# Part 1. Assessing, filtering, and export MapBiomas data to Google Drive using Google Earth Engine

## // First, upload a shapefile of your region of interest and name the object as “ROI”

## // Loading data from MapBiomas Collection 6.0, clip it to your region of interest, and keep only class 3 (“forest formation”)

var landCover = ee.Image('projects/mapbiomas-workspace/public/collection6/mapbiomas\_collection60\_integration\_v1')

 .clip(ROI).eq(3);

## // Since MapBiomas data comes in ee.Image format (where each band represents data from annual land cover classifications), convert it to ee.ImageCollection

landCover = ee.ImageCollection([landCover.select('classification\_1985'), landCover.select('classification\_1986'),

 landCover.select('classification\_1987'), landCover.select('classification\_1988'),

 landCover.select('classification\_1989'), landCover.select('classification\_1990'),

 landCover.select('classification\_1991'), landCover.select('classification\_1992'),

 landCover.select('classification\_1993'), landCover.select('classification\_1994'),

 landCover.select('classification\_1995'), landCover.select('classification\_1996'),

 landCover.select('classification\_1997'), landCover.select('classification\_1998'),

 landCover.select('classification\_1999'), landCover.select('classification\_2000'),

 landCover.select('classification\_2001'), landCover.select('classification\_2002'),

 landCover.select('classification\_2003'), landCover.select('classification\_2004'),

 landCover.select('classification\_2005'), landCover.select('classification\_2006'),

 landCover.select('classification\_2007'), landCover.select('classification\_2008'),

 landCover.select('classification\_2009'), landCover.select('classification\_2010'),

 landCover.select('classification\_2011'), landCover.select('classification\_2012'),

 landCover.select('classification\_2013'), landCover.select('classification\_2014'),

 landCover.select('classification\_2015'), landCover.select('classification\_2016'),

 landCover.select('classification\_2017'), landCover.select('classification\_2018'),

 landCover.select('classification\_2019'), landCover.select('classification\_2020')]);

## // Map it through the ee.ImageCollection to assign non-values to areas not classified as forests

landCover = landCover.map(function(img){

 return(img.updateMask(img.eq(1)));

});

## // You may want to check your collection by printing it and adding the first image to the map

//print(landCover)

//Map.addLayer(landCover.first())

## // Batch download all images

var batch = require('users/fitoprincipe/geetools:batch');

batch.Download.ImageCollection.toDrive(landCover, 'GEE', {

 name: 'mapbiomas\_{id}',

 scale: 30,

 region: roi,

 maxPixels: 1e13,

 crs: 'EPSG:4326',

 fileFormat: "GeoTIFF",

 type: 'float'

});

# Part 2. Assigning individual numbers and calculating size of forest patches using R

## ##### Function from the original clumpSize function in bFastSpatial package by Ben DeVries #####

## ##### GitHub: <https://github.com/loicdtx/bfastSpatial> #####

clumpSize <- function(x, f=1, stats=FALSE, ...){

 # Identify the clumps

 y <- clump(x, directions=4, ...)

 # Make a reclassification matrix from the frequency table of y

 rcl <- freq(y)

 # Conversion to other unit if factor is supplied

 if(f != 1)

 rcl[, 2] <- rcl[, 2] \* f

 # Optional: make a summary matrix

 if(stats){

 sumstat <- matrix(nc=1, nr=6, dimnames=list(c("Mean", "Min.", "1st Qu.", "Median", "3rd Qu.", "Max."), c("clump size")))

 sizes <- rcl[!is.na(rcl[, 1]), 2]

 sumstat[1, 1] <- mean(sizes)

 sumstat[c(2:6), 1] <- quantile(sizes)

 }

 # Reclassify y based on rcl

 rcl <- cbind(rcl[,1], rcl) # double first column (see ?reclassify)

 z <- reclassify(y, rcl=rcl, right=NA)

 # Remove NAs

 z[is.na(y)] <- NA

 # make z into a list if stats=TRUE

 if(stats)

 z <- list(clumps = z, stats = sumstat)

 return(stack(y, z))

}

## ##### Loading packages and setting preferences #####

library(raster)

rasterOptions(maxmemory=1e+06, chunksize=1e+07, progress = 'text'); message("Memory preferences set.")

