# Handbook on Agriculture Monitoring Using Google Earth Engine - part 2

*Aniruddha Ghosh (anighosh@ucdavis.edu)*

## Contents

## 1 Introduction to crop-mapping

We will explore a number of methods of identifying area planted to crops and also detect various crops. We start with performing a basic land cover classification using a supervised algorithm model to classify all pixels, and then mask non-crop pixels. In the second step, we classify the crop pixels to detect various crop types.

### 1.1 Image search and pre-processing

Load the study area boundary

```
// Import boundaries from asset
var aoi = ee.FeatureCollection('users/juliusadewopo/MzeTargetRegion_alt_dslv2');
```

Then filter imageCollection by region and mask clouds.

```
function maskL8sr(image) {
  // Bits 3 and 5 are cloud shadow and cloud, respectively.
  var cloudShadowBitMask = (1 << 3);
  var cloudsBitMask = (1 << 5);
  // Get the pixel QA band.
  var qa = image.select('pixel_qa');
  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
                 .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  return image.updateMask(mask);
}

//general collection
```

```
var l8collection = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
                  .filterBounds(aoi)
                  .map(maskL8sr)
```

## 1.2 Compute additional indices

We will compute a number of spectral indices for the classification.

```
// Function to add an NDVI band, the dependent variable.
// More details: https://landsat.usgs.gov/sites/default/files/documents/si_product_guide.pdf
var addVI = function(image) {
  var ndvi = image.normalizedDifference(['nir', 'red']).rename('NDVI').float();
  var gndvi = image.normalizedDifference(['nir', 'green']).rename('GNDVI').float();
  var grvi = image.normalizedDifference(['green', 'red']).rename('GRVI').float();
  var gbvi = image.normalizedDifference(['blue', 'green']).rename('GBVI').float();
    var ndmi = image.normalizedDifference(['nir', 'swir1']).rename('NDMI').float();
var nbr2 = image.normalizedDifference(['swir1', 'swir2']).rename('NBR2').float();
  // Compute the EVI using an expression.
  var evi = image.expression(
    '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 1))', {
      'NIR': image.select('nir'),
      'RED': image.select('red'),
      'BLUE': image.select('blue')
  }).rename('EVI').float();
   // Compute VI using expressions

  var gcvi = image.expression(
    '(NIR/GREEN) - 1', {
      'NIR': image.select('nir'),
      'GREEN': image.select('green'),
  }).rename('GCVI').float();

  // Return the masked image, with additional bands for predictors.
  return image
      // Add the response variables last.
      .addBands([ndvi,gndvi, grvi,gbvi,ndmi, nbr2, evi, gcvi])
      // Retain this metadata property for use in the chart.
      .copyProperties(image, ['system:time_start']);
};
```

Compute indices for the entire image collection.

```
// Which bands to select:L8
var bandNumbers = [1,2,3,4,5,6];
var bandNames = ee.List(['blue','green','red','nir','swir1','swir2']);

// apply over the image collection
var l8collection = l8collection.select(bandNumbers, bandNames).map(addVI);
```

## 1.3 Compute seasonal image composite

Next we define a combined reducer that is based on three percentile measures and works great for removing effect of any residual atmospheric noises to create a seamless mosaic.

```
// Use the combined reducer - here just 25-75th
var reducersCol = ee.Reducer.percentile({percentiles:[25]})
                      .combine(ee.Reducer.percentile({percentiles:[50]}), null, true)
                      .combine(ee.Reducer.percentile({percentiles:[75]}), null, true);
```

Next we create two composites over the growing season.

**Note** The following will be simplified later. We will develop the complexity at steps, starting from spectral bands, spectral indices to seasonal composites.

```
var getNames = function(base, list) {
  return ee.List(list).map(function(i) {
    return ee.String(base).cat(i);
  });
};

//SEASON 1:
var startDate = '2016-12-01';
var endDate = '2017-3-31';
var s1 = l8collection
              .filterDate(startDate, endDate)
              .reduce({
        reducer: reducersCol
});

var s1New = s1.select(s1.bandNames(),
  getNames('season1_', s1.bandNames()));

//season 2:
  var startDate = '2017-04-01';
  var endDate = '2017-7-31';
// Use the combined reducer - here just 25-75th
var s2 = l8collection
              .filterDate(startDate, endDate)
              .reduce({
        reducer: reducersCol
});

var s2New = s2.select(s2.bandNames(),
  getNames('season2_', s2.bandNames()));

//combine
var TwoSeason = s1New.addBands(s2New);
print(TwoSeason);
```

### 1.3.1 Visualization of seasonal composite

We can display images to take a look at what we're working with.

```
var vizParams = {bands: ['nir', 'red', 'green'], min: 0, max: 3000};
// to be added
```

## 1.4 Create training samples

We can import vector files from external sources to Earth Engine. We can also create Geometries, Features, or FeatureCollections using the code editor features. We can basically draw geometries and import them in the script, so they are there to work with and they show up on the map. Below is a picture of year xxx false color composite map with regions of interest displayed on top. Here I clicked 100 points for each land cover class (400 points in total). I tried to distribute them around the entire study, since a given land cover class might differ more in different geographic areas and to provide the classifier varied training data to make it more robust.

