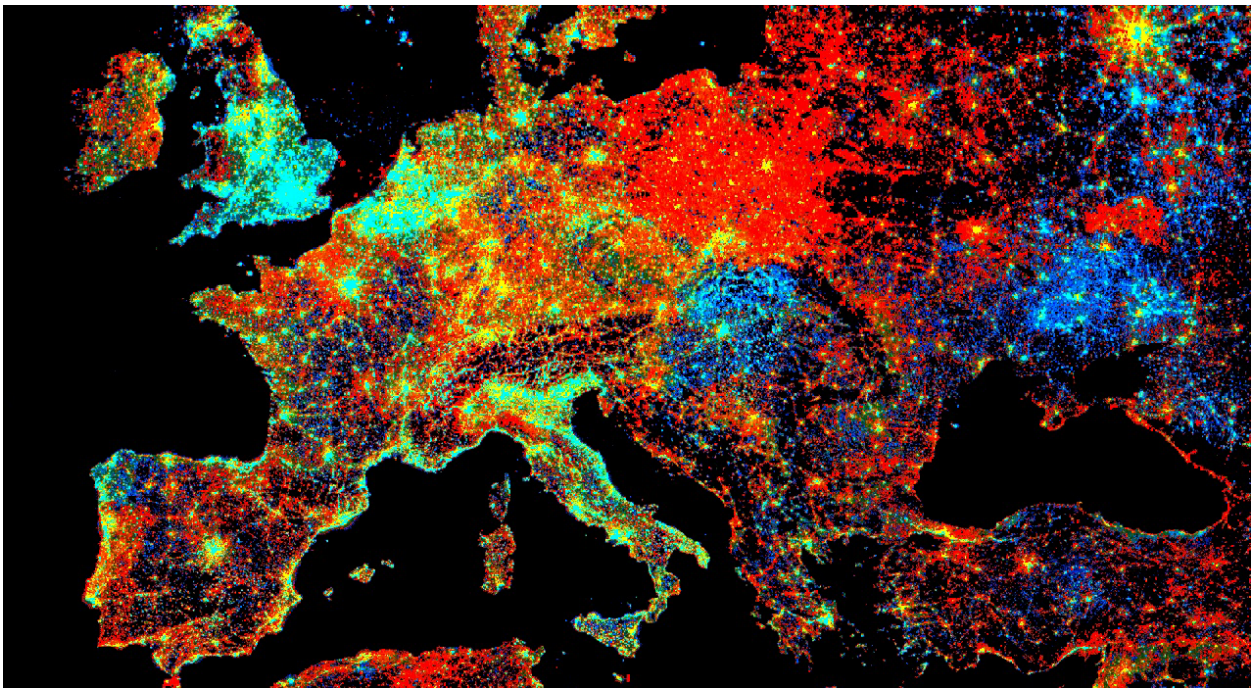
Cloud-Based Remote Sensing with Google Earth Engine



Fundamentals and Applications

or, click here to get back to the master document to access different sections)

Part A3: Terrestrial Applications

Earth's terrestrial surface is analyzed regularly by satellites, in search of both change and stability. These are of great interest to a wide cross-section of Earth Engine users, and projects across large areas illustrate both the challenges and opportunities for life on Earth. Chapters in this Part illustrate the use of Earth Engine for disturbance, understanding long-term changes of rangelands, and creating optimum study sites.

For chapters A3.6 and later,

go to this document

Chapter A3.1: Active fire monitoring

Authors
Morgan A. Crowley* and Tianjia Liu* (*shared first-authorship)

Overview

Fire monitoring across the world benefits from raw satellite imagery and processed fire-mapping datasets. Google Earth Engine supports fire monitoring throughout fire seasons with satellite data from sources like Landsat 8, Sentinel-2, and Moderate Resolution Imaging Spectroradiometer (MODIS), and by hosting multiple fire datasets from the Geostationary Operational Environmental Satellite (GOES) and the Fire Information for Resource Management System (FIRMS). In this chapter, you will access, process, and explore three fire monitoring datasets available in the data catalog. By the end of this chapter, you will learn how to use the Code Editor and user apps to summarize and compare the characteristics of fires, fire seasons, and fire-monitoring datasets.

Learning Outcomes

- Accessing and visualizing fire-monitoring datasets in the JavaScript UI.
- Adjusting previously drafted code to calculate fire characteristics in the JavaScript UI for a fire of your choice.
- Exploring fire metrics and visualization with user apps.
- Identifying pros and cons of different fire datasets for a variety of monitoring objectives.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Write a function and map it over an `ImageCollection` (Chap. F4.0).
- Filter a `FeatureCollection` to obtain a subset (Chap. F5.0, Chap. F5.1).
- Design user interfaces for an Earth Engine App (Chap. F6.3).
- Access and interact with previously made Earth Engine user apps (Chap. F6.3).

Introduction to Theory

Fire has many roles around the world. It is both a naturally occurring ecological process in fire-prone regions and a tool used by humans for land and resource management. However, fires and their emissions continue to have more extreme impacts as human settlements expand, climatic conditions become less predictable, and fire seasons lengthen (Jolly et al. 2015). To better identify and quantify the effects of fires across the globe, it is vital to monitor fires using various methods, including hand-drawn maps, ground-based sensors, GPS tracking, aerial surveys, imagery collection, and satellite-based data (Andela et al. 2019; Archibald et al. 2009; Nogueira et al. 2016; Stinson et al. 2011; Veraverbeke et al. 2014).

Different sources of satellite imagery can be used to visualize fire conditions and progressions, calculate band ratios reflecting disturbance and fire severity, and map burned areas with training data-informed classification algorithms (Crowley et al. 2019a; Crowley et al. 2019b; Hawbaker et al. 2017; Hermosilla et al. 2018; Parks 2014; Parks et al. 2019; Veraverbeke et al. 2014). Many premade fire datasets are readily available for monitoring global fire locations, extents, and progressions (Andela et al. 2019; Chuvieco et al. 2016; Giglio et al. 2016; Humber et al. 2018). In the case of existing fire datasets available for large spatial and temporal extents, remote sensing scientists apply their robust classification algorithms on satellite imagery and other geospatial data. Earth Engine makes fire monitoring more accessible by sharing multiple data sources in the data catalog so users can easily access and process these data to meet their desired objectives (Liu and Crowley 2021).

Practicum

Section 1. Fire Datasets in Google Earth Engine

In the following example, we use MODIS and GOES datasets (Table A3.1.1) to map the Bobcat Fire, a megafire that burned 115,796 acres in Los Angeles County, California, in September 2020.

Table A3.1.1 MODIS and GOES fire mapping datasets available in the Google Earth Engine data catalog

Satellite / Sensor	Dataset	Variable	Resolution	Dataset Coverage
MODIS Terra,	MOD/MYD14A1	active fires	1 km, daily	Global, 2001

MODIS Aqua*				to present
	MCD64A1	burned area	500 m, monthly**	Global, 2001 to present
GOES-16, GOES-17*	FDCF	active fires	2 km, every 15 minutes	North / South America, 2017 to present

* MODIS Terra (from 2000), MODIS Aqua (from 2002), GOES-16 (from 2016), GOES-17 (from 2017)

** Can be disaggregated into daily resolution using the MCD64A1 burn date variable

First, we need to define temporal and spatial variables to filter the datasets, namely the approximate ignition coordinates and start date of the fire.

```
// Define the location of the fire.
var lon = -117.868;
var lat = 34.241;
var zoom = 9;

// Filter datasets to a specific date range:
// start date of fire.
var inYear = 2020;
var inMonth = 9;
var inDay = 6;
```

Using the ignition date (September 6, 2020), we can define separate date ranges to filter the active fire and burned area datasets to account for differences in the temporal structure of the datasets—i.e., daily versus monthly. Here we set the temporal filter range for active fire datasets as the two-week period starting from the ignition date and for the burned area dataset as the month of September. The duration variables can be modified according to the fire of interest.

```
var durationAF = 15; // in days
var durationBA = 1; // in months
```

```
// Date range for active fires.
var startDateAF = ee.Date.fromYMD(inYear, inMonth, inDay);
var endDateAF = startDateAF.advance(durationAF, 'day');

// Date range for burned area.
var startDateBA = ee.Date.fromYMD(inYear, inMonth, 1);
var endDateBA = startDateBA.advance(durationBA, 'month');
```

With these input variables defined, we can preprocess the fire datasets. We will upload a high-resolution reference perimeter provided by the U.S. National Interagency Fire Center (NIFC) and then add the MODIS and GOES datasets from the Earth Engine data catalog.

Reference Fire Perimeter

The NIFC produces wildland fire perimeters using information from local fire agencies. For this tutorial, we uploaded the NIFC perimeter for the Bobcat Fire, converting it from a shapefile to an Earth Engine asset. We will access the asset from the book repository and use it as a reference layer to compare with the MODIS and GOES datasets.

```
// ------------------------------
// 1. Reference Perimeter (WFIGS)
// ------------------------------
// Note: each fire has multiple versions, so here we are
// filtering WFIGS by the name of the fire, sorting the
// area of the filtered polygons in descending order,
// and retrieving the polygon with the highest area.
var WFIGS = ee.FeatureCollection(
    'projects/gee-book/assets/A3-1/WFIGS');
var reference = ee.Feature(WFIGS.filter(ee.Filter.eq('irwin_In_1',
        'BOBCAT'))
    .sort('poly_Acres', false).first());
```

MODIS Active Fire Products

The gridded 1 km MODIS active fire datasets, MOD14A1 (Terra) and MYD14A1 (Aqua), have daily collection rates and global coverage (Giglio et al. 2016; Giglio 2010). The MODIS sensor is mounted on the two separate satellites, Terra and Aqua, both operated by NASA for environmental monitoring. Here we define two functions to process the fire

mask and fire radiative power (FRP) variables. The fire mask provides a categorical classification of the confidence in fire detection, where values ≥7 indicate that fire is present. FRP is a continuous variable that is a proxy for fire intensity, in units of megawatts (MW). Note that MODIS FRP must be multiplied by 0.1 to be in units of MW.

```
// ------------------------------
// 2. MODIS active fire datasets
// ------------------------------
// MOD14A1, MYD14A1 = MODIS/Terra and Aqua active fires and thermal
anomalies
// resolution: daily, gridded at 1km in sinusoidal projection
(SR-ORG:6974)
// variables: fire mask (FireMask), fire radiative power in MW (MaxFRP)
// satellite overpasses: Terra (10:30am/pm local time), Aqua (1:30am/pm
local time)

// Define the Earth Engine paths for MOD14A1 and MYD14A1, collection 6.
var mod14a1 = ee.ImageCollection('MODIS/006/MOD14A1');
var myd14a1 = ee.ImageCollection('MODIS/006/MYD14A1');

// Filter the datasets according to the date range.
var mod14a1Img = mod14a1.filterDate(startDateAF, endDateAF);
var myd14a1Img = myd14a1.filterDate(startDateAF, endDateAF);

var getFireMask = function(image) {
    // Fire Mask (FireMask): values ≥ 7 are active fire pixels
    return image.select('FireMask').gte(7);
};

var getMaxFRP = function(image) {
    // FRP (MaxFRP): MaxFRP needs to be scaled by 0.1 to be in units of
MW.
    return image.select('MaxFRP').multiply(0.1);
};

// Define the active fire mask (count of active fire pixels).
var mod14a1ImgMask = mod14a1Img.map(getFireMask).sum();
```

```
var myd14a1ImgMask = myd14a1Img.map(getFireMask).sum();

// Define the total FRP (MW).
var mod14a1ImgFrp = mod14a1Img.map(getMaxFRP).sum();
var myd14a1ImgFrp = myd14a1Img.map(getMaxFRP).sum();
```

MODIS Burned Area Product

The gridded 500 m MODIS burned area dataset, MCD64A1, is monthly with global coverage but can be disaggregated to daily resolution with its burn date variable (Giglio et al. 2016; Humber et al. 2018). Here we define a function to retrieve the burn date.

```
// ------------------------------
// 3. MODIS burned area dataset
// ------------------------------
// MCD64A1 = MODIS/Terra and Aqua combined burned area
// resolution: monthly, gridded at 500m in sinusoidal projection
(SR-ORG:6974),
// can be disaggregated to daily resolution
// variables: burn date as day of year (BurnDate)

// Define the Earth Engine paths for MCD64A1, collection 6.
var mcd64a1 = ee.ImageCollection('MODIS/006/MCD64A1');

var getBurnDate = function(image) {
    // burn day of year (BurnDate)
    return image.select('BurnDate');
};

// Define the burned area mask.
var mcd64a1Img = mcd64a1.filterDate(startDateBA, endDateBA);
var mcd64a1ImgMask = mcd64a1Img.map(getBurnDate).min();
```

GOES Active Fire Products

The gridded 2 km GOES-16 (East) and GOES-17 (West) active fire datasets cover North and South America in the full disk version (FDCF) with a temporal revisit rate of 15-minute increments (Hall et al. 2019; Schroeder et al. 2008). The two GOES satellites are operated by the National Oceanic and Atmospheric Administration (NOAA) and are

primarily used for meteorological monitoring. Note that the pixel orientation and shape differ between GOES-16 and GOES-17 because of the different viewing angles of the two satellites.

```
// ------------------------------
// 4. GOES 16/17 active fires
// ------------------------------
// GOES-16/17 - geostationary satellites over North/South America
// resolution: every 10-30 minutes, 2 km
// variables: fire mask (Mask), FRP (Power)

// Define the Earth Engine paths for GOES-16/17.
var goes16 = ee.ImageCollection('NOAA/GOES/16/FDCF');
var goes17 = ee.ImageCollection('NOAA/GOES/17/FDCF');

var filterGOES = ee.Filter.calendarRange(0, 0, 'minute');

// Filter the datasets according to the date range.
var goes16Img = goes16.filterDate(startDateAF, endDateAF)
    .filter(filterGOES);
var goes17Img = goes17.filterDate(startDateAF, endDateAF)
    .filter(filterGOES);

var getFireMask = function(image) {
    // fire mask (Mask): values from 10-35 are active fire pixels,
    // see the description for QA values to filter out low confidence
fires
    return image.select('Mask').gte(10).and(image.select('Mask')
        .lte(35));
};

var getFRP = function(image) {
    // FRP (Power), in MW
    return image.select('Power');
};

// Define the active fire mask (count of active fire pixels).
```

```
var goes16ImgMask = goes16Img.map(getFireMask).sum();
var goes17ImgMask = goes17Img.map(getFireMask).sum();

// Define the total FRP (MW).
var goes16ImgFrp = goes16Img.map(getFRP).sum();
var goes17ImgFrp = goes17Img.map(getFRP).sum();
```

Now, we will visualize the three datasets, along with the reference Bobcat Fire perimeter, and plot the layers on the Earth Engine interactive map.

```
// ------------------------------
// 5. Map Visualization - Layers
// ------------------------------
// Use the 'Layers' dropdown menu on the map panel to toggle on and off
layers.
Map.addLayer(mod14a1ImgMask.selfMask(), {
    palette: 'orange'
}, 'MOD14A1');
Map.addLayer(myd14a1ImgMask.selfMask(), {
    palette: 'red'
}, 'MYD14A1');

Map.addLayer(mcd64a1ImgMask.selfMask(), {
    palette: 'black'
}, 'MCD64A1');

Map.addLayer(goes16ImgMask.selfMask(), {
    palette: 'skyblue'
}, 'GOES16', false);
Map.addLayer(goes17ImgMask.selfMask(), {
    palette: 'purple'
}, 'GOES17', false);

Map.setCenter(lon, lat, 9);
```

We can also visualize the datasets side by side as shown in Fig. A3.1.1 by using the `ui.Panel` and `ui.Map.Linker` using the code provided by the "Linked Maps" script under **Examples** > **User Interface** in the **Scripts** panel.

```
// ------------------------------------
// 6. Map Visualization - Panel Layout
// ------------------------------------

// Define the panel layout.
var panelNames = [
    'MODIS active fires', // panel 0 - top left
    'MODIS burned area', // panel 1 - bottom left
    'GOES active fires', // panel 2 - top right
    'Reference' // panel 3 - bottom right
];

// Create a map for each visualization option.
var maps = [];
panelNames.forEach(function(name, index) {
    var map = ui.Map();
    map.setControlVisibility({
        fullscreenControl: false
    });

    if (index === 0) {
        map.addLayer(mod14a1ImgMask.selfMask(), {
            palette: 'orange'
        }, 'MOD14A1');
        map.addLayer(myd14a1ImgMask.selfMask(), {
            palette: 'red'
        }, 'MYD14A1');
        map.add(ui.Label(panelNames[0], {
            fontWeight: 'bold',
            position: 'bottom-left'
        }));
    }
    if (index == 1) {
        map.addLayer(mcd64a1ImgMask.selfMask(), {
```

```
            palette: 'black'
        }, 'MCD64A1');
        map.add(ui.Label(panelNames[1], {
            fontWeight: 'bold',
            position: 'bottom-left'
        }));
    }
    if (index == 2) {
        map.addLayer(goes16ImgMask.selfMask(), {
            palette: 'skyblue'
        }, 'GOES16');
        map.addLayer(goes17ImgMask.selfMask(), {
            palette: 'purple'
        }, 'GOES17');
        map.add(ui.Label(panelNames[2], {
            fontWeight: 'bold',
            position: 'bottom-left'
        }));
    }
    if (index == 3) {
        map.addLayer(reference, {}, 'Reference');
        map.add(ui.Label(panelNames[3], {
            fontWeight: 'bold',
            position: 'bottom-left'
        }));
    }
    maps.push(map);
});

var linker = ui.Map.Linker(maps);

// Make a label for the main title of the app.
var title = ui.Label(
    'Visualizing Fire Datasets in Google Earth Engine', {
        stretch: 'horizontal',
        textAlign: 'center',
        fontWeight: 'bold',
```

```
        fontSize: '24px'
    });

// Define a map grid of 2x2 sub panels.
var mapGrid = ui.Panel(
    [
        ui.Panel([maps[0], maps[1]], null, {
            stretch: 'both'
        }),
        ui.Panel([maps[2], maps[3]], null, {
            stretch: 'both'
        })
    ],
    ui.Panel.Layout.Flow('horizontal'), {
        stretch: 'both'
    }
);
maps[0].setCenter(lon, lat, zoom);

// Add the maps and title to the ui.root().
ui.root.widgets().reset([title, mapGrid]);
ui.root.setLayout(ui.Panel.Layout.Flow('vertical'));
```

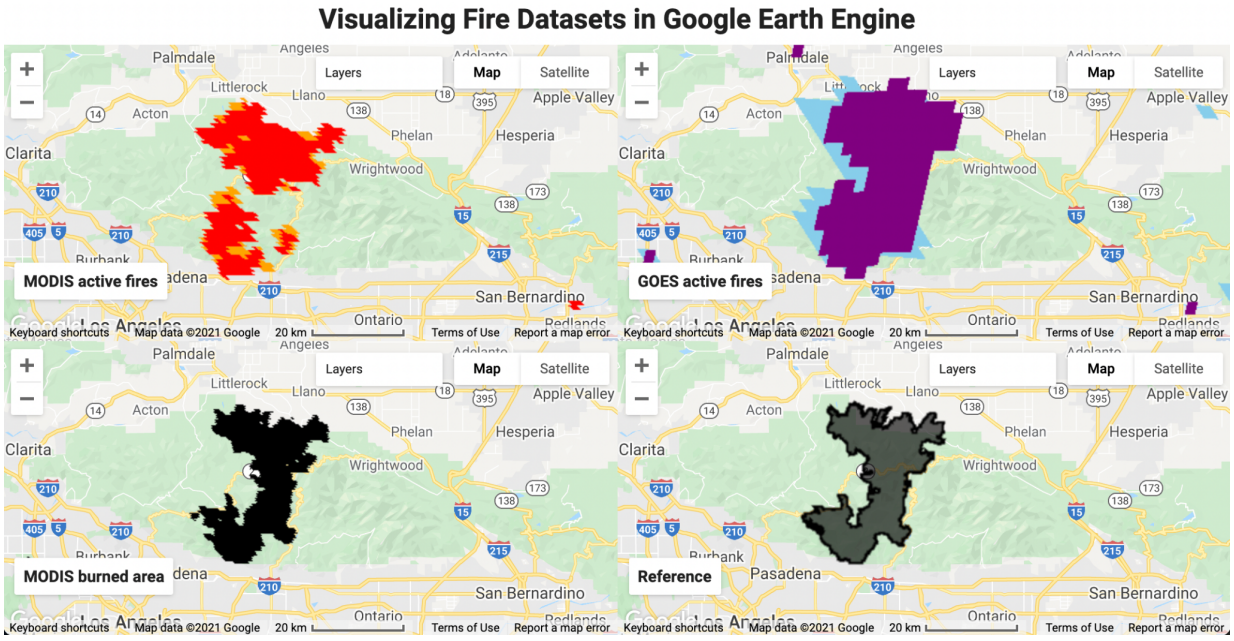


Fig. A3.1.1 Side-by-side panel comparison of the Bobcat Fire as seen in (clockwise from top left) MODIS active fires, GOES-16/17 active fires, MODIS burned area, and a reference fire perimeter from the WFIGS dataset

Code Checkpoint A31a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. How does the burned area classification (i.e., burned versus unburned) and the spatial resolution (i.e., pixel size) differ across the three datasets?

Question 2. There appears to be a gap in fire activity between the MODIS incident data and burned area map for the Bobcat Fire, as shown in the split-panel app. What differences in the datasets might account for the mismatched fire classifications? Hint: Use https://worldview.earthdata.nasa.gov/ to examine raw MODIS imagery of the fire location and date.

Question 3. How does the temporal resolution of 15 minutes for GOES impact monitoring fires in the event of smoke and haze?

Section 2. In-Depth Visualization and Analysis of Fires in Earth Engine Apps

Earth Engine Apps help to curate in-depth visualization and analysis of fires. Here we present two apps using datasets from the Earth Engine public data catalog.

In the remainder of this chapter, you will use these two apps to learn more about the Bobcat Fire and to explore findings from the two datasets.

App 1: FIRMS Active Fires

FIRMS currently monitors active fires detected by MODIS, Visible Infrared Imaging Radiometer Suite (VIIRS), and NOAA-20 in near-real time. FIRMS retains the coordinates of the centroid of pixels where one or more active fires are detected. The FIRMS dataset in the Earth Engine data catalog includes only MODIS active fires, gridded at 1 km spatial resolution. Note that the FIRMS dataset is meant for exploratory rather than rigorous scientific analyses.

The "FIRMS Active Fires" Earth Engine app allows users to visualize the spatial and temporal variation in FIRMS active fires within a defined region.

Code Checkpoint A31b. The book's repository contains information about accessing the app.

Using the control panel, you can specify the date range (start year, end year, and day-of-year range) and draw a region of interest using either a rectangle or a polygon. The map shows the number of years that one or more active fires were detected in each pixel. The chart panel shows the total daily fire counts within the region as a timeseries, where each color represents a different year (Fig. A3.1.2). You can also change the chart type to display the cumulative active fire count.

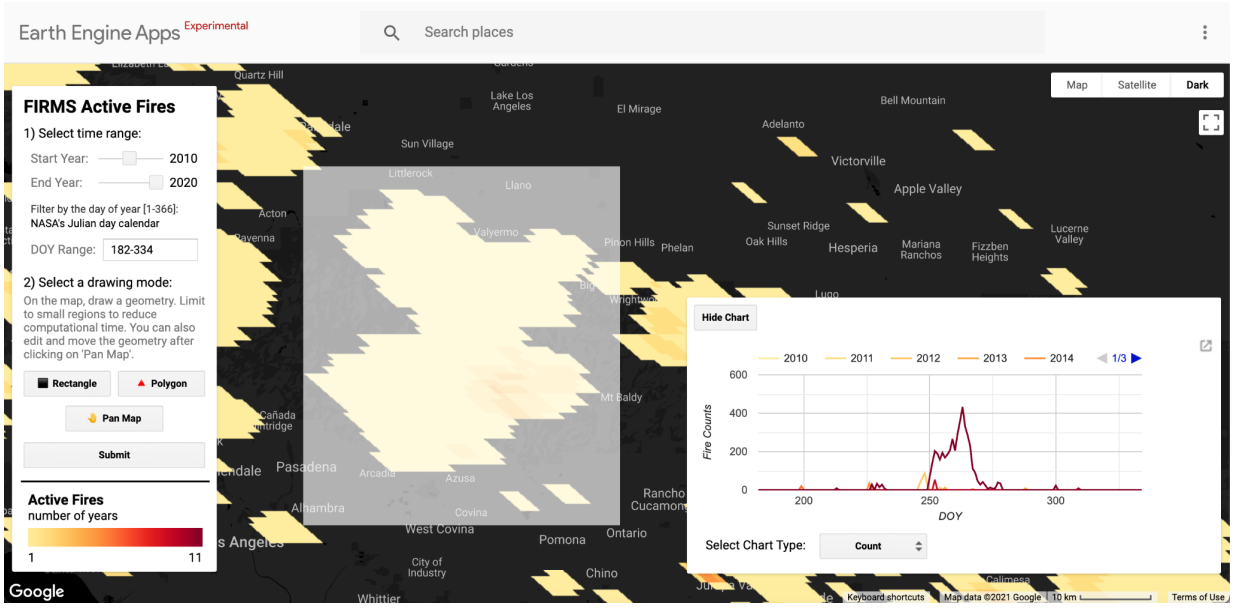


Fig. A3.1.2 Bobcat Fire in the Earth Engine app FIRMS Active Fires. The chart panel on the lower right displays the history of active fires in the Bobcat Fire area from July–November in the years 2010–2020. The map layer shows the number of years in each pixel that had at least one active fire.

Question 4. Navigate to the Bobcat Fire using the ignition coordinates (longitude, latitude). Using the satellite or map base layer, draw a polygon similar to the one shown in a Fig. A3.1.2. Submit your task and confirm your results with the above details.

Code Checkpoint A31c. The book's repository contains information about how your app should look at this point.

Question 5. Examine the chart. Approximately how many days did the fire actively burn? Hint: Hover over the chart and compare the first DOY value and the final DOY value.

App 2: U.S. Fire Dashboard

In a more advanced app, the "U.S. Fire Dashboard", GOES-16/17 gridded active fires are used to calculate a smoothed burn perimeter for the various wildfires in 2020 by modifying the code from the Google Earth Engine Medium article, "How to generate wildfire boundary maps with Earth Engine" (Restif and Hoffman 2020). The code takes

advantage of the different GOES-16 and GOES-17 pixel orientation and shape to downscale the burn classification to a finer resolution than that of the native GOES imagery. In the following example, we will use this app to visualize how the Bobcat Fire progressed in space and time from ignition (Fig. A3.1.3).

Code Checkpoint A31d. The book's repository contains information about accessing the app.