## ##### Loading directories of the folder with the forest cover classification images #####

(files <- list.files(path="pathtoyourdata", pattern=".tif", full.names=TRUE))

## ##### Stacking #####

forestCover <- stack(files)

## ##### Applying the function #####

for(i in 1:nlayers(forestCover)){

 forestLayer <- clumpSize(forestCover[[i]])

 writeRaster(forestLayer, paste0('pathtosave', 'clump\_', names(forestCover[[i]]), '.tif'), format="GTiff")

 rm(forestLayer)

 #gc() # use it if you are running out of memory

 #Sys.sleep(10) # use it and adjust the time in seconds if you are running out of memory

}

# Part 3. Calculating metrics and investigating the spatial distribution of forests using Google Earth Engine

## /\*

## First, upload the images generated in the previous R script to GEE, load them into your workspace, and name them

## as "image1", "image2", ..., "image36". The first band of the images (named 'b1') should be representative of the

## unique IDs assigned to each clump in R. The second band (named 'b2') should represent the clump size in number

## of pixels. Then, load your region of interest and name it as "ROI". Lastly, load the municipalities shapefile

## (or another shapefile representative of the scale you want to capture the processes related to forest dynamics) and

## name it as "features".

## \*/

## // Adding data as one of the properties of your images

image1 = image1.set('system:time\_start', ee.Date('1985')); image2 = image2.set('system:time\_start', ee.Date('1986'));

image3 = image3.set('system:time\_start', ee.Date('1987')); image4 = image4.set('system:time\_start', ee.Date('1988'));

image5 = image5.set('system:time\_start', ee.Date('1989')); image6 = image6.set('system:time\_start', ee.Date('1990'));

image7 = image7.set('system:time\_start', ee.Date('1991')); image8 = image8.set('system:time\_start', ee.Date('1992'));

image9 = image9.set('system:time\_start', ee.Date('1993')); image10 = image10.set('system:time\_start', ee.Date('1994'));

image11 = image11.set('system:time\_start', ee.Date('1995')); image12 = image12.set('system:time\_start', ee.Date('1996'));

image13 = image13.set('system:time\_start', ee.Date('1997')); image14 = image14.set('system:time\_start', ee.Date('1998'));

image15 = image15.set('system:time\_start', ee.Date('1999')); image16 = image16.set('system:time\_start', ee.Date('2000'));

image17 = image17.set('system:time\_start', ee.Date('2001')); image18 = image18.set('system:time\_start', ee.Date('2002'));

image19 = image19.set('system:time\_start', ee.Date('2003')); image20 = image20.set('system:time\_start', ee.Date('2004'));

image21 = image21.set('system:time\_start', ee.Date('2005')); image22 = image22.set('system:time\_start', ee.Date('2006'));

image23 = image23.set('system:time\_start', ee.Date('2007')); image24 = image24.set('system:time\_start', ee.Date('2008'));

image25 = image25.set('system:time\_start', ee.Date('2009')); image26 = image26.set('system:time\_start', ee.Date('2010'));

image27 = image27.set('system:time\_start', ee.Date('2011')); image28 = image28.set('system:time\_start', ee.Date('2012'));

image29 = image29.set('system:time\_start', ee.Date('2013')); image30 = image30.set('system:time\_start', ee.Date('2014'));

image31 = image31.set('system:time\_start', ee.Date('2015')); image32 = image32.set('system:time\_start', ee.Date('2016'));

image33 = image33.set('system:time\_start', ee.Date('2017')); image34 = image34.set('system:time\_start', ee.Date('2018'));

image35 = image35.set('system:time\_start', ee.Date('2019')); image36 = image36.set('system:time\_start', ee.Date('2020'));

## // Removing patches with less than 6 pixels (0.54 ha)

var myCollection = ee.ImageCollection([

 image1,image2,image3,image4,image5,image6,image7,image8,image9,image10,image11,image12,image13,

 image14,image15,image16,image17,image18,image19,image20,image21,image22,image23,image24,image25,

 image26,image27,image28,image29,image30,image31,image32,image33,image34,image35,image36]).map(

 function(image){

 return image.updateMask(image.select('b2').gte(6));

