**Seagrass Python Library**

Coffer, M, D. Graybill, P. Whitman, B. Schaeffer, W. Salls, R. Zimmerman, V. Hill, M. Lebrasse, J. Li, D. Keith, J. Kaldy, P. Colarusso, G. Raulerson, D. Ward, and W. Kenworthy

**Functions documented:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| create\_directory. | | | | | | | | | | | | | ......….………………………………………………………………………..1 | | | | | | | | | | |
| recursive\_search | | | | | | | | | | | | | ...……………………………………………………………………………..2 | | | | | | | | | | |
| extract\_metadata. | | | | | | | | | | | | | | ..…………………………………………………………………………….2 | | | | | | | | | |
| check\_extent\_coord | | | | | | | | | | | | | | | | | | .….……………………………………………………………………….3 | | | | | |
| coord\_to\_extent\_polygon. | | | | | | | | | | | | | | | | | | | | | | | .……………………………………………………………………4 |
| coord\_to\_rpc\_image | | | | | | | | | | | | | | | | | | | .....……………………………………………………………………….4 | | | | |
| coord\_to\_proj\_image. | | | | | | | | | | | | | | | | | | | | .......……………………………………………………………………5 | | | |
| world\_to\_pixel. | | | | | | | | | | | | ...……………………………………………………………………………...6 | | | | | | | | | | | |
| mask\_extent. | | | | | | | ...………………………………………………………………………………...7 | | | | | | | | | | | | | | | | |
| parse\_date\_time | | | | | | | | | | | | | ..……………………………………………………………………………...7 | | | | | | | | | | |
| earth\_sun\_distance. | | | | | | | | | | | | | | | | | .…………………………………………………………………………...8 | | | | | | |
| worldview\_gain | | | | | | | | | | | | | ...……………………………………………………………………………..9 | | | | | | | | | | |
| worldview\_offset | | | | | | | | | | | | | | ……………………………………………………………………………...9 | | | | | | | | | |
| worldview\_eai | | | | | | | | | | .………………………………………………………………………………10 | | | | | | | | | | | | | |
| worldview\_band\_center. | | | | | | | | | | | | | | | | | | | | | | …………………………………………………………………….11 | |
| list\_files . | | | ……………………………………………………………………………………..11 | | | | | | | | | | | | | | | | | | | | |
| unzip\_tiles | | | | .…………………………………………………………………………………...12 | | | | | | | | | | | | | | | | | | | |
| clip\_image. | | | | | ....………………………………………………………………………………..13 | | | | | | | | | | | | | | | | | | |
| project\_image | | | | | | | | | .……………………………………………………………………………….14 | | | | | | | | | | | | | | |
| rad\_cal. | | ....……………………………………………………………………………………14 | | | | | | | | | | | | | | | | | | | | | |
| embed\_dos\_value | | | | | | | | | | | | | | | .…………………………………………………………………………...15 | | | | | | | | |
| min\_dos\_value | | | | | | | | | | | ………………………………………………………………………………16 | | | | | | | | | | | | |
| atm\_cor | | ………………………………………………………………………………………17 | | | | | | | | | | | | | | | | | | | | | |
| mosaic | .……………………………………………………………………………………….18 | | | | | | | | | | | | | | | | | | | | | | |
| multipart\_shp | | | | | | | | ………………………………………………………………………………..18 | | | | | | | | | | | | | | | |
| multipart\_to\_singlepart | | | | | | | | | | | | | | | | | | | | | ……………………………………………………………………...19 | | |
| shp\_to\_roi | | | | .…………………………………………………………………………………...20 | | | | | | | | | | | | | | | | | | | |
| dcnn\_model | | | | | | .…………………………………………………………………………………21 | | | | | | | | | | | | | | | | | |
| train\_dcnn | | | | .…………………………………………………………………………………...21 | | | | | | | | | | | | | | | | | | | |
| dcnn\_classification | | | | | | | | | | | | | | | | ..…………………………………………………………………………23 | | | | | | | |

|  |  |
| --- | --- |
| create\_directory | *Create a directory* |

**Description**

Create a specified directory if it doesn't exist

**Usage**

create\_directory(fp)

**Arguments**

|  |  |
| --- | --- |
| fp | String object specifying a directory path |

**Value**

None

**Details**

This will not delete the directory if it already exists

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

output\_fp = os.path.join(wdir, "output\_folder", "output\_image.TIF")

create\_directory(os.path.dirname(output\_fp))

|  |  |
| --- | --- |
| recursive\_search | *Recursive file search* |

**Description**

Recurvsively seach for files within a directory

**Usage**

recursive\_search(input\_dir, pattern = "\*.TIF")

**Arguments**

|  |  |
| --- | --- |
| input\_dir | String object specifying a directory path |
| pattern = “.\*TIF” | String defining the search pattern. By default the function looks for files with the TIF file extension |

