Exploring the data

Training dataset

The training data set contains **154** unique airstrips located in Peru. The database format includes a unique "id", a "length" (in meters), and a geometry shape (a `LINESTRING` with 2 or more points). The coordinates for geometry points correspond to (longitude latitude).

```
| index | id | yr | largo | Activo | geometry |
| 0 | 1 | 2023 | 968.918 | 0 | LINESTRING (-70.08928656863503 -13.129844039931504,...
| 1 | 2 | 2022 | 1105.49 | 0 | LINESTRING (-69.16744237255283 -13.620679758207931,...
| 2 | 3 | 2015 | 985.018 | 0 | LINESTRING (-69.14224792429687 -13.694510447986984,...
```

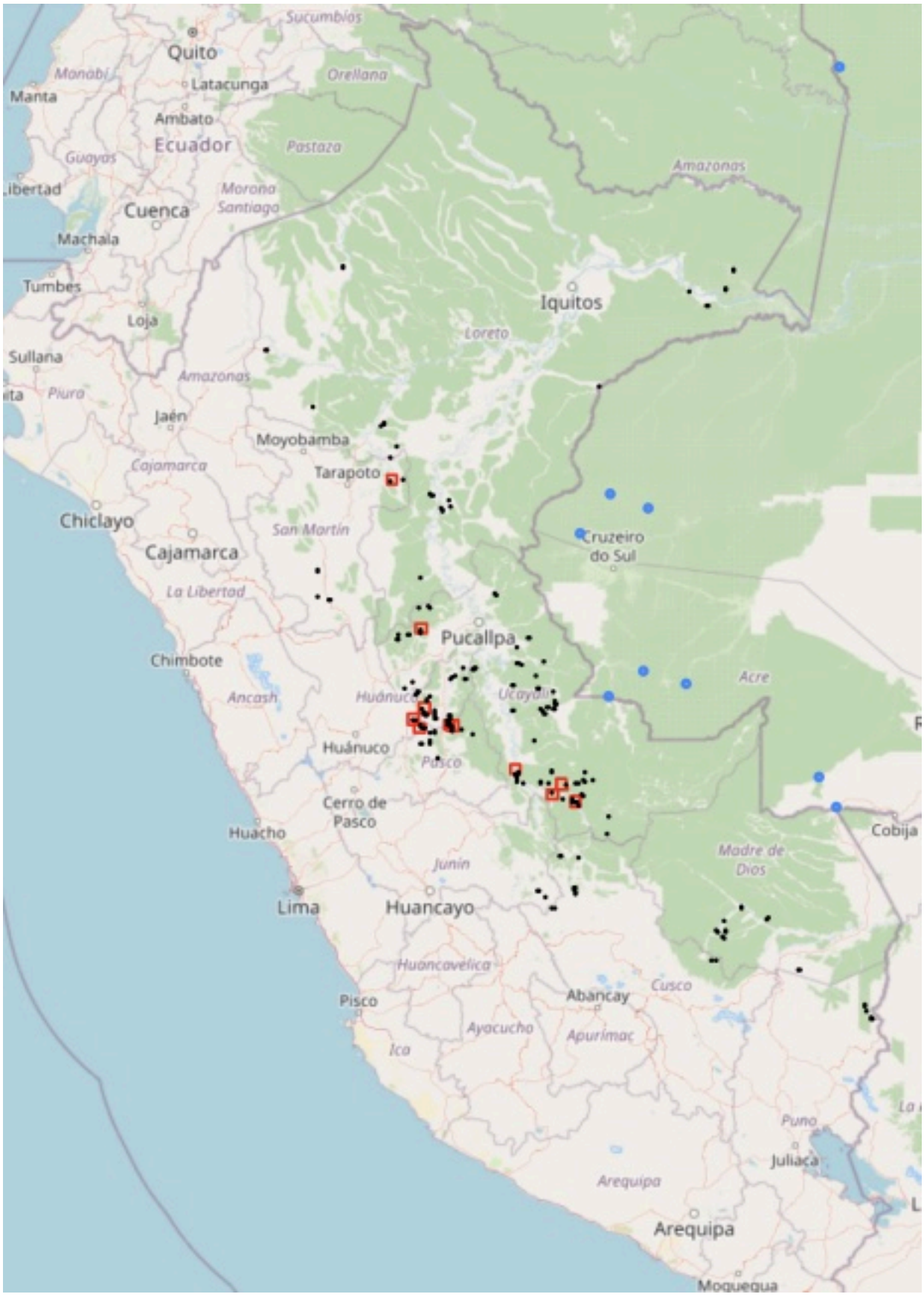
The locations of the airstrips are overlaid on the map below. The short black lines correspond to the airstrips and the red rectangles correspond to the "test" areas of interest that will be used for the competition. The blue spots correspond to entries from a separate database of airstrips that could be used to increase the size of the training set[1].

Note that some airstrips are very close to each other. This requires special attention when generating the tiles for the training set. More than one airstrip may have to be masked when creating the training data set (see the second picture with 3 airstrips). For simplicity, we skip those tricky samples due to time constraints.

For a few airstrips, we couldn't get 100% cloud-free mosaic files for the year of detection. Instead of filling or padding the gaps in the images produced by the cloud filter, we dropped those airstrips from the training set.

All airstrip images were inspected and comments were collected in an Excel spreadsheet which is available in the GitHub repository.

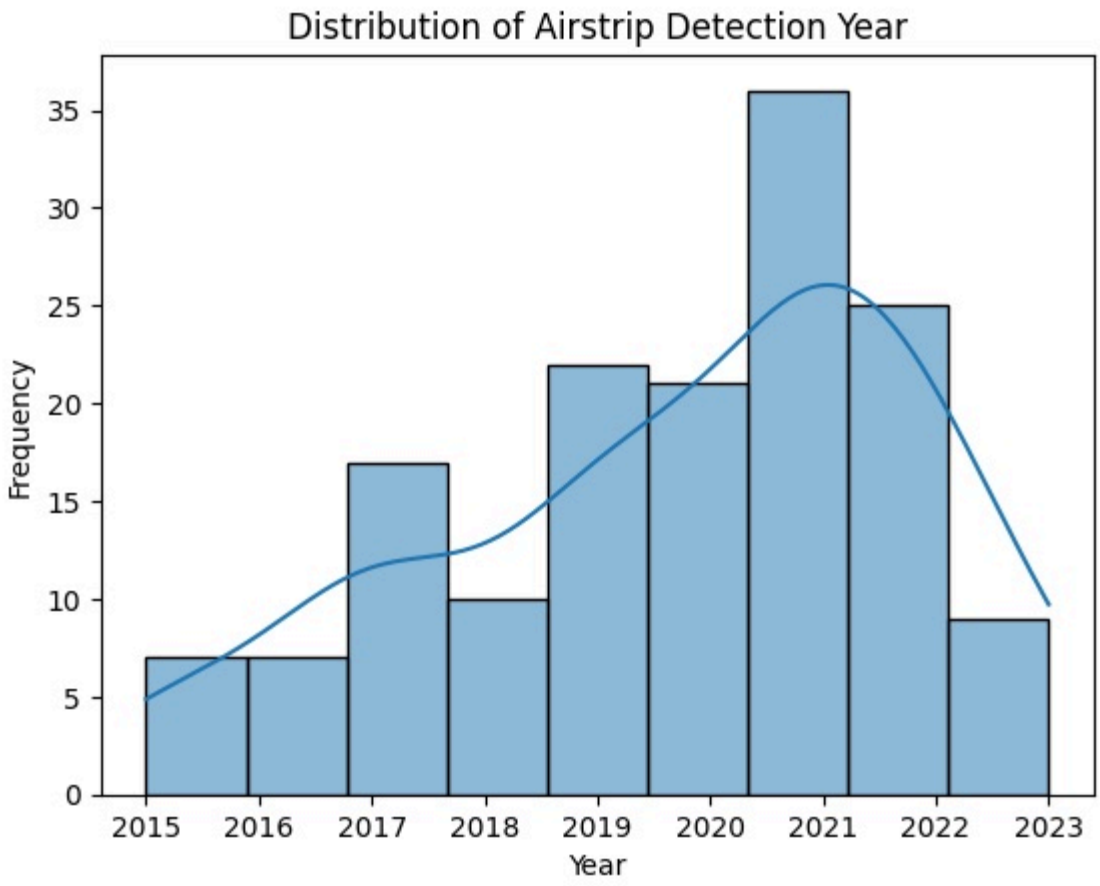
[1] We did not use this data set for the data challenge due to time constraints, but it highly recommended to increase the training set to improve the model.

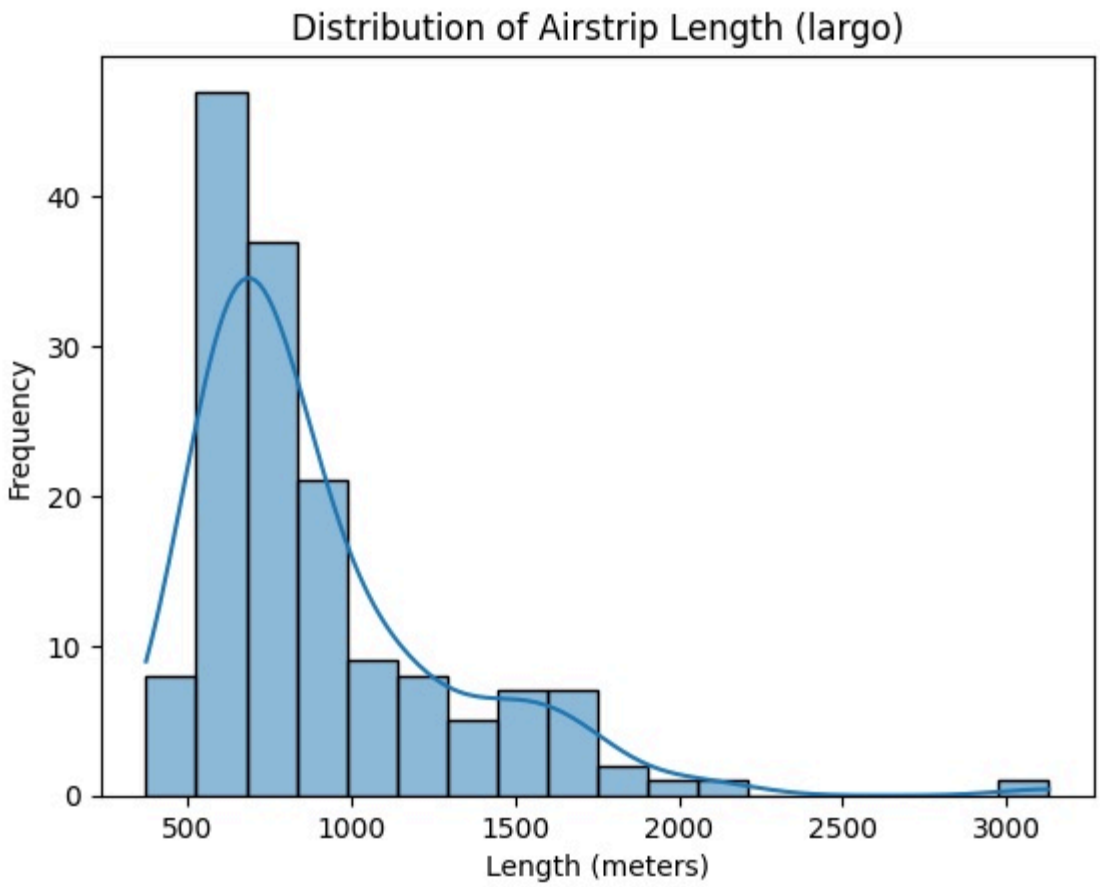

Sucumbíos
Quito
Manabí
Latacunga
Manta
Orellana
Ambato
Ecuador
Pastaza
Guayas
Amazonas
Morona Santiago
Cuenca
Machala
Tumbes
Iquitos
Loja
Loreto
Sullana
Amazonas
Piura
Jaén
Moyobamba
Cajamarca
Tarapoto
Chiclayo
San Martín
Cruzeiro do Sul
Cajamarca
La Libertad
Pucallpa
Chimbote
Acre
Ancash
Huánuco
Ucayali
Huánuco
Pasco
Cerro de Pasco
Huacho
Cobija
Madre de Dios
Junín
Lima
Huancayo
Huancavelica
Cusco
Pisco
Abancay
Ayacucho
Apurímac
Ica
Puno
Juliaca
Arequipa
Arequipa
Moquegua

The main variables for the airstrips are the year of detection and the length of the airstrip. The following plot shows the distributions.

- Notice that a significant number of detections happened before 2019. This is important because the Sentinel 2 “surface reflectance (SR)” data are not available for this region before 2019 (contrary to the information posted on the GEE site). We reverted to using the “top of the atmosphere” data (S2_HARMONIZED, i.e. Level-1C) for those years.
- The second distribution (airstrip length) has a pretty long tail. In particular, notice the entry at 3km (jumbo jet)!!! This is suspicious. This will require close inspection of the RGB images. For some airstrips, the “dirt” section where airplanes land and take off is much shorter than the cleared area identified by the `LINESTRING`. This is a matter of defining where the airstrip is truly located. This wasn’t done uniformly in the data sets provided to us. The Texas Department of Transportation Aviation Division prepared a document to assist ranchers and farmers to build safe “farm and ranch” airstrips. Their recommendations are probably valid

for our current project as the types of aircrafts are probably similar (e.g. Cessna 182S). The recommended minimum length of the airstrip is 3200′ (975 meters) + 100′ at each end (30 meters). The recommended width is 100′-200′ (30-60 meters). They also recommend a 20:1 slope at both ends to clear obstacles. That could be established by cutting trees, or by topping the trees. So, in summary, we expect the typical runway to be ~1000 meter long and 30 - 60 meter wide. Lengthwise, this is approximately consistent with the distribution from the training set. Some of the airstrips are incredibly short, but a skilled pilot and a light plane could handle this. Width-wise, we are looking at 3-6 pixels wide for the Sentinel 2 data (10-m resolution). This is consistent with most of the airstrips in the training set.





	largo
min	376.568
max	3127.76
mean	902.902
std	403.494

	yr
min	2015
max	2023
mean	2019.72
std	2.14962

Test dataset

The Areas of Interest (AOI) were provided by the Zindi team. After reviewing the submission file format and additional instructions from the Zindi team, the dimensions of the polygons defining the areas are (15410m x 15270m) or (15410m x 15250m). The local UTM projection of each polygon was used to define the Sentinel 1 / 2 pixel grid. The default Sentinel 1 / 2 pixel resolution (10m) was used for this project. Bands with lower resolutions were automatically resampled by Google Earth Engine to 10m resolution. The default CRS used by Google Earth Engine is EPSG:4326,

which required a change of CRS in the export function to Google Drive for the final image. The pixel grid was aligned to match with the AOI geometry and provide a perfect overlap at 10m resolution.

The AOI do contain known airstrips used during the training phase, but the year of detection for those airstrips may not match the year used for inference for the competition. A full year of data (i.e. year of detection) was used to generate the cloud-free composite images (see details below). All images used the surface reflectance (Level 2A) collection. The following image shows “aoi_2020_02” AOI. A number of airstrips are clearly visible in this image.

Satellite imagery

For this project we'll be using the Sentinel 1 (SAR) and Sentinel 2 (Multi Spectral) data.

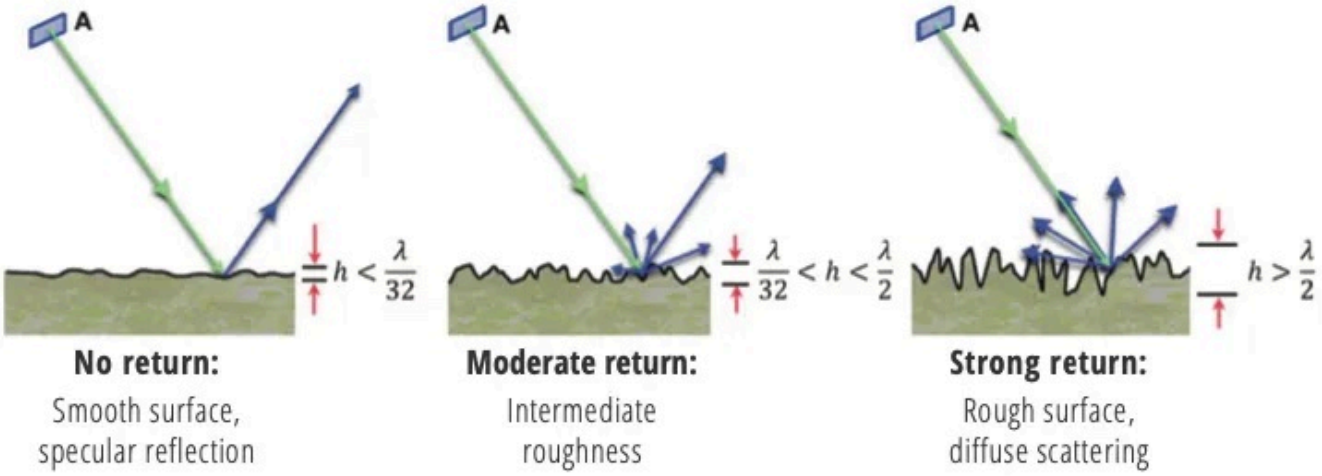
Sentinel 1

Sentinel-1 (S1) is a radar imaging mission consisting of two satellites, A and B, using Synthetic Aperture Radar (SAR) technology in the C-band. Key points include:

- **Coverage**: Each satellite provides global coverage every 12 days, with 6-day coverage over the equator when using both satellites.
- **Active Sensing**: SAR actively sends its own energy and measures the amount reflected back from Earth's surface.
- **Surface Sensitivity**: The radar signal responds to surface features like structure and moisture.
- **Polarization**: The radar signal can be polarized, which refers to the orientation of the electromagnetic wave. Common polarization combinations include.
    - VH (Vertical transmit, Horizontal receive)
    - VV (Vertical transmit, Vertical receive)
    - HH and HV are only collected over the poles and are not relevant here.
- **Data Modes**: The satellite collects data in different modes. For land, the Interferometric Wide Swath mode is most suitable.
- **Orbit Properties**: Data is collected on either 'ASCENDING' or 'DESCENDING' orbits.
- **Resolution**: Data resolution can be 10, 25, or 40 meters.

Some characteristics of radar images:

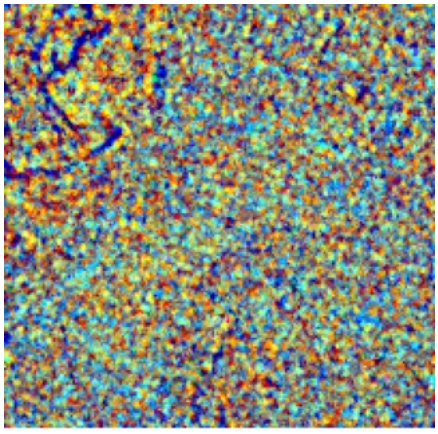
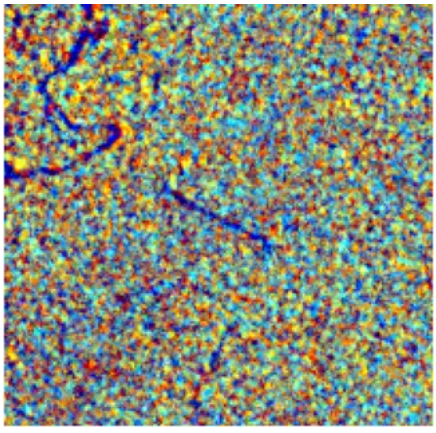
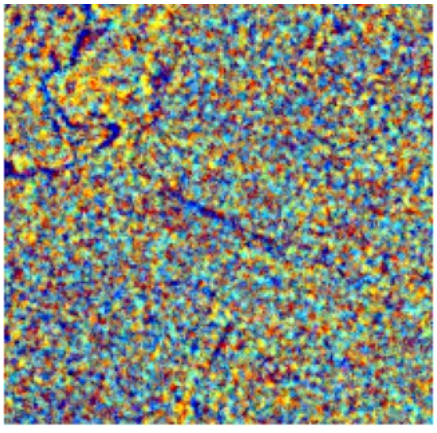
- SAR images are typically grayscale, where pixel intensity represents the backscatter
- Backscatter is the term used for the portion of the transmitted radar signal that is reflected back to the satellite after interacting with the Earth's surface.
- Pixel intensity is often converted to the **backscattering coefficient** (measured in decibels, dB), ranging from +5 dB (bright objects) to -40 dB (dark surfaces).
- **Higher backscatter intensity** generally indicates rougher surfaces.
- **Smooth surfaces** (e.g., roads, runways, calm water) appear dark because most radar energy is specularly reflected away from the satellite.
- **Rough surface scattering** (e.g., bare soil or water) is more sensitive to VV polarization.
- **Volume scattering**, from objects like leaves and branches, is better detected by cross-polarized data (VH or HV).
- **VV/VH Ratio**: A higher value indicates that surface scattering (detected by VV) is stronger than volume scattering (detected by VH). This is typically more useful for surfaces with minimal structure, like bare ground or smooth surfaces. Note: For bare ground, the VV backscatter alone is typically more informative since it responds well to surface roughness and moisture. The VH/VV ratio might not provide significant additional insights for bare ground areas.

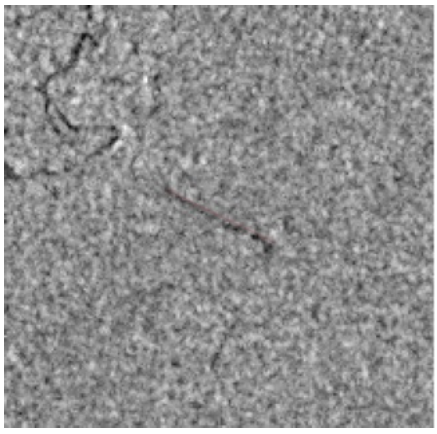
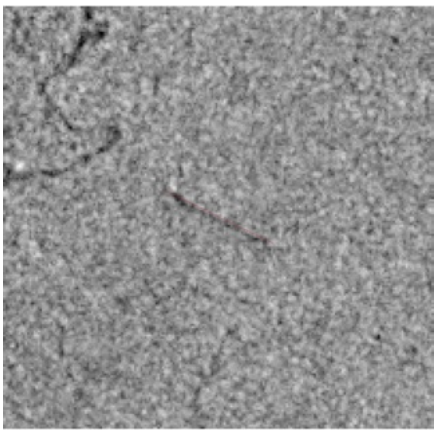
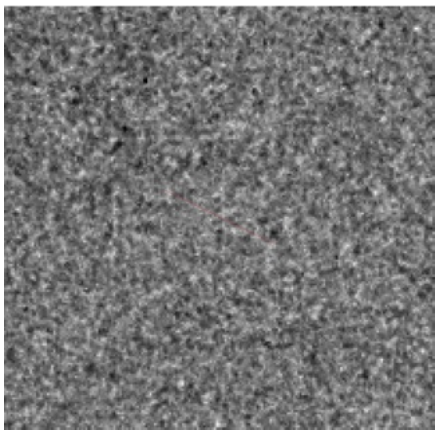


The code in this notebook filters the Sentinel-1 collection by `transmitterReceiverPolarisation`, `instrumentMode` and then calculates composites for several observation combinations that are displayed in the map to demonstrate how these characteristics affect the data.