 }

 );

## // Adding a band with forest cover = 1 and a band of area in hectare, multiply number of pixels by pixel size in ha (0.09)

myCollection = myCollection.map(function(image){

 return image.addBands(image.select('b2').where(image.select('b2'), 1))

 .addBands(image.select('b2').multiply(0.09))

 .clip(ROI)

 .rename(['id', 'nPixels', 'fCover', 'pSize\_ha']);

});

## // Use this if you need to fix geometries of the "features" object

/\*

features = features.map(function (feature) {

 var filteredGeoms = feature.geometry().geometries().map(function (geometry) {

 geometry = ee.Geometry(geometry);

 return ee.Algorithms.If(geometry.type().compareTo('Polygon'), null, geometry);

 }, true);

 return feature.setGeometry(ee.Geometry.MultiPolygon(filteredGeoms));

});

\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Class-level Landscape Metrics \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### // Generating bands for calculaling class-level landscape metrics through GEE reducers

var patchMetrics = myCollection.map(function(image){

###  // Area of very small (< 10 ha) fragments

 var vs\_ha = image.select('pSize\_ha').updateMask(image.select('pSize\_ha').lt(10));

 vs\_ha = vs\_ha.where(vs\_ha, 1);

###  // Area of small (10 - 100 ha) fragments

 var s\_ha = image.select('pSize\_ha').updateMask(image.select('pSize\_ha').gte(10).and(image.select('pSize\_ha').lt(100)));

 s\_ha = s\_ha.where(s\_ha, 1);

###  // Area of medium (100 - 1,000 ha) fragments

 var m\_ha = image.select('pSize\_ha').updateMask(image.select('pSize\_ha').gte(100).and(image.select('pSize\_ha').lt(1000)));

 m\_ha = m\_ha.where(m\_ha, 1);

###  // Area of large (> 1,000 ha) fragments

 var l\_ha = image.select('pSize\_ha').updateMask(image.select('pSize\_ha').gte(1000));

 l\_ha = l\_ha.where(l\_ha, 1);

###  // Number of very small (< 10 ha) fragments

 var vs\_id = image.select('id').updateMask(image.select('pSize\_ha').lt(10));

###  // Number of small (10 - 100 ha) fragments

 var s\_id = image.select('id').updateMask(image.select('pSize\_ha').gte(10).and(image.select('pSize\_ha').lt(100)));

###  // Number of medium (100 - 1,000 ha) fragments

 var m\_id = image.select('id').updateMask(image.select('pSize\_ha').gte(100).and(image.select('pSize\_ha').lt(1000)));

###  // Number of large (> 1,000 ha) fragments

 var l\_id = image.select('id').updateMask(image.select('pSize\_ha').gte(1000));

###  // Area of cores and edges

 var mask = image.select('fCover').unmask().clip(ROI);

 var forest = mask.eq(1);

 var nonForest = mask.neq(1);

 var dist = forest.cumulativeCost({

 source: nonForest.updateMask(nonForest),

 maxDistance: 50000

 }).updateMask(image

 .select('fCover')

 .eq(1))

 .rename('distToEdge')

 .reproject({

 crs: image.select('id').projection(), scale: 30

 }).reduceResolution({

 reducer: ee.Reducer.max()

 });

 var core = dist.updateMask(dist.gt(50)).where(dist.updateMask(dist.gt(50)), 1);

 var edge = dist.updateMask(dist.lte(50)).where(dist.updateMask(dist.lte(50)), 1);

 core = core.updateMask(image.select('fCover'));

 edge = edge.updateMask(image.select('fCover'));

 // Adding the new bands to the images

 return image.addBands(vs\_ha).addBands(s\_ha).addBands(m\_ha)

 .addBands(l\_ha).addBands(vs\_id).addBands(s\_id)

 .addBands(m\_id).addBands(l\_id).addBands(core)

 .addBands(edge)

