Code for Landsat preprocessing (link below)

<https://code.earthengine.google.com/953d59d2a9967c7ab5df2df67e571659>

var region\_name = "sumatra";

*// 1. Create a polygon ‘fc’ defining the area to process and mosaick.*

var fc = ee.Geometry.Polygon(

[[[100.02923463812999, 2.559919448517347],

[100.02923463812999, -5.002440879053686],

[106.10467409125499, -5.002440879053686],

[106.10467409125499, 2.559919448517347]]], null, false);

Map.addLayer(fc, {color: '000000'}, 'region', false);

var saName = region\_name;

var crs = 'EPSG:4326'; // CRS = WGS84 (or use 'EPSG:32647' for WGS84/UTM Zone 47N)

*// 2. Update the startJulian and endJulian variables to indicate your seasonal constraints.*

*// This supports wrapping for tropics and southern hemisphere.*

*// startJulian: Starting Julian date*

*// endJulian: Ending Julian date*

var startJulian = 1;

var endJulian = 365;

*//Use Julian calendar here: https://www-air.larc.nasa.gov/tools/jday.htm*

*//Dry season*

var day1 = 335;//1 December (335)

var day2 = 90;//30 March (90)

*//Rainy season*

var day3 = 91;//1 April (91)

var day4 = 334;//31 November (334)

*//* *3. Specify start and end years for all analyses*

*// More than a 3 year span should be provided for time series methods to work well*.

//If using Fmask as the cloud/cloud shadow masking method, this does not matter

var startYear = 2015;

var endYear = 2016;

*// 4. Specify an annual buffer to include imagery from the same season*

*// timeframe from the prior and following year. timeBuffer = 1 will result*

*//in a 3 year moving window*

//var timebuffer = 1;

*// 5. Set up Names for the export*

var outputName = 'sumatra';

*//6. Provide location composites will be exported to*

var exportPathRoot = 'users/thuansarzynski/Indonesia';

*//7. Choose medoid or median compositing method.*

*//Median tends to be smoother, while medoid retains*

*//single date of observation across all bands*

*//Specify compositing method (median or medoid)*

var compositingMethod = 'median';

*//8. Choose Top of Atmospheric (TOA) or Surface Reflectance (SR)*

*//Specify TOA or SR*

var toaOrSR = 'sr';// 'sr' or 'toa'

*//9. Specify which cloud/cloud shadow masking method*

*//Choices are fmask, cloudScoreTDOM, or hybrid to run cloudScore/TDOM and then fmask*

var cloudcloudShadowMaskingMethod = 'hybrid';

*//10. Choose whether to include Landsat 7*

*//Generally only included when data are limited*

var includeSLCOffL7 = true;

*//If cloudScoreTDOM is chosen*

*// cloudScoreThresh: If using the cloudScoreTDOMShift method-Threshold for cloud*

*// masking (lower number masks more clouds. Between 10 and 30 generally*

*// works best)*

var cloudScoreThresh = 5;

*//Percentile of cloud score to pull from time series to represent a minimum for the cloud score over time for a given pixel*

*//Reduces comission errors over cool bright surfaces*

*//Generally between 5 and 10 works well. 0 generally is a bit noisy*

var cloudScorePctl = 5;

*// zScoreThresh: Threshold for cloud shadow masking- lower number masks out*

*// less. Between -0.8 and -1.2 generally works well*

var zScoreThresh = -1;

*// shadowSumThresh: Sum of IR bands to include as shadows within TDOM and the*

*// shadow shift method (lower number masks out less)*

var shadowSumThresh = 0.35;

*// contractPixels: The radius of the number of pixels to contract (negative buffer) clouds and cloud*

*// shadows by. Intended to eliminate smaller cloud patches that are likely errors*

*// (1.5 results in a -1 pixel buffer)(0.5 results in a -0 pixel buffer)*

*// (1.5 or 2.5 generally is sufficient)*

var contractPixels = 1.5;

*// dilatePixels: The radius of the number of pixels to dilate (buffer) clouds and cloud*

*// shadows by. Intended to include edges of clouds/cloud shadows that are often missed*

*//(1.5 results in a 1 pixel buffer)(0.5 results in a 0 pixel buffer)*

*// (2.5 or 3.5 generally is sufficient)*

var dilatePixels = 3.5;

*//End user parameters*

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

var vizParamsFalse = {'min': 0.1,'max': [0.3,0.4,0.4], 'bands':'swir1,nir,red', 'gamma':1.6};

var vizParamsViz = {'min': 0.02,'max': [0.2,0.2,0.2], 'bands':'red,green,blue', 'gamma':1.6};

var sensorBandDict ={'L8TOA' : ee.List([1,2,3,4,5,9,6,'BQA']),

'L7TOA' : ee.List([0,1,2,3,4,5,7,'BQA']),

'L5TOA' : ee.List([0,1,2,3,4,5,6,'BQA']),

'L4TOA' : ee.List([0,1,2,3,4,5,6,'BQA']),

'L8SR' : ee.List([1,2,3,4,5,7,6,'pixel\_qa']),

'L7SR' : ee.List([0,1,2,3,4,5,6,'pixel\_qa']),

'L5SR' : ee.List([0,1,2,3,4,5,6,'pixel\_qa']),

'L4SR' : ee.List([0,1,2,3,4,5,6,'pixel\_qa']),

};

var sensorBandNameDict ={'TOA' : ee.List(['blue','green','red','nir','swir1','temp','swir2','BQA']),

'SR' :ee.List(['blue','green','red','nir','swir1','temp', 'swir2','pixel\_qa'])};

var collectionDict = {'L8TOA':'LANDSAT/LC08/C01/T1\_TOA',

'L7TOA':'LANDSAT/LE07/C01/T1\_TOA',

'L5TOA':'LANDSAT/LT05/C01/T1\_TOA',

'L4TOA':'LANDSAT/LT04/C01/T1\_TOA',

'L8SR':'LANDSAT/LC08/C01/T1\_SR',

'L7SR':'LANDSAT/LE07/C01/T1\_SR',

'L5SR':'LANDSAT/LT05/C01/T1\_SR',

'L4SR':'LANDSAT/LT04/C01/T1\_SR'};

var multImageDict = {'TOA':ee.Image([1,1,1,1,1,1,1,1]),'SR':ee.Image([0.0001,0.0001,0.0001,0.0001,0.0001,0.1,0.0001,1])} ;

*// Prepare dates*

if(startJulian > endJulian){endJulian = endJulian + 365}

var startDate = ee.Date.fromYMD(startYear,1,1).advance(startJulian-1,'day');

var endDate = ee.Date.fromYMD(endYear,1,1).advance(endJulian-1,'day');

*//print('Start and end dates:',startDate,endDate);*

toaOrSR = toaOrSR.toUpperCase();

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

*// A helper to apply an expression and linearly rescale the output.*

*// Used in the landsatCloudScore function below.*

function rescale(img, exp, thresholds) {

return img.expression(exp, {img: img})

.subtract(thresholds[0]).divide(thresholds[1] - thresholds[0]);

}

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

*// Compute a cloud score and adds a band that represents the cloud mask.*

*// This expects the input image to have the common band names:*

*// ["red", "blue", etc]*

function landsatCloudScore(img) {

*// Compute several indicators of cloudiness and take the minimum of them.*

var score = ee.Image(1.0);