**Value**

List object containing filepath strings that match the specified pattern

**Details**

**Example**

wdir = "C:\\Users\\JDoe"

unzipped\_images = recursive\_search(input\_dir = wdir)

|  |  |
| --- | --- |
| extract\_metadata | *Extract data from a metadata file* |

**Description**

Extract a specified instance of a metadata variable from a parsed XML file

**Usage**

extract\_metadata(metadata\_file, metadata\_var, instance = 0)

**Arguments**

|  |  |
| --- | --- |
| metadata\_file | Document object representing the content of a XML file |
| metadata\_var | String object defining the variable to look for within metadata\_file |
| instance = 0 | Int defining which instance of the variable should be returned. Default is set to 0 |

**Value**

String object

**Details**

Only one instance of the variable can be extracted at a time

**Example**

wdir = "C:\\Users\\JDoe"

image\_fp = os.path.join(wdir, "input\_image.TIF")

metadata\_fp = image\_fp.replace(".TIF", ".XML")

metadata = minidom.parse(metadata\_fp) # Parse metadata .XML file

solar\_elevation = extract\_metadata(metadata\_file = metadata, metadata\_var = 'MEANSUNEL', instance = 0) # Get solar elevation from the metadata (in degrees)

|  |  |
| --- | --- |
| check\_extent\_coord | *Check if coordinates have been supplied correctly* |

**Description**

Basic check to determine if the correct number of coordinates have been supplied to define an extent

**Usage**

check\_extent\_coord(ul\_lon, ul\_lat, lr\_lon, lr\_lat)

**Arguments**

|  |  |
| --- | --- |
| ul\_lon | Upper left longitude in decimal degrees |
| ul\_lat | Upper left latitude in decimal degrees |
| lr\_lon | Lower right longitude in decimal degrees |
| lr\_lat | Lower right latitude in decimal degrees |

**Value**

None

**Details**

This function only checks if four coordinates have been supplied

**Example**

extent\_coord = [-76.7, 34.7, -76.5, 34.5]

check\_extent\_coord(\*extent\_coord)

ul\_lon, ul\_lat, lr\_lon, lr\_lat = [-76.7, 34.7, -76.5, 34.5]

check\_extent\_coord(ul\_lon, ul\_lat, lr\_lon, lr\_lat)

|  |  |
| --- | --- |
| coord\_to\_extent\_polygon | *Convert extent coordinates to polygon* |

**Description**

Convert upper left and lower right longitude and latitude coordinates to a polygon

**Usage**

coord\_to\_extent\_polygon(extent\_coord)

**Arguments**

|  |  |
| --- | --- |
| extent\_coord | List object containing the upper left and lower right longitude and latitude of an area-of-interest in decimal degrees. It should be ordered upper left longitude, upper left latitude, lower right longitude, lower right latitude |

**Value**

Polygon object

**Details**

Output polygon will use latitude/longitude

**Example**

extent\_coord = [-76.7, 34.7, -76.5, 34.5] # set the aoi extent with ul\_lon, ul\_lat, lr\_lon, lr\_lat (decimal degrees)

aoi\_polygon = coord\_to\_extent\_polygon(extent\_coord = aoi\_extent)

|  |  |
| --- | --- |
| coord\_to\_rpc\_image | *Convert latitude, longitude to row, column for an image that uses RPCs* |

**Description**

Convert latitude, longitude to row, column for an image that is georeferenced using rational polynomial coefficients (RPCs)

**Usage**

coord\_to\_rpc\_image(lon, lat, rpc\_coeff, height = 0)

**Arguments**

|  |  |
| --- | --- |
| lon | Longitude in decimal degrees |
| lat | Latitude in decimal degrees |
| rpc\_coeff | A dictionary object retrieved from image metadata containing rational polynomial coefficients |
| height = 0 | Height in meters. Default is zero |

**Value**

List object containing column and row positions ([col, row])

**Details**

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF")

ds = gdal.Open(input\_image)

extent\_coord = [-76.7, 34.7, -76.5, 34.5]

ul\_lon, ul\_lat, lr\_lon, lr\_lat = extent\_coord

ul\_col, ul\_row = coord\_to\_rpc\_image(lat = ul\_lat, lon = ul\_lon, height = 0, rpc\_coeff = ds.GetMetadata('RPC'))

lr\_col, lr\_row = coord\_to\_rpc\_image(lat = lr\_lat, lon = lr\_lon, height = 0, rpc\_coeff = ds.GetMetadata('RPC'))

|  |  |
| --- | --- |
| coord\_to\_proj\_image | *Convert latitude, longitude to x, y coordinates in an image's coordinate system* |

**Description**

Convert latitude, longitude to x, y coordinates in an image's coordinate system

**Usage**

coord\_to\_proj\_image(lon, lat, projection)

**Arguments**

|  |  |
| --- | --- |
| lon | Longitude in decimal degrees |
| lat | Latitude in decimal degrees |
| projection | Projeciton string retrieved from image metadata |

**Value**

List object containing x, y coordinates ([x, y])

**Details**

Projection string must be in proj4 or wkt format. It can be extracted from an input image using GDAL