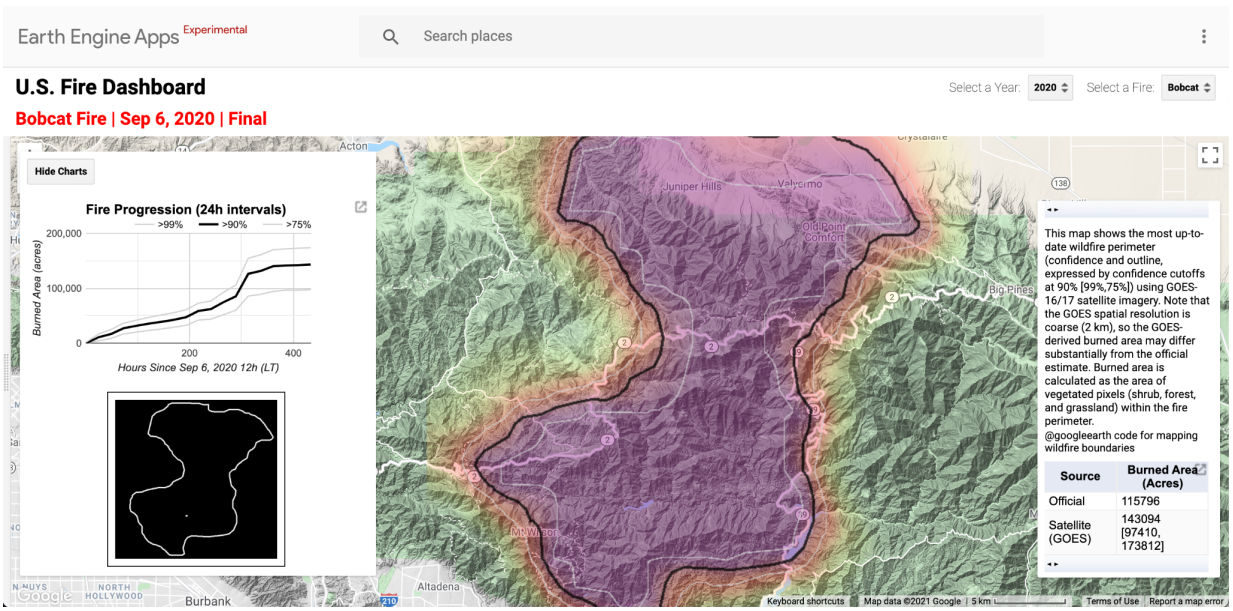


Fig. A3.1.3 Bobcat Fire in the Earth Engine app U.S. Fire Dashboard. The map displays the smoothed burn perimeter derived from GOES active fires, where the color gradient represents the confidence in the classification of the burned area. The black line shows the burn perimeter derived from pixels with >90% confidence. The charts on the left-hand panel show the evolution of the Bobcat Fire in terms of its cumulative area and spatial growth.

Question 6. Navigate to the Bobcat Fire using the dropdown panel at the top right corner of the app and wait a few minutes for the results to load. Examine the fire progression chart drawn at the top of the left panel. How many hours did it take for the fire to reach its maximum burned area? Now examine the animated fire progression GIF at the bottom of the left panel. What direction did the Bobcat Fire burn?

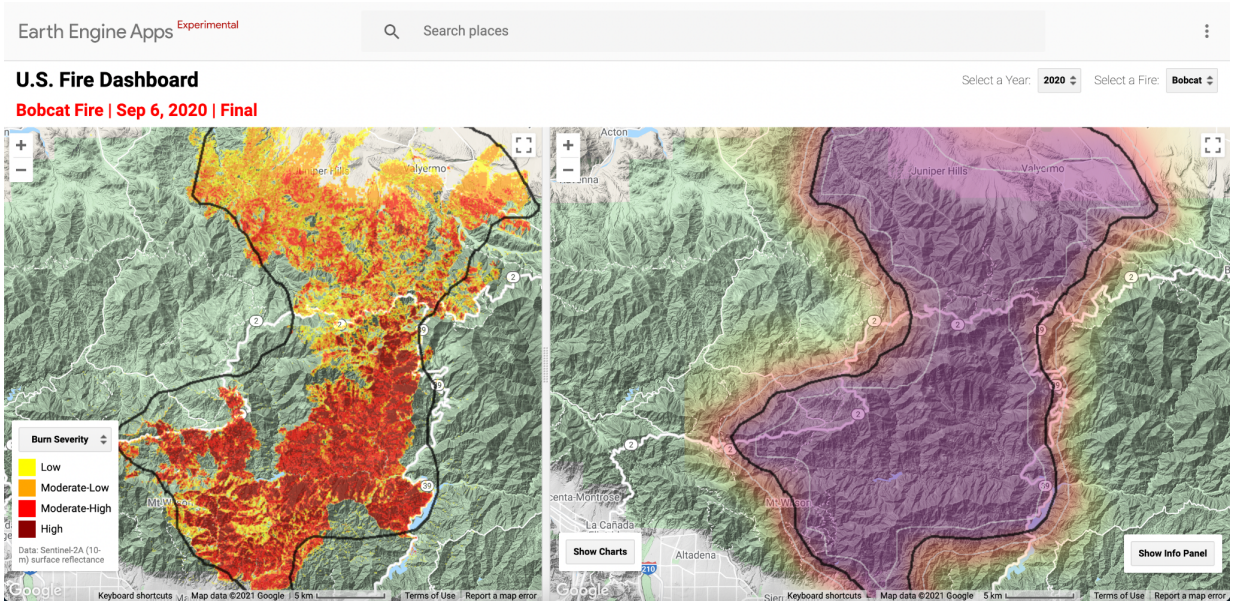


Fig. A3.1.4 Bobcat Fire in the Earth Engine app U.S. Fire Dashboard (https://globalfires.earthengine.app/view/us-fire-dashboard). The left-hand panel shows the burn severity derived from Sentinel-2A surface reflectance. The right-hand panel shows the confidence of the burned area classification derived from GOES active fires. The black line in both panels shows the burn perimeter derived from pixels having >90% confidence.

Question 7. Now drag the split panel from the left to reveal a second panel. Use the dropdown menu on the legend to navigate to the burn severity option. Let the results load and you will see a burn severity map calculated from Sentinel-2 imagery in the left-hand panel. For this question, consider all burn severities as burned area. Where do the Sentinel-2 map and the GOES map for the Bobcat Fire agree on the burned area (Fig. A3.1.4)? Where do they disagree?

Question 8. Having explored the different datasets and the two apps, what features would you include in your own wildfire mapping app? Explore additional data sources in the Synthesis section to get more inspiration for your app.

Synthesis

Assignment 1. You are now familiar with three fire datasets available in Earth Engine. Table A3.1.2 presents some additional fire datasets that are also available in the data

catalog. Use the example codes written in each dataset's description in the data catalog to load and explore the datasets in the Code Editor.

Table A3.1.2 Additional external fire mapping datasets that can be uploaded into Google Earth Engine. For each dataset, we describe their variable type, resolution, geographic coverage, and temporal extent features.

Dataset	**Variable**	**Resolution**	**Geographic Coverage**	**Temporal Extent**
FireCCI51	burned area	250 m raster, daily	global	2001–2019
GlobFire Fire Event	fire boundaries	Polygon, daily and final	global	2001–2021
MODIS FIRMS Near-Real-Time Hotspot	active fires	1 km, daily	global	2000–present

Assignment 2. While Earth Engine provides access to many existing fire datasets, there are other commonly used fire mapping datasets that are also quite useful for examining active and past fires. Select and import one of the external fire datasets shown in **Table A3.1.3** into Earth Engine as a personal asset to compare it with the active fire datasets already loaded in your script.

Table A3.1.3 Additional external fire-related datasets that can be uploaded into Google Earth Engine. For each dataset, we describe their variable type, resolution, geographic coverage, and temporal extent features.

Dataset	**Variable**	**Resolution**	**Geographic Coverage**	**Temporal Extent**
Monitoring Trends in Burn Severity	burned severity and perimeters	30 m, final	United States	1984–2019
Canadian National Fire Database	fire locations, perimeters, and burned area	30 m, final	Canada	1980–2020

Landsat Burned Area	burned area	30 m, 8-day	United States	1984–present

Assignment 3. In addition to comparing active fire maps using the datasets suggested in Tables A3.1.2 and A3.1.3, you can examine fire conditions and impacts using ancillary datasets readily available in Earth Engine. For example, you can overlay the fire datasets with vegetation conditions and fire regimes from the LANDFIRE program to better understand the ecological context of active fires.

Select one ancillary dataset to load from the data catalog and explore it alongside an active fire dataset. Land cover datasets such as MODIS, the USGS National Land Cover Database, and Copernicus Global Land Cover can help indicate types of fires and where they are occurring. By examining active fire maps with aerosol and other emission data from Sentinel-5P and MODIS Multi-Angle Implementation of Atmospheric Correction, you can begin to identify relationships between fires and air quality. These are just some of the many analyses you can explore using ancillary datasets that are already on hand in the data catalog.

Conclusion

Earth Engine provides access to multiple fire monitoring datasets that are useful for tracking active fires throughout fire seasons and retrospectively for prior years. In this chapter, you examined one fire using three datasets (MODIS active fire, MODIS burned areas, and GOES active fire). You learned how to access, visualize, and analyze the raster datasets by adjusting code in the Code Editor and interacting with the data in premade user apps. By comparing the fire mapping characteristics of each dataset, you learned how to weigh the pros and cons of existing datasets for meeting fire mapping objectives. Now that you understand the basics of what we look for in fire mapping, you can compare additional fire datasets available in Earth Engine, or explore how to make your dataset using satellite and other geospatial data.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Andela N, Morton DC, Giglio L, et al (2017) A human-driven decline in global burned area. Science 356:1356–1362. https://doi.org/10.1126/science.aal4108

Andela N, Morton DC, Giglio L, et al (2019) The Global Fire Atlas of individual fire size, duration, speed and direction. Earth Syst Sci Data 11:529–552. https://doi.org/10.5194/essd-11-529-2019

Archibald S, Roy DP, van Wilgen BW, Scholes RJ (2009) What limits fire? An examination of drivers of burnt area in Southern Africa. Glob Chang Biol 15:613–630. https://doi.org/10.1111/j.1365-2486.2008.01754.x

Chuvieco E, Yue C, Heil A, et al (2016) A new global burned area product for climate assessment of fire impacts. Glob Ecol Biogeogr 25:619–629. https://doi.org/10.1111/geb.12440

Crowley MA, Cardille JA, White JC, Wulder MA (2019) Generating intra-year metrics of wildfire progression using multiple open-access satellite data streams. Remote Sens Environ 232:111295. https://doi.org/10.1016/j.rse.2019.111295

Crowley MA, Cardille JA, White JC, Wulder MA (2019) Multi-sensor, multi-scale, Bayesian data synthesis for mapping within-year wildfire progression. Remote Sens Lett 10:302–311. https://doi.org/10.1080/2150704X.2018.1536300

Giglio L (2010) MODIS collection 5 active fire product user's guide version 2.4. Sci. Cyst. Appl. Incl. University of Maryland

Giglio L, Schroeder W, Justice CO (2016) The collection 6 MODIS active fire detection algorithm and fire products. Remote Sens Environ 178:31–41. https://doi.org/10.1016/j.rse.2016.02.054

Hall JV, Zhang R, Schroeder W, et al (2019) Validation of GOES-16 ABI and MSG SEVIRI active fire products. Int J Appl Earth Obs Geoinf 83:101928. https://doi.org/10.1016/j.jag.2019.101928

Hawbaker TJ, Vanderhoof MK, Beal YJ, et al (2017) Mapping burned areas using dense time-series of Landsat data. Remote Sens Environ 198:504–522. https://doi.org/10.1016/j.rse.2017.06.027

Hermosilla T, Wulder MA, White JC, et al (2018) Disturbance-informed annual land cover classification maps of Canada's forested ecosystems for a 29-year Landsat time series. Can J Remote Sens 44:67–87. https://doi.org/10.1080/07038992.2018.1437719

Humber ML, Boschetti L, Giglio L, Justice CO (2019) Spatial and temporal intercomparison of four global burned area products. Int J Digit Earth 12:460–484. https://doi.org/10.1080/17538947.2018.1433727

Jolly WM, Cochrane MA, Freeborn PH, et al (2015) Climate-induced variations in global wildfire danger from 1979 to 2013. Nat Commun 6:1–11. https://doi.org/10.1038/ncomms8537

Liu T, Crowley MA (2021) Detection and impacts of tiling artifacts in MODIS burned area classification. IOP SciNotes 2:014003. https://doi.org/10.1088/2633-1357/abd8e2

Nogueira JMP, Ruffault J, Chuvieco E, Mouillot F (2017) Can we go beyond burned area in the assessment of global remote sensing products with fire patch metrics? Remote Sens 9:7. https://doi.org/10.3390/rs9010007

Parks SA (2014) Mapping day-of-burning with coarse-resolution satellite fire-detection data. Int J Wildl Fire 23:215–223. https://doi.org/10.1071/WF13138

Parks SA, Holsinger LM, Koontz MJ, et al (2019) Giving ecological meaning to satellite-derived fire severity metrics across North American forests. Remote Sens 11:1735. https://doi.org/10.3390/rs11141735

Restif BC, Hoffman A, Engineers SS, Response C (2020) How to generate wildfire boundary maps with Earth Engine. In: Google Earth and Earth Engine. https://medium.com/google-earth/how-to-generate-wildfire-boundary-maps-with-earth-engine-b38eadc97a38. Accessed 1 Oct 2020

Schroeder W, Prins E, Giglio L, et al (2008) Validation of GOES and MODIS active fire detection products using ASTER and ETM+ data. Remote Sens Environ 112:2711–2726. https://doi.org/10.1016/j.rse.2008.01.005

Stinson G, Kurz WA, Smyth CE, et al (2011) An inventory-based analysis of Canada's managed forest carbon dynamics, 1990 to 2008. Glob Chang Biol 17:2227–2244. https://doi.org/10.1111/j.1365-2486.2010.02369.x

Veraverbeke S, Sedano F, Hook SJ, et al (2014) Mapping the daily progression of large wildland fires using MODIS active fire data. Int J Wildl Fire 23:655–667. https://doi.org/10.1071/WF13015

Chapter A3.2: Mangroves

Author
Aurélie Shapiro

Overview

Mangrove ecosystems are tropical coastal forests that are adapted to saltwater environments. Their unique qualities of existing primarily in moist environments at low elevation along shorelines, lack of seasonality, and compact pattern make them relatively easy to identify in satellite images. In this chapter we present a series of automated steps, including water masking, to extract mangroves from a fusion of optical and active radar data. Furthermore, as global mangrove datasets are readily available in Google Earth Engine, we present an approach to automatically extract training data from existing information, saving time and effort in your supervised classification. The method can then be adapted to create subsequent maps from your own results to produce changes in mangrove ecosystems over time.

Learning Outcomes

- Fusing Sentinel-1 and -2 optical/radar sensors.
- Sampling points on an image to create training and testing datasets.
- Calculating additional indices to add to the image stack.
- Applying an automatic water masking function and buffering to focus classification on coastal areas likely to have mangroves.
- Understanding supervised classification with random forests using automatically derived training data.
- Evaluating training and model accuracy.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Perform pixel-based supervised or unsupervised classification (Chap. F2.1).
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Perform image morphological operations (Chap. F3.2).

- Create or access image mosaics (Chap. F4.3).
- Interpret Otsu's method for partitioning a histogram (Chap. A2.3).

Introduction to Theory

Mangrove ecosystems are highly productive environments that provide essential services, notably the storage of blue carbon, stabilization and protection of coastlines from storms and coastal events, and the provision of nurseries for fish. Mangrove ecosystems also enhance associated coral reef ecosystems, which are crucial to supporting local livelihoods (Bryan-Brown et al. 2020). Mangrove forests consist of specialized species adapted to saltwater environments located in tropical and subtropical latitudes. Over 1.3 billion people live in tropical coastal areas and rely on mangrove ecosystems for their health, safety, and livelihood. It is important to map, monitor, and quantify their change over time in order to properly conserve and restore them.

In this chapter, we will review the basic process for mapping mangroves using Sentinel-1 and -2 imagery. Mangroves are particularly recognizable in satellite imagery by their wetness—these ecosystems thrive at the water/land interface, making them easy to distinguish with satellite sensors that are sensitive to vegetation and water. We use sensor fusion to combine the advantages of multiple sensor types. For optical data, we extract relevant vegetation and water indices to discern mangroves, and we use active radar data for its capacity to detect water and derive canopy texture information.

In this chapter, we will show you how to evaluate mangroves at 10 m resolution using a fusion of optical and radar sensors (Sentinel-1 and -2) and derived indices. We will also implement automatic water masking (see also Chap. A2.3) and a random forest machine-learning supervised (see Chaps. F2.1 and F2.2) classification, to which you can potentially add your own improvements as needed.

If you are interested in learning more about mangrove mapping with Earth Engine, a thorough workflow for analyzing Landsat imagery for mangroves with a light graphic user interface is presented as the Google Earth Engine Mangrove Mapping Methodology (GEMMM) in Yancho et al. (2020). The advantages of this approach are the evaluation of the shoreline buffer areas, high- and low-tide imagery, user-friendly interface, and the freely available and well-explained code.

Practicum

Several assets are provided for you to work with. As a first step, define the area of interest (`aoi`) and view it. In this case, we choose the Sundarbans ecosystem on the border of India and Bangladesh, which is an iconic mangrove forest (known for its mysterious native tiger population) and a simple example to showcase water masking and mangrove mapping:

```
// Create an ee.Geometry.
var aoi = ee.Geometry.Polygon([
    [
        [88.3, 22.61],
        [90, 22.61],
        [90, 21.47],
        [88.3, 21.47]
    ]
]);

// Locate a coordinate in the aoi with land and water.
var point = ee.Geometry.Point([89.2595, 21.7317]);

// Position the map.
Map.centerObject(point, 13);
Map.addLayer(aoi, {}, 'AOI');

// Sentinel-1 wet season data.
var wetS1 = ee.Image(
    'projects/gee-book/assets/A3-2/wet_season_tscan_2020');
// Sentinel-1 dry season data.
var dryS1 = ee.Image(
    'projects/gee-book/assets/A3-2/dry_season_tscan_2020');
// Sentinel-2 mosaic.
var S2 =
ee.Image('projects/gee-book/assets/A3-2/Sundarbans_S2_2020');
```

We will fuse radar and optical data at 10 m resolution for this exercise using multitemporal composites that were developed using the Food and Agriculture

Organization (FAO) freely available SEPAL, which is a cloud-based image and data processing platform that has several modules built on Earth Engine. The available recipes let you choose dates and processing parameters, and export composites directly to your Earth Engine account. For radar, we used SEPAL to produce two composite images created from multi-temporal statistics derived from Sentinel-1 data, which is a compilation of all filtered, terrain-corrected images that are available in Earth Engine (Esch et al. 2018, Mulissa et al. 2021, Vollrath et al. 2020). Additionally, the SEPAL platform calculates statistics (standard deviation, minimum, maximum). We developed two timescan images for the 2020 dry (March to September) and wet (October to April) seasons of the study area.

The Sentinel-2 optical composite was also generated in SEPAL, applying the bidirectional reflectance distribution function (BRDF) correction to surface reflectance corrected images, producing the median value of all cloud-free pixels for 2020.

You will now access both exported SEPAL composites using the Code Editor.

```
//Visualize the input data.
var s1VisParams = {
    bands: ['VV_min', 'VH_min', 'VVVH_ratio_min'],
    min: -36,
    max: 3
};
var s2VisParams = {
    bands: ['swir1', 'nir', 'red'],
    min: 82,
    max: 3236
};

Map.addLayer(dryS1, s1VisParams, 'S1 dry', false);
Map.addLayer(wetS1, s1VisParams, 'S1 wet', false);
Map.addLayer(S2, s2VisParams, 'S2 2020');
```

Section 1. Deriving Additional Indices

It's a good idea to complement your data stack with additional indices (see Chap. F2.0) that are relevant to mangroves—such as indices sensitive to water, greenness, and

vegetation (see Wang et al. 2018). You can add these via band calculations and equations, and we will present several here. But the list of indices is virtually endless; what is useful can depend on the location.

To compute the normalized vegetation index (NDVI) using an existing Earth Engine NDVI function, add this line:

```
var NDVI = S2.normalizedDifference(['nir', 'red']).rename(['NDVI']);
```

You can also use an image expression (see Chap. F3.1) for any calculation, such as a band ratio:

```
var ratio_swir1_nir = S2.expression(
        'swir1/(nir+0.1)', {
            'swir1': S2.select('swir1'),
            'nir': S2.select('nir')
        })
    .rename('ratio_swir1_nir_wet');
```

You add the `rename` function so you can recognize the band more easily in your data stack. You create a data stack by adding the different indices to the input image by using `addBands` and the name of the index or expression. Don't forget to add your Sentinel-1 data too:

```
var data_stack = S2.addBands(NDVI).addBands(ratio_swir1_nir).addBands(
    dryS1).addBands(wetS1).addBands(S2);
```

And finally, you can see the names of all your bands by entering:

```
print(data_stack);
```

Question 1. What other indices could be useful for mapping mangroves?

There are a number of useful articles on mangrove mapping; if you know how they are calculated, you can add new indices with image expressions.

Code Checkpoint A32a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Automatic Water Masking and Buffering

As explained above, mangroves are found close to shores, which tend to be at low elevations. The next steps involve automatic water masking to delineate land from sea; then we will use an existing dataset to buffer the area of interest so that we are only mapping mangroves where we would expect to find them.

We will use the Canny edge detector and Otsu thresholding (Donchyts et al. 2016) approach to automatically detect water. For this we use the point provided at the beginning of the script that is near land and water. The function will then automatically identify an appropriate threshold that delineates land pixels from water, based on the calculation of edges in a selected region with both land and water. This approach is also seen in Chap. A2.3, using different parameters and settings, where it is described in detail.

Paste the code below to add functionality that can compute the threshold, detect edges, and create the water mask:

```
/***
 * This script computes surface water mask using
 * Canny Edge detector and Otsu thresholding.
 * See the following paper for details:
 * http://www.mdpi.com/2072-4292/8/5/386
 *
 * Author: Gennadii Donchyts
 * Contributors: Nicholas Clinton
 *
 */

/***
 * Return the DN that maximizes interclass variance in B5 (in the
region).
 */
var otsu = function(histogram) {
```

```
    histogram = ee.Dictionary(histogram);

    var counts = ee.Array(histogram.get('histogram'));
    var means = ee.Array(histogram.get('bucketMeans'));
    var size = means.length().get([0]);
    var total = counts.reduce(ee.Reducer.sum(), [0]).get([0]);
    var sum = means.multiply(counts).reduce(ee.Reducer.sum(), [0])
        .get([0]);
    var mean = sum.divide(total);

    var indices = ee.List.sequence(1, size);

    // Compute between sum of squares, where each mean partitions the
data.
    var bss = indices.map(function(i) {
        var aCounts = counts.slice(0, 0, i);
        var aCount = aCounts.reduce(ee.Reducer.sum(), [0])
            .get([0]);
        var aMeans = means.slice(0, 0, i);
        var aMean = aMeans.multiply(aCounts)
            .reduce(ee.Reducer.sum(), [0]).get([0])
            .divide(aCount);
        var bCount = total.subtract(aCount);
        var bMean = sum.subtract(aCount.multiply(aMean))
            .divide(bCount);
        return aCount.multiply(aMean.subtract(mean).pow(
            2)).add(
            bCount.multiply(bMean.subtract(mean).pow(
                2)));
    });

    // Return the mean value corresponding to the maximum BSS.
    return means.sort(bss).get([-1]);
};

/***
 * Compute a threshold using Otsu method (bimodal).
```

```
 */

function computeThresholdUsingOtsu(image, scale, bounds,
    cannyThreshold,
    cannySigma, minValue, debug) {
    // Clip image edges.
    var mask = image.mask().gt(0)
        .focal_min(ee.Number(scale).multiply(3), 'circle', 'meters');

    // Detect sharp changes.
    var edge = ee.Algorithms.CannyEdgeDetector(image, cannyThreshold,
        cannySigma);
    edge = edge.multiply(mask);

    // Buffer around NDWI edges.
    var edgeBuffer = edge
        .focal_max(ee.Number(scale).multiply(1), 'square', 'meters');
    var imageEdge = image.mask(edgeBuffer);

    // Compute threshold using Otsu thresholding.
    var buckets = 100;
    var hist = ee.Dictionary(ee.Dictionary(imageEdge
            .reduceRegion({
                reducer: ee.Reducer.histogram(buckets),
                geometry: bounds,
                scale: scale,
                maxPixels: 1e9
            }))
        .values()
        .get(0));

    var threshold = ee.Number(ee.Algorithms.If({
        condition: hist.contains('bucketMeans'),
        trueCase: otsu(hist),
        falseCase: 0.3
    }));
```

```
    if (debug) {
        Map.addLayer(edge.mask(edge), {
            palette: ['ff0000']
        }, 'edges', false);
        print('Threshold: ', threshold);
        print(ui.Chart.image.histogram(image, bounds, scale,
            buckets));
        print(ui.Chart.image.histogram(imageEdge, bounds, scale,
            buckets));
    }

    return minValue !== 'undefined' ? threshold.max(minValue) :
        threshold;
}

var bounds = ee.Geometry(Map.getBounds(true));

var image = data_stack;
print('image', image);

var ndwi_for_water = image.normalizedDifference(['green', 'nir']);
var debug = true;
var scale = 10;
var cannyThreshold = 0.9;
var cannySigma = 1;
var minValue = -0.1;
var th = computeThresholdUsingOtsu(ndwi_for_water, scale, bounds,
    cannyThreshold, cannySigma, minValue, debug);

print('th', th);

function getEdge(mask) {
    return mask.subtract(mask.focal_min(1));
}

var water_mask = ndwi_for_water.mask(ndwi_for_water.gt(th));
```

```
th.evaluate(function(th) {
    Map.addLayer(water_mask, {
        palette: '0000ff'
    }, 'water mask (th=' + th + ')');
});
```

You'll notice that new layers are loaded in the map, which include the edge detection and a water mask that identifies all marine and surface water (Fig. A3.2.1).

Question 2. Is the point well located to appropriately identify the water mask? What happens with turbid water?

Move the point around and see if it improves the automatic water masking. Turbid water can be an issue and may not be detected by the mask. Be certain that these muddy waters are not classified as mangrove later on.

Fig. A3.2.1 The automatic water mask identifies all open and surface water pixels

Next, we create the land mask by inverting the water mask, and removing any areas with elevation greater than 40 m above sea level using the NASADEM (Digital Elevation Model from NASA) data collection. This will ensure we aren't erroneously mapping mangroves far inland, where they don't occur. You can of course change the elevation threshold according to your study area.

```
// Create land mask area.
var land = water_mask.unmask();
var land_mask = land.eq(0);
Map.addLayer(land_mask, {}, 'Land mask', false);

// Remove areas with elevation greater than mangrove elevation
threshold.
var elev_thresh = 40;
var dem = ee.Image('NASA/NASADEM_HGT/001').select('elevation');
var elev_mask = dem.lte(elev_thresh);
var land_mask = land_mask.updateMask(elev_mask);
```

Next, we will buffer the area of interest to map mangroves only in areas where they might realistically be found. For this we will use the Global Mangrove Dataset from 2000 available in Earth Engine (note: this is one of several mangrove datasets; you could also have used other datasets such as Global Mangrove Watch or any other available raster data.)

The Global Mangrove Dataset was derived from Landsat 2000 (Giri et al. 2011); we will buffer 1000 m around it. The 1000 m buffer allows for the possibility that some mangroves were missed in the original map, and that mangroves might have expanded in some areas since 2000. You can change the buffer distance to any value suitable for your study area.

```
// Load global mangrove dataset as reference for training.
var mangrove_ref = ee.ImageCollection('LANDSAT/MANGROVE_FORESTS')
    .filterBounds(aoi)
    .first()
    .clip(aoi);
Map.addLayer(mangrove_ref, {
    palette: 'Green'
}, 'Reference Mangroves', false);

// Buffer around known mangrove area with a specified distance.
var buffer_dist = 1000;
var mang_buffer = mangrove_ref
    .focal_max(buffer_dist, 'square', 'meters')
```

```
    .rename('mangrove_buffer');
Map.addLayer(mang_buffer, {}, 'Mangrove Buffer', false);
```

Question 3. Can the buffer distance or elevation threshold be changed to capture more mangroves or remove extra areas where mangroves aren't likely to be found - and that we don't need to classify? We don't want to miss any mangrove areas, but we also want an efficient code that does not process areas that can't be mangroves, which can add processing complexity and commit easily avoidable errors. To restrict the processing to the potential mangrove area, can you change the buffer distance or elevation threshold to best capture the area you are interested in?

