Publication: Doggart, N., Mugasha, W.A., Mpiri, A., Morgan-Brown, T., Sallu, S.M., Spracklen, D.V., 2023. Agricultural fallows are the main driver of natural forest regeneration in Tanzania. Environmental Research Letters.

Script Author: T. Morgan-Brown

This Google Earth Engine script is used to map areas of regeneration and was used to identify the field survey sample plots in the ten field survey sampling clusters.

//Thresholds

var min\_for\_hv = '2100'; // minimum HV for forest in 2020

var afterThresh = '2100'; // minimum HV for forest in change year

var minGain = '0.85'; // minimum relative decrease in HV for deforestation

var geometry = image.geometry();

var extent = geometry.bounds();

var P1years = ee.List.sequence(2007, 2010); //PALSAR 1 from 2007 - 2010

print(P1years);

var P2years1 = ee.List.sequence(2015, 2017); //PALSAR 1 from 2007 - 2010

var P2years2 = ee.List.sequence(2019, 2020); //PALSAR 2 from 2019 - 2020

// Load later Sentinel 2 for reference and sort by cloud cover

// note palsar mosaics are mostly dry season and less cloud cover usually means dry season images

// ensure the later date chosen above is after Sentinel 2 imagery becomes available

var sentCollection = ee.ImageCollection('COPERNICUS/S2').filterBounds(extent).sort('CLOUDY\_PIXEL\_PERCENTAGE',false);

var sent = sentCollection.filterDate({

start: ee.Date.fromYMD(2020, 1, 1),

end: ee.Date.fromYMD(2020,5,30)

}).mosaic().clip(extent);

var sentVis = {

bands : ['B8','B3','B2'] ,

min: [1576, 597, 766],

max: [3971, 1331, 1347],

};

Map.addLayer(sent, sentVis, 'Sentinel2',0);

var p = ee.ImageCollection('JAXA/ALOS/PALSAR/YEARLY/SAR')

.filterBounds(extent)

.select('HV');

print(p);

var HV\_col = p.map(RefinedLee);

print(HV\_col);

var HV2017 = HV\_col.filter(ee.Filter.eq("system:index", '2017')).first();

var HV2018 = HV\_col.filter(ee.Filter.eq("system:index", '2018')).first();

var HV2019 = HV\_col.filter(ee.Filter.eq("system:index", '2019')).first();

// Use bicubic resampling during registration.

var HV2017r = HV2017.resample('bicubic');

var HV2018r = HV2018.resample('bicubic');

var HV2019r = HV2019.resample('bicubic');

// Choose to register using only the 'R' band.

var HV2017r = HV2017r.clip(extent);

var HV2018r = HV2018r.clip(extent);

var HV2019r = HV2019r.clip(extent);

// Determine the displacement by matching only the 'R' bands.

var displacement = HV2017r.displacement({

referenceImage: HV2019r,

maxOffset: 100.0,

patchWidth: 100.0

});

// Determine the displacement by matching only the 'R' bands.

var displacement1 = HV2018r.displacement({

referenceImage: HV2019r,

maxOffset: 100.0,

patchWidth: 100.0

});

var HV2007 = HV\_col.filter(ee.Filter.eq("system:index", '2007')).first().displace(displacement);

var HV2008 = HV\_col.filter(ee.Filter.eq("system:index", '2008')).first().displace(displacement);

var HV2009 = HV\_col.filter(ee.Filter.eq("system:index", '2009')).first().displace(displacement);

var HV2010 = HV\_col.filter(ee.Filter.eq("system:index", '2010')).first().displace(displacement);

var HV2015 = HV\_col.filter(ee.Filter.eq("system:index", '2015')).first().displace(displacement);

var HV2016 = HV\_col.filter(ee.Filter.eq("system:index", '2016')).first().displace(displacement);

var HV2017 = HV\_col.filter(ee.Filter.eq("system:index", '2017')).first().displace(displacement);

var HV2018 = HV\_col.filter(ee.Filter.eq("system:index", '2018')).first().displace(displacement1);

var HV2019 = HV\_col.filter(ee.Filter.eq("system:index", '2019')).first();

var HV2020 = HV\_col.filter(ee.Filter.eq("system:index", '2020')).first();

var HVlist = [HV2007, HV2008, HV2009, HV2010, HV2015, HV2016, HV2017, HV2018, HV2019];

var HV\_col = ee.ImageCollection.fromImages(HVlist);

var forest = HV2020.gte(ee.Number.parse(min\_for\_hv)).and(image.eq(1));

var forest\_sieved = forest.eq(1).and(forest.connectedPixelCount(125).gt(7));

var end\_forest = forest\_sieved.add((forest.connectedPixelCount(125).lt(8)).and(forest.eq(0)));

function map\_gain(img) {

var ratio = img.divide(HV2020);

var gain = img.lt(ee.Number.parse(min\_for\_hv)).and(end\_forest.eq(1)).and(ratio.lte(ee.Number.parse(minGain)));

var gain\_sieved = gain.eq(1).and(gain.connectedPixelCount(125, false).gt(7));

return(gain\_sieved);