 .rename(['id','nPixels','fCover','pSize\_ha',

 'vs\_ha','s\_ha','m\_ha','l\_ha','vs\_id',

 's\_id','m\_id','l\_id','core','edge']);

});

## // Mapping the reducers through years

var loopsteps = ee.List.sequence(1985, 2020, 1);

var patchMetricsOutput = ee.FeatureCollection(loopsteps.map(function (x) {

###  // Convert to ee.Feature

 return ee.Feature(null,{

###  // Adding the year as a property

 '0\_year': ee.Date.fromYMD(ee.Number(x), 1, 1),

###  // Adding the area of very small forests as a property

 '11\_vs\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('vs\_ha').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the area of small forests as a property

 '12\_s\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('s\_ha').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the area of medium forests as a property

 '13\_m\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('m\_ha').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the area of large forests as a property

 '14\_l\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('l\_ha').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the total area of forests as a property

 '15\_total\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('fCover').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the number o very small fragments as a property

 '21\_vs\_id': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('vs\_id').reduceRegion({

 reducer: ee.Reducer.countDistinctNonNull(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the number of small fragments as a property

 '22\_s\_id': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('s\_id').reduceRegion({

 reducer: ee.Reducer.countDistinctNonNull(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the number of medium fragments as a property

 '23\_m\_id': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('m\_id').reduceRegion({

 reducer: ee.Reducer.countDistinctNonNull(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the number of large fragments as a property

 '24\_l\_id': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('l\_id').reduceRegion({

 reducer: ee.Reducer.countDistinctNonNull(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the total number of fragments as a property

 '25\_total\_id': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('id').reduceRegion({

 reducer: ee.Reducer.countDistinctNonNull(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the area of the largest fragment as a property

 '3\_largFrag\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('nPixels').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.max(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the mean fragment area as a property

 '4\_meanFrag\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('id').toInt().addBands(

 patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('nPixels').multiply(0.09)).stratifiedSample({

 numPoints: 1,

 classBand: 'id',

 region: ROI,

 scale:30,

 dropNulls: true

 }).reduceColumns({reducer: ee.Reducer.mean(), selectors:['nPixels']

 }),

###  // Adding the total core area as a property

 '5\_core\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('core').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the total edge area as a property

 '6\_edge\_ha': patchMetrics.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha','vs\_ha','s\_ha','m\_ha','l\_ha',

 'vs\_id','s\_id','m\_id','l\_id','core','edge'])

 .select('edge').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 })

 });

}));

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Deforestation and Forest Regeneration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## // Generating bands for calculaling deforestation and forest regeneration through GEE reducers

var loopsteps = ee.List.sequence(1987, 2017, 1);

var defForRest = ee.ImageCollection.fromImages(loopsteps.map(function(x){

###  // Setting dates for moving window

 var firstYear = ee.Date.fromYMD(ee.Number(x).subtract(2), 1, 1);

 var secondYear = ee.Date.fromYMD(ee.Number(x).subtract(1), 1, 1);

 var thirdYear = ee.Date.fromYMD(ee.Number(x), 1, 1);

 var fourthYear = ee.Date.fromYMD(ee.Number(x).add(1), 1, 1);

 var fifthYear = ee.Date.fromYMD(ee.Number(x).add(2), 1, 1);

 var sixthYear = ee.Date.fromYMD(ee.Number(x).add(3), 1, 1);

###  // Selecting images using dates

 var firstImage = myCollection.filter(ee.Filter.eq("system:time\_start", firstYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var secondImage = myCollection.filter(ee.Filter.eq("system:time\_start", secondYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var thirdImage = myCollection.filter(ee.Filter.eq("system:time\_start", thirdYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var fourthImage = myCollection.filter(ee.Filter.eq("system:time\_start", fourthYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var fifthImage = myCollection.filter(ee.Filter.eq("system:time\_start", fifthYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var sixthImage = myCollection.filter(ee.Filter.eq("system:time\_start", sixthYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

