SAR Intensity Ratio Time Series Change Detection

Usage Documentation

Table of Contents

I. Dependencies

This script makes use of the Google Earth Engine (GEE) Python API. Therefore, the user must have a GEE account and the *ee* Python package installed. When running the script, the user will be prompted to authenticate their account.

In addition, please make sure the following libraries are installed:

- NumPy
- Pandas
- SciPy
- Matplotlib

II. Background

Synthetic Aperture Radar (SAR) is an active remote sensing method that penetrates cloud cover and provides coverage during both day and night. Changes in the properties of ground surface targets can be detected using images of received backscatter intensity. Within a pixel, the reflectance, orientation, roughness, moisture content, and dielectric properties of all scatters in the area affect the backscatter intensity. The most common change detection method is a ratio of two intensity images (I_{ratio}). Changes in land cover, such as with floods, deforestation, or landslides can be found this way. In the case of landslides, the replacing of existing ground cover, often forest, with the bare earth of a landslide scar typically reduces the amount of radiation backscattered to the satellite, and therefore the area affected will have a higher intensity ratio than unaffected areas.

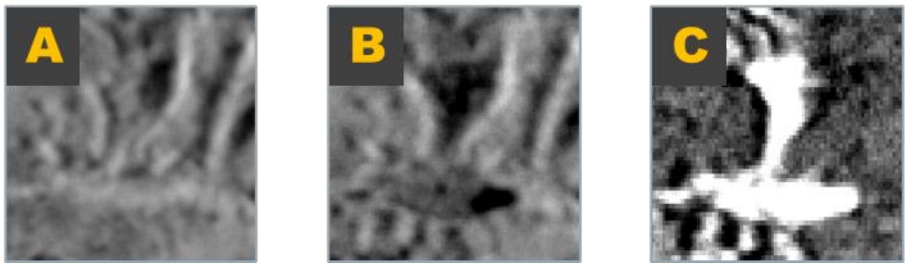

Figure 1. Grayscale images of a) backscatter intensity before landslide, b) intensity after, and c) ratio of the first two images. The white pixels of high ratio indicate a landslide dozens of meters in length, with a scarp at the top and toe emptying into a valley below.

Handwerger et al. (2021) have developed code that automates this process using Google Earth Engine (GEE), intended to detect landslide events. Their JavaScript code runs directly in the GEE code editor. Rather than taking the ratio of a single image before and a single image after an event, they create a stack of images for each time period, then find the median of the stacks. Stacking increases the signal-to-noise ratio. Using GEE eliminates the need to download data to a local drive; saving quite a bit of time since one Sentinel-1 image is usually about 4 gigabytes in size. Since the GEE Sentinel-1 data is in decibels ($10*\log_{10}(I)$, where I is intensity), the operation performed is: $I_{ratio} = I_{pre} – I_{post}$. This is an equivalent procedure to taking the ratio of raw intensity due to the rules of logarithms. Additionally, they found that pixels above the 99th percentile of the overall image characterized landslides well.

III. Purpose

This document, and the accompanying code, describes an attempt to employ the Handwerger et al. (2021) method in successively repeating intervals. The time between the SAR stacks can be thought of as the 'detection window' – a time interval in which changes in the Earth's surface can be discriminated. By iteratively moving this window forward, a time series of intensity ratio images is created.

Users are expected to input either polygons, or a region to grid over. In either case, the grid cells or polygons represent zones to check for changes within. The percentage of pixels above the 99th percentile is calculated for each zone for each window. From there, the locations and dates of possible changes are found. To help determine if each change was real, the code outputs a graph of the time series, optical imagery of the event, and mean NDVI of the event area.

IV. Methods

To run the code, two files are needed. First, *SAR_I_ratio_Timeseries.ipynb*. This contains the bulk of the code and functions; therefore, users need only open this Jupyter Notebook and follow the steps contained within. Second, *ChangeDetect.py*, which contains a single function – this is the code from Handwerger et al. (2021), converted to Python. No dataset needs to be included in the working directory with these two files, as the input data will be stored in the cloud by Earth Engine.

Workflow

1. Define area of interest and create a regular grid over the area.
2. Create stacks of SAR images at initially defined dates, then find median of each stack to create before/after image.
3. Find the 99th percentile of the ratio between before and after images.
4. For each cell in the grid, find total count of pixels and number of pixels above the 99th percentile, then save to a table.
5. Repeat steps 2 – 4, moving forward by a set number of days. Calculate percentage of pixels over 99th percentile in each cell, for each window.
6. Run peak detection over the time series of each cell. Only count peaks that are above a certain threshold.
7. Record date and cell ID of each possible detection from step 6.
8. For every possible detection, export a Sentinel-2 satellite optical image from before and after the given date. Then find NDVI before and after each possible detection.

