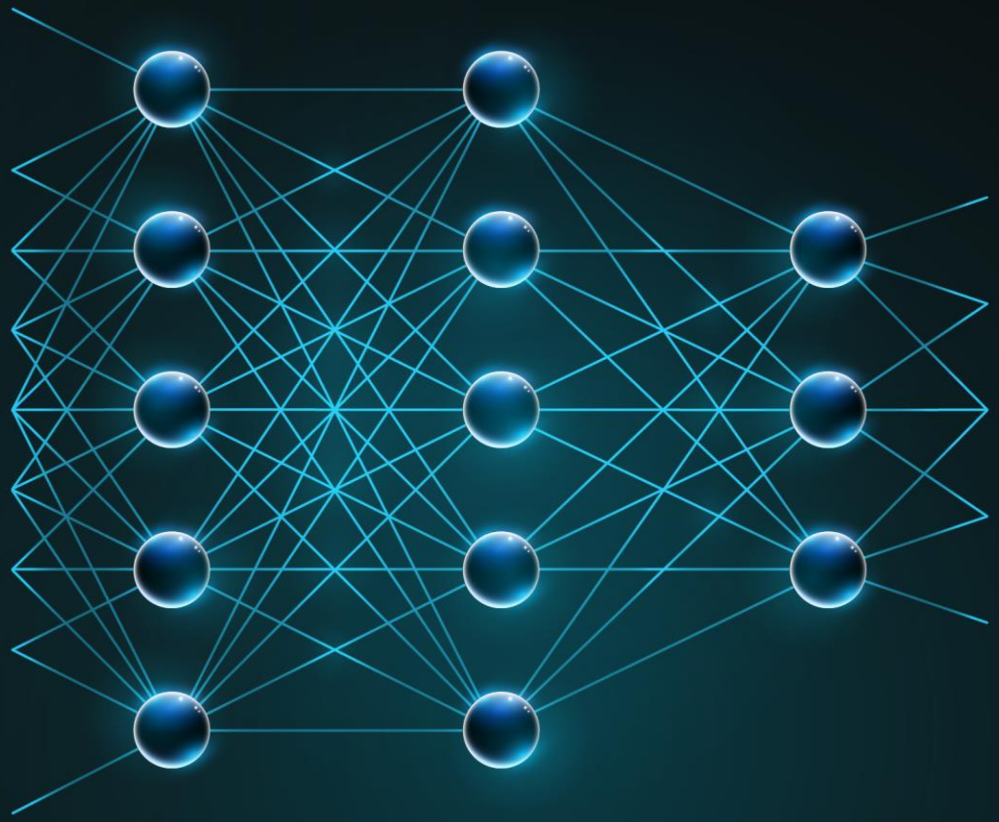
DATA SCIENCE SUMMIT

MACHINE LEARNING EDITION

Geospatial Deep Learning:

Winning Strategies for Segmenting Kelp Forests from Satellite Imagery

Michał Wierzbiński
Lead ML Engineer, Spyrosoft



Organizatorem wydarzenia jest Fundacja Academic Partners

ACADEMIC PARTNERS

Agenda

1. Case study: Kelp Wanted Competition
2. Data Processing
3. Modelling
4. Prediction Post-Processing
5. What Did Not Work?
6. Next Steps?
7. Summary

Repo with code

Who am I?

- Lead ML Engineer @Spyrosoft
- 7 years of experience
- Started as a Cloud Developer
- Specializing in Deep Learning solutions for the Geospatial Industry
- Specialty coffee enthusiast

Areas of expertise: Geospatial, Remote Sensing, Earth Observation, Satellite Imagery, Computer Vision, Deep Learning, ML & AI, MLOps, Cloud

