

# Introduction to SpecAI.Seg

```
library(SpecAI.Seg)
```

In this vignette, we will discuss the usage of this package for the purpose of hyperspectral image (HSI) segmentation. We will also demonstrate all the main function usage and options. This package can be found at <https://github.com/scoutiii/SpecAI.Seg>. Note that the gradient calculations can take several minutes to run depending on your computer speed.

## Introduction

---

Hyperspectral images (HSI) are very similar to color images but differ in the number of channels. Gray-scale images contain 1 channel of information, and color images contain 3 channels of information representing red, green, and blue. HSI on the other hand can have anywhere between 100 to 300 channels representing a whole range of different color wavelengths in the electromagnetic spectrum. In R, we simply store them as arrays, where the first two dimensions are the spatial dimensions, and the third dimension is the spectral dimension.

HSI contains a lot of information, but can be difficult to analyze on a per-pixel basis. We can use image segmentation to help summarize the spectral information, but also capture some spatial information. Image segmentation is the process of breaking an image into groups of similar pixels. It has been studied since the 1970s, so there are many image segmentation algorithms. For this package, we implement the Watershed segmentation algorithm, which has been extended to HSI. This package is based on the [SpecAI.Seg](#) package in Python, though with less functionality ([Jarman, 2022](#)).

The rest of this vignette will first show how we can download different standard HSI, and how we can make some basic plots to visualize these images. Next we will describe how to run the Watershed segmentation algorithm, along with the various parameters that can change the segmentation results. Lastly we will show how to make some basic plots of the segmentation results.

## Data Loading

---

Before segmenting or plotting the hyperspectral images, we must first download and clean the data. This package contains two functions to download and save the HSI data, `get_data()` and `get_all_data()`. The `get_all_data()` function takes no parameters and downloads each of the images in the `image_details` list. The `get_data()` function can have four parameters: `name`, `folder`, `verbose` and `clip_p`. `name` is a string argument for the name of the image to be downloaded and loaded and the only parameter that must be included in the function call. The possible datasets are “indianpines”, “salinas”, “paviau”, “paviac”, “ksc”, and “botswana”. `Folder` is a string argument for the name of the folder to save the data to, it has a default value of “./HSI\_Data/”. `Verbose` is a Boolean value that determines if messages are to be displayed as the function downloads the image data, its default is true. Lastly, `clip_p` is a number between zero and one that represents the percentile the image will be clipped to. This sets the outlier values to the `clip_p` percentile given, for example any value above the 99.75th percentile will be set to be equal to the 99.75th percentile, similarly if a value is below .25th percentile. The `get_data()` function saves the image data and outputs an `HSI_data` class object that can be assigned to a variable as shown below where we load in the Indian Pines image.

```
ip <- get_data("indianpines")
#> Reading in pre-downloaded data...
```

## Plotting and Summaries

One can plot the false coloring of the image by simply using the `plot` function. Additionally, one can summarize the dataset features by using the `summary` function.

```
summary(ip)
#> The Data name is IndianPines
#> This image is 145 x 145 pixels with 200 Layers.
#> The RGB bands are: 44 22 12
#> The gt Levels are:
#> [1] "0 Undefined"           "1 Alfalfa"
#> [3] "2 Corn-notill"          "3 Corn-mintill"
#> [5] "4 Corn"                 "5 Grass-pasture"
#> [7] "6 Grass-trees"          "7 Grass-pasture-mowed"
#> [9] "8 Hay-windrowed"        "9 Oats"
#> [11] "10 Soybean-notill"      "11 Soybean-mintill"
#> [13] "12 Soybean-cClean"      "13 Wheat"
#> [15] "14 Woods"              "15 Buildings-Grass-Trees-Drives"
#> [17] "16 Stone-Steel-Towers"
plot(ip)
```



## Watershed Segmentation

In order to use the Watershed segmentation algorithm, we need to find the gradient for the image. We calculate the Robust Color Morphological Gradient as defined in (Tarabalka *et al.*, 2010). First, for every pixel  $X_p$ , find the set of  $e$  neighboring pixels  $\chi = \{X_p^1, X_p^2, \dots, X_p^e\}$  where  $X_p \in \chi$ . Let  $\chi^r \subset \chi$ , such that the  $r$  pairs of pixels with the largest distance have been removed. Second, find the RCMG as

$\nabla_{\chi^r, d}^{RCM} = \max_{i, j \in \chi^r} \{d(X_p^i, X_p^j)\}$ , where  $d$  is an appropriate distance function. The original paper suggests

using the Euclidean distance function, however, there are other options that may be better for HSI, such as Cosine Distance, Spectral Angle Mapper, Euclidean Cumulative Sum, Kullback-Leibler Pseudo Divergence, and others (Deborah, 2016).

