Deep Learning with Satellite Data

Max Langenkamp Feb 2, 2019 · 6 min read

“The rockets and the satellites, spaceships that we’re creating now, we’re pollinating the universe.” -Neil Young

Overview— Satellite Data—Data Collection— Model — Results

Overview

While at the University of Sannio in Benevento, Italy this January, my friend Tuomas Oikarinen and I created a (semi-automated) pipeline for downloading publicly available images, and trained a 3-D Convolutional Neural Network on the data. Ultimately, our model achieves a balanced accuracy of around 0.65 on Sentinel-2 optical satellite imagery. I will go into more detail regarding the results (and why this model might actually be useful). Under the guidance of Silvia Ullo and with some help from her graduate students Maria Pia Del Rosso, Alejandro Sebastianelli, and Federico Picarillo, we ended up submitting a paper[1] to a remote sensing conference!

Here is an immediate link to the Github repository if you want to just dive straight into the code.

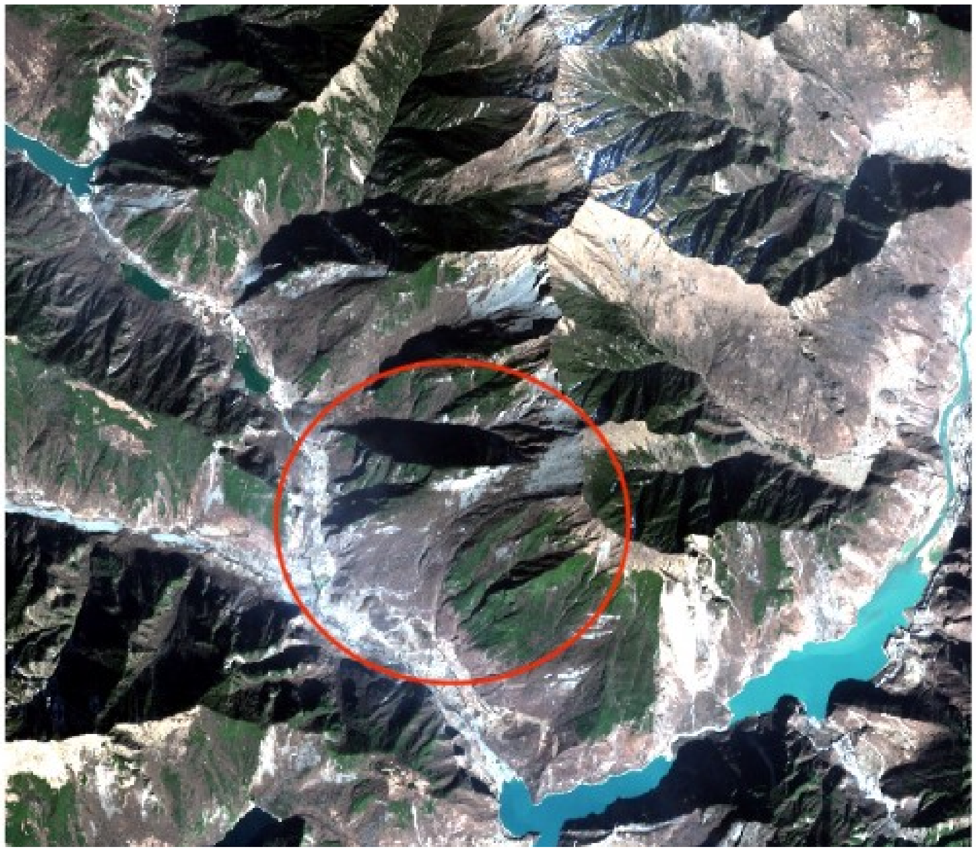
I will describe in a relatively technical manner (code included) how to quickly download some satellite images from the google earth engine and then use them to train a 3-