Data Structure

Input Data:

1. **Earth Engine asset**
   The asset is read in as a feature collection. An asset can be created within the Earth Engine code editor by drawing a geometry on the map view, importing it to the editor, then exporting to an asset. This is more useful for a single rectangular area of interest. Alternatively, shapefiles can be imported directly as assets, allowing for hundreds of polygons to be used as an input, if not more. The code can process the data in three ways: grid over bounding box, grid over geometries, or use raw geometries. For information on handling these cases, see *Parameters,* section V.

Intermediate Data:

1. **Earth Engine images and image collections**
   All SAR data are represented as one of these object types. This includes the intensity images themselves and products such as the intensity ratio images. Fortunately, as hundreds or thousands of images may be processed across all windows, computations are done server-side.
2. **CSVs of zonal statistics results**
   Two CSV files are exported from GEE into user's Google Drive account:
   - Number of pixels above 99th percentile per zone per window
   - Total pixels within zone per window

   These CSV files must be downloaded and brought into the directory that contains *SAR_I_ratio_Timeseries.ipynb*. In addition, the mean date of each window is saved as a CSV to the current working directory.
3. **NumPy array of I_{ratio} time series for each polygon**
   The CSV data are re-organized so that each row represents one zone, and the columns represent the statistic over time.
4. **List of detection dates and locations**
   Through peak detection analysis of the NumPy arrays, dates and locations of possible change events are obtained.

Outputs:

1. **Time series chart of each grid cell with possible detection**
   This figure automatically sizes to fit the total number of detections. The peaks found with the peak detection analysis are plotted on top of the time series.
2. **Before/after Sentinel-2 image of each grid cell with possible detection**
   Using GEE, optical imagery from Sentinel-2 is clipped to the extent of each grid cell.
3. **CSV of mean NDVI of each grid cell before/after possible detection**
   Calculated with Sentinel-2 optical data using GEE.

Functions

A number of custom functions are called in the main script.

***I_Ratio*()** – Contained within *ChangeDetect.py*. Calculates amplitude ratio of SAR stacks. It is called once for every window to create an intensity ratio image.

***MakeGrid*()** – Takes a polygon and creates a grid of polygons inside the shape. It can handle a single polygon, or a collection of disconnected polygons.

***Add_ID_to_Features*()** – Gives a unique ID number to each feature in a feature collection, as a new property. This will work on the collection created with *MakeGrid()* or with any other feature collection.

***WindowDate_to_CSV*()** – Converts NumPy array of epoch times into human-readable format. The input is an array of dates of windows, obtained from *MovingWindowDetectChange*().

***ZonalStats*()** – Computes zonal statistics across an image collection as a table. An output value is computed for every zone in the input zone collection, using every image in the value collection.

***MovingWindowDetectChange*()** – Performs SAR intensity change detection algorithm over a moving window. Makes use of *ZonalStats*() to find the number of pixels above the 99th percentile in a zone and the total number of pixels within a zone.

***Get_BeforeAfter_Imagery*()** – Obtains cloud-masked Sentinel-2 imagery as GeoTIFFs, before and after a given date. Clips the imagery to a specific region and exports the results to Google Drive.

***Get_BeforeAfter_NDVI*()** – Obtains cloud-masked median NDVI over time. From this median NDVI image, finds the mean NDVI within a region. Performs this task before and after a given date.

V. Main Script

The main script can be found within *SAR_I_ratio_Timeseries.ipynb* after the function definitions. Initial parameters may be changed to suit the user, then *MovingWindowDetectChange*() is ran for the number of windows specified. The resulting time series CSVs are read into a NumPy array and peak detection performed with SciPy's *argrelextrema*(). From the resulting dates and locations of detections, there is an output of time series graphs, optical images, and an NDVI table.

Example Case

For demonstration purposes, the usage document and Jupyter Notebook use a dataset near Hiroshima, Japan. A rainfall event between June 28 and July 9, 2018 triggered thousands of landslides. A small subset area of interest and year of windows from 2018-2019 are used, intersecting the dates and locations of some of the landslides that occurred. All figures and code snippets refer to this example.

Parameters

Provide a path to the Earth Engine asset to import as a feature collection object. An asset could be created specifically to use with this code, or a pre-existing asset can be used.

```
ee_import = ee.FeatureCollection("users/bvoelker/hiroshima_landslide_subset")
```

This is used to define the zones used for zonal statistics. There are several possibilities for how to define the zones:

1. **Grid over bounding box**. **This is the default method**. Users will not have to change anything, except grid cell spacing if desired. Note that a smaller *grid_size* is more likely to cause Earth Engine to run out of memory when running the script.

```
aoi = ee_import.geometry().bounds()
grid_size = 500 # meters -- defines the spacing of grid cells.
grid = MakeGrid(aoi, grid_size)
grid = Add_ID_to_Features(grid)
```

