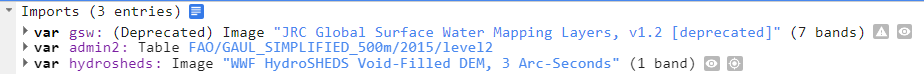
Code:



var beforeStart = '2019-02-16'

var beforeEnd = '2019-03-16'

var afterStart = '2019-03-17'

var afterEnd = '2019-03-25'

var ernakulam = admin2.filter(ee.Filter.eq('ADM2\_NAME', 'Gorgan'))

var geometry = ernakulam.geometry()

Map.addLayer(geometry, {color: 'grey'}, 'Gorgan District')

var collection= ee.ImageCollection('COPERNICUS/S1\_GRD')

.filter(ee.Filter.eq('instrumentMode','IW'))

.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))

.filter(ee.Filter.eq('orbitProperties\_pass', 'DESCENDING'))

.filter(ee.Filter.eq('resolution\_meters',10))

.filterBounds(geometry)

.select('VH');

var beforeCollection = collection.filterDate(beforeStart, beforeEnd)

var afterCollection = collection.filterDate(afterStart,afterEnd)

var before = beforeCollection.mosaic().clip(geometry);

var after = afterCollection.mosaic().clip(geometry);

Map.addLayer(before, {min:-25,max:0}, 'Before Floods', false);

Map.addLayer(after, {min:-25,max:0}, 'After Floods', false);

var beforeFiltered = ee.Image(toDB(RefinedLee(toNatural(before))))

var afterFiltered = ee.Image(toDB(RefinedLee(toNatural(after))))

Map.addLayer(beforeFiltered, {min:-25,max:0}, 'Before Filtered', false);

Map.addLayer(afterFiltered, {min:-25,max:0}, 'After Filtered', false);

var difference = afterFiltered.divide(beforeFiltered);

// Define a threshold

var diffThreshold = 1.25;

// Initial estimate of flooded pixels

var flooded = difference.gt(diffThreshold).rename('water').selfMask();

Map.addLayer(flooded, {min:0, max:1, palette: ['orange']}, 'Initial Flood Area', false);

// Mask out area with permanent/semi-permanent water

var permanentWater = gsw.select('seasonality').gte(5).clip(geometry)

var flooded = flooded.where(permanentWater, 0).selfMask()

Map.addLayer(permanentWater.selfMask(), {min:0, max:1, palette: ['blue']}, 'Permanent Water')

// Mask out areas with more than 6 percent slope using the HydroSHEDS DEM

var slopeThreshold = 6;

var terrain = ee.Algorithms.Terrain(hydrosheds);

var slope = terrain.select('slope');

var flooded = flooded.updateMask(slope.lt(slopeThreshold));

Map.addLayer(slope.gte(slopeThreshold).selfMask(), {min:0, max:1, palette: ['cyan']}, 'Steep Areas', false)

// Remove isolated pixels

// connectedPixelCount is Zoom dependent, so visual result will vary

var connectedPixelThreshold = 8;

var connections = flooded.connectedPixelCount(25)

var flooded = flooded.updateMask(connections.gt(connectedPixelThreshold))

Map.addLayer(connections.lte(connectedPixelThreshold).selfMask(), {min:0, max:1, palette: ['yellow']}, 'Disconnected Areas', false)

Map.addLayer(flooded, {min:0, max:1, palette: ['red']}, 'Flooded Areas');

// Calculate Affected Area

print('Total District Area (Ha)', geometry.area().divide(10000))

var stats = flooded.multiply(ee.Image.pixelArea()).reduceRegion({

reducer: ee.Reducer.sum(),

geometry: geometry,

scale: 30,

maxPixels: 1e10,

tileScale: 16

})

print('Flooded Area (Ha)', ee.Number(stats.get('water')).divide(10000))

// If the above computation times out, you can export it

var flooded\_area = ee.Number(stats.get('water')).divide(10000);

var feature = ee.Feature(null, {'flooded\_area': flooded\_area})

var fc = ee.FeatureCollection([feature])