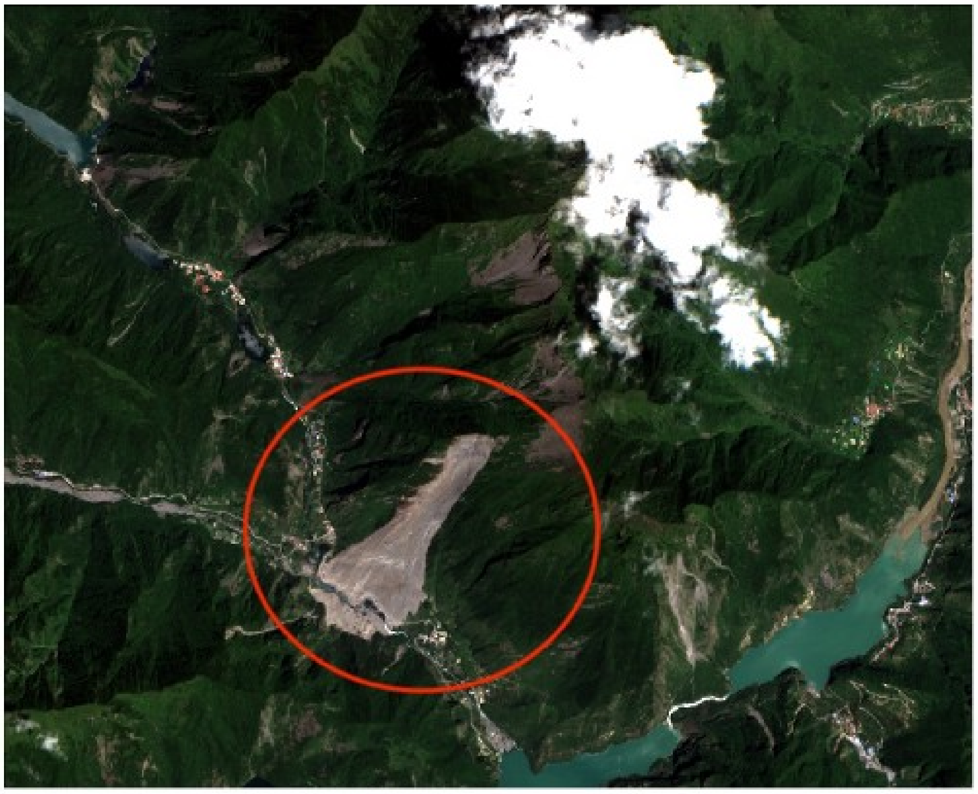
Specifically, we are using pairs of images (one of a landslide, one of the same geographic region but no landslide) as the training examples. However, this should be helpful for any cases that involve using public satellite data for image models.

Satellite Data

Before diving into the data collection, it is worthwhile to get a general understanding of the types of satellite images that we will be using. The most easily processed and widely available up-to-date satellite data is from the Copernicus Satellite program. The satellites that make up the Copernicus programme are called the Sentinel satellites. For the purposes of looking at changes on the ground, Sentinel-1 and Sentinel-2 satellites are most useful. Sentinel-1 provides radar imaging, and a higher level of resolution than Sentinel-2, but is also more difficult to use. You have to process the radar images more than the Sentinel-2 images, which are fit for use basically right after you crop and download them. We ended up using Sentinel-2 images to train our model.

Below are the best examples we could find of satellite images before and after a landslide.



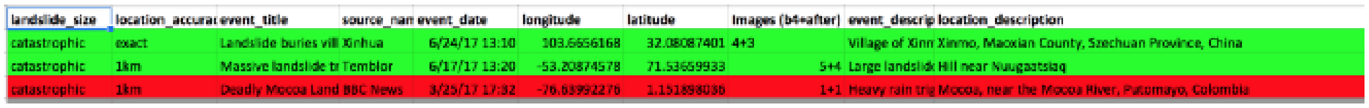


After the landslide

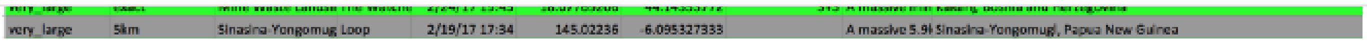
The vast majority of the other image pairs we had were not so clear cut. Even for this example, you can see that the cloud coverage, lighting conditions, and vegetation color are very different. What’s more, all the different approaches we could find to automatically process landslides used features such as slope, land cover, aspect, topographic wetness index etc fed into SVMs (see here and here for papers on these approaches). The question we are interested in here is *can a model learn anything at all about hazard detection from a small number of optical images?*