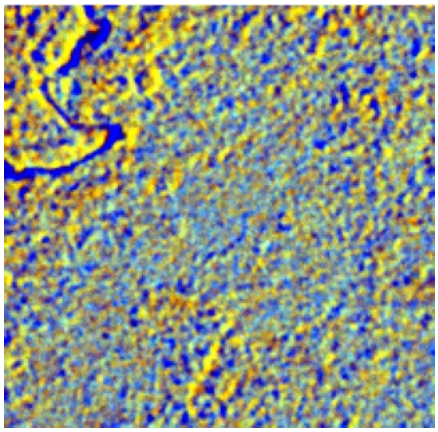
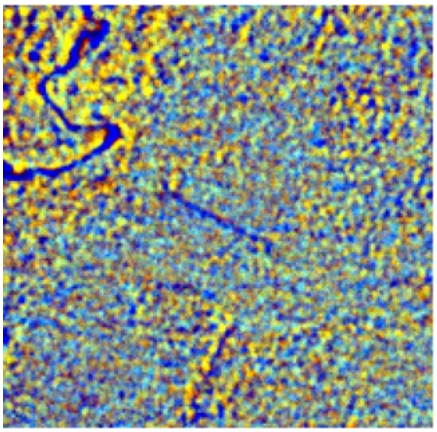
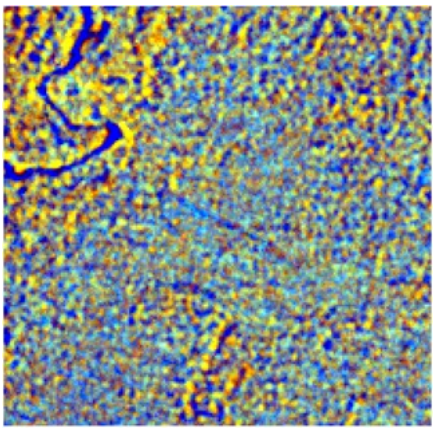
Airstrip index = 9, Detection Year = 2022

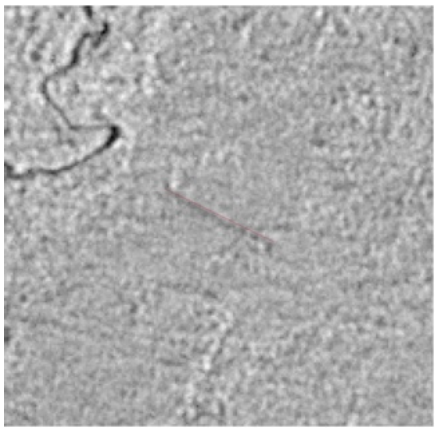
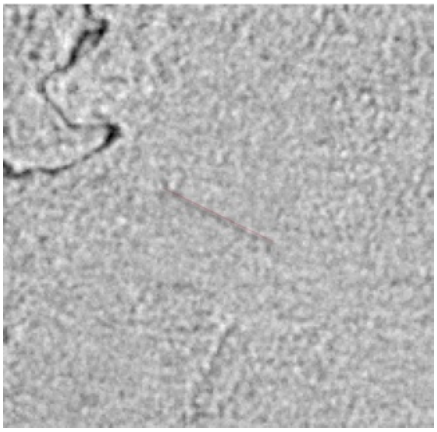
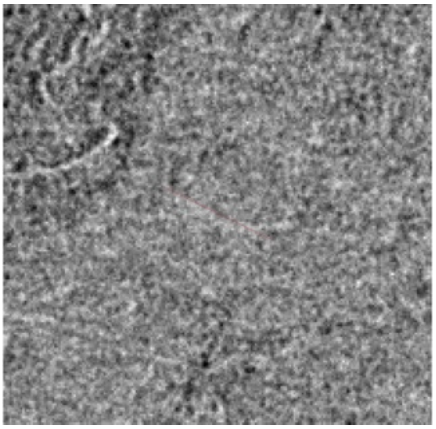
Monthly Median Composites

June 2022 VV - Red; VH - Green; VV/VH - Blue	**July 2022** VV - Red; VH - Green; VV/VH - Blue	**September 2022** VV - Red; VH - Green; VV/VH - Blue
		
July 2022 VV - Grayscale	**July 2022** VH - Grayscale	**July 2022** VV/VH - Grayscale

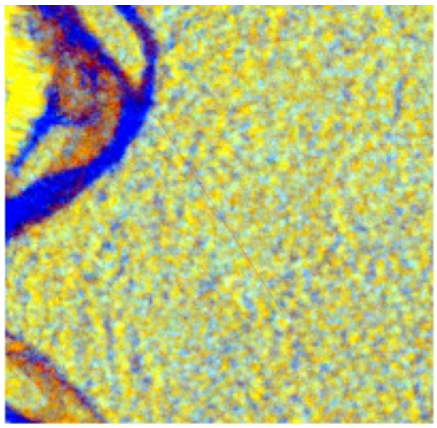
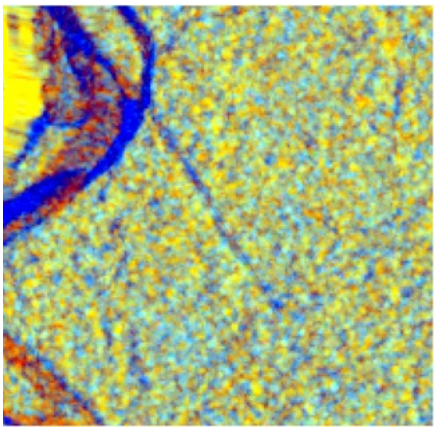
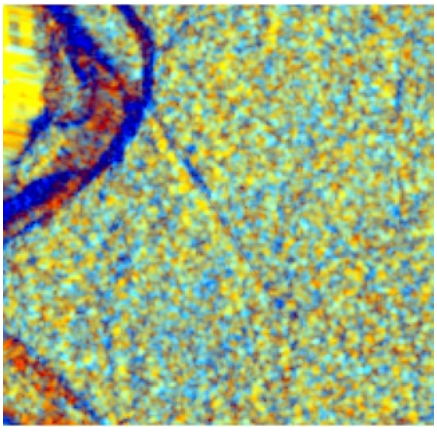
Annual Median Composites

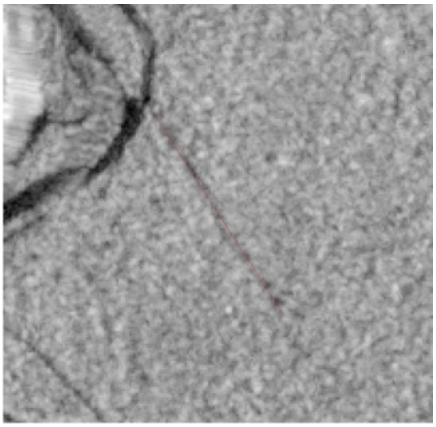
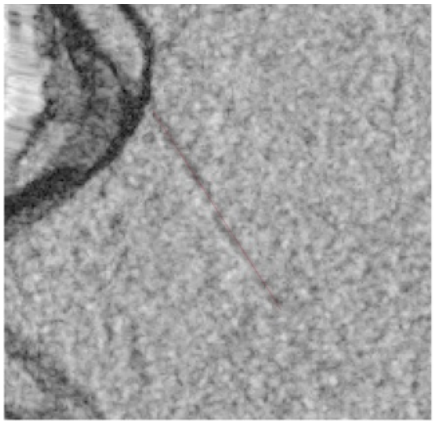
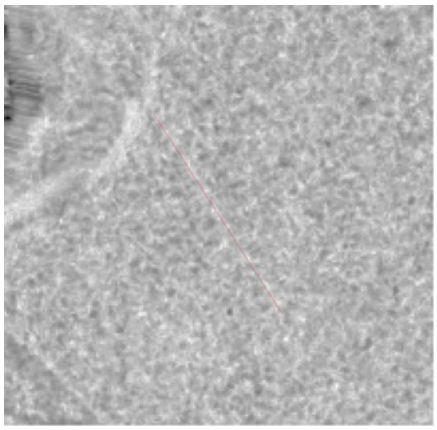
2021 VV - Red; VH - Green; VV/VH - Blue	**2022** VV - Red; VH - Green; VV/VH - Blue	**2023** VV - Red; VH - Green; VV/VH - Blue
		
2022 VV - Grayscale	**2022** VH - Grayscale	**2022** VV/VH - Grayscale

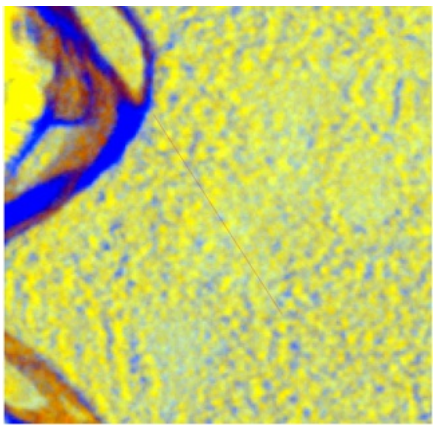
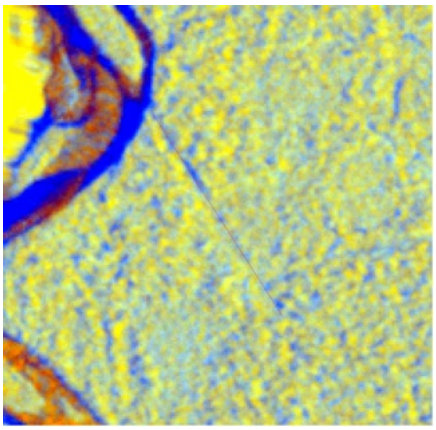
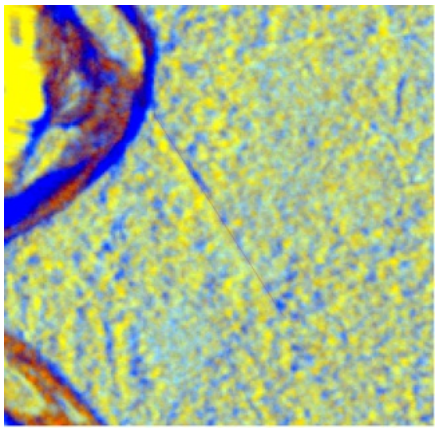
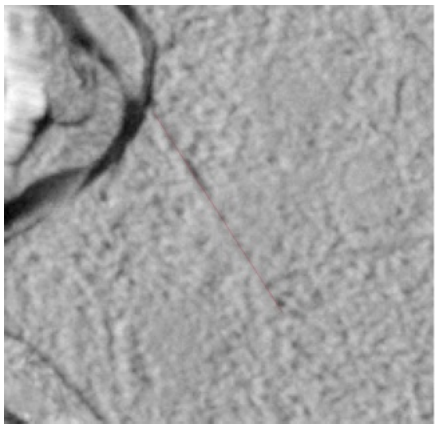
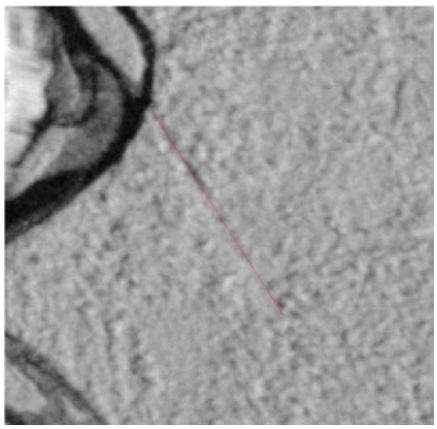
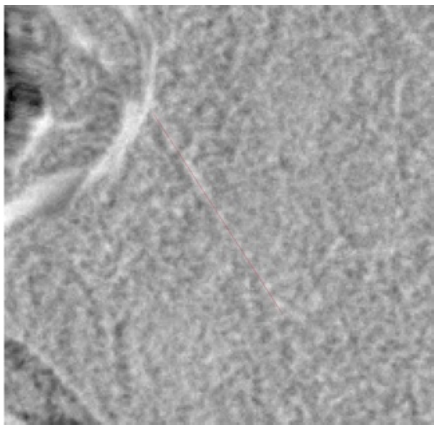
Airstrip index = 1, Detection Year = 2022

Monthly Median Composites

March 2022 VV - Red; VH - Green; VV/VH - Blue	**July 2022** VV - Red; VH - Green; VV/VH - Blue	**September 2022** VV - Red; VH - Green; VV/VH - Blue
		
July 2022 VV - Grayscale	**July 2022** VH - Grayscale	**July 2022** VV/VH - Grayscale

Annual Median Composites

2021 VV - Red; VH - Green; VV/VH - Blue	**2022** VV - Red; VH - Green; VV/VH - Blue	**2023** VV - Red; VH - Green; VV/VH - Blue
		
2022 VV - Grayscale	**2022** VH - Grayscale	**2022** VV/VH - Grayscale
		

Sentinel 2

Sentinel-2 (S2) is a wide-swath, high-resolution, multispectral imaging mission with a global 5-day revisit frequency. The S2 Multispectral Instrument (MSI) samples 13 spectral bands: visible and NIR at 10 meters, red edge and Short Wave Infra-Red (SWIR) at 20 meters, and atmospheric bands at 60 meters spatial resolution. It provides data suitable for assessing state and change of vegetation, soil, and water cover.

The first Sentinel 2 satellite (2A) started providing data in mid 2015. Sentinel 2C was launched in September 2024. The data are accessible through the Google Earth Engine and images can be downloaded to Google Drive. See Python Jupyter Notebook for details.

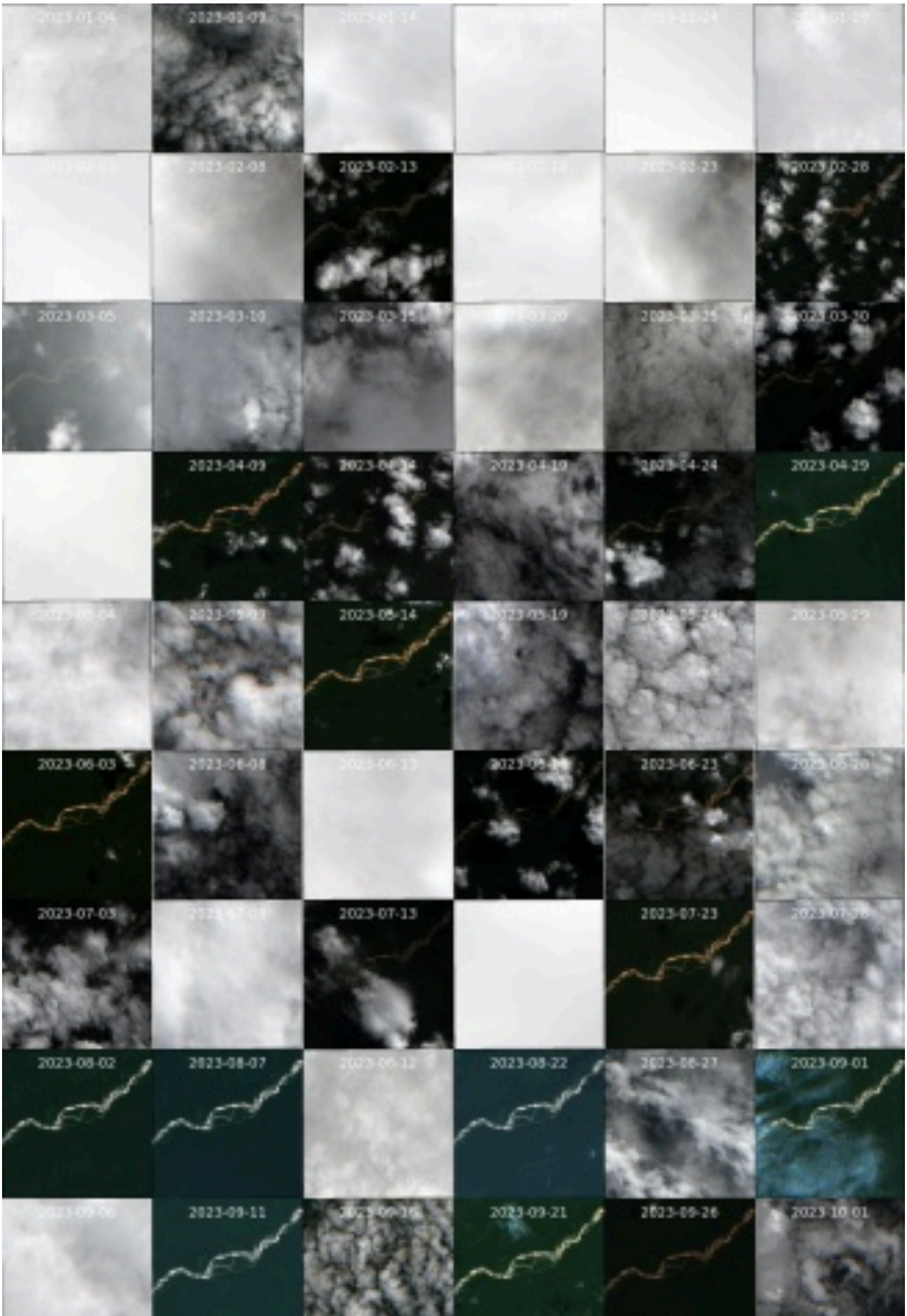
Visualizing the multi spectral data

The Jupyter notebook contains several methods for downloading or visualizing images. The code is evolving quickly as several approaches resulted in suboptimal results. Here is a short list of lessons learned:

- The Amazon region is constantly covered by clouds (unlike southern AZ)! This is particularly challenging for visualizing changes in ground cover over time. For some areas, more than 50% of the images contain clouds.
- Filtering clouds is essential and different methods are available. For instance, the tutorials provided by Climate Change AI used a standard cut on the fraction of pixels containing clouds. This worked fine for Germany, especially when combining the data for a full year, but it didn't work well for the Amazon region. Google offers a second image collection (`CLOUD SCORE+`) with tunable parameters to mask clouds and shadows that can easily be linked with the main Sentinel 2 image collection to create composite images.

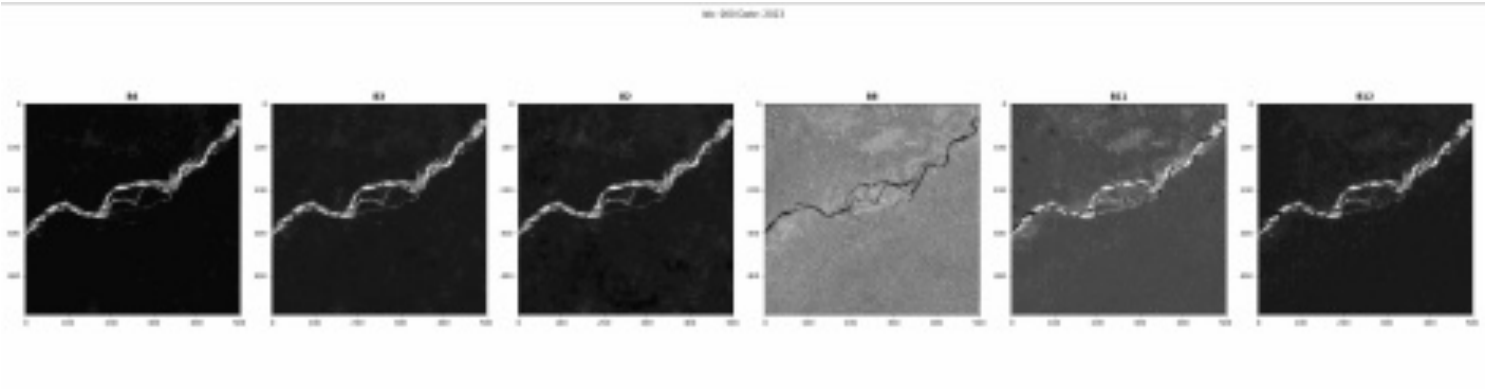
The next image shows a time series of Sentinel 2 data taken for a fixed location (~5km x 5km) for the year 2023. The Python notebook was used to download the GeoTIFF files for this region to a Google Drive. Another function in the same notebook was used to load all the individual images and produce this gallery of images.

- For this particular location, about 1/2 the data is useless as clouds obscure the view of the ground.
- The individual images can only show 3 bands, and in this case bands B4,B3,B2 (R,G,B) are used to produce the image. The GeoTIFF files contain 6 bands [B4, B3, B2, B8, B11, B12].


2023-01-04
2023-01-09
2023-02-08
2023-02-13
2023-02-28
2023-03-05
2023-03-10
2023-04-09
2023-04-19
2023-04-24
2023-04-29
2023-05-14
2023-06-03
2023-07-03
2023-07-13
2023-07-23
2023-08-02
2023-08-07
2023-08-22
2023-09-01
2023-09-11
2023-09-21
2023-10-01

The Google cloud mask image collection was used to produce annual cloud-free (mostly) images for all airstrips in the training set starting in 2015. The Sentinel 2 Level-1C data doesn't start until mid-2015 and for some areas combining 6 months of data isn't sufficient to produce a composite image or a cloud-free image covering the entire area of interest. Manual sorting and inspection of images is necessary before generating masks and images for the training data set.

The next image shows the cloudless composite for the same area of interest for the year 2023. The individual panels show gray-scale images from each band (B4, B3, B2, B8, B11, B12). The airstrip (at the center of the image) is visible in all bands.