We will now mask the mangrove buffer, create the area to classify and mask it from the data stack.

```
// Mask land from mangrove buffer.
var area_to_classify = mang_buffer.updateMask(land_mask).selfMask();
Map.addLayer(area_to_classify,
    {},
    'Mangrove buffer with water and elevation mask',
    false);
var image_to_classify = data_stack.updateMask(area_to_classify);
Map.addLayer(image_to_classify,
    {
        bands: ['swir1', 'nir', 'red'],
        min: 82,
        max: 3236
    },
    'Masked Data Stack',
    false);
```

Code Checkpoint A32b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Creating Training Data and Running and Evaluating a Random Forest Classification

We will now automatically select mangrove and non-mangrove areas as training data. We use morphological image processing (see Chap. F3.2) to select areas deep inside the reference mangrove dataset; these are areas we can be sure are mangroves, because mangrove forests tend to be lost or deforested at the edges rather than in the interior. Using the same theory, we select areas far away from mangroves as our non-forest areas. This approach allows us to use a relatively older dataset from 2000 for current data training. We will extract mangrove and non-mangrove from the reference data.

```
// Create training data from existing data
// Class values: mangrove = 1, not mangrove = 0
var ref_mangrove = mangrove_ref.unmask();
var mangroveVis = {
    min: 0,
    max: 1,
    palette: ['grey', 'green']
};
Map.addLayer(ref_mangrove, mangroveVis, 'mangrove = 1');

// Class values: not mangrove = 1 and mangrove = 0
var notmang = ref_mangrove.eq(0);
var notMangroveVis = {
    min: 0,
    max: 1,
    palette: ['grey', 'red']
};
Map.addLayer(notmang, notMangroveVis, 'not mangrove = 1', false);
```

We then use erosion and dilation (as described in Chap. F3.2) to identify areas at the center of mangrove forests and far from outside edges. We put everything together in a training layer where mangroves = 1, non-mangroves = 2, and everything else = 0.

```
// Define a kernel for core mangrove areas.
var kernel = ee.Kernel.circle({
```

```
    radius: 3
});

// Perform a dilation to identify core mangroves.
var mang_dilate = ref_mangrove
    .focal_min({
        kernel: kernel,
        iterations: 3
    });
var mang_dilate = mang_dilate.updateMask(mang_dilate);
var mang_dilate = mang_dilate.rename('auto_train').unmask();
Map.addLayer(mang_dilate, {}, 'Core mangrove areas to sample', false);

// Do the same for non-mangrove areas.
var kernel1 = ee.Kernel.circle({
    radius: 3
});
var notmang_dilate = notmang
    .focal_min({
        kernel: kernel1,
        iterations: 2
    });
var notmang_dilate = notmang_dilate.updateMask(notmang_dilate);
var notmang_dilate = notmang_dilate.multiply(2).unmask().rename(
    'auto_train');
Map.addLayer(notmang_dilate, {}, 'Not mangrove areas to sample',
    false);

// Core mangrove = 1, core non mangrove = 2, neither = 0.
var train_labels = notmang_dilate.add(mang_dilate).clip(aoi);
var train_labels = train_labels.int8().updateMask(area_to_classify);
var trainingVis = {
    min: 0,
    max: 2,
    palette: ['grey', 'green', 'red']
};
Map.addLayer(train_labels, trainingVis, 'Training areas', false);
```

Question 4. How do the kernel radius and iteration parameters identify or miss important core mangrove areas?

To obtain good training data, we want samples located throughout the area of interest. Sometimes, if the mangroves are in very small patches, if the radius is too large, or if there are too many iterations, we don't end up with enough core forest to sample.

Change the parameters to see what works best for you. You may need to zoom in to see the core mangrove areas.

The next step is the classification (see Chap. F2.1). We will collect random samples from the training areas to train and run the random forest classifier. We will conduct two classifications. One is for validation, to obtain the test accuracy and determine how consistent the training areas are between two random samples.

```
// Begin Classification.
// Get image and bands for training - including automatic training
band.
var trainingImage = image_to_classify.addBands(train_labels);
var trainingBands = trainingImage.bandNames();
print(trainingBands, 'training bands');

// Get training samples and classify.
// Select the number of training samples per class.
var numPoints = 2000;
var numPoints2 = 2000;

var training = trainingImage.stratifiedSample({
    numPoints: 0,
    classBand: 'auto_train',
    region: aoi,
    scale: 100,
    classValues: [1, 2],
    classPoints: [numPoints, numPoints2],
    seed: 0,
    dropNulls: true,
    tileScale: 16,
```

```
});

var validation = trainingImage.stratifiedSample({
    numPoints: 0,
    classBand: 'auto_train',
    region: aoi,
    scale: 100,
    classValues: [1, 2],
    classPoints: [numPoints, numPoints2],
    seed: 1,
    dropNulls: true,
    tileScale: 16,
});

// Create a random forest classifier and train it.
var nTrees = 50;
var classifier = ee.Classifier.smileRandomForest(nTrees)
    .train(training, 'auto_train');

var classified = image_to_classify.classify(classifier);

// Classify the test set.
var validated = validation.classify(classifier);

// Get a confusion matrix representing resubstitution accuracy.
var trainAccuracy = classifier.confusionMatrix();
print('Resubstitution error matrix: ', trainAccuracy);
print('Training overall accuracy: ', trainAccuracy.accuracy());
var testAccuracy = validated.errorMatrix('mangrove',
    'classification');
```

The training accuracy is over 99%, which is very good. According to the substitution matrix, only a few training points seem to be confused when they are randomly replaced.

In addition, we can estimate variable importance, or how much each band contributes to the final random forest model. These are always good metrics to observe, as you could remove the least important bands from the training image if you wanted to.

```
var dict = classifier.explain();
print('Explain:', dict);
var variable_importance = ee.Feature(null, ee.Dictionary(dict).get(
    'importance'));

// Chart variable importance.
var chart = ui.Chart.feature.byProperty(variable_importance)
    .setChartType('ColumnChart')
    .setOptions({
        title: 'Random Forest Variable Importance',
        legend: {
            position: 'none'
        },
        hAxis: {
            title: 'Bands'
        },
        vAxis: {
            title: 'Importance'
        }
    });
print(chart);
```

Question 5. What are the most important bands in the classification model?

Question 6. Based on the chart, is one of the sensors more important in the classification model? Which bands? Why do you think that is?

Next, we will visualize the final classification. We can apply a filter to remove individual pixels, which effectively applies a minimum mapping unit (MMU). In this case, any areas with fewer than 25 connected pixels are filtered out.

```
var classificationVis = {
    min: 1,
    max: 2,
    palette: ['green', 'grey']
};
Map.addLayer(classified, classificationVis,
```

```
    'Mangrove Classification');

// Clean up results to remove small patches/pixels.
var mang_only = classified.eq(1);
// Compute the number of pixels in each connected mangrove patch
// and apply the minimum mapping unit (number of pixels).
var mang_patchsize = mang_only.connectedPixelCount();

//mask pixels based on the number of connected neighbors
var mmu = 25;
var mang_mmu = mang_patchsize.gte(mmu);
var mang_mmu = classified.updateMask(mang_mmu).toInt8();
Map.addLayer(mang_mmu, classificationVis, 'Mangrove Map MMU');
```

Your map window should resemble Fig. A3.2.2 with mangroves in green, and non-mangroves in gray.

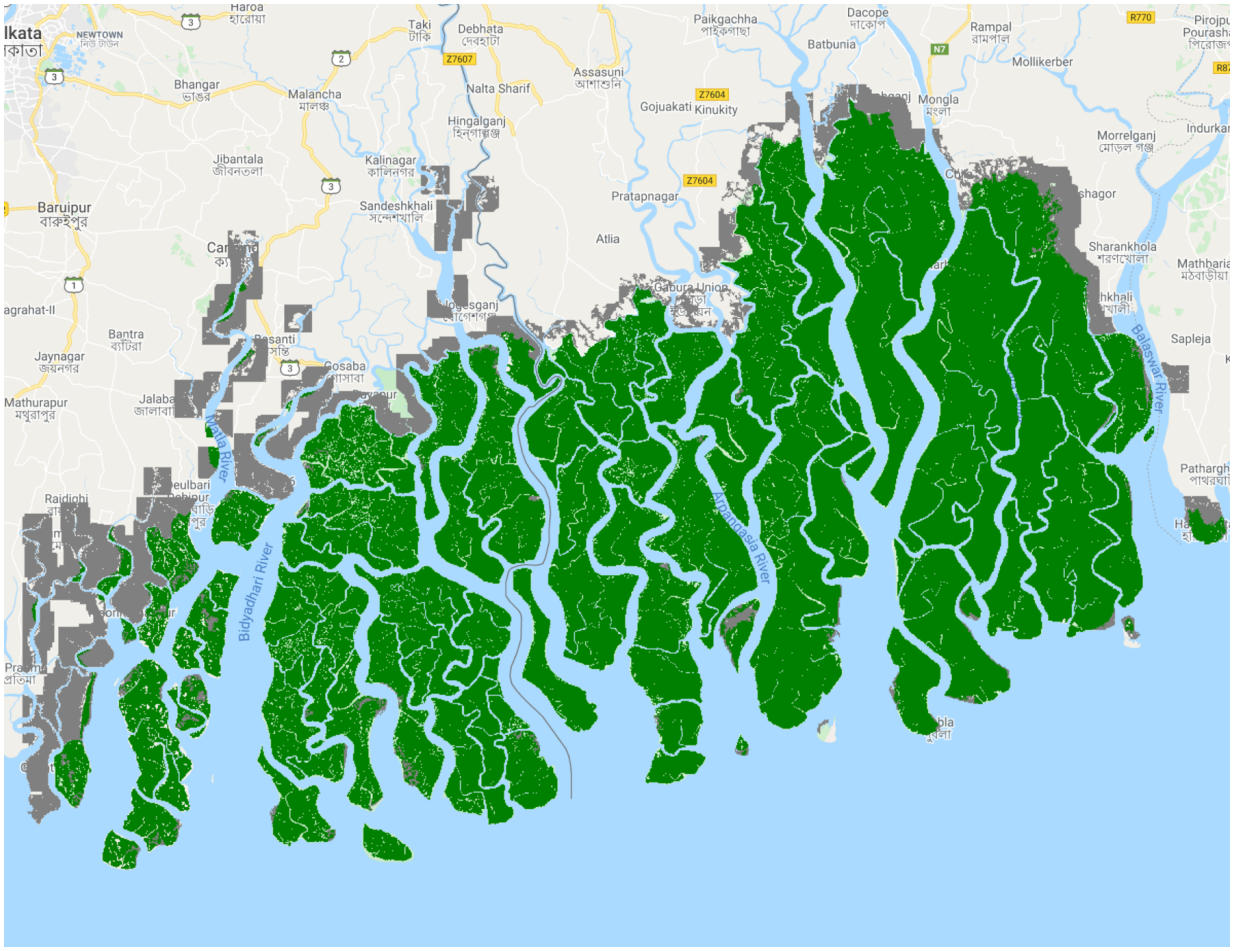


Fig. A3.2.2 The final mangrove classification

Code Checkpoint A32c. The book's repository contains a script that shows what your code should look like at this point.

In the **Console** window, you'll see the substitution matrix, training accuracy, and a chart of variable importance.

Question 7. Do you notice any errors in the map? Omissions or commissions? How does the map compare to 2000?

Question 8. You should be able to see small differences in the mangrove extent since 2000. What do you see? What could be the causes for the changes?

Question 9. How does the MMU parameter change the output map?
Try out different values and see what the results look like.

Synthesis

Assignment 1. With what you learned in this chapter, you can fuse Sentinel-1 and -2 data to create your own map of mangroves for anywhere in the world. You might now test out the approach in another part of the world, or use your own training data for a more refined classification model. You can add your own point data in the map and merge with the training data to improve a classification, or clean areas that need to be removed by drawing polygons and masking them in the classification.

Assignment 2. You can also add more indices, remove ones that aren't as informative, create a map using earlier imagery, use the later date classification as reference data, and look for differences via post-classification change. Additional indices can be found in script **A32s1 - Supplemental** in the book's repository. Using these other indices, does that change your estimation of what areas are stable mangrove, and where have we observed gains or losses?

Conclusion

Mangroves are dynamic ecosystems that can expand over time, and that also can be lost through natural and anthropogenic causes. The power of Earth Engine lies in the cloud-based, lightning-fast, automated approach to workflows, particularly with automated training data collection. This process would take days when performed offline in traditional remote sensing software, especially over large areas. The Earth Engine approach is not only fast but also consistent. The same method can be applied to images from different dates to assess mangrove changes over time—both gain and loss.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Donchyts G, Schellekens J, Winsemius H, et al (2016) A 30 m resolution surface water mask including estimation of positional and thematic differences using Landsat 8, SRTM and OpenStreetMap: A case study in the Murray-Darling basin, Australia. Remote Sens 8:386. https://doi.org/10.3390/rs8050386

Esch T, Üreyen S, Zeidler J, et al (2018) Exploiting big Earth data from space–first experiences with the TimeScan processing chain. Big Earth Data 2:36–55. https://doi.org/10.1080/20964471.2018.1433790

Giri C, Ochieng E, Tieszen LL, et al (2011) Status and distribution of mangrove forests of the world using Earth observation satellite data. Glob Ecol Biogeogr 20:154–159. https://doi.org/10.1111/j.1466-8238.2010.00584.x

Mullissa A, Vollrath A, Odongo-Braun C, et al (2021) Sentinel-1 SAR backscatter analysis ready data preparation in Google Earth Engine. Remote Sens 13:1954. https://doi.org/10.3390/rs13101954

Vollrath A, Mullissa A, Reiche J (2020) Angular-based radiometric slope correction for Sentinel-1 on Google Earth Engine. Remote Sens 12:1867. https://doi.org/10.3390/rs12111867

Wang D, Wan B, Qiu P, et al (2018) Evaluating the performance of Sentinel-2, Landsat 8 and Pléiades-1 in mapping mangrove extent and species. Remote Sens 10:1468. https://doi.org/10.3390/rs10091468

Yancho JMM, Jones TG, Gandhi SR, et al (2020) The Google Earth Engine mangrove mapping methodology (GEEMMM). Remote Sens 12:1–35. https://doi.org/10.3390/rs12223758

Chapter A3.3: Mangroves II - Change Mapping

Authors

Celio de Sousa, David Lagomasino, and Lola Fatoyinbo

Overview

The purpose of this chapter is to present two methods of land cover extent change detection: map-to-map and map-to-image using anomaly data. In a map-to-map approach, changes are extracted by subtracting a two-date pair of land cover extent maps: that is, time 2 (T2) extent minus time 1 (T1) extent. By comparison, a map-to-image approach uses a baseline extent map within which changes are calculated based on a T2 image where the change classes are defined by threshold values. In this lab, we will perform a map-to-map change detection between two mangrove extent maps from 2000 and 2020, and also perform a vegetation index anomaly analysis to detect changes within a mangrove extent map from the year 2000 in Guinea, West Africa.

Learning Outcomes

- Performing change detection by contrasting two preexistent mangrove extent maps.
- Harmonizing across Landsat generations to create consistent long-term series
- Calculating the anomaly of a given vegetation index based on its long-term average value.
- Performing change detection by interpreting vegetation index anomalies

Helps if you know how to:

- Recognize similarities and differences among satellite spectral bands (Part F1, Part F2, Part F3).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Use `normalizedDifference` to calculate vegetation indices (Chap. F2.0).
- Perform a supervised image classification (Chap. F2.1).
- Work with array images (Chap. F3.1, Chap. F4.6).
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Summarize an image with `reduceRegion` (Chap. F3.1).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).
- Perform a two-period change detection (Chap. F4.4).

Introduction to Theory

Mangrove forests are among the most productive ecosystems on Earth, providing a wide range of critical economic, ecological, and societal services. However, mangroves worldwide have undergone intense conversion, with human-related disturbances being one of the main causes of global mangrove loss in recent decades (Goldberg 2020). Thus, regular monitoring of the dynamics of mangrove ecosystems worldwide is key for establishing better coastal management policies and mangrove conservation initiatives.

Readily available, low-cost, and repeated-coverage remote sensing data provides numerous advantages for monitoring mangrove forests and detecting changes in the landscape over time. In this context, the Landsat data archive, which covers more than 35 years, has been widely used for mapping land cover dynamics worldwide. Most notably, the Landsat mission coupled with Google Earth Engine's computing infrastructure have been leveraged to develop widely used global datasets at 30 m spatial resolution, such as the global forest cover (Hansen et al. 2013) and global mangrove extent (Bunting et al. 2018) datasets.

However, as extensively explored in the literature, change detection using remotely sensed data is challenged by several factors, including (but not limited to) the following: spatial, spectral, thematic, and temporal constraints, atmospheric conditions, high cloud-coverage, and differences in sensors' spectral characteristics. These differences may have a direct effect on the ability to accurately detect and monitor changes in the landscape, including mangrove forests. As explored by Roy et al. (2016), the Landsat TM/ETM+ and OLI sensors present small but potentially significant differences between their spectral characteristics, with the greatest differences in the near-infrared (NIR) and the shortwave infrared (SWIR) bands—the most important spectral bands for vegetation studies.

In this chapter, we take advantage of the statistical functions presented in Roy et al. (2016) to transform between the comparable TM/ETM+ and OLI bands in order to ensure inter-sensor harmonized spectral information and temporal continuity. We will use this harmonized Landsat TM/ETM+/OLI collection to derive anomaly-based changes in mangrove extent over 21 years (2000–2020) in Guinea, West Africa.

Practicum

Section 1. Map-to-Map Change Detection

Previous chapters demonstrated how to perform a general supervised classification, as well as how to apply those classifiers and functions to create a mangrove extent map (Chap. A3.2). In this section, we will build upon those techniques to perform a map-to-map change analysis. Paste the code below into a new script, which will access pre-created assets for two time periods: 2000 and 2020.

```
var areaOfstudy = ee.FeatureCollection(
    'projects/gee-book/assets/A3-3/Border5km');
var mangrove2000 = ee.Image(
    'projects/gee-book/assets/A3-3/MangroveGuinea2000_v2');
var mangrove2020 = ee.Image(
    'projects/gee-book/assets/A3-3/MangroveGuinea2020_v2');
```

Start by setting the map center around Conakry, Guinea, and adding your 2000 and 2020 mangrove extent to the map using a color of your choice and naming them `'Mangrove Extent 2000'` and `'Mangrove Extent 2020'`:

```
Map.setCenter(-13.6007, 9.6295, 10);
// Sets the map center to Conakry, Guinea
Map.addLayer(areaOfstudy, {}, 'Area of Study');
Map.addLayer(mangrove2000, {
    palette: '#16a596'
}, 'Mangrove Extent 2000');
Map.addLayer(mangrove2020, {
    palette: '#9ad3bc'
}, 'Mangrove Extent 2020');
```

Because the assets have the value of 1 (one) assigned to mangrove pixels and 0 (zero) to everything else, you can derive losses and gains from both mangrove extent maps with a simple subtraction of T1 (2000) from T2 (2020). The value of 1 has been assigned to mangrove pixels and everything else has been masked. For the mathematical operation between the two layers to work, every pixel has to have a value assigned to it.

In this case, you can unmask previously masked pixels and assign the value 0 to them using `unmask(0)`. Finally, subtract T1 from T2 into a new variable `change`:

```
var mang2020 = mangrove2020.unmask(0);
var mang2000 = mangrove2000.unmask(0);
var change = mang2020.subtract(mang2000)
    .clip(areaOfstudy);
```

Pixels in the raster `change` will have values of −1, 0, or 1, which represent loss/conversion, no change, and gains, respectively:

- `-1` = no mangroves in 2020, mangroves in 2000 (0 − 1 = −1);
- `0` = mangroves in 2020, mangroves in 2000 (1 − 1 = 0);
- `1` = mangroves in 2020, no mangroves in 2000 (1 − 0 = 1)

Finally, add `change` to the map:

```
var paletteCHANGE = [
    'red', // Loss/conversion
    'white', // No Change
    'green', // Gain/Expansion
];

Map.addLayer(change, {
    min: -1,
    max: 1,
    palette: paletteCHANGE
}, 'Changes 2000-2020');
```

You can calculate the area of expansion/conversion by isolating the pixels of gain/loss from the `change` into `gain` and `loss`. Then, calculate the area of each pixel using `ee.Image.pixelArea` and multiplying by the count of pixels in `gain` and `loss` using `multiply`. The default unit is square meters (m²). You can use `divide` to transform into square kilometers (`divide(1000000)`) or hectares (`divide(10000)`). Finally, use `ee.Reducer.sum` to sum all area values for both `gain` and `loss` and print them to the **Console** (Fig. A3.3.1).

```
// Calculate the area of each pixel
var gain = change.eq(1);
var loss = change.eq(-1);

var gainArea = gain.multiply(ee.Image.pixelArea().divide(1000000));
var lossArea = loss.multiply(ee.Image.pixelArea().divide(1000000));

// Sum all the areas
var statsgain = gainArea.reduceRegion({
    reducer: ee.Reducer.sum(),
    scale: 30,
    maxPixels: 1e14
});

var statsloss = lossArea.reduceRegion({
    reducer: ee.Reducer.sum(),
    scale: 30,
    maxPixels: 1e14
});

print(statsgain.get('classification'),
    'km² of new mangroves in 2020');
print(statsloss.get('classification'),
    'of mangrove was lost in 2020');

Map.addLayer(gain.selfMask(), {
    palette: 'green'
}, 'Gains');
Map.addLayer(loss.selfMask(), {
    palette: 'red'
}, 'Loss');
```

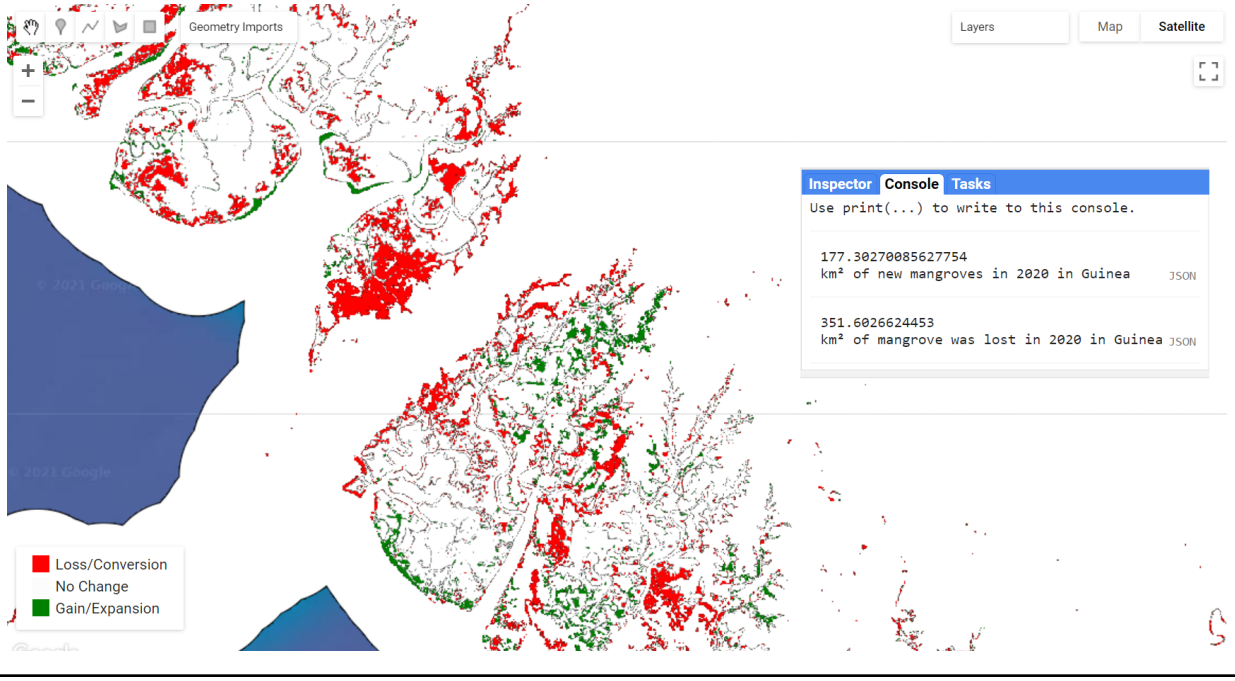


Fig. A3.3.1 Map-to-map changes in mangrove extent in Guinea from 2000–2020

Code Checkpoint A33a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. What are some of the issues that may arise when using a map-to-map change detection? Explain how these different dates may affect the classification output and, consequently, the change output.

Section 2. Map-to-Image Change Detection

Using the mangrove extent you created in the previous section, we will look at another way to detect changes without having to classify an image. In this case, change classes are defined by threshold values of a specific metric, such as a vegetation index or a spectral image band. In this section, we will take advantage of the Landsat archive to create an average reference value of a given vegetation index at an earlier date T1 and see how it compares to its later value at T2.

The first assumption of this approach is that changes will happen within a buffer zone from the baseline extent. Start by setting the baseline extent and buffer zone using `focal_max`:

```
var buffer = 1000; // In meters
var extentBuffer = mangrove2000.focal_max(buffer, 'circle', 'meters');
Map.addLayer(mangrove2000, {
    palette: '#000000'
}, 'Baseline', false);
Map.addLayer(extentBuffer, {
    palette: '#0e49b5',
    opacity: 0.3
}, 'Mangrove Buffer', false);
```

Harmonizing Landsat 5/7/8 Image Collections
As described in the beginning of this chapter, the Landsat TM/ETM+ and OLI sensors present differences between their spectral characteristics. Thus, to ensure inter-sensor harmonized spectral information and temporal continuity, we will harmonize the entire Landsat image archive using the statistical functions presented in Roy et al. (2016). For that, start by defining the temporal parameters for the harmonization:

```
var startYear = 1984;
var endyear = 2020;
var startDay = '01-01';
var endDay = '12-31';
```

Next, we will create several functions for the harmonization of the Landsat archive:

Harmonization Function: `harmonizationRoy` uses the regression coefficients (slopes and intercepts) retrieved from Roy et al. (2016) and performs a linear transformation of ETM+ spectral space to OLI spectral space:

```
var harmonizationRoy = function(oli) {
    var slopes = ee.Image.constant([0.9785, 0.9542, 0.9825,
        1.0073, 1.0171, 0.9949
    ]);
```

```
    var itcp = ee.Image.constant([-0.0095, -0.0016, -0.0022, -
        0.0021, -0.0030, 0.0029
    ]);
    var y = oli.select(['B2', 'B3', 'B4', 'B5', 'B6', 'B7'], [
            'B1', 'B2', 'B3', 'B4', 'B5', 'B7'
        ])
        .resample('bicubic')
        .subtract(itcp.multiply(10000)).divide(slopes)
        .set('system:time_start', oli.get('system:time_start'));
    return y.toShort();
};
```

Retrieve a Particular Sensor Function: `getSRcollection` will be used to retrieve individual sensor collections based on the temporal parameters and the harmonization function above. Additionally, this function will mask cloud, cloud shadow, and snow based on the Landsat quality assessment bands:

```
var getSRcollection = function(year, startDay, endYear, endDay,
    sensor) {
    var srCollection = ee.ImageCollection('LANDSAT/' + sensor +
            '/C01/T1_SR')
        .filterDate(year + '-' + startDay, endYear + '-' + endDay)
        .map(function(img) {
            var dat;
            if (sensor == 'LC08') {
                dat = harmonizationRoy(img.unmask());
            } else {
                dat = img.select(['B1', 'B2', 'B3', 'B4',
                        'B5', 'B7'
                    ])
                    .unmask()
                    .resample('bicubic')
                    .set('system:time_start', img.get(
                        'system:time_start'));
            }
            // Cloud, cloud shadow and snow mask.
            var qa = img.select('pixel_qa');
```

```
            var mask = qa.bitwiseAnd(8).eq(0).and(
                qa.bitwiseAnd(16).eq(0)).and(
                qa.bitwiseAnd(32).eq(0));
            return dat.mask(mask);
        });
    return srCollection;
};
```

Combining the Collections Function: `getCombinedSRcollection` will merge all the individual L5/L7/L8 collections into one:

```
var getCombinedSRcollection = function(year, startDay, endYear,
    endDay) {
    var lt5 = getSRcollection(year, startDay, endYear, endDay,
        'LT05');
    var le7 = getSRcollection(year, startDay, endYear, endDay,
        'LE07');
    var lc8 = getSRcollection(year, startDay, endYear, endDay,
        'LC08');
    var mergedCollection = ee.ImageCollection(le7.merge(lc8)
        .merge(lt5));
    return mergedCollection;
};
```

Vegetation Indices: `addIndices` calculates several vegetation/spectral indices based on the harmonized Landsat bands. In this example, we are including the Normalized Difference Vegetation Index (NDVI), the Enhanced Vegetation Index (EVI), the Soil Adjusted Vegetation Index (SAVI), the Normalized Difference Mangrove Index (NDMI), the Normalized Difference Water index (NDWI), and the Modified Normalized Difference Water Index (MNDWI). Here is where you can include your own vegetation indices:

```
var addIndices = function(image) {
    var ndvi = image.normalizedDifference(['B4', 'B3']).rename(
        'NDVI');
    var evi = image.expression(
        '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 1))', {
```

```
            'NIR': image.select('B4'),
            'RED': image.select('B3'),
            'BLUE': image.select('B1')
        }).rename('EVI');
    var savi = image.expression(
        '((NIR - RED) / (NIR + RED + 0.5) * (0.5 + 1))', {
            'NIR': image.select('B4'),
            'RED': image.select('B3'),
            'BLUE': image.select('B1')
        }).rename('SAVI');
    var ndmi = image.normalizedDifference(['B7', 'B2']).rename(
        'NDMI');
    var ndwi = image.normalizedDifference(['B5', 'B4']).rename(
        'NDWI');
    var mndwi = image.normalizedDifference(['B2', 'B5']).rename(
        'MNDWI');
    return image.addBands(ndvi)
        .addBands(evi)
        .addBands(savi)
        .addBands(ndmi)
        .addBands(ndwi)
        .addBands(mndwi);
};
```

Finally, `collectionSR_wIndex` will include the final harmonized collection with all the sensors based on the temporal parameters defined previously, with all spectral bands and vegetation/spectral indices:

```
var collectionSR_wIndex = getCombinedSRcollection(startYear, startDay,
    endyear, endDay).map(addIndices);
```

Filter this collection by the bounds of the area of study:

```
var collectionL5L7L8 = collectionSR_wIndex.filterBounds(areaOfstudy);
```

Question 2. Based on your knowledge and the functions described above, what are the main fundamental differences between Landsat TM, ETM+, and OLI?