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF")

ds = gdal.Open(input\_image)

extent\_coord = [-76.7, 34.7, -76.5, 34.5]

ul\_lon, ul\_lat, lr\_lon, lr\_lat = extent\_coord

ul\_x, ul\_y = coord\_to\_proj\_image(lon = ul\_lon, lat = ul\_lat, projection = ds.GetProjection())

lr\_x, lr\_y = coord\_to\_proj\_image(lon = lr\_lon, lat = lr\_lat, projection = ds.GetProjection())

|  |  |
| --- | --- |
| world\_to\_pixel | *Convert x, y coordinates to column and row position* |

**Description**

Convert x,y coordinates in a projected image's coordinate system to column and row position

**Usage**

world\_to\_pixel(x, y, geotransform)

**Arguments**

|  |  |
| --- | --- |
| x | x coordinate within image coordinate system |
| y | y coordinate within image coordinate system |
| geotransform | Geotransform object retrieved from image metadata |

**Value**

List object containing column and row positions ([col, row])

**Details**

Geotransform object can be extracted from the metadata of an input image using GDAL. It should contain six elements. GT(2) and GT(4) coefficients are sheer and rotation terms. GT(1) is pixel width and GT(5) is pixel height. GT(0) is the x coordinate of the upper left pixel and GT(3) is the y coordinate of the upper left pixel.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF")

ds = gdal.Open(input\_image)

extent\_coord = [-76.7, 34.7, -76.5, 34.5]

ul\_lon, ul\_lat, lr\_lon, lr\_lat = extent\_coord

ul\_x, ul\_y = coord\_to\_proj\_image(lon = ul\_lon, lat = ul\_lat, projection = ds.GetProjection())

lr\_x, lr\_y = coord\_to\_proj\_image(lon = lr\_lon, lat = lr\_lat, projection = ds.GetProjection())

ul\_col, ul\_row = world\_to\_pixel(x = ul\_x, y = ul\_y, geotransform = ds.GetGeoTransform())

lr\_col, lr\_row = world\_to\_pixel(x = lr\_x, y = lr\_y, geotransform = ds.GetGeoTransform())

|  |  |
| --- | --- |
| mask\_extent | *Compute a mask’s column and row extent* |

**Description**

Compute a mask’s column and row extent within an image

**Usage**

mask\_extent(ds, aoi\_extent)

**Arguments**

|  |  |
| --- | --- |
| ds | GDAL image object |
| aoi\_extent | List object containing the upper left and lower right longitude and latitude of an area-of-interest in decimal degrees. It should be ordered upper left longitude, upper left latitude, lower right longitude, lower right latitude |

**Value**

List object containing column and row positions of the upper left and lower right extent ([ul\_col, ul\_row, lr\_col, lr\_row])

**Details**

The function will return [None, None, None, None] if the specified extent coordinates don’t overlap the input image. If the input image and extent coordinates do overlap, the mask extent will only encompass the overlapping area.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF")

ds = gdal.Open(input\_image)

extent\_coord = [-76.7, 34.7, -76.5, 34.5]

mask\_pos = mask\_extent(ds, extent\_coord)

ul\_col, ul\_row, lr\_col, lr\_row = mask\_pos

|  |  |
| --- | --- |
| parse\_date\_time | *Parse a date-time string* |

**Description**

Parse a date-time stringretrieved from a WorldView metadata file

**Usage**

parse\_date\_time(date\_time\_string)

**Arguments**

|  |  |
| --- | --- |
| date\_time\_string | String object giving date and time with the following format:  [YYYY-MM-DDTHH:MM:SS.SSSSSS] |

**Value**

Struct\_time object

**Details**

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

metadata\_fp = os.path.join(wdir, "image\_metadata.XML")

metadata = minidom.parse(metadata\_fp) # Parse metadata .XML file

aquisition\_date\_time = extract\_metadata(metadata\_file = metadata, metadata\_var = 'TLCTIME', instance = 0) # Get aquisition date-time string from the metadata

parsed = parse\_date\_time(aquisition\_date\_time)

|  |  |
| --- | --- |
| earth\_sun\_distance | *Convert struct\_time to sun-earth distance* |

**Description**

Calculate sun-earth distance in astronomical units from a parsed worldview date-time string

**Usage**

earth\_sun\_distance(parsed\_date\_time)

**Arguments**

|  |  |
| --- | --- |
| parsed\_date\_time | Struct\_time object |

**Value**

Float object

**Details**

The Earth-Sun distance is returned in Astronomical Units (AU) and should have a value between 0.983 and 1.017. At least six decimal places should be carried in the Earth-Sun distance for use in radiometric balancing or top-ofatmosphere reflectance calculations (Updike and Comp 2010; Kuester 2016).