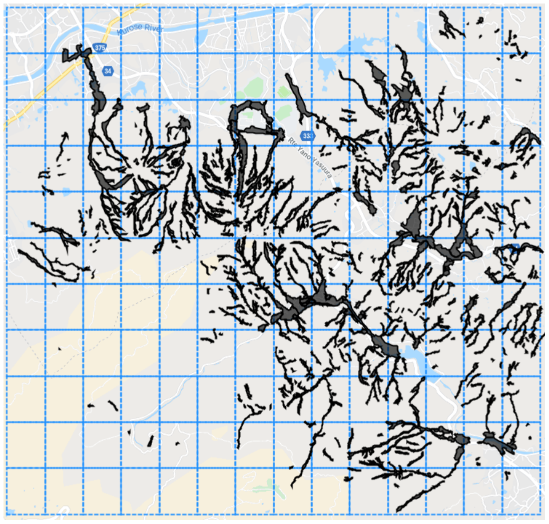

Figure 2. Grid over dataset boundary with 500 meter spacing.

2. **Grid over geometries.** Simply remove the *bounds*() method from the above code. The grid cells will fit to the shapes in the data, but very small or thin features will be excluded (Fig. 3). Not ideal for the example dataset, but useful for a collection of polygons that are all larger than *grid_size*.

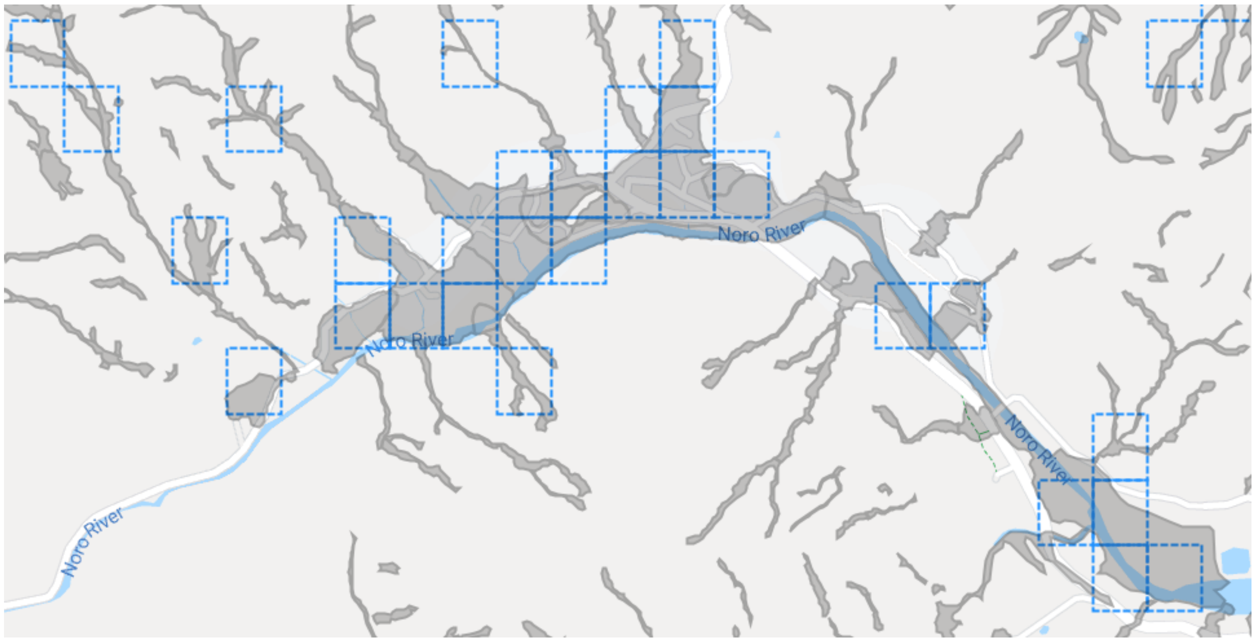

Figure 3. Grid over landslide polygons with 100 meter spacing.

3. **Use raw polygon geometries.** To accomplish this, don't use the *MakeGrid*() function. However, the *grid* variable is referred to elsewhere in the script, so just define *grid* as the polygons. In practice, it would look like:

```
ee_import = ee.FeatureCollection("users/bvoelker/hiroshima_landslide_subset")
grid = ee_import.geometry()
grid = Add_ID_to_Features(grid)
```

