// Import Landsat imagery.

var landsat7 = ee.ImageCollection('LANDSAT/LE07/C02/T1\_L2');

var landsat8 = ee.ImageCollection('LANDSAT/LC08/C02/T1\_L2');

// Functions to rename Landsat 7 and 8 images.

function renameL7(img) {

return img.rename(['BLUE', 'GREEN', 'RED', 'NIR', 'SWIR1',

'SWIR2', 'TEMP1', 'ATMOS\_OPACITY', 'QA\_CLOUD',

'ATRAN', 'CDIST',

'DRAD', 'EMIS', 'EMSD', 'QA', 'TRAD', 'URAD',

'QA\_PIXEL',

'QA\_RADSAT'

]);

}

function renameL8(img) {

return img.rename(['AEROS', 'BLUE', 'GREEN', 'RED', 'NIR',

'SWIR1',

'SWIR2', 'TEMP1', 'QA\_AEROSOL', 'ATRAN', 'CDIST',

'DRAD', 'EMIS',

'EMSD', 'QA', 'TRAD', 'URAD', 'QA\_PIXEL', 'QA\_RADSAT'

]);

}

// Functions to mask out clouds, shadows, and other unwanted features.

function addMask(img) {

// Bit 0: Fill

// Bit 1: Dilated Cloud

// Bit 2: Cirrus (high confidence) (L8) or unused (L7)

// Bit 3: Cloud

// Bit 4: Cloud Shadow

// Bit 5: Snow

// Bit 6: Clear

// 0: Cloud or Dilated Cloud bits are set

// 1: Cloud and Dilated Cloud bits are not set

// Bit 7: Water

var clear = img.select('QA\_PIXEL').bitwiseAnd(64).neq(0);

clear = clear.updateMask(clear).rename(['pxqa\_clear']);

var water = img.select('QA\_PIXEL').bitwiseAnd(128).neq(0);

water = water.updateMask(water).rename(['pxqa\_water']);

var cloud\_shadow = img.select('QA\_PIXEL').bitwiseAnd(16).neq(0);

cloud\_shadow = cloud\_shadow.updateMask(cloud\_shadow).rename([

'pxqa\_cloudshadow'

]);

var snow = img.select('QA\_PIXEL').bitwiseAnd(32).neq(0);

snow = snow.updateMask(snow).rename(['pxqa\_snow']);

var masks = ee.Image.cat([

clear, water, cloud\_shadow, snow

]);

return img.addBands(masks);

}

function maskQAClear(img) {

return img.updateMask(img.select('pxqa\_clear'));

}

// Function to add GCVI as a band.

function addVIs(img){

var gcvi = img.expression('(nir / green) - 1', {

nir: img.select('NIR'),

green: img.select('GREEN')

}).select([0], ['GCVI']);

return ee.Image.cat([img, gcvi]);

}

// Define study time period.

var start\_date = '2020-01-01';

var end\_date = '2020-12-31';

// Pull Landsat 7 and 8 imagery over the study area between start and end dates.

var landsat7coll = landsat7

.filterBounds(geometry)

.filterDate(start\_date, end\_date)

.map(renameL7);

var landsat8coll = landsat8

.filterDate(start\_date, end\_date)

.filterBounds(geometry)

.map(renameL8);

// Merge Landsat 7 and 8 collections.

var landsat = landsat7coll.merge(landsat8coll)

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

// Mask out non-clear pixels, add VIs and time variables.

landsat = landsat.map(addMask)

.map(maskQAClear)

.map(addVIs);

// Map.addLayer(landsat.median(),

// {bands: 'GCVI', min: 0.1, max: 0.9, palette: ['white', 'green']},

// 'NDVI Mosaic');

// Visualize GCVI time series at one location.

var point = ee.Geometry.Point([78.1041536,17.3232067]);

var landsatChart = ui.Chart.image.series(landsat.select('GCVI'),

point)

.setChartType('ScatterChart')

.setOptions({

title: 'Landsat GCVI time series',

lineWidth: 1,

pointSize: 3,

});

print(landsatChart);

// var gcp = ee.FeatureCollection('projects/operating-pod-348405/assets/mycsv');

// var cdl = ee.ImageCollection("ESA/WorldCover/v100").first();

// Map.addLayer(gcp, {color:'black'}, 'CDL 2020');

// 2. Add bands for harmonics

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

// Function that adds time band to an image.

function addTimeUnit(image, refdate) {

var date = image.date();

var dyear = date.difference(refdate, 'year');

var t = image.select(0).multiply(0).add(dyear).select([0], ['t'])

.float();

var imageplus = image.addBands(t);

return imageplus;