Data Collection

The labels for us came from the NASA Global Landslide Catalog, which documents landslides from the early 2000s to late 2017. Here’s a screenshot of the edited version of what we had to work with:



landslide_size	location_accurac	event_title	source_nam	event_date	longitude	latitude	Images (b4+after)	event_descrip	location_description
catastrophic	exact	Landslide buries vill	Xinhua	6/24/17 13:10	103.6656168	32.08087401	4+3	Village of Xinn	Xinmo, Maoxian County, Szechuan Province, China
catastrophic	1km	Massive landslide tr	Temblor	6/17/17 13:20	-53.20874578	71.53659933	5+4	Large landslid	Hill near Nuugaatsiaq
catastrophic	1km	Deadly Mocoa Land	BBC News	3/25/17 17:32	-76.63992276	1.151898036	1+1	Heavy rain trig	Mocoa, near the Mocoa River, Putomayo, Colombia



very_large	exact	[illegible]	2/24/17 13:45	[illegible]	[illegible]	315	A massive [illegible]	[illegible], Bosnia and Herzegovina
very_large	5km	Sinasina-Yongomug Loop	2/19/17 17:34	145.02236	-6.095327333		A massive 5.9	Sinasina-Yongomug, Papua New Guinea

Processed Landslide Catalog

Green means good data, red means poor data, grey means the location accuracy was too low to guarantee that the cropping would capture it.

The first and most difficult part of this whole project was the collection of clean examples of the labeled data. After spending much time messing around with ESA Copernicus SciHub (the European Space Agency's online user interface for downloading satellite images), we realised that it would be quite difficult to automate the download process. First issue was the fact that you could not manually enter the coordinates anywhere on the website, and also there was the fact that you had to download the entire satellite 'image', which covered an area that was about a third the size of Africa for a given date. This ended up being over 7 Gb for each example; far too much for the storage capacity of our laptops. To add insult to injury, you had to spend half an hour preprocessing the image in SNAP, a geographic data processing software (incidentally also very difficult to automate).

Instead, we turned to Google Earth Engine, which could filter by date, crop, display cloud density and provide download links all at the click of a button! It did take us a while to figure out how to do this because of the lack of examples/sparse documentation on the earth engine. The starting guide did give an accessible overview of the functions you could expect of their Javascript API, so I still recommend starting there.

Quick note: you will have to sign up to become a google earth engine developer to access the console that we used. Fortunately, this process is very easy and you should be approved almost immediately. Follow the instructions to sign up here.