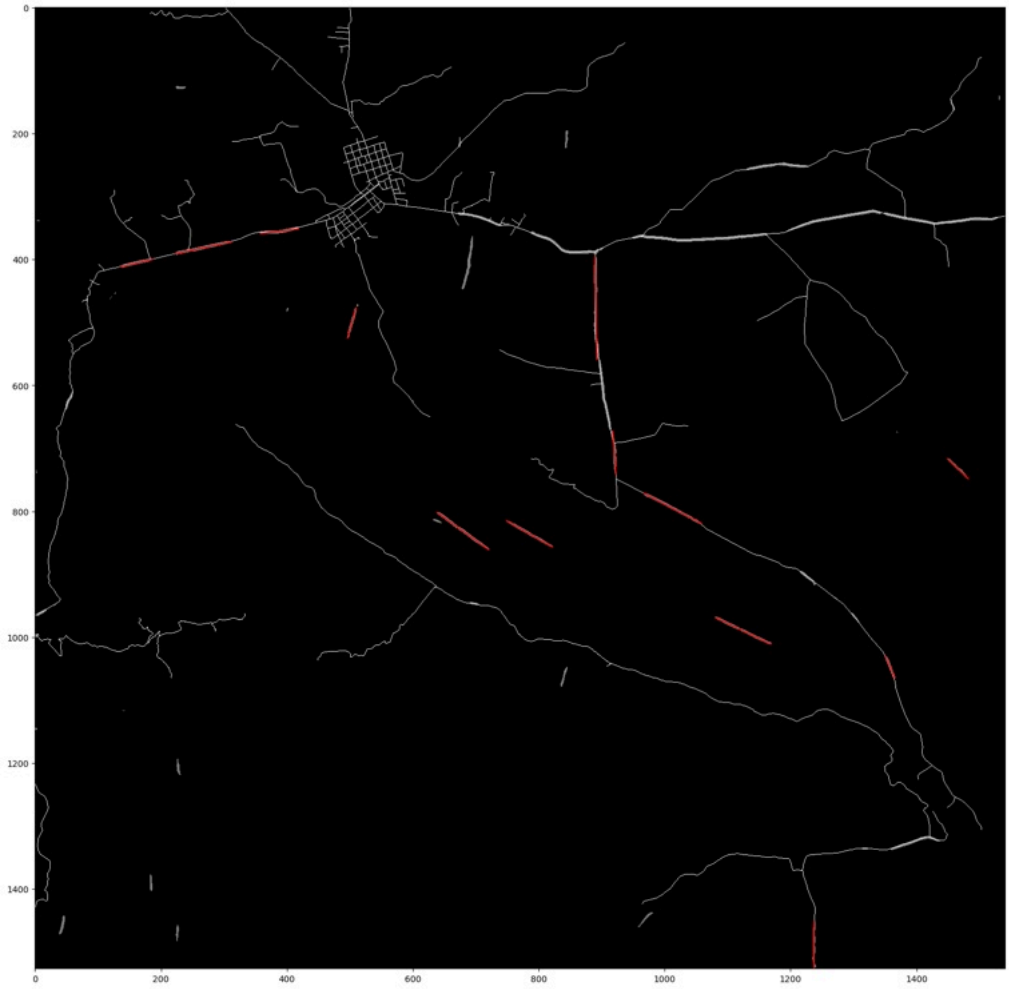


Sentinel 1 / 2 combined images for this project.

Sentinel 1 and 2 composite images were produced simultaneously for this project. Although Sentinel 1 images are not impacted by clouds, all images for a given year were combined to create a composite image using a median reducer. VV, VH, and VV/VH bands were used for this project. For the Sentinel 2 data, cloud-free composite images were created for the same time interval and the default parameters were used for filtering clouds. Six bands were selected: B4 (red), B3 (green), B2 (blue), B8 (nir), B11 (swir 1), and B12 (swir 2). The resulting composite images contain 9 bands (6x Sentinel-2, 3x Sentinel-1).

Road Imagery

Due to some limitations of our approach, image post-processing was included to reduce the number of false positive detections from the output of the model. Unfortunately, the training dataset didn't include enough tiles located around villages with small road networks. Dirt roads are very similar to airstrips, e.g. width-wise, and can cause confusion. Due to time constraints, we had to improvise a solution, hence the image cleaning step. Geospatial information about roads was obtained from the Open Street Map project. The Python API provides easy access to GeoJSON objects within the bounding box of our areas of interest. The following image shows airstrip detections (red lines) as well as the network of roads. See the image cleaning section for details on the analysis process.



Deep learning model

Model and framework selection

PyTorch or torchgeo? Do we need torchgeo to handle the GeoTIFF files? The Python package `rasterio` can load the GeoTIFF files and export multi-band images to the model. Is there anything else that would require torchgeo? After reviewing the project documentation, we didn't see significant advantages to using torchgeo, so **PyTorch** was used for the framework.

For simplicity and speed of development a standard **`UNET`** model was chosen as the first model for this project. Given the limited size of the training set, a **ResNet50** encoder with pre-trained weights from **ImageNet** (RGB only) was chosen as a starting point.

This semantic segmentation model was configured with a single class. The output of the model is a binary format: **0 for background, 1 for airstrip**.

The evaluation metrics for this model are **accuracy**, **F1 score**, and **Jaccard index**. Given the small ratio of mask area to background area, the F1 score and Jaccard index should be a much better metric than accuracy. The Zindi competition metric is accuracy, but the calculation is performed over a much

smaller area as only detected runways with a background buffer are to be submitted (reduced file size).

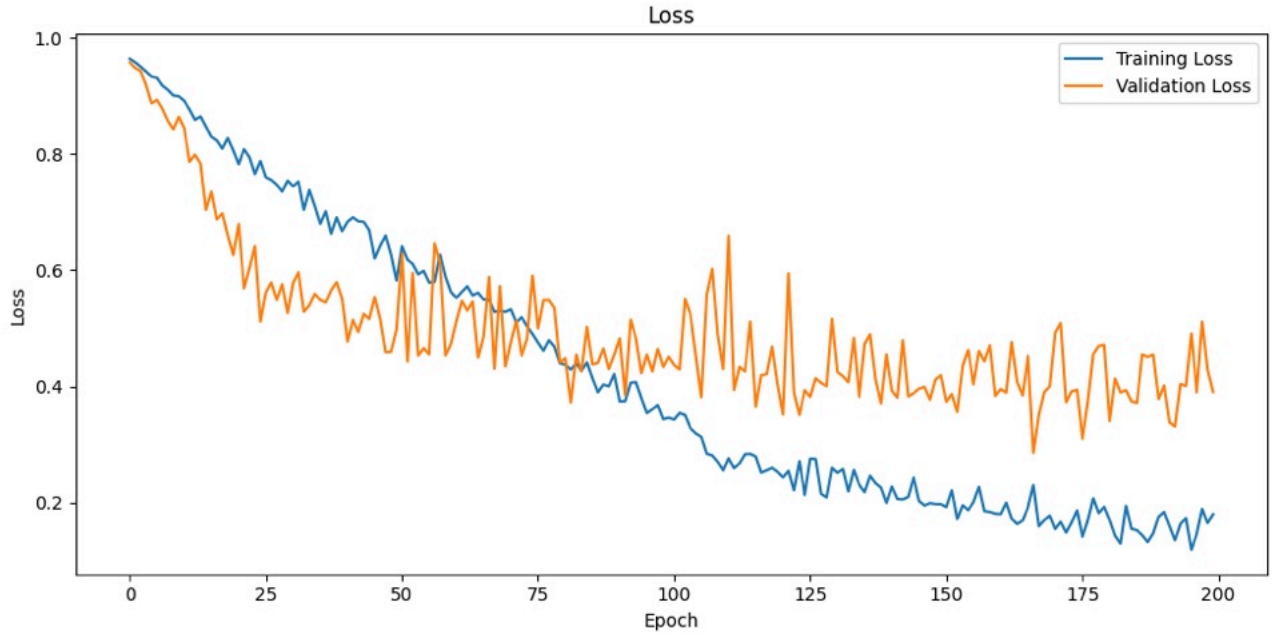
Using the default RGB bands with the ResNet50 encoder, the model size is described below. Adding additional bands increases the number of parameters.

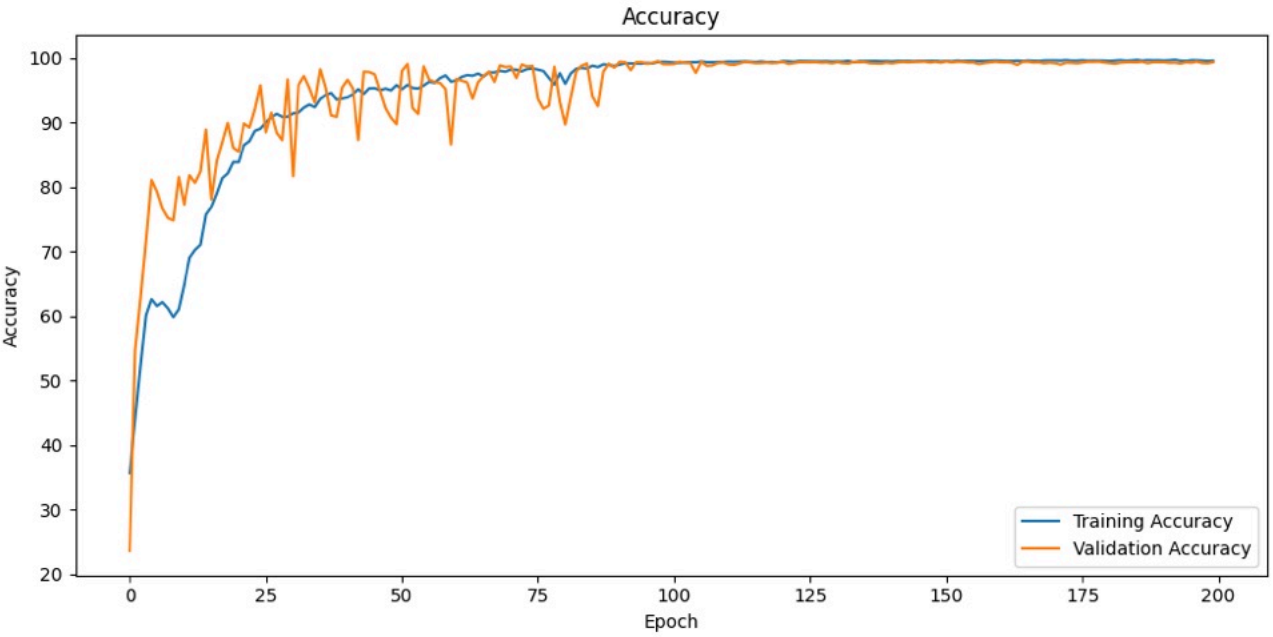
```
==========================================================================
Total params: 32,521,105
Trainable params: 32,521,105
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 463.01
Params size (MB): 124.06
Estimated Total Size (MB): 587.64
--------------------------------------------------------------------------
```

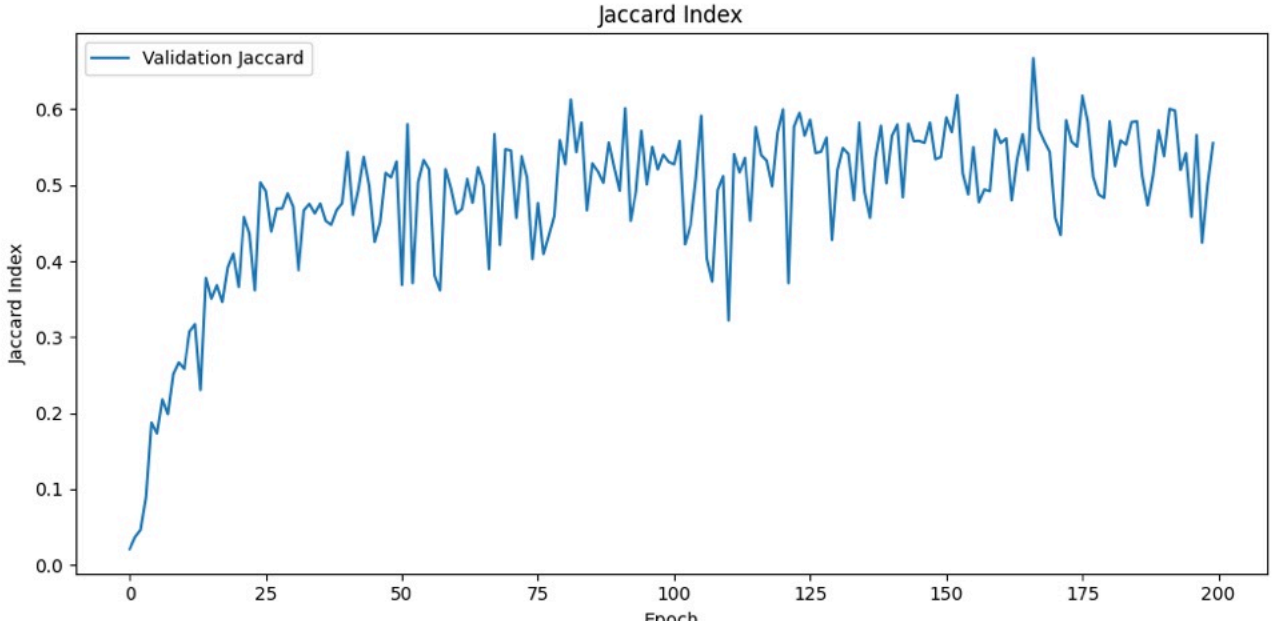
Hyperparameter tuning

The baseline version of the model uses a ResNet50 encoder with pre-trained weights optimized on the ImageNet dataset of images and labels. Only 3 channels are used initially (RGB – Sentinel 2 bands B4,B3,B2). The **Dice** loss function is used as a starting point as well as the **Adam** optimizer. Other loss functions are available in the code as options. A **batch size** value of 8 is used as a starting point and could be optimized at a later time. The **learning rates** (pretrained weights and new weights can have different rates) are the main hyperparameters that will be optimized as a first step to improve model performance. For the first iteration, a learning rate value of 0.0001 is used for the optimizer.

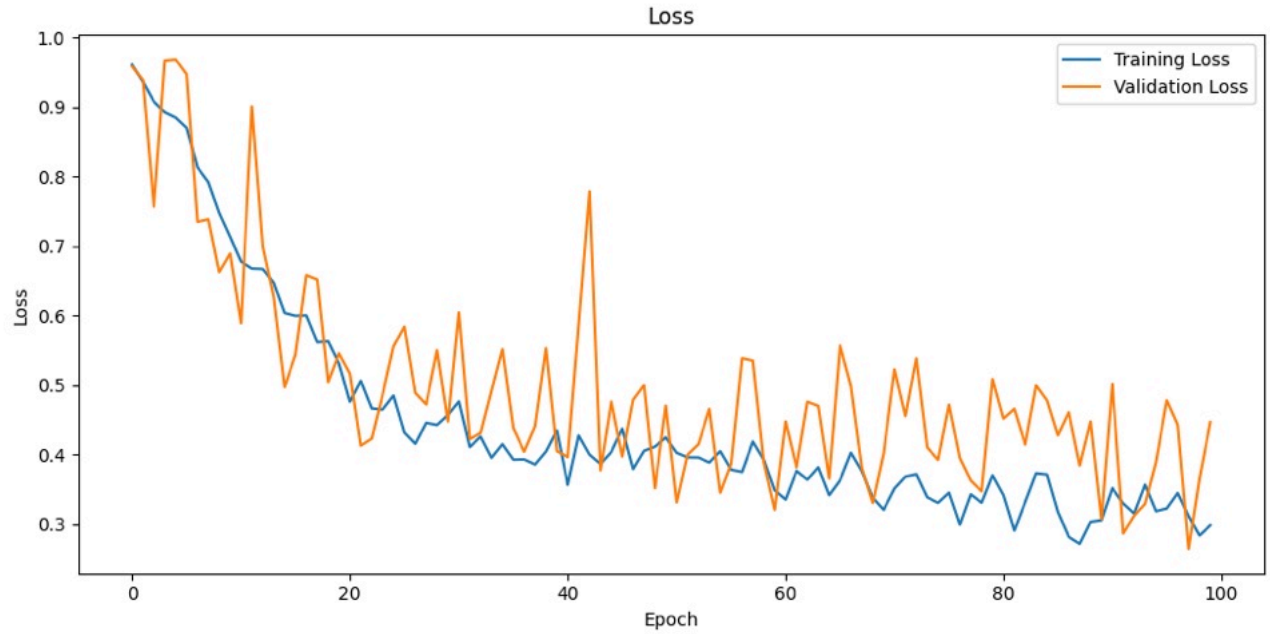
The following image shows the output of the loss function, the model accuracy, and the Jaccard index (validation only) as a function of training epoch. The training loss decreases linearly up to epoch ~100, then starts to flatten out. The validation loss quickly reaches a plateau after ~25-30 epochs. Training and validation loss reach similar values around epoch 75. This is also when the accuracy and Jaccard loss reach a plateau. Based on those results, it could be argued that stopping training after 75 epochs prevents overtraining... The current version of the code identifies the “best” model as the model that minimizes the validation loss. This would correspond to epoch ~165 for this training run. The validation loss fluctuates significantly due to the small number of images, so this result is questionable.

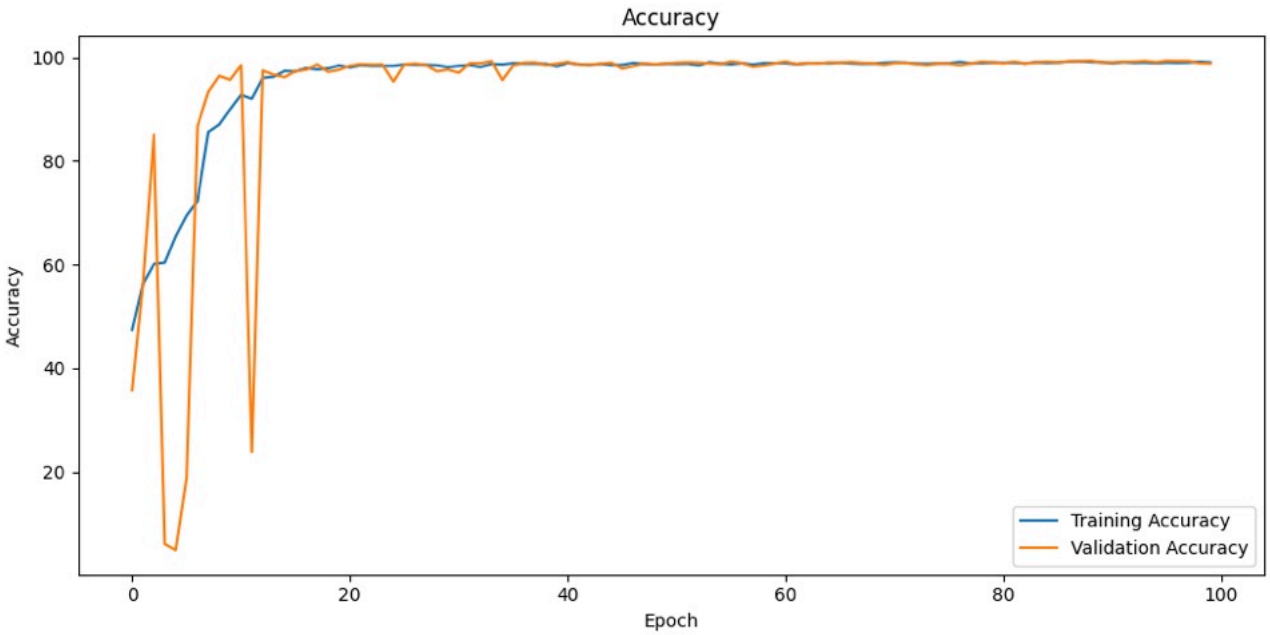

Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.8
0.6
0.4
0.2
0
25
50
75
100
125
150
175
200

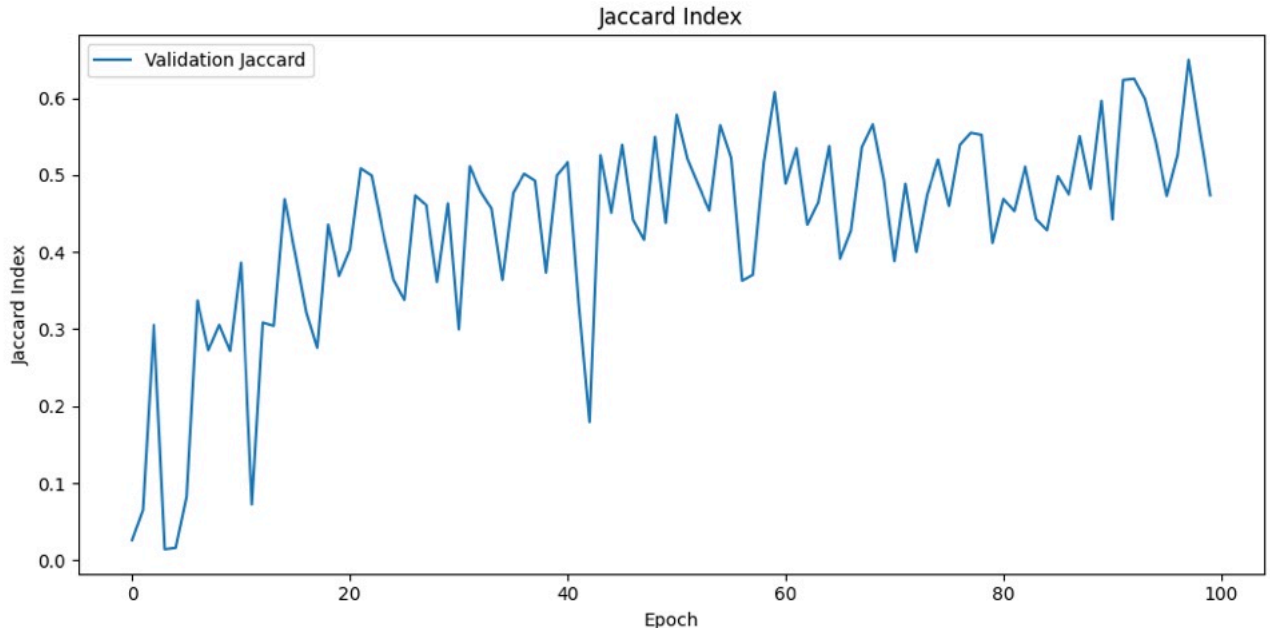

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
90
80
70
60
50
40
30
20
0
25
50
75
100
125
150
175
200


Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
25
50
75
100
125
150
175
200

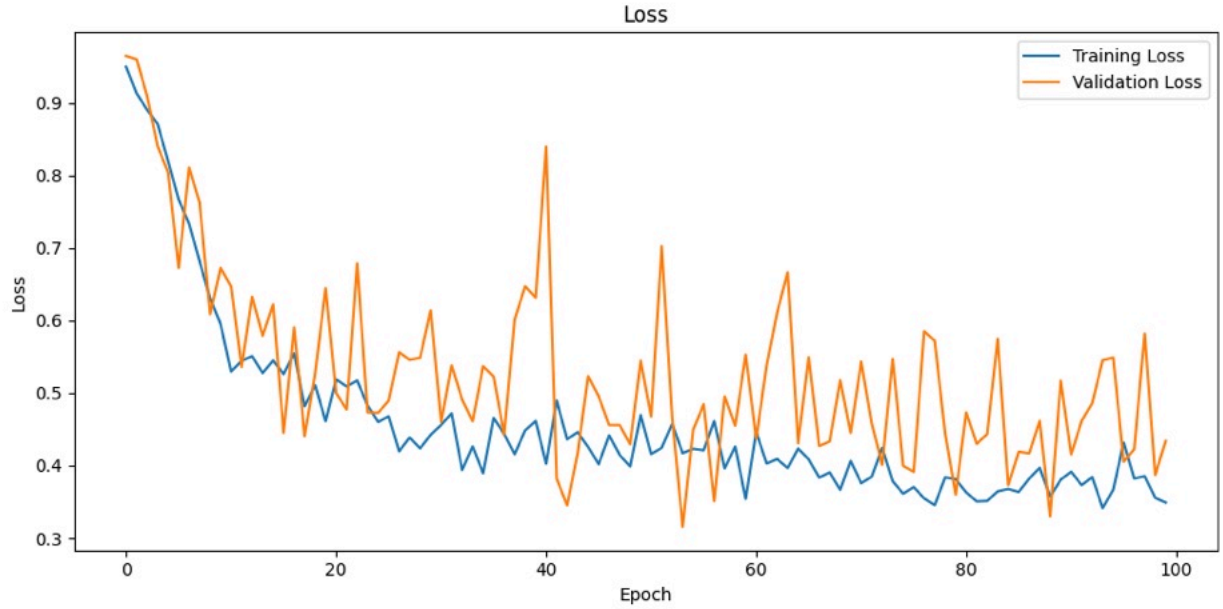
The learning rate was increased to a value of **0.0005** (5x nominal) and the model training was restarted from scratch. Only 100 epochs were used as the model was expected to learn faster. The following plot shows the output of the training process. The model learns much faster and transitions to a flatter slope around epoch 30. Notice that the training and validation loss values are in better agreement during training.