}

// Function that adds harmonic basis to an image.

function addHarmonics(image, omega, refdate) {

image = addTimeUnit(image, refdate);

var timeRadians = image.select('t').multiply(2 \* Math.PI \* omega);

var timeRadians2 = image.select('t').multiply(4 \* Math.PI \*

omega);

return image

.addBands(timeRadians.cos().rename('cos'))

.addBands(timeRadians.sin().rename('sin'))

.addBands(timeRadians2.cos().rename('cos2'))

.addBands(timeRadians2.sin().rename('sin2'))

.addBands(timeRadians.divide(timeRadians)

.rename('constant'));

}

// Apply addHarmonics to Landsat image collection.

var omega = 1;

var landsatPlus = landsat.map(

function(image) {

return addHarmonics(image, omega, start\_date);

});

print('Landsat collection with harmonic basis: ', landsatPlus);

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

// 3. Get harmonic coefficients

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

// Function to run linear regression on an image.

function arrayimgHarmonicRegr(harmonicColl, dependent, independents) {

independents = ee.List(independents);

dependent = ee.String(dependent);

var regression = harmonicColl

.select(independents.add(dependent))

.reduce(ee.Reducer.linearRegression(independents.length(),

1));

return regression;

}

// Function to extract and rename regression coefficients.

function imageHarmonicRegr(harmonicColl, dependent, independents) {

var hregr = arrayimgHarmonicRegr(harmonicColl, dependent,

independents);

independents = ee.List(independents);

dependent = ee.String(dependent);

var newNames = independents.map(function(b) {

return dependent.cat(ee.String('\_')).cat(ee.String(

b));

});

var imgCoeffs = hregr.select('coefficients')

.arrayProject([0])

.arrayFlatten([independents])

.select(independents, newNames);

return imgCoeffs;

}

// Function to apply imageHarmonicRegr and create a multi-band image.

function getHarmonicCoeffs(harmonicColl, bands, independents) {

var coefficients = ee.ImageCollection.fromImages(bands.map(

function(band) {

return imageHarmonicRegr(harmonicColl, band,

independents);

}));

return coefficients.toBands();

}

// Apply getHarmonicCoeffs to ImageCollection.

var bands = ['NIR', 'SWIR1', 'SWIR2', 'GCVI'];

var independents = ee.List(['constant', 'cos', 'sin', 'cos2',

'sin2']);

var harmonics = getHarmonicCoeffs(landsatPlus, bands, independents);

harmonics = harmonics.clip(geometry);

harmonics = harmonics.multiply(10000).toInt32();

// Compute fitted values.

var gcviHarmonicCoefficients = harmonics

.select([

'3\_GCVI\_constant', '3\_GCVI\_cos',

'3\_GCVI\_sin', '3\_GCVI\_cos2', '3\_GCVI\_sin2'

])

.divide(10000);

var fittedHarmonic = landsatPlus.map(function(image) {

return image.addBands(

image.select(independents)

.multiply(gcviHarmonicCoefficients)

.reduce('sum')

.rename('fitted')

);

});

// Visualize the fitted harmonics in a chart.

var harmonicsChart = ui.Chart.image.series(

fittedHarmonic.select(

['fitted', 'GCVI']), point, ee.Reducer.mean(), 30)

.setSeriesNames(['GCVI', 'Fitted'])

.setOptions({

title: 'Landsat GCVI time series and fitted harmonic regression values',

lineWidth: 1,

pointSize: 3,

});

print(harmonicsChart);

var harmonicsPlus = ee.Image.cat([harmonics]);

// Export image to asset.

var filename = 'harmonics';

Export.image.toAsset({

image: harmonicsPlus,

description: filename,

assetId: 'projects/operating-pod-348405/harmonics',

dimensions: null,

region: geometry,

scale: 30,

maxPixels: 1e12

});

// Visualize harmonic coefficients on map.

var visImage = ee.Image.cat([

harmonics.select('3\_GCVI\_cos').divide(7000).add(0.6),

harmonics.select('3\_GCVI\_sin').divide(7000).add(0.5),

harmonics.select('3\_GCVI\_constant').divide(7000).subtract(

0.6)

]);

Map.addLayer(visImage, {

min: -0.5,

max: 0.5

}, 'Harmonic coefficient false color');

// Get harmonic coefficient band names.

var bands = harmonicsPlus.bandNames();

bands = bands.remove('system:index');

var dataset = ee.Image.cat([harmonicsPlus.select(bands)]);

var worldcover = ee.ImageCollection("ESA/WorldCover/v100").first();

var classification = worldcover.select('Map').clip(table)

var cropland = classification.eq(80)

Map.addLayer(cropland, {min:0, max:1, palette: ['white', 'green']}, 'Cropland')

var maskImg = classification.eq(10)

var data = classification.updateMask(maskImg)

var palette = [

'3399FF', //(0) Water

'990000', //(1) Cropland

'999999', //(2) Urban

'FA9C44', //(3) Openforest

'80FF00', //(4) Agriculture

];

// Compute the area of cropland in square meters.