*// Clouds are reasonably bright in the blue band.*

score = score.min(rescale(img, 'img.blue', [0.1, 0.3]));

*// Clouds are reasonably bright in all visible bands.*

score = score.min(rescale(img, 'img.red + img.green + img.blue', [0.2, 0.8]));

*// Clouds are reasonably bright in all infrared bands.*

score = score.min(rescale(img, 'img.nir + img.swir1 + img.swir2', [0.3, 0.8]));

*// Clouds are reasonably cool in temperature.*

score = score.min(rescale(img,'img.temp', [300, 290]));

*// However, clouds are not snow.*

var ndsi = img.normalizedDifference(['green', 'swir1']);

score = score.min(rescale(ndsi, 'img', [0.8, 0.6]));

*// var ss = snowScore(img).select(['snowScore']);*

*// score = score.min(rescale(ss, 'img', [0.3, 0]));*

score = score.multiply(100).byte();

*//score = score.lt(cloudThresh).focal\_max(contractPixels).focal\_min(dilatePixels).rename('cloudMask');*

*//img = img.updateMask(score);*

return score;

}

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

//FUNCTIONS

function cFmaskCloud(img){

var cloud = img.select('pixel\_qa').bitwiseAnd(32).neq(0);

return img.updateMask(cloud.not());

}

function cFmaskCloudShadow(img){

var cloud\_shadow = img.select('pixel\_qa').bitwiseAnd(8).neq(0);

return img.updateMask(cloud\_shadow.not());

}

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

*//Function for finding dark outliers in time series.*

*//Original concept written by Carson Stam and adapted by Ian Housman.*

function simpleTDOM2(collection,zScoreThresh,shadowSumThresh,contractPixels,dilatePixels){

var shadowSumBands = ['nir','swir1'];

*//Get some pixel-wise stats for the time series*

var irStdDev = collection.select(shadowSumBands).reduce(ee.Reducer.stdDev());

var irMean = collection.select(shadowSumBands).mean();

*//Mask out dark dark outliers*

collection = collection.map(function(img){

var zScore = img.select(shadowSumBands).subtract(irMean).divide(irStdDev);

var irSum = img.select(shadowSumBands).reduce(ee.Reducer.sum());

var TDOMMask = zScore.lt(zScoreThresh).reduce(ee.Reducer.sum()).eq(2)

.and(irSum.lt(shadowSumThresh));

TDOMMask = TDOMMask.focal\_min(contractPixels).focal\_max(dilatePixels);

return img.updateMask(TDOMMask.not());

});

return collection;

}

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

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

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

*//Get Landsat data*

var l4s = ee.ImageCollection(collectionDict['L4'+ toaOrSR])

.filterDate(startDate,endDate)

.filter(ee.Filter.calendarRange(startJulian,endJulian))

.filterBounds(fc)

.select(sensorBandDict['L4'+ toaOrSR],sensorBandNameDict[toaOrSR])

var l5s = ee.ImageCollection(collectionDict['L5'+ toaOrSR])

.filterDate(startDate,endDate)

.filter(ee.Filter.calendarRange(startJulian,endJulian))

.filterBounds(fc)

.select(sensorBandDict['L5'+ toaOrSR],sensorBandNameDict[toaOrSR])

var l8s = ee.ImageCollection(collectionDict['L8'+ toaOrSR])

.filterDate(startDate,endDate)

.filter(ee.Filter.calendarRange(startJulian,endJulian))

.filterBounds(fc)

.select(sensorBandDict['L8'+ toaOrSR],sensorBandNameDict[toaOrSR])

var ls;

var l7s

if(includeSLCOffL7){

print('Including All Landsat 7');

l7s = ee.ImageCollection(collectionDict['L7'+toaOrSR])

.filterDate(startDate,endDate)

.filter(ee.Filter.calendarRange(startJulian,endJulian))

.filterBounds(fc)

.select(sensorBandDict['L7'+ toaOrSR],sensorBandNameDict[ toaOrSR])

}else{

print('Only including SLC On Landat 7');

l7s= ee.ImageCollection(collectionDict['L7'+toaOrSR])

.filterDate(ee.Date.fromYMD(1998,1,1),ee.Date.fromYMD(2003,5,31))

.filterDate(startDate,endDate)

.filter(ee.Filter.calendarRange(startJulian,endJulian))

.filterBounds(fc)

.select(sensorBandDict['L8'+ toaOrSR],sensorBandNameDict[toaOrSR])

}

ls = ee.ImageCollection(l5s.merge(l8s))

.filterBounds(fc)

.sort('system:time\_start');

*//Make sure all bands have data*

ls = ls.map(function(img){

img = img.updateMask(img.mask().reduce(ee.Reducer.min()))

return img.multiply(multImageDict[toaOrSR]).copyProperties(img,['system:time\_start','SOLAR\_ZENITH\_ANGLE','SOLAR\_AZIMUTH\_ANGLE','CLOUD\_COVER'])});//for SR

*//Apply relevant cloud masking methods*

if(cloudcloudShadowMaskingMethod.toLowerCase() === 'cloudscoretdom' || cloudcloudShadowMaskingMethod.toLowerCase() === 'hybrid' || toaOrSR.toLowerCase() === 'toa'){

print('Running cloudScore');

*//Add cloudScore*

var ls = ls.map(function(img){

var cs = landsatCloudScore(img).rename(['cloudScore']);

return img.addBands(cs);

});

*//Find low cloud score pctl for each pixel to avoid comission errors*

var minCloudScore = ls.select(['cloudScore']).reduce(ee.Reducer.percentile([cloudScorePctl]));

*//Apply cloudScore*

var ls = ls.select('blue','green','red','nir','swir1','temp','swir2','pixel\_qa','cloudScore').map(function(img){

var cloudMask = img.select(['cloudScore']).lt(minCloudScore.add(cloudScoreThresh))

.focal\_max(contractPixels).focal\_min(dilatePixels).rename('cloudMask');

return img.updateMask(cloudMask)})

}

if((cloudcloudShadowMaskingMethod.toLowerCase() === 'fmask' ||cloudcloudShadowMaskingMethod.toLowerCase() === 'hybrid') && toaOrSR.toLowerCase() != 'toa'){

print('Extracting cFmask cloud masks')

ls = ls.map(cFmaskCloud);

}

if(cloudcloudShadowMaskingMethod.toLowerCase() === 'cloudscoretdom' || cloudcloudShadowMaskingMethod.toLowerCase() === 'hybrid' || toaOrSR.toLowerCase() === 'toa'){

print('Running TDOM');

*// Find and mask out dark outliers*

ls = simpleTDOM2(ls,zScoreThresh,shadowSumThresh,contractPixels,dilatePixels);

}

if((cloudcloudShadowMaskingMethod.toLowerCase() === 'fmask' ||cloudcloudShadowMaskingMethod.toLowerCase() === 'hybrid') && toaOrSR.toLowerCase() != 'toa'){

print('Extracting cFmask cloud shadow masks')

ls = ls.map(cFmaskCloudShadow);

}

print('ls',ls);