Export.table.toDrive({

collection: fc,

description: 'Flooded\_Area\_Export',

folder: 'earthengine',

fileNamePrefix: 'flooded\_area',

fileFormat: 'CSV'})

//############################

// Speckle Filtering Functions

//############################

// Function to convert from dB

function toNatural(img) {

return ee.Image(10.0).pow(img.select(0).divide(10.0));

}

//Function to convert to dB

function toDB(img) {

return ee.Image(img).log10().multiply(10.0);

}

//Apllying a Refined Lee Speckle filter as coded in the SNAP 3.0 S1TBX:

//https://github.com/senbox-org/s1tbx/blob/master/s1tbx-op-sar-processing/src/main/java/org/esa/s1tbx/sar/gpf/filtering/SpeckleFilters/RefinedLee.java

//Adapted by Guido Lemoine

// by Guido Lemoine

function RefinedLee(img) {

// img must be in natural units, i.e. not in dB!

// Set up 3x3 kernels

var weights3 = ee.List.repeat(ee.List.repeat(1,3),3);

var kernel3 = ee.Kernel.fixed(3,3, weights3, 1, 1, false);

var mean3 = img.reduceNeighborhood(ee.Reducer.mean(), kernel3);

var variance3 = img.reduceNeighborhood(ee.Reducer.variance(), kernel3);

// Use a sample of the 3x3 windows inside a 7x7 windows to determine gradients and directions

var sample\_weights = ee.List([[0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0], [0,1,0,1,0,1,0], [0,0,0,0,0,0,0], [0,1,0,1,0,1,0],[0,0,0,0,0,0,0]]);

var sample\_kernel = ee.Kernel.fixed(7,7, sample\_weights, 3,3, false);

// Calculate mean and variance for the sampled windows and store as 9 bands

var sample\_mean = mean3.neighborhoodToBands(sample\_kernel);

var sample\_var = variance3.neighborhoodToBands(sample\_kernel);

// Determine the 4 gradients for the sampled windows

var gradients = sample\_mean.select(1).subtract(sample\_mean.select(7)).abs();

gradients = gradients.addBands(sample\_mean.select(6).subtract(sample\_mean.select(2)).abs());

gradients = gradients.addBands(sample\_mean.select(3).subtract(sample\_mean.select(5)).abs());

gradients = gradients.addBands(sample\_mean.select(0).subtract(sample\_mean.select(8)).abs());

// And find the maximum gradient amongst gradient bands

var max\_gradient = gradients.reduce(ee.Reducer.max());

// Create a mask for band pixels that are the maximum gradient

var gradmask = gradients.eq(max\_gradient);

// duplicate gradmask bands: each gradient represents 2 directions

gradmask = gradmask.addBands(gradmask);

// Determine the 8 directions

var directions = sample\_mean.select(1).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(7))).multiply(1);

directions = directions.addBands(sample\_mean.select(6).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(2))).multiply(2));

directions = directions.addBands(sample\_mean.select(3).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(5))).multiply(3));

directions = directions.addBands(sample\_mean.select(0).subtract(sample\_mean.select(4)).gt(sample\_mean.select(4).subtract(sample\_mean.select(8))).multiply(4));

// The next 4 are the not() of the previous 4

directions = directions.addBands(directions.select(0).not().multiply(5));

directions = directions.addBands(directions.select(1).not().multiply(6));

directions = directions.addBands(directions.select(2).not().multiply(7));

directions = directions.addBands(directions.select(3).not().multiply(8));

// Mask all values that are not 1-8

directions = directions.updateMask(gradmask);

// "collapse" the stack into a singe band image (due to masking, each pixel has just one value (1-8) in it's directional band, and is otherwise masked)

directions = directions.reduce(ee.Reducer.sum());

//var pal = ['ffffff','ff0000','ffff00', '00ff00', '00ffff', '0000ff', 'ff00ff', '000000'];

