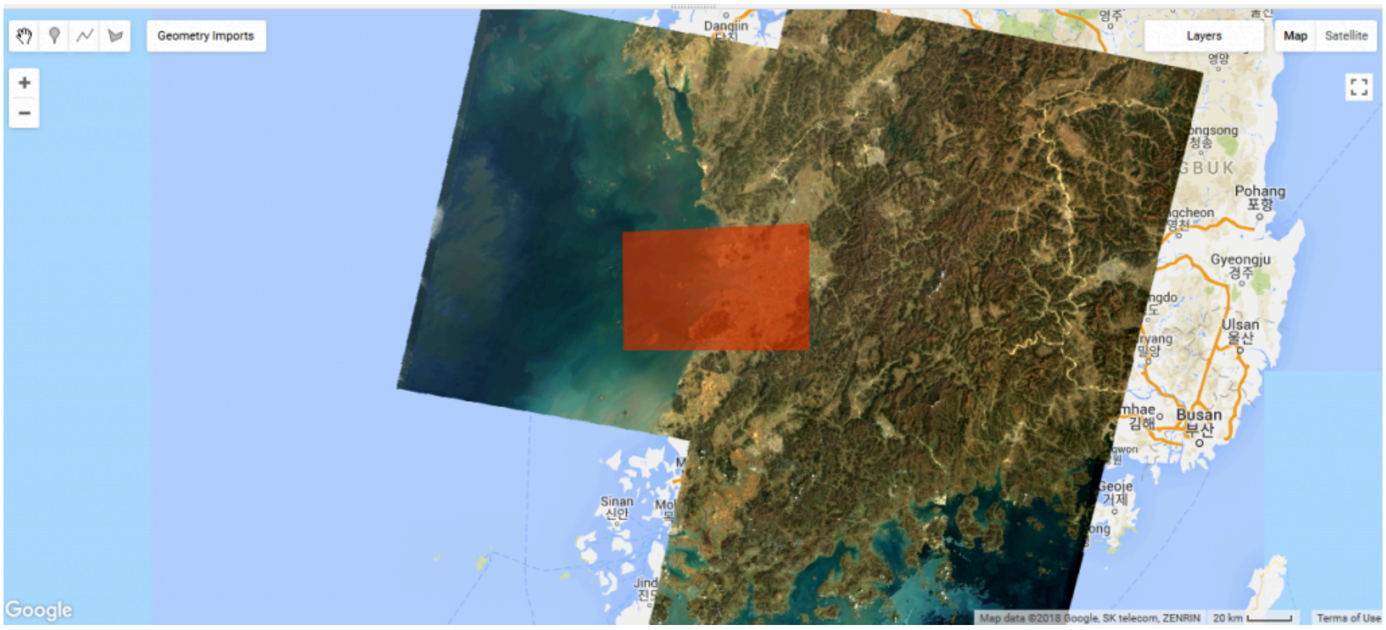
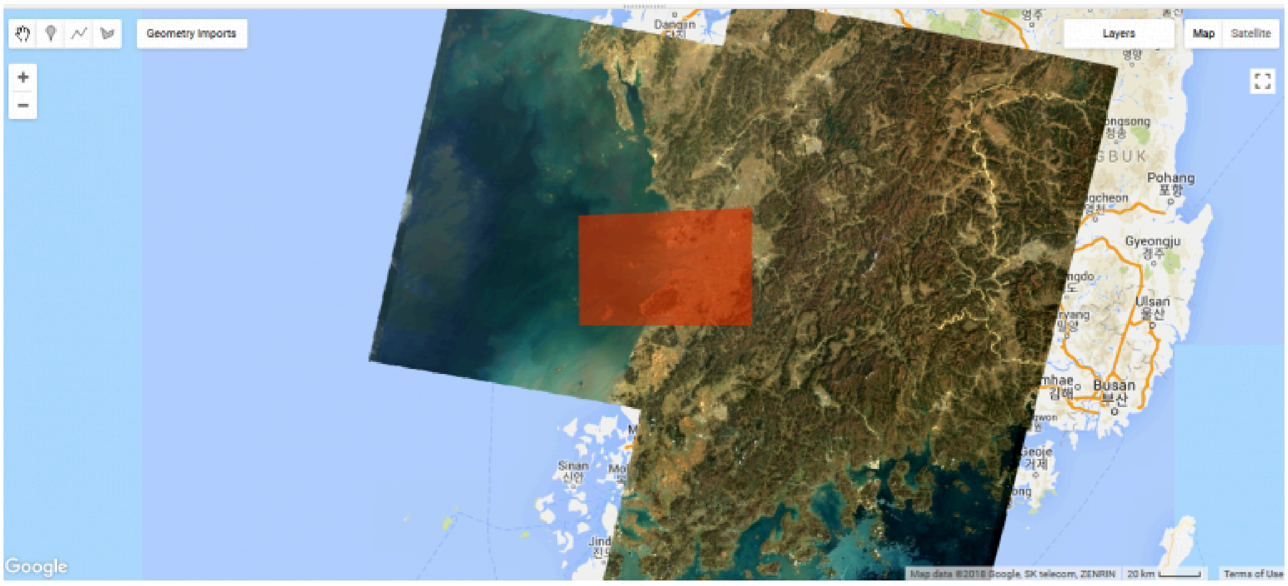
acgeospatial