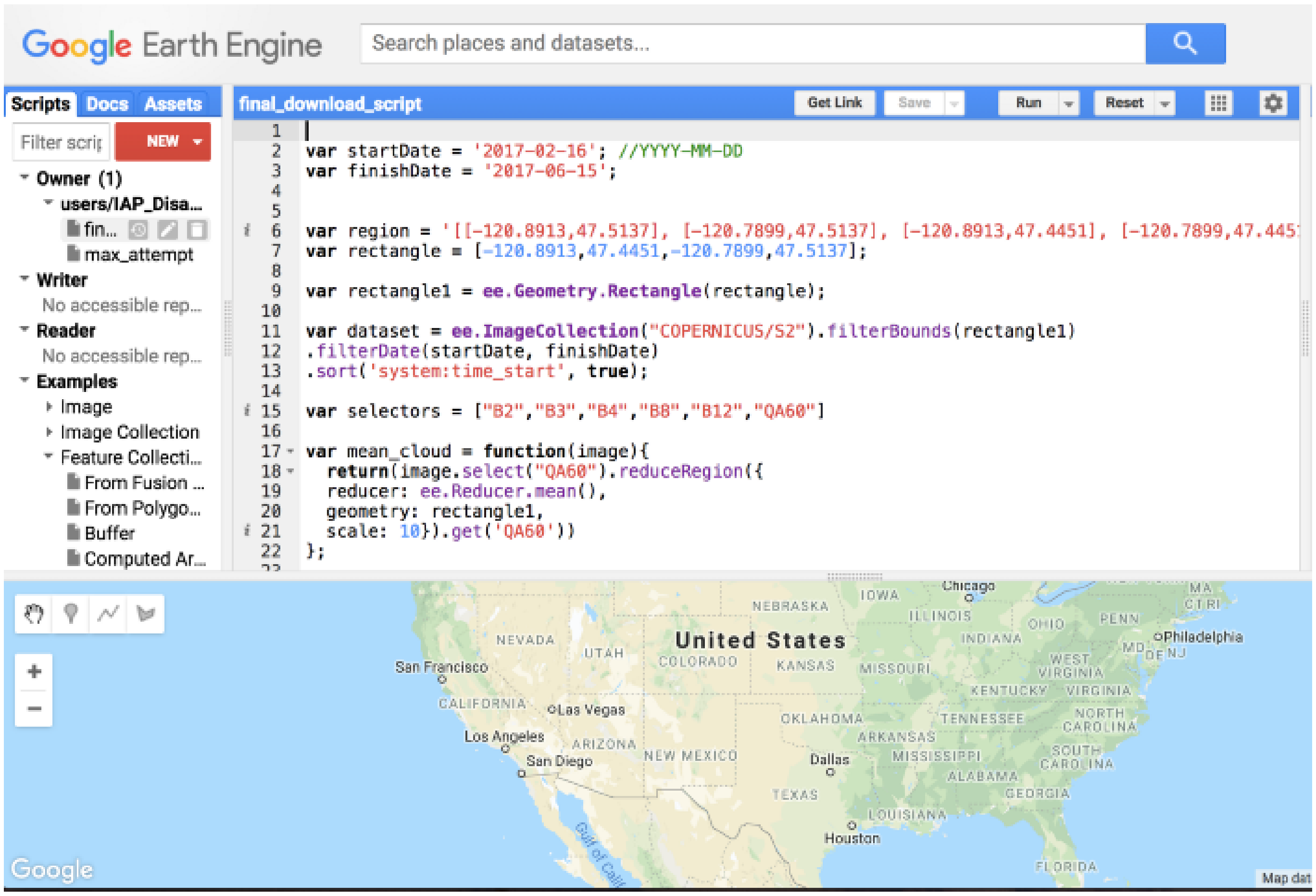
How our final (semi) automated pipeline ended up working was as follows:

1. Enter coordinates into `cropping_coordinates.py` to calculate the coordinates of a 10 km square.

the date range to one that you'd like.

3. Download the images with less than 100 for the mean cloud density given by clicking the link printed in the console output.

Voilá! You have your preprocessed dataset.