**Important** Read the geometry tools guide to learn how to generate training polygons.

**Insert image**

`FeatureCollections` for each class can be merged into a single `FeatureCollection`. I exported this `FeatureCollection` to my Earth Engine Asset for future use and sharing with other collaborators. That way we could run the classification script with the training sites at a later time, rather than doing everything at once in one very long script.

```
var samples = crop.merge(water).merge(soil).merge(vegetation);
//add export
```

We can repeat this process for different years as the land use patterns may change between years. Creating training dataset is one of the most important steps in classification and often overlooked. Throughout the guide I will emphasize the best practices of creating training samples and how to integrate that in the project design.

## 1.5 Training Classifier

Earth Engine has support for a number of different classification algorithms, including random forests, naive Bayes, and support vector machines. We chose to use the random forest algorithm in this example as various researchers found that it often performs better than other classifiers under the same condition. Random Forest, or popularly known as the RF classifier, doesn't make assumptions about the distribution of data.

**Optional reading** If you're interested in the details of how a random forest classification works, Liaw and Wiener (2002) gives a nice overview.

```
// name of the bands
var bands = TwoSeason.bandNames();

// extract band values from the input image
var samplevals = TwoSeason.sampleRegions({
  collection: samples,
  properties: ['landcover']
  scale: 30
});

// define the random forest classier
var classifier = ee.Classifier.randomForest({
        numberOfTrees: 500,
        variablesPerSplit: 8,
        minLeafPopulation: 2,
        seed: 1});

// Train the classifier
var trained = classifier.train(samplevals, "landcover", bands);

// Classify the image with the same bands used for training.
var classified = TwoSeason.select(bands).classify(trained);

// Display the inputs and the results.
Map.centerObject(samples, 11);
Map.addLayer(TwoSeason, {bands: ['B4', 'B3', 'B2'], max: 0.4}, 'image');
Map.addLayer(classified,
             {min: 0, max: 3, palette: ['red', 'green', 'blue']},
             'classification');

// Export the classification result to Google drive for additional analysis

Export.image.toDrive({
    image: classified,
    description: "export_landsat_classification",
    fileNamePrefix: 'landsat8_classified',
    folder: 'GEE_export',
    scale: 30,
    region: aoi,
    maxPixels: 1e13
});
```

### 1.5.1 Accuracy assesment

We can also split the samples into training and independent validation samples, perform model training with training samples and test the model performance with validation samples.

```
// Step 1: Assign random numbers in preparation for a test/train split
var seed = 100;
samples = samples.randomColumn('random', seed);

// Join the landcover & random values with all pixels in each polygon in the training datasets.
var samplevals = TwoSeaosn.sampleRegions({
    collection: samples,
    properties: ['landcover', 'random'],
    scale: 30
});


//Step 2: Perform a  30/70 test/train split using the random numbers  generated in Step 1 to assess mod

var training = samplevals.filterMetadata('random', 'less_than', 0.7);
var testing = samplevals.filterMetadata('random', 'not_less_than', 0.7);

// Step 3: Train the classifier
// define the random forest classier
var classifier = ee.Classifier.randomForest({
        numberOfTrees: 500,
        variablesPerSplit: 8,
        minLeafPopulation: 2,
        seed: 1});

// Train the classifier.
var trained = classifier.train(training, "cropland", bands);

// Confusion matrix with independent samples
var validation = testing.classify(trained);

// Generate accuracy metrics
var errorMatrix = validation.errorMatrix('landcover', 'classification');

// print results
```

The accuracy matrices will help us to understand how good the classifiers performed.

Explain accuracy matrices here.

## 1.6 Calculating area under cropland

We will create a new image where the value of each pixel was its area in square meters using the `ee.Image.pixelArea()` function. We can convert this to square kilometers so the numbers would be more understandable, producing `areaImageSqKm` as a new Image. Earth Engine allows you to multiply Images, in which case pixel 1 in Image A is multiplied by pixel 1 in Image B to produce the value of pixel 1 in Image C, and so on. Since the value of each non-fallowed pixel in `cropBinary` Image was 0, multiplying `cropBinary` by `areaImageSqKm` will produce a new image, `cropArea`, where each pixel's value was its area in square kilometers if it had been fallowed and zero otherwise.

```
// Calculate fallowed area by pixel (0 if pixel was not fallowed)
var areaImageSqM = ee.Image.pixelArea()
    .clip(roi);
var areaImageSqKm = areaImageSqM.multiply(0.000001);
var cropArea = cropBinary.multiply(areaImageSqKm);
```

Now we could sum all the pixels in `cropArea` using a Reducer to get the total fallowed area. To make this work, we can performed the computation at a different resolution (e.g. 1000-meter spatial resolution); smaller resolutions (like 30 meters, Landsat's spatial resolution) can be too fine for Earth Engine's computation engine to handle and made it time out.

```
// Calculate total fallowed area in square kilometers.
var totalCroppedArea = cropArea.reduceRegion({reducer: ee.Reducer.sum(),
    geometry: roi,
    scale: 1000});
print('Total cropped area, sq km:', totalCroppedArea);
```

The end result from the classification approach: xxxx square kilometers of cropland. The study are is xxxx square kilometers total. This makes the estimate of crop land to be xx%.