Freelancer and Consultant in GIS, Earth Observation, Computer Vision, Machine Learning, Python, Data Analytics

Time series on Landsat data with Google Earth Engine

18th September 2018 / blog





Almost a year ago I wrote a post on building time-lapse imagery with Google Earth Engine. It shows how to process the Landsat 8 top of atmosphere collection into an mp4 format. It talks about choosing the path and row and filtering on clouds, selecting the bands and converting to 8 bit imagery. I did all this in Python. You can read about it here and get the code here.

This summer I wanted to expand a bit on this; I set myself the challenge to try and replicate the Google Earth Timelapse website. It is only when you attempt to recreate something that you properly realise the effort and complexity involved. Having understood the challenges I can see some of the artifacts/data problems still in the time-lapse page.

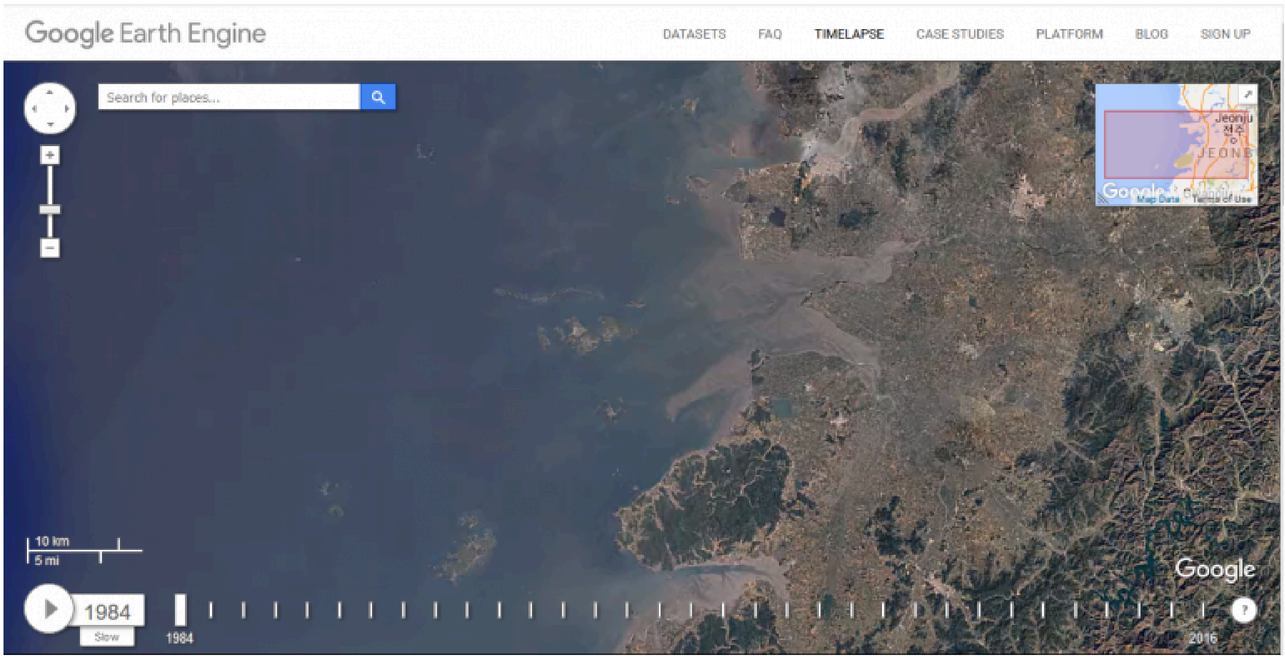
Without the free and open policy of USGS this just would not be financially viable and without the ease of access and sheer computing of Google Earth Engine the infrastructure and pre-processing would make this a pipe dream only. Before I start I want to state that this work is not for commercial gain – this is a research/education project only.