*//Reducing ls to prevent time out error, over 5000 features.*

*//var medianls = ls.reduce(ee.Reducer.median())*

*//print('ls',medianls); // reduction works but image bands are averaged*

var reducer = ee.Reducer.percentile([50]);//Reducer for compositing

var perc\_ls = ls.reduce(reducer)

var count = ls.select(['nir']).count();

Map.addLayer(count.clip(geometry),{bands: ['nir'],min:1,max:26},'Number of clear pixels');

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LOAD DATASETS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*//Map.addLayer(boxTOP)*

*// Rename band filenames in metadata from the output composite*

var composites = perc\_ls.select([0,1,2,3,4,5,6],['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7']);

*//print(composites,'collection of images');*

*// Mosaic the old composite (as image collection) into a new composite (as image)*

var landsat = composites.clip(fc);

*// Display full image composite*

Map.addLayer(landsat, {bands: ['B5', 'B4', 'B2'], min: 0.05, max: [0.3,0.4,0.4], gamma: 1.6}, 'Landsat', false);

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EXPORTING IMAGES*

*//var sq1 = fc;*

*//Split/clip image composite into regions for exporting*

*//var clip\_1 = landsat.clip(sq1);*

*//Export split image composite to drive*

*//Export.image(clip\_1, 'KAL\_2015\_Landsat', {'scale':30, 'maxPixels':1e13, 'region': geometry});*

Export.image.toAsset({

image:landsat,

scale:30,

region: fc,

maxPixels:1e13,

description: 'SUM\_Landsat'

});

*//Export count image*

*//Export.image(count.clip(geometry), 'count', {'scale':30, 'maxPixels':1e13, 'region': region: fc});*

Code for Sar preprocessing (link below)

<https://code.earthengine.google.com/cfb96b19308821b10ce87ec515af7731>

// Define area of interest

var region\_name = "sumatra";

var fc = ee.Geometry.Polygon(

[[[100.02923463812999, 2.559919448517347],

[100.02923463812999, -5.002440879053686],

[106.10467409125499, -5.002440879053686],

[106.10467409125499, 2.559919448517347]]], null, false);

Map.addLayer(fc, {color: '000000'}, 'region', false);

var saName = region\_name;

var crs = 'EPSG:4326'; // CRS = WGS84 (or use 'EPSG:32647' for WGS84/UTM Zone 47N)

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Getting HH and HV sigma naught bands\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/////SPECKLE FILTERING on HH band//////*

var collectionHH = ee.ImageCollection('JAXA/ALOS/PALSAR/YEARLY/SAR').select('HH');

*// Convert DN to gamma\_nought in natural values*

collectionHH = collectionHH.map(function(img) { return ee.Image(img).pow(2).divide(ee.Number(10.0).pow(8.3))});

print(collectionHH);

*//Get metadata of HH polarization for 2015*

var im1 = ee.Image(collectionHH.filterMetadata('system:index', 'equals', '2015').first());

*/////SPECKLE FILTERING on HV band//////*

var collectionHV = ee.ImageCollection('JAXA/ALOS/PALSAR/YEARLY/SAR').select('HV');

*// Convert DN to gamma\_nought in natural values*

collectionHV = collectionHV.map(function(img) { return ee.Image(img).pow(2).divide(ee.Number(10.0).pow(8.3))});

print(collectionHV);

*//Get metadata of HV polarization for 2015*

var im4 = ee.Image(collectionHV.filterMetadata('system:index', 'equals', '2015').first());

*// Functions to convert from/to dB*

function toNatural(img) {

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

}

function toDB(img) {

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

}

*///////// Function for the Refined Lee speckle filter////////////////*

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 its directional band, and is otherwise masked)*

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

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

*// And 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']]));

*//return(result);*

}

*// Generate a dB version of the unfiltered image*

var SUM\_Palsar\_unfilteredHH = toDB(im1)

Map.addLayer(SUM\_Palsar\_unfilteredHH.clip(fc), {min:-25.0, max:0.0}, 'SUM\_Palsar\_unfilteredHH', false);

*// Filter the image with Refined Lee filter*

var SUM\_Palsar\_filteredHH = toDB(RefinedLee(im1));

Map.addLayer(SUM\_Palsar\_filteredHH.clip(fc), {min:-25.0, max:0.0}, 'SUM\_Palsar\_filteredHH', false);

*//Generate a dB version of the unfiltered image*

var SUM\_Palsar\_unfilteredHV = toDB(im4);

Map.addLayer(SUM\_Palsar\_unfilteredHV.clip(fc), {min:-25.0, max:0.0}, 'SUM\_Palsar\_unfilteredHV', false);

*// Filter the image with Refined Lee filter*

var SUM\_Palsar\_filteredHV = toDB(RefinedLee(im4));

Map.addLayer(SUM\_Palsar\_filteredHV.clip(fc), {min:-25.0, max:0.0}, 'SUM\_Palsar\_filteredHV', false);

// \*PROCESS IMAGE ASSET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Export.image.toAsset(palsar\_clip,'Perak\_2015\_Palsar\_box',{'scale':30,'maxPixels':1e13,'crs':crs,'region': palsar\_clip});

//Export.image.toDrive({image:box, description:'Boxcar 3x3 filtered HH 2015', scale:30, region: fc});

Export.image.toAsset({image: SUM\_Palsar\_unfilteredHH, description: 'SUM\_Palsar\_unfilteredHH', scale: 25, region:fc

, maxPixels: 1e13});

Export.image.toAsset({image: SUM\_Palsar\_filteredHH, description: 'SUM\_Palsar\_filteredHH', scale: 25, region: fc, maxPixels: 1e13});

Export.image.toAsset({image: SUM\_Palsar\_unfilteredHV, description: 'SUM\_Palsar\_unfilteredHV', scale: 25, region: fc, maxPixels: 1e13});

Export.image.toAsset({image: SUM\_Palsar\_filteredHV, description: 'SUM\_Palsar\_filteredHV', scale: 25, region: fc, maxPixels: 1e13});

*//Export.table(metadataFC, saName+'\_Composites\_Metadata'); // Trouble running this under Tasks*

Code for Sar-only classification (link below)

<https://code.earthengine.google.com/a3ea96b7136e5abe0b44dad25c36d351>

*//Import asset///*

var BGC = ee.FeatureCollection("users/thuansarzynski/Indonesia/BGC"),

FOR = ee.FeatureCollection("users/thuansarzynski/Indonesia/FOR"),

FSM = ee.FeatureCollection("users/thuansarzynski/Indonesia/FSM"),

OPM = ee.FeatureCollection("users/thuansarzynski/Indonesia/OPM"),

URB = ee.FeatureCollection("users/thuansarzynski/Indonesia/URB"),

WTR = ee.FeatureCollection("users/thuansarzynski/Indonesia/WTR");

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEFINE RANDOM SEED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

var seed = 2015;

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEFINE EXTENT AND VIEW \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

var region\_name = "Sumatra";

var fc = ee.Geometry.Polygon(

[[[100.02923463812999, 2.559919448517347],

[100.02923463812999, -5.002440879053686],

[106.10467409125499, -5.002440879053686],

[106.10467409125499, 2.559919448517347]]], null, false);