###  // Remapping

 firstImage = firstImage.where(firstImage, 100000).unmask(200000).clip(ROI);

 secondImage = secondImage.where(secondImage, 10000).unmask(20000).clip(ROI);

 thirdImage = thirdImage.where(thirdImage, 1000).unmask(2000).clip(ROI);

 fourthImage = fourthImage.where(fourthImage, 100).unmask(200).clip(ROI);

 fifthImage = fifthImage.where(fifthImage, 10).unmask(20).clip(ROI);

 sixthImage = sixthImage.where(sixthImage, 1).unmask(2).clip(ROI);

###  // Summing and reclassifying

 var aff = firstImage.add(secondImage).add(thirdImage).add(fourthImage).add(fifthImage).add(sixthImage);

 aff = aff.updateMask(aff.eq(221111));

 aff = aff.where(aff, 1); // in Ha

 var def = firstImage.add(secondImage).add(thirdImage).add(fourthImage);

 def = def.updateMask(def.eq(112200));

 var defYear = def.where(def, ee.Number(x)); // in Ha

 def = def.where(def, 1); // in Ha

###  // Deforestation and fragment size

 var first = myCollection.filter(ee.Filter.eq("system:time\_start", firstYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('pSize\_ha');

 var second = myCollection.filter(ee.Filter.eq("system:time\_start", secondYear))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('pSize\_ha');

 var def\_fragSize = ee.ImageCollection([first, second]).mean().updateMask(def.where(def, 1));

 var vs\_def = def\_fragSize.updateMask(def\_fragSize.lt(10));

 var s\_def = def\_fragSize.updateMask(def\_fragSize.gte(10).and(def\_fragSize.lt(100)));

 var m\_def = def\_fragSize.updateMask(def\_fragSize.gte(100).and(def\_fragSize.lt(1000)));

 var l\_def = def\_fragSize.updateMask(def\_fragSize.gte(1000));

 vs\_def = vs\_def.where(vs\_def, 1);

 s\_def = s\_def.where(s\_def, 1);

 m\_def = m\_def.where(m\_def, 1);

 l\_def = l\_def.where(l\_def, 1);

###  // Merging bands

 return def.addBands(defYear).addBands(aff).addBands(vs\_def).addBands(s\_def).addBands(m\_def).addBands(l\_def)

 .set('system:time\_start', thirdYear).rename(['deforestation', 'defYear', 'fRestoration',

 'defVS', 'defS', 'defM', 'defL']);

 })

);

## // Applying the reducers

var defForRegOutput = ee.FeatureCollection(loopsteps.map(function (x) {

###  // Convert to ee.Feature

 return ee.Feature(null,{

###  // Adding the year as a property

 '0\_year': ee.Date.fromYMD(ee.Number(x), 1, 1),

###  // Adding the deforestation of very small forests as a property

 '11\_defVS': defForRest.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRegeneration','defVS','defS','defM','defL'])

 .select('defVS').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the deforestation of small forests as a property

 '12\_defS': defForRest.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRegeneration','defVS','defS','defM','defL'])

 .select('defS').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the deforestation of medium forests as a property

 '13\_defVS': defForRest.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRegeneration','defVS','defS','defM','defL'])

 .select('defM').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the deforestation of large forests as a property

 '14\_defL': defForRest.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRegeneration','defVS','defS','defM','defL'])

 .select('defL').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the deforestation as a property

 '15\_def': defForRest.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRegeneration','defVS','defS','defM','defL'])

 .select('deforestation').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the forest regeneration as a property

 '2\_fRegeneration': defForRest.filter(ee.Filter.eq('system:time\_start', ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRegeneration','defVS','defS','defM','defL'])

 .select('fRegeneration').multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 })

 });

}));

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Current Older and Younger Forests \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## // Selecting the first deforestation event for each pixel

var firstDef = defForReg.select('defYear').min(); // first event of deforestation

## // Selecting image from the first year of your time series

var olderForests = myCollection.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(ee.Number(1985), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

## // Selecting older forests (pixels that were forests in the first year of your time series with no deforestation)

var notDef = firstDef.where(firstDef, 1).unmask().clip(ROI);

notDef = notDef.updateMask(notDef.eq(0));

olderForests = olderForests.updateMask(notDef.eq(0));