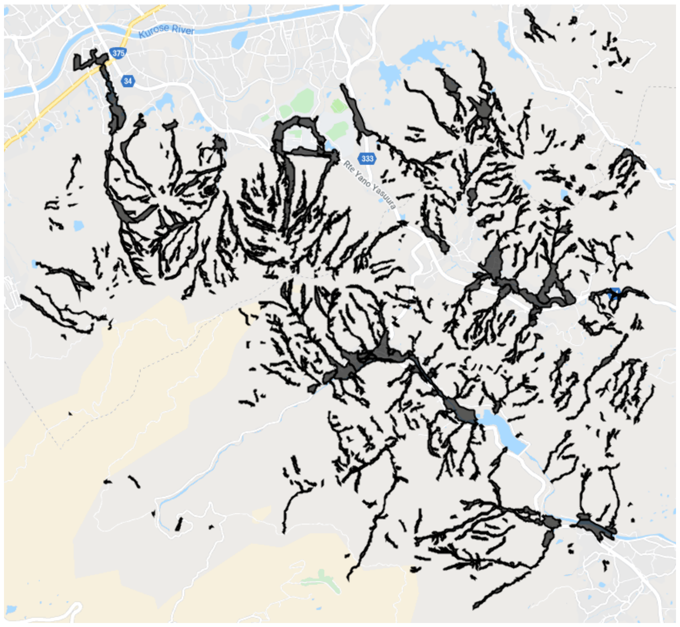

Figure 4. No grid. Polygon shapes are used as zones.

Next, set the threshold for the detection window (*detection_threshold*). During the peak detection step, only peaks above this threshold will be counted. The remaining parameters to set involve the detection window. Set the size, of windows and stacks, then the number of windows to use. The *pre_window_Start* variable defines the start date of the first pre-window stack – then it will be incremented forward an amount equal to *sizeWindows* for a number of time steps equal to *numWindows* when running *MovingWindowDetectChange*()

```
sizeWindows = 7 # days

numWindows = 52 # approx. 1 year

sizePreStack = 365 # days

sizePostStack = 60 # days

pre_window_Start = ee.Date('2017-03-01T00:00') # format: 'yyyy-mm-dd-HH':MM
```

A value of *sizePostStack* is chosen based on the results of Handwerger et al (2021). They found that detection capability increases with more images in the stack but increases less rapidly after two months. A 60-day stack size was chosen to find peaks quickly, near the time of the event, while maintaining as much detection capability as possible. Meanwhile, *sizePreStack* is one year so that an entire cycle of seasonal vegetation change is captured and averaged over.

The window start time (*window_Start*), window end time (*window_End*), and end time of the post-window stack (*post_window_End*) are determined automatically based on the values of *sizePreStack*, *sizeWindows*, and *sizePostStack*, respectively. All times are incremented forward in the same way as *pre_window_Start*.

Zonal Statistics Time Series

Zonal statistics is the process of extracting summary statistics for zones defined by one dataset, with values coming from a raster dataset. This is performed for every window, using the zones defined by *grid* and value raster data being the intensity ratio image obtained from *I_ratio*(). Additionally, a NumPy array containing the mean date between the start and end of each window is obtained. The process is encapsulated within *MovingWindowDetectChange*().

```
# Create an array to hold running list of window dates.
windowEpochTime = np.empty(numWindows, dtype = float)

for window in range(numWindows):
    ChangeImg = I_Ratio(aoi, slope_threshold, curv_threshold,
                pre_window_Start, window_Start,
                window_End, post_window_End)

    # Get the approximate date of the window by finding the mean of the begir
    windowDateAvg = (window_Start.getInfo()['value'] + window_End.getInfo()
    # Add the date of this window to the array.
    windowEpochTime[window] = windowDateAvg

    # Prepare for the next computation.
    # Move all dates forward a set number of days equal to the window size.
    pre_window_Start = pre_window_Start.advance(sizeWindows, 'day')
    window_Start = window_Start.advance(sizeWindows, 'day')
    window_End = window_End.advance(sizeWindows, 'day')
    post_window_End = post_window_End.advance(sizeWindows, 'day')
```

Sometimes there might not be any Sentinel-2 data for a given date. The function checks to see if *ChangeImg* is null, and if it is, skips to the next window. An image collection named *ChangeCollection* is created that contains every *ChangeImg*.

From here, *ZonalStats*() is called to obtain statistics for every zone, for every window. In GEE, *map*() can be used on an image collection to apply a function to every image inside. This is used by *ZonalStats*() to apply Earth Engine's *reduceRegion*() to every image. *ZonalStats*() is called twice to obtain the *ee.Reducer.sum*() and the *ee.Reducer.count*() statistics.