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

metadata\_fp = os.path.join(wdir, "image\_metadata.XML")

metadata = minidom.parse(metadata\_fp) # Parse metadata .XML file

aquisition\_date\_time = extract\_metadata(metadata\_file = metadata, metadata\_var = 'TLCTIME', instance = 0) # Get aquisition date-time string from the metadata

parsed\_acquisition = parse\_date\_time(aquisition\_date\_time)

dist\_au = earth\_sun\_distance(parsed\_acquisition)

|  |  |
| --- | --- |
| worldview\_gain | *Gain values* |

**Description**

Gain values for WorldView-2 & Worldview-3 sensors

**Usage**

worldview\_gain(sat\_id, band\_num)

**Arguments**

|  |  |
| --- | --- |
| sat\_id | String object specifying the satellite id. Either ‘WV02’ or WV03’ |
| band\_num | Band number. Indexing starts from zero |

**Value**

Float object

**Details**

|  |  |  |
| --- | --- | --- |
| **Band** | **WorldView-2** | **WorldView-3** |
| 0 | 1.151 | 0.905 |
| 1 | 0.988 | 0.940 |
| 2 | 0.936 | 0.938 |
| 3 | 0.949 | 0.962 |
| 4 | 0.952 | 0.964 |
| 5 | 0.974 | 1.000 |
| 6 | 0.961 | 0.961 |
| 7 | 1.002 | 0.978 |

See Kuester (2017) for more information

**Example**

satellite = ‘WV02’

gain = worldview\_gain(sat\_id = satellite, band\_num = 0)

|  |  |
| --- | --- |
| worldview\_offset | *Offset values* |

**Description**

Offset values for WorldView-2 & Worldview-3 sensors

**Usage**

worldview\_offset(sat\_id, band\_num)

**Arguments**

|  |  |
| --- | --- |
| sat\_id | String object specifying the satellite id. Either ‘WV02’ or WV03’ |
| band\_num | Band number. Indexing starts from zero |

**Value**

Float object

**Details**

|  |  |  |
| --- | --- | --- |
| **Band** | **WorldView-2** | **WorldView-3** |
| 0 | -7.478 | -8.604 |
| 1 | -5.736 | -5.809 |
| 2 | -3.546 | -4.996 |
| 3 | -3.564 | -3.649 |
| 4 | -2.512 | -3.021 |
| 5 | -4.120 | -4.521 |
| 6 | -3.300 | -5.522 |
| 7 | -2.891 | -2.992 |

See Kuester (2017) for more information

**Example**

satellite = ‘WV02’

gain = worldview\_offset(sat\_id = satellite, band\_num = 0)

|  |  |
| --- | --- |
| worldview\_eai | *Band averaged exoatmospheric solar irradiance* |

**Description**

Band averaged exoatmospheric solar irradiance for WorldView-2 & WorldView-3 sensors

**Usage**

worldview\_eai(sat\_id, band\_num)

**Arguments**

|  |  |
| --- | --- |
| sat\_id | String object specifying the satellite id. Either ‘WV02’ or WV03’ |
| band\_num | Band number. Indexing starts from zero |

**Value**

Float object

**Details**

|  |  |  |
| --- | --- | --- |
| **Band** | **WorldView-2** | **WorldView-3** |
| 0 | 1773.81 | 1757.89 |
| 1 | 2007.27 | 2004.61 |
| 2 | 1829.62 | 1830.18 |
| 3 | 1701.85 | 1712.07 |
| 4 | 1538.85 | 1535.33 |
| 5 | 1346.09 | 1348.08 |
| 6 | 1053.21 | 1055.94 |
| 7 | 856.599 | 858.77 |

See Kuester (2017) for more information

**Example**

satellite = ‘WV02’

gain = worldview\_eai(sat\_id = satellite, band\_num = 0)

|  |  |
| --- | --- |
| worldview\_band\_center | *Band centers* |

**Description**

Band centers for WorldView-2 & WorldView-3 sensors

**Usage**

worldview\_band\_center(sat\_id, band\_num)

**Arguments**

|  |  |
| --- | --- |
| sat\_id | String object specifying the satellite id. Either ‘WV02’ or WV03’ |
| band\_num | Band number. Indexing starts from zero |

**Value**

Float object

**Details**

|  |  |  |
| --- | --- | --- |
| **Band** | **WorldView-2** | **WorldView-3** |
| 0 | 0.4273 | 0.4274 |
| 1 | 0.4779 | 0.4819 |
| 2 | 0.5462 | 0.5471 |
| 3 | 0.6078 | 0.6043 |
| 4 | 0.6588 | 0.6601 |
| 5 | 0.7237 | 0.7227 |
| 6 | 0.8313 | 0.8240 |
| 7 | 0.9080 | 0.9136 |

See Kuester (2017) for more information

**Example**

satellite = ‘WV02’

gain = worldview\_band\_center(sat\_id = satellite, band\_num = 0)

|  |  |
| --- | --- |
| list\_files | *List the WorldView multispectral tiles within a zipped folder* |

**Description**

List the WorldView multispectral tiles within a zipped folder and extract pertinent info. Extracted info includes filepaths, cloud cover, and area-of-interest coverage.