The Challenge

Put simply, to recreate Google time-lapse. That is, imagery in time series from 1984 to 2016. I am going to look at the Saemangeum land reclamation project area in Gunsan in South Korea as this area has experienced big infrastructure and ecological change during this time. In 2010 this area developed the world’s longest sea wall.

https://phys.org/news/2010-04-skorea-world-longest-seawall.html

Here is how it looked in 1984:



And here is how it looked in 2016:

What are the Earth Observation Challenges?

1. This area covers more than 1 scene / row of Landsat
2. This time period is covered by 3 Landsat sensors: Landsat 5, Landsat 7 ETM and Landsat 8. We need to combine these.
3. Landsat 7 ETM suffered a scan line failure in 2003. You can read about it here.
4. Landsat 8 has different band numbers. Band 1 in Landsat 5 and Landsat 7 ETM is blue; in Landsat 8 Band 2 is blue. Read about that here.
5. We need to decide on how to choose which image represents each year.
6. We need to try and visualise each image so they appear 'similar'.

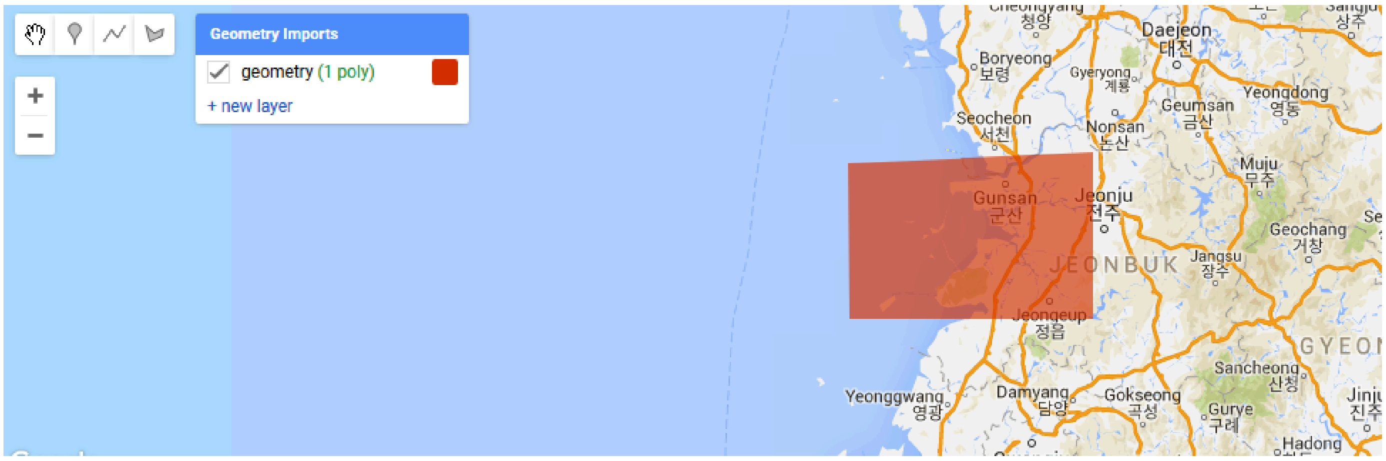
At the time of writing, the combined total number of images in the three collections in Google Earth Engine (mentioned in point 2, above) is 2197.

This data alone would have cost me ~$1.3 million at $600 a scene when I first started work.

The more I think about these challenges, especially relating to the complexity of the data, the more I appreciate the scale at which Google time-lapse was built. It was not a trivial task.

Addressing the challenges / the method

I used Earth Engine's ability to draw geometries. You could import your own via a fusion table if you prefer. I wrote about that here – see point 3.