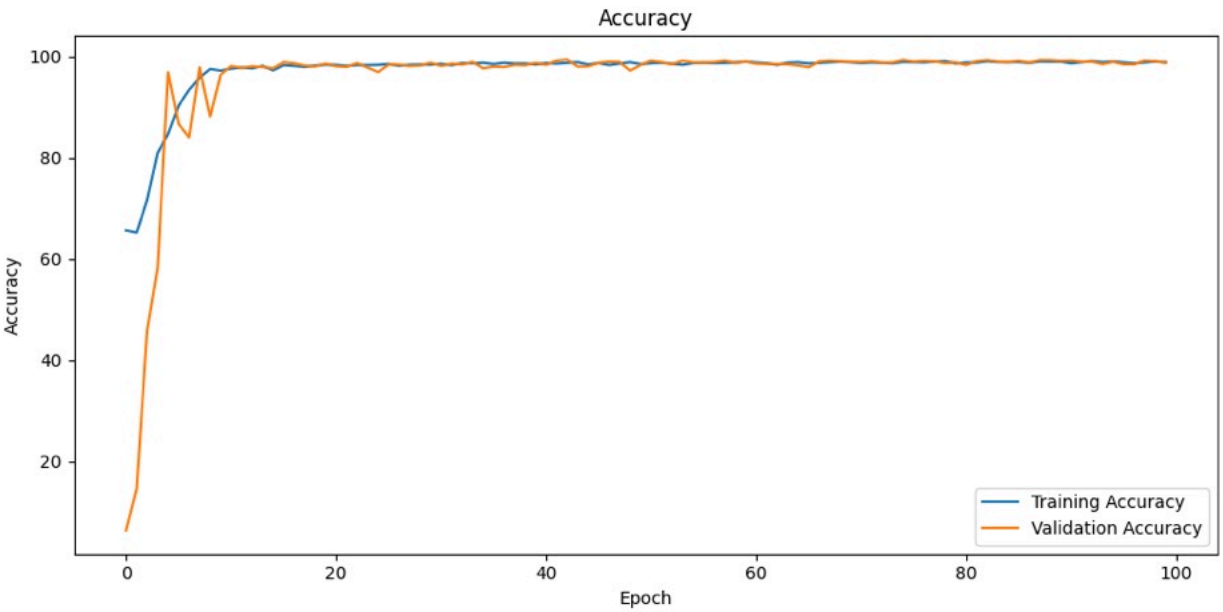

Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100

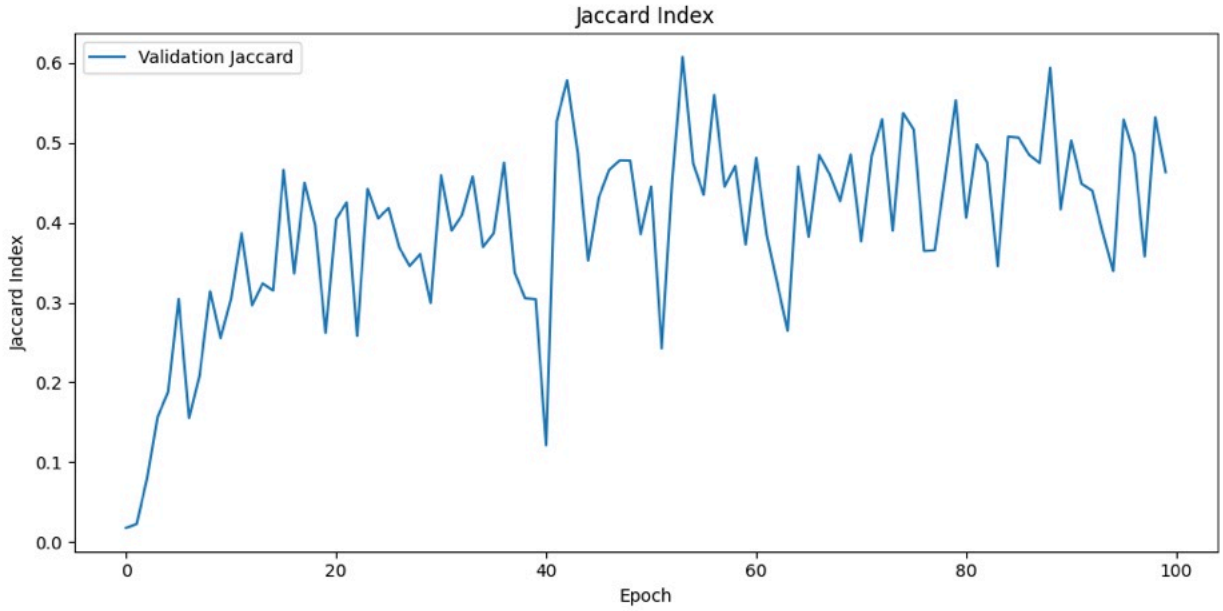

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
20
40
60
80
100


Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

The learning rate was subsequently increased to a value of **0.001** (10x nominal). The model learns faster in the first few epochs but does not converge faster.


Loss
Training Loss
Validation Loss
Loss
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100


Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
20
40
60
80
100


Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

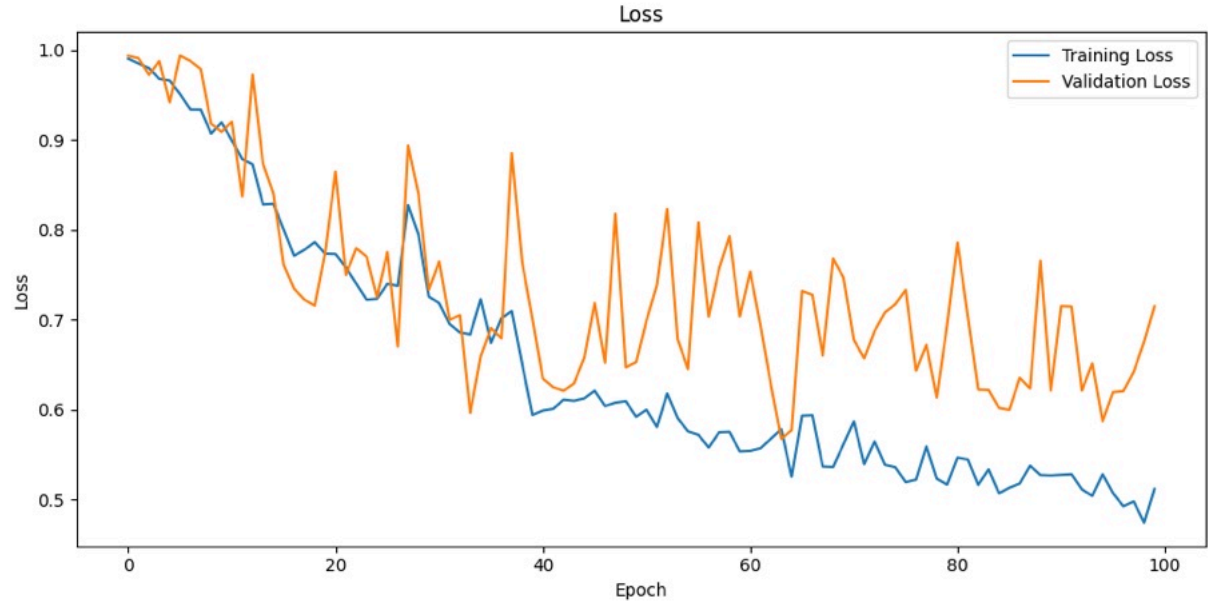
Based on those results, it is not clear if a learning rate value of 0.001 or 0.0005 is more efficient. We can use both as we investigate the impact of changing the mask width in the next tests. For the rest of testing a value of **0.0005** was used.

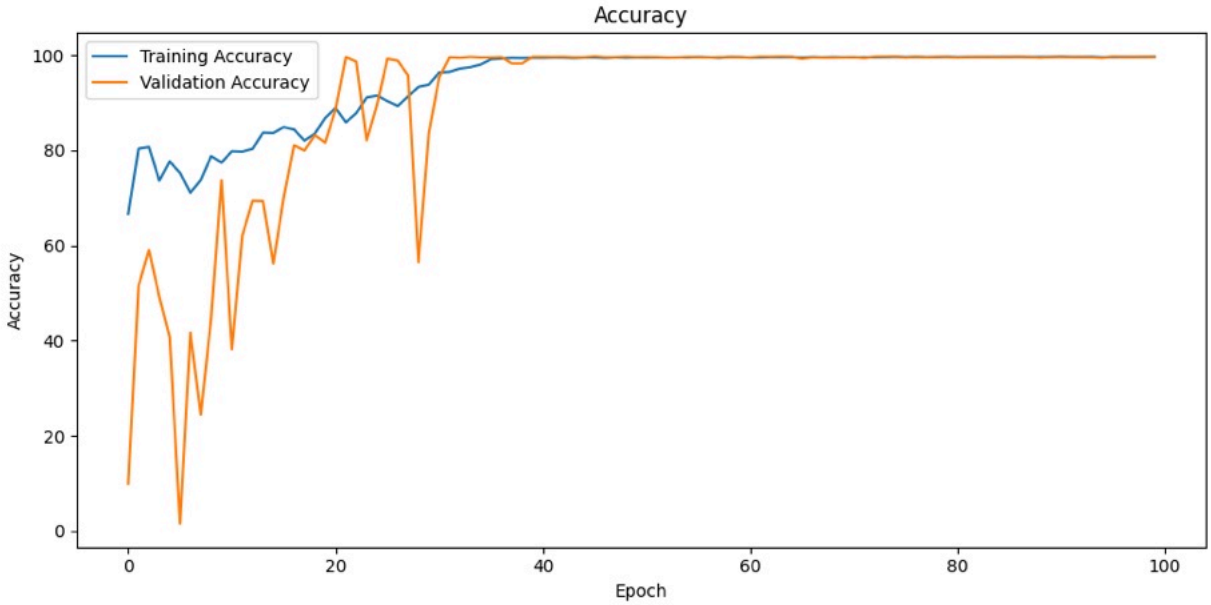
Impact of mask width on the model performance

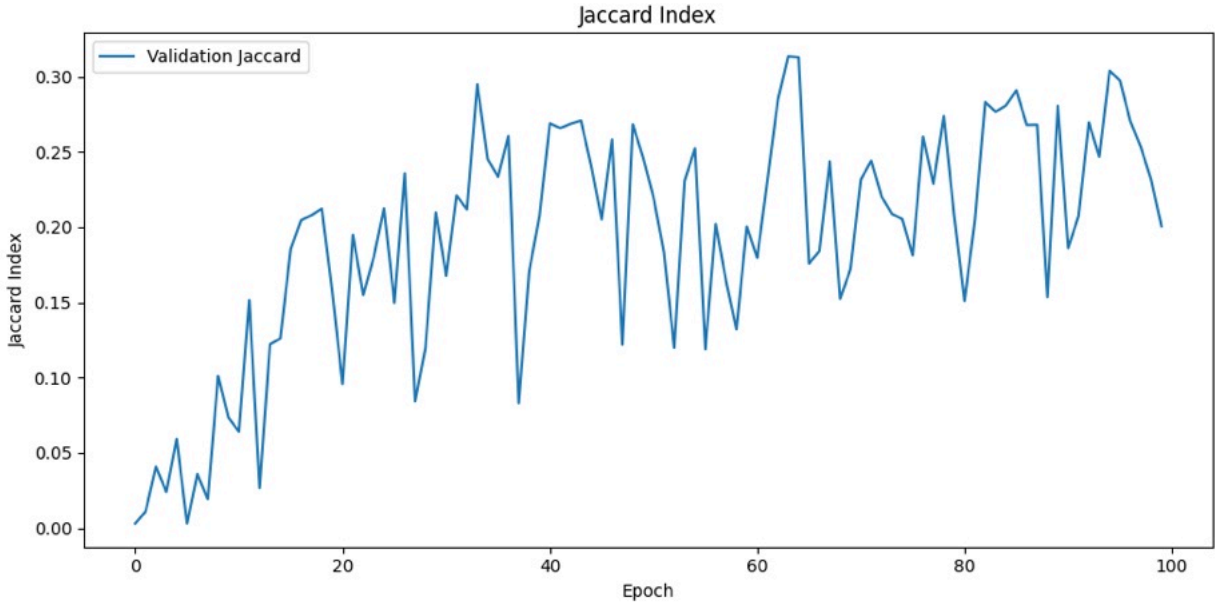
Masks were generated with different buffer widths (10,20,50,100,200 meters). The steps are as follows:

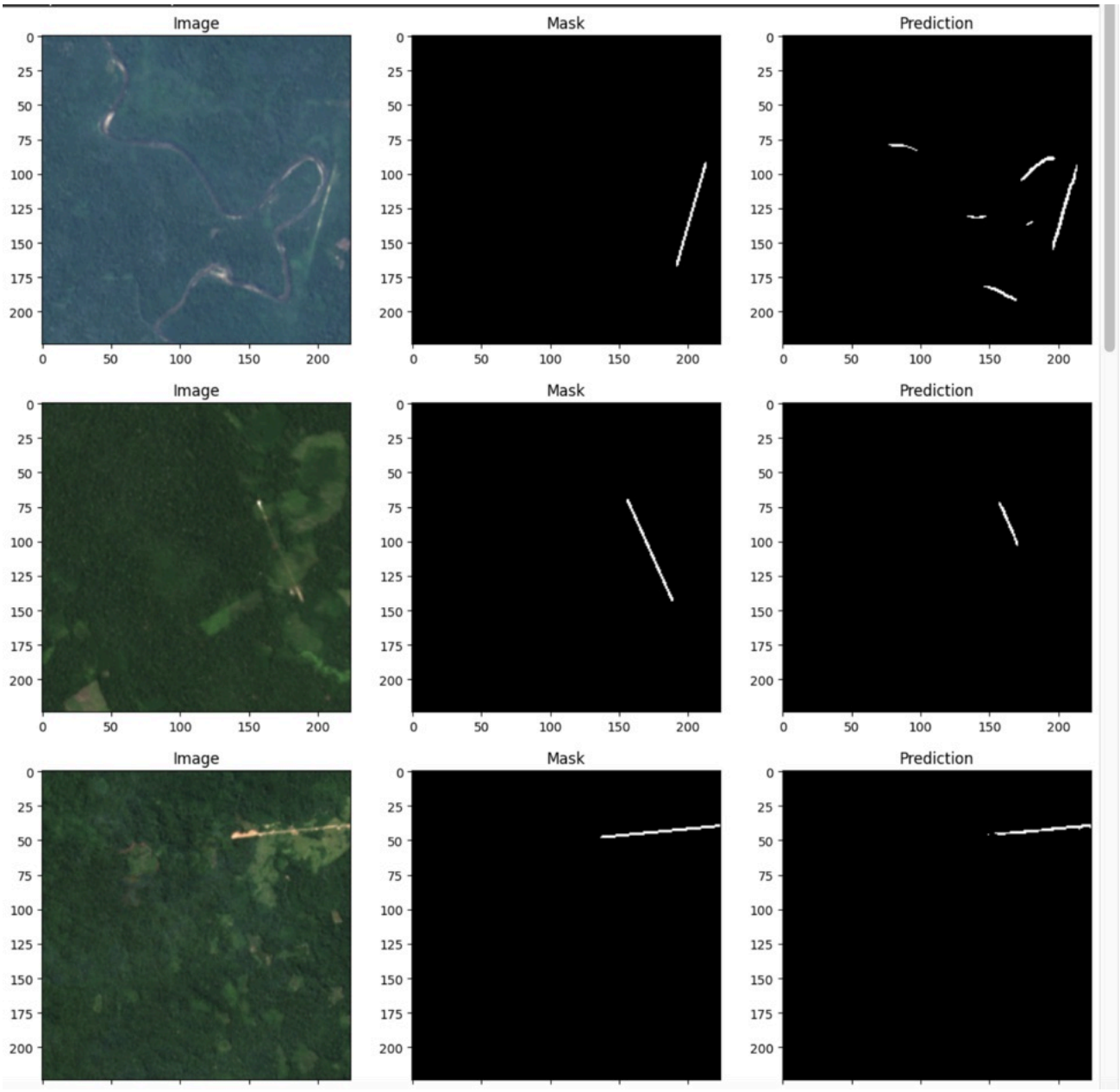
- Using `rasterio` to load the training image, extract the image dimensions and the bounding box parameters including the CRS.
- Create a mask with all pixels set to a value of 0.
- Load the shape file for that particular airstrip and convert the `LINESTRING` object to a `POLYGON` by creating a buffer of X meters around the LINESTRING.
- Add the `POLYGON` to the mask and set the value of all pixels intersecting the `POLYGON` to 1.

The initial buffer width was 50 meters. The previous results were all obtained with the nominal 50-meter buffer. The following results show the output of the model with a **10-meter** buffer. The model learns at a good rate, but notice that the Jaccard index is very low, which is probably expected for smaller areas.


Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100

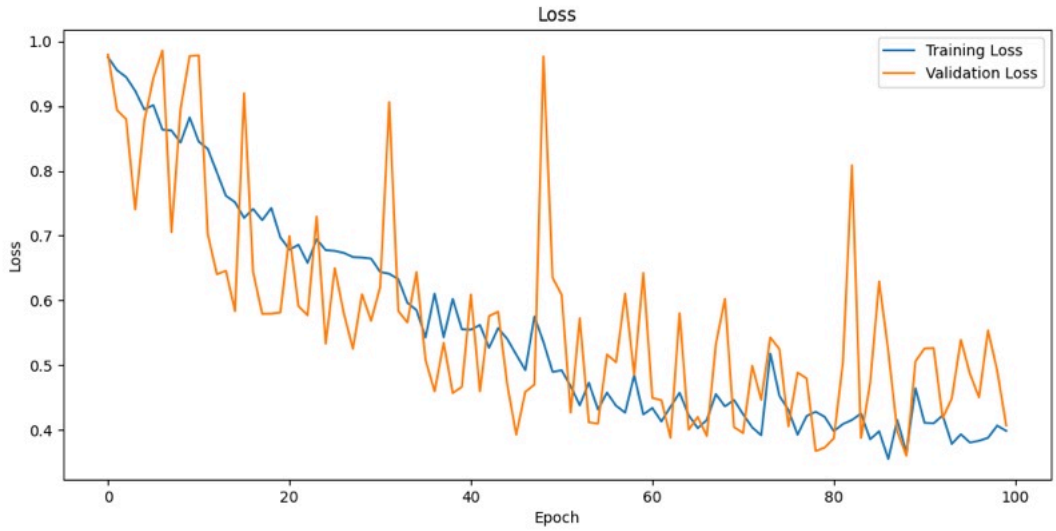

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
0
20
40
60
80
100

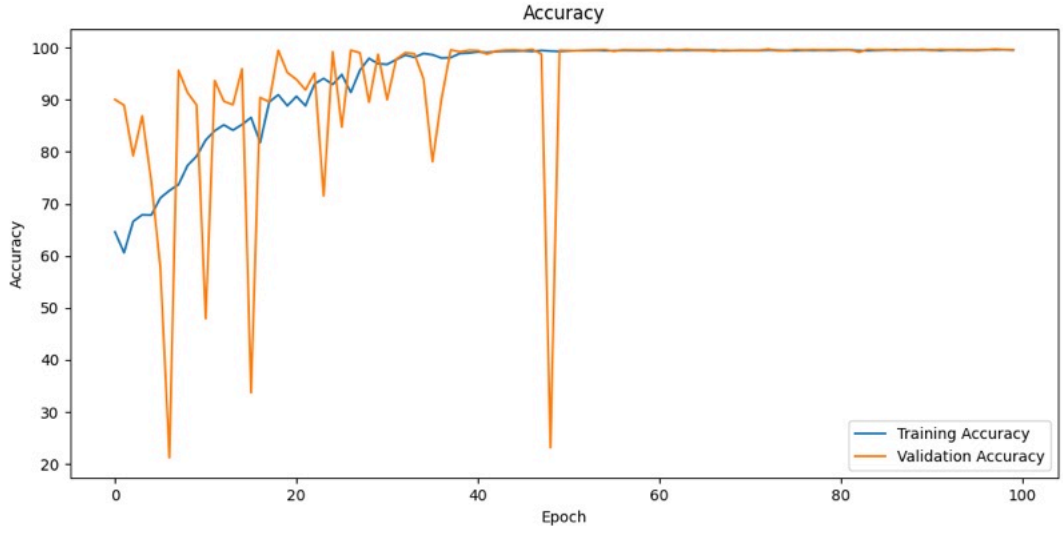

Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100

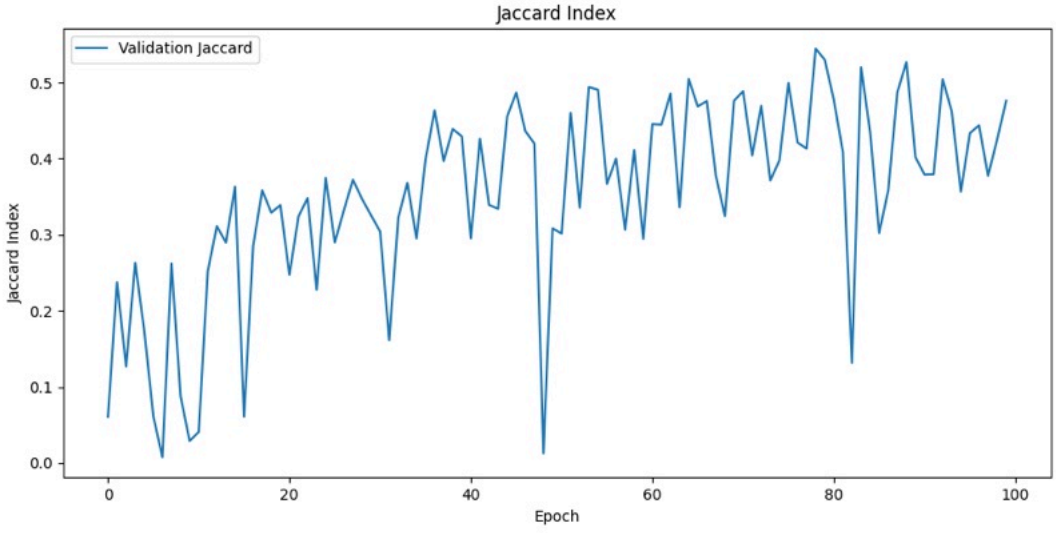


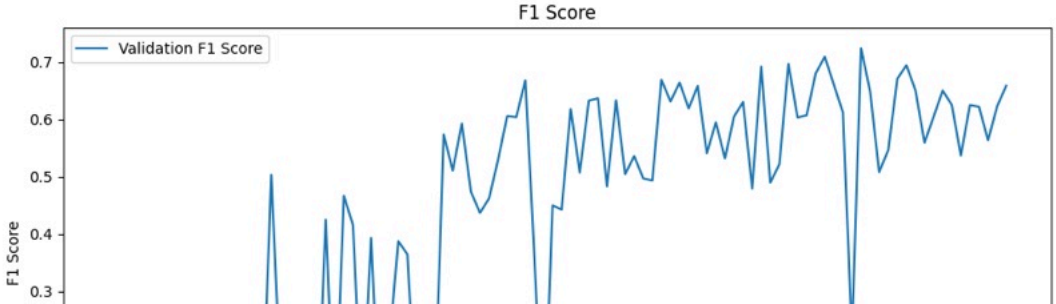
The image just above this paragraph shows a few issues with the model. It “triggers” on dry river beds and underestimates the length of the airstrip.