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Deforestation of Older and Younger Forests \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

var loopsteps = ee.List.sequence(1987, 2017, 1);

## // Applying reducers for calculaling deforestation of older and younger forests

var oldYoung = ee.FeatureCollection(loopsteps.map(function (x) {

###  // Selecting older forests

 var oldf = myCollection.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(ee.Number(1985), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var mask2 = firstDef.updateMask(firstDef.lt(ee.Number(x))).unmask().clip(ROI);

 mask2 = mask2.updateMask(mask2.eq(0)).add(1);

 oldf = oldf.updateMask(mask2);

###  // Selecting younger forests

 var youngf = myCollection.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(ee.Number(x).subtract(1), 1, 1)))

 .toBands().rename(['id','nPixels','fCover','pSize\_ha']).select('fCover');

 var mask3 = oldf.unmask().clip(ROI).updateMask(oldf.unmask().clip(ROI).eq(0)).add(1);

 youngf = youngf.updateMask(mask3);

###  // Calculating deforestation of older forests

 var defold = defForReg.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRestoration','defVS','defS','defM','defL']).select('deforestation');

 defold = defold.updateMask(oldf);

###  // Calculating deforestation of younger forests

 var defyoung = defForReg.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(ee.Number(x), 1, 1)))

 .toBands().rename(['deforestation','defYear','fRestoration','defVS','defS','defM','defL']).select('deforestation');

 defyoung = defyoung.updateMask(defold.unmask().clip(ROI).updateMask(defold.unmask().clip(ROI).eq(0)).add(1));

###  // Returning ee.Feature

 return ee.Feature(null,{

 '0\_year': ee.Date.fromYMD(ee.Number(x), 1, 1),

 '1\_def\_old': defold.multiply(0.09)

 .reduceRegion({reducer: ee.Reducer.sum(), geometry: ROI,

 maxPixels: 1e13, scale: 30}),

 '2\_def\_young': defyoung.multiply(0.09)

 .reduceRegion({reducer: ee.Reducer.sum(), geometry: ROI,

 maxPixels: 1e13, scale: 30})

 });

}));

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Zonal statistics \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## // Area

var zonal\_ha = patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2020, 1, 1)))

 .toBands().select(['35\_vs\_ha','35\_s\_ha','35\_m\_ha','35\_l\_ha','35\_fCover'])

 .addBands(olderForests).addBands(youngerForests)

 .addBands(myCollection3.select('deforestation').sum())

 .addBands(myCollection3.select('fRestoration').sum())

 .multiply(0.09)

 .rename(['11\_vs\_ha\_2020','12\_s\_ha\_2020','13\_m\_ha\_2020','14\_l\_ha\_2020','15\_total\_ha\_2020',

 '21\_old\_ha\_2017','22\_young\_ha\_2017','31\_def\_accum\_2017','32\_fRest\_accum\_2017']);

var zonalStats\_ha = zonal\_ha

 .reduceRegions({

 collection: features,

 reducer: ee.Reducer.sum(),

 scale: 30

});

## // Number of fragments

var zonal\_id = patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2020, 1, 1)))

 .toBands().select(['35\_vs\_id','35\_s\_id','35\_m\_id','35\_l\_id','35\_id'])

 .rename(['vs\_id\_2020','s\_id\_2020','m\_id\_2020','l\_id\_2020','total\_id\_2020']);

var zonalStats\_id = zonal\_id

 .reduceRegions({

 collection: features,

 reducer: ee.Reducer.countDistinctNonNull(),

 scale: 30

});

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Identification of the Largest Fragments \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## // Selecting fragments larger than 5000 ha

var list = patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_id')

 .updateMask(patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_pSize\_ha').gte(5000)).reduceRegion({

 reducer: ee.Reducer.frequencyHistogram(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 });

## // Manually check for fragments' ID [using print(list), it should be the left column values] and add it to the line below

list = ee.List([17378, 38316, 40142, 79332]);