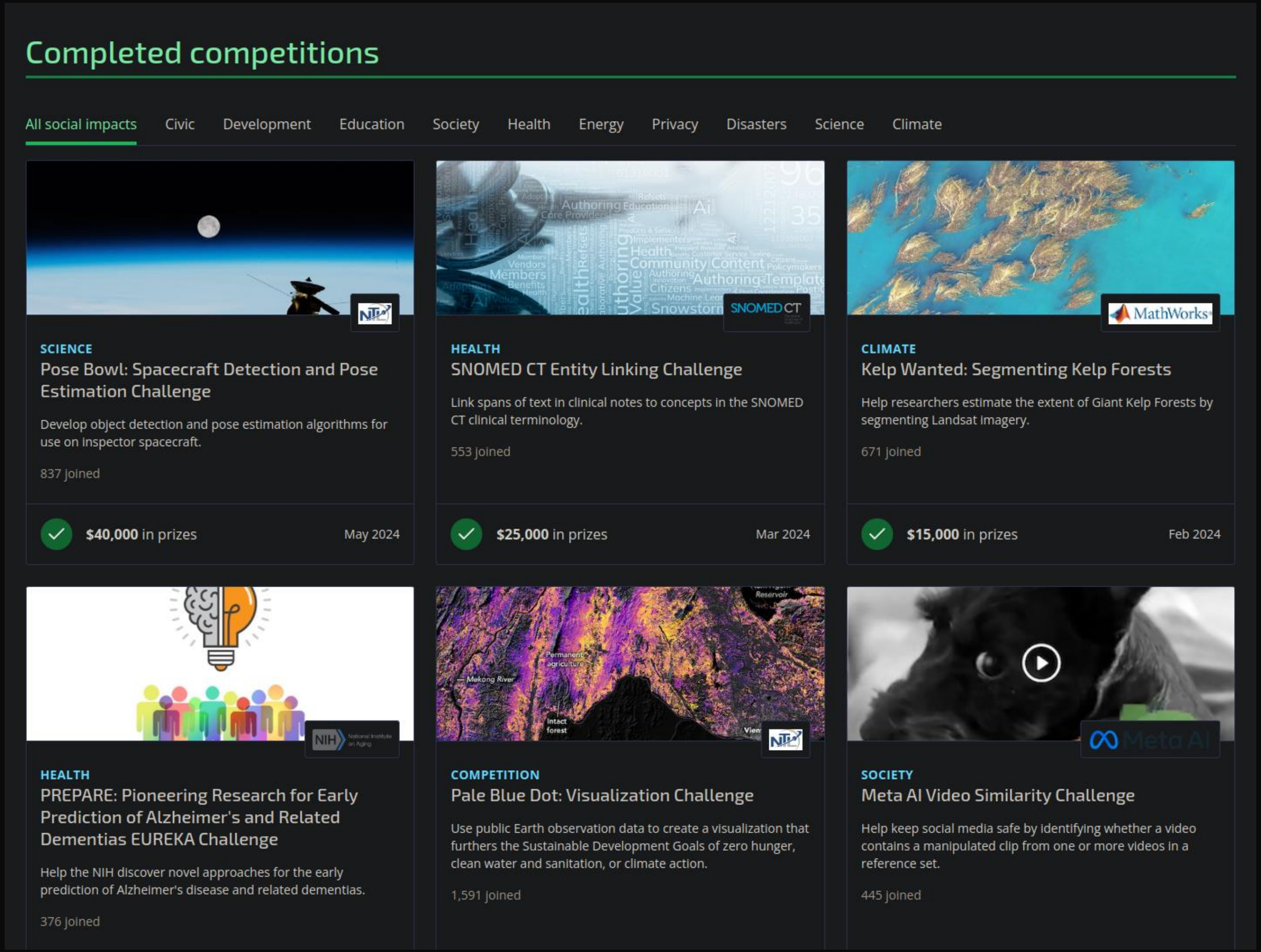
spyrosoft

Case study: Kelp Wanted: Segmenting Kelp Forests

2nd place solution

- Competition platform: DrivenData
- Labels from: Kelpwatch.org
- Goal: Help researchers estimate the extent of Giant Kelp Forests by segmenting Landsat imagery
- Task: semantic segmentation
- 350 x 350 pixel "tiles" of Landsat satellite imagery with 7 channels
- 30m / pixel spatial resolution
- Target metric:

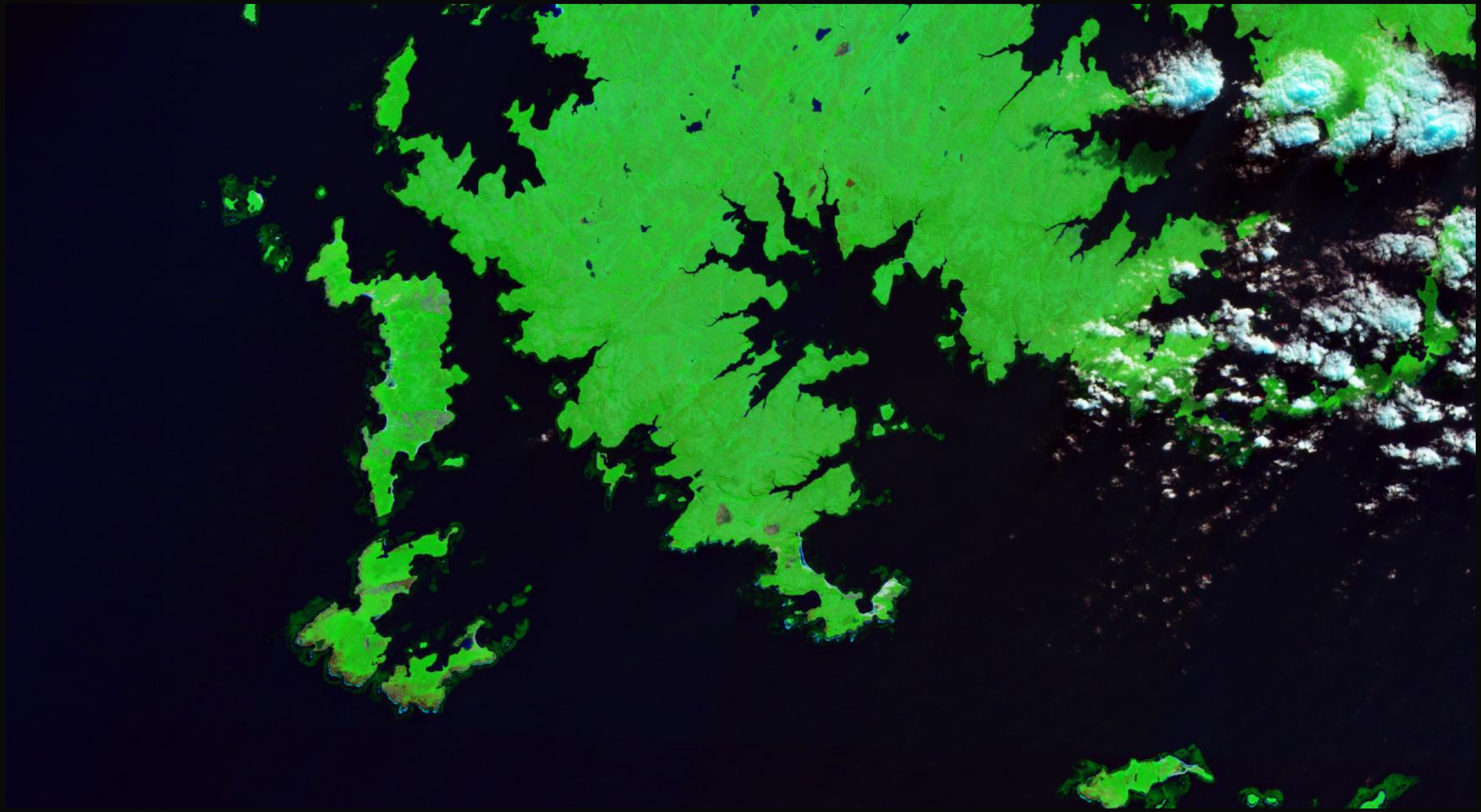
DICE score = 2 |A∩B| / (|A|+|B|)
= 2 * TP / (2 * TP + FP + FN)