**Usage**

list\_files(zip\_fp, aoi\_extent)

**Arguments**

|  |  |
| --- | --- |
| zip\_fp | String object specifying the location of a zipped folder |
| aoi\_extent | List object containing the upper left and lower right longitude and latitude of an area-of-interest in decimal degrees. It should be ordered upper left longitude, upper left latitude, lower right longitude, lower right latitude |

**Value**

DataFrame object

**Details**

The function will only look for multispectral tiles and associated metadata within the zipped folder. It will not look for panchromatic tiles.

**Example**

wdir = "C:\\Users\\JDoe" # set working directory

extent\_coord = [-76.7, 34.7, -76.5, 34.5] # set the aoi extent with ul\_lon, ul\_lat, lr\_lon, lr\_lat (decimal degrees)

input\_zip = os.path.join(wdir, "worldview\_tiles.zip")

folder\_df = list\_files(zip\_fp = input\_zip, aoi\_extent = extent\_coord)

|  |  |
| --- | --- |
| unzip\_tiles | *Unzip specified tiles* |

**Description**

Unzip a list of specified multispectral tiles and associated metadata files

**Usage**

unzip\_tiles(zip\_fp, tile\_dir, output\_dir)

**Arguments**

|  |  |
| --- | --- |
| zip\_fp | String object specifying the location of a zipped folder |
| tile\_dir | List object specifying the location of tiles within zipped folder that should be unzipped |
| output\_dir | String object specifying a directory path for unzipping |

**Value**

None

**Details**

The function will maintain the directory structure from the input zipped folder

**Example**

# unzip data

wdir = "C:\\Users\\JDoe" # Set working directory

extent\_coord = [-76.7, 34.7, -76.5, 34.5] # set the aoi extent with ul\_lon, ul\_lat, lr\_lon, lr\_lat (decimal degrees)

input\_zip = os.path.join(wdir, "worldview\_tiles.zip")

folder\_df = list\_files(zip\_fp = input\_zip, aoi\_extent = extent\_coord)

dir\_list = folder\_df['DIRECTORY'].tolist()

output\_folder = os.path.join(wdir, "unzipped")

unzip\_folder(zip\_fp = input\_zip, tile\_dir = [dir\_list[i] for i in [1]], output\_dir = output\_folder)

|  |  |
| --- | --- |
| clip\_image | *Clip image* |

**Description**

Clip image by area-of-interest extent

**Usage**

clip\_image(image\_fp, output\_fp, aoi\_extent, ds\_nodata = None)

**Arguments**

|  |  |
| --- | --- |
| image\_fp | String object specifying the filepath of the input image |
| output\_fp | String object specifying the desired filepath of the output image |
| aoi\_extent | List object containing the upper left and lower right longitude and latitude of an area-of-interest in decimal degrees. It should be ordered upper left longitude, upper left latitude, lower right longitude, lower right latitude. |
| ds\_nodata = None | An int or a float object specifying desired no data value. Default is set to None, which will produce an output image with the same no data value as the input image. |

**Value**

None

**Details**

The function only works on images that have been projected. Level 1B images from Maxar use rational polynomial coefficients instead of a projection. Those images must be projected before this function can be used.

**Example**

# clip images

wdir = "C:\\Users\\JDoe" # Set working directory

extent\_coord = [-76.7, 34.7, -76.5, 34.5] # set the aoi extent with ul\_lon, ul\_lat, lr\_lon, lr\_lat (decimal degrees)

input\_image = os.path.join(wdir, "projected\_image.TIF")

output\_image = os.path.join(wdir, "clipped\_image.TIF")

clip\_image(image\_fp = input\_image, output\_fp = output\_image, aoi\_extent = extent\_coord)

|  |  |
| --- | --- |
| project\_image | *Project image* |

**Description**

Project image into specified coordinate system

**Usage**

project\_image(image\_fp, output\_fp, target\_coord, res, rpc = False)

**Arguments**

|  |  |
| --- | --- |
| image\_fp | String object specifying the filepath of the input image |
| output\_fp | String object specifying the desired filepath of the output image |
| target\_coord | Projection string |
| res | Resolution in units used by desired projection |
| rpc = False | Boolean object to signify whether input image is georeferenced using rational polynomial coefficients. Default is set to False |
| resample\_method = “bilinear” | String object specifying the method to be used for resampling during projection. The default method is bilinear interpolation. Other options include: “near”, “cubic”, “cubicspline”, “lanczos”, “average”, “rms”, “mode”, “max”, “min”, “med”, “q1”, “q3”, and “sum”. |

**Value**

None

**Details**

This function requires that the input image is georeferenced using a projection or rational polynomial coefficients.

