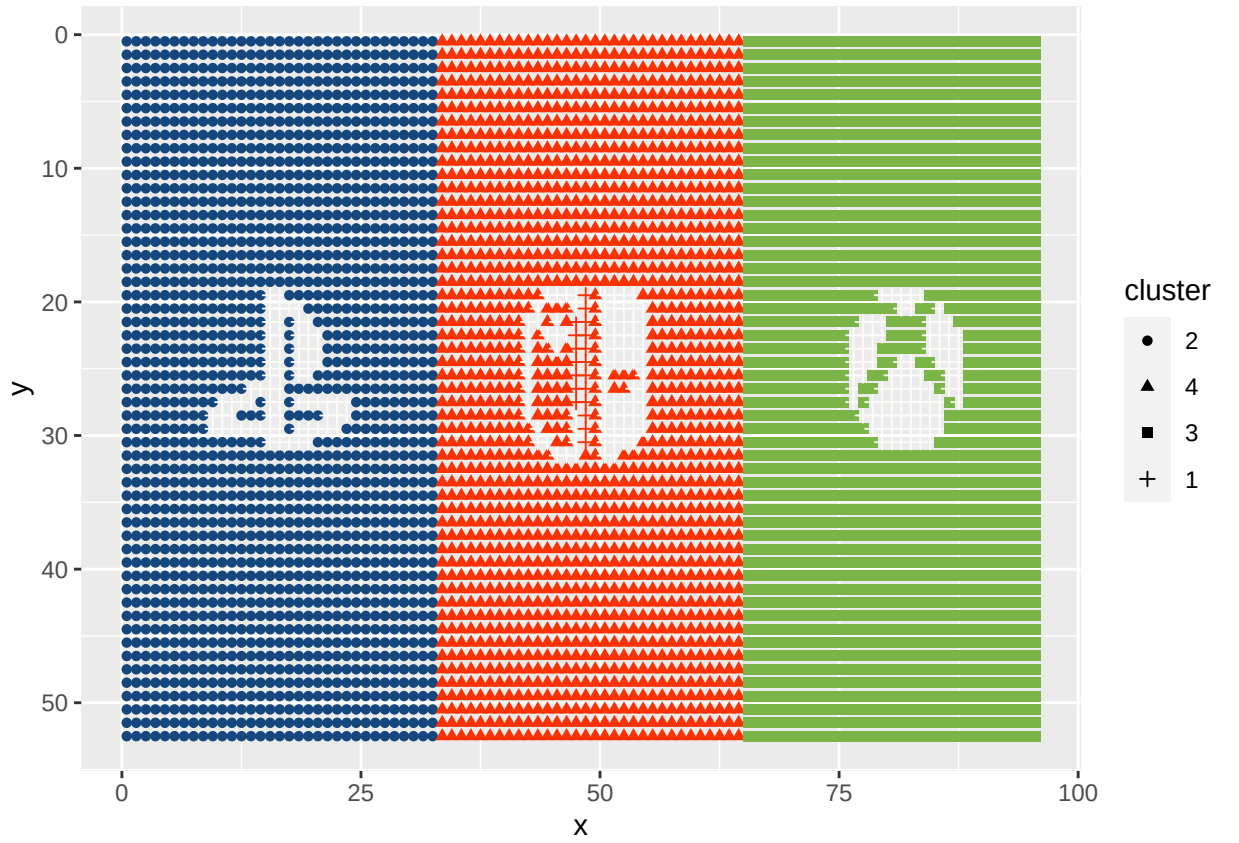


By the comparison, the cross-stitch pattern doesn't seem to be appropriate. That may be caused by the large image and small x_size value since horizontal stitches are not enough to cover all points.

The output may be varied with different x_size value, so if the output doesn't follow the expectation, we can change x_size value to adjust.

Therefore in this example, we can increase x_size from 50 to 100.

```
k=4

make_pattern(cluster_info,k,100,black_white = FALSE,background_colour = NULL)
```



At this time, the cross-stitch is very closed, so we needn't to revise our parameter further.

Reference

1.Simon Barthelme (2020). imager: Image Processing Library Based on 'CImg'. R package version 0.42.3. https://CRAN.R-project.org/package=imager

2.Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

3.Kuhn et al., (2020). Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles. https://www.tidymodels.org

4.Pebesma, E.J., R.S. Bivand, 2005. Classes and methods for spatial data in R. R News 5 (2), https://cran.r-project.org/doc/Rnews/.

5.Roger S. Bivand, Edzer Pebesma, Virgilio Gomez-Rubio, 2013. Applied spatial data analysis with R, Second edition. Springer, NY. https://asdar-book.org/

6.Hadley Wickham and Dana Seidel (2020). scales: Scale Functions for Visualization. R package version 1.1.1. https://CRAN.R-project.org/package=scales

7.Claus O. Wilke (2020). cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'. R package version 1.1.0. https://CRAN.R-project.org/package=cowplot

8.Sharla Gelfand (2020). dmc: Convert Colour to DMC Embroidery Floss and Back. R package version 0.0.0.9001.