**Superpixels**

“Introduced by Ren and Malik in 2003, superpixels group pixels similar in color and other low-level properties. In this respect, superpixels address two problems inherent to the processing of digital images: firstly, pixels are merely a result of discretization; and secondly, the high number of pixels in large images prevents many algorithms from being computationally feasible. Ren and Malik introduce superpixels as more natural entities – grouping pixels which perceptually belong together while heavily reducing the number of primitives for subsequent algorithms.”

The *superpixels()* function of the *OpenImageR* package is based on the open source C++ code of the [IMAGE AND VISUAL REPRESENTATION LAB IVRL](https://ivrl.epfl.ch/research-2/research-current/research-superpixels/) and allows users to receive *labels* and *output slic images*. The following code chunk illustrates how this can be done using either the “slic” or the “slico” method,

library(OpenImageR)

path = system.file("tmp\_images", "slic\_im.png", package = "OpenImageR")

im = readImage(path)

#--------------

# "slic" method

#--------------

res\_slic = superpixels(input\_image = im,

method = "slic",

superpixel = 200,

compactness = 20,

return\_slic\_data = TRUE,

return\_labels = TRUE,

write\_slic = "",

verbose = TRUE)

str(res\_slic)

List of 2

$ slic\_data: num [1:360, 1:360, 1:3] 255 255 255 255 255 255 255 255 255 255 ...

$ labels : num [1:360, 1:360] 0 0 0 0 0 0 0 0 0 0 ...

#---------------

# "slico" method [ the "slico" method does not require the "compactness" parameter ]

#---------------

res\_slico = superpixels(input\_image = im,

method = "slico",

superpixel = 200,

return\_slic\_data = TRUE,

return\_labels = TRUE,

write\_slic = "",

verbose = TRUE)

str(res\_slico)

List of 2

$ slic\_data: num [1:360, 1:360, 1:3] 255 255 255 255 255 255 255 255 255 255 ...

$ labels : num [1:360, 1:360] 0 0 0 0 0 0 0 0 0 0 ...

#-----------------------------

# plot both "slic" and "slico"

#-----------------------------

par(mfrow=c(1,2), mar = c(0.2, 0.2, 0.2, 0.2))

plot\_slic = OpenImageR::NormalizeObject(res\_slic$slic\_data)

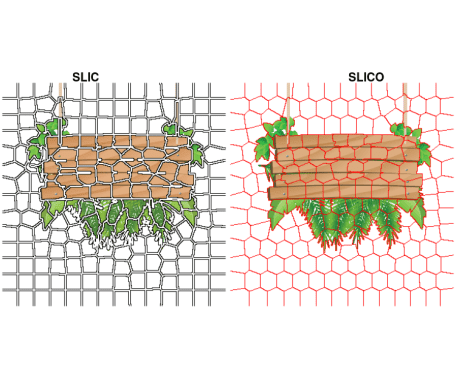
plot\_slic = grDevices::as.raster(plot\_slic)

graphics::plot(plot\_slic)

plot\_slico = OpenImageR::NormalizeObject(res\_slico$slic\_data)

plot\_slico = grDevices::as.raster(plot\_slico)

graphics::plot(plot\_slico)



“*SLIC uses the same compactness parameter (chosen by user) for all superpixels in the image. If the image is smooth in certain regions but highly textured in others, SLIC produces smooth regular-sized superpixels in the smooth regions and highly irregular superpixels in the textured regions. So, it becomes tricky choosing the right parameter for each image. SLICO does away with this problem completely. The user no longer has to set the compactness parameter or try different values of it. SLICO adaptively chooses the compactness parameter for each superpixel differently. This generates regular shaped superpixels in both textured and non textured regions alike. The improvement comes with hardly any compromise on the computational efficiency – SLICO continues to be as fast as SLIC*.”

More information about the parameters of the *superpixels()* function can be found in the package documentation.

**Image segmentation**

“In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.”

**Affinity Propagation**

“Affinity propagation is a new algorithm that takes as input measures of similarity between pairs of data points and simultaneously considers all data points as potential exemplars. Real-valued messages are exchanged between data points until a high-quality set of exemplars and corresponding clusters gradually emerges. We have used affinity propagation to solve a variety of clustering problems and we found that it uniformly found clusters with much lower error than those found by other methods, and it did so in less than one-hundredth the amount of time. Because of its simplicity, general applicability, and performance, we believe affinity propagation will prove to be of broad value in science and engineering.”