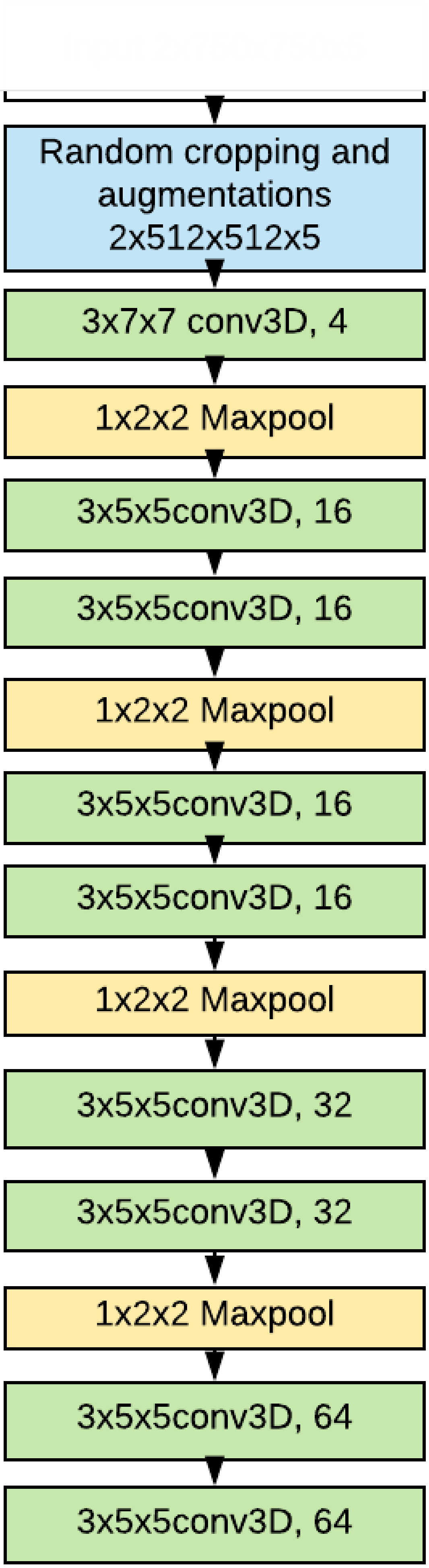


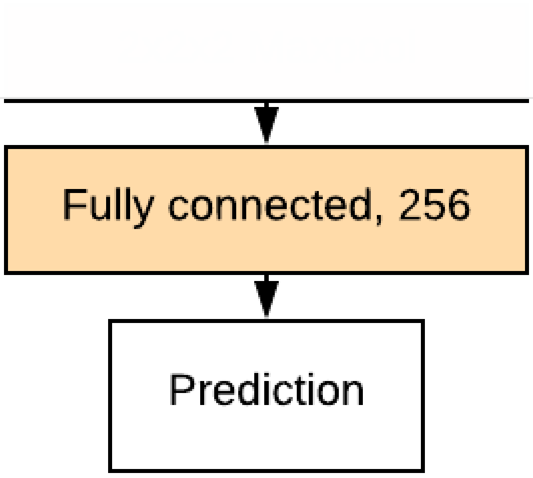
```
var startDate = '2017-02-16'; //YYYY-MM-DD
var finishDate = '2017-06-15';


var region = '[[-120.8913,47.5137], [-120.7899,47.5137], [-120.8913,47.4451], [-120.7899,47.445
var rectangle = [-120.8913,47.4451,-120.7899,47.5137];

var rectangle1 = ee.Geometry.Rectangle(rectangle);

var dataset = ee.ImageCollection("COPERNICUS/S2").filterBounds(rectangle1)
.filterDate(startDate, finishDate)
.sort('system:time_start', true);

var selectors = ["B2","B3","B4","B8","B12","QA60"]

var mean_cloud = function(image){
  return(image.select("QA60").reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: rectangle1,
  scale: 10}).get('QA60'))
};
```

Google Earth Engine Dev Console

(A quick technical note: for our model, we chose to use 5 out of 12 available bands for each image. You can change this in javascript code. Each band captures a different spectrum of wavelength, and you can read more about it on the google earth engine Sentinel-2 dataset description).

Model


Random cropping and augmentations 2x512x512x5
3x7x7 conv3D, 4
1x2x2 Maxpool
3x5x5conv3D, 16
3x5x5conv3D, 16
1x2x2 Maxpool
3x5x5conv3D, 16
3x5x5conv3D, 16
1x2x2 Maxpool
3x5x5conv3D, 32
3x5x5conv3D, 32
1x2x2 Maxpool
3x5x5conv3D, 64
3x5x5conv3D, 64