//Map.addLayer(directions.reduce(ee.Reducer.sum()), {min:1, max:8, palette: pal}, 'Directions', false);

var sample\_stats = sample\_var.divide(sample\_mean.multiply(sample\_mean));

// Calculate localNoiseVariance

var sigmaV = sample\_stats.toArray().arraySort().arraySlice(0,0,5).arrayReduce(ee.Reducer.mean(), [0]);

// Set up the 7\*7 kernels for directional statistics

var rect\_weights = ee.List.repeat(ee.List.repeat(0,7),3).cat(ee.List.repeat(ee.List.repeat(1,7),4));

var diag\_weights = ee.List([[1,0,0,0,0,0,0], [1,1,0,0,0,0,0], [1,1,1,0,0,0,0],

[1,1,1,1,0,0,0], [1,1,1,1,1,0,0], [1,1,1,1,1,1,0], [1,1,1,1,1,1,1]]);

var rect\_kernel = ee.Kernel.fixed(7,7, rect\_weights, 3, 3, false);

var diag\_kernel = ee.Kernel.fixed(7,7, diag\_weights, 3, 3, false);

// Create stacks for mean and variance using the original kernels. Mask with relevant direction.

var dir\_mean = img.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel).updateMask(directions.eq(1));

var dir\_var = img.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel).updateMask(directions.eq(1));

dir\_mean = dir\_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel).updateMask(directions.eq(2)));

dir\_var = dir\_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel).updateMask(directions.eq(2)));

// and add the bands for rotated kernels

for (var i=1; i<4; i++) {

dir\_mean = dir\_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(), rect\_kernel.rotate(i)).updateMask(directions.eq(2\*i+1)));

dir\_var = dir\_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(), rect\_kernel.rotate(i)).updateMask(directions.eq(2\*i+1)));

dir\_mean = dir\_mean.addBands(img.reduceNeighborhood(ee.Reducer.mean(), diag\_kernel.rotate(i)).updateMask(directions.eq(2\*i+2)));

dir\_var = dir\_var.addBands(img.reduceNeighborhood(ee.Reducer.variance(), diag\_kernel.rotate(i)).updateMask(directions.eq(2\*i+2)));

}

// "collapse" the stack into a single band image (due to masking, each pixel has just one value in it's directional band, and is otherwise masked)

dir\_mean = dir\_mean.reduce(ee.Reducer.sum());

dir\_var = dir\_var.reduce(ee.Reducer.sum());

// A finally generate the filtered value

var varX = dir\_var.subtract(dir\_mean.multiply(dir\_mean).multiply(sigmaV)).divide(sigmaV.add(1.0));

var b = varX.divide(dir\_var);

var result = dir\_mean.add(b.multiply(img.subtract(dir\_mean)));

return(result.arrayFlatten([['sum']]));

}

// legend

// set position of panel

var legend = ui.Panel({

style: {

position: 'top-right',

padding: '8px 15px'

}

});

// Create legend title

var legendTitle = ui.Label({

value: 'Legend',

style: {

fontWeight: 'bold',

fontSize: '18px',

margin: '0 0 4px 0',

padding: '0'

}

});

// Add the title to the panel

legend.add(legendTitle);

// Creates and styles 1 row of the legend.

var makeRow = function(color, name) {

// Create the label that is actually the colored box.

var colorBox = ui.Label({

style: {

backgroundColor: '#' + color,

// Use padding to give the box height and width.

padding: '8px',

margin: '0 0 4px 0'

}

});

// Create the label filled with the description text.

var description = ui.Label({

value: name,

style: {margin: '0 0 4px 6px'}

});

// return the panel

return ui.Panel({

widgets: [colorBox, description],

layout: ui.Panel.Layout.Flow('horizontal')

});

};

// Palette with the colors

//var palette =['000000', 'ffffff', 'FF0000'];

var palette =[ '1500ff', 'FF0000'];

// name of the legend

//var names = ['Boston boundary','0','400,000'];

var names = ['Permanent water bodies ','Flooded area'];