**Image segmentation based on superpixels (SLIC, SLICO) and Affinity Propagation (AP)**

The methodology is the following:

* First, the SLICAP technique uses the SLIC superpixel algorithm to form an over-segmentation of an image
* Then, a similarity is constructed based on the features of superpixels
* Finally, the AP algorithm clusters these superpixels with the similarities obtained

The advantages of this method compared to other methods are the following:

* similarities are not computed based on all data (image), but on the resulted superpixels (reduction of the computational burden)
* the user does not have to specify the number of clusters (AP algorithm) as is the case with other clustering methods such as Kmeans
* the *negative Euclidean distance* (although the author uses 3 different similarity measures (page 1526), I’ll use the first one in my code because it gives better results)
* *weights for the 3 colour channels* (wL, wA, wB) because they keep balance so as to be consistent with human perception
* a *colorradius* to adjust the number of clusters. If its value is low, the number of targets would increase, which leads to more detailed segmentation results

The following code snippet first reads the input image and then performs image segmentation based on SLIC superpixels and AP clustering,

library(SuperpixelImageSegmentation)

path = system.file("images", "BSR\_bsds500\_image.jpg", package = "SuperpixelImageSegmentation")

im = OpenImageR::readImage(path)



init = Image\_Segmentation$new()

spx = init$spixel\_segmentation(input\_image = im,

superpixel = 600,

AP\_data = TRUE,

use\_median = TRUE,

sim\_wL = 3,

sim\_wA = 10,

sim\_wB = 10,

sim\_color\_radius = 10,

verbose = TRUE)

Sequential computation starts ...

WARNING: The input data has values between 0.000000 and 1.000000. The image-data will be multiplied by the value: 255!

The super-pixel slic method as pre-processing step was used / completed!

The similarity matrix based on super-pixels was computed!

It took 179 iterations for affinity propagation to complete!

6 clusters were chosen based on super-pixels and affinity propagation!

Image data based on Affinity Propagation clustering ('AP\_image\_data') will be returned!

Elapsed time: 0 hours and 0 minutes and 2 seconds.

str(spx)

List of 4

$ KMeans\_image\_data: num[0 , 0 , 0 ]

$ masks : list()

..- attr(\*, "dim")= int [1:2] 0 0

$ centr : num[0 , 0 ]

$ AP\_image\_data : num [1:481, 1:321, 1:3] 1 1 1 1 1 1 1 1 1 1 ...

By using a *colorradius* of 10 (the author uses by default 20) I receive the following output,

OpenImageR::imageShow(spx$AP\_image\_data)



Decreasing the number of superpixels (from 600 to 200) or modifying the colorradius (from 10 to 20) will affect the quality of the output image, but it will also reduce the computation time.

**Superpixels, AP and Kmeans (or Mini-Batch-Kmeans)**

As a follow-up to the author’s method, one might take advantage of the automatic way the AP algorithm determines the number of clusters in order to perform vector quantization using kmeans (this will give qualitative better results at the cost of increasing computation time). I included additional parameters in the *spixel\_segmentation()* method of the *Image\_Segmentation* R6-class for this purpose and I’ll use the following image, which appears also in the article,

path\_km = system.file("images", "airplane.jpg", package = "SuperpixelImageSegmentation")

im\_km = OpenImageR::readImage(path\_km)

OpenImageR::imageShow(im\_km)



spx\_km = init$spixel\_segmentation(input\_image = im\_km,

superpixel = 600,

AP\_data = TRUE,

use\_median = TRUE,

sim\_wL = 3,

sim\_wA = 10,

sim\_wB = 10,

sim\_color\_radius = 10,

kmeans\_method = "kmeans",

kmeans\_initializer = "kmeans++",

kmeans\_num\_init = 3,

kmeans\_max\_iters = 100,

verbose = TRUE)

Sequential computation starts ...

WARNING: The input data has values between 0.000000 and 1.000000. The image-data will be multiplied by the value: 255!

The super-pixel slic method as pre-processing step was used / completed!

The similarity matrix based on super-pixels was computed!

It took 146 iterations for affinity propagation to complete!

10 clusters were chosen based on super-pixels and affinity propagation!

Image data based on Affinity Propagation clustering ('AP\_image\_data') will be returned!

The kmeans algorithm based on the number of affinity-propagation-clusters was completed!

Pre-processing of the kmeans-output-image is completed!

Elapsed time: 0 hours and 0 minutes and 8 seconds.

str(spx\_km)

List of 4

$ KMeans\_image\_data: num [1:321, 1:481, 1:3] 0.1 0.1 0.26 0.389 0.432 ...

$ masks :List of 10

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

..$ : num[0 , 0 , 0 ]

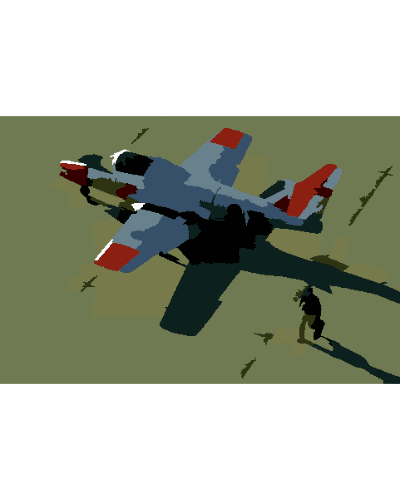
..- attr(\*, "dim")= int [1:2] 10 1

$ centr : num [1:10, 1:3] 0.432 0.251 0.863 0.552 0.26 ...