Map.addLayer(fc, {color: '000000'}, 'Sumatra', false);

var saName = region\_name;

var crs = 'EPSG:4326'; // CRS = WGS84 (or use 'EPSG:32647' for WGS84/UTM Zone 47N)

*/\*\*\*\*\*\*\*\*\*\*\*\*\*LOAD DATASETS \*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//ALOS PALSAR*

*// Load PALSAR image assets*

var hh2015s = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_filteredHH'); // HH sigma0

var hh2015r = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_unfilteredHH'); // HH raw

var hv2015s = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_filteredHV'); // HV sigma0

var hv2015r = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_unfilteredHV'); // HV raw

//Click on the link below to get authorization to access assets: preprocessed Sar images

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_filteredHH>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_unfilteredHH>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_filteredHV>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_unfilteredHV>

*// Generate indices using raw channels*

*//(ratio, average, difference, normalized difference index, NL Index)*

var rt2015a = hh2015r.divide(hv2015r); *// ratio HH/HV*

var rt2015b = hv2015r.divide(hh2015r); *// ratio HV/HH*

var ave2015 = (hh2015r.add(hv2015r)).divide(2); *// average*

var dif2015 = hh2015r.subtract(hv2015r); *// difference*

var ndi2015 = (hh2015r.subtract(hv2015r)).divide(hh2015r.add(hv2015r)); *// normalized index*

var nli2015 = (hh2015r.multiply(hv2015r)).divide(hh2015r.add(hv2015r)); *// NL index*

*// Rescale floating point to integer*

var scaledhh2015 = hh2015s.expression('1000\*b("sum")').int32();

var scaledhv2015 = hv2015s.expression('1000\*b("sum")').int32();

*// Rename rescaled Sigma0 channels*

scaledhh2015 = scaledhh2015.rename('HH');

scaledhv2015 = scaledhv2015.rename('HV');

*// Stack rescaled Sigma0 channels*

var dual2015 = scaledhh2015.addBands(scaledhv2015);

*// Calculate GLCM texture measures*

var textureMeasures = ['HH\_asm', 'HH\_contrast', 'HH\_corr', 'HH\_var', 'HH\_idm', 'HH\_savg', 'HH\_ent', 'HH\_diss', 'HV\_asm', 'HV\_contrast', 'HV\_corr', 'HV\_var', 'HV\_idm', 'HV\_savg', 'HV\_ent', 'HV\_diss'];

var glcm2015 = dual2015.glcmTexture({size: 1, average: true }).select(textureMeasures); *// 3x3 kernel*

*/\*\*\*\*\*\*\*\*\*\*\*\*CREATE COMPOSITE STACK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// Rename band filenames in metadata*

hh2015s = hh2015s.rename('HH');

hv2015s = hv2015s.rename('HV');

rt2015a = rt2015a.rename('RT1');

rt2015b = rt2015b.rename('RT2');

ave2015 = ave2015.rename('AVE');

dif2015 = dif2015.rename('DIF');

ndi2015 = ndi2015.rename('NDI');

nli2015 = nli2015.rename('NLI');

*// Create image collection from images*

var stackSAR2015 = hh2015s.addBands(hv2015s).addBands(rt2015a).addBands(rt2015b).addBands(ave2015).addBands(dif2015).addBands(ndi2015).addBands(nli2015).addBands(glcm2015).float();

var bandsSAR = ['HH', 'HV', 'RT1', 'RT2', 'AVE', 'DIF', 'NDI', 'NLI', 'HH\_asm', 'HH\_contrast', 'HH\_corr', 'HH\_var', 'HH\_idm', 'HH\_savg', 'HH\_ent', 'HH\_diss', 'HV\_asm', 'HV\_contrast', 'HV\_corr', 'HV\_var', 'HV\_idm', 'HV\_savg', 'HV\_ent', 'HV\_diss'];

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEFINE REGION OF INTEREST\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//define the samples with the input features*

*/// give a land cover value (1-6) to each polygon or feature*

var MergeLandType;

var refList=[FOR,BGC,FSM,OPM,URB,WTR]

for(var i=0;i<refList.length;i++){

var EachLandType=refList[i].map(function(feature){

return feature.set({landcover:i+1});

});

if (i===0){MergeLandType=EachLandType}

if(i!==0){MergeLandType=MergeLandType.merge(EachLandType)}

}

print(MergeLandType);

*// Initialise random column and values for ROI feature collection*

var sumatraROI = MergeLandType;

sumatraROI = sumatraROI.randomColumn('random1', seed);

var train = sumatraROI.filter(ee.Filter.lte('random1', 0.7)); // 70% of ROI for training

var test = sumatraROI.filter(ee.Filter.gt('random1', 0.7)); // 30% of ROI for testing

*//Map.addLayer(train, {'color': '000000'}, 'ROI Train', true);*

*//Map.addLayer(test, {'color': 'FF0000'}, 'ROI Test', true);*

*// Initialise random column and values for ROI feature collection*

train = train.randomColumn('random', seed);

test = test.randomColumn('random', seed);

*// Create training ROIs from the image dataset*

var roiTrainSAR = stackSAR2015.select(bandsSAR).sampleRegions({

collection: train,

properties: ['landcover', 'random'],

scale: 25 });

*// Create testing ROIs from the image dataset*

var roiTestSAR = stackSAR2015.select(bandsSAR).sampleRegions({

collection: test,

properties: ['landcover', 'random'],

scale: 25 });

*// Partition the regions of interest into training and testing areas*

var trainingSAR = roiTrainSAR.filter(ee.Filter.lte('random', 0.7)); // sample 70% training region from ROI

var testingSAR = roiTestSAR.filter(ee.Filter.lte('random', 0.7)); // sample 70% testing region from ROI

*// Print number of regions of interest for training and testing at the console*

*//print('Training, SAR, n =', trainingSAR.aggregate\_count('.all'));*

*//print('Testing, SAR, n =', testingSAR.aggregate\_count('.all'));*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXECUTE CLASSIFICATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// SAR ONLY*

*// Classification using Random Forest algorithm*

var classifierSAR = ee.Classifier.randomForest(100,0,10,0.5,false,seed).train({

features: trainingSAR, classProperty: 'landcover', inputProperties: bandsSAR});

*// Classify the image Random Forest algorithm*

*// Classify the validation data*

var validationSAR = testingSAR.classify(classifierSAR);

*// Calculate accuracy metrics*

var emS = validationSAR.errorMatrix('landcover', 'classification'); // *Error matrix*

var oaS = emS.accuracy(); // *Overall accuracy*

var ksS = emS.kappa(); // *Kappa statistic*

var uaS = emS.consumersAccuracy().project([1]); *// Consumer's accuracy*

var paS = emS.producersAccuracy().project([0]); *// Producer's accuracy*

var f1S = (uaS.multiply(paS).multiply(2.0)).divide(uaS.add(paS)); // F1-statistic

*//print('Error Matrix, SAR: ', emS); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('Overall Accuracy, SAR: ', oaS); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('Kappa Statistic, SAR: ', ksS); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('User\'s Accuracy (rows), SAR:', uaS); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('Producer\'s Accuracy (cols), SAR:', paS); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('F1 Score, SAR: ', f1S); // User memory limit exceeded error so I export matrix on Google Drive*

var exportemS = ee.Feature(null, {matrix: emS.array()});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportemS),

description: 'Error\_Matrix\_sar',

fileFormat: 'CSV'