The input value raster to *ZonalStats*() function is binary, where 0 represents a pixel under the 99th percentile, and 1 represents a pixel above the 99th percentile. Therefore, the sum statistic represents the total number of pixels over the 99th percentile in a zone.

```
def ZonalStats(valueCollection, zoneCollection, statistic, scale = 10, tileScale = 16):
  def ZS_Map(image):
    """Define zonal statistics operation, then apply to entire image collection
    Adapted from:
    https://gis.stackexchange.com/questions/333392/gee-reduceregions-for-an-image-collection """
    return image.reduceRegions(collection = zoneCollection,
                               reducer = statistic,
                               scale = scale,
                               tileScale = tileScale)

  reduced = valueCollection.map(ZS_Map)
```

Peak Detection

As a result of *MovingWindowDetectChange*(), two CSV files are created that contain the zonal statistics (sum and count) of the zones described in the *Parameters* section. The CSVs are formatted so that each row is the data for one zone, and the columns are the properties of the zone, including the statistic. Each successive row is a new zone, all within the same window. This process repeats until all zones are used, where the next window begins, starting the pattern over from zone ID 1. Thus, the windows are vertically 'stacked' on each other, and the data for a single zone are separated, making it hard to plot a time series.

The data is re-organized with NumPy so that the data from successive windows is all gathered in one row. Now, each row represents all data for a zone, and the columns represent the statistic across successive window dates.

```
data_sum = np.transpose(data_sum_raw['sum']
                        .values
                        .reshape(-1, numPolys)
                       )
data_count = np.transpose(data_count_raw['count']
                          .values
                          .reshape(-1, numPolys)
                         )
```

The number of rows in these arrays are equal to the number of zones, and the number or columns equal to the number of windows. An array containing the unique ID numbers of the zones and a DataFrame containing the window dates are also obtained, to be cross-referenced with the time series array.

Finally, dividing *data_sum* by *data_count* creates an array with the ratio of pixels above the 99th percentile for each zone, called *data_normalized*. Conceptually, it may also be thought of as the percentage of the area of the zone with I_{ratio} values above the threshold. Here, rows still represent zones, and columns are the ratio across each successive window. This array is the input to the peak detection function.

```
>>> print(data_normalized)
```

```
[[0.0004 0.0004 0.0012 ... 0.0008 0.0004 0.0012]
 [0.0029 0.0021 0.0025 ... 0.0067 0.0067 0.0075]
 [0.0009 0.0005 0.0005 ... 0.0014 0.0005 0.0018]
 ...
 [0.0164 0.0304 0.0336 ... 0.0124 0.0124 0.0116]
 [0.0004 0.     0.     ... 0.0008 0.0004 0.0008]
 [0.034  0.0608 0.0684 ... 0.0036 0.0056 0.0084]]
```

Peak detection is handled with *argrelextrema*() from SciPy's signal processing toolbox. Now that the data is in the proper format, we can just iterate through the rows in *data_normalized*, putting each row as an argument, i.e.

```
for row_number, row in enumerate(data_normalized):
        peak_index = scipy.signal.argrelextrema(row, comparator = np.greater, order = 3)
```

This obtains the indices of peaks. To filter out potential false positives, within the loop, some additional criteria are imposed. First, only peaks above *detection_threshold* are counted. Second, detections won't be counted unless they are separated by an amount of time equal to about half the time duration of the pre-window stack. This is because *argrelextrema*() may detect multiple peaks close together. Also, if there is a change event, such as a landslide, there will be a certain time period where both pre-event and post-event images will be in the stack. Since the stacks are averaged, the pre-event data will be dominant until half of the time period of the stack has passed, and then the post-event data will become the new baseline. For example, if the pre-event stack is one year, and a peak is detected in February, all peaks until August are ignored, because there is uncertainty about whether the peaks in between these dates are due to the February event, or a new event.

VI. Outputs

Plot Generation

For each result in the peak detection step, the entire time series is plotted, with the detected peaks in the series marked (Fig. 4). The dimensions of the subplot automatically adjust to the number of peak detections. Because the subplots are arranged in a rectangular grid, there may be a few empty plots at the end if the number of detections was lower than the total number of subplots.