Question 3. What are the issues that may arise when using the same function for calculating vegetation indices using surface reflectance data acquired by ETM+ and OLI sensors?

Vegetation Index Anomaly

By definition, anomaly is anything that deviates from what is standard, normal, or expected. Usually, anomalies are calculated by subtracting a long-term average of a variable from the actual value of that variable at a given time. For example, if X = actual value of average NDVI for mangroves in 2020, and Y = long-term average NDVI of mangroves (an average over many years), then the anomaly = X − Y. If the anomaly values are zero (or very close to zero), it means that NDVI remained relatively stable in that period, which indicates that there has not been any significant disturbance in that area. On the other hand, a positive anomaly means that the NDVI signal is greater than its long-term average, which indicates that vegetation has shown growth in that area; similarly, a negative anomaly means that the NDVI signal is weaker than its long-term average, indicating a potential loss in the area.

To calculate the anomaly, start by defining the index you want to compute the anomaly for and the reference period to get the average value. In this example, we are going use the 16 years before the mangrove extent baseline in 2000:

```
var index = 'NDVI';
var ref_start = '1984-01-01'; // Start of the period
var ref_end = '1999-12-31'; // End of the period
```

Next, create the reference collection using `collectionL5L7L8` and the parameters above. You can print the size of this reference collection to the **Console** using `print` and `size`:

```
var reference = collectionL5L7L8
    .filterDate(ref_start, ref_end)
    .select(index)
    .sort('system:time_start', true);
```

```
print('Number of images in Reference Collection', reference.size());
```

You can now calculate the mean value (and other statistics) for the reference collection `reference`. Mask the results by the baseline mangrove extent using `extentBuffer`:

```
var mean = reference.mean().mask(extentBuffer);
var median = reference.median().mask(extentBuffer);
var max = reference.max().mask(extentBuffer);
var min = reference.min().mask(extentBuffer);
```

Now that we have our long-term reference metrics, you can define the period for which you want to compute the gains and losses. In this example, we will use the full period of 2000–2020. However, any combination of years is possible depending on what period you are interested in. Then, an anomaly function can be created to subtract the metric from the average of your period of interest:

```
var period_start = '2000-01-01'; // Full period
var period_end = '2020-12-31';

var anomalyfunction = function(image) {
    return image.subtract(mean)
        .set('system:time_start', image.get('system:time_start'));
};
```

Finally, map the `anomalyfunction` to the Landsat collection filtered by your `period_start` and `period_end`:

```
var series = collectionL5L7L8.filterDate(period_start, period_end)
    .map(anomalyfunction);
```

The object `series` will have all the spectral bands and vegetation/spectral indices for the time period defined above. Their values, however, will be different from the original collection since we subtracted the average value of the reference period. The next step is to sum all the values for the index from `series` and divide by the number of images available:

```
var seriesSum = series.select(index).sum().mask(extentBuffer);
var numImages = series.select(index).count().mask(extentBuffer);
var anomaly = seriesSum.divide(numImages);
```

Add the anomaly layer to the map using a color ramp of your choice (Fig. A3.3.2):

```
var visAnon = {
    min: -0.20,
    max: 0.20,
    palette: ['#481567FF', '#482677FF', '#453781FF', '#404788FF',
        '#39568CFF', '#33638DFF', '#2D708EFF', '#287D8EFF',
        '#238A8DFF',
        '#1F968BFF', '#20A387FF', '#29AF7FFF', '#3CBB75FF',
        '#55C667FF',
        '#73D055FF', '#95D840FF', '#B8DE29FF', '#DCE319FF',
        '#FDE725FF'
    ]
};
Map.addLayer(anomaly, visAnon, index + ' anomaly');
```

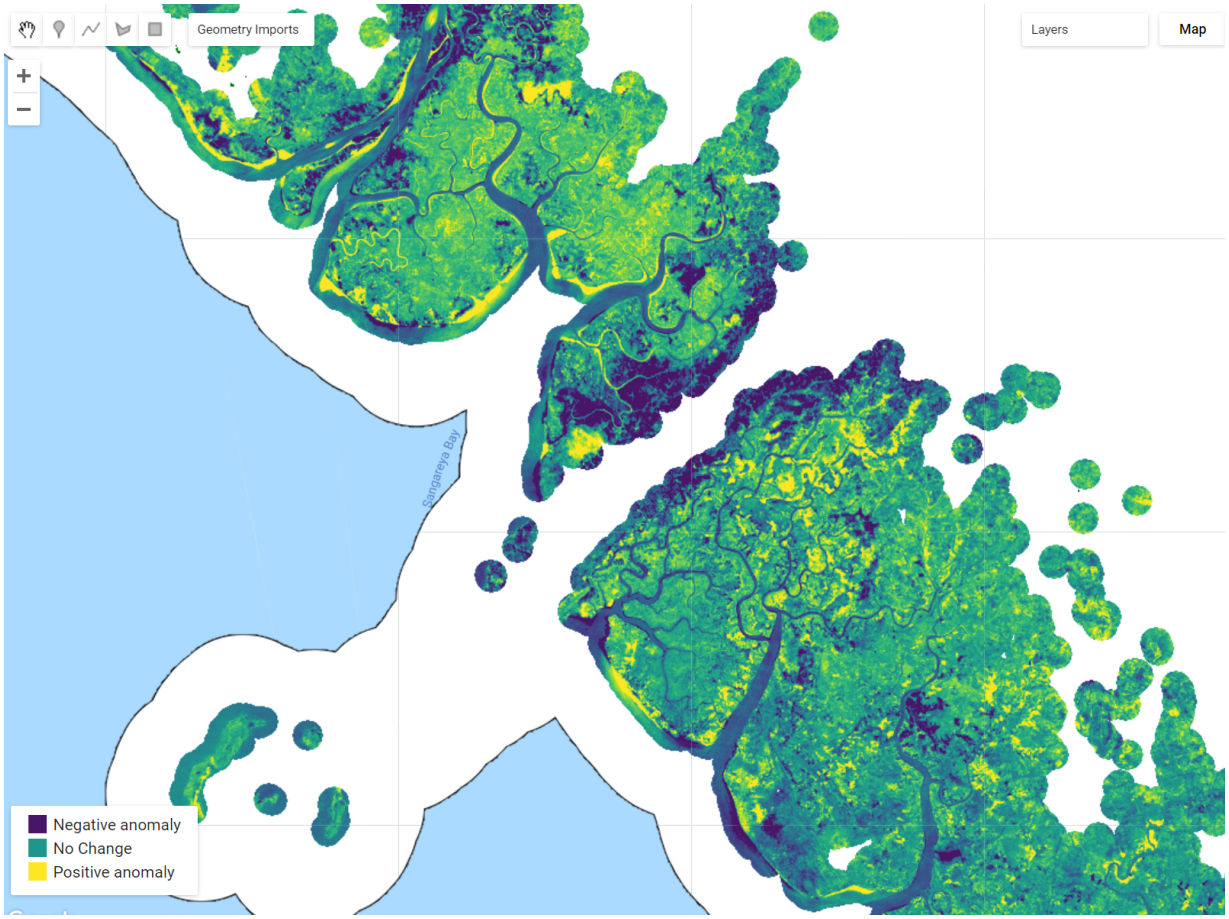


Fig. A3.3.2 NDVI anomaly for the period of 2000–2000

You can then extract loss areas by selecting a value threshold on the anomaly (Fig. A3.3.3):

```
var thresholdLoss = -0.05;
var lossfromndvi = anomaly.lte(thresholdLoss)
    .selfMask()
    .updateMask(
        mangrove2000
    ); // Only show the losses within the mangrove extent of year 2000

Map.addLayer(lossfromndvi, {
    palette: ['orange']
```

```
}, 'Loss from Anomaly 00-20');

var thresholdGain = 0.20;
var gainfromndvi = anomaly.gte(thresholdGain)
    .selfMask()
    .updateMask(
        extentBuffer
    ); // Only show the gains within the mangrove extent buffer of year
2000

Map.addLayer(gainfromndvi, {
    palette: ['blue']
}, 'Gain from Anomaly 00-20');
```

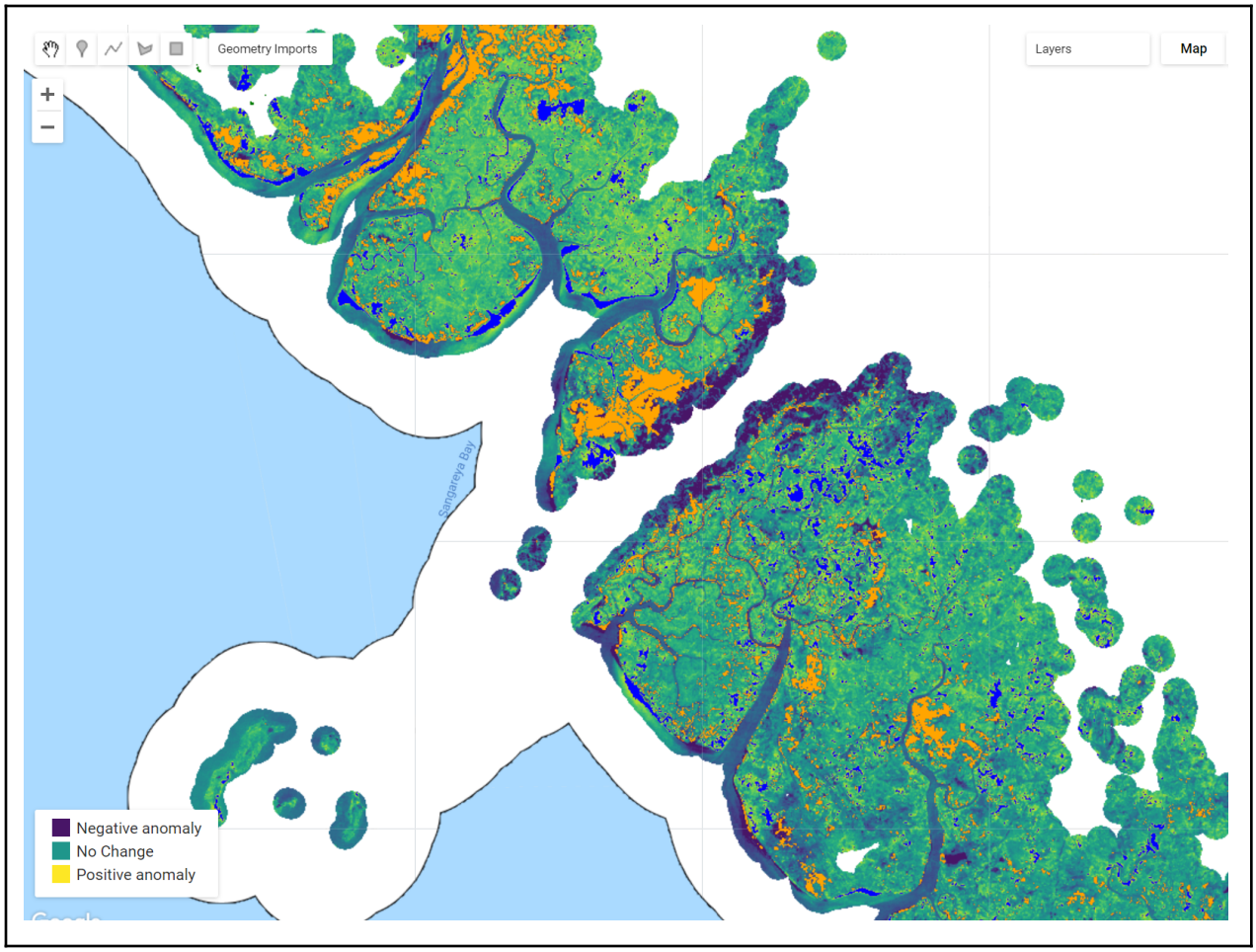


Fig. A3.3.3 NDVI anomaly losses (orange) and gains (blue) based on change threshold values

Code Checkpoint A33b. The book's repository contains a script that shows what your code should look like at this point.

Question 4. What are the main challenges of a map-to-image change detection approach? Think from a perspective of baseline extent, spectral index, and metric used (e.g., mean versus median).

Synthesis

With the content covered in this chapter, you will be able to reproduce a map-to-map and a map-to-image change detection to your own area of interest. Additionally, the anomaly analysis can be used to detect changes in other land cover types, including (but not limited to) forests (forest cover loss) and agricultural land (crop harvest).

Assignment 1. Practice your land cover classification skills by using the code in script **A33s1 - Supplemental** in the book's repository. In this code, we show how to create a mangrove extent map for Guinea using manually and automatically selected samples. The code covers masking techniques and using data from Google Earth Engine Catalog to assist in the classification workflow. Using the techniques you have learned, create two 30 m Landsat-based mangrove extent maps for the year of 2000 and 2020 for Guinea, West Africa.

Assignment 2. Using what you learned in this chapter, compile the areas (in km²) of mangrove change in Guinea derived from the anomaly analysis using: (a) vegetation spectral index versus a water-based spectral index; (b) mean versus median; and (c) five-year intervals (2000–2005, 2005–2010, 2010–2015, and 2015–2020.

Assignment 3. Practice the classification and compare the map-to-map and map-to-image change approaches for another land cover type/area of your choice, such as the following:

- Agricultural land expansion in the Nile Delta, Egypt.
- Forest burn/loss near Mount Hood, Oregon, United States.

Conclusion

Google Earth Engine's computing infrastructure has been revolutionizing time-consuming remote sensing processes, creating a new way for rapid land cover classification and change detection at large scales. In the case of mangrove forests—a highly dynamic ecosystem that has been under increasing anthropogenic pressure—these approaches allow for a rapid and consistent monitoring of change in extent over time. These approaches using Earth Engine may be particularly useful for developing countries where, until very recently, the high computational power and the difficulties of distributing nontrivial classification algorithms across multiple computational workstations has challenged classification and change detection at large scales.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Bunting P, Rosenqvist A, Lucas RM, et al (2018) The global mangrove watch - A new 2010 global baseline of mangrove extent. Remote Sens 10:1669. https://doi.org/10.3390/rs10101669

Goldberg L, Lagomasino D, Thomas N, Fatoyinbo T (2020) Global declines in human-driven mangrove loss. Glob Chang Biol 26:5844–5855. https://doi.org/10.1111/gcb.15275

Hansen MC, Potapov PV, Moore R, et al (2013) High-resolution global maps of 21st-century forest cover change. Science 342:850–853. https://doi.org/science.1244693

Roy DP, Kovalskyy V, Zhang HK, et al (2016) Characterization of Landsat-7 to Landsat-8 reflective wavelength and normalized difference vegetation index continuity. Remote Sens Environ 185:57–70. https://doi.org/10.1016/j.rse.2015.12.024

Chapter A3.4: Forest Degradation and Deforestation

Authors
Carlos Souza Jr., Karis Tenneson, John Dilger, Crystal Wespestad, Eric Bullock

Overview

Tropical forests are being disturbed by deforestation and forest degradation at an unprecedented pace (Hansen et al. 2013, Bullock et al. 2020). Deforestation completely removes the original forest cover and replaces it with another land cover type, such as pasture or agriculture fields. Generally speaking, forest degradation is a temporary or permanent disturbance, often caused by predatory logging, fires, or forest fragmentation, where the tree loss does not entirely change the land cover type. Forest degradation leads to a more complex environment with a mixture of vegetation, soil, tree trunks and branches, and fire ash. Defining a boundary between deforestation and forest degradation is not straightforward; at the time this chapter was written, there was no universally accepted definition for forest degradation (Aryal et al. 2021). Furthermore, the signal of forest degradation often disappears within one to two years, making degraded forests spectrally similar to undisturbed forests. Due to these factors, detecting and mapping forest degradation with remotely sensed optical data is more challenging than mapping deforestation.

The purpose of this chapter is to present a spectral unmixing algorithm and the Normalized Difference Fraction Index (NDFI) to detect and map both forest degradation and deforestation in tropical forests. This spectral unmixing model uses a set of generic endmembers (Souza et al. 2005a) to process any Landsat Surface Reflectance (Tier 1) scene available in Google Earth Engine. We present two examples of change detection applications, one comparing a pair of images acquired at different times a year apart by making a temporal color composite and an empirically defined change threshold, and another using a more extensive and dense time series approach.

Learning Outcomes

- Calculating NDFI, the Normalized Difference Fraction Index
- Interpreting fraction images and NDFI using a temporal color composite.

- Analyzing deforestation and forest degradation with NDFI
- Running a time-series change detection to detect forest change.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Use drawing tools to create points, lines, and polygons (Chap. F2.1).
- Run and interpret spectral unmixing models (Chap. F3.1).
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Aggregate data to build a time series (Chap. F4.2).
- Perform a two-period change detection (Chap. F4.4).

Introduction to Theory

Landsat imagery has been extensively used to monitor deforestation (Woodcock et al. 2020). However, detecting and mapping forest degradation associated with selective logging is more elaborate and challenging. First efforts involved the application of spectral and textural indices to enhance the detection of canopy damage created by logging (Asner et al. 2002, Souza et al. 2005a), but these turned out to be more helpful in enhancing logging infrastructure using Landsat shortwave infrared bands (i.e., roads and log landings; Matricardi et al. 2007).

Alternatively, spectral mixture analysis (SMA) has been proposed to overcome the challenge of using whole-pixel information to detect and classify forest degradation. Landsat pixels typically contain a mixture of land cover components (Adams et al. 1995). The SMA method is based on the linear spectral unmixing model, as described in Chap. F3.1. The identification of the nature and number of pure spectra (often referred to in this context as "endmembers") in the image scene is an important step in obtaining correct SMA models. In logged forests (and also in burned forest and forest edges), mixed pixels predominate and are expected to have a combination of green vegetation (GV), soil, non-photosynthetic vegetation (NPV), and shade-covered materials. Therefore, fraction images derived from SMA analyses are more suitable to enhance the detectability of logging infrastructure and canopy damage within degraded forests. For example, soil fractions reveal log landings and logging roads (Souza and Barreto 2000), while the NPV

highlights forest canopy damage (Cochrane and Souza 1998, Souza et al. 2003), and the areas decreasing in GV indicate forest canopy gaps (Asner et al. 2004).

A study has shown that it is possible to generalize the SMA model to Landsat sensors (including TM, ETM+, and OLI) (Small 2004). Souza et al. (2005a) expanded the generalized SMA to handle five endmembers—GV, NPV, Soil, Shade, and Cloud —expected within forest degradation areas and proposed a novel compositional index based on SMA fractions, the Normalized Difference Fraction Index (NDFI).

The NDFI is computed as:

$$NDFI = \frac{GVshade - (NPV + Soil)}{GVshade + NPV + Soil} \quad \text{(A3.4.1)}$$

where $GVshade$ is the shade-normalized GV fraction given by,

$$GVshade = \frac{GV}{100 - Shade} \quad \text{(A3.4.2)}$$

NDFI values range from −1 to 1. For intact forests, NDFI shows high values (i.e., about 1) due to the combination of high GVshade (i.e., high GV and canopy Shade) and low NPV and Soil values. The NPV and Soil fractions increase as forests are more degraded, lowering NDFI values relative to the intact forests. Deforested areas exhibit very low GV and Shade and high NPV and Soil, making it possible to distinguish them from degraded forests based on NDFI magnitude.

Recent studies compared NDFI with other spectral indices. NDFI generated more accurate results in deforestation detection (Schultz et al. 2016) and forest degradation (Bullock et al. 2018) in time-series analysis. One of the key components for its success is lowering unwanted noise and accounting for illumination variability through the shade normalization applied to the GV fraction.

Practicum

Section 1. Spectral Mixture Analysis Model

Let us first define the Landsat endmembers based on Souza et al. (2005a). These endmembers were developed and tested in the Amazon. These endmembers work well in many other environments (see example applications for calculating NDFI in non-Amazonian tropical forest in Schultz et al. 2016 [Ethiopia and Viet Nam], Kusbach et al. 2017 [central Europe], and Hirschmugl et al. 2013 [Cameroon and Central African Republic]). If you are working in a different region, assess how well they perform for your forest types. The ratio of these endmembers that makes up the spectral signature of each pixel gives a good indication of the plant health and composition for that area. When the ratio shifts over time towards one or more of the endmembers we can quantify how the landscape is changing.

Below, we will create a new variable `endmembers` by copying the values from the code block below. The six numbers in square brackets define the pure reflectance values for the blue, green, red, SWIR1, and SWIR2 bands for each endmember material.

```
// SMA Model - Section 1

// Define the Landsat endmembers (source: Souza et al. 2005)
// They can be applied to Landsat 5, 7, 8, and potentially 9.
var endmembers = [
  [0.0119,0.0475,0.0169,0.625,0.2399,0.0675], // GV
  [0.1514,0.1597,0.1421,0.3053,0.7707,0.1975], // NPV
  [0.1799,0.2479,0.3158,0.5437,0.7707,0.6646], // Soil
  [0.4031,0.8714,0.79,0.8989,0.7002,0.6607] // Cloud
];
```

We will choose a single Landsat 5 image to work with for now, and select the bands we will need for the SMA and NDFI calculation.

Create a new variable `image` and assign it to the Landsat 5 image from the code block below. Select the visible, near infrared, and shortwave infrared bands. Then, center the

map around the Landsat 5 image with a zoom scale of 10.

```
// Select a Landsat 5 scene on which to apply the SMA model.
var image = ee.Image('LANDSAT/LT05/C02/T1_L2/LT05_226068_19840411')
    .multiply(0.0000275).add(-0.2);

// Center the map on the image object.
Map.centerObject(image, 10);

// Define and select the Landsat bands to apply the SMA model.
// use ['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B7'] for
Landsat 5 and 7.
// use ['SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7'] for
Landsat 8.
var bands = ['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B7'];
image = image.select(bands);
```

Next, we will need to create a couple of functions to use the endmembers and create the NDFI image. We will need to unmix the input Landsat image.

First, we will create a new function `getSMAFractions` that takes two parameters: `image` and `endmembers`. To unmix the image, we first select the visible, near-infrared, and short wave infrared bands, and call the `unmix` function with the endmembers used as the argument. Note: The order in which the bands are selected matters. For each endmember (GV, NPV, Soil, and Cloud) each value in the array represents the pure value for each band being passed in. For example, if the selected bands were ['SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7'] then the selected endmembers at position 0 would be:

- 'SR_B2' GV: 119
- 'SR_B2' NPV: 1514
- 'SR_B2' Soil: 1799
- 'SR_B2' Cloud: 4031

```
// Unmixing image using Singular Value Decomposition.
var getSMAFractions = function(image, endmembers) {
    var unmixed = ee.Image(image)
        .select([0, 1, 2, 3, 4,
```

```
            5
        ]) // Use the visible, NIR, and SWIR bands only!
        .unmix(endmembers)
        .max(0) // Remove negative fractions, mostly Soil.
        .rename('GV', 'NPV', 'Soil', 'Cloud');
    return ee.Image(unmixed.copyProperties(image));
};
```

We will now write the function to calculate NDFI. We could write it out line by line for each time we want to perform SMA or calculate NDFI. But having the equation implemented as a function gives us many benefits, including reducing redundancy, allowing us to map the function over a collection of images, and reducing errors from typos or other little bugs that can find their way into our code.

We will use the fraction images obtained with the `getSMAFractions` function above to calculate Shade, GVs, and NDFI using image expressions. This procedure will return a multiband image with the Shade, GVs, and NDFI bands added to the input image.

First, we will create a new variable `sma` and pass in the image and endmembers as arguments. Then calculate the Shade and GV shade-normalized (GVs) fractions from the SMA bands, and add the Shade and GVs bands to the SMA image. We calculate NDFI using an expression implementing Eq. A3.4.1, and add the new band to the SMA image.

```
// Calculate GVS and NDFI and add them to image fractions.
// Run the SMA model passing the Landsat image and the endmembers.
var sma = getSMAFractions(image, endmembers);

Map.addLayer(sma, {
    bands: ['NPV', 'GV', 'Soil'],
    min: 0,
    max: 0.45
}, 'sma');

// Calculate the Shade and GV shade-normalized (GVs) fractions from the
SMA bands.
var Shade = sma.reduce(ee.Reducer.sum())
    .subtract(1.0)
```

```
    .abs()
    .rename('Shade');

var GVs = sma.select('GV')
    .divide(Shade.subtract(1.0).abs())
    .rename('GVs');

// Add the new bands to the SMA image variable.
sma = sma.addBands([Shade, GVs]);

// Calculate the NDFI using image expression.
var NDFI = sma.expression(
    '(GVs - (NPV + Soil))  / (GVs + NPV + Soil)', {
        'GVs': sma.select('GVs'),
        'NPV': sma.select('NPV'),
        'Soil': sma.select('Soil')
    }).rename('NDFI');

// Add the NDFI band to the SMA image.
sma = sma.addBands(NDFI);
```

We will use a color palette that spans from white, to pink (i.e., bare land), to yellow, to green to visualize the NDFI image. Higher values of NDFI will be green, while lower values will span the colors of white, pink, and yellow. Copy the code block below into your Code Editor.

```
// Define NDFI color table.
var palettes = require(
    'projects/gee-edu/book:Part A - Applications/A3 - Terrestrial
Applications/A3.4 Forest Degradation and
Deforestation/modules/palettes'
);

var ndfiColors = palettes.ndfiColors;
```

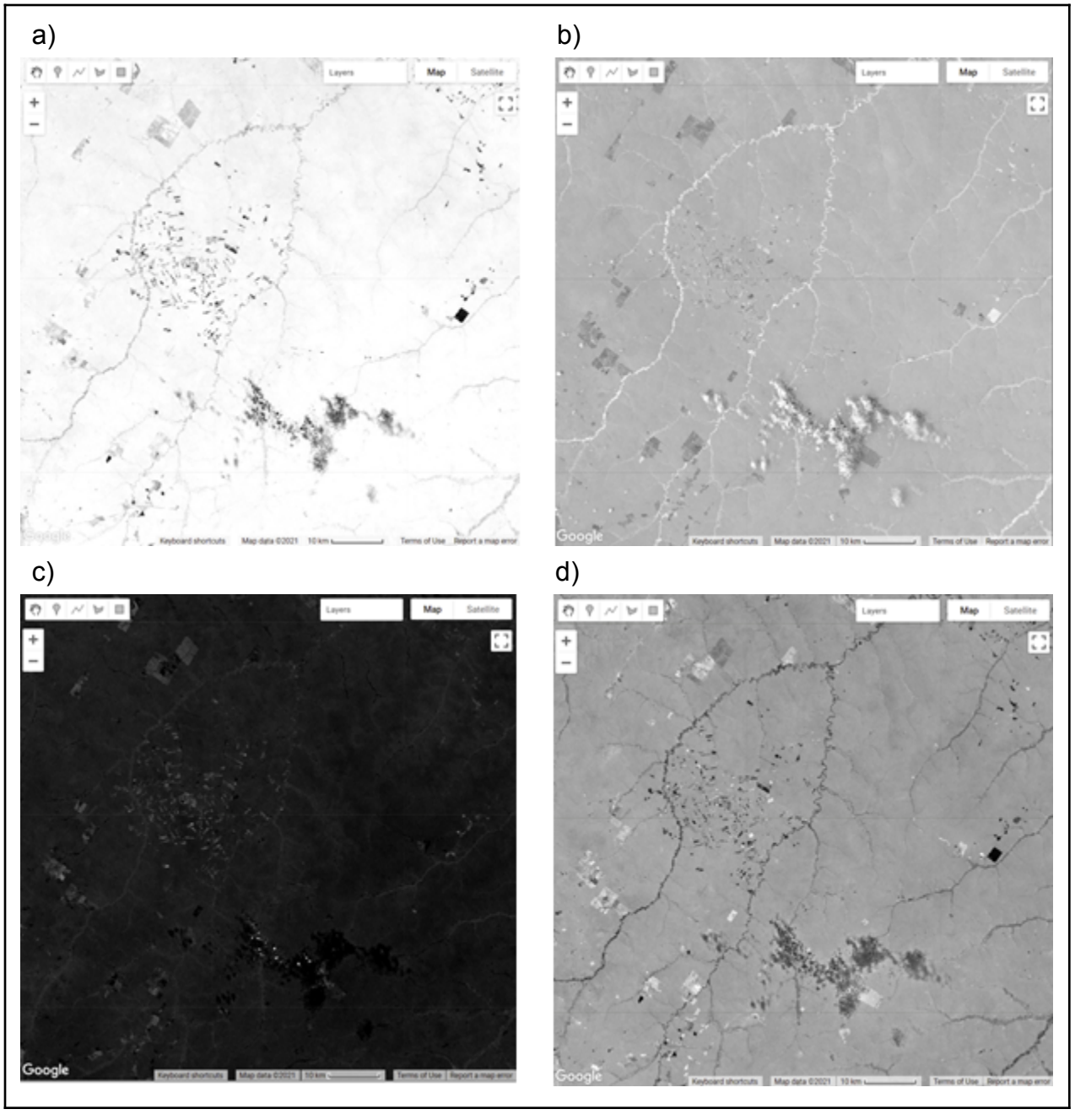
Next, we can visualize all the hard work we have done unmixing each endmember and the NDFI bands (Fig. A3.4.1).