What does the Kelp look like on satellite imagery?

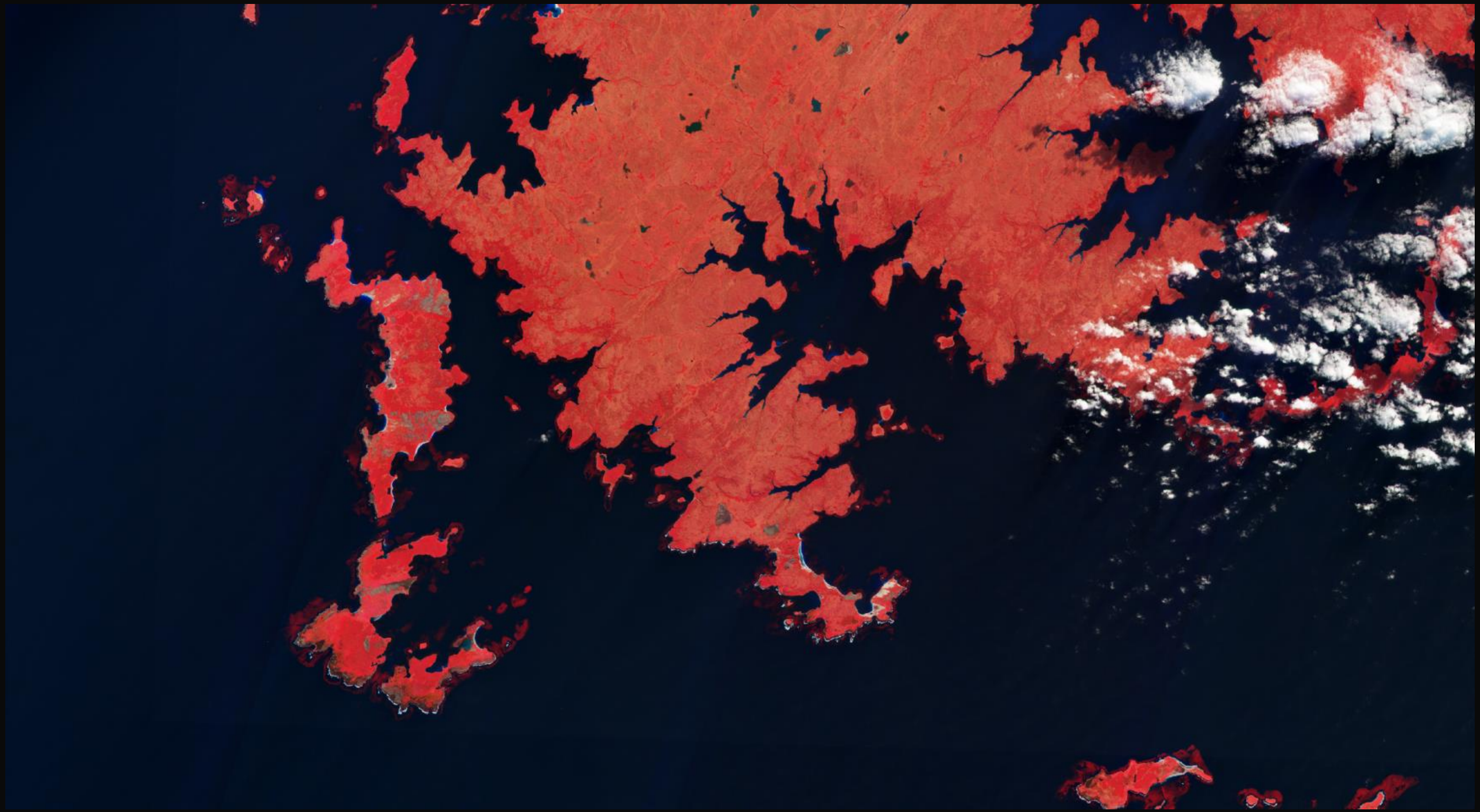
Source: ESA – Sentinel 2 L2A via Microsoft Planetary Computer

What does the Kelp look like on satellite imagery?



Source: ESA – Sentinel 2 L2A via Microsoft Planetary Computer

What does the Kelp look like on satellite imagery?