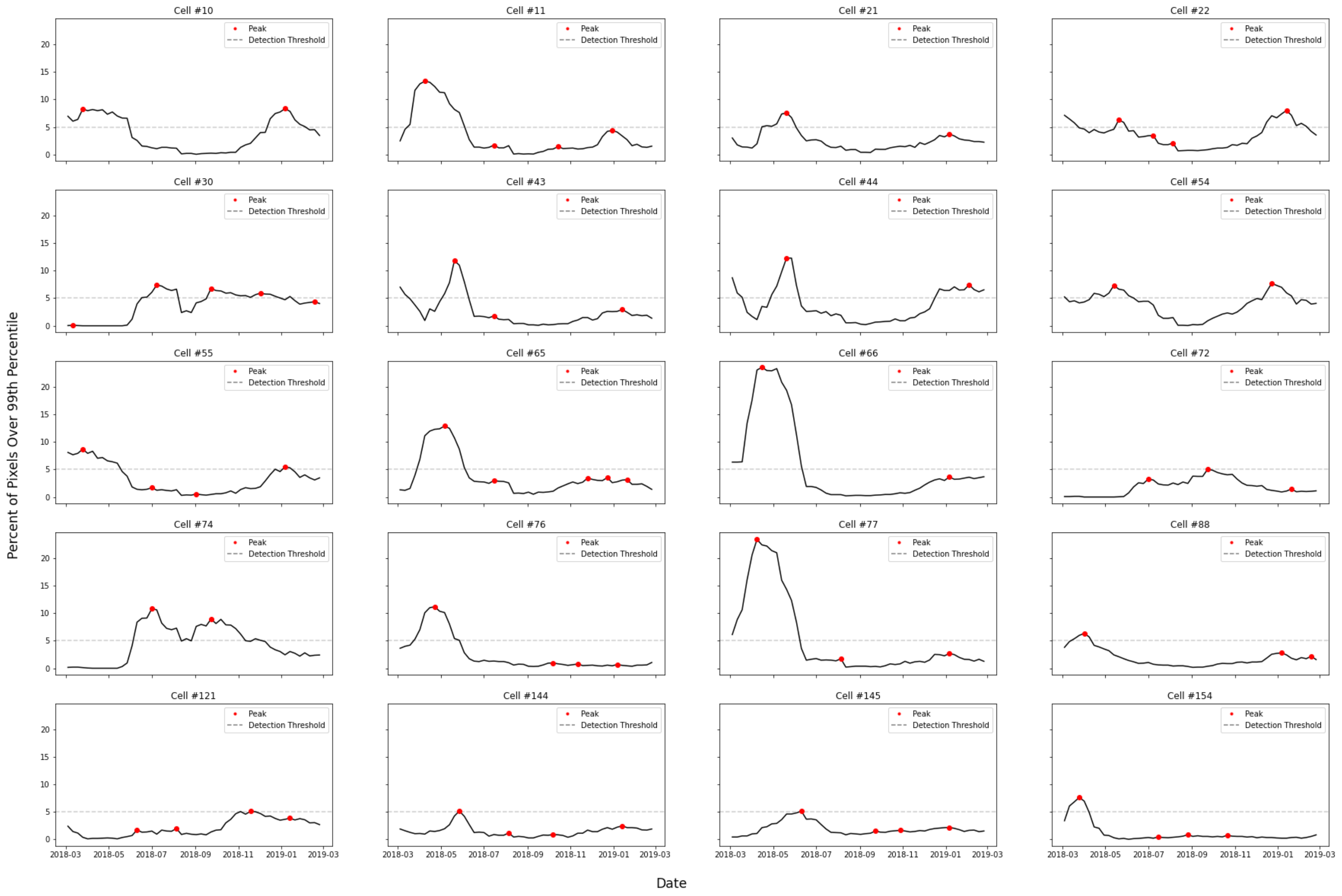

Figure 5. Time series plots for example case.

Optical Imagery

With *Get_BeforeAfter_Imagery*(), cloud-masked Sentinel-2 images are obtained for each zone with a detection. GEE is used to obtain the imagery and clip it to the extent of each zone. Each image is exported as a GeoTIFF to the user's Google Drive account.