## // Map to calculate metrics for the largest fragments

var largerFrags = ee.FeatureCollection(list.map(function (x) {

 return ee.Feature(null,{

###  // Adding the ID as a property

 '0\_id': ee.Number(x),

###  // Adding the area as a property

 '1\_size\_ha': patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_fCover')

 .updateMask(patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_id').eq(ee.Number(x))).multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the older forests area as a property

 '2\_old\_ha': olderForests.updateMask(patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_id').eq(ee.Number(x))).multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the younger forests area as a property

 '3\_young\_ha': youngerForests.updateMask(patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_id').eq(ee.Number(x))).multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the core area as a property

 '4\_core\_ha': patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_core')

 .updateMask(patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_id').eq(ee.Number(x))).multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 }),

###  // Adding the edge area as a property

 '5\_edge\_ha': patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_edge')

 .updateMask(patchMetrics.filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(2017, 1, 1)))

 .toBands().select('32\_id').eq(ee.Number(x))).multiply(0.09).reduceRegion({

 reducer: ee.Reducer.sum(),

 geometry: ROI,

 maxPixels: 1e13,

 scale: 30

 })

 });

}));

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Habitat quality \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## // Selecting images

var core2017Reclass = img2017.select('32\_core').unmask().clip(ROI);

var edge2017Reclass = img2017.select('32\_edge').unmask().clip(ROI);

var old2017Reclass = olderForests.unmask().clip(ROI);

var young2017Reclass = youngerForests.unmask().clip(ROI);

## // New values according to patch size (1 - 4)

var patchSizeReclass = img2017.select('32\_vs\_ha').where(img2017.select('32\_vs\_ha'), 1).unmask(0).clip(ROI).add(

 img2017.select('32\_s\_ha').where(img2017.select('32\_s\_ha'), 2).unmask(0).clip(ROI)).add(

 img2017.select('32\_m\_ha').where(img2017.select('32\_m\_ha'), 3).unmask(0).clip(ROI)).add(

 img2017.select('32\_l\_ha').where(img2017.select('32\_l\_ha'), 4).unmask(0).clip(ROI));

patchSizeReclass = patchSizeReclass.updateMask(patchSizeReclass.gt(0));

## // New values according to core and edge (10 and 20)

var edgeReclass = edge2017Reclass.where(edge2017Reclass, 10).unmask(0).clip(ROI);

var coreReclass = core2017Reclass.where(core2017Reclass, 20).unmask(0).clip(ROI);

var coreEdgeReclass = coreReclass.add(edgeReclass).updateMask(coreReclass.add(edgeReclass).gt(0));

## // New values according to age (100 and 200)

var olderReclass = old2017Reclass.where(old2017Reclass, 200).unmask(0).clip(ROI);

var youngerReclass = young2017Reclass.where(young2017Reclass, 100).unmask(0).clip(ROI);

var oldYoungReclass = olderReclass.add(youngerReclass).updateMask(olderReclass.add(youngerReclass).gt(0));

## // Generating habitat quality layer

var habQuality = patchSizeReclass.add(coreEdgeReclass).add(oldYoungReclass);

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Exporting \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## // Run the batch downloads one at a time

/\*

var batch = require('users/fitoprincipe/geetools:batch');

batch.Download.ImageCollection.toDrive(patchMetrics.map(function(image){

 return ee.Image([]).addBands(image.select('id')).toInt().clip(ROI).copyProperties({source: image, exclude: ['count']})}),

 'yourgoogledrivefoldername', {

 name: 'id\_{id}',

 scale: 30,

 region: ROI,

 maxPixels: 1e13,

 skipEmptyTiles: true,

 crs: 'EPSG:4326',

 fileFormat: "GeoTIFF"

});

\*/

/\*

var batch = require('users/fitoprincipe/geetools:batch');

batch.Download.ImageCollection.toDrive(patchMetrics.map(function(image){

 return ee.Image([]).addBands(image.select('pSize\_ha')).toFloat().clip(ROI).copyProperties({source: image, exclude: ['count']})}),

 'yourgoogledrivefoldername', {

 name: 'pSize\_{id}',

 scale: 30,

 region: ROI,

 maxPixels: 1e13,

 skipEmptyTiles: true,

 crs: 'EPSG:4326',

 fileFormat: "GeoTIFF"