I have called this area of interest 'geometry' in the code below. This is the default name that Earth Engine gives a new geometry. Feel free to change it to something more relevant to your area of interest.

Next up let's load in the tier 1 surface reflectance data for Landsat 5, Landsat 7 ETM and Landsat 8. All the collections are filtered where cloud cover is less than (ee.Filter.lt) 25. In the case of Landsat 8 I have set this value to 5 (as I have more data in this collection). This part selects all the scenes where total cloud cover is less than 25%. Experiment with this remembering that the lower the filter the greater the chance of no images for the year.

The time-lapse website is true-colour imagery (think of the view from an aeroplane – that is what I mean by true colour). With this in mind we need to select the bands 3,2,1 from Landsat 5 and Landsat 7 ETM and bands 4,3,2 from Landsat 8. We will do that but we must line the band numbers up first. Finally, filter the collection by the geometry bounds defined above.

```
var L5coll = ee.ImageCollection('LANDSAT/LT05/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',25))
.select(['B3', 'B2', 'B1'])
.filterBounds(geometry)

var L7coll = ee.ImageCollection('LANDSAT/LE07/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',25))
.select(['B3', 'B2', 'B1'])
.filterBounds(geometry)

var L8coll = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',5))
```

```
.filterBounds(geometry)
// don't select the bands yet!
```

Re-numbering Landsat 8 bands

In order to merge the Landsat 5 and Landsat 7 ETM collections with Landsat 8 the band numbers need to align. To do this the map function in Earth Engine is very useful. This project uses two really useful functions – map and reduce (more on these later). The map function allows the user to apply the same operation to every image in a collection (image collection or feature collection).

In our case we will rename each band which will correspond with those we have selected in both the Landsat 5 and Landsat 7 ETM collections. For example B1 becomes B0, B2 becomes B1 and so on. To do this we use the map function to return image.rename across all the images in the collection.

```
map(function(image){
  return image.rename(['B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7',
'B8', 'B9', 'B10', 'B11']);
})
```

What is really nice about Earth Engine is that you can build this into our initial filtering. So the Landsat 8 collection selection now looks like this:

```
var L8coll = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',5))
.filterBounds(geometry)
.map(function(image){
  return image.rename(['B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7',
'B8', 'B9', 'B10', 'B11']);
})
.select(['B3', 'B2', 'B1']);
```

The problem with Landsat 7 ETM

Here is the problem with Landsat 7 images. From 2003 onward they look like this:

I am indebted to the solution posted in this Earth Engine sample:
https://code.earthengine.google.com/16e86cfbd845a7b583100cb8c27348e9

Which is based on this method:

https://landsat.usgs.gov/gap-filling-landsat-7-slc-single-scenes-using-erdas-imagine-TM

If you read the above page you can see that there are five suggested solutions for different software packages. In Earth Engine we can use the map function again to apply a focal mean and then a blend to each image in the collection using this code:

```
.map(function(image){
  var filled1a = image.focal_mean(1, 'square', 'pixels', 2)
  return filled1a.blend(image);
})
```

focal_mean(*radius*, *kernelType*, *units*, *iterations*, *kernel*)

It is worth adjusting the *radius* and *iteration* parameters in the image.focal_mean to get the best result. The larger these values the greater the blurring on the data. Note that by default the *kernel* parameter is null; if you parse anything in here you will override the *kernelType* parameter. I found the above worked reasonably well for what I was trying to achieve.

filled1a.blend(image)

This part will blend the filled1a (focal_mean result) with the initial input image. In the end the resulting image will look like this:

This seems like a good improvement. Add this piece of code into the above filtered collection. The top of the script should now look like this:

```
var L5coll = ee.ImageCollection('LANDSAT/LT05/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',25))
.select(['B3', 'B2', 'B1'])
.filterBounds(geometry)

var L7coll = ee.ImageCollection('LANDSAT/LE07/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',25))
.select(['B3', 'B2', 'B1'])
.filterBounds(geometry)
.map(function(image){
  var filled1a = image.focal_mean(2, 'square', 'pixels', 1)
  return filled1a.blend(image);
})

var L8coll = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
.filter(ee.Filter.lt('CLOUD_COVER',5))
.filterBounds(geometry)
.map(function(image){
  return image.rename(['B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7',
'B8', 'B9', 'B10', 'B11']);
})
.select(['B3', 'B2', 'B1']);
```