});

var exportoaS = ee.Feature(null, {matrix: oaS});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportoaS),

description: 'Overall\_accuracy\_sar',

fileFormat: 'CSV'

});

var exportksS = ee.Feature(null, {matrix: ksS});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportksS),

description: 'Kappa\_statistic\_sar',

fileFormat: 'CSV'

});

var exportuaS = ee.Feature(null, {matrix: uaS});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportuaS),

description: 'Users\_accuracy\_sar',

fileFormat: 'CSV'

});

var exportpaS = ee.Feature(null, {matrix: paS});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportpaS),

description: 'Producers\_accuracy\_sar',

fileFormat: 'CSV'

});

var exportf1S = ee.Feature(null, {matrix: f1S});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportf1S),

description: 'F1\_score\_sar',

fileFormat: 'CSV'

});

*// Classify the image Random Forest algorithm*

var classifiedSAR = stackSAR2015.select(bandsSAR).classify(classifierSAR);

*/// FILTER CLASSIFICATION*

var filteredSAR = classifiedSAR.reduceNeighborhood({

reducer: ee.Reducer.mode(),

kernel: ee.Kernel.square(1),

});

*/// DISPLAY RGB COMPOSITES*

*//Map.addLayer(hh2015s.addBands(hv2015s).addBands(rt2015a), {min: [-26.0033, -36.6691, 1], max: [-4.57395, -10.4865, 8]}, 'RGB 2015 PALSAR', false);*

*/// DISPLAY CLASSIFICATION*

*//Create a palette for displaying the classified images*

var palette = ['34b317', 'f7fa23', '4afa1e', 'fe4cf1', 'b0b8b4', '2f53f5'];

*//var palette darkgreen, yellow, lightgreen, pink, turquoise, darkblue*

*// Display classified image*

Map.addLayer(classifiedSAR, {min: 1, max: 6, palette: palette}, '2015 Classification, PALSAR', true); //slow to display

*// Layer error: user memory limit exceeded*

*// Display classified mode image*

*//Map.addLayer(filteredSAR, {min: 1, max: 6, palette: palette}, '2015 Mode, PALSAR', false); //very slow to display*

*// Define classification legend*

var colors = ['34b317', 'f7fa23', '4afa1e', 'fe4cf1', 'b0b8b4', '2f53f5'];

var names = ["Forest", "Bareground-Crop", "Forest-Shrub", "Oil palm", "Built-up area","Water"];

var legend = ui.Panel({style: {position: 'bottom-left'}});

legend.add(ui.Label({

value: "Land Cover Classification",

style: {

fontWeight: 'bold',

fontSize: '16px',

margin: '0 0 4px 0',

padding: '0px'

}

}));

*// Iterate classification legend entries*

var entry; for (var x = 0; x<6; x++){

entry = [ui.Label({style:{color:colors[x],margin: '0 0 4px 0'}, value: '•'}), *//don't kow how to write instead of '•' how to write a rectangle?*

ui.Label({

value: names[x],

style: {

margin: '0 0 4px 4px'

}

})

];

legend.add(ui.Panel(entry, ui.Panel.Layout.Flow('horizontal'))); }

*// Display classification legend*

Map.add(legend);

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXPORT CLASSIFIED IMAGES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// Classified images for SAR*

Export.image.toDrive({

image: classifiedSAR.uint8(),

description: 'Classification\_SAR\_2015',

folder: 'Google Earth Engine',

region: fc,

scale: 25,

maxPixels: 1000000000,

});

Export.image.toDrive({

image: filteredSAR.uint8(),

description: 'Classification\_Mode\_SAR\_2015',

folder: 'Google Earth Engine',

region: fc,

scale: 25,

maxPixels: 1000000000,

});

Code for Landsat – only classification (link below)

<https://code.earthengine.google.com/e7406d61ce5113a9c4f99f8d1ae974b7>

*/// Import assets ////*

var BGC = ee.FeatureCollection("users/thuansarzynski/Indonesia/BGC"),

FOR = ee.FeatureCollection("users/thuansarzynski/Indonesia/FOR"),

FSM = ee.FeatureCollection("users/thuansarzynski/Indonesia/FSM"),

OPM = ee.FeatureCollection("users/thuansarzynski/Indonesia/OPM"),

URB = ee.FeatureCollection("users/thuansarzynski/Indonesia/URB"),

WTR = ee.FeatureCollection("users/thuansarzynski/Indonesia/WTR");

*// Click on the link below to get authorization to access assets: sampled ROI*

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/URB>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/WTR>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/OPM>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/FSM>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/FOR>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/BGC>

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEFINE RANDOM SEED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

var seed = 2015;

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEFINE EXTENT AND VIEW \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

var region\_name = "Sumatra";

var fc = ee.Geometry.Polygon(

[[[100.02923463812999, 2.559919448517347],

[100.02923463812999, -5.002440879053686],

[106.10467409125499, -5.002440879053686],

[106.10467409125499, 2.559919448517347]]], null, false);

Map.addLayer(fc, {color: '000000'}, 'Sumatra', false);

var saName = region\_name;

var crs = 'EPSG:4326'; // CRS = WGS84 (or use 'EPSG:32647' for WGS84/UTM Zone 47N)

*/\*\*\*\*\*\*\*\*\*\*\*\*\*LOAD DATASETS \*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//LANDSAT -> click on the link below to access asset: preprocessed Landsat images*

var sumatra2015=ee.Image('users/thuansarzynski/Indonesia/SUM\_Landsat');

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Landsat>

*// Mosaic Landsat image asset*

var composite2015 = sumatra2015.select([0,1,2,3,4,5,6],['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B10']);

*// Indices from Landsat bands*

var ndvi2015 = composite2015.normalizedDifference(['B5', 'B4']); *// Normalised Difference Vegetation Index (NDVI)*

var lswi2015 = composite2015.normalizedDifference(['B5', 'B6']); *// Land Surface Water Index (LSWI)*

var ndti2015 = composite2015.normalizedDifference(['B6', 'B7']); *// Normalised Difference Till Index (NDTI)*

var stvi2015 = composite2015.expression('((b("B6") - b("B4")) / (b("B6") + b("B4") + 0.1)) \* (1.1 - (b("B7") / 2))'); *// Soil Adjusted Total Vegetation Index (SATVI)*

var evi2015 = composite2015.expression('2.5 \* ((b("B5") - b("B4")) / (b("B5") + 6 \* b("B4") - 7.5 \* b("B2") + 1))'); *// Enhanced Vegetation Index*

*/\*\*\*\*\*\*\*\*\*\*\*\*CREATE COMPOSITE STACK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// Rename band filenames in metadata*

ndvi2015 = ndvi2015.rename('NDVI');

lswi2015 = lswi2015.rename('LSWI');

ndti2015 = ndti2015.rename('NDTI');

stvi2015 = stvi2015.rename('SATVI');

evi2015 = evi2015.rename('EVI');

var stackLandsat2015 = composite2015.addBands(ndvi2015).addBands(lswi2015).addBands(ndti2015)