Create an image visualization object with bands 5, 4, and 3, chosen for visualization along with a min and max. Add the Landsat 5 image to the map using the image visualization object.

Now add each of the SMA bands and the NDFI band to the map. Note: Rather than defining the min, max, and bands for each of these in separate variables, we can pass in the object directly to the `Map.addLayer` function.

```
var imageVis = {
    'bands': ['SR_B5', 'SR_B4', 'SR_B3'],
    'min': 0,
    'max': 0.4
};

// Add the Landsat color composite to the map.
Map.addLayer(image, imageVis, 'Landsat 5 RGB-543', true);

// Add the fraction images to the map.
Map.addLayer(sma.select('Soil'), {
    min: 0,
    max: 0.2
}, 'Soil');
Map.addLayer(sma.select('GV'), {
    min: 0,
    max: 0.6
}, 'GV');
Map.addLayer(sma.select('NPV'), {
    min: 0,
    max: 0.2
}, 'NPV');
Map.addLayer(sma.select('Shade'), {
    min: 0,
    max: 0.8
}, 'Shade');
Map.addLayer(sma.select('GVs'), {
    min: 0,
```

```
    max: 0.9
}, 'GVs');
Map.addLayer(sma.select('NDFI'), {
    palette: ndfiColors
}, 'NDFI');
```



e)

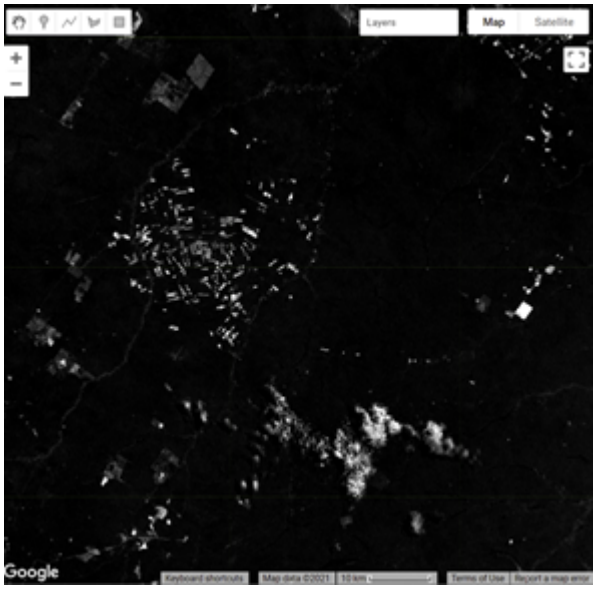


Fig. A3.4.1 Example maps of a) green vegetation shade normalized fraction (more vegetation is whiter); b) shade fraction (more shade is whiter); c) non-photosynthetic vegetation fraction (more NPV is whiter); d) green vegetation fraction (more vegetation is whiter); e) soil fraction (more soil is whiter).

The last thing we will do in this section is to create a water and cloud mask. Water and clouds will have lower NDFI values. While water may not be too much of an issue, clouds will impact how we monitor forest degradation and loss, and thus we can simply mask them out. We can mask them using a thresholding method based on the values of our fraction images.

First, create a new function variable `getWaterMask` that takes an SMA image as the only argument.

Next, create a water mask using threshold values for the Shade, GV, and Soil bands, where Shade is greater than or equal to 0.65; GV is less than or equal to 0.15; and Soil is less than or equal to 0.05.

Now create a cloud mask by applying a threshold of 0.1 or greater to the Cloud band.

```
var getWaterMask = function(sma) {
    var waterMask = (sma.select('Shade').gte(0.65))
```

```
        .and(sma.select('GV').lte(0.15))
        .and(sma.select('Soil').lte(0.05));
    return waterMask.rename('Water');
};

// You can use the variable below to get the cloud mask.
var cloud = sma.select('Cloud').gte(0.1);
var water = getWaterMask(sma);
```

Next, we will combine the cloud and water masks using the max reducer. Since we want to mask both water and clouds, using the max works quite nicely here—as opposed to adding the images together—so we do not need to worry about overlaps.

Now add the cloud and water mask as a layer to the map.

Apply the cloud and water mask to the NDFI band using the `updateMask` function and invert the mask using the `not` function. Note: `updateMask` considers zeroes as invalid (i.e., to be masked) and ones as valid (i.e., to be kept). Since our original mask had values of 1 for cloud and water, we use the `not` function to invert the values.

```
var cloudWaterMask = cloud.max(water);
Map.addLayer(cloudWaterMask.selfMask(),
    {
        min: 1,
        max: 1,
        palette: 'blue'
    },
    'Cloud and water mask');

// Mask NDFI.
var maskedNDFI = sma.select('NDFI').updateMask(cloudWaterMask.not());
Map.addLayer(maskedNDFI, {
    palette: ndfiColors
}, 'NDFI');
```

Code Checkpoint A34a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Deforestation and Forest Degradation Change Detection

To observe changes in the landscape over time, you need to create an NDFI image for two points in time and then calculate the difference between them. Previous studies have shown that images should not be more than one year apart because the forest degradation disturbance quickly disappears with tree foliage and understory vegetation growth. Changes in NDFI are a good indicator of forest change.

Now we will start using the functions we have built. First, we perform the SMA on a Landsat 5 image. Recall that the SMA function wants only the visible, near-infrared, and shortwave infrared bands. When using Landsat 5, those correspond to bands 1–5 and band 7.

First, create a variable for the Landsat 5 scene specified in the code block below and select the visible, near-infrared, and shortwave infrared bands. Use the `getSMAFractions` function with the Landsat image and the endmembers. Rename the output SMA bands as GV, NPV, Soil, and Cloud.

```
// Select two Landsat 5 scenes on which to apply the SMA model.

// Select Landsat bands used for forest change detection.
var imageTime0 = ee.Image(
        'LANDSAT/LT05/C02/T1_L2/LT05_226068_20000509')
    .multiply(0.0000275).add(-0.2);
var bands = ['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B7'];
imageTime0 = imageTime0.select(bands);

// Run the SMA model.
var smaTime0 = getSMAFractions(imageTime0, endmembers);
```

Before we move on to calculate the NDFI, we will add the previous Landsat scene and the fractional images to the map to inspect them (example results in Fig. A3.4.2).

```
// Center the image object.
Map.centerObject(imageTime0, 10);
```

```
// Define the visualization parameters.
var imageVis = {
    'opacity': 1,
    'bands': ['SR_B5', 'SR_B4', 'SR_B3'],
    'min': 0,
    'max': 0.4,
    'gamma': 1
};
// Scale to the expected maximum fraction values.
var fractionVis = {
    'opacity': 1,
    'min': 0.0,
    'max': 0.5
};

// Add the Landsat color composite to the map.
Map.addLayer(imageTime0, imageVis, 'Landsat 5 RGB 543', true);

// Add the fraction images to the map.
Map.addLayer(smaTime0.select('Soil'), fractionVis, 'Soil Fraction');
Map.addLayer(smaTime0.select('GV'), fractionVis, 'GV Fraction');
Map.addLayer(smaTime0.select('NPV'), fractionVis, 'NPV Fraction');
```

a)	b)

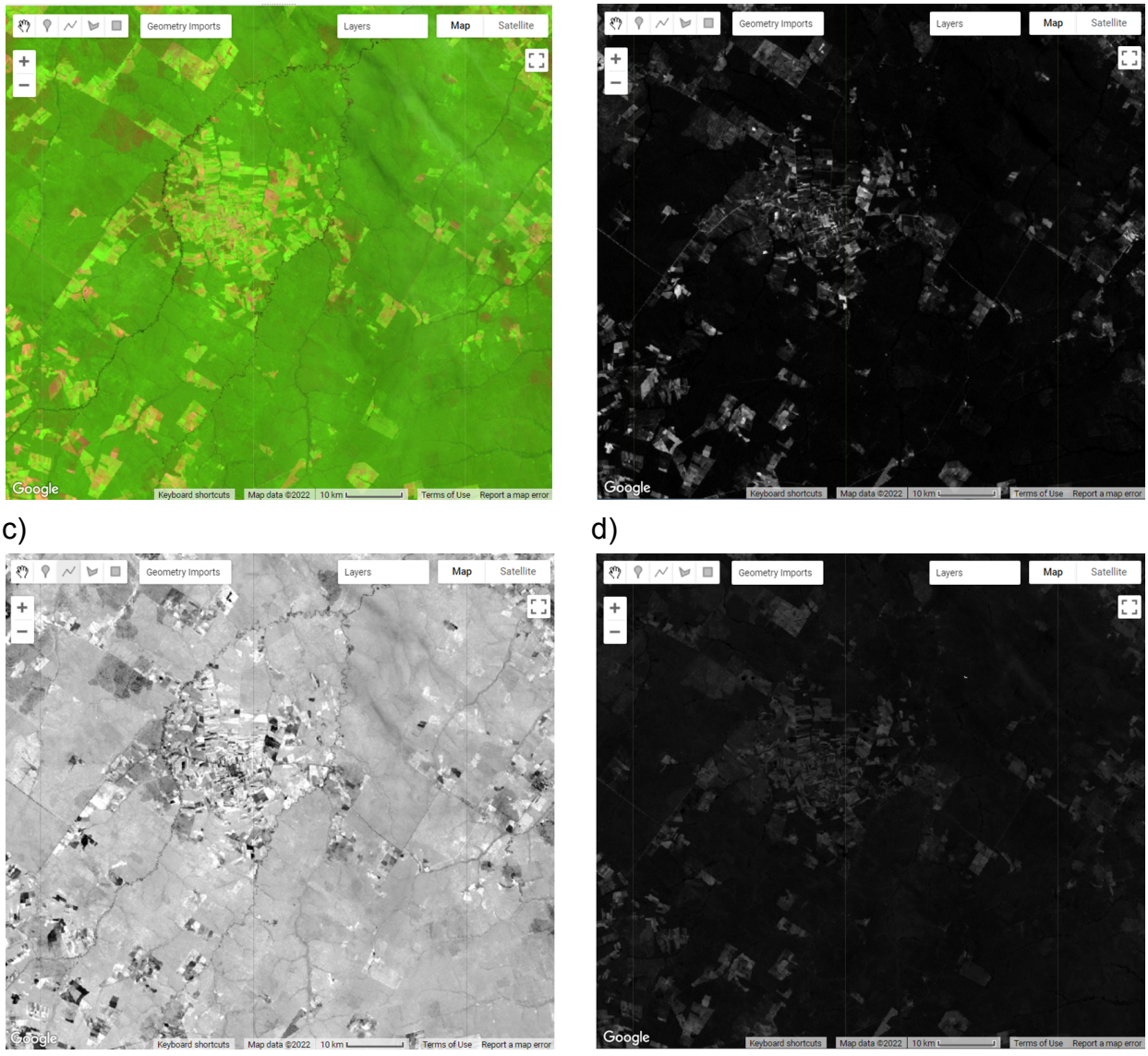


Fig. A3.4.2 Example maps of a) RGB composite of bands 5-4-3 (green is vegetation and brown is barren ground); b) soil fraction (more soil is whiter); c) green vegetation fraction (more vegetation is whiter), d) non-photosynthetic vegetation fraction (more NPV is whiter).

Next, let's set up a function to systematically reproduce the work we did computing NDFI, since we will need it a couple more times. Hereafter, we will be able to call that function instead of needing to explicitly write out each step, thus simplifying our code and making it easier to change if we need to later.

```
function getNDFI(smaImage) {
    // Calculate the Shade and GV shade-normalized (GVs) fractions
    // from the SMA bands.
    var Shade = smaImage.reduce(ee.Reducer.sum())
        .subtract(1.0)
        .abs()
        .rename('Shade');

    var GVs = smaImage.select('GV')
        .divide(Shade.subtract(1.0).abs())
        .rename('GVs');

    // Add the new bands to the SMA image variable.
    smaImage = smaImage.addBands([Shade, GVs]);

    var ndfi = smaImage.expression(
        '(GVs - (NPV + Soil))  / (GVs + NPV + Soil)', {
            'GVs': smaImage.select('GVs'),
            'NPV': smaImage.select('NPV'),
            'Soil': smaImage.select('Soil')
        }
    ).rename('NDFI');

    return ndfi;
}
```

Then, calculate NDFI for the earlier Landsat image's SMA bands (use `smaTime0` as an input), using the `getNDFI` function you wrote. Add this NDFI image to the map displayed in the `ndfiColors` palette (Fig. A3.4.3). This will serve as your calculated NDFI for your earlier point in time, the pre-change time (this time was defined as `imageTime0`).

```
// Create the initial NDFI image and add it to the map.
var ndfiTime0 = getNDFI(smaTime0);
Map.addLayer(ndfiTime0,
    {
```

```
        bands: ['NDFI'],
        min: -1,
        max: 1,
        palette: ndfiColors
    },
    'NDFI t0',
    false);
```

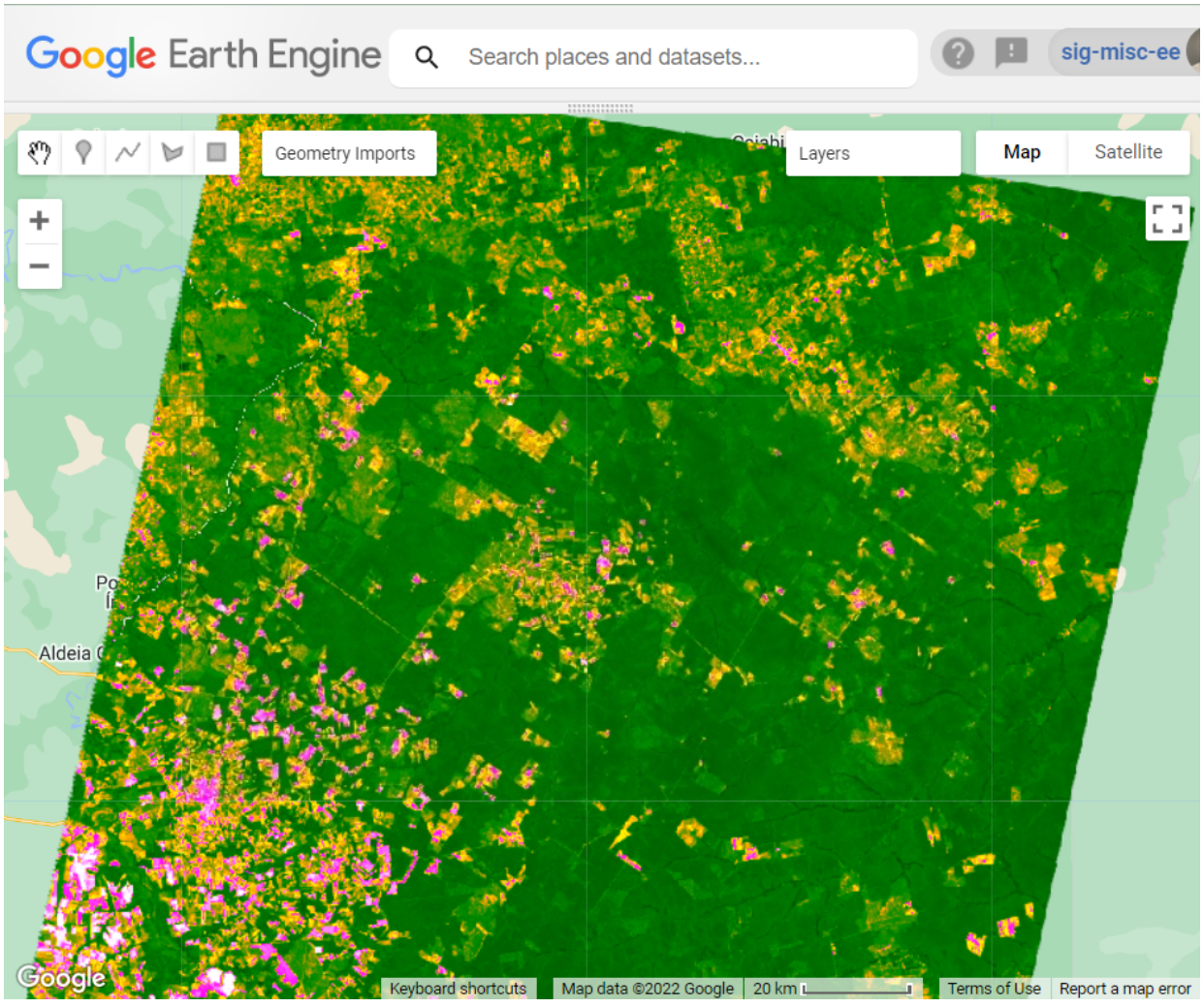


Fig. A3.4.3 NDFI image for the previous year obtained from the Landsat 5 (path/row 226/068) scene acquired on May 9, 2000. Orange colors indicate signs of forest disturbance associated with fires and selective logging. Pink and white colors are dry

vegetation and bare soil in old deforested areas. Orange colors in pasturelands mean dry vegetation.

Next, you will repeat this procedure to calculate the SMA fractions and NDFI of the second Landsat 5 image (`smaTime1`). You will utilize the same SMA method.

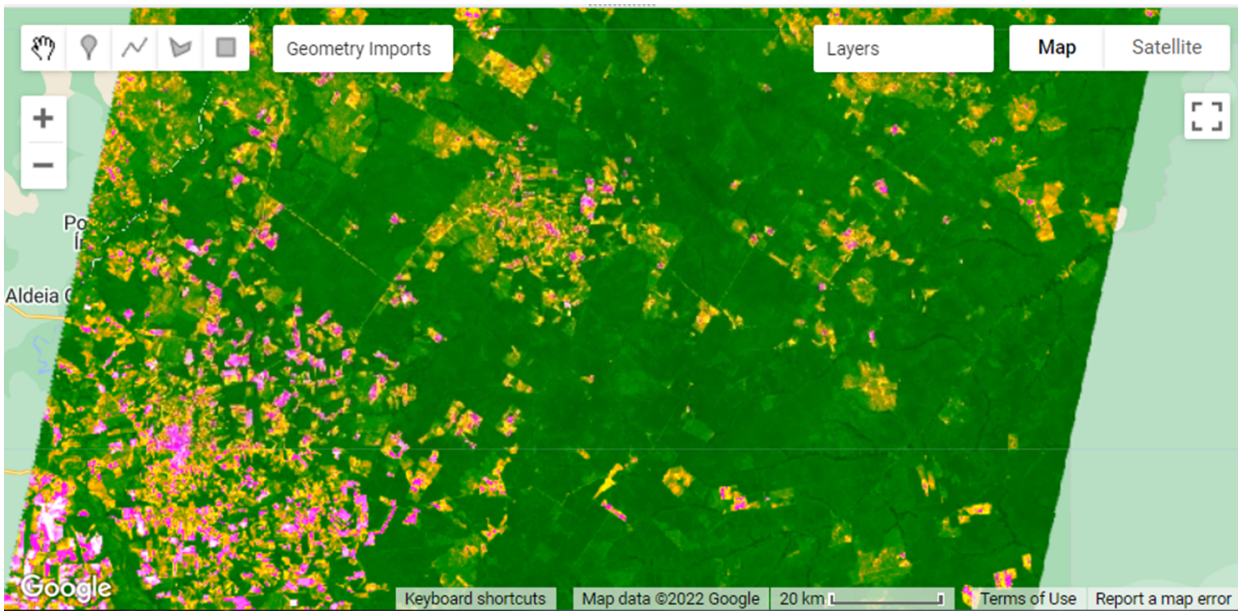
First, create a new variable (`imageTime1`), using the Landsat 5 scene from the code block below. Note that the band numbers for Landsat 8 are different from those for Landsat 5. In general, you should always make sure to check band names when working with multiple Landsat collections. Then, you will calculate the SMA fractions. The `getSMAFractions` function will rename the outputs to "GV", "NPV", "Soil", and "Cloud". Then, you will calculate NDFI for the new scene. Add an RGB composite and the NDFI to the map.

```
// Select a second Landsat 5 scene on which to apply the SMA model.
var imageTime1 = ee.Image(
        'LANDSAT/LT05/C02/T1_L2/LT05_226068_20010629')
    .multiply(0.0000275).add(-0.2)
    .select(['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B7']);
var smaTime1 = getSMAFractions(imageTime1, endmembers);

// Create the second NDFI image and add it to the map.
var ndfiTime1 = getNDFI(smaTime1);

Map.addLayer(imageTime1, imageVis, 'Landsat 5 t1 RGB-5', true);
Map.addLayer(ndfiTime1,
    {
        bands: ['NDFI'],
        min: -1,
        max: 1,
        palette: ndfiColors
    },
    'NDFI_t1',
    false);
```

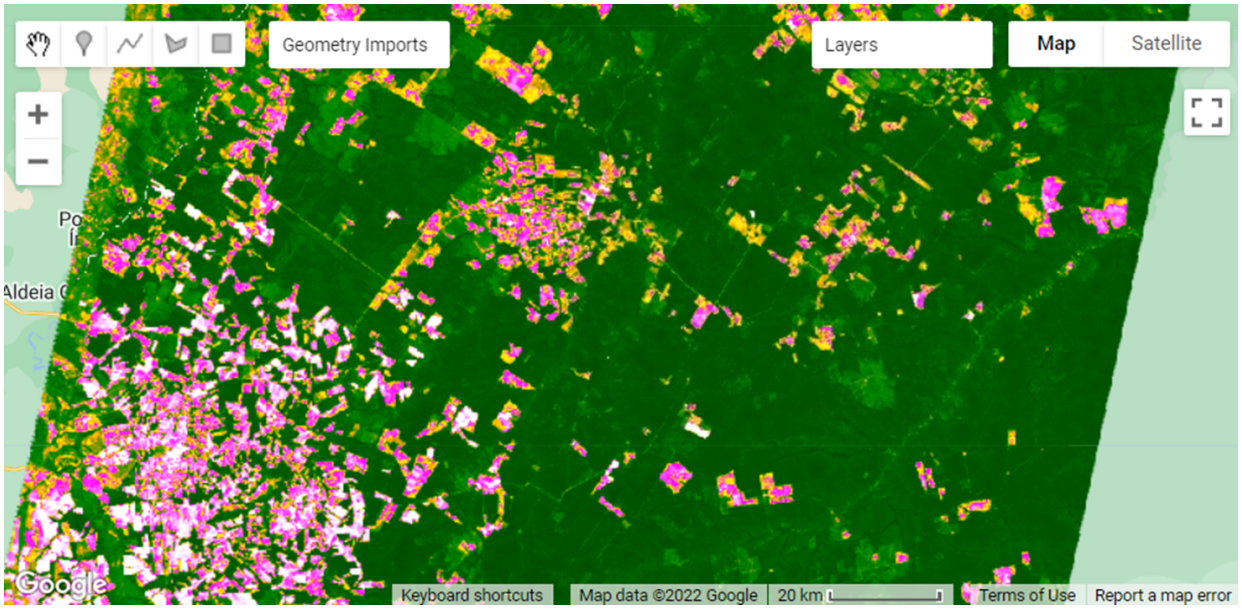
Before looking at changes between the two NDFI images, use the opacity slider in the **Layers** panel of the map to manually view the differences between the NDFI outputs, and then the RGB outputs. Where do you expect to see the greatest changes between the two?

An easier way to enhance changes using a pair of images is to build a temporal color composite. To construct a temporal color composite, we add the first image date to the R color channel and the second to the G and B channels. An example of RGB color composite (as first described in Chap. F1.1) is shown in Fig. A3.4.4, using the `NDFI_t0` (R) and `NDFI_t1` (G and B). Deforestation appears in bright red colors since we assigned the first NDFI image to the R color channel, indicating forest in the previous year, and removal in the following (i.e., G and B colors have low NDFI values due to forest removal). In contrast, the cyan colors in the NDFI temporal color composite indicate vegetation regrowth in the second year. The gradient of dark to gray colors suggests no change in NDFI between the two dates.

a)



b)



c)

d)

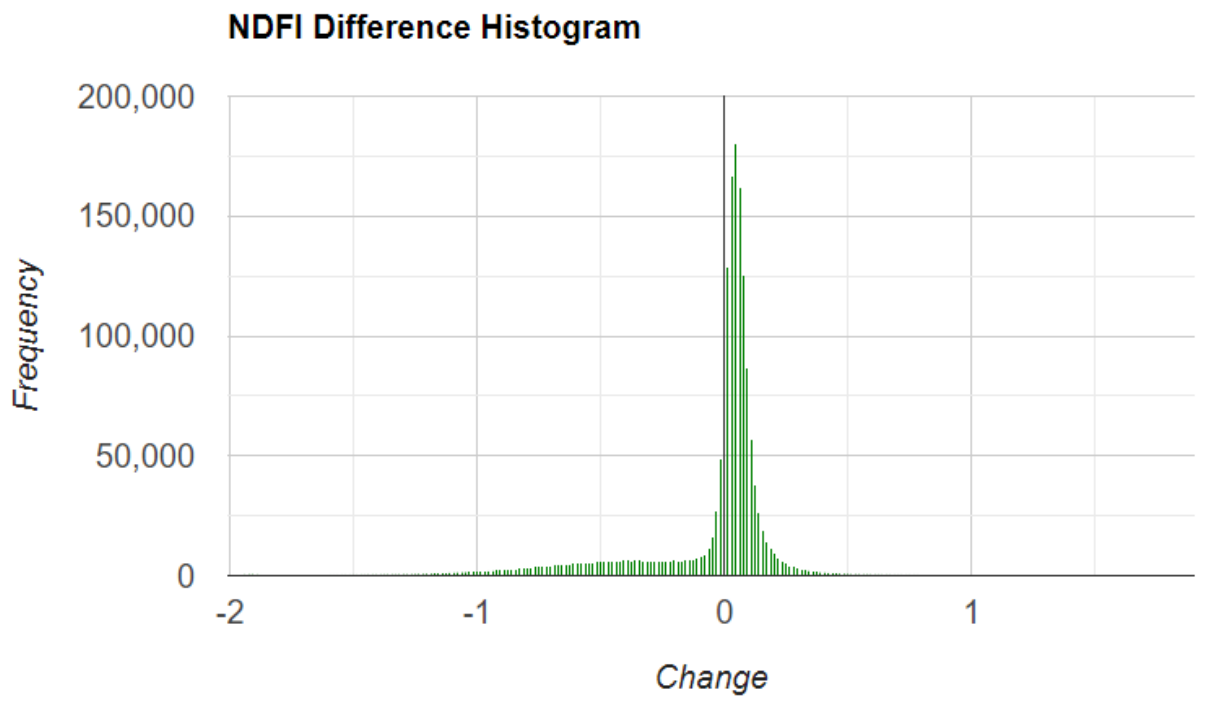


Fig. A3.4.4 Example of NDFI obtained for the first (a) and second (b) Landsat images, and a visualization of them using a temporal RGB color composite (c). Intense red colors are newly deforested areas, and light red are selectively logged forests. Cyan colors indicate vegetation regrowth in areas that were logged or burned a year ago or more. Example of NDFI difference histogram (d), with positive values indicating an increase in NDFI over time and negative values a decrease. Values at zero suggest no change.