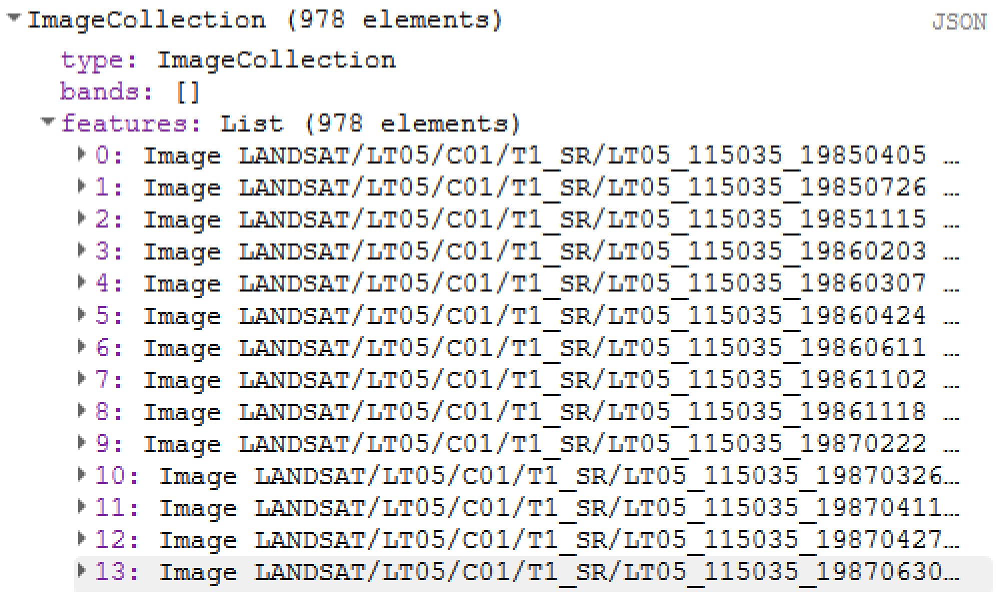
We are now ready to amalgamate these three collections together.

Merging collections

Pushing these three collections now that we have them in the best condition is done using the merge command in Earth Engine.

```
var collection_merge =
ee.ImageCollection(L5coll.merge(L7coll.merge(L8coll)));
print (collection_merge)
```

Here a new variable is created called collection_merge and L5coll is merged with L7coll which has already been merged with L8coll. The trick here is to watch the brackets. The second line prints the collection_merge to the console. Now I am down to 1084 images in my collection from the initial input of 2197 (for the AOI). This is down to filtering for clouds. Unless I inspect all 978 images I don't know what my coverage is. I would hope to have at least one image for each year.



```
▾ImageCollection (978 elements)                          JSON
   type: ImageCollection
   bands: []
  ▾features: List (978 elements)
    ▸0: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19850405 …
    ▸1: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19850726 …
    ▸2: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19851115 …
    ▸3: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19860203 …
    ▸4: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19860307 …
    ▸5: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19860424 …
    ▸6: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19860611 …
    ▸7: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19861102 …
    ▸8: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19861118 …
    ▸9: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19870222 …
    ▸10: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19870326…
    ▸11: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19870411…
    ▸12: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19870427…
    ▸13: Image LANDSAT/LT05/C01/T1_SR/LT05_115035_19870630…
```

Building a collection containing one image per year

To replicate the timelapse we need to assign one image to each year from 1984 – 2018. To start, build a list that we can use to iterate containing these dates. To do this use the ee.sequence.list command and parse the start number of the list and the end number of the list.