.addBands(stvi2015).addBands(evi2015);

var bandsLandsat = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B10', 'NDVI', 'LSWI', 'NDTI', 'SATVI', 'EVI'];

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEFINE REGION OF INTEREST\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//define the samples with the input features*

*/// give a land cover value (1-6) to each polygon or feature*

var MergeLandType;

var refList=[FOR,BGC,FSM,OPM,URB,WTR]

for(var i=0;i<refList.length;i++){

var EachLandType=refList[i].map(function(feature){

return feature.set({landcover:i+1});

});

if (i===0){MergeLandType=EachLandType}

if(i!==0){MergeLandType=MergeLandType.merge(EachLandType)}

}

print(MergeLandType);

*// Initialise random column and values for ROI feature collection*

var sumatraROI = MergeLandType;

sumatraROI = sumatraROI.randomColumn('random1', seed);

var train = sumatraROI.filter(ee.Filter.lte('random1', 0.7));

var test = sumatraROI.filter(ee.Filter.gt('random1', 0.7));

//Map.addLayer(train, {'color': '000000'}, 'ROI Train', true);

//Map.addLayer(test, {'color': 'FF0000'}, 'ROI Test', true);

*// Initialise random column and values for ROI feature collection*

train = train.randomColumn('random', seed);

test = test.randomColumn('random', seed);

*// Create training ROIs from the image dataset*

var roiTrainLandsat = stackLandsat2015.select(bandsLandsat).sampleRegions({

collection: train,

properties: ['landcover', 'random'],

scale: 30 });

*// Create testing ROIs from the image dataset*

var roiTestLandsat = stackLandsat2015.select(bandsLandsat).sampleRegions({

collection: test,

properties: ['landcover', 'random'],

scale: 30 });

*// Partition the regions of interest into training and testing areas*

var trainingLandsat = roiTrainLandsat.filter(ee.Filter.lte('random', 0.7));

var testingLandsat = roiTestLandsat.filter(ee.Filter.lte('random', 0.7));

*// Print number of regions of interest for training and testing at the console*

*//print('Training, Landsat, n =', trainingLandsat.aggregate\_count('.all'));*

*//print('Testing, Landsat, n =', testingLandsat.aggregate\_count('.all'));*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXECUTE CLASSIFICATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*// LANDSAT ONLY*

*// Classification using Random Forest algorithm*

var classifierLandsat = ee.Classifier.randomForest(100,0,10,0.5,false,seed).train({

features: trainingLandsat,

classProperty: 'landcover',

inputProperties: bandsLandsat });

*// Classify the validation data*

var validationLandsat = testingLandsat.classify(classifierLandsat);

*// Calculate accuracy metrics*

var emL = validationLandsat.errorMatrix('landcover', 'classification'); *// Error matrix*

var oaL = emL.accuracy(); *// Overall accuracy*

var ksL = emL.kappa(); *// Kappa statistic*

var uaL = emL.consumersAccuracy().project([1]); *// Consumer's accuracy*

var paL = emL.producersAccuracy().project([0]); *// Producer's accuracy*

var f1L = (uaL.multiply(paL).multiply(2.0)).divide(uaL.add(paL)); *// F1-statistic*

*//print('Error Matrix, Landsat: ', emL);*

*//print('Overall Accuracy, Landsat: ', oaL);*

*//print('Kappa Statistic, Landsat: ', ksL);*

*//print('User\'s Accuracy (rows), Landsat:', uaL);*

*//print('Producer\'s Accuracy (cols), Landsat:', paL);*

*//print('F1 Score, Landsat: ', f1L);*

var exportemL = ee.Feature(null, {matrix: emL.array()});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportemL),

description: 'Error\_Matrix\_landsat',

fileFormat: 'CSV'

});

var exportoaL = ee.Feature(null, {matrix: oaL});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportoaL),

description: 'Overall\_accuracy\_landsat',

fileFormat: 'CSV'

});

var exportksL = ee.Feature(null, {matrix: ksL});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportksL),

description: 'Kappa\_statistic\_landsat',

fileFormat: 'CSV'

});

var exportuaL = ee.Feature(null, {matrix: uaL});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportuaL),

description: 'Users\_accuracy\_landsat',

fileFormat: 'CSV'

});

var exportpaL = ee.Feature(null, {matrix: paL});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportpaL),

description: 'Producers\_accuracy\_landsat',

fileFormat: 'CSV'

});

var exportf1L = ee.Feature(null, {matrix: f1L});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportf1L),

description: 'F1\_score\_landsat',

fileFormat: 'CSV'

});

*// Classify the image Random Forest algorithm*

var classifiedLandsat = stackLandsat2015.select(bandsLandsat).classify(classifierLandsat);

*/// FILTER CLASSIFICATION*

var filteredLandsat = classifiedLandsat.reduceNeighborhood({

reducer: ee.Reducer.mode(),

kernel: ee.Kernel.square(1),

});

*/// DISPLAY RGB COMPOSITES*

Map.addLayer(composite2015, {bands: ['B5', 'B4', 'B3'], min: 0, max: 0.3}, 'RGB 2015 Landsat', false);

*/// DISPLAY CLASSIFICATION*

*//Create a palette for displaying the classified images*

var palette = ['34b317', 'f7fa23', '4afa1e', 'fe4cf1', 'b0b8b4', '2f53f5'];

*//var palette darkgreen, yellow, lightgreen, pink, turquoise, darkblue*

*// Display classified image*

Map.addLayer(classifiedLandsat, {min: 1, max: 6, palette: palette}, '2015 Classification, Landsat', true); *//slow to display*

*// Layer error: user memory limit exceeded*

*// Display classified mode image*

Map.addLayer(filteredLandsat, {min: 1, max: 6, palette: palette}, '2015 Mode, Landsat', false); //very slow to display

*// Layer error: user memory limit exceeded*

*// Define classification legend*

var colors = ['34b317', 'f7fa23', '4afa1e', 'fe4cf1', 'b0b8b4', '2f53f5'];

var names = ["Forest", "Bareground-Crop", "Forest-Shrub", "Oil palm", "Built-up area","Water"];

var legend = ui.Panel({style: {position: 'bottom-left'}});

legend.add(ui.Label({

value: "Land Cover Classification",

style: {

fontWeight: 'bold',

fontSize: '16px',

margin: '0 0 4px 0',

padding: '0px'

}

}));

*// Iterate classification legend entries*

var entry; for (var x = 0; x<6; x++){

entry = [

ui.Label({style:{color:colors[x],margin: '0 0 4px 0'}, value: '•'}), //don't kow how to write instead of '•' how to write a rectangle?

ui.Label({

value: names[x],

style: {

margin: '0 0 4px 4px'

}

})

];

legend.add(ui.Panel(entry, ui.Panel.Layout.Flow('horizontal'))); }

*// Display classification legend*

Map.add(legend);

Export.image.toDrive({

image: classifiedLandsat.uint8(),

description: 'Classification\_Landsat\_2015',

folder: 'Google Earth Engine',

region: fc,

scale: 30,

maxPixels: 1000000000,

});

Export.image.toDrive({

image: filteredLandsat.uint8(),

description: 'Classification\_Landsat\_2015\_mode',

folder: 'Google Earth Engine',

region: fc,

scale: 30,

maxPixels: 1000000000,

});