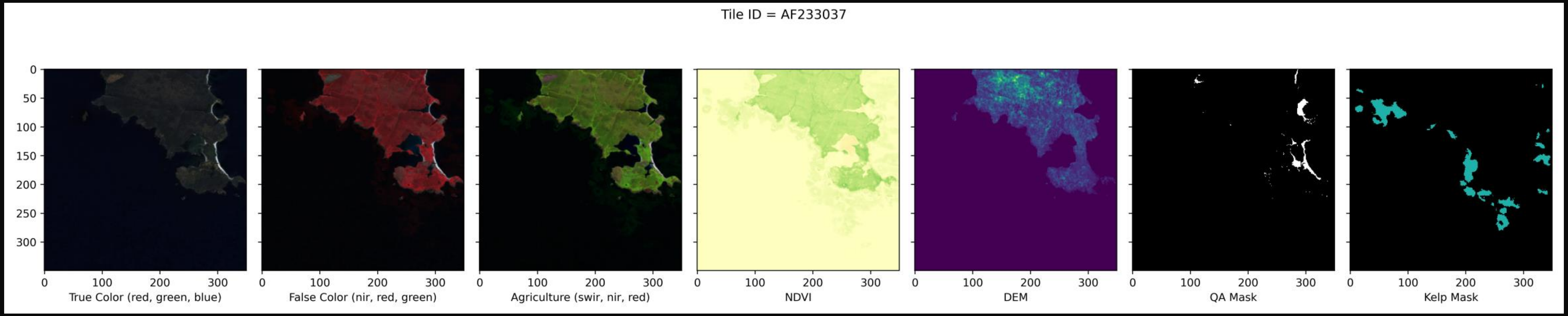


Source: ESA – Sentinel 2 L2A via Microsoft Planetary Computer

Data overview

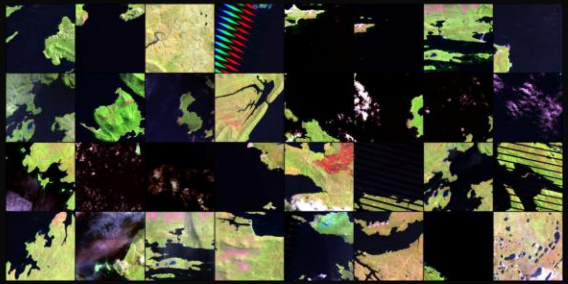
Input Bands:

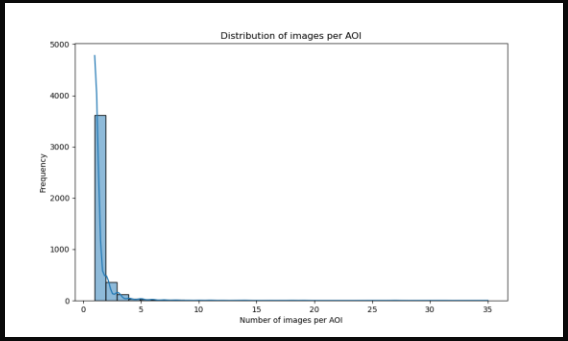
- SWIR (Shortwave Infrared)
- NIR (Near-Infrared)
- Red
- Green
- Blue
- Quality Mask (NaN values, defective/saturated pixels, cloud cover)
- Digital Elevation Model

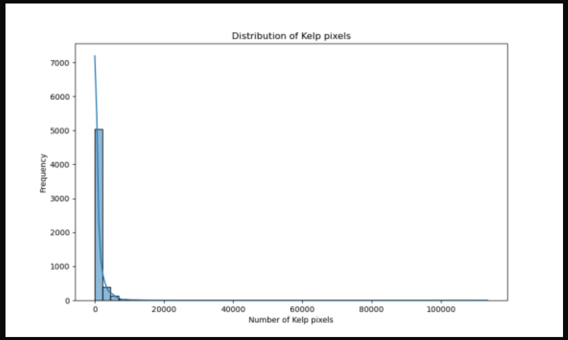


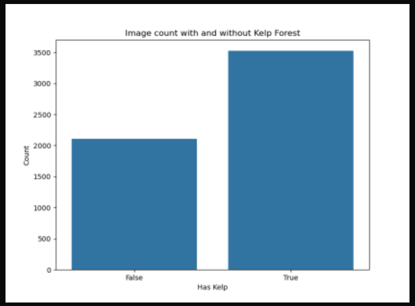
EDA

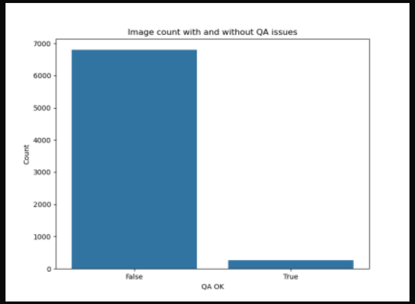
- Visual inspection of the images using composites:
    - True color
    - False color
    - Shortwave Infrared
- DEM, QA and Mask band visualization
- Descriptive Statistics:
    - Images per Splits
    - Distribution of images per AOI
    - DEM NaN pixels distribution
    - Distribution of images with and without Kelp
    - Kelp Pixels Distribution
    - High Kelp Pixels distribution (>40% of image)
    - QA corrupted pixels percentage
    - Water pixels distribution (DEM == 0 or NANs)
- Per Band Stats: min, max, median, mean, std, q01, q99