For this package, we implemented a `rcmg_euclidean` and `rcmg_cos` gradient calculation which implements the euclidean and cosine distances respectively. Each is implemented in Rcpp armadillo to help reduce computation time. It is recommend to use `calc_grad` function for error checking. For example, we can calculate the RCMG using Euclidean distance with:

```
grad_e <- calc_grad(ip)
plot(grad_e, log = TRUE)
```



Additionally, we can use the cosine distance function, which produces a slightly different gradient:

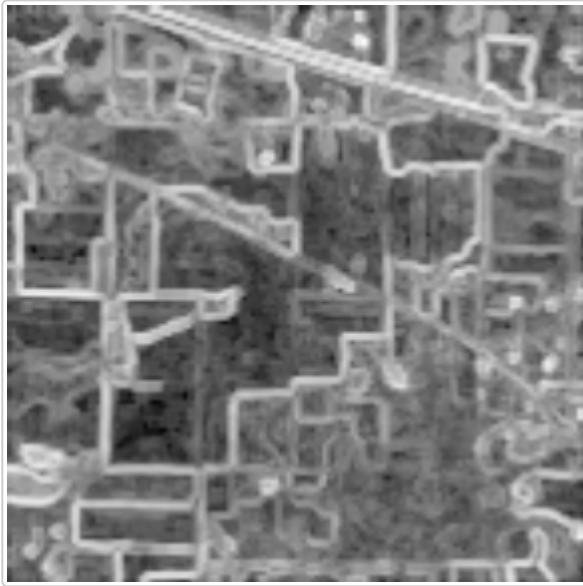
```
grad_c <- calc_grad(ip, "cos")
plot(grad_c, log = TRUE)
```



Another parameter that can be tuned is  $r$ , which is the number of pairs of pixels to remove. This will create overall smaller gradient values when you use a larger  $r$ . For example, here is the Euclidean RCMG using

$r = 2$ :

```
grad_2 <- calc_grad(ip, "cos", 2)
plot(grad_2, log = TRUE)
```



One parameter that we don't allow to change is the window size. Our implementation sets the window size to be 1, which corresponds to finding the 8 neighboring pixels.

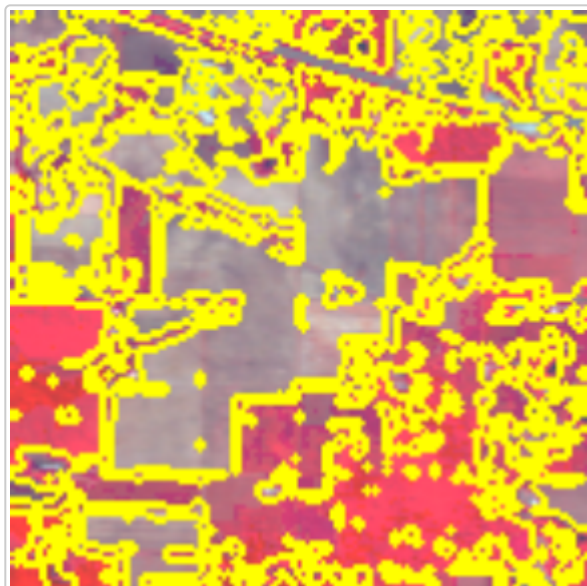
Once the gradient of the HSI has been calculated, `watershed_hsi` allows users to segment that gradient into distinct pieces using the watershed algorithm.

```
seg <- watershed_hsi(grad_e, tolerance = 1, ext = 200)
```

`watershed_hsi` takes three arguments, a gradient (or object of class `HSI_data`), a tolerance parameter, and an `ext` parameter. The gradient will typically be what is returned by `calc_grad`. The tolerance parameter is dependent on the scale of your gradient, and will combine objects with a height difference that is less than tolerance. The `ext` parameter functions as a way to smooth out the segmented image. This implementation of the watershed algorithm also minimizes gradient noise by ignoring the lower quantiles of said gradient. `watershed_hsi` will return an object of class `HSI_seg`, which contains the segmented image, and the gradient used as an attribute of the object.

```
marked_img <- mark_boundaries(seg, ip$img_rgb, c(1, 1, 0))

ggmap::ggimage(marked_img)
```



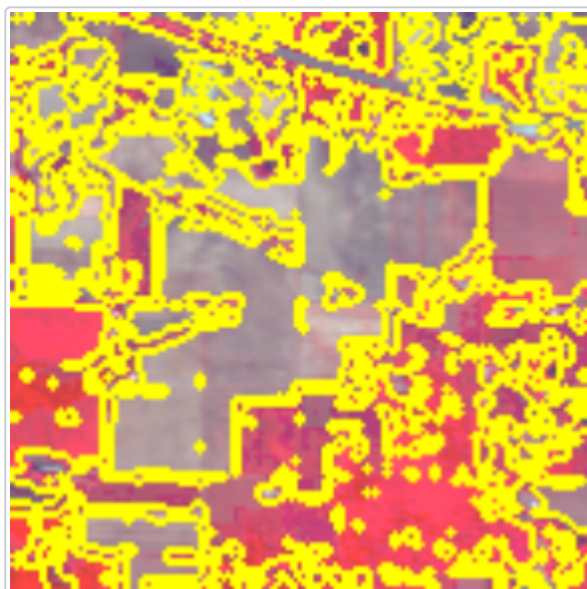
`mark_boundaries` takes the segmented gradient, an RGB version of the image, and an RGB color vector. It uses erosion and dilation to mark the RGB image along the boundaries of the segmented gradient. The color vector can be changed freely, but has a default of yellow (maximum red and green). This function will typically be used automatically when plotting an object of class `HSI_seg`.

## Plotting and Summaries

---

The segmentation results can again be easily plotted using `plot`, with the segmentation and the dataset as arguments:

```
plot(seg, ip)
```



## References

---

Deborah H. (2016). Towards spectral mathematical morphology. Université de Poitiers; Norwegian University of science and technology ....

Jarman S. (2022). SpecAI.seg. <https://pypi.org/project/SpecAI.Seg/>.

Tarabalka Y., Chanussot J. & Benediktsson J.A. (2010). Segmentation and classification of hyperspectral images using watershed transformation. Pattern Recognition 43 (7): 2367–2379.