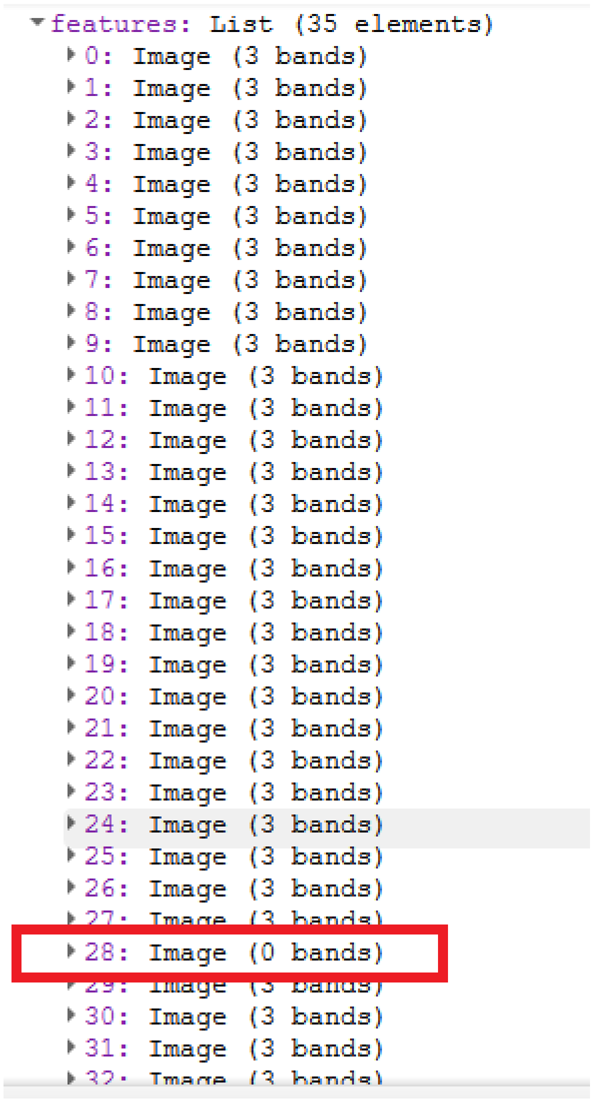
```
var years = ee.List.sequence(1984, 2018)
print (years)
```

Next we need to assign a new variable as a new collection to contain each of our images. In this case I have called it collectYear. I parse the list of year values and then use the map function to create an image for each year. I need to define the start date and end date and then use these to filter my image collection input (collection_merge – created in the previous section). Finally when I have all the images for a year I call the median reducer function. This converts these images to one median image on a pixel by pixel basis, which I call collectYear.

```
var collectYear = ee.ImageCollection(years
  .map(function(y) {
    var start = ee.Date.fromYMD(y, 1, 1)
    var end = start.advance(12, 'month');
    return collection_merge.filterDate(start,
end).reduce(ee.Reducer.median()

}))
```

I now have an image collection called collectYear with 35 elements. Or in laymans terms 35 images. Except I don’t, and here is why:



```
▾features: List (35 elements)
  ▸0: Image (3 bands)
  ▸1: Image (3 bands)
  ▸2: Image (3 bands)
  ▸3: Image (3 bands)
  ▸4: Image (3 bands)
  ▸5: Image (3 bands)
  ▸6: Image (3 bands)
  ▸7: Image (3 bands)
  ▸8: Image (3 bands)
  ▸9: Image (3 bands)
  ▸10: Image (3 bands)
  ▸11: Image (3 bands)
  ▸12: Image (3 bands)
  ▸13: Image (3 bands)
  ▸14: Image (3 bands)
  ▸15: Image (3 bands)
  ▸16: Image (3 bands)
  ▸17: Image (3 bands)
  ▸18: Image (3 bands)
  ▸19: Image (3 bands)
  ▸20: Image (3 bands)
  ▸21: Image (3 bands)
  ▸22: Image (3 bands)
  ▸23: Image (3 bands)
  ▸24: Image (3 bands)
  ▸25: Image (3 bands)
  ▸26: Image (3 bands)
  ▸27: Image (3 bands)
  ▸28: Image (0 bands)
  ▸29: Image (3 bands)
  ▸30: Image (3 bands)
  ▸31: Image (3 bands)
  ▸32: Image (3 bands)
```

My 28th image, the image from 2012, has 0 bands. When I create my video this will give me a blank image. I tried changing the cloud cover parameters to try and get some more images but I couldn't get this to work. In the end I had to remove the blank image from the collection. Another challenge!