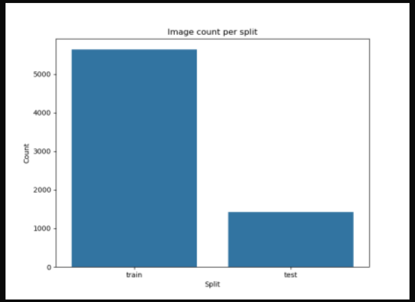


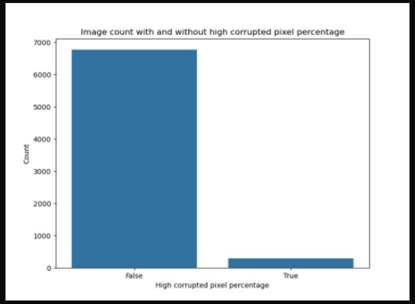






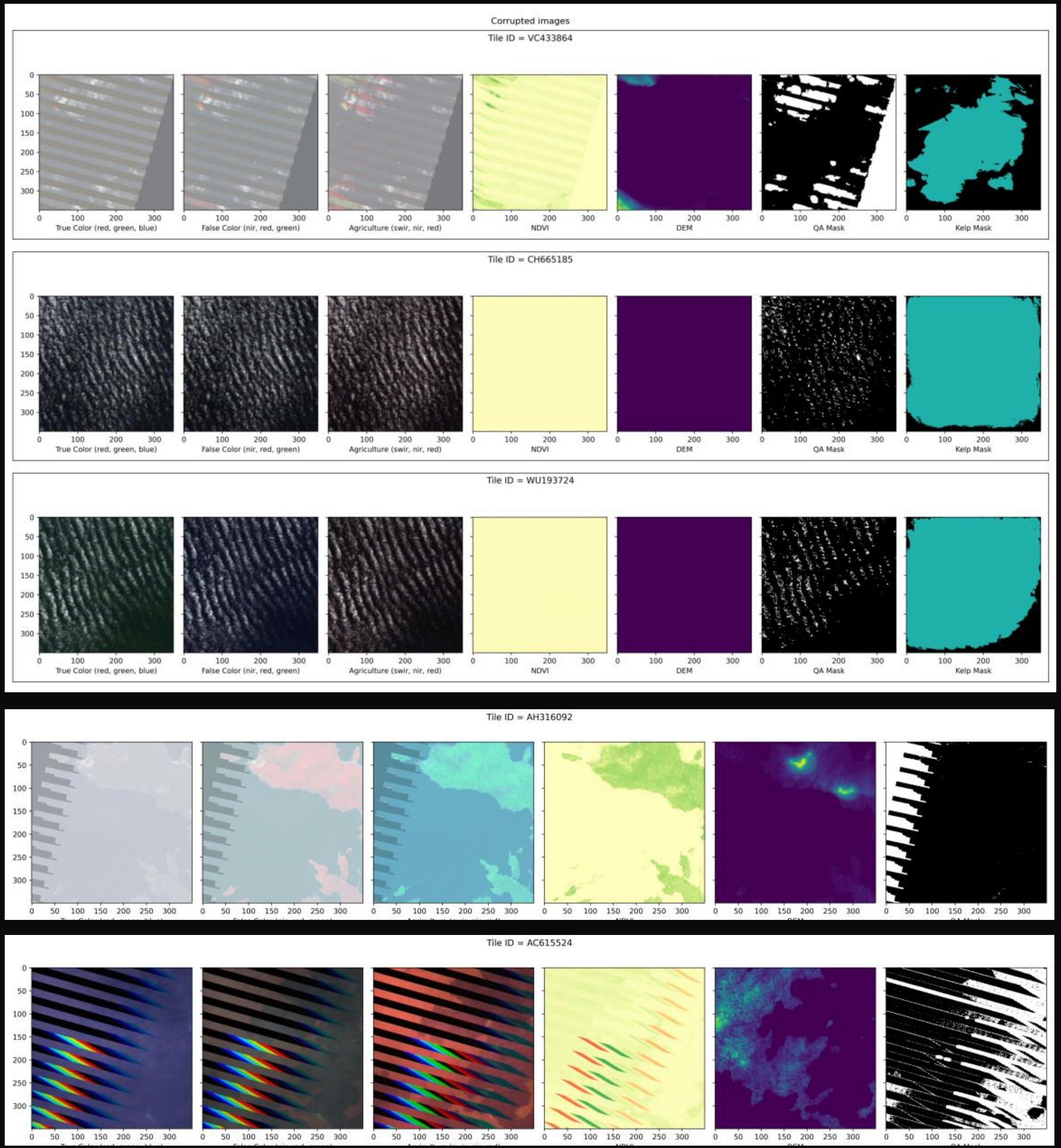






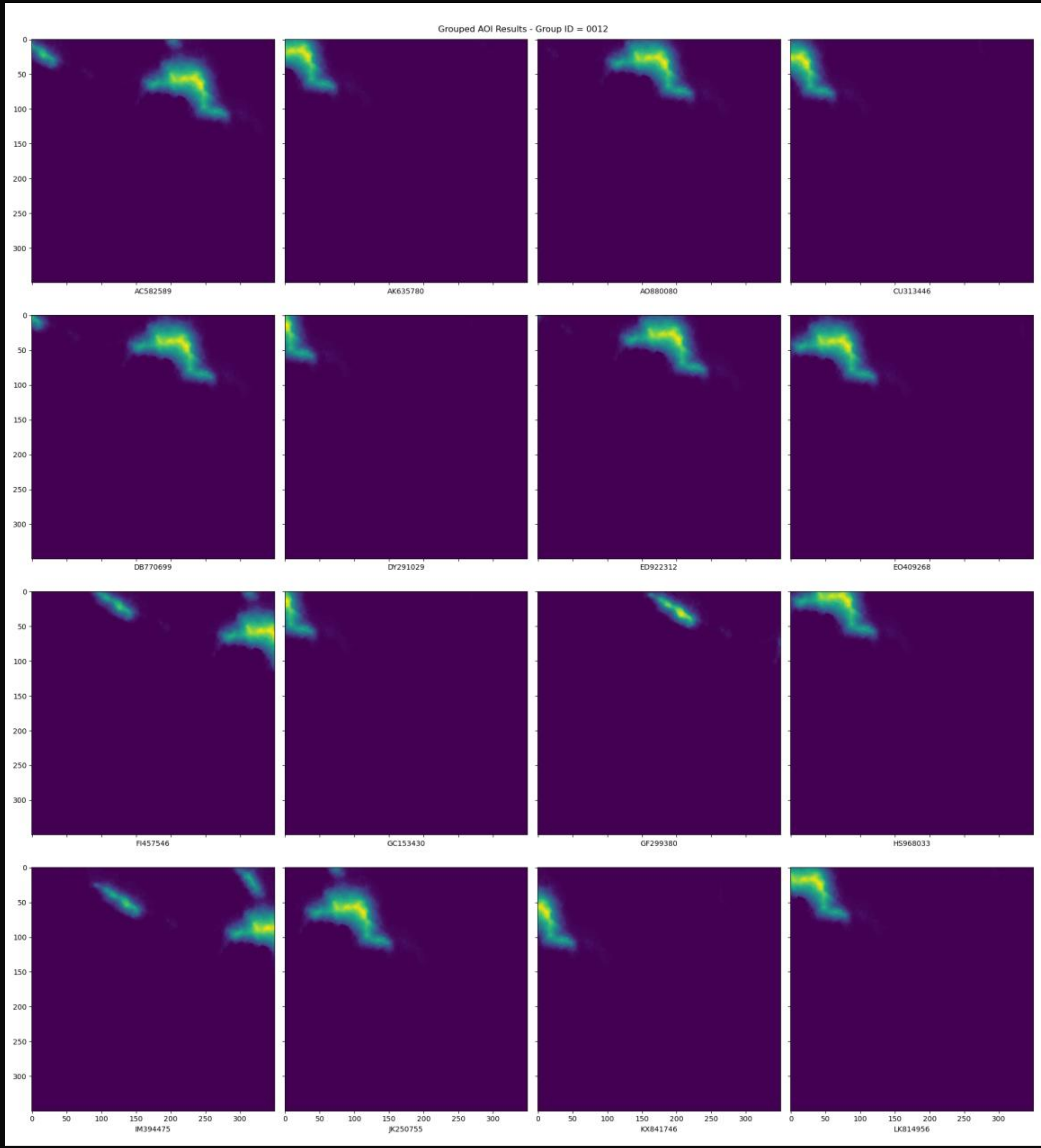
Data issues

- 3 tiles had corrupted masks
- Misaligned DEM layer
- Over 2/5 images had no Kelp pixels (all land or open water images)
- 91 images with over 70% corrupted or missing pixels
- 768 images with over 98% water pixels
- Landsat itself has various issues:
  - Saturated and Cloudy pixels
  - Striping artifacts
  - Missing values (NaNs)



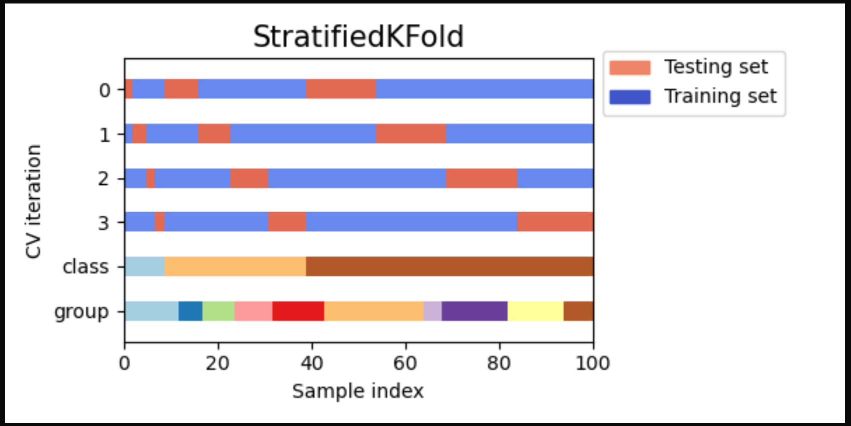
Deduplication

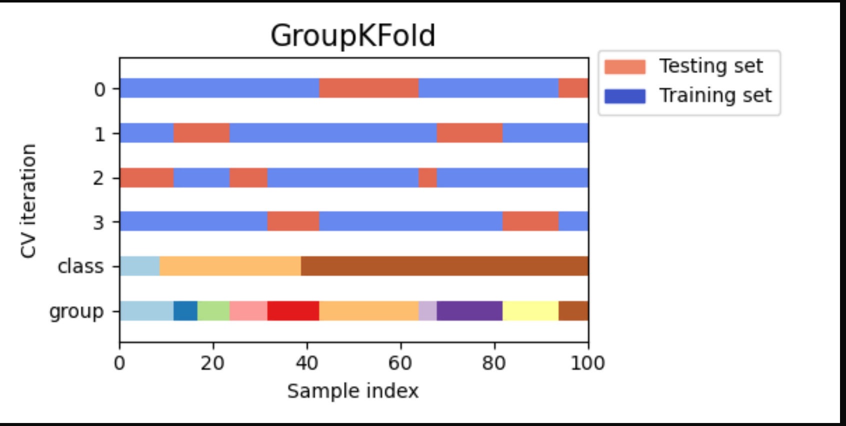
- Multiple "similar looking" images detected via manual inspection
- Grouped Areas of Interest (AOIs) via cosine similarity of embeddings generated by pre-trained ResNet-50
- Used DEM layer as input
- Similarity threshold = 0.97
- Results: 3313 unique AOI groups