Code for Combined Landsat x Sar classification (link below)

<https://code.earthengine.google.com/aad21e1ace34ca89ae2e75cef2a740b0>

*/// Import assets ////*

var BGC = ee.FeatureCollection("users/thuansarzynski/Indonesia/BGC"),

FOR = ee.FeatureCollection("users/thuansarzynski/Indonesia/FOR"),

FSM = ee.FeatureCollection("users/thuansarzynski/Indonesia/FSM"),

OPM = ee.FeatureCollection("users/thuansarzynski/Indonesia/OPM"),

URB = ee.FeatureCollection("users/thuansarzynski/Indonesia/URB"),

WTR = ee.FeatureCollection("users/thuansarzynski/Indonesia/WTR");

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEFINE RANDOM SEED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

var seed = 2015;

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DEFINE EXTENT AND VIEW \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

var region\_name = "Sumatra";

var fc = ee.Geometry.Polygon(

[[[100.02923463812999, 2.559919448517347],

[100.02923463812999, -5.002440879053686],

[106.10467409125499, -5.002440879053686],

[106.10467409125499, 2.559919448517347]]], null, false);

*//Map.addLayer(fc, {color: '000000'}, 'Sumatra', false);*

var saName = region\_name;

var crs = 'EPSG:4326'; // CRS = WGS84 (or use 'EPSG:32647' for WGS84/UTM Zone 47N)j

*/\*\*\*\*\*\*\*\*\*\*\*\*\*LOAD DATASETS \*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//LANDSAT 🡪 click on the link to access asset: preprocessed Landsat image.*

var sumatra2015=ee.Image('users/thuansarzynski/Indonesia/SUM\_Landsat');

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Landsat>

*// Mosaic Landsat image asset*

var composite2015 = sumatra2015.select([0,1,2,3,4,5,6],['B2', 'B3', 'B4', 'B5', 'B6', 'B10', 'B7']);

*// Indices from Landsat bands*

var ndvi2015 = composite2015.normalizedDifference(['B5', 'B4']); *// Normalised Difference* Vegetation Index (NDVI)

var lswi2015 = composite2015.normalizedDifference(['B5', 'B6']); *// Land Surface Water Index (LSWI)*

var ndti2015 = composite2015.normalizedDifference(['B6', 'B7']); *// Normalised Difference Till Index (NDTI)*

var stvi2015 = composite2015.expression('((b("B6") - b("B4")) / (b("B6") + b("B4") + 0.1)) \* (1.1 - (b("B7") / 2))'); *// Soil Adjusted Total Vegetation Index (SATVI)*

var evi2015 = composite2015.expression('2.5 \* ((b("B5") - b("B4")) / (b("B5") + 6 \* b("B4") - 7.5 \* b("B2") + 1))'); *// Enhanced Vegetation Index*

*//ALOS PALSAR click on the link below to access asset: preprocessed Sar image.*

*// Load PALSAR image assets*

var hh2015s = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_filteredHH'); *// HH sigma0*

var hh2015r = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_unfilteredHH'); *// HH raw*

var hv2015s = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_filteredHV'); *// HV sigma0*

var hv2015r = ee.Image('users/thuansarzynski/Indonesia/SUM\_Palsar\_unfilteredHV'); *// HV raw*

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_filteredHH>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_filteredHV>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_unfilteredHH>

<https://code.earthengine.google.com/?asset=users/thuansarzynski/Indonesia/SUM_Palsar_unfilteredHV>

*// Generate indices using raw channels*

*//(ratio, average, difference, normalized difference index, NL Index)*

var rt2015a = hh2015r.divide(hv2015r); *// ratio HH/HV*

var rt2015b = hv2015r.divide(hh2015r); *// ratio HV/HH*

var ave2015 = (hh2015r.add(hv2015r)).divide(2); *// average*

var dif2015 = hh2015r.subtract(hv2015r); *// difference*

var ndi2015 = (hh2015r.subtract(hv2015r)).divide(hh2015r.add(hv2015r)); *// normalized index*

var nli2015 = (hh2015r.multiply(hv2015r)).divide(hh2015r.add(hv2015r)); *// NL index*

*// Rescale floating point to integer*

var scaledhh2015 = hh2015s.expression('1000\*b("sum")').int32();

var scaledhv2015 = hv2015s.expression('1000\*b("sum")').int32();

*// Rename rescaled Sigma0 channels*

scaledhh2015 = scaledhh2015.rename('HH');

scaledhv2015 = scaledhv2015.rename('HV');

*// Stack rescaled Sigma0 channels*

var dual2015 = scaledhh2015.addBands(scaledhv2015);

*// Calculate GLCM texture measures*

var textureMeasures = ['HH\_asm', 'HH\_contrast', 'HH\_corr', 'HH\_var', 'HH\_idm', 'HH\_savg', 'HH\_ent', 'HH\_diss',

'HV\_asm', 'HV\_contrast', 'HV\_corr', 'HV\_var', 'HV\_idm', 'HV\_savg', 'HV\_ent', 'HV\_diss'];

var glcm2015 = dual2015.glcmTexture({size: 1, average: true }).select(textureMeasures); // 3x3 kernel

*/\*\*\*\*\*\*\*\*\*\*\*\*CREATE COMPOSITE STACK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// Rename band filenames in metadata*

ndvi2015 = ndvi2015.rename('NDVI');

lswi2015 = lswi2015.rename('LSWI');

ndti2015 = ndti2015.rename('NDTI');

stvi2015 = stvi2015.rename('SATVI');

evi2015 = evi2015.rename('EVI');

hh2015s = hh2015s.rename('HH');

hv2015s = hv2015s.rename('HV');

rt2015a = rt2015a.rename('RT1');

rt2015b = rt2015b.rename('RT2');

ave2015 = ave2015.rename('AVE');

dif2015 = dif2015.rename('DIF');

ndi2015 = ndi2015.rename('NDI');

nli2015 = nli2015.rename('NLI');

*// Create image collection from images*

var stackCombined2015 = composite2015.addBands(ndvi2015).addBands(lswi2015).addBands(ndti2015)

.addBands(stvi2015).addBands(evi2015).addBands(hh2015s).addBands(hv2015s).addBands(rt2015a)

.addBands(rt2015b).addBands(ave2015).addBands(dif2015).addBands(ndi2015).addBands(nli2015)

.addBands(glcm2015).float();

var bandsCombined = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B10', 'NDVI', 'LSWI', 'NDTI', 'SATVI',

'EVI','HH', 'HV', 'RT1', 'RT2', 'AVE', 'DIF', 'NDI', 'NLI',

'HH\_asm', 'HH\_contrast', 'HH\_corr', 'HH\_var', 'HH\_idm', 'HH\_savg', 'HH\_ent', 'HH\_diss',

'HV\_asm', 'HV\_contrast', 'HV\_corr', 'HV\_var', 'HV\_idm', 'HV\_savg', 'HV\_ent', 'HV\_diss'];

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*DEFINE REGION OF INTEREST\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//define the samples with the input features*

*/// give a land cover value (1-6) to each polygon or feature*

var MergeLandType;

var refList=[FOR,BGC,FSM,OPM,URB,WTR]

for(var i=0;i<refList.length;i++){

var EachLandType=refList[i].map(function(feature){

return feature.set({landcover:i+1});