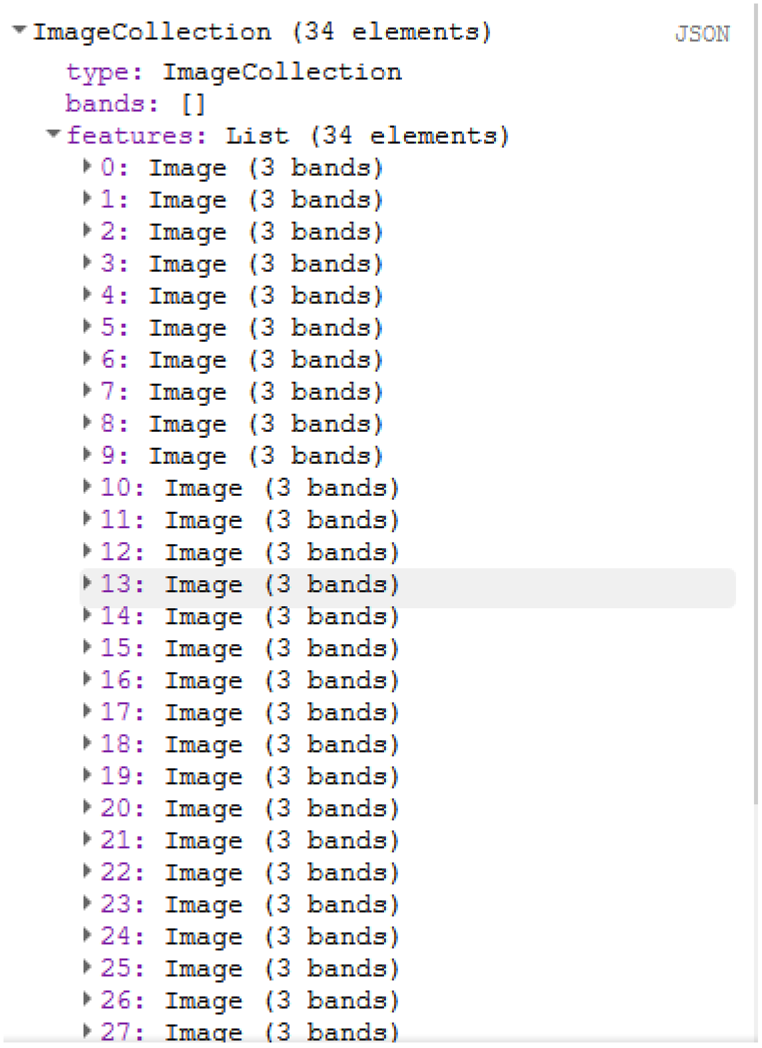
Removing nulls / nodata or empty images from a collection

I know in this case my empty image is element 28, so I could just remove that one. However if I don't know which one it is, say I am trying to build a monthly timeseries, I don't want to inspect larger image collections. So another function is needed.

This time I am going to set up another variable called nullimages and again use the map function (you can see how useful this function is) to count the number of bands and then filter on this number. For each image in the collection I am going to count the number of bands (image.bandName().length()) and if that count is equal to three I will assign the image back into nullimages.

```
var nullimages = collectYear
    .map(function(image) {
      return image.set('count', image.bandNames().length())
    })
    .filter(ee.Filter.eq('count', 3))
print(nullimages)
```

By printing this result to the command window I can see that I now have my 34 images (no image for 2012 sadly):



```
▾ImageCollection (34 elements)                JSON
   type: ImageCollection
   bands: []
  ▾features: List (34 elements)
    ▸0: Image (3 bands)
    ▸1: Image (3 bands)
    ▸2: Image (3 bands)
    ▸3: Image (3 bands)
    ▸4: Image (3 bands)
    ▸5: Image (3 bands)
    ▸6: Image (3 bands)
    ▸7: Image (3 bands)
    ▸8: Image (3 bands)
    ▸9: Image (3 bands)
    ▸10: Image (3 bands)
    ▸11: Image (3 bands)
    ▸12: Image (3 bands)
    ▸13: Image (3 bands)
    ▸14: Image (3 bands)
    ▸15: Image (3 bands)
    ▸16: Image (3 bands)
    ▸17: Image (3 bands)
    ▸18: Image (3 bands)
    ▸19: Image (3 bands)
    ▸20: Image (3 bands)
    ▸21: Image (3 bands)
    ▸22: Image (3 bands)
    ▸23: Image (3 bands)
    ▸24: Image (3 bands)
    ▸25: Image (3 bands)
    ▸26: Image (3 bands)
    ▸27: Image (3 bands)
```

Great. So can I finally export to video for my time series? Nope, not yet.