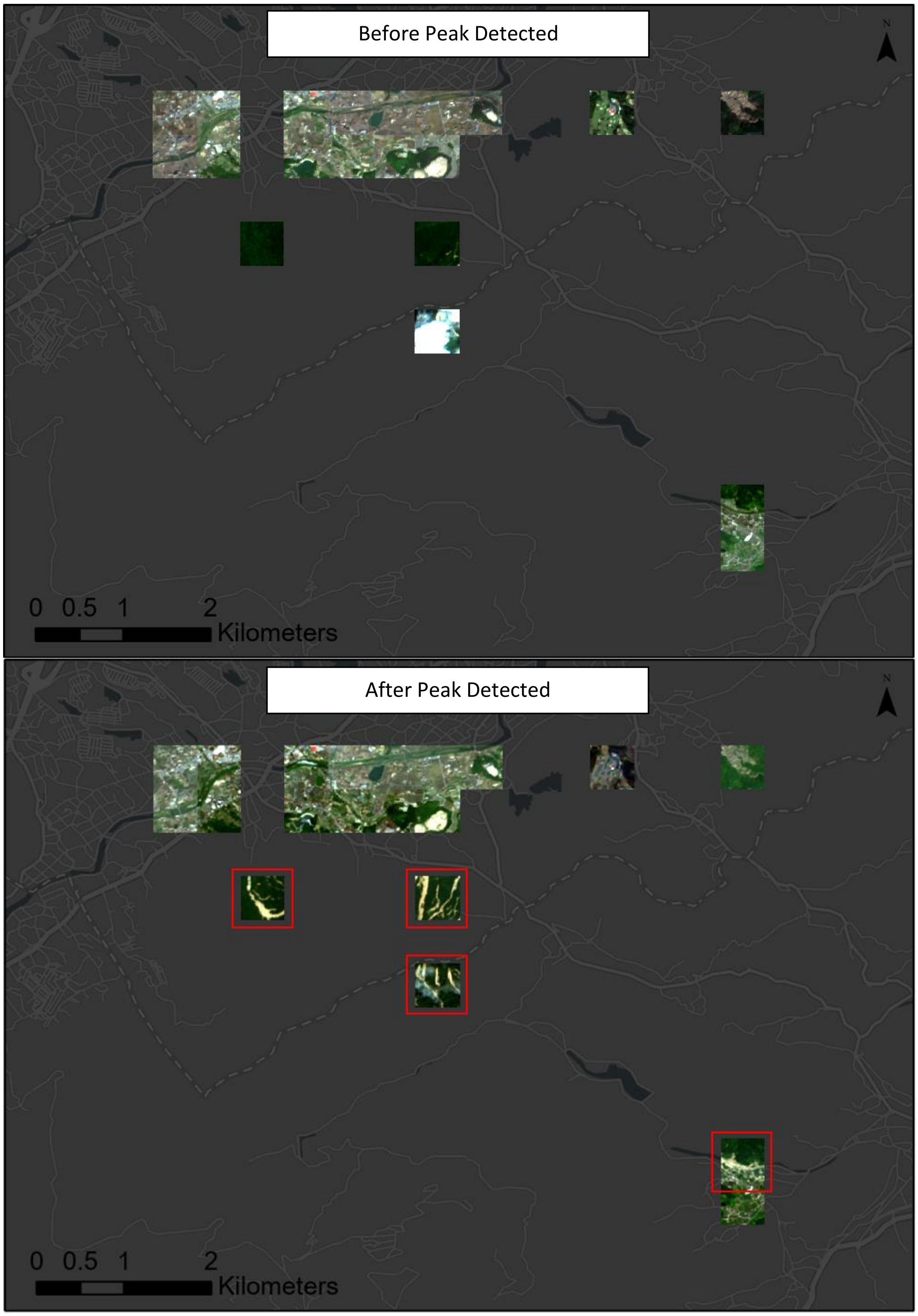

Figure 6. All pre- and post-detection images for the grid cells in the example case. Four cells contain landslides, highlighted with red boxes. One image is covered by clouds despite the cloud masking.

NDVI

Get_BeforeAfter_NDVI() is used to create a CSV of the mean Normalized Difference Vegetation Index (NDVI) of each zone before and after the date of each peak detection. NDVI is calculated from Sentinel-2 data with GEE.

Table 1. NDVI for the grid cells in the example case, pre- and post-detection. The ID of the cell, generated earlier with the *Add_ID_to_Features*() function, and date of detection, are included.

ID	Date	Pre NDVI	Post NDVI
10	3/25/2018 5:00	0.164177	0.25031
10	1/6/2019 4:00	0.196435	0.182316
11	4/8/2018 5:00	0.216988	0.240879
21	5/20/2018 5:00	0.365149	0.428984
22	5/20/2018 5:00	0.228891	0.276234
22	1/13/2019 4:00	0.150376	0.128278
30	7/8/2018 5:00	0.559331	0.575752
43	5/20/2018 5:00	0.28158	0.341352
44	5/20/2018 5:00	0.243761	0.317787
44	2/3/2019 4:00	0.158803	0.170699
54	5/13/2018 5:00	0.326755	0.381318
54	12/23/2018 4:00	0.234118	0.217529
55	3/25/2018 5:00	0.176404	0.22897
55	1/6/2019 4:00	0.212484	0.182407
65	5/6/2018 5:00	0.307716	0.401053
66	4/15/2018 5:00	0.191677	0.222427
72	9/23/2018 5:00	0.419714	0.592007
74	7/1/2018 5:00	0.769376	0.527396
76	4/22/2018 5:00	0.290595	0.373543
77	4/8/2018 5:00	0.195938	0.246079
88	4/1/2018 5:00	0.246511	0.381764
121	11/18/2018 4:00	0.485494	0.310667
144	5/27/2018 5:00	0.437316	0.499782
145	6/10/2018 5:00	0.529366	0.561476
154	3/25/2018 5:00	0.412785	0.491633

VII. Discussion

The code was able to find some landslides in the example case (Fig. 6), but not all (Fig. 7). The resolution of the Sentinel-1 data is 10 meters, so it may be difficult to resolve many of the narrow, channelized debris flows in the region. Selecting the *detection_threshold* is arbitrary: lower thresholds will include more peak detections, but there a higher chance that they are erroneous. Because the 99th percentile of the data is used as a threshold, even if there are no change events, pixels will still be classified above the threshold, causing false positives. The cells at the top of Figure 6 may be of this nature. The detections for these are likely from random noise or seasonal changes in vegetation.