**Example**

# project images

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF"))

output\_image = os.path.join(wdir, "projected\_image.TIF")

project\_image(image\_fp = input\_image, output\_fp = output\_image, target\_coord = "EPSG:32618", res = 2, rpc = True)

|  |  |
| --- | --- |
| rad\_cal | *Convert Level 1B image to remote sensing reflectance* |

**Description**

Convert a Level 1B WorldView image from uncalibrated digital numbers to top-of-atmosphere remote sensing reflectance

**Usage**

rad\_cal(image\_fp, output\_fp, aoi\_extent = None, ds\_nodata = -32768)

**Arguments**

|  |  |
| --- | --- |
| image\_fp | String object specifying the filepath of the input image |
| output\_fp | String object specifying the desired filepath of the output image |
| aoi\_extent = None | List object containing the upper left and lower right longitude and latitude of an area-of-interest in decimal degrees. It should be ordered upper left longitude, upper left latitude, lower right longitude, lower right latitude. The extent coordinates will be used to mask the image. Default is set to None, which will mean the entire image is processed without masking. |
| dst\_nodata =  -32768 | Desired no data value for output image. Default is set to -32768 |

**Value**

None

**Details**

The output image will be scaled by 10000 so that it can be saved as signed short (16-bit) data to reduce file size and processing time. See Updike and Comp (2010); Kuester (2016); Coffer et al. (2020)for more information about radiometrically calibrating of WorldView-2 & -3 images. This function will embed the satellite id (‘WV02’ or ‘WV03’) in the output image’s metadata under the ‘sat\_id’ tag.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

extent\_coord = [-76.7, 34.7, -76.5, 34.5] # set the aoi extent with ul\_lon, ul\_lat, lr\_lon, lr\_lat (decimal degrees)

input\_image = os.path.join(wdir, "input\_image.TIF")

output\_image = os.path.join(wdir, "rad\_cal\_image.TIF")

rad\_cal(image\_fp = input\_image, output\_fp = output\_image, aoi\_extent = extent\_coord)

|  |  |
| --- | --- |
| embed\_dos\_value | *Calculate Dark Object Subtraction value for an image and embed it the metadata of the image* |

**Description**

Calculate a Dark Object Subtraction value for an image and embed it the metadata of the image

**Usage**

embed\_dos\_val(image\_fp, green\_band, nir\_band, dos\_band, ndwi\_threshold = 0, ds\_nodata = None)

**Arguments**

|  |  |
| --- | --- |
| image\_fp | String object specifying the filepath of the input image |
| green\_band | Int object specifying the band number for the green band. Indexing starts at 0 |
| nir\_band | Int object specifying the band number for the NIR1 band. Indexing starts at 0 |
| dos\_band | Int object specifying the band number for the Dark Object Subtraction band. Indexing starts at 0 |
| ndwi\_threshold = 0 | Int or float object specifying the Normalized Difference Water Index threshold. Default is set to zero |
| ds\_nodata = None | Signed or unsigned int, short, or float specifying the no data value of the input image. The default is None. If the default is used, the no data value will be extracted from the input image metadata |

**Value**

None

**Details**

SeeCoffer et al. (2020) for more information about extracting the Dark Object Subtraction value from a WorldView-2 & -3 image. This function will embed the Dark Object Subtraction value in the input image’s metadata under a tag called ‘dos\_value’.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "rad\_cal\_image.TIF")

embed\_dos\_val(image\_fp = input\_image, green\_band = 2, nir\_band = 6, dos\_band = 5, ndwi\_threshold = 0)

|  |  |
| --- | --- |
| min\_dos\_value | *Select the minimum Dark Object Subtraction value from a list of images* |

**Description**

Select the minimum Dark Object Subtraction value from a list of images

**Usage**

min\_dos\_value(image\_list)

**Arguments**

|  |  |
| --- | --- |
| image\_list | List object containing image filepaths |

**Value**

Float object

**Details**

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

rad\_cal\_images = glob.glob(os.path.join(wdir, "rad\_cal", "\*.TIF"))

min\_dos = min\_dos\_value(rad\_cal\_images)

|  |  |
| --- | --- |
| atm\_cor | *Perform Dark Object Subtraction atmospheric correction* |

**Description**

**Usage**

atm\_cor(image\_fp, output\_fp, dos\_band, rayleighExp, dos\_value = None, satellite = None)

**Arguments**

|  |  |
| --- | --- |
| image\_fp | String object specifying the filepath of the input image |
| output\_fp | String object specifying the filepath of the output image |
| dos\_band | Int object specifying the band number for the Dark Object Subtraction band. Indexing starts at 0 |
| rayleighExp | Float or int object specifying the Rayleigh exponent to use for atmospheric correction |
| dos\_value = None | Float object specifying the Dark Object Subtraction anchor value to use for atmospheric correction. The default is set to None. If no value is supplied then the dos\_value will be extracted from the metadata of the image. |
| satellite | String object specifying the satellite that captured the input image. Default is None. If the default is used, the satellite id will be extracted from the input image metadata. Values can either be None, ‘WV02’, or ‘WV03’. |

**Value**

None

**Details**

The output image will be scaled by 10000 so that it can be saved as signed short (16-bit) data to reduce file size and processing time. See Chavez (1988) and Coffer et al. (2020) for more information about the Dark Object Subtraction atmospheric correction process.