// var pixelArea = ee.Image.pixelArea();

// var croplandArea = cropland.multiply(pixelArea);

// // Reduce the area image to a scalar (sum of areas).

// var stats = croplandArea.reduceRegion({

// reducer: ee.Reducer.sum(),

// geometry: table,

// scale: 10,

// maxPixels: 1e13

// });

// Print the total cropland area.

// print(stats, 'esa area');

var getpixel = cropland.multiply(ee.Image.pixelArea()).divide(10000).reduceRegion({

reducer:ee.Reducer.sum(),

geometry:table,

scale: 30,

maxPixels:1e13,

tileScale: 16

});

print(getpixel, 'esa in ha')

//Make training data by 'overlaying' the points on the image.

var points = dataset.select(bands).sampleRegions({

collection: newfc,

properties: ['LULC'],

scale: 30

}).randomColumn();

// /Randomly split the samples to set some aside for testing the model's accuracy

//using the "random" column. Roughly 80% for training, 20% for testing.

var split = 0.8;

var training = points.filter(ee.Filter.lt('random', split));

var testing = points.filter(ee.Filter.gte('random', split));

//Print these variables to see how much training and testing data you are using

print('Samples n =', points.aggregate\_count('.all'));

print('Training n =', training.aggregate\_count('.all'));

print('Testing n =', testing.aggregate\_count('.all'));

//\*\*\*\*\*\*Part 4: Random Forest Classification and Accuracy Assessments\*\*\*\*\*\*

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

//Run the RF model using 300 trees and 5 randomly selected predictors per split ("(300,5)").

//Train using bands and land cover property and pull the land cover property from classes

var classifier = ee.Classifier.smileRandomForest(300,5).train({

features: training,

classProperty: 'LULC',

inputProperties: bands

});

//Test the accuracy of the model

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

//Print Confusion Matrix and Overall Accuracy

var confusionMatrix = classifier.confusionMatrix();

print('Confusion matrix: ', confusionMatrix);

print('Training Overall Accuracy: ', confusionMatrix.accuracy());

var kappa = confusionMatrix.kappa();

print('Training Kappa', kappa);

var validation = testing.classify(classifier);

var testAccuracy = validation.errorMatrix('LULC', 'classification');

print('Validation Error Matrix RF: ', testAccuracy);

print('Validation Overall Accuracy RF: ', testAccuracy.accuracy());

var kappa1 = testAccuracy.kappa();

print('Validation Kappa', kappa1);

//Apply the trained classifier to the image

var classified = harmonicsPlus.select(bands).classify(classifier);

Map.addLayer(classified, {min: 0, max: 4, palette: palette}, 'Train APPROACH');

var regionClassified = classified.multiply(10000).toInt32();

var classPlus = ee.Image.cat([regionClassified]);

Export.image.toDrive({

image : classPlus ,

description: 'classifiedimage',

folder: 'earthengine',

fileNamePrefix: 'classifiedimage',

region: geometry,

scale: 30,

maxPixels: 1e9

});

var classified = dataset.select(bands).classify(classifier).clip(table).focal\_mode();

//Print the number of pixels for each class

var analysis\_image = classified.select("classification")

var class\_1 = analysis\_image.updateMask(analysis\_image.eq(0))

var class\_2 = analysis\_image.updateMask(analysis\_image.eq(1))

var class\_3 = analysis\_image.updateMask(analysis\_image.eq(2))

var class\_4 = analysis\_image.updateMask(analysis\_image.eq(3))

var class\_5 = analysis\_image.updateMask(analysis\_image.eq(4))

var all = class\_1.addBands(class\_2).addBands(class\_3).addBands(class\_4).addBands(class\_5)

var count\_pixels\_one = all.reduceRegion({

reducer: ee.Reducer.count(),

geometry: table,

scale:30,

maxPixels: 1e19,

})

print(count\_pixels\_one, "PIXEL APPROACH: pixels for each class")

//Use reduceRegion with a Sum reducer to calculate total area

var getpixel = class\_1.multiply(ee.Image.pixelArea()).divide(10000).reduceRegion({

reducer:ee.Reducer.sum(),

geometry:table,

scale: 30,

maxPixels:1e13,

tileScale: 16

}).get('classification');

print(getpixel, 'pixel in ha')