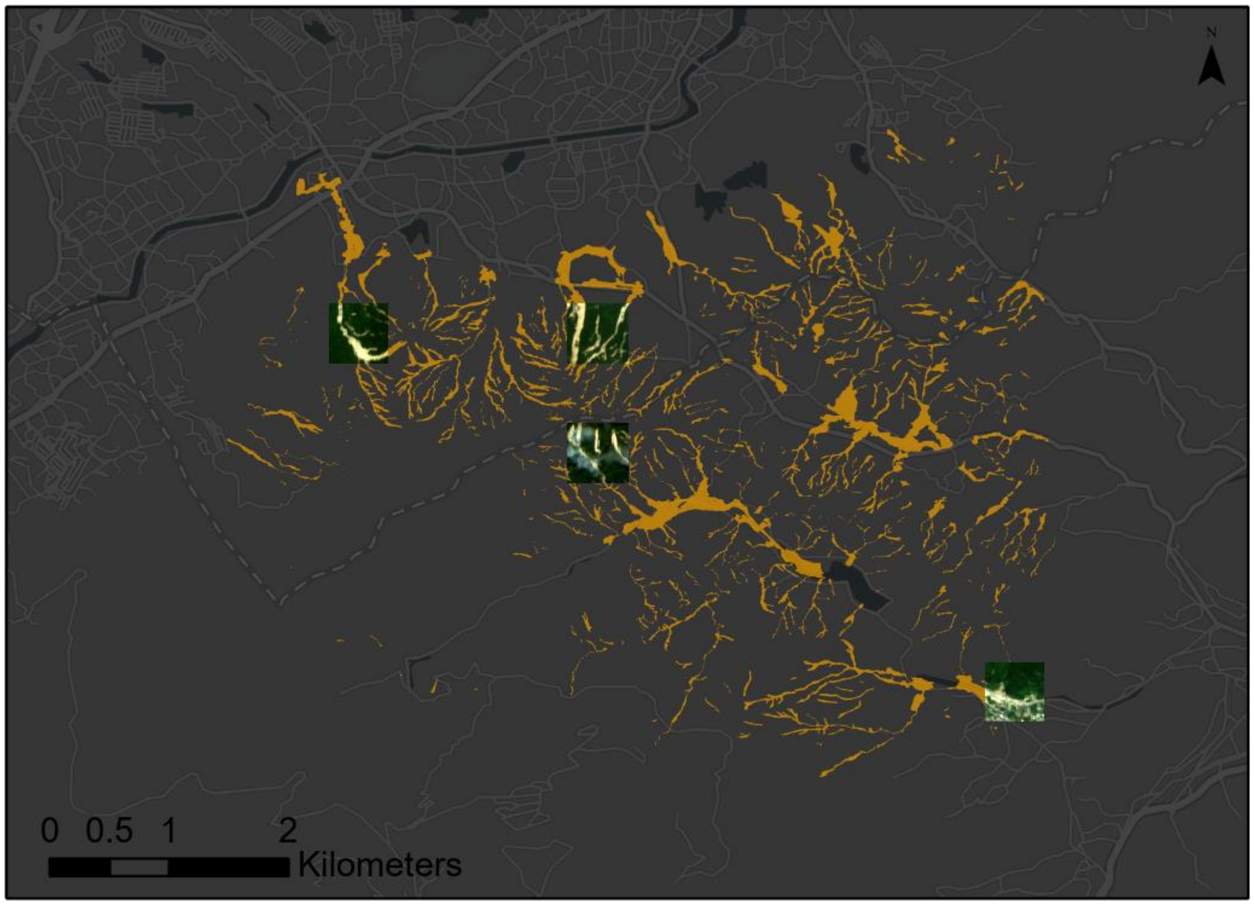

Figure 7. Zones that included landslides on top of inventory of landslides events in July 2018 (orange polygons).

The SAR Intensity Ratio Time Series Change Detection algorithm shows some potential, but with limitations. It was able to successfully catalogue dates and locations of possible changes and provide useful output data. It is possible that it would perform better at finding more extensive, wide-ranging changes, such as flooding, deforestation, or even just larger landslides. A large, homogeneous area of change should take up a large area of a grid cell, however, this has not yet been tested. The method of detection itself could be improved upon. Instead of percentage of pixels over the 99th percentile threshold, a different method such as kernel density of Getis-Ord Gi* Hotspot Analysis could be used. Such methods could account for both the statistics of the data and spatial clustering of high I_{ratio} pixels.

VIII. References

Handwerger, A. L., Jones, S. Y., Amatya, P., Kerner, H. R., Kirschbaum, D. B., and Huang, M.-H.: Strategies for landslide detection using open-access synthetic aperture radar backscatter change in Google Earth Engine, Nat. Hazards Earth Syst. Sci. Discuss. [preprint], https://doi.org/10.5194/nhess-2021-283, in review, 2021.