Visualise the images

With all this data preparation it is perhaps easy to forget that we need to view the images. If I left this collection without stretching it this would be my first image from 1984.

This is pretty disappointing for all that data preparation work!

Earth Engine lets you look at the first image in the collection and it is with this that you can inspect values and adjust the stretch. Assign the first image in the collection to a new variable called medianFirst. Call Map.addLayer and parse in the new image, define the bands you want to display (we only have the three) set the min and max stretches and give the layer a name.

```
var medianFirst = collection.first()
Map.addLayer(medianFirst, {bands: ['B3', 'B2', 'B1'], min: 300, max:
1800}, 'first image');
```

The resulting image will look similar to below:

Almost there. If these visualisation parameters work then we need to apply these across our nullimages collection. Again we use the map function. The syntax is similar to what we have seen before. The only difference this time is that we return image.visualize and the stretch to our new collection, which I have called finalCollection.

```
var finalCollection = nullimages.map(function(image){
  return image.visualize({bands: ['B3', 'B2', 'B1'], min: 300, max:
1800});
})
```

Now we have an image collection of 34 nicely stretched images from 1984 to 2018 (missing 2012). All that is left to do is export them to video.

Export to video

The final part of this project was to save it as an mp4 file. Earth Engine is brilliantly set up for that. Call the Export.video.toDrive and parse in the collection (finalCollection) a description ('yearly'), the dimensions (1080 – we want HD!), framesPerSecond (1) and the region we are interested in (geometry – we created this right at the start):

```
Export.video.toDrive({
  collection: finalCollection,
  description: 'yearly',
  dimensions: 1080,
  framesPerSecond: 1,
  region: geometry
});
```

Click on run. Run the Task. Eventually this will appear in your Google Drive. You should be able to see the construction of the world's longest sea wall.

Something like this:

The code

The full code is here

Recap

The aim here was to try and replicate Google timelapse over a small area. I wanted to use data from 1984 to 2018. These are the steps relating to the challenges that I took:

- Re-numbering Landsat 8 bands
- The problem with Landsat 7 ETM
- Merging collections
- Building a collection containing one image per year
- Removing nulls / nodata or empty images from a collection
- Visualise the images
- Export to video

The end?

There are still refinements I'd like to make to these images. When I find time I may revisit this project. Balancing the images better and improving the mosaicing would all help.

Only by doing projects like this do I fully appreciate the effort that has gone into creating Google Timelapse. Also, working on projects like this highlights the needs / challenges that are required to build a data refinery as Descartes Labs are doing.

Perhaps to the layman Earth Observation appears to be just beautiful images. In reality, like any data based project, perhaps 90% of the work is getting the data in the structure you can work with.

I am indebted to the Google Earth Engine Developers Forums for others sharing code that has helped me build this project.

I am a freelancer able to help you with your projects. I offer consultancy, training and writing. I'd be delighted to hear from you. Please check out the books I have written on QGIS 3.4

https://www.packtpub.com/application-development/learn-qgis-fourth-edition

https://www.packtpub.com/application-development/qgis-quick-start-guide

---

I have grouped all my previous blogs (technical stuff / tutorials / opinions / ideas) at http://gis.acgeospatial.co.uk.

Feel free to connect or follow me; I am always keen to talk about Earth Observation.

I am @map_andrew on twitter

Tags: EARTH OBSERVATION GOOGLE EARTH ENGINE JAVASCRIPT TEMPORAL TIMELAPSE

Contact

About

Podcast

Training

Clients

Contact me

Book

Python Course info 2024

Contact me

Contact me

Search for...

Recent Comments

Archives

September 2025

July 2025

June 2025

May 2025

February 2025

August 2024

May 2024

March 2024

February 2024

July 2023

February 2023

November 2022

August 2022

July 2022

June 2022

May 2022

April 2022

December 2021

July 2021

January 2021

December 2020

October 2020

July 2020

June 2020

May 2020

April 2020

January 2020

December 2019

October 2019

August 2019

July 2019

June 2019

May 2019

April 2019

March 2019

February 2019

Neve | Powered by WordPress