**Example**

# atmospheric correction

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "rad\_cal\_image.TIF")

output\_image = os.path.join(wdir, "atm\_corr\_image.TIF")

atm\_cor(image\_fp = input\_image, output\_fp = output\_image, dos\_band = 5, rayleighExp = 4, dos\_value = None, satellite = None)

|  |  |
| --- | --- |
| mosaic | *Mosaic a list of images* |

**Description**

Mosaic a list of images into a single scene

**Usage**

mosaic(image\_list, output\_fp)

**Arguments**

|  |  |
| --- | --- |
| image\_list | List object containing image filepaths |
| output\_fp | String object specifying desired output filepath |

**Value**

None

**Details**

The function outputs a GDAL virtual raster object (VRT). VRT files are small header files that point to the constituent tiles that make up a mosaic and specifies how they should be handled relative to one another. VRT files can be opened and viewed in ArcMap (Pro) the same way you would a typical image. Similarly, VRT files can be processed in Python the same way you would a typical image.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_list = glob.glob(os.path.join(wdir, "input\_images", "\*.TIF"))

output\_image = os.path.join(wdir, "mosaic.vrt")

mosaic(image\_list = input\_list, output\_fp = output\_image)

|  |  |
| --- | --- |
| multipart\_shp | *Check if a shapefile is multipart of singlepart* |

**Description**

Check if a shapefile is multipart or singlepart

**Usage**

multipart\_shp(shp\_fp)

**Arguments**

|  |  |
| --- | --- |
| shp\_fp | String object specifying the filepath of the input shapfile |

**Value**

Boolean object. True if multipart and False if singlepart

**Details**

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_shp = os.path.join(wdir, "ROIs", "multipart\_ROIs.shp")

multipart\_shp(shp\_fp = input\_shp)

|  |  |
| --- | --- |
| multipart\_to\_singlepart | *Convert a multipart shapefile to a singlepart shapefile* |

**Description**

**Usage**

multipart\_to\_singlepart(shp\_fp, out\_fp, output\_proj = None)

**Arguments**

|  |  |
| --- | --- |
| shp\_fp | String object specifying the filepath of the input shapfile |
| out\_fp | String object specifying the filepath of the output shapefile |
| output\_proj = None | Projection string for the output shapefile. Default is set to None which result in an output shapefile with the same projection as the input shapefile |

**Value**

None

**Details**

This function will output a new shapefile with singlepart geometry instead of multipart geometry. Unless otherwise specified, the output shapefile will be georeferenced the same as the input shapefile.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_shp = os.path.join(wdir, "multipart\_ROIs.shp")

output\_shp = os.path.join(wdir, "singlepart\_ROIs.shp")

multipart\_to\_singlepart(shp\_fp = input\_shp, out\_fp = output\_shp)

|  |  |
| --- | --- |
| roi\_classes | *List the region-of-interst classes* |

**Description**

List the unique class names within a region-of-interest shapefile

**Usage**

roi\_classes(shp\_fp, field\_name = 'CLASS\_NAME')

**Arguments**

|  |  |
| --- | --- |
| shp\_fp | String object specifying the filepath of the input shapfile |
| field\_name = 'CLASS\_NAME' | The name of the field that contains the class name for each polygon within the shapefile. It is set to 'CLASS\_NAME' by default. The function will prompt you to input the field\_name, if it is left as None. |

**Value**

List object containing region-of-interest class names

**Details**

This function assumes that there is a field (column) within the shapefile attribute table that specifies the class that each polygon belongs to.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_shp = os.path.join(wdir, "singlepart\_ROIs.shp")

class\_names = roi\_classes(shp\_fp = input\_shp, field\_name = 'CLASS\_NAME')

|  |  |
| --- | --- |
| shp\_to\_roi | *Extract image regions-of-interest* |

**Description**

Use a singlepart polygon shapefile to extract image regions-of-interest from a project input image

**Usage**

shp\_to\_roi(shp\_fp, image\_fp, output\_dir, field\_name = 'CLASS\_NAME')

**Arguments**

|  |  |
| --- | --- |
| shp\_fp | String object specifying the filepath of the input shapfile |
| image\_fp | String object specifying the filepath of the input image |
| output\_dir | String object specifying the desired directory to output image regions-of-interest |
| field\_name = 'CLASS\_NAME' | The name of the field that contains the class name for each polygon within the shapefile. It is set to 'CLASS\_NAME' by default. The function will prompt you to input the field\_name, if it is left as None. |

**Value**

None

**Details**

This function will output a series of images extracted from the input image using each region-of-interest polygon within the input shapefile and label it with the polygon’s class name attribute. Input shapefile must be singlepart geometry.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF")

output\_ROIs = os.path.join(wdir, "extracted\_ROIs")

input\_shp = os.path.join(wdir, "singlepart\_ROIs.shp")

shp\_to\_roi(image\_fp = input\_image, output\_dir = output\_ROIs, shp\_fp = input\_shp)

|  |  |
| --- | --- |
| dcnn\_model | *Define a deep convolutional neural network* |