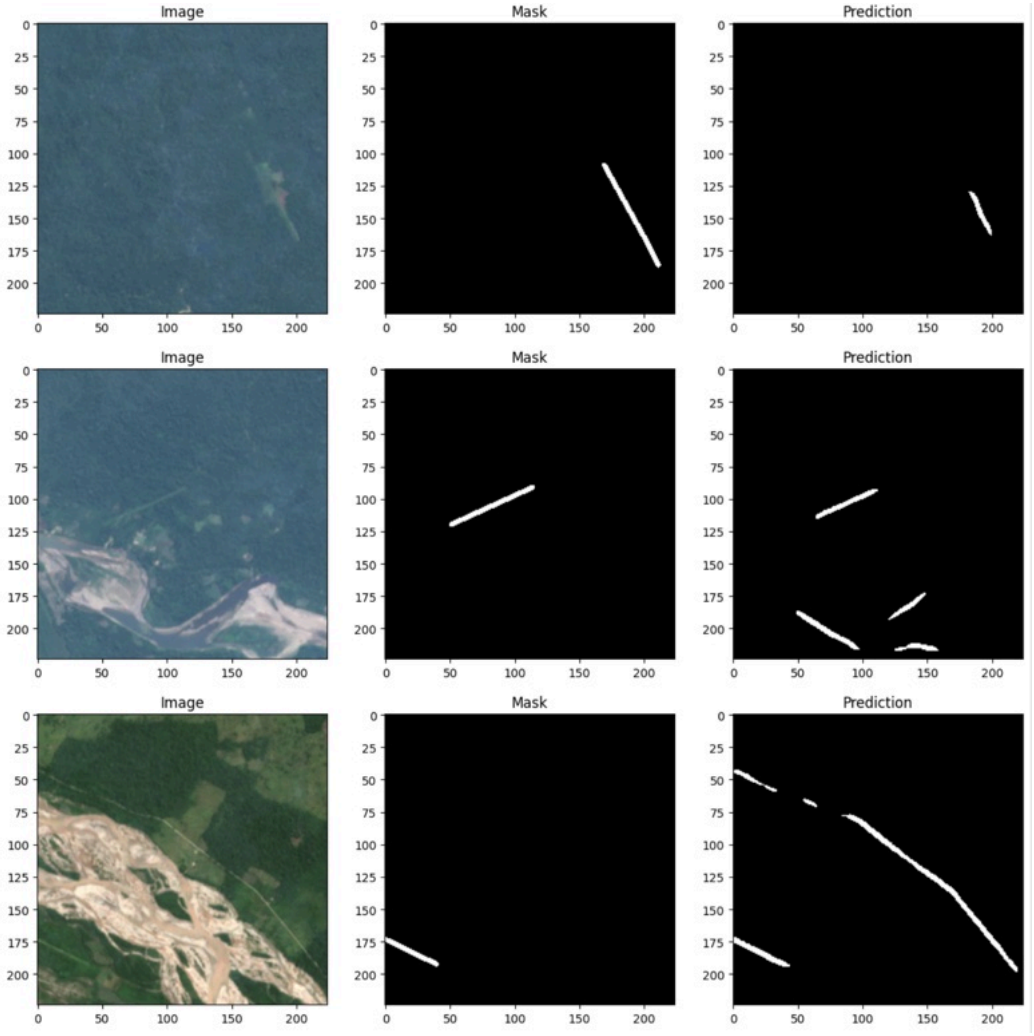
Now on testing with a **20 meter** buffer.


Loss
Training Loss
Validation Loss
Loss
Epoch

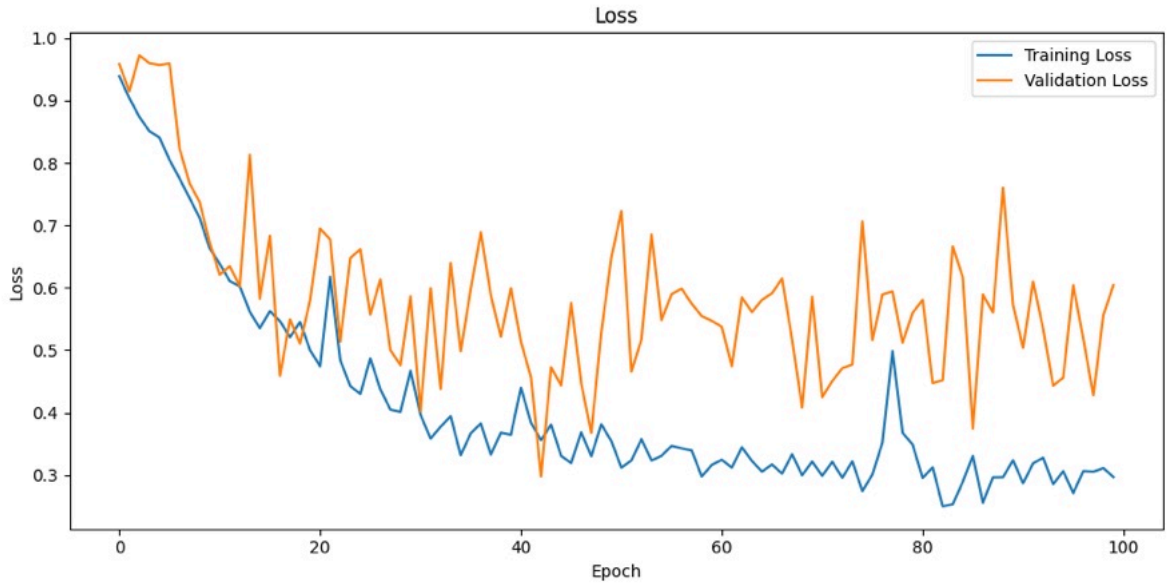

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch

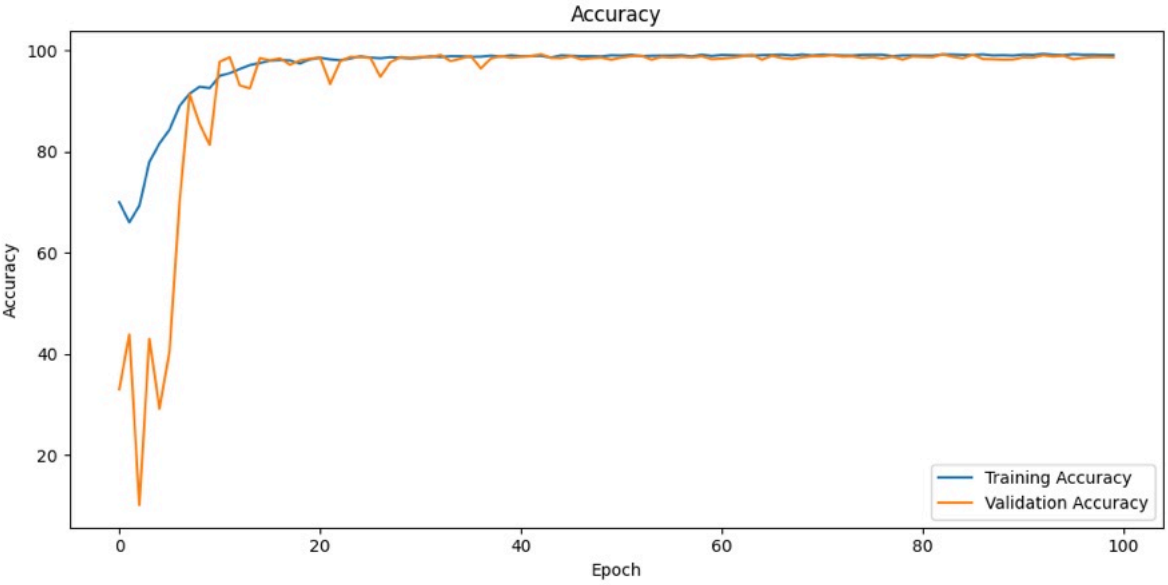

Jaccard Index
Validation Jaccard
Jaccard Index
Epoch

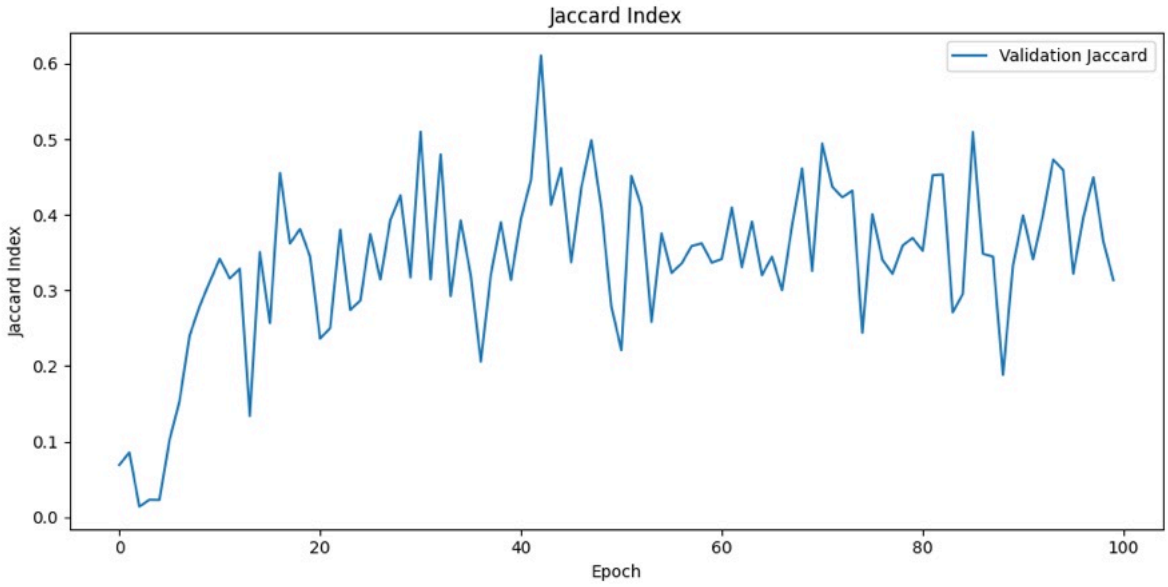

F1 Score
Validation F1 Score
F1 Score

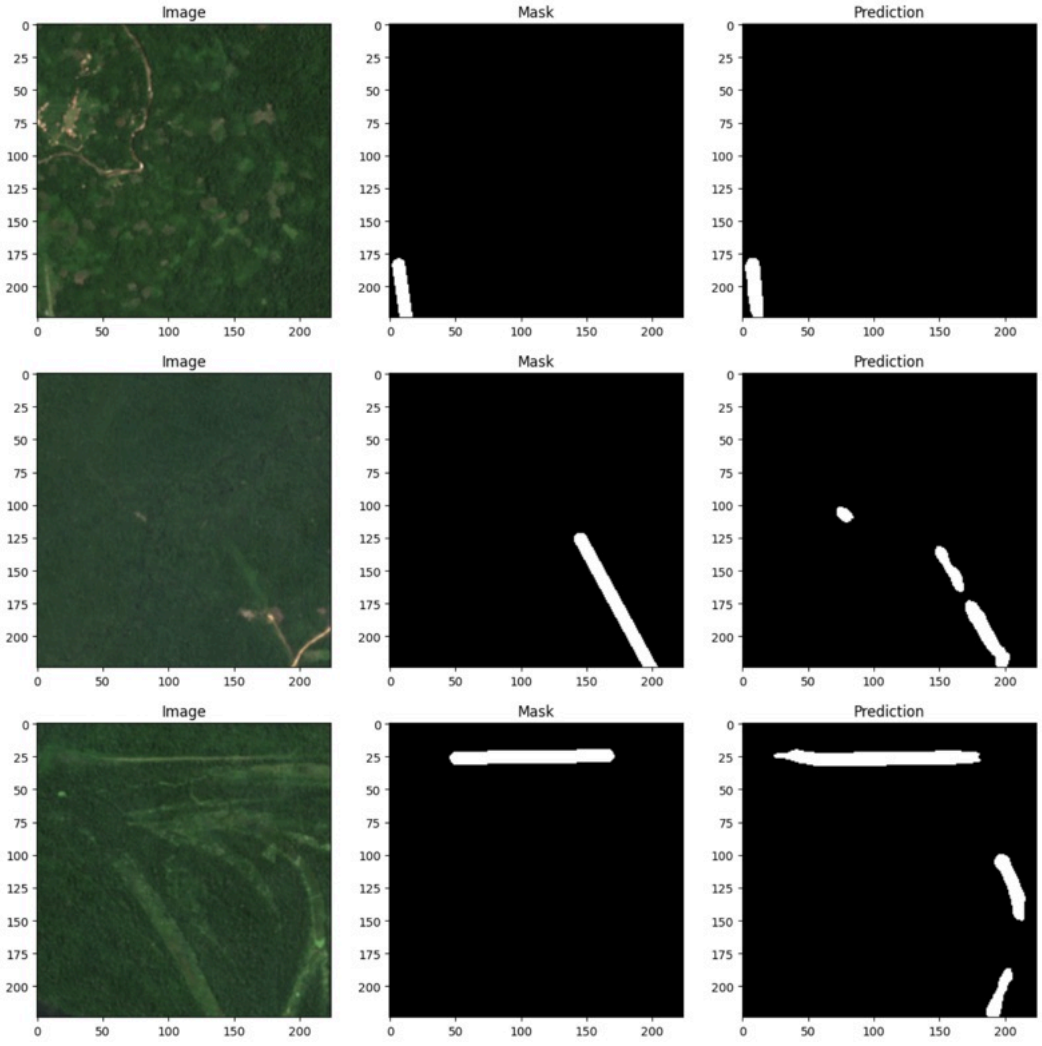


Now on testing with a **50 meter** buffer.

