



# Week 11

Mining Asset Detection (MAD)



# 1001 ways to layer

- Goal: For each quadrant, for each season, generate a .tif image of the quadrant with the highest possible fidelity.
- How?
  - Downloading multiple images and their respective pixel based quality indicators
  - Combine them according to their quality indicators

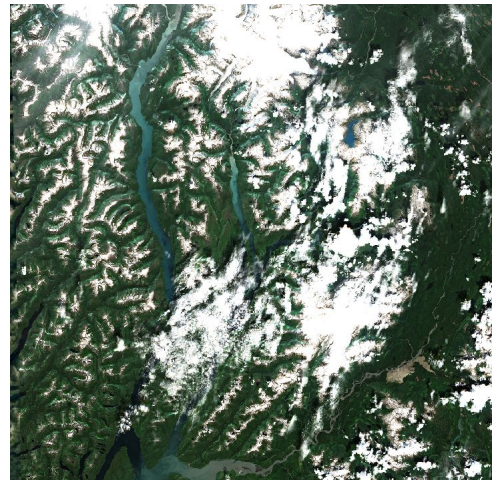
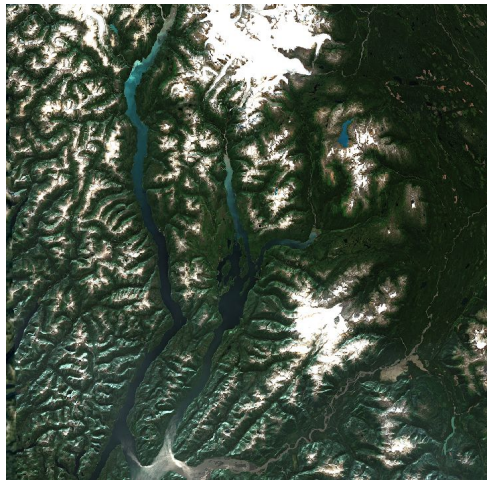
**Note:** Unlike my theory last time we do not have to rely on a blame system as the quality indicators are already given.



## 1001 ways to layer - Combination

- Define a ground layer (image with least clouds)
- Generate a cloud mask for each image (Cut out the clouds)
- Position ground layer on the lowest layer
- Layer the most cloudy image on top, repeat until masked least cloudy image on top
- A sandwich of sorts.

## 1001 ways to layer - Example

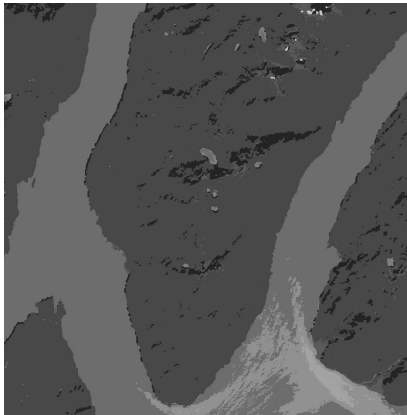


# 1001 ways to layer - Focus

- Clouds are the most limiting factor for quality

=> Layer the different images according to their cloud percentage, from least to most

- Quality indicators are faulty when it comes to soft fog/clouds!





# 1001 ways to layer - Combination supervision

- Solution:
  - Most important layer is the one with the least cloud percentage
  - Hand pick this image from the set of least cloudy images and regenerate the image

Essentially a manual blame system!

Under the assumption that 10% of images are nonoptimal => 490 images to handpick


Quick visual task, order of seconds.



## 1001 ways to layer - Resulting database size

Currently each tile has:

- 13 upsampled bands
  - 3 with 8 bit integers
  - 10 with 16 bit floats
- 10980 x 10980 pixels in total  $\approx$  120m pixels
- 1 Tile  $\approx$  2.xGB
- All tiles  $\approx$  8-9TB
- With multiple seasons per tile  $\approx$  20-30TB!



## 1001 ways to layer - Why size matters

- Makes all processes cumbersome
  - Think of the labelling task for the test set
  - Constant moving of data
- If mistakes happen (and they will) much more costly to fix them.
- Requires some hardware tweaks to manage them uniformly (in case of a server where these would be stored)





## 1001 ways to layer - Resulting database size

What we could do:

- Discretize the floating points to 8bit integers
  - Previously a single pixel  $3 * 8 \text{ bit} + 10 * 16 \text{ bit} = 184 \text{ bit}$
  - Afterwards  $13 * 8 \text{ bit} = 104 \text{ bit}$

=> Nearly 50% decrease

**Possible loss of nuance.** But to some degree inevitable as this happens anyway for the feeding of YOLO.



## 1001 ways to layer - Resulting database size

What we could do:

- Use .jp2 for storage.
  - Already done by AWS (not much more loss of information)
  - Can decrease the size from 10X to not at all depending on the image.

Requires more computation during image creation, but should decrease storage drastically.