**Description**

Initialize a keras model object

**Usage**

dcnn\_model(numChannels, dimension, numClasses)

**Arguments**

|  |  |
| --- | --- |
| numChannels | Int object specifying the number of bands (or channels) the image contains that will be used for training and classification |
| dimension | Int object specifying the sampling (kernel) dimensions |
| numClasses | Int object specifying the number of desired classes |

**Value**

Keras model object

**Details**

This function will initialize a Keras model object with…[insert description from Megan’s manuscript]

**Example**

class\_names = ['DEEP\_WATER', 'SUBMERGED\_SAND', 'SEAGRASS', 'LAND']

multi\_model = dcnn\_model(numChannels = 8, dimension = 3, numClasses = len(class\_names))

|  |  |
| --- | --- |
| train\_dcnn | *Train a deep convolutional neural network* |

**Description**

Train a deep convolutional neural network using a series of image regions-of-interest

**Usage**

train\_dcnn(save\_directory, training\_data\_directory, class\_names, numChannels, dimension, …)

**Arguments**

|  |  |
| --- | --- |
| cnnFileName | String object specifying the filepath for the trained dcnn model. The filename specified in this filepath should end with .h5 file extension. |
| training\_data\_directory | String object specifying the directory that contains the training data |
| class\_names | List object containing class names |
| numChannels | Int object specifying the number of bands (or channels) the image contains that will be used for training and classification |
| dimension | Int object specifying the sampling (kernel) dimensions |
| selected\_sample\_per\_class = 20000 | Maximum number of samples per class. Default is set to 20000 |
| balanced\_option = 'balanced' | Two possible values: ‘balanced’ and ‘unbalanced’. Default is set to ‘balanced’. If ‘balanced’, data will be upsampled or downsampled through random sampling to balance the number of samples per class. If ‘unbalanced’, the number of samples per class will be unaltered. |
| epochs = 500 | The number of epochs. This is the number of times that the model is exposed to all of the training data. The default is set to 500 |
| batchSize = 256 | The model is trained using batches of training data. The default is set to 256 |
| deleted\_channels = [] | A list object containing undesidered bands within the training data. The default is set to None, which means that all bands will be used to train the model. |

**Value**

None

**Details**

This function uses a series of image regions-of-interest to train a deep convultional neural network. An .h5 file containg the trained model will be outputted along with an plot of training loss and accuracy and .txt file containing the model training output.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

dcnn\_fp = os.path.join(wdir, "dcnn\_model.h5")

input\_data = os.path.join(wdir, "ROIs")

image\_classes = ['DEEP\_WATER', 'SUBMERGED\_SAND', 'SEAGRASS', 'LAND']

train\_dcnn(save\_directory = dcnn\_fp, training\_data\_directory = input\_data, class\_names = image\_classes, numChannels = 8, dimension = 3)

|  |  |
| --- | --- |
| dcnn\_classification | *Classify an image with a deep convolutional neural network* |

**Description**

Classify an input image using a deep convolutional neural network

**Usage**

dcnn\_classification(image\_fp, dcnn\_fp, output\_fp)

**Arguments**

|  |  |
| --- | --- |
| image\_fp | String object specifying the filepath of the input image |
| dcnn\_fp | String object specifying the filepath of a .h5 model file |
| output\_fp | String object specifying the desired filepath of the output image classification |
| bSize = 256 | The processing block size. The default is set to 256. |

**Value**

None

**Details**

This function uses a trained deep convultional neural network to classify an input image. The output classification will be 8-bit with the same x,y dimensions and spatial reference as the input image.

**Example**

wdir = "C:\\Users\\JDoe" # Set working directory

input\_image = os.path.join(wdir, "input\_image.TIF")

output\_classification = os.path.join(wdir, "image\_classification.TIF")

dcnn\_fp = os.path.join(wdir, "dcnn\_model.h5")

dcnn\_classification(image\_fp = input\_image, dcnn\_fp = dcnn\_fp, output\_fp = output\_classification)

**References**

Chavez, P. S. (1988). An improved dark-object subtraction technique for atmospheric scattering correction of multispectral data. *Remote Sensing of Environment, 24*(3), 459-479, doi:10.1016/0034-4257(88)90019-3.

Coffer, M. M., Schaeffer, B. A., Zimmerman, R. C., Hill, V., Li, J., Islam, K. A., et al. (2020). Performance across WorldView-2 and RapidEye for reproducible seagrass mapping. *Remote Sensing of Environment, 250*, 112036, doi:10.1016/j.rse.2020.112036.

Kuester, M. (2016). Radiometric Use of WorldView-3 Imagery. DigitalGlobe.

Kuester, M. (2017). Absolute Radiometric Calibration Adjustment Factors 2016 Season Results. DigitalGlobe.

Updike, T., & Comp, C. (2010). Radiometric Use of WorldView-2 Imagery. DigitalGlobe.