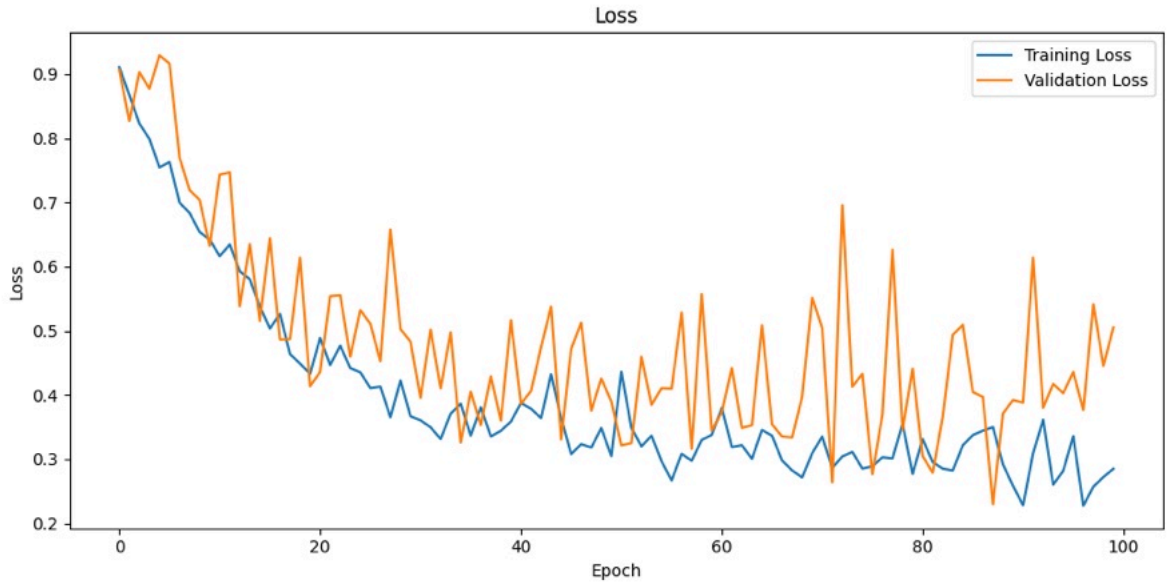

Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100

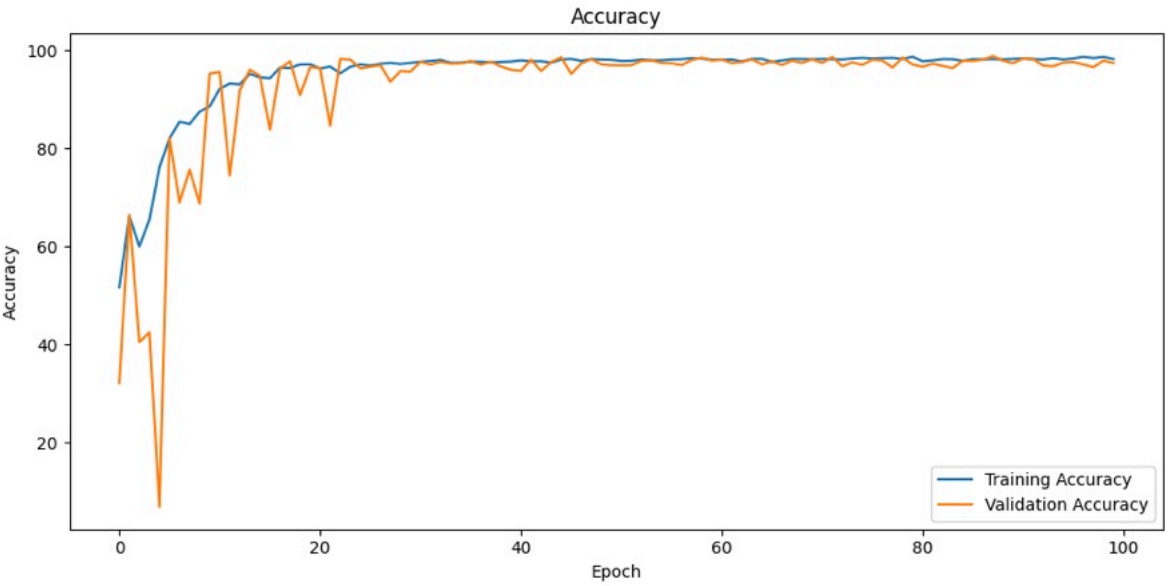

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
20
40
60
80
100

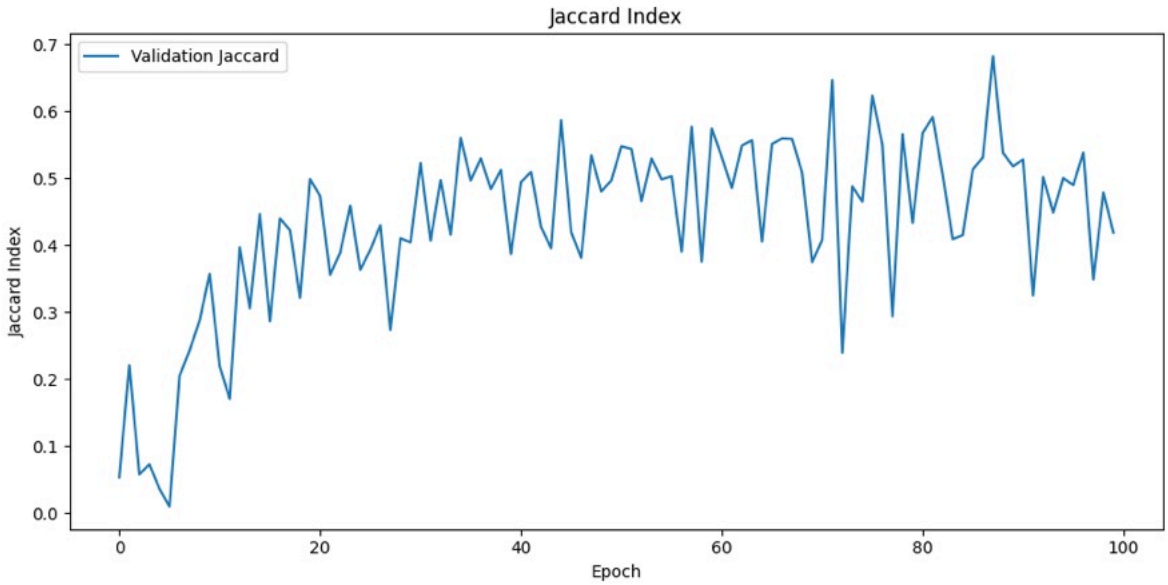

Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

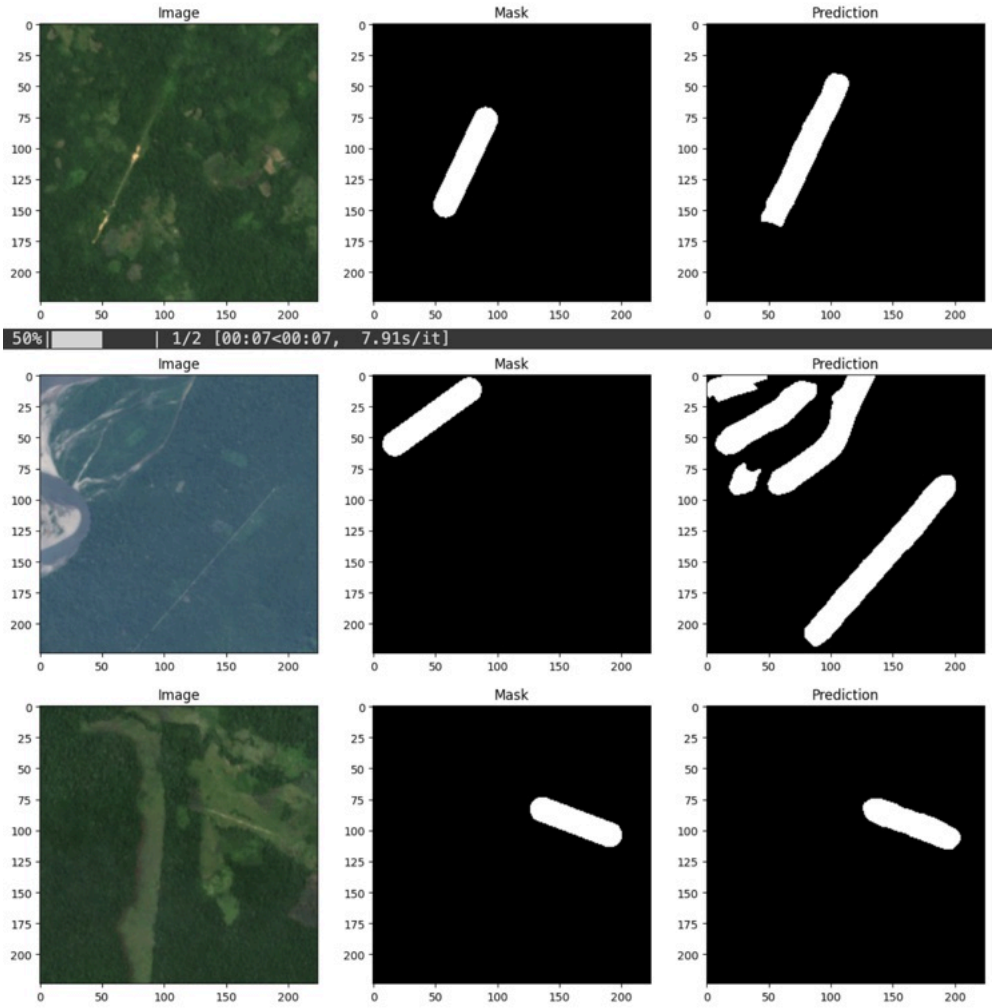


Now on testing with a **100 meter** buffer.

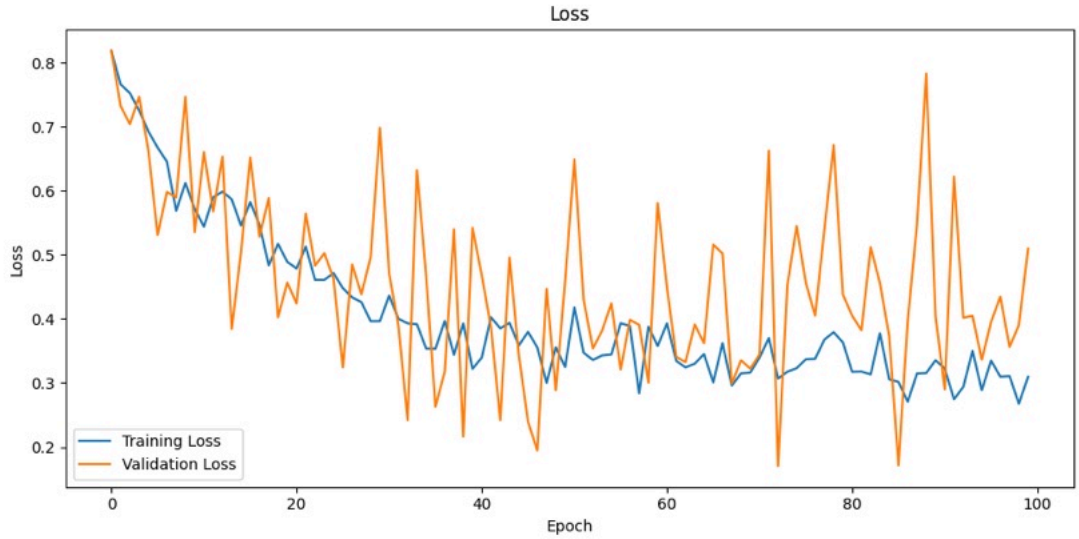

Loss
Training Loss
Validation Loss
Loss
Epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100

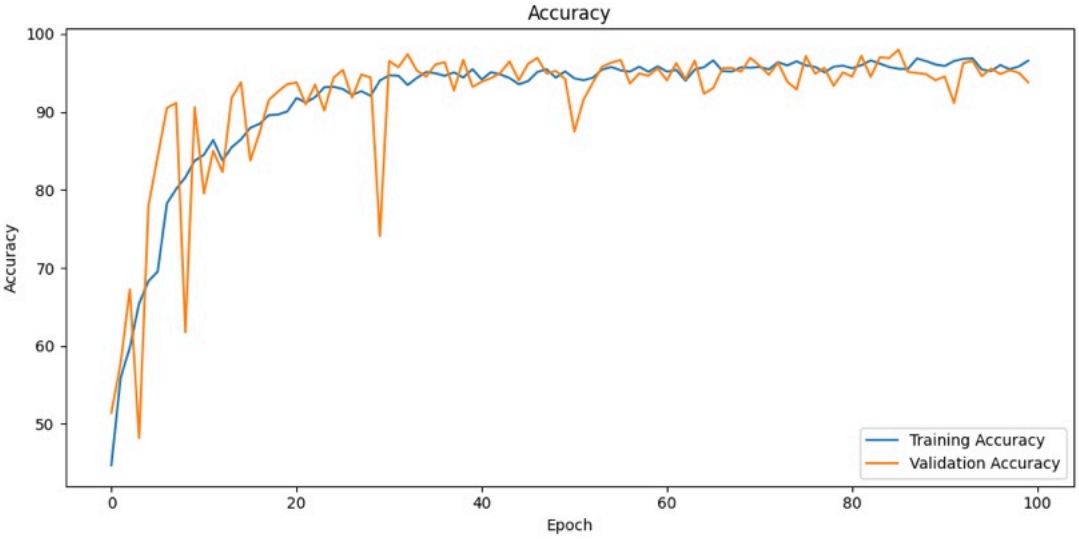

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
20
40
60
80
100
0
20
40
60
80
100

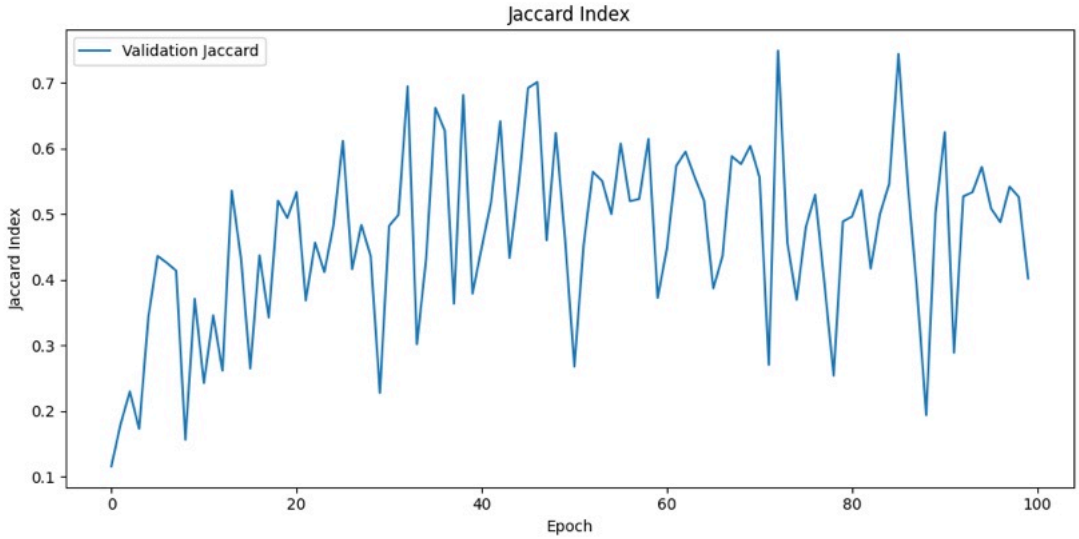

Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100

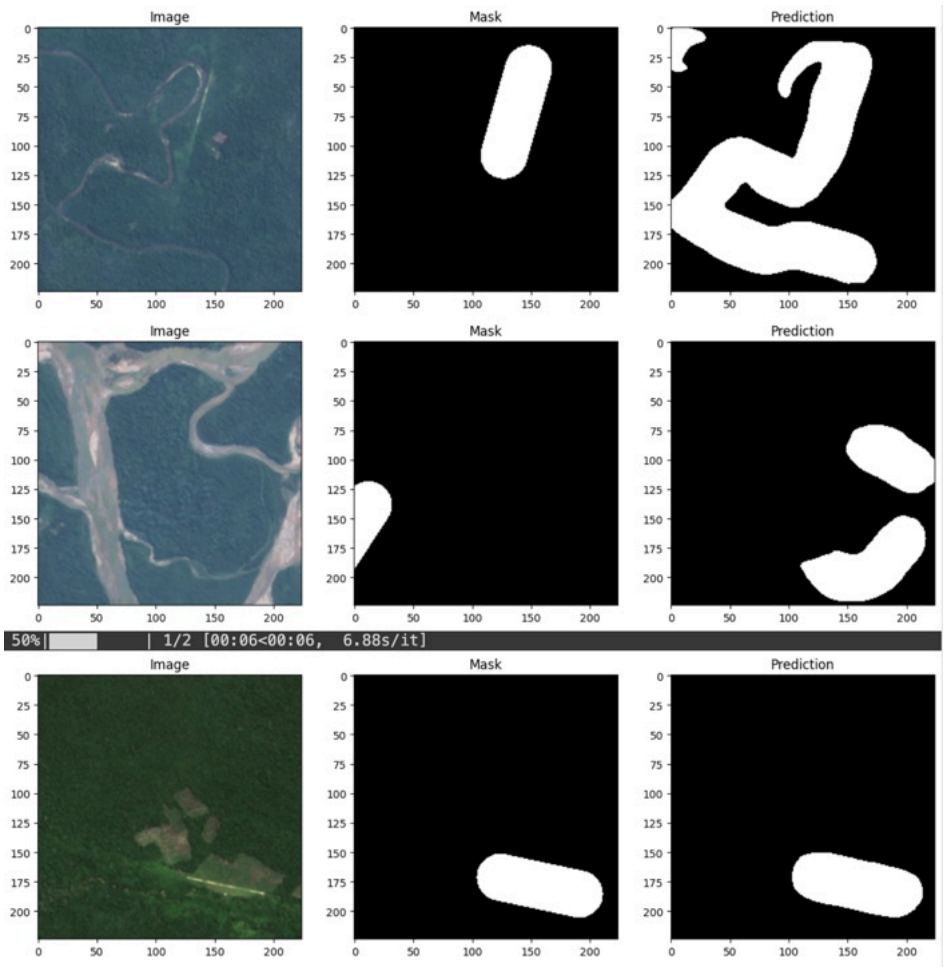


Now on testing with a **200 meter** buffer.


Loss
Loss
Epoch
Training Loss
Validation Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100


Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy
100
90
80
70
60
50
0
20
40
60
80
100


Jaccard Index
Jaccard Index
Epoch
Validation Jaccard
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100



Visualizing the model predictions for the entire dataset

The predictions were exported to animated GIF files and stored on the shared Google Drive. Comparing the results side by side shows that the larger masks (100m - 200m) do not perform well. There is no easy way to select the optimal mask width, but it looks like the 20-meter buffer is a good compromise. This also corresponds to the expected physical width of a typical airstrip (~40 meter wide) and matched the visual information on the Sentinel images. We decided to select the **20m buffer** to complete the coding to get inference, image cleaning, and the scoring submissions.

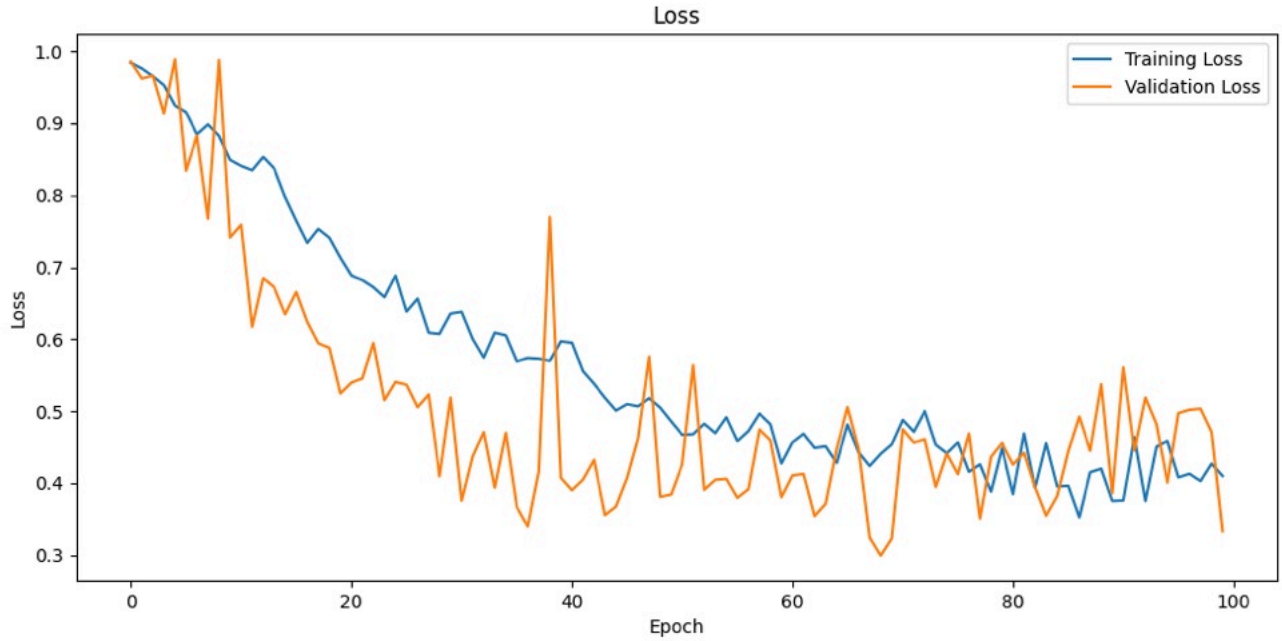
Multispectral modeling

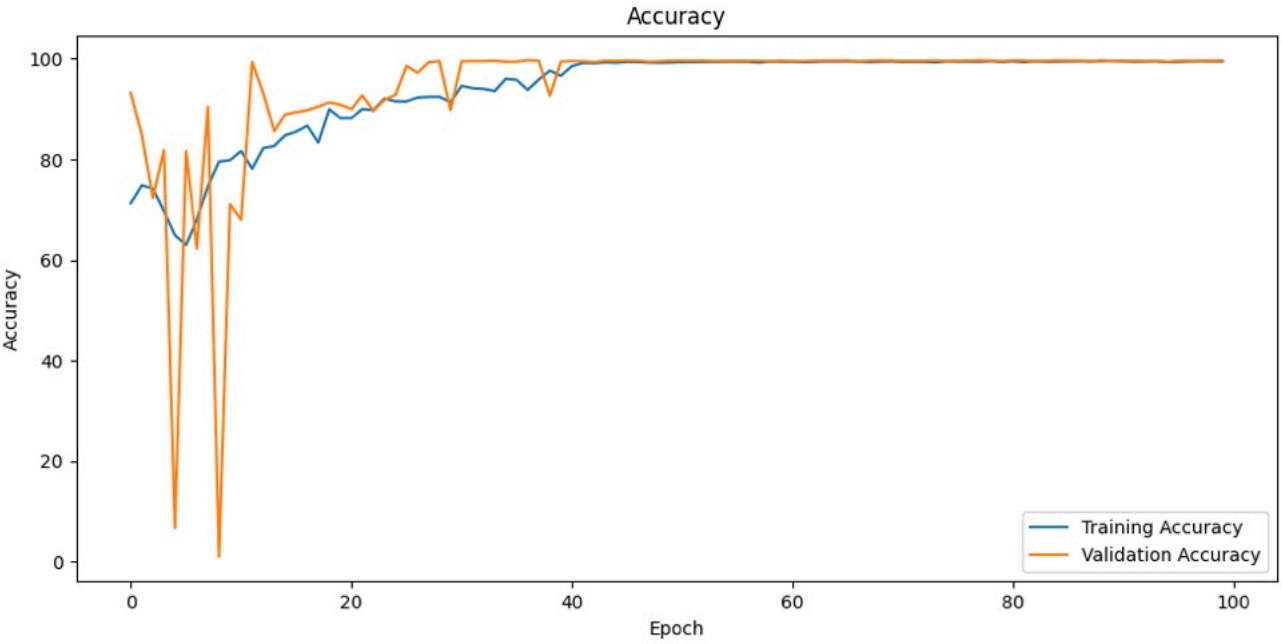
The model supports additional bands in addition to the standard RGB bands. The `Dataset` class and the image augmentation function include an optional parameter to extract and normalize additional bands from the GeoTIFF files. The model definition also includes a parameter to specify the number of input channels. It is critical to make sure the optional parameters are consistent. Additional bands require additional parameters in the model and those parameters are initialized with random values. Adding a single band (e.g. VV) leads to the following specs for the model. Notice an increase of 3136 trainable parameters compared to the RGB model.

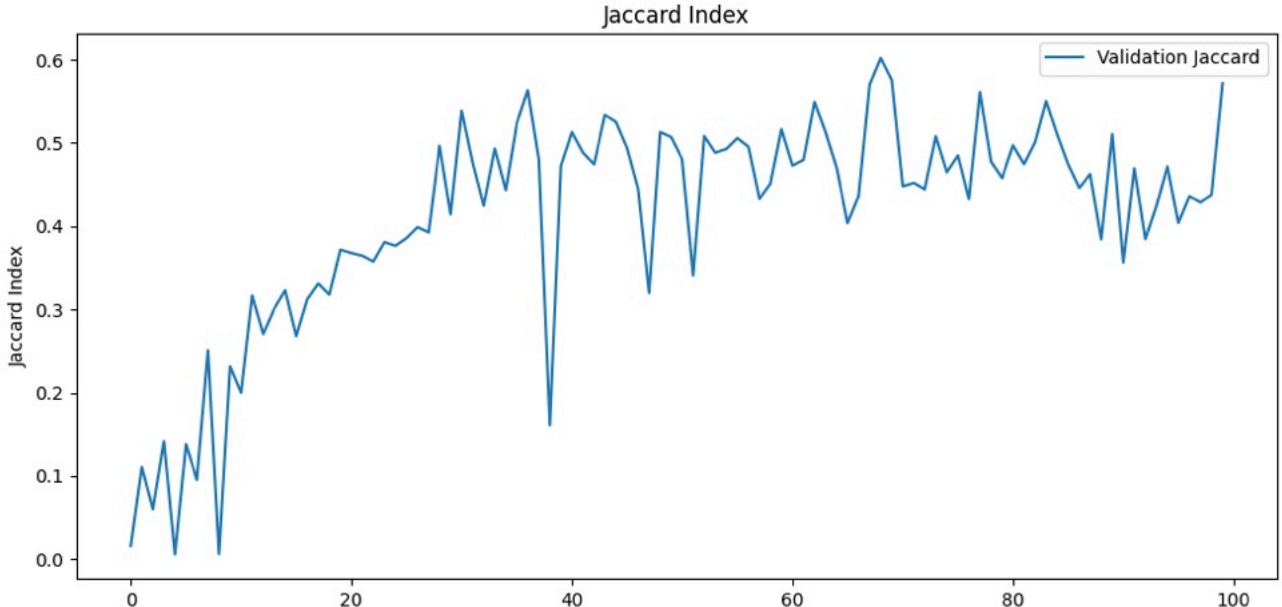
```
==========================================================================
Total params: 32,524,241
```

```
Trainable params: 32,524,241
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.77
Forward/backward pass size (MB): 463.01
Params size (MB): 124.07
Estimated Total Size (MB): 587.85
----------------------------------------------------------------
```

The first iteration of training with the optional Sentinel 1 **VV** band and a 20 meter mask buffer is summarized in the following plot. It is not clear why the validation loss is lower than the model loss for most of the training. The *best model* is obtained after 68-70 epochs as the selection is based on the lowest value of loss for the validation set. This model also has the maximum value for the Jaccard Index and F1 score.