}

// Define a palette for the change map.

var palette = [

'ffffff', // Non-Forest (0)

'68d86a', // FOREST (1)

'ffff00', // Loss 2007 (2)

'ffe62e', // Loss 2008 (2)

'fecc5c', // Loss 2009 (2)

'fea447', // Loss 2010 (2)

'fd7b33', // Loss 2015 (3)

'fe3d19', // Loss 2016 (4)

'ff0000', // Loss 2017 (5)

'ca000b', // Loss 2018 (6)

'950016' // Loss 2019 (7)

];

// Define a palette for the Forest Map.

var palette\_FNF = [

'ffffff', // Non-Forest (0)

'68d86a', // FOREST (1)

];

// Define a palette for the loss map.

var palette\_loss = [

'ffffff', // Non-change (0)

'ff0000', // Loss (1)

];

//Map.addLayer(ee.Image(nonforest), {min: 0, max: 1, palette: palette\_loss}, 'Gain\_since\_2007', 0);

var gain\_col = HV\_col.map(map\_gain).toList(9);

print(gain\_col);

var gain\_map = end\_forest

.where(ee.Image(gain\_col.get(0)).eq(1), 2)

.where(ee.Image(gain\_col.get(1)).eq(1), 3)

.where(ee.Image(gain\_col.get(2)).eq(1), 4)

.where(ee.Image(gain\_col.get(3)).eq(1), 5)

.where(ee.Image(gain\_col.get(4)).eq(1), 6)

.where(ee.Image(gain\_col.get(5)).eq(1), 7)

.where(ee.Image(gain\_col.get(6)).eq(1), 8)

.where(ee.Image(gain\_col.get(7)).eq(1), 9)

.where(ee.Image(gain\_col.get(8)).eq(1), 10);

var gain\_map\_simp = end\_forest

.where(ee.Image(gain\_col.get(0)).eq(1), 2)

.where(ee.Image(gain\_col.get(1)).eq(1), 2)

.where(ee.Image(gain\_col.get(2)).eq(1), 2)

.where(ee.Image(gain\_col.get(3)).eq(1), 2)

.where(ee.Image(gain\_col.get(4)).eq(1), 3)

.where(ee.Image(gain\_col.get(5)).eq(1), 3)

.where(ee.Image(gain\_col.get(6)).eq(1), 3)

.where(ee.Image(gain\_col.get(7)).eq(1), 3)

.where(ee.Image(gain\_col.get(8)).eq(1), 3);

var strat\_samp = gain\_map.stratifiedSample({numPoints: 6, geometries: true});

//var geom = strat\_samp.geometry();

//var geom2 = geom.buffer(15);

//var pixelOut = image.paint(geom2,1,1);

//Map.addLayer(pixelOut,{'palette': '98ff00'},'points',0);

// Export an ee.FeatureCollection as an Earth Engine asset.

//Export.table.toAsset({

// collection: strat\_samp,

// description:'export\_sample\_to\_asset',

// assetId: 'Mvomero\_strat\_sample',

//});

// Export the image, specifying scale and region.

Export.image.toAsset({

image: gain\_map\_simp,

description: 'Mvomero\_Gain\_PAL\_2007-2020\_HVthres\_2100',

scale: 25,

region: extent,

maxPixels: 1e9

});

// Display the classification result and the input image.

Map.addLayer(ee.Image(gain\_map), {min: 0, max: 10, palette: palette}, 'Lindi\_Gain\_PAL\_2010-2020', 0);

//Map.addLayer(ee.Image(loss\_col.get(5)), {min: 0, max: 1, palette: palette\_loss}, 'loss\_2020', 0);

// Export the image, specifying scale and region.

Export.image.toDrive({

image: gain\_map,

description: 'Mvomero\_Gain\_PAL\_2007-2020\_HVthres\_2100',

scale: 25,

region: extent,

maxPixels: 1e9

});

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)));

// Convert the array image to a scaler image.

result = result.arrayGet(0);

return(result);