To find the change between our images, a simple difference method can be applied by subtracting the previous image from the current image. Then, we can apply an empirically defined threshold to classify the changes based on the inspection of the histogram and the NDFI temporal color composite. For more information on two-date differencing, see Chap. F4.4.

Next, we will combine the two NDFI images for time `t0` and `t1`. Create a new variable `ndfiChange` and subtract the current NDFI image (made using Landsat 5 imagery from

2001) from the previous NDFI image (made using Landsat 5 imagery from 2000). Note: The NDFI is calculated using the image expression method applied to the SMA fraction presented in the code above. We then combine the two NDFI bands from 2000 and 2001 in one variable to display the change over time.

```
// Combine the two NDFI images in a single variable.
var ndfi = ndfiTime0.select('NDFI')
    .addBands(ndfiTime1.select('NDFI'))
    .rename('NDFI_t0', 'NDFI_t1');

// Calculate the NDFI change.
var ndfiChange = ndfi.select('NDFI_t1')
    .subtract(ndfi.select('NDFI_t0'))
    .rename('NDFI Change');
```

Using the polygon drawing tool (as described in Chap. F2.1), draw a region that covers most of the `ndfiChange` image and rename it `region` in the variable import panel.

Next, make a histogram named `histNDFIChange` that bins the `ndfiChange` image using the region you just drew, and prints it to the **Console**. Optionally, click **Run** to view the histogram and identify some potential change thresholds. In your case the histogram may look slightly different.

```
var options = {
    title: 'NDFI Difference Histogram',
    fontSize: 20,
    hAxis: {
        title: 'Change'
    },
    vAxis: {
        title: 'Frequency'
    },
    series: {
        0: {
            color: 'green'
        }
    }
```

```
};

// Inspect the histogram of the NDFI change image to define threshold
// values for classification. Make the histogram, set the options.
var histNDFIChange = ui.Chart.image.histogram(
        ndfiChange.select('NDFI Change'), region, 30)
    .setSeriesNames(['NDFI Change'])
    .setOptions(options);

print(histNDFIChange);
```

Classify the difference image into new deforestation (red), forest degradation (orange), regrowth (cyan), and forest (green). The classification is based on slicing the NDFI difference image. Old deforested areas are detected using the NDFI first image date (`t0`).

Add the change classification and difference images to the map and click **Run**.

```
// Classify the NDFI difference image based on thresholds
// obtained from its histogram.
var changeClassification = ndfiChange.expression(
        '(b(0) >= -0.095 && b(0) <= 0.095) ? 1 :' +
        //  No forest change
        '(b(0) >= -0.250 && b(0) <= -0.095) ? 2 :' + // Logging
        '(b(0) <= -0.250) ? 3 :' + // Deforestation
        '(b(0) >= 0.095) ? 4  : 0') // Vegetation regrowth
    .updateMask(ndfi.select('NDFI_t0').gt(
        0.60)); // mask out no forest

// Use a simple threshold to get forest in the first image date.
var forest = ndfi.select('NDFI_t0').gt(0.60);
```

Finally, add code to add all the new layers to the map and click **Run**.

```
// Add layers to map
Map.addLayer(ndfi, {
    'bands': ['NDFI_t0', 'NDFI_t1', 'NDFI_t1']
```

```
}, 'NDFI Change');
Map.addLayer(ndfiChange, {}, 'NDFI Difference');
Map.addLayer(forest, {}, 'Forest t0 ');
Map.addLayer(changeClassification, {
        palette: ['000000', '1eaf0c', 'ffc239', 'ff422f',
            '74fff9']
    },
    'Change Classification');
```

Fig. A3.4.5 shows an example of forest changes between two dates displayed using the cutoffs defined by the histogram. Categorizing the whole map into a simple system of no change, old deforestation, new deforestation, partial forest disturbance, and regrowth makes the forest use patterns in the region clearer. However, you must be careful with the thresholds you chose from the histogram when creating and interpreting the NDFI changes in this way, as those values can drastically alter your map. Consider when it might be more appropriate to use an RGB temporal color composite, and in what situations the classified map using the empirically defined thresholds would be better.

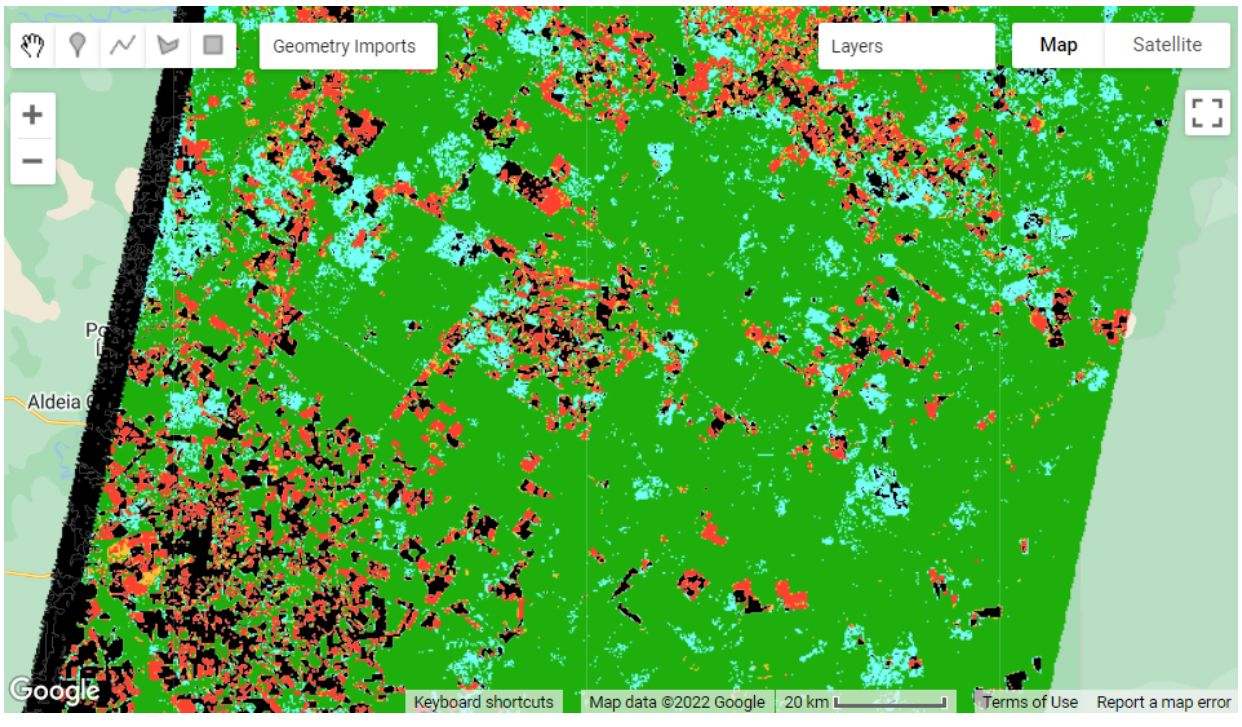


Fig. A3.4.5 Example of the change detection using two NDFI images. Green is remaining forest; black is old deforestation as detected in the first NDFI image; red

highlights new deforestation; orange shows forest disturbance by logging or fires; and cyan is vegetation regrowth of forest since the previous date.

Code Checkpoint A34b. The book's repository contains a script that shows what your code should look like at this point.

Save your script for your own future use, as outlined in Chap. F1.0. Then, refresh the page to begin with a new script for the next section.

Section 3: Deforestation and Forest Degradation Time Series Analysis

To assess deforestation and forest degradation with a time series, we can use a Google Earth Engine tool called CODED (Bullock et al. 2018, Bullock et al. 2020). The algorithm is based on previous work in continuous land cover monitoring (Zhu and Woodcock 2014) and NDFI-based degradation mapping using spectral unmixing models (Souza et al. 2003 and 2005a). CODED has both a user-interface application and an API, which can be accessed in the Earth Engine Code Editor. For this lesson we will use the API.

Code Checkpoint A34c. The book's repository contains a script to use to begin this section. You will need to start with that script and script and paste code below into it. The checkpoint accesses the modules of the CODED API and support functions. Note: importing large modules can cause your browser to hang for a moment while they load.

Detecting degraded forest regions requires knowledge of the characteristics of the forest of interest when it is in its normal healthy state. Higher NDFI values, near 1, typically indicate a healthy forest, but the magnitude range of NDFI for a healthy forest is dependent on forest density, type of forest ecosystem, and the seasonality of the area. Therefore, CODED uses a training period to find the typical values of NDFI generally seen in each forest over time. For each pixel, a regression model, similar to what was first presented in Chap. F4.6, is fit for NDFI values within the training period. The regression model is composed of a constant for overall NDFI magnitude, a sine and cosine term encapsulating seasonal and intra-annual variability, and an RMSE to account for noise. In this way, CODED can find typical temporal patterns in the landscape, account for clouds and sensor noise, and better distinguish forests from non-forested areas.

The code below sets up the study area, accesses the Landsat collection to be used, and defines the study period. For this example, we will use the geometry of the previous Landsat scene as our study area. Then, we define a new variable `studyArea` and assign it to the image we used when first exploring NDFI, retrieving its geometry using the `geometry` method. Then, we use the `utils` module to call the `Inputs.getLandsat` function, which accesses the `ImageCollection`. We then filter the images to a start date and end date of interest.. Note: CODED calibrates to find the typical variations in NDFI for the region by observing the NDFI patterns over time, so it is a good idea to filter the Landsat data so you have an extra six months to a year of data before the time period in which you are truly interested. For example, if you wanted to study forest change from 2000–2010, it would be good practice to use 1999 as the start date and 2010 as the end date.. Paste the code below into your starter script you opened to begin this section.

```
// We will use the geometry of the image from the previous section as
// the study area.
var studyArea = ee.Image(
        'LANDSAT/LT05/C02/T1_L2/LT05_226068_19840411')
    .geometry();

// Get cloud masked (Fmask) Landsat imagery.
var landsat = utils.Inputs.getLandsat()
    .filterBounds(studyArea)
    .filterDate('1984-01-01', '2021-01-01');
```

Getting our `ImageCollection` this way saves us quite a bit of coding. By default the returned collection will use every available Landsat mission, perform some simple cloud masking, and generate our image fractions of green vegetation (GV), soil, non-photosynthetic vegetation (NPV), shade-covered material, and NDFI, as well as additional indices.

First, make a new variable `gfwImage` and add the path to the Global Forest Change product from Hansen et al. (2013), which is used to create a forest mask. Define a threshold of 40 for the percent canopy cover for the mask. Apply the threshold to the `treecover2000` band from the `gfwImage`. You could also choose to use a pre-prepared forest mask of your own instead of selecting one from the Global Forest Change product. This might be useful if you have a local high-quality forest mask, as it may slightly improve your results or provide consistency with other work you have done in the area.

```
// Make a forest mask
var gfwImage = ee.Image('UMD/hansen/global_forest_change_2019_v1_7');

// Get areas of forest cover above the threshold
var treeCover = 40;
var forestMask = gfwImage.select('treecover2000')
    .gte(treeCover)
    .rename('landcover');
```

To identify degradation and deforestation, distinguished by whether the land cover remains forest or not after the event, we will use a prepared dataset of forest and non-forested areas. This data already has the predictor data for each feature, which will speed up the computation. Alternatively, refer back to Chapter F2.1 on classification for details on how to create your own unique forest and non-forested area dataset and ensure each feature has a `year` property with the year collected as an integer.

```
var samples = ee.FeatureCollection(
    'projects/gee-book/assets/A3-4/sample_with_pred_hansen_2010');
```

There are many parameters that can be adjusted when running CODED, as summarized below.

- `minObservations`: The minimum number of consecutive observations required to label a disturbance event..
- `chiSquareProbability`: The chi-squared probability is a threshold that controls the sensitivity to change.
- `training`: A training dataset used to specify forest and non-forest. In this example, we will set it to the combination of the *forest* and *non-forest* training points using the `merge` function.
- `forestValue`: The integer value of forest in your training data.
- `startYear`, `endYear`: The start and end years to perform change detection.
- `classBands`: The bands used to train the coefficients.
- Note: `prepTraining` tells the algorithm to add the coefficients to your samples for training the classifier. This will initiate a task to export the prepared samples for later use if you wish. In this example, we will set it to false.

```
var minObservations = 4;
var chiSquareProbability = 0.97;
var training = samples;
var forestValue = 1;
var startYear = 1990;
var endYear = 2020;
var classBands = ['NDFI', 'GV', 'Shade', 'NPV', 'Soil'];
var prepTraining = false;
```

With the parameters defined, we now have everything needed to run CODED. CODED takes a single argument, which is stored as a JavaScript dictionary. Since we defined all the parameters as variables, it may seem redundant to put them into a dictionary, but having them as variables can be helpful when you are exploring functionality and adjusting parameter values frequently. Of course, entering the values directly into the dictionary would work as well.

Create a new dictionary and assign each parameter variable to a key of the same name.

```
//---------------- CODED parameters
var codedParams = {
    minObservations: minObservations,
    chiSquareProbability: chiSquareProbability,
    training: training,
    studyArea: studyArea,
    forestValue: forestValue,
    forestMask: forestMask,
    classBands: classBands,
    collection: landsat,
    startYear: startYear,
    endYear: endYear,
    prepTraining: prepTraining
};

// -------------- Run CODED
var results = api.ChangeDetection.coded(codedParams);
print(results);
```

Run CODED by clicking **Run**. This will set up the run, and issue the call to the `ChangeDetection.coded` function (Fig. A3.4.6).

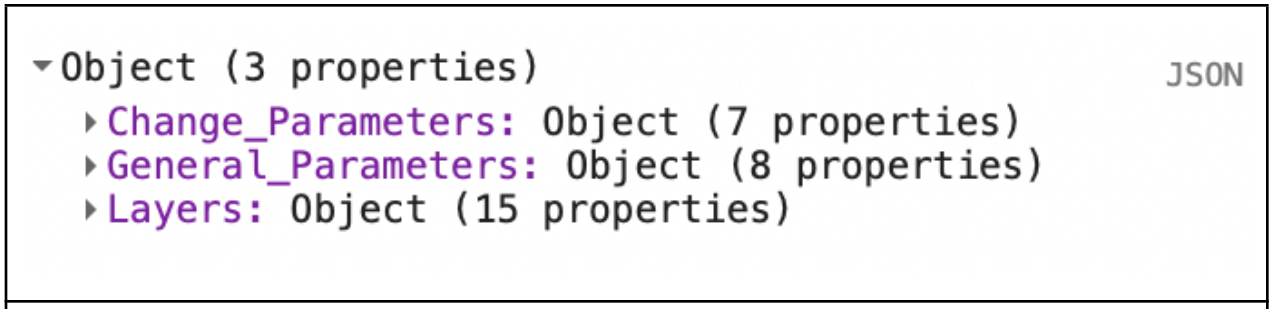


Fig. A3.4.6 The result of running the CODED change detection algorithm is an object with the general and change parameters used for the run and a layers object that has all the image outputs

Code Checkpoint A34d. The book's repository contains a script that shows what your code should look like at this point.

Next, you will relabel some of the results so that they are easier to understand and work with. You will rename the degradation layers to something more human readable, like 'degradation_1', 'degradation_2', etc. Rename the deforestation layers to something more human readable, like 'deforestation_1', 'deforestation_2', etc.

Set a variable for the mask layer. This is the same mask that was passed into CODED, so retrieving it in this method is not strictly necessary.

Set a variable for the change output that is the concatenation of the degradation and deforestation outputs. This is mostly for organizational purposes. Since change is more rare, self masking removes all the non-change pixels, and casting to `Int32` helps keep all the bands in the same type, which you would need for exporting to a geoTIFF.

Set a variable `mag` to the minimum magnitude. There are multiple magnitude bands that correspond to the number of temporal segments that are retrieved when running CODED. These bands are reduced by the minimum since we want to find areas where the greatest negative change has occurred.

```
// Format the results for exporting.
var degradation = results.Layers.DatesOfDegradation
```

```
    .rename(['degradation_1', 'degradation_2',
        'degradation_3', 'degradation_4'
    ]);
var deforestation = results.Layers.DatesOfDeforestation
    .rename(['deforestation_1', 'deforestation_2',
        'deforestation_3', 'deforestation_4'
    ]);
var mask = results.Layers.mask.rename('mask');
var change = ee.Image.cat([degradation, deforestation]).selfMask()
    .toInt32();
var mag = results.Layers.magnitude.reduce(ee.Reducer.min())
    .rename('magnitude');
```

Finally, we can combine all of this information into a stratified—or classified—output layer of forest, non-forest, degradation, and deforestation. We can define a function to take in each of the outputs and apply some logic to decide these categories. A new threshold we need to apply is the magnitude, `magThreshold`. This threshold will define the minimum amount of change we want to qualify as a degradation or deforestation event. This is a post-processing step in which we will stratify the results into categories of stable forest, stable non-forest, degradation, and deforestation.

Next, you will create a new function named `makeStrata` that takes an image and a threshold as the arguments. The base of the stratified image will be the mask which is remapped from [0,1] to [2,1]. This keeps the forest class at a value of 1 and updates the non-forest class to a value of 2.

Then, you will create a binary mask of the minimum threshold using the magnitude threshold parameter. Then, create a binary degradation image using all the degradation bands and then multiply it by the magnitude mask. Similarly, you'll create a binary deforestation image using all the deforestation bands and then multiply it by the magnitude mask. Update the strata image using the `where` functions to first assign degradation to 3 and next to assign deforestation to 4. The `where` functions are applied sequentially. Deforestation needs to be updated last because in this case, areas of degradation could overlap with areas that are also deforestation.

```
var makeStrata = function(img, magThreshold) {
    var strata = img.select('mask').remap([0, 1], [2, 1]);
```

```
    var mag = img.select('magnitude').lte(magThreshold);

    var deg = img.select(['deg.*']).gt(0).reduce(ee.Reducer.max())
        .multiply(mag);
    var def = img.select(['def.*']).gt(0).reduce(ee.Reducer.max())
        .multiply(mag);
    strata = strata.where(deg, 3).where(def, 4);

    return strata.clip(studyArea);
};
```

Concatenate the mask, change, and mag bands into a single image. Define a magnitude threshold of -0.6. Apply the `makeStrata` function using the full output image and the magnitude threshold. Then, add code to export the strata to your assets.

```
var fullOutput = ee.Image.cat([mask, change, mag]);
var magnitudeThresh = -0.6;
var strata = makeStrata(ee.Image(fullOutput), magnitudeThresh)
    .rename('strata');

Export.image.toAsset({
    image: strata,
    description: 'strata',
    region: studyArea,
    scale: 30,
    maxPixels: 1e13,
});
```

Click **Run.** The CODED algorithm is computationally heavy, so the results need to be exported before they can be viewed on the map in the Code Editor. Once the export has finished, you can add the layer to the map. We have created the asset for you and stored it in the book repository; you can access it with the code below:

```
var exportedStrata = ee.Image('projects/gee-book/assets/A3-4/strata');
Map.addLayer(exportedStrata,
    {
```

```
        min: 1,
        max: 4,
        palette: 'green,black,yellow,red'
    },
    'strata');

Map.setCenter(-55.0828, -11.24, 11);
```

The resulting map should look something like the example in Fig. A3.4.9. With these stratified results, you can see the relative amounts and geographic distribution of forest that have been degraded or deforested in your time period of interest. What patterns do you observe? Does forest degradation or deforestation seem more prevalent?

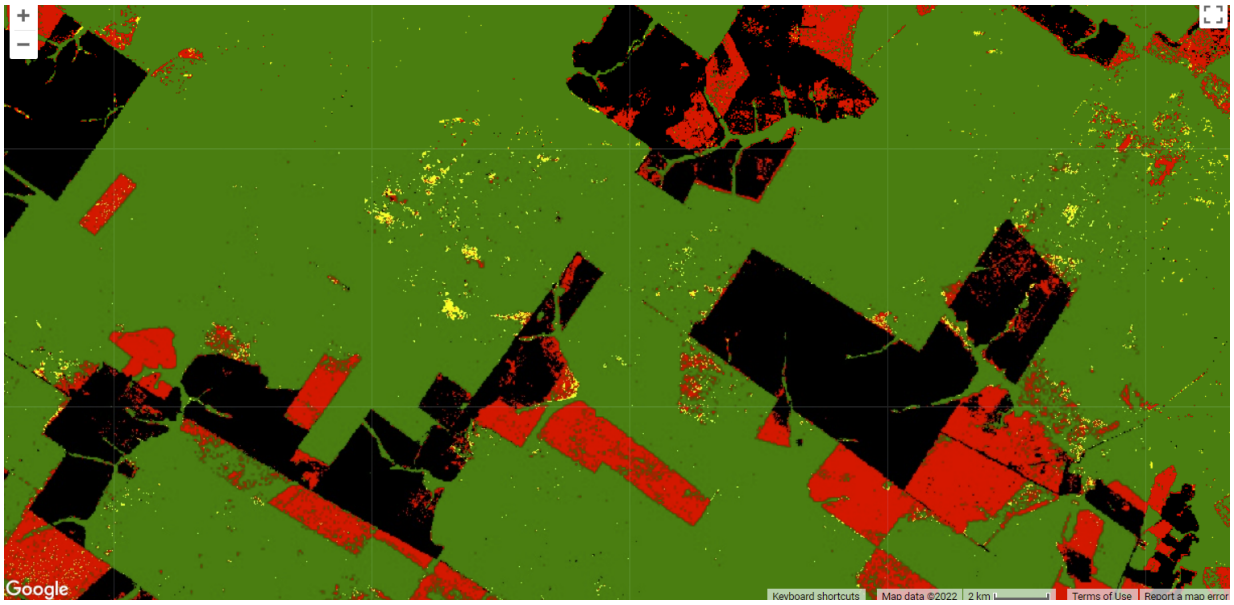


Fig. A3.4.9 Example of classified CODED results for 1990 to 2020. Non-forested areas are black, stable forest areas are green, deforestation is red, and degradation is yellow.

Code Checkpoint A34e. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

Assignment 1. Study how adjusting each parameter affects your results.

a. When you decrease the chi-squared probability, do you see more or less deforestation, and more or less degradation?
b. When you decrease the magnitude threshold, do you see more or less deforestation, and more or less degradation?
c. When you decrease the number of observations, do you see more or less deforestation, and more or less degradation?

Assignment 2. Try adjusting the CODED parameters to see how they affect the detection of degradation and deforestation in a study area where you highly suspect from the imagery that these events have occured. Observe in the map whether you are overestimating or underestimating the disturbed forest area with each parameter combination.

Conclusion

In this chapter you learned about the spectral unmixing algorithm (SMA) and the Normalized Difference Fraction Index (NDFI) in order to map forest degradation and deforestation. We presented two examples of change detection applications: two-image differencing, and an NDFI time-series approach using the CODED algorithm. NDFI is sensitive to subtle changes in forest composition, making it ideal for detection of forest degradation. CODED's use of a regression model fitting of NDFI over a training period informs us more about what the NDFI magnitudes and variability of a forest should be in a healthy state, so changes from the norm can be more confidently labeled as forest disturbance events. However, your chosen CODED parameters have an important impact on your resulting map of forest loss and forest degradation.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Adams JB, Sabol DE, Kapos V, et al (1995) Classification of multispectral images based on fractions of endmembers: Application to land-cover change in the Brazilian Amazon. Remote Sens Environ 52:137–154. https://doi.org/10.1016/0034-4257(94)00098-8

Aryal RR, Wespestad C, Kennedy RE, et al (2021) Lessons learned while implementing a time-series approach to forest canopy disturbance detection in Nepal. Remote Sens 13:2666. https://doi.org/10.3390/rs13142666

Asner GP, Keller M, Pereira R, Zweede JC (2002) Remote sensing of selective logging in Amazonia: Assessing limitations based on detailed field observations, Landsat ETM+, and textural analysis. Remote Sens Environ 80:483–496. https://doi.org/10.1016/S0034-4257(01)00326-1

Asner GP, Keller M, Pereira R, et al (2004) Canopy damage and recovery after selective logging in Amazonia: Field and satellite studies. Ecol Appl 14:280–298. https://doi.org/10.1890/01-6019

Bullock E (2018) Background and Motivation — CODED 0.2 Documentation. https://coded.readthedocs.io/en/latest/background.html. Accessed 28 May 2021

Bullock EL, Woodcock CE, Souza C, Olofsson P (2020) Satellite-based estimates reveal widespread forest degradation in the Amazon. Glob Chang Biol 26:2956–2969. https://doi.org/10.1111/gcb.15029

Bullock E, Nolte C, Reboredo Segovia A (2018) Project impact assessment on deforestation and forest degradation: Forest Disturbance Dataset. 1–44

Cochrane MA (1998) Linear mixture model classification of burned forests in the Eastern Amazon. Int J Remote Sens 19:3433–3440. https://doi.org/10.1080/014311698214109

Hansen MC, Potapov P V, Moore R, et al (2013) High-resolution global maps of 21st-century forest cover change. Science 342:850–853. https://doi.org/science.1244693

Hirschmugl M, Steinegger M, Gallaun H, Schardt M (2013) Mapping forest degradation due to selective logging by means of time series analysis: Case studies in Central Africa. Remote Sens 6:756–775. https://doi.org/10.3390/rs6010756

Kusbach A, Friedl M, Zouhar V, et al (2017) Assessing forest classification in a landscape-level framework: An example from Central European forests. Forests 8:461. https://doi.org/10.3390/f8120461

Matricardi EAT, Skole DL, Cochrane MA, et al (2007) Multi-temporal assessment of selective logging in the Brazilian Amazon using Landsat data. Int J Remote Sens 28:63–82. https://doi.org/10.1080/01431160600763014

Schultz M, Clevers JGPW, Carter S, et al (2016) Performance of vegetation indices from Landsat time series in deforestation monitoring. Int J Appl Earth Obs Geoinf 52:318–327. https://doi.org/10.1016/j.jag.2016.06.020

Small C (2004) The Landsat ETM+ spectral mixing space. Remote Sens Environ 93:1–17. https://doi.org/10.1016/j.rse.2004.06.007

Souza Jr CM, Barreto P (2000) An alternative approach for detecting and monitoring selectively logged forests in the Amazon. Int J Remote Sens 21:173–179. https://doi.org/10.1080/014311600211064

Souza Jr CM, Firestone L, Silva LM, Roberts D (2003) Mapping forest degradation in the Eastern Amazon from SPOT 4 through spectral mixture models. Remote Sens Environ 87:494–506. https://doi.org/10.1016/j.rse.2002.08.002

Souza Jr CM, Roberts DA, Cochrane MA (2005) Combining spectral and spatial information to map canopy damage from selective logging and forest fires. Remote Sens Environ 98:329–343. https://doi.org/10.1016/j.rse.2005.07.013

Woodcock CE, Loveland TR, Herold M, Bauer ME (2020) Transitioning from change detection to monitoring with remote sensing: A paradigm shift. Remote Sens Environ 238:111558. https://doi.org/10.1016/j.rse.2019.111558

Zhu Z, Woodcock CE (2014) Continuous change detection and classification of land cover using all available Landsat data. Remote Sens Environ 144:152–171. https://doi.org/10.1016/j.rse.2014.01.011

Chapter A3.5: Deforestation Viewed from Multiple Sensors

Authors
Xiaojing Tang

Overview
Combining data from multiple sensors is the best way to increase data density and hence detect change faster. The purpose of this chapter is to demonstrate a simple method of combining Landsat, Sentinel-2, and Sentinel-1 data for monitoring tropical forest disturbance. You will learn how to import, preprocess, and fuse optical and synthetic aperture radar (SAR) remote sensing data. You will also learn how to monitor change using time-series models.