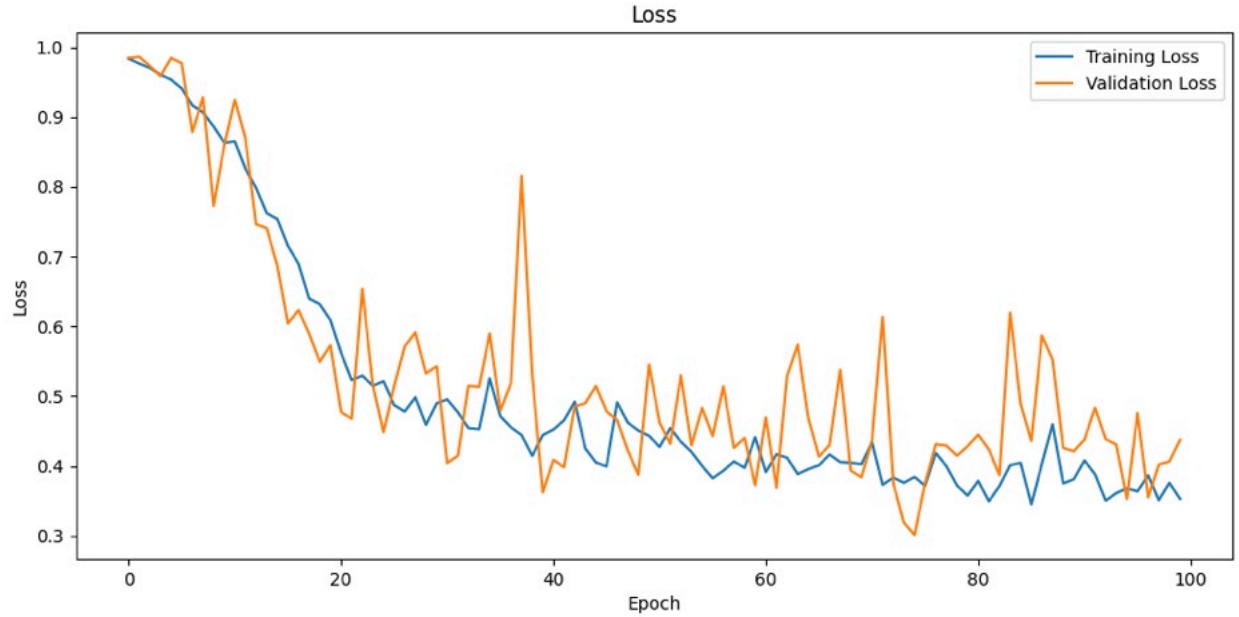

Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100

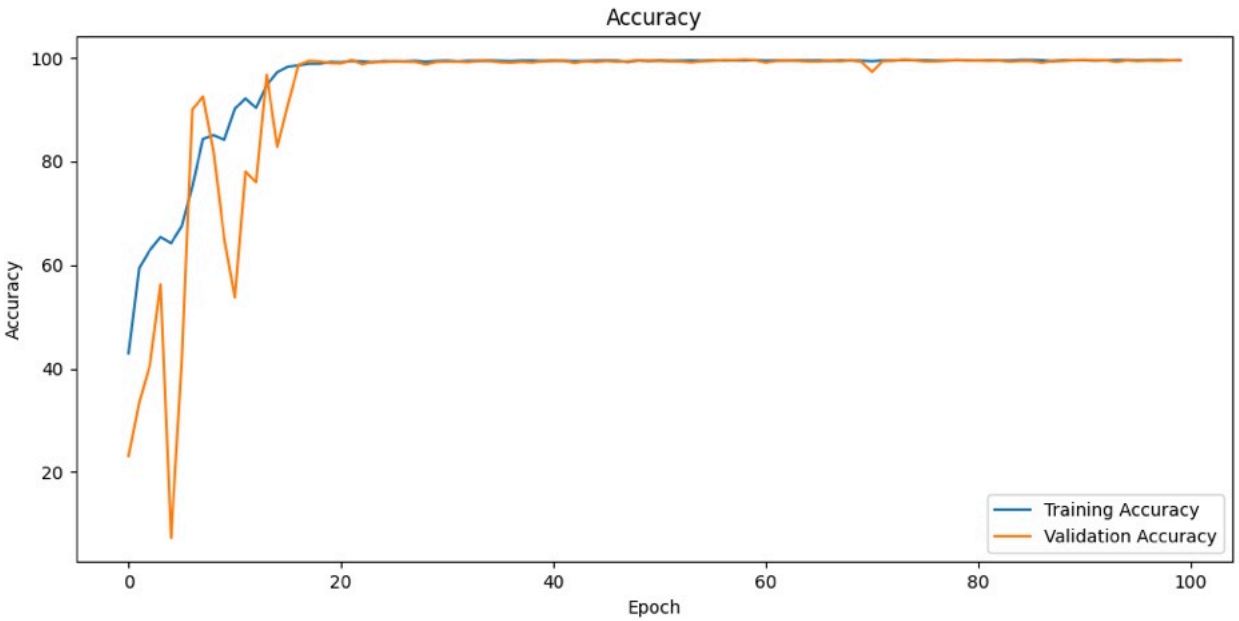

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
0
20
40
60
80
100

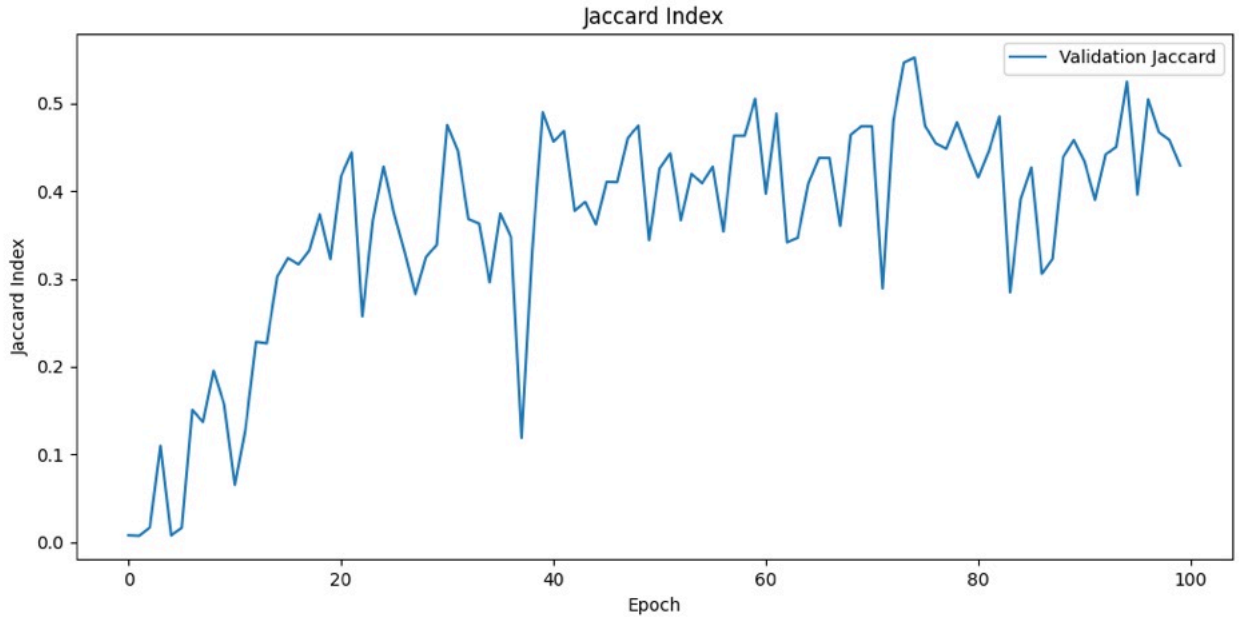

Jaccard Index
Validation Jaccard
Jaccard Index
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

Comparing the model predictions with the standard 3-band model does not appear to yield a significant improvement. This is a qualitative assessment from side-by-side visualization of the RGB image, the mask, and the prediction.

Replacing the **VV** band with the **NIR** band (B8) yields the following results. The visual inspection of the side-by-side predictions is really interesting. In some cases it appears like a combination of the models with VV and NIR bands would yield a better agreement with the ground truth.

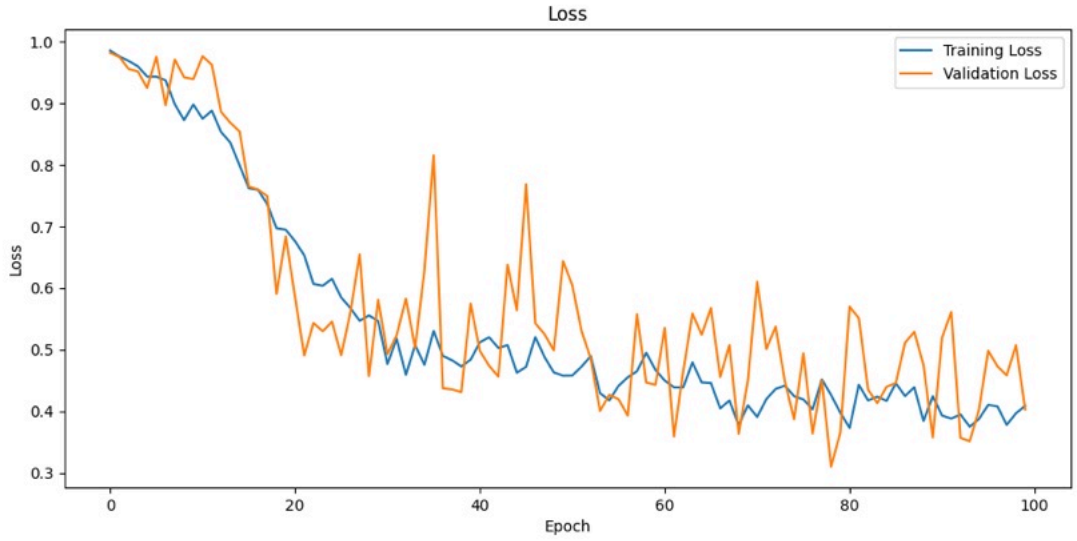

Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100

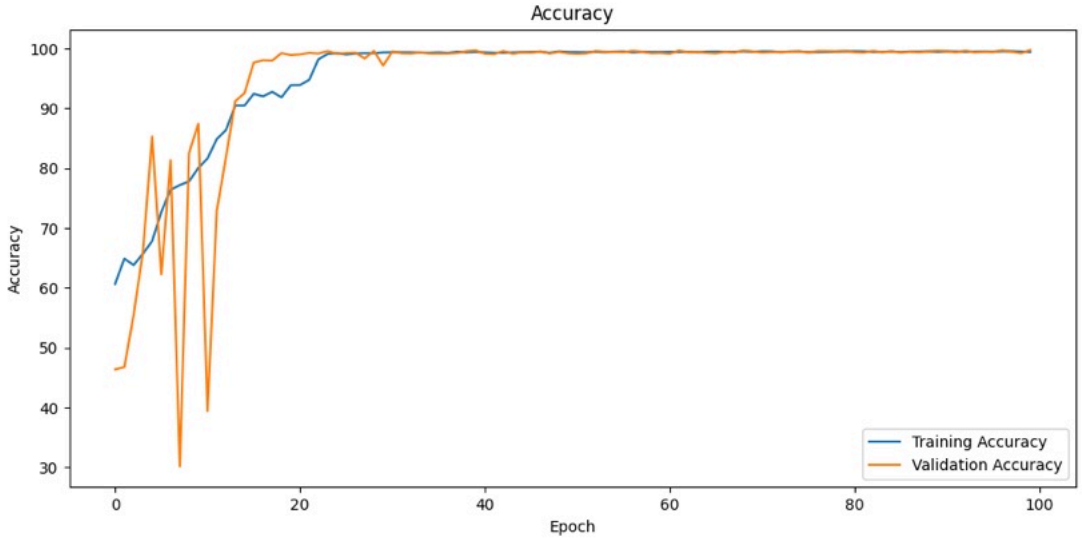

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
20
40
60
80
100

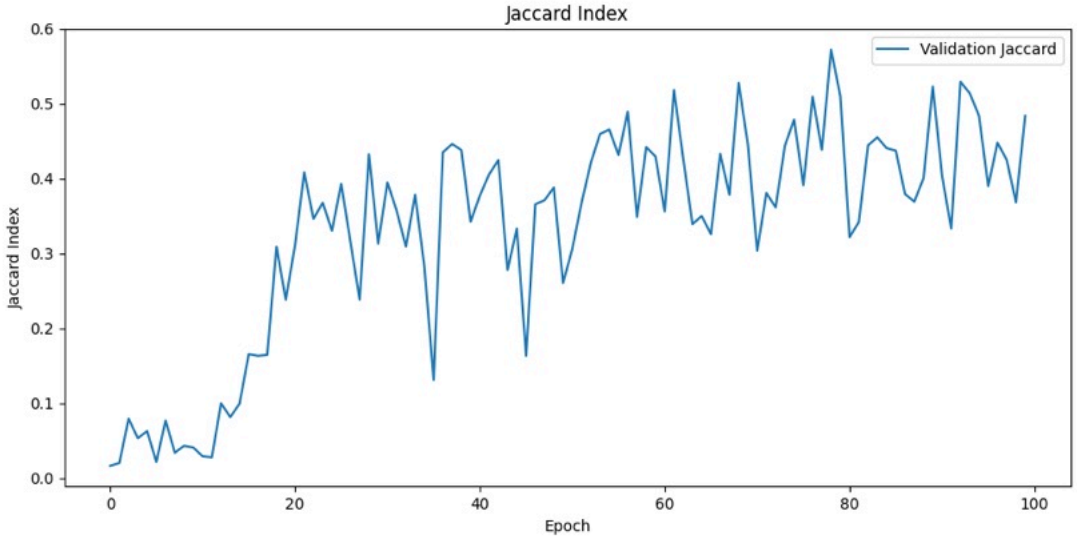

Jaccard Index
Validation Jaccard
Jaccard Index
Epoch
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

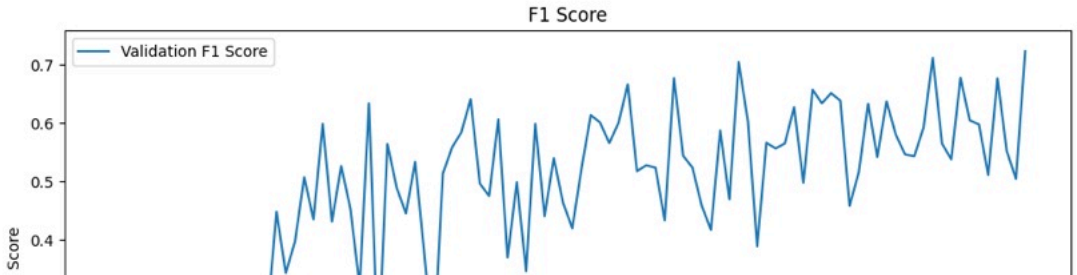
F1 Score

Now testing the model with the **NIR** and **VV** bands. This is the model that was selected for all scoring submissions.

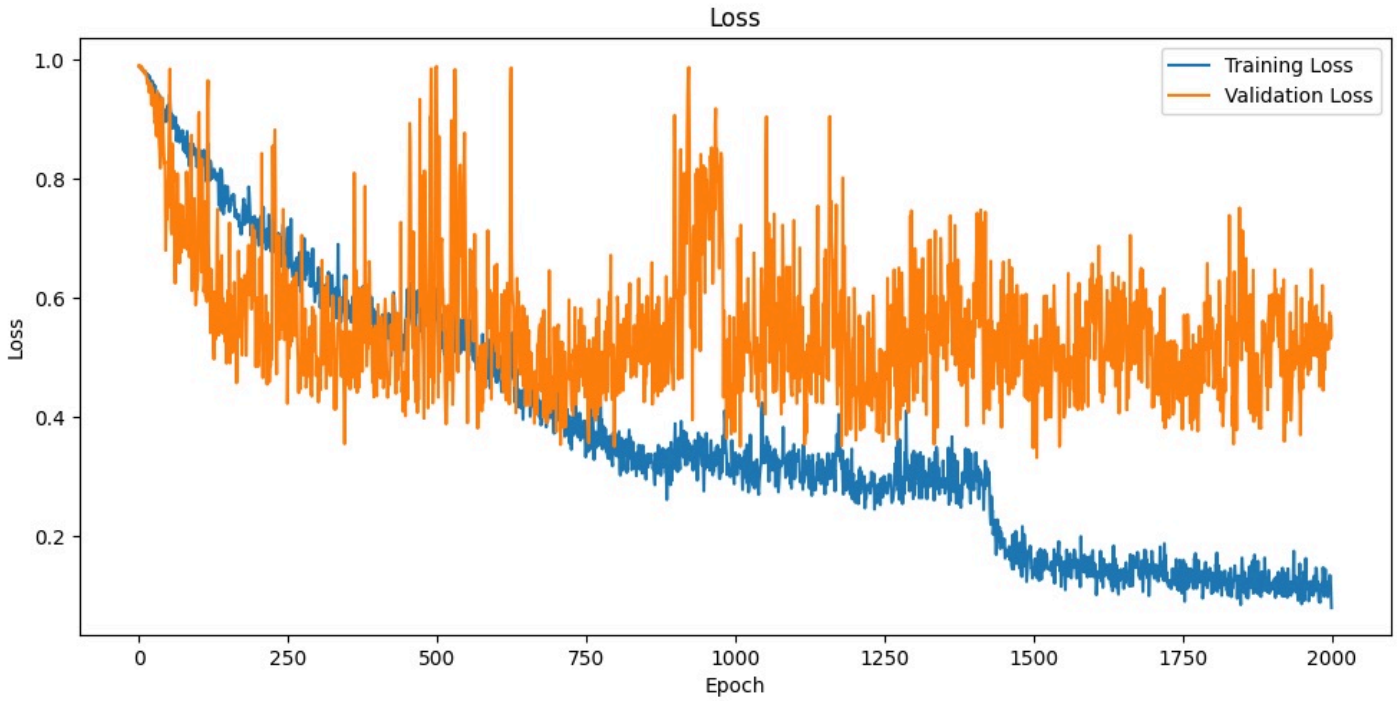

Loss
Training Loss
Validation Loss
Loss
Epoch

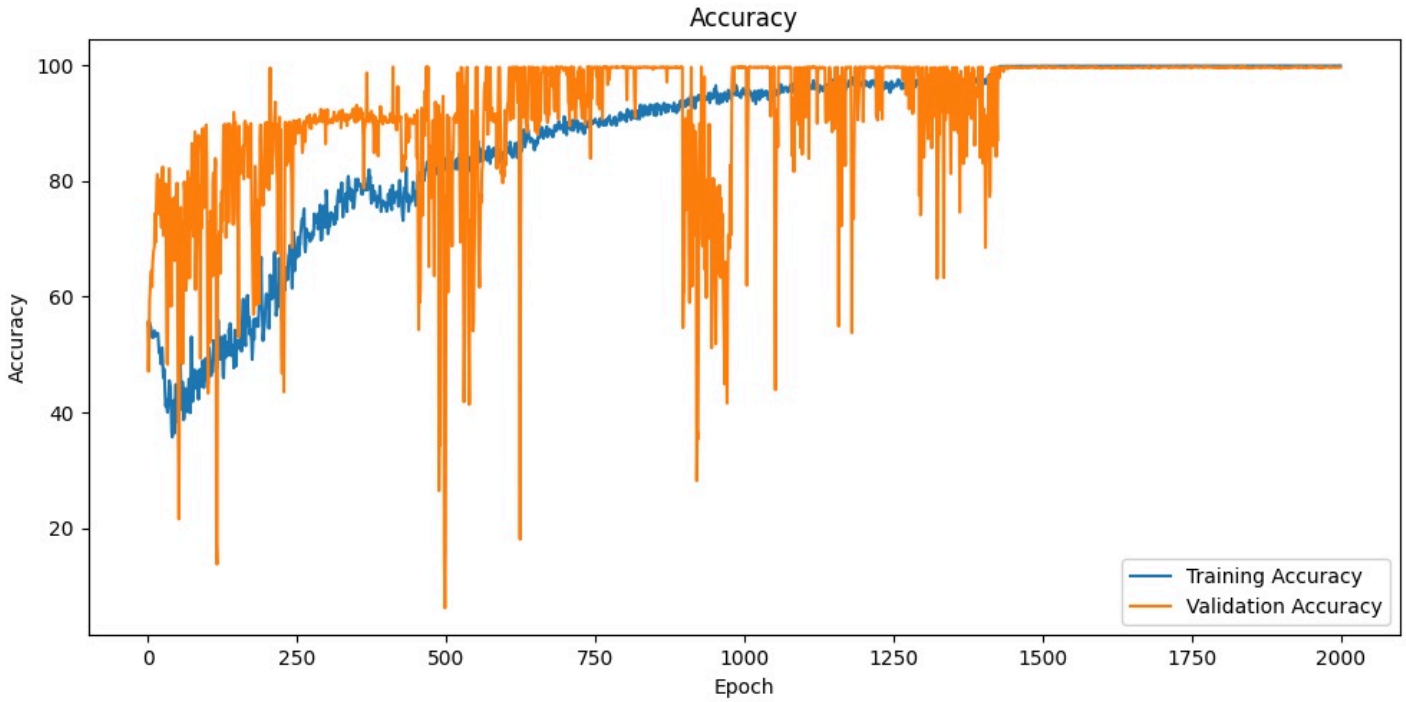

Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch

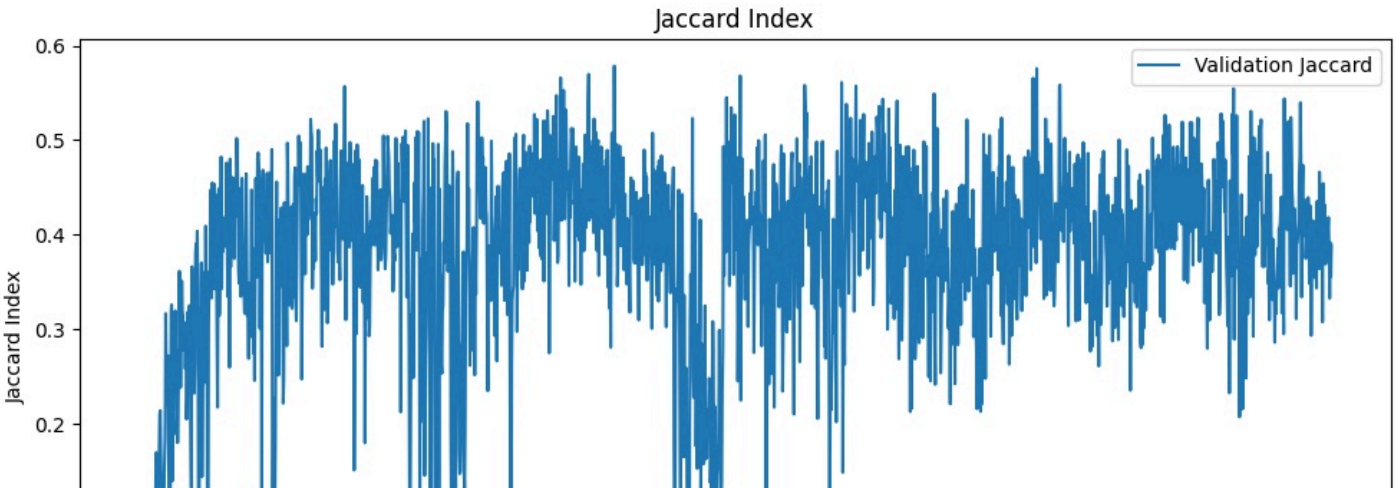

Jaccard Index
Validation Jaccard
Jaccard Index
Epoch


F1 Score
Validation F1 Score
Score

A final attempt was made to train the model with lower values for the learning rates (1e-4 and 2e-5) and include the weight decay parameter (5e-4). The next image shows the output from the training run. Around epoch 1400, a sudden transition is visible and the loss value drops abruptly. This model appears to be overtrained and it learned the training set. The best model is defined as the model with the lowest loss value on the validation set. This is probably not the best way to do it, but that's how it is defined in the code. The best model was saved to file and new predictions could be made and compared to the last set.