});

\*/

/\*

var batch = require('users/fitoprincipe/geetools:batch');

batch.Download.ImageCollection.toDrive(patchMetrics.map(function(image){

 var edge = image.select('edge').where(image.select('edge'), 1).unmask(0).clip(ROI);

 var core = image.select('core').where(image.select('core'), 2).unmask(0).clip(ROI);

 var coreEdge = edge.add(core).updateMask(edge.add(core).gt(0)).rename('coreEdge');

 return ee.Image([]).addBands(coreEdge.select('coreEdge')).toInt().clip(ROI).copyProperties({source: image, exclude: ['count']})}),

 'yourgoogledrivefoldername', {

 name: 'coreEdge\_{id}',

 scale: 30,

 region: ROI,

 maxPixels: 1e13,

 skipEmptyTiles: true,

 crs: 'EPSG:4326',

 fileFormat: "GeoTIFF"

});

\*/

/\*

var batch = require('users/fitoprincipe/geetools:batch');

batch.Download.ImageCollection.toDrive(defForReg.map(function(image){

 return ee.Image([]).addBands(image.select('deforestation')).toInt().clip(ROI).copyProperties({source: image, exclude: ['count']})}),

 'yourgoogledrivefoldername', {

 name: 'def\_{id}',

 scale: 30,

 region: ROI,

 maxPixels: 1e13,

 skipEmptyTiles: true,

 crs: 'EPSG:4326',

 fileFormat: "GeoTIFF"

});

\*/

/\*

var batch = require('users/fitoprincipe/geetools:batch');

batch.Download.ImageCollection.toDrive(defForReg.map(function(image){

 return ee.Image([]).addBands(image.select('fRegeneration')).toInt().clip(ROI).copyProperties({source: image, exclude: ['count']})}),

 'yourgoogledrivefoldername', {

 name: 'fReg\_{id}',

 scale: 30,

 region: ROI,

 maxPixels: 1e13,

 skipEmptyTiles: true,

 crs: 'EPSG:4326',

 fileFormat: "GeoTIFF"

});

\*/

Export.image.toDrive({image: olderForests.where(olderForests, 2).unmask().clip(ROI)

 .add(youngerForests.unmask().clip(ROI))

 .updateMask(myCollection

 .filter(ee.Filter.eq("system:time\_start", ee.Date.fromYMD(ee.Number(2017), 1, 1)))

 .toBands()

 .rename(['id','nPixels','fCover','pSize\_ha'])

 .select('fCover')),

 description: "OldYoung\_tiff",

 region: ROI,

 scale: 30,

 maxPixels: 1e13,

 fileFormat: "GeoTIFF"});

Export.image.toDrive({image: largerFrags\_img,

 description: "largerFrags\_tiff",

 region: ROI,

 scale: 30,

 maxPixels: 1e13,

 fileFormat: "GeoTIFF"});

Export.image.toDrive({image: habQuality.float().clip(ROI),

 description: "habitatQuality\_tiff",

 region: ROI,

 scale: 30,

 maxPixels: 1e13,

 fileFormat: "GeoTIFF"});

Export.table.toDrive({

 collection: zonalStats\_ha,

 description: 'zonalStats\_ha\_shp',

 fileFormat: 'SHP'

});

Export.table.toDrive({

 collection: zonalStats\_id,

 description: 'zonalStats\_id\_shp',

 fileFormat: 'SHP'

});

Export.table.toDrive({

 collection: patchMetricsOutput,

 description: 'patchMetrics\_2020\_csv',

 fileFormat: 'CSV'

});

Export.table.toDrive({

 collection: defForRegOutput,

 description: 'def\_fRest\_2017\_csv',

 fileFormat: 'CSV'

});

Export.table.toDrive({

 collection: defOldYoung,

 description: 'def\_oldYoung\_2017\_csv',

 fileFormat: 'CSV'

});

Export.table.toDrive({

 collection: largerFrags,

 description: 'largerFrags\_id\_csv',

 fileFormat: 'CSV'

});