Learning Outcomes
- Combining optical and SAR images for change detection.
- Fitting a time-series model to an `ImageCollection`.
- Using established time-series models to detect anomalies in new images.
- Performing change detection with a rolling monitoring window.
- Monitoring forest disturbance in near real time.

Helps if you know how to:
- Understand the characteristics and preprocessing of Landsat and Sentinel images (Part F1).
- Understand regressions in Earth Engine (Chap. F3.0).
- Work with array images (Chap. F3.1, Chap. F4.6).
- Understand the concept of spectral unmixing (Chap. F3.2).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Fit linear and nonlinear functions with regression in an `ImageCollection` time series (Chap. F4.6).
- Export and import results as Earth Engine assets (Chap. F5.0).

Introduction to Theory

Deforestation and forest degradation are large sources of carbon emissions and negatively impact biodiversity, food security, and human well-being. The ability to quickly and accurately detect forest disturbance events is essential for preventing future forest loss and mitigating the negative effects. Combining optical and radar data has the potential to achieve faster detection of forest disturbance than using an individual system. The main challenge is the methodological approach for fusing the different datasets. The Fusion Near Real-Time (FNRT) algorithm (Tang et al. 2022) is a monitoring algorithm for tropical forest disturbance that combines data from Landsat, Sentinel-2, and Sentinel-1. For each sensor system, the FNRT algorithm fits time-series models over data from a three-year training period prior to the one-year monitoring period. FNRT then produces a change score for each new observation collected during the monitoring period based on the residuals and the root-mean-square error (RMSE) of the time-series model. The change scores from all three different sensors are then combined to form one dense time series for change detection. In this chapter, we will learn how to run a simplified version of FNRT to monitor forest disturbance in 2020 for a test area located in the Brazilian Amazon.

Practicum

This lab is designed for advanced users of Earth Engine. We assume that you already know how to import and preprocess large quantities of Landsat, Sentinel-2, and Sentinel-1 data as image collections. The code for importing and preprocessing input data will be provided in the example script, but it will not be discussed in detail in this chapter. To learn more about FNRT, refer to Tang et al. (2022).

Section 1. Understand How FNRT Works

For this lab, we will use a graphical user interface to help us understand conceptually how FNRT combines data from different sensors and detects forest disturbance. Fig. A3.5.1 shows the user interface of the app.

Code Checkpoint A35a. The book's repository contains information about accessing the app.

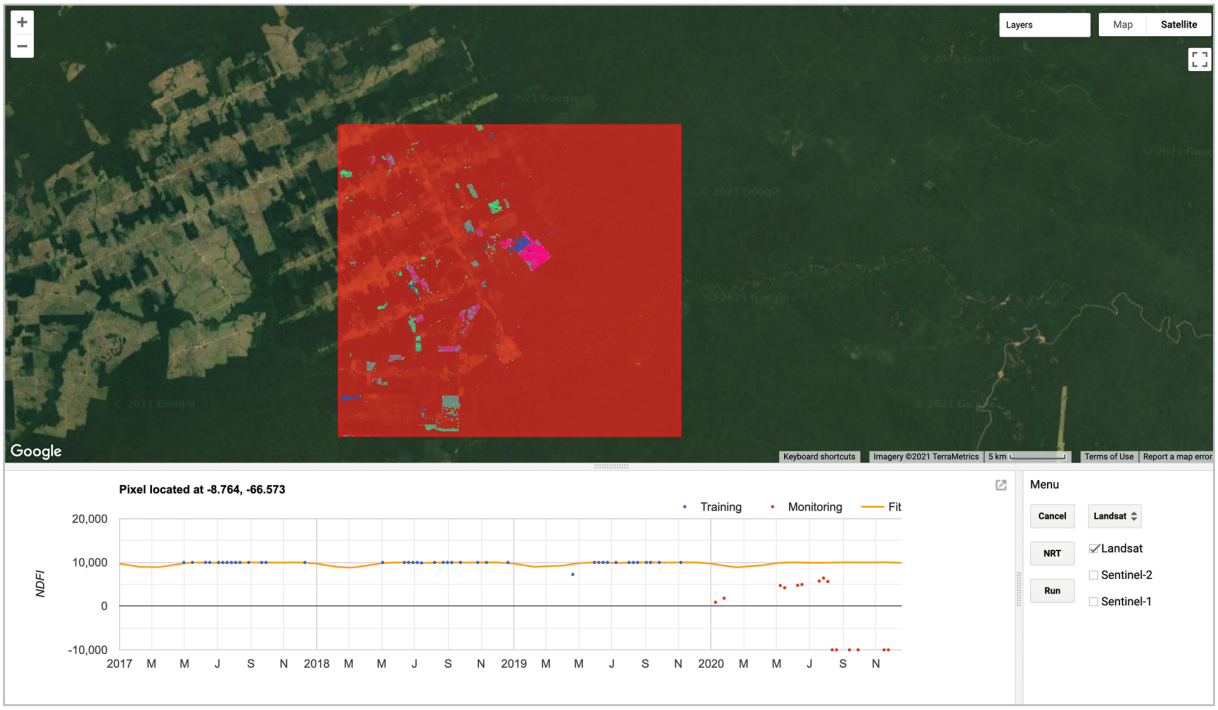


Fig. A3.5.1 User interface of FNRT app. The red box highlights a test site selected for demonstration of the algorithm. At the bottom left is a panel for showing plots of time-series data and model fit. The control menu is at the bottom right.

There are three main tools that can be used to explore the FNRT algorithm. Note that only one plotting tool can be activated at a time, and each tool can be deactivated by clicking the button again:

1. The **Fit** button activates the "fit time-series model" tool, which allows users to click anywhere in the map area, load the time-series data, and fit a harmonic model for the pixel covering the clicked location, using data from the sensor selected in the widget just to the right of the **Fit** button.
2. The **Monitor** button activates the "near real-time monitoring" tool, which allows users to click anywhere in the map area and plot the results of FNRT for the pixel covering the clicked location, using data from all the sensors that were checked via the checkbox widgets to the right of the **Monitor** button.
3. The **Run** button runs FNRT for the entire test area using data from the sensors that are checked, and loads the resulting forest disturbance map to the map area.

Let's start by looking at some examples of time series so we have a better understanding of the model. We first need to know where to look for changes. Let's try to have only "Landsat" checked, and click **Run**. This will run FNRT with only Landsat data and quickly produce a forest disturbance map for 2020 (Fig. A3.5.2a). Now click **Fit** to activate the "Fit time-series model" tool, and then click on a pixel that shows forest disturbance activity according to the map. Wait a few seconds to let the app generate the time-series plot in the time series panel. Once it is completed, you should see a plot similar to Fig. A3.5.2c.

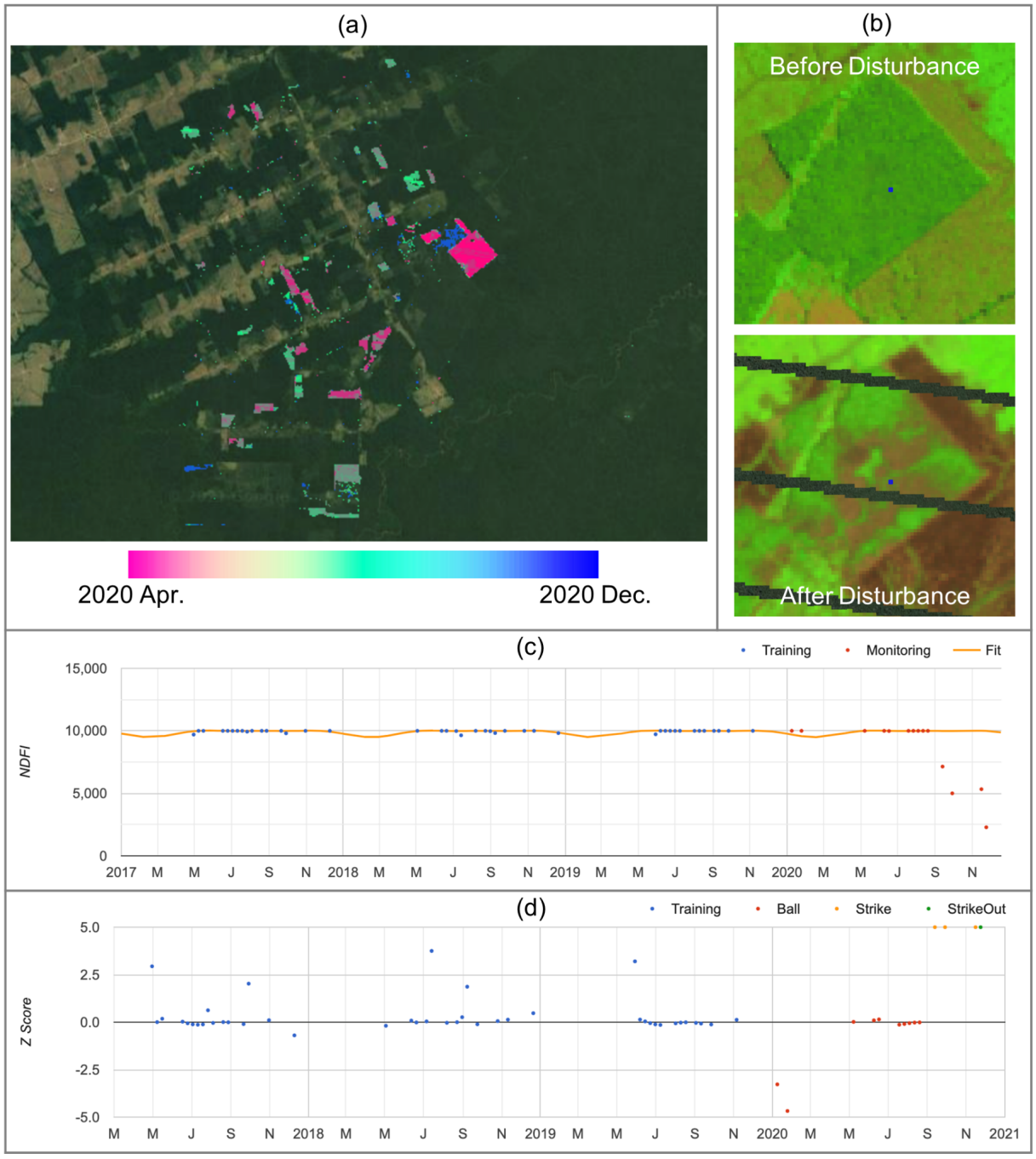


Fig. A3.5.2 (**a**) Forest disturbance map of 2020 for the test area created using Landsat data only; (**b**) Landsat images before and after the forest disturbance; (**c**) time series of Landsat NDFI and model fit; and (**d**) time series of change scores and change detection result

The model-fit plot (Fig. A3.5.2c) shows data from Landsat for the training period (blue) and monitoring period (red). It also shows a harmonic time-series model fit to the training data (orange line). The model fit extends into the monitoring period, which represents the model prediction. Notice that the satellite observations (red points) are very close to the model fit before the forest disturbance occurs. After the disturbance, the observations deviate away from the model fit. The algorithm monitors change by keeping track of differences between the observations and the model predictions.

Now click **Monitor** to activate the "near real-time monitoring" tool, and click the same pixel in the map. The app will then make a plot similar to (d) in Fig. A3.5.2. The near real-time monitoring plot (Fig. A3.5.2d) shows the change scores (Z-score) for the training and the monitoring period. A large change score indicates that something looks different in the satellite image. But sometimes that can be caused by a cloud or a cloud shadow that slipped through the masking process (e.g., the large change scores in the training period). Therefore, the monitoring algorithm looks for multiple large scores within a short monitoring window as an indicator of real forest disturbance.

The monitoring algorithm works through all the observations available during the monitoring period chronologically. Using baseball as an analogy, we flag each change score above a threshold as a "strike" and each change score below the threshold as a "ball." If the algorithm finds multiple strikes in a short monitoring window, a "strikeout" would then be called on the pixel and the date of the first strike would be recorded as the date of change (Fig. A3.5.2d).

Question 1. How many strikes were detected before a forest disturbance was confirmed (strikeout)?

When performing multi-sensor data fusion, FNRT would fit a separate time-series model and calculate change scores using data from each sensor (Fig. 3.4.3a–c). The change scores calculated using data from the three sensors are then combined, and change monitoring is applied to the combined scores (Fig. 3.4.4d).

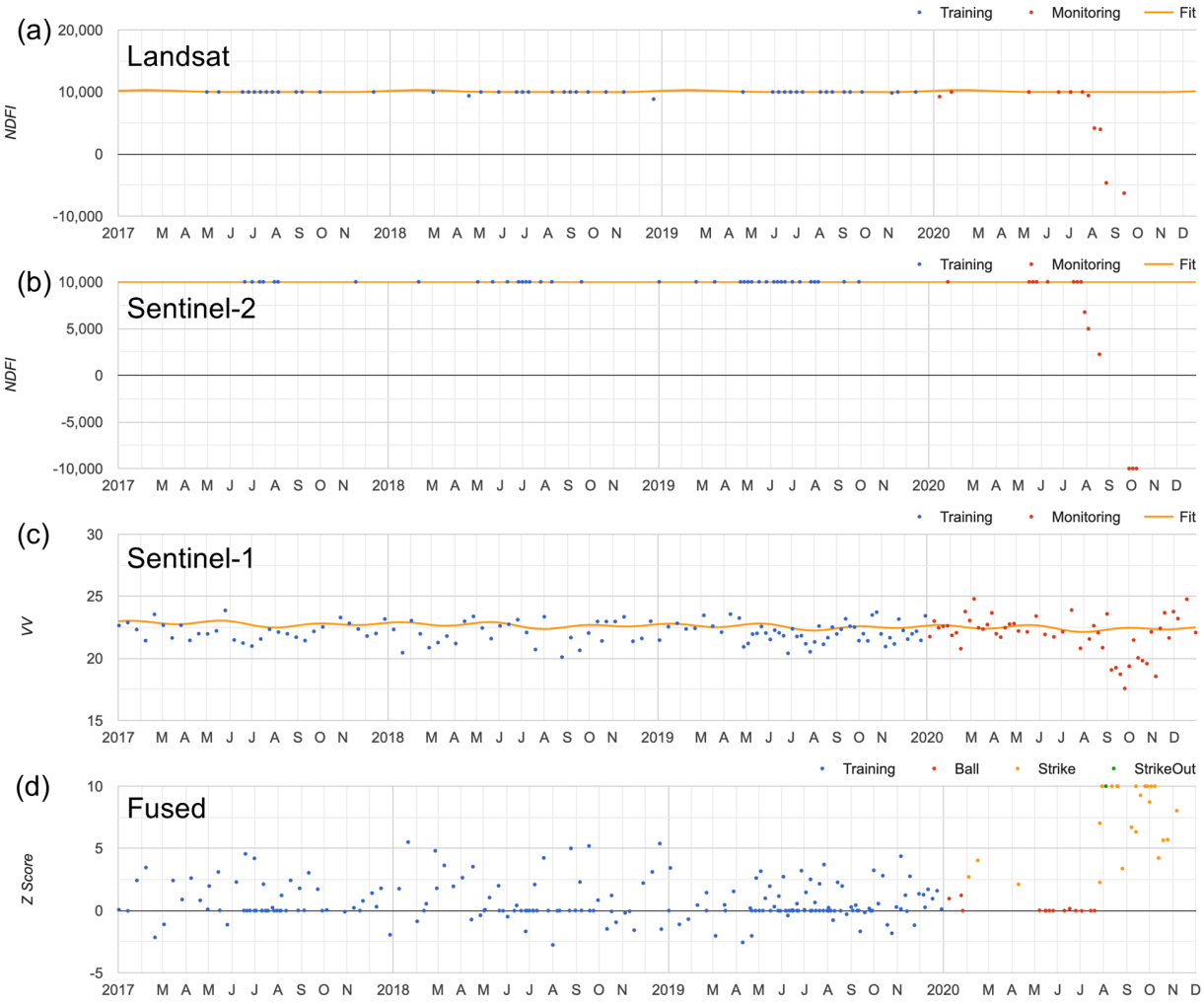


Fig. A3.5.3 Time series data and model fit using (**a**) Landsat, (**b**) Sentinel-2, and (**c**) Sentinel-1 data; (**d**) shows the combined time series of change scores from all three sensors, and the change detection result

The FNRT app also allows us to run change monitoring for the entire test area by clicking the **Run** button. A forest disturbance map showing the date of the disturbances is then added to the map. The users can use the checkbox widgets to choose which data stream is included in the process of making the map (e.g., Fig. A3.5.4). In general, optical data is more sensitive to forest disturbance than SAR data. Combining multiple data streams can help detect the disturbance earlier, compared to using data from each individual sensor.

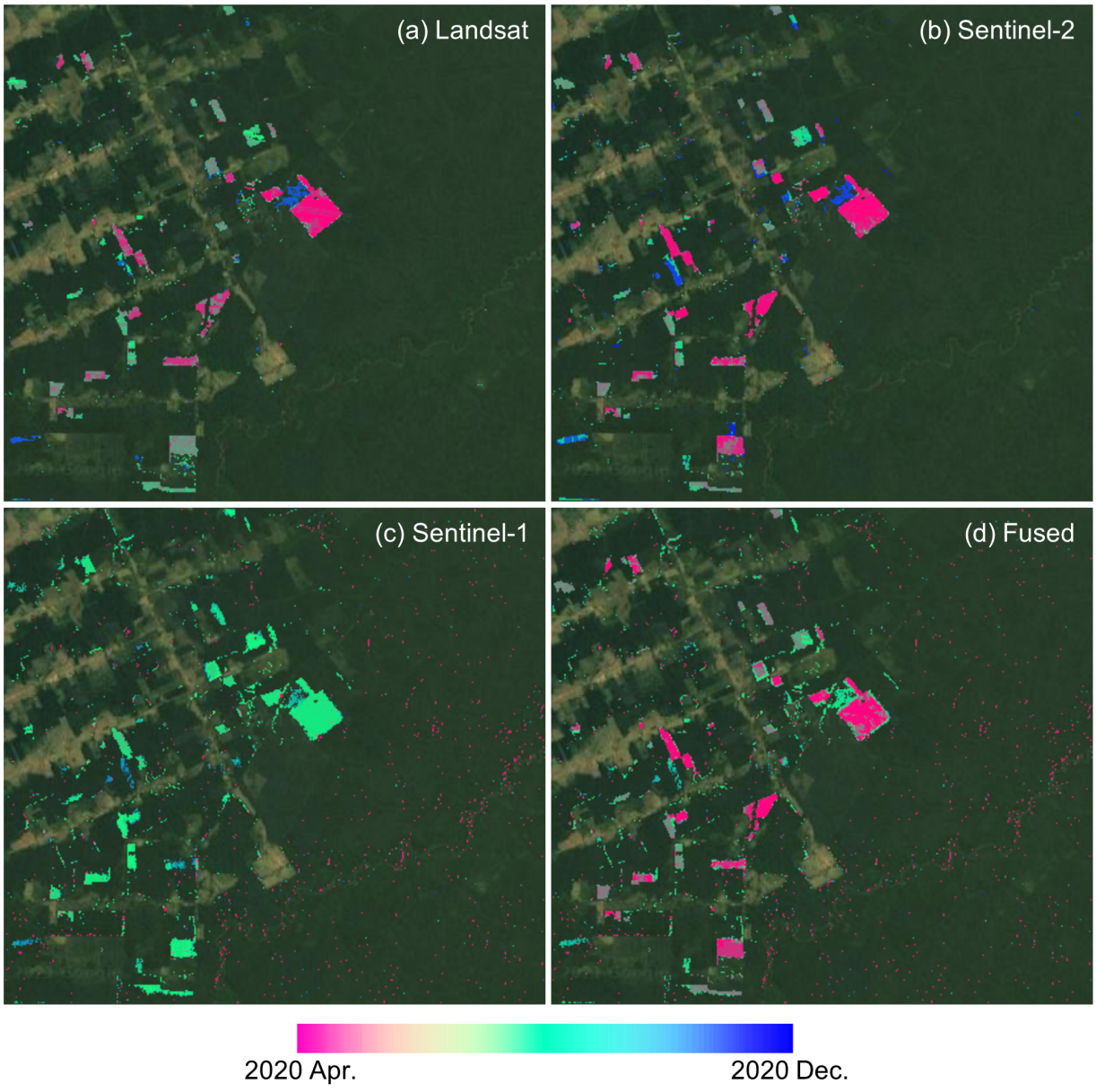


Fig. A3.5.4 Forest disturbance map made using data from (**a**) Landsat, (**b**) Sentinel-2, (**c**) Sentinel-1, and (**d**) all three sensors.

Question 2. Can you find forest disturbances that were detected earlier through the use of data from all three sensors rather than using just Landsat data?

Now that we understand conceptually how FNRT works, let's create a simple script to implement it.

Section 2. Define Study Area and Model Parameters

Section 2.1. Study Area

There are several ways to define a study area: 1) import an existing `FeatureCollection` from an Earth Engine asset; 2) use the drawing tool of the Code Editor; and 3) create a polygon by entering the coordinates. Here we select a box, approximately 20 km by 20 km, located in the Brazilian Amazon as our study area.

```
var testArea = ee.Geometry.Polygon(
    [
        [
            [-66.73156878460787, -8.662236005089952],
            [-66.73156878460787, -8.916025640576244],
            [-66.44867083538912, -8.916025640576244],
            [-66.44867083538912, -8.662236005089952]
        ]
    ]);

Map.centerObject(testArea);
```

Note that because of the complexity of the FNRT algorithm, it requires quite a lot of computational resources and can result in a "User Memory Limit" error if applied to too large of an area. Divide the study area into multiple tiles if you are trying to monitor a large area.

Section 2.2. Model Parameters

A few model parameters need to be defined and can be tuned to optimize the algorithm for specific regions or monitoring conditions.

```
// Start and end of the training and monitoring period.
var trainPeriod = ee.Dictionary({
    'start': '2017-01-01',
    'end': '2020-01-01'
});
```

```
var monitorPeriod = ee.Dictionary({
    'start': '2020-01-01',
    'end': '2021-01-01'
});

// Near-real-time monitoring parameters.
var nrtParam = {
    z: 2,
    m: 5,
    n: 4
};

// Sensor specific parameters.
var lstParam = {
    band: 'NDFI',
    minRMSE: 0.05,
    strikeOnly: false
};
var s2Param = {
    band: 'NDFI',
    minRMSE: 0.05,
    strikeOnly: false
};
var s1Param = {
    band: 'VV',
    minRMSE: 0.01,
    strikeOnly: true
};
```

For demonstration of the algorithm, we define the year 2020 as our monitoring period (`monitorPeriod`). We then recommend using the three years prior to the monitoring period as the training period (`trainPeriod`) to establish baseline time-series models.

A few parameters can be used to control the sensitivity of the algorithm: a z-threshold (`z`) defines the minimum change score required to flag a possible change observation; an m-size (`m`) defines the size of the monitoring window, or how many consecutive observations we check for change; and an n-change (`n`) defines the number of possible

change observations within the monitoring window required to confirm a forest disturbance.

For each sensor we also need to select the band (`band`) to use for monitoring; a minimum RMSE (`minRMSE`) to prevent overly sensitive detection caused by unusually good model fit; and a strike-only flag (`strikeOnly`) to allow a data stream to contribute only to confirmation of a change.

Code Checkpoint A35b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Import and Preprocess Data

We need to import data from each sensor for the training period and the monitoring period as two seperate image collections. Each image in the `ImageCollection` needs to be preprocessed. For optical data, we need to apply spectral unmixing and calculate the Normalized Difference Fraction Index (NDFI) (Souza et al. 2005). Since spectral unmixing would be applied to both Landsat and Sentinel-2 data, let's first implement a common "unmixing" function.

Several studies (e.g., Bullock et al. 2020, Chen et al. 2021) have shown that spectral unmixing and NDFI work very well in detecting tropical forest disturbance. Here we use the same endmembers used in Souza et al. (2005) for the spectral unmixing, and calculate NDFI based on Equations 4 and 5 in Souza et al. (2005).

```
var unmixing = function(col) {

    // Define endmembers and cloud fraction threshold.
    var gv = [500, 900, 400, 6100, 3000, 1000];
    var npv = [1400, 1700, 2200, 3000, 5500, 3000];
    var soil = [2000, 3000, 3400, 5800, 6000, 5800];
    var shade = [0, 0, 0, 0, 0, 0];
    var cloud = [9000, 9600, 8000, 7800, 7200, 6500];
    var cfThreshold = 0.05;

    return col.map(function(img) {
        // Select the spectral bands and perform unmixing
        var unmixed = img.select(['Blue', 'Green', 'Red',
```

```
            'NIR',
            'SWIR1', 'SWIR2'
        ])
        .unmix([gv, shade, npv, soil, cloud], true,
            true)
        .rename(['GV', 'Shade', 'NPV', 'Soil',
            'Cloud'
        ]);

    // Calculate Normalized Difference Fraction Index.+
    var NDFI = unmixed.expression(
        '10000 * ((GV / (1 - SHADE)) - (NPV + SOIL)) / ' +
        '((GV / (1 - SHADE)) + (NPV + SOIL))', {
            'GV': unmixed.select('GV'),
            'SHADE': unmixed.select('Shade'),
            'NPV': unmixed.select('NPV'),
            'SOIL': unmixed.select('Soil')
        }).rename('NDFI');

    // Mask cloudy pixel.
    var maskCloud = unmixed.select('Cloud').lt(
        cfThreshold);
    // Mask all shade pixel.
    var maskShade = unmixed.select('Shade').lt(1);
    // Mask pixel where NDFI cannot be calculated.
    var maskNDFI = unmixed.expression(
        '(GV / (1 - SHADE)) + (NPV + SOIL)', {
            'GV': unmixed.select('GV'),
            'SHADE': unmixed.select('Shade'),
            'NPV': unmixed.select('NPV'),
            'SOIL': unmixed.select('Soil')
        }).gt(0);

    // Scale fractions to 0-10000 and apply masks.
    return img
        .addBands(unmixed.select(['GV', 'Shade',
                'NPV', 'Soil'
```

```
                ])
                .multiply(10000))
            .addBands(NDFI)
            .updateMask(maskCloud)
            .updateMask(maskNDFI)
            .updateMask(maskShade);
    });
};
```

For Landsat data, we need to load the Landsat 7 and Landsat 8 Surface Reflectance product as an `ImageCollection`. We can filter the collection based on our study area and time period. Each image in the `ImageCollection` is masked based on the QA band; then spectral unmixing is applied and NDFI calculated. The `loadLandsatData` function is provided for importing and preprocessing Landsat data.

Preprocessing of Sentinel-2 data is similar to that of Landsat data because they are both optical remote sensing data and are almost identical in many ways. We need to load the Sentinel-2 top-of-atmosphere product (surface reflectance data do not have long enough time series for our purpose) and the Sentinel-2 cloud probability product as image collections, and filter based on our study area and time period. We then join the two collections. Each image in the collection is then masked based on both the QA band and the cloud probability.