$ AP\_image\_data : num [1:321, 1:481, 1:3] 0.404 0.404 0.404 0.404 0.404 ...

The following image shows the output based on *Superpixels + AP*,

OpenImageR::imageShow(spx\_km$AP\_image\_data)



and the following based on *Superpixels + AP + Kmeans (vector quantization)*,

OpenImageR::imageShow(spx\_km$KMeans\_image\_data)



The *spixel\_segmentation()* takes as method parameter the *Mini-Batch-Kmeans* too, which can decrease the computation time at the cost of a reduced image quality. However, the advantage of using *Superpixels + AP + Kmeans (or Mini-Batch-Kmeans)* *lies in the fact that the user does not have to specify the number of clusters beforehand*,

spx\_mbkm = init$spixel\_segmentation(input\_image = im\_km,

superpixel = 600,

AP\_data = TRUE,

use\_median = TRUE,

sim\_wL = 3,

sim\_wA = 10,

sim\_wB = 10,

sim\_color\_radius = 10,

kmeans\_method = "mini\_batch\_kmeans",

kmeans\_initializer = "kmeans++",

kmeans\_num\_init = 3,

kmeans\_max\_iters = 100,

minib\_kmeans\_batch = 10,

minib\_kmeans\_init\_fraction = 0.75,

verbose = TRUE)

Sequential computation starts ...

WARNING: The input data has values between 0.000000 and 1.000000. The image-data will be multiplied by the value: 255!

The super-pixel slic method as pre-processing step was used / completed!

The similarity matrix based on super-pixels was computed!

It took 146 iterations for affinity propagation to complete!

10 clusters were chosen based on super-pixels and affinity propagation!

Image data based on Affinity Propagation clustering ('AP\_image\_data') will be returned!

The mini-batch-kmeans algorithm based on the number of affinity-propagation-clusters was completed!

Pre-processing of the mini-batch-kmeans-output-image is completed!

Elapsed time: 0 hours and 0 minutes and 3 seconds.

The following image is based on *Superpixels + AP + Mini-Batch-Kmeans*,

OpenImageR::imageShow(spx\_mbkm$KMeans\_image\_data)



Further parameters which can be adjusted in the *spixel\_segmentation()* method are:

* *method* (“slic”, “slico”)
* *colour\_type* (“RGB”, “LAB” or “HSV”)

**Image Masks**

The *spixel\_masks\_show()* method in combination with the *adjust\_centroids\_and\_return\_masks* parameter (set to TRUE), allows users to also plot the masks belonging to each separate cluster,

spx\_masks = init$spixel\_segmentation(input\_image = im\_km,

superpixel = 600,

AP\_data = TRUE,

use\_median = TRUE,

sim\_wL = 3,

sim\_wA = 10,

sim\_wB = 10,

sim\_color\_radius = 10,

kmeans\_method = "kmeans",

kmeans\_initializer = "kmeans++",

adjust\_centroids\_and\_return\_masks = TRUE,

verbose = TRUE)

Sequential computation starts ...

WARNING: The input data has values between 0.000000 and 1.000000. The image-data will be multiplied by the value: 255!

The super-pixel slic method as pre-processing step was used / completed!

The similarity matrix based on super-pixels was computed!

It took 146 iterations for affinity propagation to complete!

10 clusters were chosen based on super-pixels and affinity propagation!

Image data based on Affinity Propagation clustering ('AP\_image\_data') will be returned!

The kmeans algorithm based on the number of affinity-propagation-clusters was completed!

NOTE: The 'KMeans\_image\_data' will be returned in black & white colour due to the fact that the 'adjust\_centroids\_and\_return\_masks' parameter was set to TRUE!

Pre-processing of the kmeans-output-image is completed!

The centroids were adjusted and the image-masks will be returned!

Elapsed time: 0 hours and 0 minutes and 8 seconds.

str(spx\_masks)

List of 4

$ KMeans\_image\_data: num [1:321, 1:481, 1:3] 5 5 4 7 0 0 7 7 7 7 ...

$ masks :List of 10

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0.404 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0.294 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0.0784 0.1216 0 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0.392 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0 0 0 0 0 0 ...

..$ : num [1:321, 1:481, 1:3] 0 0 0 0 0 0 0 0 0 0 ...

..- attr(\*, "dim")= int [1:2] 10 1

$ centr : num [1:10, 1:3] 0 1 2 3 4 5 6 7 8 9 ...

$ AP\_image\_data : num [1:321, 1:481, 1:3] 0.404 0.404 0.404 0.404 0.404 ...

the *display\_all* parameter enables plotting of masks either all at once or sequentially,

init$spixel\_masks\_show(display\_all = TRUE)