Train-val-test splits

- Train & validation sets: stratified Grouped 10-Fold split
- Avoids evaluating on similar AOIs (data leakage)
- Keeps proportion of labels similar across the splits
- Test data provided by competition host





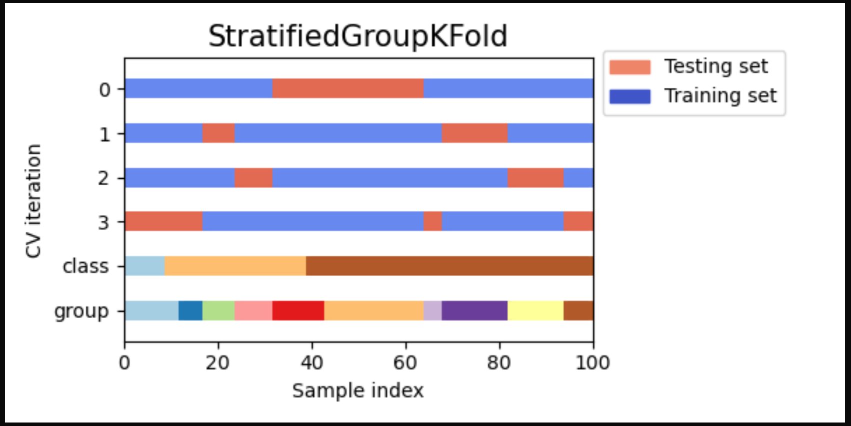


Image source: scikit-learn docs

Baseline

- segmentation-models-pytorch
- pytorch-lightning
- UNet with ResNet-50 encoder
- Random split
- Z-score normalization
- Inputs: SWIR, NIR, R, G, B, QA, DEM & NDVI
- Batch size = 32
- Adam optimizer
- Constant learning rate schedule – 3e-4
- Cross entropy loss
- 10 epochs
- LB Score = **0.6569**

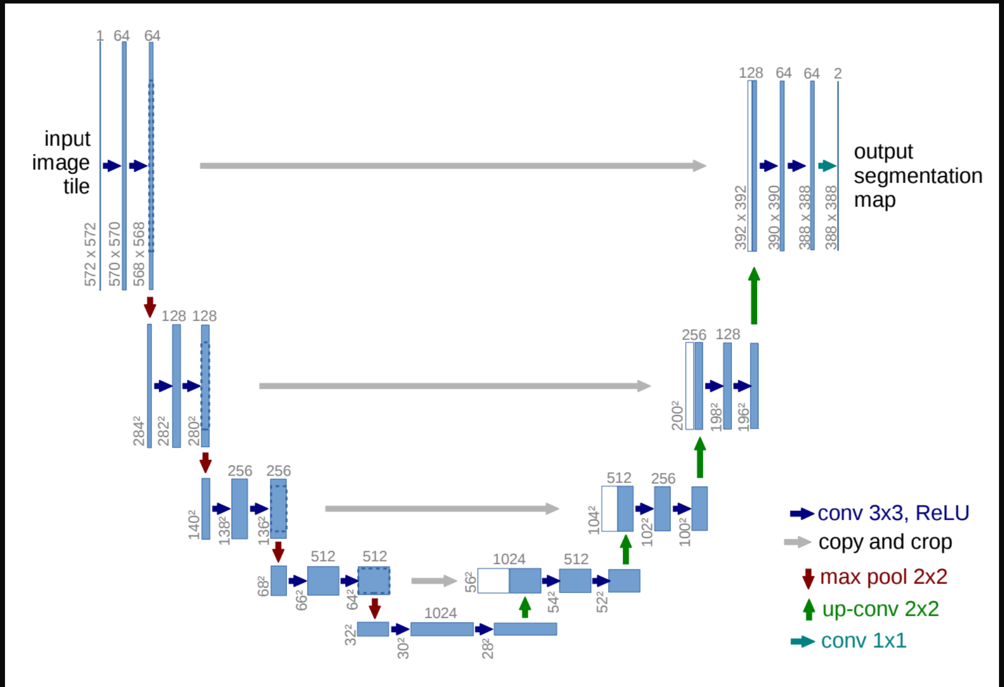
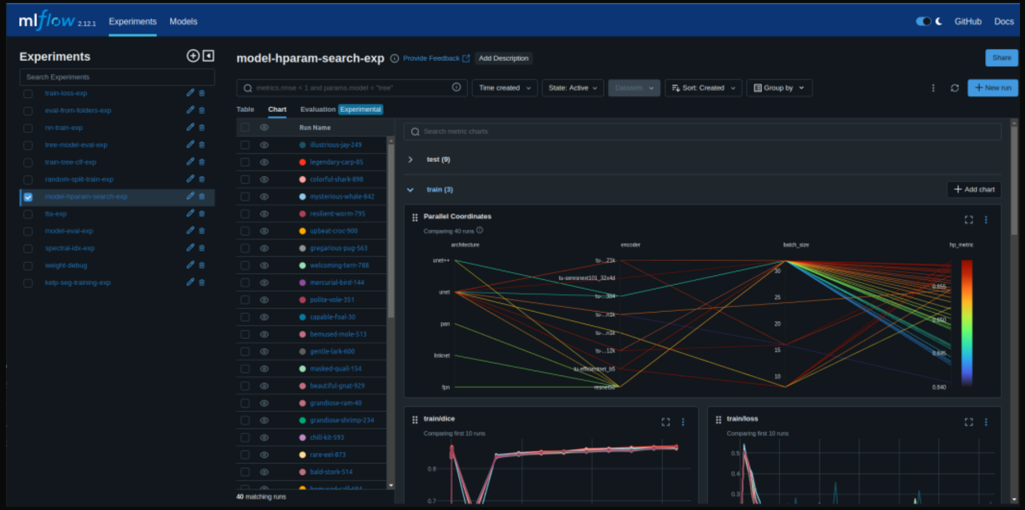
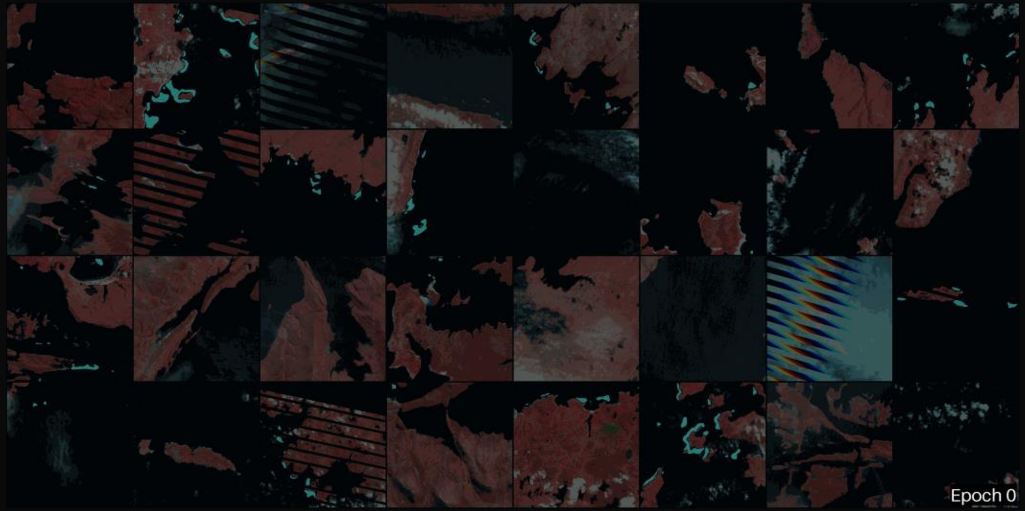