// Add color and and names

for (var i = 0; i < 2; i++) {

legend.add(makeRow(palette[i], names[i]));

}

// add legend to map (alternatively you can also print the legend to the console)

Map.add(legend);

///////////////////////////////////////////////////////////////////////////////////

// ROI

// Change coordinates below

var ROI = ernakulam

// set map center and zoom level

// ------------------------------

// ----- define time frames -----

// ------------------------------

// Landsat 7: 1999 - present

// Landsat 7 has revisit time of 16 days

var L\_T1\_a = '2019-02-16'

var L\_T1\_b = '2019-03-16'

var L\_T2\_a = '2019-3-16'

var L\_T2\_b = '2019-4-16'

// ----------------------------

// ----------- NAIP -----------

// ----------------------------

// -------------------------------

// ----------- LANDSAT -----------

// -------------------------------

//cloud mask

var maskcloud\_L = function(image) {

var qa = image.select('pixel\_qa');

// If the cloud bit (5) is set and the cloud confidence (7) is high

// or the cloud shadow bit is set (3), then it's a bad pixel.

var cloud = qa.bitwiseAnd(1 << 7)

.and(qa.bitwiseAnd(1 << 9))

.or(qa.bitwiseAnd(1 << 5));

// Remove edge pixels that don't occur in all bands

var mask2 = image.mask().reduce(ee.Reducer.min());

return image.updateMask(cloud.not()).updateMask(mask2);

};

// -------- get RGB Visual Imagery for your ROI --------

var landsat = ee.ImageCollection('LANDSAT/LE07/C01/T1\_SR')

.map(maskcloud\_L)

.filterBounds(ROI);

var landsat\_T1 = landsat.filterDate(L\_T1\_a,L\_T1\_b).median().clip(ROI);

var landsat\_T2 = landsat.filterDate(L\_T2\_a,L\_T2\_b).median().clip(ROI);

// we want to use the red green blue band so we use B3red), B2(green), B1(blue)

Map.addLayer(landsat\_T1 ,{bands:['B3','B2','B1'], min:0,max:2500}, 'Landsat 7 T1', 0)

Map.addLayer(landsat\_T2 ,{bands:['B3','B2','B1'], min:0,max:2500}, 'Landsat 7 T2', 0)

// -------- Calculate NDVI using red and Near Infrared Band --------

// use .normalizedDifference to calculate NDVI using near infrared (B4) and red (B3)

var landsat\_ndvi\_t1 = landsat\_T1.normalizedDifference(['B4', 'B3']).rename('NDVI\_Landsat\_T1');

var landsat\_ndvi\_t2 = landsat\_T2.normalizedDifference(['B4', 'B3']).rename('NDVI\_Landsat\_T2');

Map.addLayer(landsat\_ndvi\_t1, {min:-0.15, max:0.7, palette: ['brown', 'white', 'green']}, 'Landsat NDVI T1 30m', 0);

Map.addLayer(landsat\_ndvi\_t2, {min:-0.15, max:0.7, palette: ['brown', 'white', 'green']}, 'Landsat NDVI T2 30m', 0);

// -------- NDVI histogram between T1 T2 --------

// use the histogram to compare dsitribution of NDVI value between the two timeframes

var NDVI\_T1\_T2 = landsat\_ndvi\_t1.addBands(landsat\_ndvi\_t2);

var chart =

ui.Chart.image.histogram({image: NDVI\_T1\_T2, region: ROI, scale: 30, maxPixels: 1e13})

.setSeriesNames(['LANDSAT T1', 'LANDSAT T2'])

.setOptions({

title: 'LANDSAT NDVI Histograms for T1 and T2',

hAxis: {title: 'NDVI', titleTextStyle: {italic: false, bold: true}},

vAxis: {title: 'Count', titleTextStyle: {italic: false, bold: true}},

colors: ['cf513e', '1d6b99'],

opacity:0

});

print(chart);