Loss
Training Loss
Validation Loss
Loss
Epoch
1.0
0.8
0.6
0.4
0.2
0
250
500
750
1000
1250
1500
1750
2000


Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
100
80
60
40
20
0
250
500
750
1000
1250
1500
1750
2000


Jaccard Index
Validation Jaccard
Jaccard Index
0.6
0.5
0.4
0.3
0.2

Inference

The code splits the AOI image into non-overlapping (224px x 224px) tiles, runs the model on each tile, and stitches the outputs into a composite image.

Segmentation / tiling of test areas

Using the 224×224 dimension of the input layer, we can fit 6×6 non-overlapping tiles, and need a solution to cover the edges of the area that are excluded. Padding the edges of the image with 0s solves this issue. The main disadvantage of using non-overlapping tiles is that airstrip can be truncated between tiles if the model doesn't detect the pixels on the edge of the image, or if there is a small gap at the edge. This is something that was seen in the output files.

A better solution would be to overlap the tiles and use a custom blending algorithm to prevent truncation at the edge of the tiles. Unfortunately we ran out of time to implement this solution.

Image Cleaning

The final model was trained using masks with a padding of 20m on all sides. During inference, the model is expected to identify airstrips with a cluster of pixels with about the same width (~2 x 20m = 40m). The objective of the challenge is to identify the airstrips and to provide information about their locations and dimensions. The instructions from the organizers are to locate the airstrips, create a 200 meter buffer in all directions around the airstrips, and report those larger areas for scoring.

The clusters of pixels produced by the model are the starting point for this final step. The predictions on the test set are pretty good but errors are visible when the model predicts airstrips on dirt roads (common). Visual inspection of those false detections is quite revealing. In many instances, the section of road is much shorter than a typical airstrip, or the road segment shows significant curvature. Can we use this information and somehow reject those false detections?

All airstrips are straight and narrow lines as seen in the Sentinel data. This can be used to discriminate against other types of surface features falsely identified by the model. Using statistics from the training set, the shortest airstrip is 376 meters (~38 pixels) with a width of ~40-50 meters (4-5 pixels).

As of this writing, the following approach was implemented to clean the images and reject false detections.

1. Identify all the clusters using the *scikit-image* package and loop over each cluster.
2. Clusters with a single pixel are excluded right away.
3. Use 2-component Principal Component Analysis (PCA) to get the principal axis of the cluster. This is equivalent to fitting a straight line through the cluster.
4. The output of the PCA can be used to calculate the length of the principal axis (length of the airstip), the orthogonal variance (i.e. width), and the coordinates of the start and end points of the line.
5. Using the length and width of the line, a filter can reject clusters that do not meet selection cuts. At the time of this writing, the filter rejects objects narrower than 30 meters or wider than 70 meter wide, and less than 350 meters IF they do not intersect the image boundary (truncation). See distributions below.
6. For the remaining clusters, using the coordinates of the end points (in pixel space), the CRS of the region and the bounding box coordinates, the coordinates of the end points can be converted into geographic / projection space.

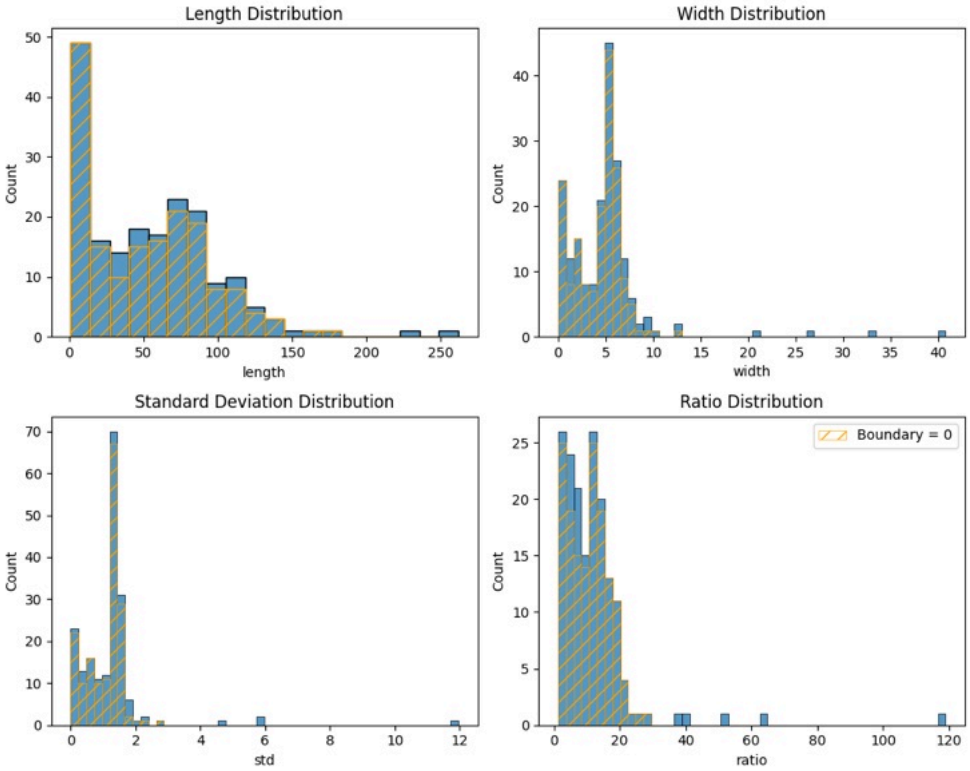
7. Using the coordinates of the end points, a `LINESTRING` GeoJSON object is created.
8. A proper `POLYGON` object is created by adding a 20m buffer around the `LINESTRING` object to define the airstrip, and a second `POLYGON` object is created with a buffer of (20 + 200) meters to define the background area for scoring.
9. Using the `POLYGON` objects, a cleaner prediction map is produced using the same bonding box and CRS and saved to a GeoTIFF file.

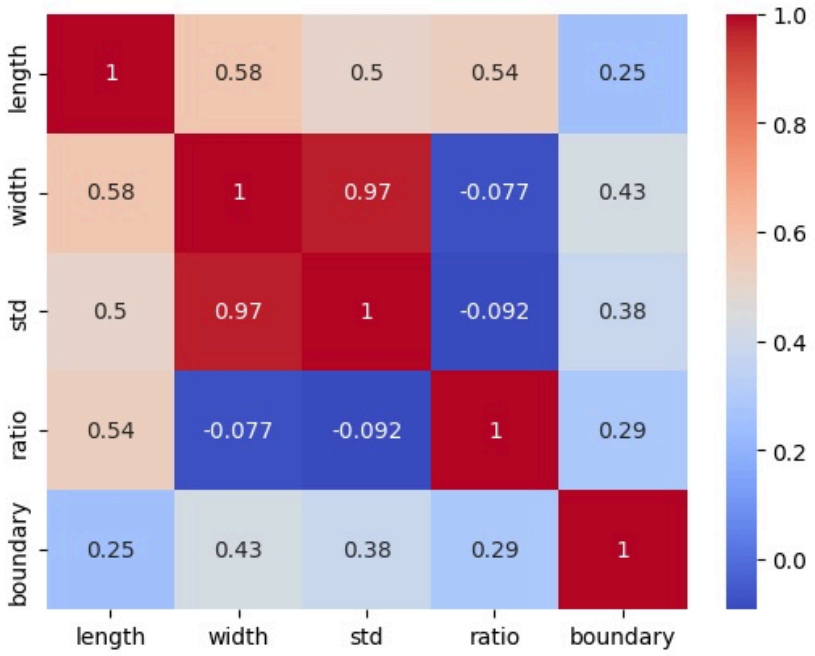
Looking at the clusters of pixels, we can analyze the parameters calculated with the PCA. The following plots highlight a few statistical distributions. The first figure shows the length, width, standard deviation, and length / width ratio distributions. The dark blue histograms include data from all clusters. The dashed histograms do not include clusters that intersect with the edge of the image. The length distribution shows a large number of clusters with relatively short lengths. Visual inspection of the predicted masks reveals that those short clusters are usually associated with short road segments, not airstrips. Given that all airstrips in the training sets were above 350 meters in length (35 pixels), we can safely cut / reject those clusters during the image cleaning process. Similarly, the width histogram shows a significant number of clusters with very narrow width (0-3 pixels). A few of those clusters are right at the edge of the image and appear to be artifacts of the model.

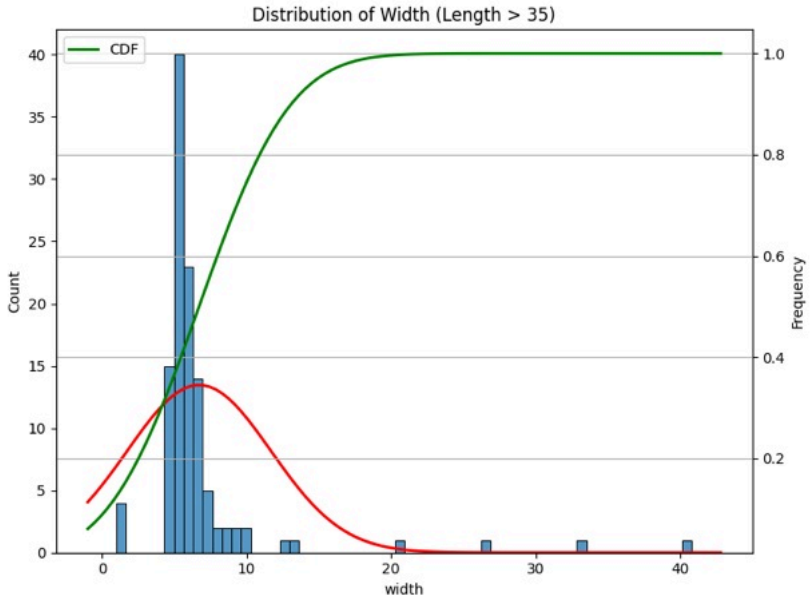
The second plot shows the correlation between the different variables. The width and standard deviation are nearly perfectly correlated (expected). Only one variable is necessary for rejecting clusters and we'll use width moving forward.

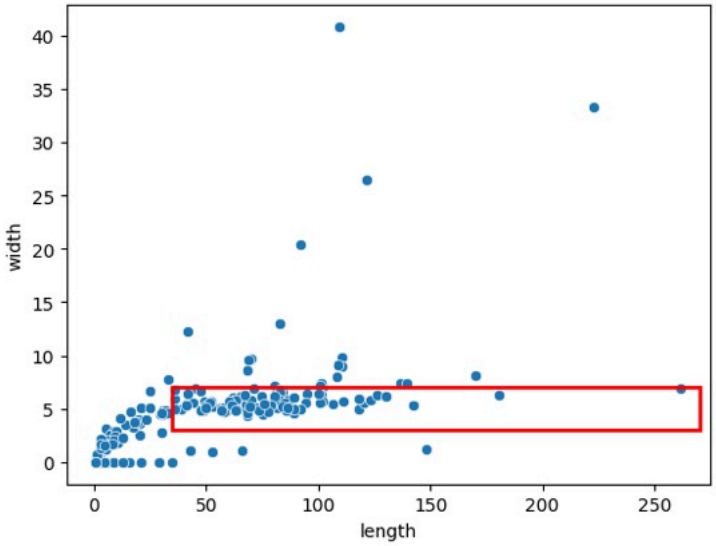
The third plot shows the distribution of cluster width after filtering for a length of more than 35 pixels (350 meters). The peak of the distribution is very narrow, which is consistent with the uniform width of airstrips. Outliers on both sides of the peak are either artifacts on the edge of the image (~ 1 pixel wide), or roads / rivers with some curvature. A selective cut on the width parameter should prove useful with removing artifacts and a subset of roads mis-identified by the model.

The fourth plot is a scatter plot of width versus length. The red rectangle is a selection cut that would remove short clusters and clusters that are too narrow or wide, i.e. `[length > 35 AND 3 < width < 7]`.


Length Distribution
Count
length
Width Distribution
Count
width
Standard Deviation Distribution
Count
std
Ratio Distribution
Boundary = 0
Count
ratio


length
width
std
ratio
boundary
1
0.58
0.5
0.54
0.25
0.58
1
0.97
-0.077
0.43
0.5
0.97
1
-0.092
0.38
0.54
-0.077
-0.092
1
0.29
0.25
0.43
0.38
0.29
1
1.0
0.8
0.6
0.4
0.2
0.0





The number of road segments falsely detected as airstrips is quite significant in this dataset. Retraining the model with a lot more regions with roads is not quite realistic at this time (less than a week left). A more efficient approach is to use Open Street Map (OSM) data to identify the roads and eliminate predicted airstrips that intersect with road segments. The road segments are `LINESTRING` objects and we already wrote code to check for intersecting objects when exploring the Sentinel data. The next picture shows the remaining airstrips (bright white areas) after checking for intersections with OSM. This solution is very efficient at this late stage of the competition and was implemented as part of the image cleaning process.

The last step is to open the prediction map and extract the pixels with a value of 1 (airstrip) and 0 (background) and store them in a Pandas DataFrame. The DataFrame is then exported to a CSV file for submission to the competition.

Submission format and scoring

This is the summary of the submissions to the Zindi data challenge. All files were saved on the Google drive for reference.

Filename	Score	Comments

output_file_B4_B3_B2_B8_VV_20241105_1015MST.csv	0.000765601	This is the first submission. Clearly something is wrong.
output_file_B4_B3_B2_B8_VV_20241105_1040MST.csv	0.047901837	Using the same file, but setting all pixel values to 0, instead of 1. The score increased significantly. Our previous understanding that all pixels in the 200 meter buffer should have values of 1 was incorrect. The logical conclusion is that the airstrip should be labeled with 1s, and the 200 meter buffer around the airstrip should have values of 0. The image cleaning code must be updated.
output_file_B4_B3_B2_B8_VV_20241105_1505MST.csv	0.043887601	Several modifications to the image cleaning code. The linestring (cluster fit) is first used to create a “signal” polygon with a 20 meter buffer with value 1. It is then converted to a “buffer” polygon with a 200 meter buffer with value 2. The value 2 is required to show it above the background value of 0 for the mask. The value 2 is later converted to a value of 0 for submission.
output_file_B4_B3_B2_B8_VV_20241105_1555MST.csv	0.395836781	Flipped columns and rows.
output_file_B4_B3_B2_B8_VV_20241106_1415MST.csv	0.398630193	Optimized the cut on cluster width (min and max value). Marginal gain.
output_file_B4_B3_B2_B8_VV_20241106_2000MST.csv	0.496751365	Remove airstrips that overlap with roads.
output_file_B4_B3_B2_B8_VV_20241107_1115MST.csv	0.554461181	Made the buffer 220 meters (airstrip is 20m buffer around LINESTRING).
output_file_B4_B3_B2_B8_VV_20241108_0955MST.csv	0.54483943	10 meter buffer for airstrip
output_file_B4_B3_B2_B8_VV_20241108_1000MST.csv	0.559530706	30 meter buffer for airstrip
output_file_B4_B3_B2_B8_VV_20241110_1745MST.csv	0.43339265	Model with lower learning rate and 2000 epochs. Overtrained!
output_file_B4_B3_B2_B8_VV_20241111_1125MST.csv	**0.580967555**	Square ends on the buffer around the aristrip. I.e. increase area of background region.

Usage Instructions

Data Preparation

Run `Data_Visualization/explore_sentinel_data.ipynb` to download satellite data for training and inference. Make sure you have ~5GB of space for data storage.

- Open the table of content on Google Colab to see the main sections of this code.
- The first two sections install and import packages required for this project. Your Google Drive file system is mounted and linked to the computer node. ALL the GeoTIFF data will be downloaded to your Google drive.
- The third section authenticates with your Google Earth Engine. You need to replace the project name in the code cell to match your account! Google Earth Engine is required to download Sentinel data for training and of the test AOIs.
- The next section contains many functions to interact with or download Sentinel data. Run all cells in that section.
- The section **Sentinel data** defines the bands we’ll be downloading and loads the airstrip training shape file provided by the Zindi team.
- The last section is where all the data are downloaded. First the test AOI files, then the airstrips images used for training. In both subsections you must edit the name of the Google Drive folder where the

data will be stored. We suggest using different directories for the test AOI and the images for training. Once the Google Drive folders are set, you can run the cells. Notice that for each item in the for loop, a task will be created on Google Earth Engine. The loops will execute quickly, but the tasks will get queued on the server. You should monitor the tasks using your Google Earth Engine account.

- It takes about 10 minutes to generate one image for a test AOI and save it to your drive. There are 11 of those. It takes about a minute to generate an image for an airstrip, there are 108 of those in our training set. Google Earth Engine can usually process two tasks in parallel, so it will take a few hours to generate the full set of training and test AOI data. Be patient and monitor the status of the tasks on GEE.

Run `utils/GeoTIFF_modifications.ipynb` to remove border columns with NaN values for the airstrip training images. We are not sure why this doesn't impact the test AOI images, but this is an issue we detected in the GeoTIFF files. The images should be 512px x 512px, but for a subset of airstrip images an additional column is added and filled with NaN. It should be possible to fix this issue.

- The Python notebook requires access to your Google drive. You need to edit the path to the images and the script will do the rest.
- Notice that the corrected images are saved on the local computer node. The corrected images use the same name as the original files. You need to download the new images and decide how to overwrite the original files.
- Using the images with NaN will cause the model to stop executing as NaN values are not valid inputs, so don't skip this step.

Run `utils/Generate_airstrip_masks.ipynb` to generate the masks for training. The default mask buffer width is 20 meters, which corresponds to the value we used for the scoring.

- The script requires access to your Google Drive.
- The link to the stored airstrip images must be updated to match your local setup.
- You also must provide a link to a Google Drive directory where you want the masks to be saved.
- The script will automatically loop over all the airstrip images, load the `LINESTRING` object and add a 20 meter buffer around the `LINESTRING` to define the area of the airstrip.
- All pixels inside the boundary of the airstrip are assigned a value of 1, and all other pixels in the image are assigned a value of 0.

At this point you should have separate directories with the airstrip training images, and their corresponding masks. Make sure there is a one-to-one match by listing the content of both directories. This is a critical step as the data loader won't check that for you.

Model Training

Run the notebook `zindi_airstrip_training.ipynb` to train the model. It is recommended to run this code on a GPU. Look at the table of content on Google Colab to see the different steps for this process.

- After the usual package install and module loading, the code will download the training data (airstrip images and masks) to the local computer node. After mounting your Google drive, make sure to edit the paths where the images and masks are stored. The files will be compressed and copied to the local disk and then uncompressed.
- The next section contains a long list of classes and functions to handle the data and the model for training. This includes the model definition (Unet), data augmentation functions, etc. Run all cells in that section.
- Before the main loop, the code checks if a GPU is available. It is highly recommended to use a GPU for training for efficiency.

- The main loop starts by loading the data sets (images and masks). By default only the RGB bands are used, but additional bands can be added. For the challenge our team also included the NIR and the first SAR band. A total of 5 bands is used in this model.
- The training set is broken into training / validation / test set with a split of (0.8, 0.1, 0.1).
- The data loaders are finally configured with a batch size of 8 and 2 workers.
- An optional cell is included to visualize the data. It loads a batch of images with their corresponding masks. This is useful for a sanity check.
- The next section instantiates the model. In this case we use 5 channels and the Unet model uses a ResNet50 encoder and imagenet weights.
- The model is then configured as follows:
    - Loss function: Dice
    - Optimizer: Adam
    - Learning rates: 5e-4 for pre-trained and new parameters
    - Weight decay: 0.0
- The model is then run for 100 epochs. The best model is selected as the model with the lowest loss value for the validation set.
- The model is then tested on the small test data set.
- Finally the “best” model can and should be saved to your Google drive.
- The last section of the code loads the best model in eval mode and runs inference on the entire training set. This is a good way of visualizing where the model fails.

Inference

Run `zindi_airstrip_prediction.ipynb` to generate predictions on new satellite images. Results will be saved in Google Colab runtime. Make sure to download or copy the output files to your drive.

- The path to the inference AOIs must be provided. See notes in the notebook. Similarly the path to the model file must be edited to match your configuration.

Post-processing

Run `image_cleaning.ipynb` to deal with roads and add buffers to predicted runways for scoring. This code is used to address some limitations of the model. For instance, the model tends to identify short segments of deforested areas, or road segments, as airstrips. Post-processing of the predictions can identify a significant fraction of those events.

- After the usual installation and loading of modules, the code mounts your Google drive.
- A description of the image cleaning process with the various steps is documented in the code. Run all the cells until you get to the main loop. Notice in particular that roads from Open Street Map are downloaded as GeoJSON objects and intersections with airstrips are easily identified.
- Edit the path where the prediction files from the model are stored.
- The code loops over each file and does image cleaning before saving a final result file to disk.
- Two output formats are available. For the data challenge we had to create a background buffer around the airstrips, which is the default format. The other format assigns all pixels outside the airstrip area a value of 0 and a value of 1 to pixels within the airstrip area.
- The code finally produces a CSV file for scoring.
- The files are produced on the local computer node. Don’t forget to download those files to your local disk or move them to your Google drive.

Recommendations and lessons learned

- Increase the size of the training data set.
    - Use the external database of airstrips and generate additional training tiles and masks.
    - Add tiles with dirt roads to help the model learn subtle differences between roads and airstrips
- Explore other loss functions and possibly develop custom functions to use the geometrical shape of airstrips.
- Investigate adding a class for dirt roads and use the OSM data to generate masks.
- Try different semantic segmentation models, e.g. DeepLab v3+
- Add the SWIR 1&2 and the second SAR bands and train the model with the extra information.
- Overlap the tiles during inference and develop image blending code to avoid truncation effects between tiles.