});

if (i===0){MergeLandType=EachLandType}

if(i!==0){MergeLandType=MergeLandType.merge(EachLandType)}

}

print(MergeLandType);

*// Initialise random column and values for ROI feature collection*

var sumatraROI = MergeLandType;

*// Initialise random column and values for ROI feature collection*

sumatraROI = sumatraROI.randomColumn('random1', seed);

var train = sumatraROI.filter(ee.Filter.lte('random1', 0.7));

var test = sumatraROI.filter(ee.Filter.gt('random1', 0.7));

*//Map.addLayer(train, {'color': '000000'}, 'ROI Train', true);*

*//Map.addLayer(test, {'color': 'FF0000'}, 'ROI Test', true);*

*// Initialise random column and values for ROI feature collection*

train = train.randomColumn('random', seed);

test = test.randomColumn('random', seed);

*// Create training ROIs from the image dataset*

var roiTrainCombined = stackCombined2015.select(bandsCombined).sampleRegions({

collection: train,

properties: ['landcover', 'random'],

scale: 30 });

*// Create testing ROIs from the image dataset*

var roiTestCombined = stackCombined2015.select(bandsCombined).sampleRegions({

collection: test,

properties: ['landcover', 'random'],

scale: 30 });

*// Partition the regions of interest into training and testing areas*

var trainingCombined = roiTrainCombined.filter(ee.Filter.lte('random', 0.7));

var testingCombined = roiTestCombined.filter(ee.Filter.lte('random', 0.7));

*// Print number of regions of interest for training and testing at the console*

*//print('Training, Combined, n =', trainingCombined.aggregate\_count('.all'));*

*//print('Testing, Combined, n =', testingCombined.aggregate\_count('.all'));*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXECUTE CLASSIFICATION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// COMBINED LANDSAT+SAR*

*// Classification using Random Forest algorithm*

var classifierCombined = ee.Classifier.randomForest(100,0,10,0.5,false,seed).train({

features: trainingCombined,classProperty:'landcover',

inputProperties: bandsCombined });

*// Classify the validation data*

var validationCombined = testingCombined.classify(classifierCombined);

*// Calculate accuracy metrics*

var emC = validationCombined.errorMatrix('landcover', 'classification'); *// Error matrix*

var oaC = emC.accuracy(); *// Overall accuracy*

var ksC = emC.kappa(); *// Kappa statistic*

var uaC = emC.consumersAccuracy().project([1]); *// Consumer's accuracy*

var paC = emC.producersAccuracy().project([0]); *// Producer's accuracy*

var f1C = (uaC.multiply(paC).multiply(2.0)).divide(uaC.add(paC)); *// F1-statistic*

*//print('Error Matrix, Combined: ', emC); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('Overall Accuracy, Combined: ', oaC); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('Kappa Statistic, Combined: ', ksC); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('User\'s Accuracy (rows), Combined:', uaC); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('Producer\'s Accuracy (cols), Combined:', paC); // User memory limit exceeded error so I export matrix on Google Drive*

*//print('F1 Score, Combined: ', f1C); // User memory limit exceeded error so I export matrix on Google Drive*

var exportemC = ee.Feature(null, {matrix: emC.array()});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportemC),

description: 'Error\_Matrix\_comb',

fileFormat: 'CSV'});

var exportoaC = ee.Feature(null, {matrix: oaC});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportoaC),

description: 'Overall\_accuracy\_comb',

fileFormat: 'CSV'});

var exportksC = ee.Feature(null, {matrix: ksC});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportksC),

description: 'Kappa\_statistic\_comb',

fileFormat: 'CSV'});

var exportuaC = ee.Feature(null, {matrix: uaC});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportuaC),

description: 'Users\_accuracy\_comb',

fileFormat: 'CSV'});

var exportpaC = ee.Feature(null, {matrix: paC});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportpaC),

description: 'Producers\_accuracy\_comb',

fileFormat: 'CSV'});

var exportf1C = ee.Feature(null, {matrix: f1C});

*// Export the FeatureCollection to have a longer computation and memory space*

Export.table.toDrive({

collection: ee.FeatureCollection(exportf1C),

description: 'F1\_score\_comb',

fileFormat: 'CSV'});

*// Classify the image Random Forest algorithm*

var classifiedCombined = stackCombined2015.select(bandsCombined).classify(classifierCombined);

*/// FILTER CLASSIFICATION*

var filteredCombined = classifiedCombined.reduceNeighborhood({

reducer: ee.Reducer.mode(),

kernel: ee.Kernel.square(1),

});

*/// DISPLAY RGB COMPOSITES*

Map.addLayer(hh2015s.addBands(hv2015s).addBands(rt2015a), {min: [-26.0033, -36.6691, 1], max: [-4.57395, -10.4865, 8]}, 'RGB 2015 Combined', false);

*/// DISPLAY CLASSIFICATION*

*// Display classified image*

*//Create a palette for displaying the classified images*

var palette = ['34b317', 'f7fa23', '4afa1e', 'fe4cf1', 'b0b8b4', '2f53f5'];

*//var palette darkgreen, yellow, lightgreen, pink, turquoise, darkblue*

*// Display classified image*

Map.addLayer(classifiedCombined, {min: 1, max: 6, palette: palette}, '2015 Classification, Combined', true); //slow to display

*// Layer error: user memory limit exceeded*

*// Display classified mode image*

Map.addLayer(filteredCombined, {min: 1, max: 6, palette: palette}, '2015 Mode, Combined', false); *//very slow to display*

*// Layer error: user memory limit exceeded*

*// Define classification legend*

var colors = ['34b317', 'f7fa23', '4afa1e', 'fe4cf1', 'b0b8b4', '2f53f5'];

var names = ["Forest", "Bareground-Crop", "Forest-Shrub", "Oil palm", "Built-up area","Water"];

var legend = ui.Panel({style: {position: 'bottom-left'}});

legend.add(ui.Label({

value: "Land Cover Classification",

style: {

fontWeight: 'bold',

fontSize: '16px',

margin: '0 0 4px 0',

padding: '0px'

}

}));

*// Iterate classification legend entries*

var entry; for (var x = 0; x<6; x++){

entry = [

ui.Label({style:{color:colors[x],margin: '0 0 4px 0'}, value: '•'}), //don't kow how to write instead of '•' how to write a rectangle?

ui.Label({

value: names[x],

style: {

margin: '0 0 4px 4px'

}

})

];

legend.add(ui.Panel(entry, ui.Panel.Layout.Flow('horizontal'))); }

*// Display classification legend*

Map.add(legend);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXPORT CLASSIFIED IMAGES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

*// Classified images for Combined Landsat+SAR*

Export.image.toDrive({

image: classifiedCombined.uint8(),

description: 'Classification\_Combined\_2015',

folder: 'Google Earth Engine',

region: fc,

scale: 30,

maxPixels: 1000000000,

});

Export.image.toDrive({

image: filteredCombined.uint8(),

description: 'Classification\_Combined\_Mode\_2015',

folder: 'Google Earth Engine',

region: fc,

scale: 30,

maxPixels: 1000000000,

});