// ---------------------------------

// ----- time series Landsat 7 -----

// ---------------------------------

//NDVI calculation:

var ndvi\_func = function (i) {

var ndvi = i.normalizedDifference (['B4', 'B3']).rename ('NDVI')

return i.addBands(ndvi);

}

var ndvi\_landsat = landsat

.map(ndvi\_func) // use this function on each image in the collection

.filterDate('2000-01-01','2020-12-31') // define the date, you can use your T1 -T2

var chart = ui.Chart.image.seriesByRegion({

imageCollection: ndvi\_landsat,

regions: ROI,

reducer: ee.Reducer.mean(),

band: 'NDVI',

scale: 30, // here is the pixel size of the image (landsat is 30m per pixel)

}).setOptions({

title: 'LANDSAT 7 - NDVI',

hAxis: {title: 'Year', titleTextStyle: {italic: false, bold: true}},

vAxis: {title: 'NDVI', titleTextStyle: {italic: false, bold: true}},

colors: ['red'],

});

print(chart);

// ---------

// ---------

//----------

ROI=flooded\_area

// define variables by using the “var” command, here defining data collection time frame

// data available from 1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015, 2020, 2025, 2030 (projection). Range (start and end) is required.

// here we use 2000 to 2020 as our time range

var T1\_start = '1980-01-01'

var T1\_ends = '1980-12-31'

var T2\_start = '2020-01-01'

var T2\_ends = '2020-12-31'

// ------------------------------------------------------

// ------ Loading the dataset we want to work with ------

// ------------------------------------------------------

// first we create a layer to remove large water bodies such as seas, ocean

// we used elevatoin data from Shuttle Radar Topography Mission (SRTM)

var water\_mask = ee.Image("CGIAR/SRTM90\_V4").mask()

//Population data

// Check the data source: https://developers.google.com/earth-engine/datasets/catalog/JRC\_GHSL\_P2023A\_GHS\_POP

// click on Bands and you can see the data you want to use is the population count

// ee.ImageCollection is a function to load a data raster collection

// imageCollection is essentially collection of raster images

// once the collection is loaded, we select the band (data) we want to use

var pop\_collection = ee.ImageCollection("JRC/GHSL/P2023A/GHS\_POP")

.select('population\_count')

// use print to see data attributes in the console panel, notice that under features we will see 12 images

print(pop\_collection)

// to work with an image in an image collection, we must filter the image based on a criteria

// in this case, we want to create two image variable for 2 time frames, for temporal comparison - here we will filter the collection by the time frames we set

// once we filter an iamge collection, we have to transform it to a single image layer using a "reducer". This is a REQUIRED

// .first() is a REQUIRED command typically intended for data collection that has multiple layer

// here there is only 1 layer for each time frame, but still required

// .updateMask is to mask out (remove) large water body from the filter image

var T1\_pop = pop\_collection.filterDate(T1\_start, T1\_ends).first().updateMask(water\_mask).clip(geometry);

var T2\_pop = pop\_collection.filterDate(T2\_start, T2\_ends).first().updateMask(water\_mask).clip(geometry);

print(T2\_pop)

// CALCULATE population change by subtracting T2 with T1

var T2T1\_pop\_change = T2\_pop.subtract(T1\_pop);

// DISPLAY images in GEE using Map.addLayer(image, {min, max, palette}, 'Layer Name' )

Map.addLayer(T1\_pop, {min: 0, max: 300, palette: [ 'ffffff', 'grey' ]}, 'Population T1');

Map.addLayer(T2\_pop, {min: 0, max: 300, palette: [ 'ffffff', 'grey' ]}, 'Population T2');

//use symmetric min and max to maintain white as zero, red for increase and blue for decrease

Map.addLayer(T2T1\_pop\_change, { min:-30, max: 30, palette: [ 'blue','white', 'red' ]}, 'Pop Change T1-T2');

Map.addLayer(flooded, {min:0, max:1, palette: ['red']}, 'Flooded Areas');