For Sentinel-1 data, we use the Sentinel-1 SAR GRD product. Preprocessing of Sentinel-1 data includes: 1) calculation of backscattering; 2) calculation of the ratio of VH and VV; 3) calculation of three-pixel spatial mean; 4) choice of the orbital pass with the most data for the region of interest; and 5) radiometric slope correction using volume model. Please refer to Mullissa et al. (2021) for more detail on the preprocessing of SAR data.

Here we assume you have already learned how to load and preprocess large quantities of Landsat, Sentinel-2, and Sentinel-1 data. We will not discuss these steps in detail here in this lab. Instead, three functions (`loadLandsatData`, `loadS2Data`, and `loadS1Data`) are provided in a separate script with which we can import using the `require` function. Now we can load all the required input data. We also need to apply the spectral unmixing and calculate NDFI for the Landsat and Sentinel-2 data.

```
var input = require(
    'projects/gee-edu/book:Part A - Applications/A3 - Terrestrial
Applications/A3.5 Deforestation Viewed from Multiple
Sensors/modules/Inputs'
);
var lstTraining = unmixing(input.loadLandsatData(testArea,
    trainPeriod));
var lstMonitoring = unmixing(input.loadLandsatData(testArea,
    monitorPeriod));
var s2Training = unmixing(input.loadS2Data(testArea, trainPeriod));
var s2Monitoring = unmixing(input.loadS2Data(testArea,
    monitorPeriod));
var s1Training = input.loadS1Data(testArea, trainPeriod);
var s1Monitoring = input.loadS1Data(testArea, monitorPeriod);
```

In addition to the remote sensing data, we also need to create a forest mask to limit our monitoring to forested areas. Here we will use the Hansen Global Forest Change dataset. We first use the "tree canopy cover" layer to find areas that had larger than 50% canopy cover in 2000. We then remove all the forest loss area during 2001–2019 and add in all the forest gain from 2000–2012 to create a mask of forested area by the beginning of our monitoring period.

```
var hansen = ee.Image('UMD/hansen/global_forest_change_2020_v1_8')
    .unmask();
var forestMask = hansen.select('treecover2000')
    .gt(50)
    .add(hansen.select('gain'))
    .subtract(hansen.select('loss'))
    .add(hansen.select('lossyear')
        .eq(20))
    .gt(0)
    .clip(testArea);
```

Before we move on to the next section, let's do some simple checking. Print out the image collections and check whether the correct data is loaded. You can also load the forest mask and visually examine the quality of the mask.

```
var maskVis = {
    min: 0,
    max: 1,
    palette: ['blue', 'green']
};
Map.addLayer(forestMask, maskVis, 'Forest Mask');
print('lstTraining', lstTraining);
print('lstMonitoring', lstMonitoring);
print('s2Training', s2Training);
print('s2Monitoring', s2Monitoring);
print('s1Training', s1Training);
print('s1Monitoring', s1Monitoring);
```

Question 3. How many images are available from each sensor for our monitoring period? Are you surprised by the numbers? (Note that one sensor may have more data because the study area happened to be in the side-lap zone where two orbit passes overlap.) What bands are included in the data of each sensor?

Code Checkpoint A35c. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Establish Baseline Time Series Model

Before we can monitor change in near real time, we need to establish baseline time-series models using the data of the training period, so that we know what to expect from new observations in the monitoring period. Due to the different physical nature of optical and radar data, it is easier to fit a separate time-series model to data from each sensor. Here we use a harmonic time-series model similar to the one used in the Continuous Change Detection and Classification (CCDC) algorithm (Zhu and Woodcock 2014). Because the timestamps of all the input images are stored in the unit of milliseconds while the harmonic model prefers a unit of years, we first need to define a function to convert any date object into the unit of fractional year (e.g. 2015-03-01 to 2015.1612).

```
var toFracYear = function(date) {
    var year = date.get('year');
    var fYear = date.difference(
```

```
        ee.Date.fromYMD(year, 1, 1), 'year');
    return year.add(fYear);
};
```

Here we define a function (`fitHarmonicMode`) to fit a harmonic model to an `ImageCollection` of time-series data. We first construct the dependent variables according to Equation 1 in Zhu and Woodcock (2014) and add them to each of the input images as new bands (using `addDependents` function). Then we use a reducer (`ee.Reducer.robustLinearRegression`) to fit the model to the data using robust linear regression. The function returns the model coefficients and the model RMSE.

```
var fitHarmonicModel = function(col, band) {
    // Function to add dependent variables to an image.
    var addDependents = function(img) {
        // Transform time variable to fractional year.
        var t = ee.Number(toFracYear(
            ee.Date(img.get('system:time_start')), 1));
        var omega = 2.0 * Math.PI;
        // Construct dependent variables image.
        var dependents = ee.Image.constant([
                1, t,
                t.multiply(omega).cos(),
                t.multiply(omega).sin(),
                t.multiply(omega * 2).cos(),
                t.multiply(omega * 2).sin(),
                t.multiply(omega * 3).cos(),
                t.multiply(omega * 3).sin()
            ])
            .float()
            .rename(['INTP', 'SLP', 'COS', 'SIN',
                'COS2', 'SIN2', 'COS3', 'SIN3'
            ]);
        return img.addBands(dependents);
    };

    // Function to add dependent variable images to all images.
    var prepareData = function(col, band) {
```

```
        return ee.ImageCollection(col.map(function(img) {
            return addDependents(img.select(band))
                .select(['INTP', 'SLP', 'COS',
                    'SIN',
                    'COS2', 'SIN2', 'COS3',
                    'SIN3',
                    band
                ])
                .updateMask(img.select(band)
                    .mask());
        }));
    };

    var col2 = prepareData(col, band);
    // Fit model to data using robust linear regression.
    var ccd = col2
        .reduce(ee.Reducer.robustLinearRegression(8, 1), 4)
        .rename([band + '_coefs', band + '_rmse']);

    // Return model coefficients and model rmse.
    return ccd.select(band + '_coefs').arrayTranspose()
        .addBands(ccd.select(band + '_rmse'));
};
```

With this function (`fitHarmonicModel`) we can fit the harmonic time-series model to the training data of all three sensors. It is always a good practice to also add some metadata to the results for future reference. The model fitting process will take a few minutes and quite a lot of computer memory. Therefore it is recommended that we run them as tasks and save the results as assets before we proceed to the next step.

```
// Fit harmonic models to training data of all sensors.
var lstModel = fitHarmonicModel(lstTraining, lstParam.band)
    .set({
        region: 'test',
        sensor: 'Landsat'
    });
var s2Model = fitHarmonicModel(s2Training, s2Param.band)
```

```
    .set({
        region: 'test',
        sensor: 'Sentinel-2'
    });
var s1Model = fitHarmonicModel(s1Training, s2Param.band)
    .set({
        region: 'test',
        sensor: 'Sentinel-1'
    });

// Define function to save the results.
var saveModel = function(model, prefix) {
    Export.image.toAsset({
        image: model,
        scale: 30,
        assetId: prefix + '_CCD',
        description: 'Save_' + prefix + '_CCD',
        region: testArea,
        maxPixels: 1e13,
        pyramidingPolicy: {
            '.default': 'sample'
        }
    });
};

// Run the saving function.
saveModel(lstModel, 'LST');
saveModel(s2Model, 'S2');
saveModel(s1Model, 'S1');
```

Run the script. The model fitting results in export tasks to create assets. Instead of exporting the results for this exercise, however, you should proceed to the next section. In that section we will continue with a precomputed asset that is the same as what the above tasks would create.

Code Checkpoint A35d. The book's repository contains a script that shows what your code should look like at this point.

Section 5. Create Predicted Values for the Monitoring Period

We will now start exploring the pre-exported results mentioned in the previous section. Place this code below the checkpoint of the previous section:

```
var models = ee.ImageCollection('projects/gee-book/assets/A3-5/ccd');
var lstModel = models
    .filterMetadata('sensor', 'equals', 'Landsat').first();
var s2Model = models
    .filterMetadata('sensor', 'equals', 'Sentinel-2').first();
var s1Model = models
    .filterMetadata('sensor', 'equals', 'Sentinel-1').first();
```

In this section we will use the time-series models that we produced in Section 4 to create a predicted image (also commonly referred to as a synthetic image; see Zhu et al. 2015) for any given date. The predicted image can give us a good estimate of the reflectance (or index such as NDFI) value of a place assuming no change has occurred. We can then compare the predicted image to the image collected by the satellite on the same date. Areas that look very different would likely be areas that have changed. Note that all the coefficients of the harmonic model were saved as a single-band array. So the very first step here is to define a function `dearrayModel` to convert the array image into a multiband image with each coefficient in one band.

```
var dearrayModel = function(model, band) {
    band = band + '_';

    // Function to extract a non-harmonic coefficients.
    var genCoefImg = function(model, band, coef) {
        var zeros = ee.Array(0).repeat(0, 1);
        var coefImg = model.select(band + coef)
            .arrayCat(zeros, 0).float()
            .arraySlice(0, 0, 1);
        return ee.Image(coefImg
            .arrayFlatten([
                [ee.String('S1_')
                    .cat(band).cat(coef)
                ]
```

```
            ]));
    };

    // Function to extract harmonic coefficients.
    var genHarmImg = function(model, band) {
        var harms = ['INTP', 'SLP', 'COS', 'SIN',
            'COS2', 'SIN2', 'COS3', 'SIN3'
        ];
        var zeros = ee.Image(ee.Array([
                ee.List.repeat(0, harms.length)
            ]))
            .arrayRepeat(0, 1);
        var coefImg = model.select(band + 'coefs')
            .arrayCat(zeros, 0).float()
            .arraySlice(0, 0, 1);
        return ee.Image(coefImg
            .arrayFlatten([
                [ee.String(band).cat('coef')], harms
            ]));
    };

    // Extract harmonic coefficients and rmse.
    var rmse = genCoefImg(model, band, 'rmse');
    var coef = genHarmImg(model, band);
    return ee.Image.cat(rmse, coef);
};
```

Second, we need to define a function that creates a predicted image for all the dates for which a real image is available during the monitoring period.

```
var createPredImg = function(modelImg, img, band, sensor) {
    // Reformat date.
    var date = toFracYear(ee.Date(img.get('system:time_start')));
    var dateString = ee.Date(img.get('system:time_start'))
        .format('yyyyMMdd');
    // List of coefficients .
    var coefs = ['INTP', 'SLP', 'COS', 'SIN',
```

```
        'COS2', 'SIN2', 'COS3', 'SIN3'
    ];
    // Get coefficients images from model image.
    var coef = ee.Image(coefs.map(function(coef) {
        return modelImg.select(".*".concat(coef));
    })).rename(coefs);
    var t = ee.Number(date);
    var omega = 2.0 * Math.PI;
    // Construct dependent variables.
    var pred = ee.Image.constant([
            1, t,
            t.multiply(omega).cos(),
            t.multiply(omega).sin(),
            t.multiply(omega * 2).cos(),
            t.multiply(omega * 2).sin(),
            t.multiply(omega * 3).cos(),
            t.multiply(omega * 3).sin()
        ])
        .float();
    // Matrix multiply dependent variables with coefficients.
    return pred.multiply(coef).reduce('sum')
        // Add original image and rename bands.
        .addBands(img, [band]).rename(['predicted', band])
        // Preserve some metadata.
        .set({
            'sensor': sensor,
            'system:time_start': img.get('system:time_start'),
            'dateString': dateString
        });
};
```

The `createPredImg` function calculates a predicted image based on the date of the real image and the model coefficients. It returns a new image with both the predicted image and the real image.

We can then apply the `createPredImg` function to each image in the image collections of the monitoring data. We can define another function to `map` over an `ImageCollection` to apply `createPredImg` to all images in the collection.

```
var addPredicted = function(data, modelImg, band, sensor) {
    return ee.ImageCollection(data.map(function(img) {
        return createPredImg(modelImg, img, band,
            sensor);
    }));
};
```

Let's apply `addPredicted` to monitoring data of all three sensors. Note this process is still sensor-specific. Let's print out the result for Landsat to examine the structure of the output.

```
// Convert models to non-array images.
var lstModelImg = dearrayModel(lstModel, lstParam.band);
var s2ModelImg = dearrayModel(s2Model, s2Param.band);
var s1ModelImg = dearrayModel(s1Model, s1Param.band);

// Add predicted image to each real image.
var lstPredicted = addPredicted(lstMonitoring, lstModelImg,
    lstParam.band, 'Landsat');
var s2Predicted = addPredicted(s2Monitoring, s2ModelImg,
    s2Param.band, 'Sentinel-2');
var s1Predicted = addPredicted(s1Monitoring, s1ModelImg,
    s1Param.band, 'Sentinel-1');

print('lstPredicted', lstPredicted);
```

Question 4. How many images are in the `ImageCollection` of the predicted images? Is it the same number as the original `ImageCollection` of the monitoring data?
Code Checkpoint A35e. The book's repository contains a script that shows what your code should look like at this point.

Section 6. Calculate Change Scores

Assuming a good model fit, the predicted image would be very close to the real image of the same date for areas with no change. So what we need to do now is examine each pair of real and predicted images and look for areas where they differ from each other. And if we merge the three data streams, we would be able to significantly increase the

data density and potentially detect changes faster. But before we can fuse the three data streams together, we need to produce a normalized measurement of the likelihood of change.

The FNRT algorithm calculates a change score (Tang et al. 2019) very similar to a z-score. First, it calculates the residuals defined as the difference between the predicted value and the observed value. The residuals are then scaled with the RMSE of the time-series model to get the change score. If a change score is above the preset threshold (`z` in the parameters), then it is flagged as "possible change" (or a "strike" in baseball terminology). The implementation of this part is quite simple:

```
// Function to calculate residuals.
var addResiduals = function(data, band) {
    return ee.ImageCollection(data.map(function(img) {
        return img.select('predicted')
            // Restrict predicted value to be under 10000
            .where(img.select('predicted').gt(10000),
                10000)
            // Calculate the residual
            .subtract(img.select(band))
            .rename('residual')
            // Save some metadata
            .set({
                'sensor': img.get('sensor'),
                'system:time_start': img.get(
                    'system:time_start'),
                'dateString': img.get(
                    'dateString')
            });
    }));
};

// Function to calculate change score and flag change.
var addChangeScores = function(data, rmse, minRMSE,
    threshold, strikeOnly) {
    // If strikeOnly then we need a mask for balls.
    var mask = ee.Image(0);
```

```
    if (strikeOnly) {
        mask = ee.Image(1);
    }

    return ee.ImageCollection(data.map(function(img) {
        // Calculate change score
        var z = img.divide(rmse.max(minRMSE));
        // Check if score is above threshold
        var strike = z.multiply(z.gt(threshold));
        // Create the output image.
        var zStack = ee.Image.cat(z, strike).rename([
                'z', 'strike'
            ])
            .set({
                'sensor': img.get('sensor'),
                'system:time_start': img.get(
                    'system:time_start')
            });
        // Mask balls if strikeOnly.
        return zStack.updateMask(strike.gt(0).or(
            mask));
    }));
};
```

Note that here we restrict the predicted value to be under 10000 because that is the theoretical maximum for NDFI.

We also need to have a minimum RMSE (`minRMSE`) when scaling the residuals. This is because when using indices such as NDFI, it is sometimes possible to have perfect model fits resulting in very low or even zero RMSE, which would then cause the model to be overly sensitive to subtle variations in the monitoring data. The `minRMSE` is calculated for each pixel as a percentage of the average values of all observations in the training period.

For this particular implementation of FNRT, the determination of a strike is directional, meaning that we are looking for changes only in one direction. This is because we already expect that a forest disturbance will cause a decrease in NDFI or VV

backscattering. Modification would be required if monitoring in another direction (e.g., if using SWIR reflectance) or bidirectional monitoring is desired. Let's apply the two new functions to our three data streams:

```
// Add residuals to collection of predicted images.
var lstResiduals = addResiduals(lstPredicted, lstParam.band);
var s2Residuals = addResiduals(s2Predicted, s2Param.band);
var s1Residuals = addResiduals(s1Predicted, s1Param.band);

// Add change score to residuals.
var lstScores = addChangeScores(
    lstResiduals, lstModelImg.select('.*rmse'),
    lstPredicted.select(lstParam.band).mean()
    .abs().multiply(lstParam.minRMSE),
    nrtParam.z, lstParam.strikeOnly);
var s2Scores = addChangeScores(
    s2Residuals, s2ModelImg.select('.*rmse'),
    s2Predicted.select(s2Param.band).mean()
    .abs().multiply(s2Param.minRMSE),
    nrtParam.z, s2Param.strikeOnly);
var s1Scores = addChangeScores(
    s1Residuals, s1ModelImg.select('.*rmse'),
    s1Predicted.select(s1Param.band).mean()
    .abs().multiply(s1Param.minRMSE),
    nrtParam.z, s1Param.strikeOnly);

print('lstScores', lstScores);
```

Code Checkpoint A35f. The book's repository contains a script that shows what your code should look like at this point.

Question 5. What bands are included in the change scores images?

Section 7. Multisensor Data Fusion and Change Detection

Now we are finally ready to fuse the three data streams together. This step is quite straightforward: we just need to `merge` the three image collections of the change scores, and then `sort` them by their timestamps.

```
var fused = lstScores.merge(s2Scores).merge(s1Scores)
    .sort('system:time_start');
```

The change monitoring function uses a binary array to keep track of the number of strikes within a monitoring window. The size of the monitoring window is defined by the model parameters (`nrtParam.m`). The default monitoring window size is five.

We first initiate the monitoring window with a binary array of 0 with a size of five bits (00000) for each pixel. When we iterate through all the change score images chronologically, we shift the binary array left for those pixels that were not masked in the current change score image (00000 → 0000). We then append the current change flag (strike or not) to the shifted binary array (0000 → 00001). The process continues, as we iterate through all the change scores images (e.g., 00001 → 00011 → 00110 → 01101 → 11011). Every time a new value is appended to the binary array, we also check how many strikes (1s) are there. If the number of strikes exceeds the required number (`nrtParam.n`) to flag a change, then a change is flagged and the date of the change recorded.

```
var monitorChange = function(changeScores, nrtParam) {
    // Initialize an empty image.
    var zeros = ee.Image(0).addBands(ee.Image(0))
        .rename(['change', 'date']);
    // Determine shift size based on size of monitoring window.
    var shift = Math.pow(2, nrtParam.m - 1) - 1;
    // Function to monitor.
    var monitor = function(img, result) {
        // Retrieve change image at last step.
        var change = ee.Image(result).select('change');
        // Retrieve change date image at last step.
        var date = ee.Image(result).select('date');
        // Create a shift image to shift the change binary array
        // left for one space so that new one can be appended.
        var shiftImg = img.select('z').mask().eq(0)
            .multiply(shift + 1).add(shift);
        change = change.bitwiseAnd(shiftImg)
            .multiply(shiftImg.eq(shift).add(1))
```

```
            .add(img.select('strike').unmask().gt(0));
        // Check if there are enough strike in the current
        // monitoring window to flag a change.
        date = date.add(change.bitCount().gte(nrtParam.n)
            // Ignore pixels where change already detected.
            .multiply(date.eq(0))
            // Record change date where change is flagged.
            .multiply(ee.Number(toFracYear(
                ee.Date(img.get(
                    'system:time_start')), 1))));
        // Combine change and date layer for next iteration.
        return (change.addBands(date));
    };

    // Iterate through the time series and look for change.
    return ee.Image(changeScores.iterate(monitor, zeros))
        // Select change date layer and selfmask.
        .select('date').rename('Alerts').selfMask();
};
```

Now we apply the monitoring function (`monitorChange`) to the fused data (`fused`) using the default model parameters (`nrtParam`). We can then add the result, the forest disturbance map, to the map panel to visually examine it.

```
var alerts = monitorChange(fused, nrtParam).updateMask(forestMask);
print('alerts', alerts);

// Define a visualization parameter.
var altVisParam = {
    min: 2020.4,
    max: 2021,
    palette: ['FF0080', 'EC1280', 'DA2480', 'C83680', 'B64880',
        'A35B80', '916D80', '7F7F80', '6D9180', '5BA380',
        '48B680', '36C880', '24DA80', '12EC80', '00FF80',
        '00EB89', '00D793', '00C49D', '00B0A7', '009CB0',
        '0089BA', '0075C4', '0062CE', '004ED7', '003AE1',
        '0027EB', '0013F5', '0000FF'
```

```
    ]
};
Map.centerObject(testArea, 10);
Map.addLayer(alerts, altVisParam, 'Forest Disturbance Map (2020)');
Map.setOptions('SATELLITE');
```

Code Checkpoint A35g. The book's repository contains a script that shows what your code should look like at this point.

If we run the whole script now, we can create a map of forest disturbance that occurred in 2020 for our test area. The visualization of the map shows us the timing of all the forest disturbance detected by the FNRT algorithm (Fig. A3.5.5). You will notice a few large disturbance events. There may also be some tiny speckles, which are likely errors of commission. Try adjusting the monitoring parameters (`nrtParam`) to see if you can improve the map. Another common practice is to apply a spatial filter to filter out scattered single pixels of change (Bullock et al. 2020).

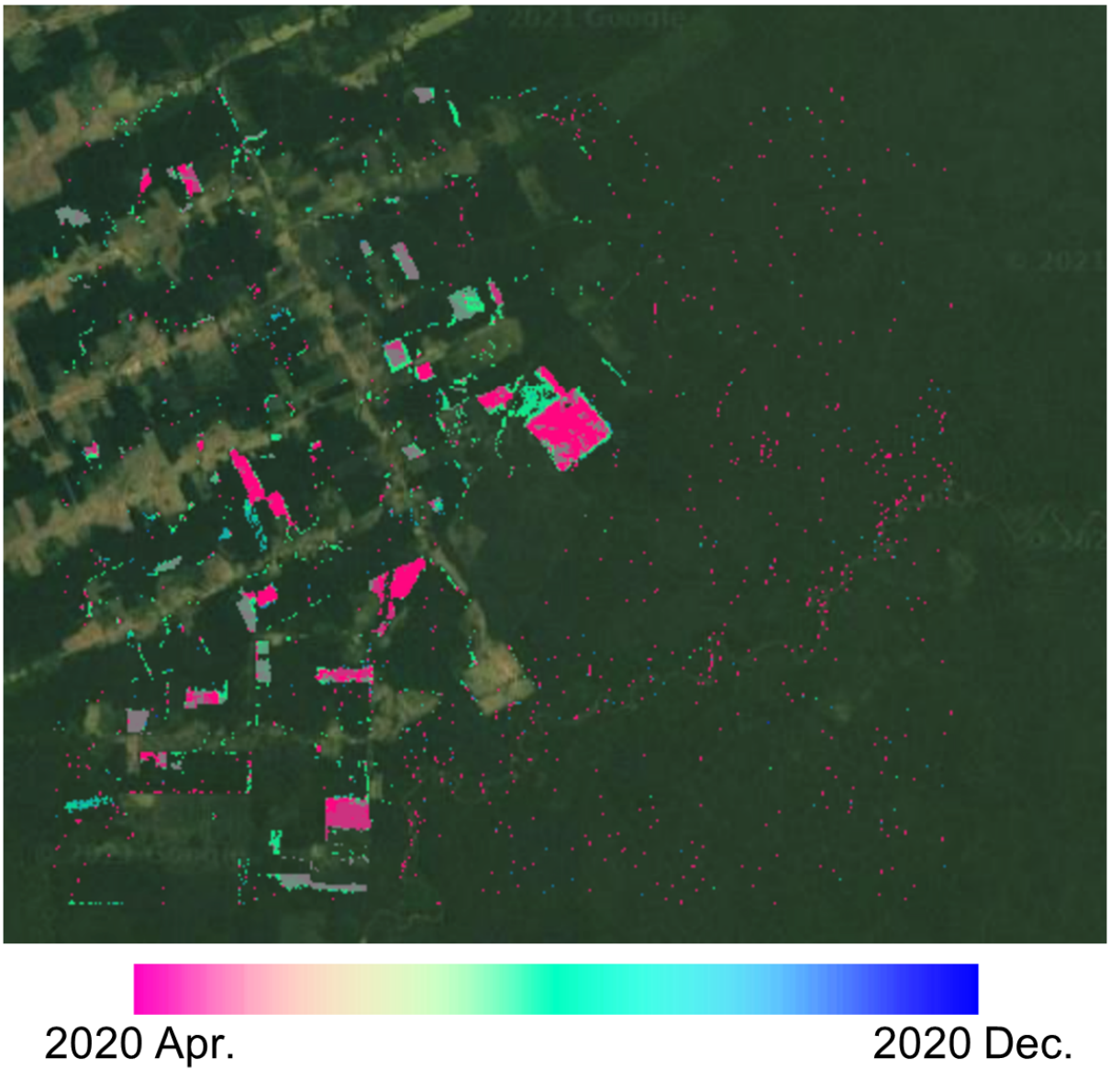


Fig. A3.5.5 Map of forest disturbance occurring in 2020 for a selected test area located on the frontline of deforestation in the Brazilian Amazon

Question 6. Can you run the change detection without Sentinel-1 input or with just Landsat input? Was forest disturbance detected faster when we used data from all three sensors?

Synthesis

Now that you have learned how to implement the simplified version of FNRT for the test area that we selected, let's try to use it to do some real monitoring of tropical forest disturbance.

Assignment 1. Modify the code and apply it in a different area of your choice. Note that you will need to keep the area small in order to be able to run everything easily in the Code Editor.

Assignment 2. Try to monitor more recent changes by changing the monitoring period to 2021. Note you would need to change the training period to 2018–2020, and also update the forest mask.

Assignment 3. If you want to try a more challenging task, you can integrate your results into the FNRT app that we used in Sect. 1.

Conclusion

In this chapter we experimented with a simple way to combine data from three different sensors (Landsat, Sentinel-2, and Sentinel-1) for the purpose of detecting tropical forest disturbance as early as possible. We started with loading and preprocessing the input data. We learned how to fit a harmonic model to time-series data, and how to make predicted observations based on a model fit. We also learned how to look for changes with a moving monitoring window over a time series of change scores. The full FNRT algorithm is more complex than the "lite" version that we implemented in this lab. If interested, please refer to Tang et al. (2022) for more details.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Bullock EL, Woodcock CE, Olofsson P (2020) Monitoring tropical forest degradation using spectral unmixing and Landsat time series analysis. Remote Sens Environ 238:110968. https://doi.org/10.1016/j.rse.2018.11.011

Chen S, Woodcock CE, Bullock EL, et al (2021) Monitoring temperate forest degradation on Google Earth Engine using Landsat time series analysis. Remote Sens Environ 265:112648. https://doi.org/10.1016/j.rse.2021.112648

Mullissa A, Vollrath A, Odongo-Braun C, et al (2021) Sentinel-1 SAR backscatter analysis ready data preparation in Google Earth Engine. Remote Sens 13:1954. https://doi.org/10.3390/rs13101954

Souza Jr CM, Roberts DA, Cochrane MA (2005) Combining spectral and spatial information to map canopy damage from selective logging and forest fires. Remote Sens Environ 98:329–343. https://doi.org/10.1016/j.rse.2005.07.013

Tang X, Bratley KE, Cho K, et al (2022) Near real-time monitoring of tropical forest disturbance by fusion of Landsat, Sentinel-2, and Sentinel-1 data. Remote Sens Environ, in review

Tang X, Bullock EL, Olofsson P, et al (2019) Near real-time monitoring of tropical forest disturbance: New algorithms and assessment framework. Remote Sens Environ 224:202–218. https://doi.org/10.1016/j.rse.2019.02.003

Zhu Z, Woodcock CE (2014) Continuous change detection and classification of land cover using all available Landsat data. Remote Sens Environ 144:152–171. https://doi.org/10.1016/j.rse.2014.01.011

Zhu Z, Woodcock CE, Holden C, Yang Z (2015) Generating synthetic Landsat images based on all available Landsat data: Predicting Landsat surface reflectance at any given time. Remote Sens Environ 162:67–83. https://doi.org/10.1016/j.rse.2015.02.009

Outline

Below is an outline of this document, including every section header.