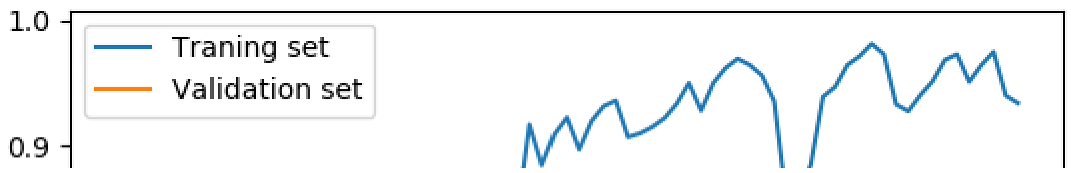
Our CNN

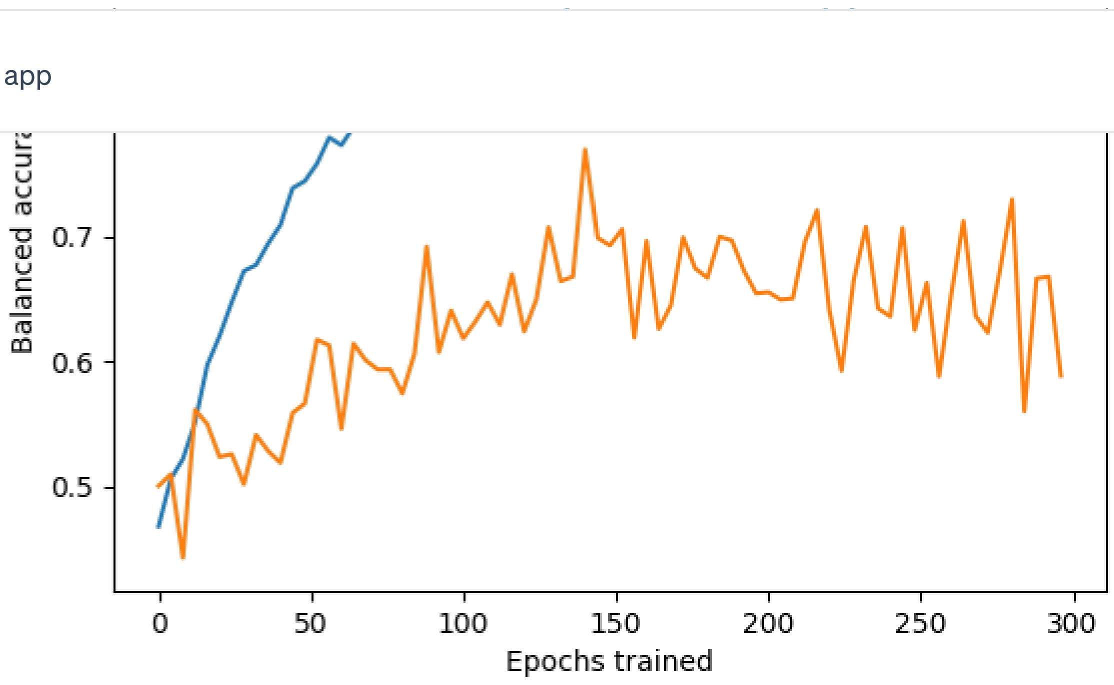
For our model, we defined an 8 layer CNN in Tensorflow (see code here). It wasn't anything really special; we began with 5 layers and noticed that performance improved slightly as we added more layers. Contrary to what you would expect with regular Feedforward Networks (see this nice StackExchange discussion for some background theory), there is no consensus as to how deep to make your CNNs. You just have to try it.

Because we ended up working with pairs of images that were on the order of ~120 and not 12,000 as would be desired, we added a couple augmentations to make our model more robust. First we randomly cropped the images using the `tf.random_crop` function, then we would randomly flip and rotate a random portion of the images. We also tried adding random Gaussian noise to the images but that worsened performance and really slowed the training process down.

Results

We used 5-fold cross validation so we could use more training data and still have a good sense of how our model does on the evaluation set.





Graph of Accuracy over Epochs

It *is* possible to learn something from optical images alone, without access to a bunch of private land-features! Clearly, however, with a validation accuracy that settles to around 0.65, the signal is weak. The most obvious source of improvement would come from more training examples. While we did exhaust the NASA Open Landslide Catalog of good examples, there should be many national catalogs of landslides along with their date, size and coordinates.

I hope you found this guide useful, or at least mildly interesting. I certainly enjoyed researching it and explaining the motivations behind our research.

Tuomas and I (third and second from the right) at the European Space Agency

Ciao!

[1] S. L. Ullo, M.S. Langenkamp, T.P. Oikarinen, M.P. Del Rosso, A. Sebastianelli, F. Piccirillo, S. Sica, “Landslide Geohazard Assessment with Convolutional Neural Networks Using Sentinel-2 Imagery Data,” *IGARSS 2019–2019 IEEE International Geoscience and Remote Sensing Symposium*, Yokohama, Japan, 2019, pp. 9646–9649.

Towards Data Science Machine Learning