Image source: https://arxiv.org/abs/1505.04597

Experiment tracking - MLFlow

- Log everything
- Model hyperparameters
- Experiment config
- Training & validation curves
- DICE, IOU, Accuracy, Precision, Recall, F1
- Learning rate schedule
- Model inputs: as composites and individual bands
- Model predictions on few batches every epoch
- Use Parallel Coordinate plot for hyper-parameter optimization





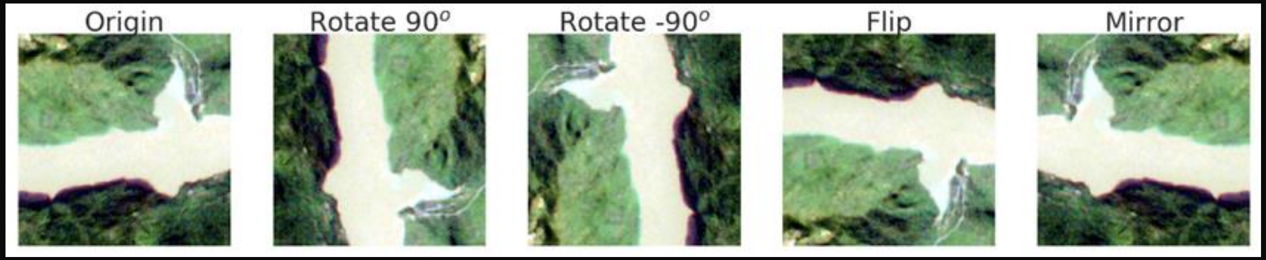
Data sampling

- Weighted Random Sampler
- Per-image weight
- Determine the optimal per-image weight:
  - *has_kelp* - a flag indicating if the image has kelp in it
  - *kelp_pixels_pct* - percentage of all pixels marked as kelp
  - *dem_nan_pixels_pct* - percentage of all DEM pixels marked as NaN
  - *dem_zero_pixels_pct* - percentage of all DEM pixels with value=zero
  - *almost_all_water* - a flag indicating that over 98% of the DEM layer pixels are water
  - *qa_ok* - a flag indicating that no pixels are corrupted in the QA band
  - *qa_corrupted_pixels_pct* - percentage of corrupted pixels in the QA band
- Weights calculated using hyperparameter search

Samples per epoch	has_kelp	kelp_pixels_pct	qa_ok	qa_corrupted_pixels_pct	almost_all_water	dem_nan_pixels_pct	dem_zero_pixels_pct	val/dice
10240	3	0.5	-1	0	-1	0.25	0	0.845

Augmentation strategies

- Used basic augmentations:
  - Vertical & Horizontal Flips
  - Rotations [-90 - 90 deg.]
  - Pad to 352x352 (needed by UNet)
  - Append Spectral Indices

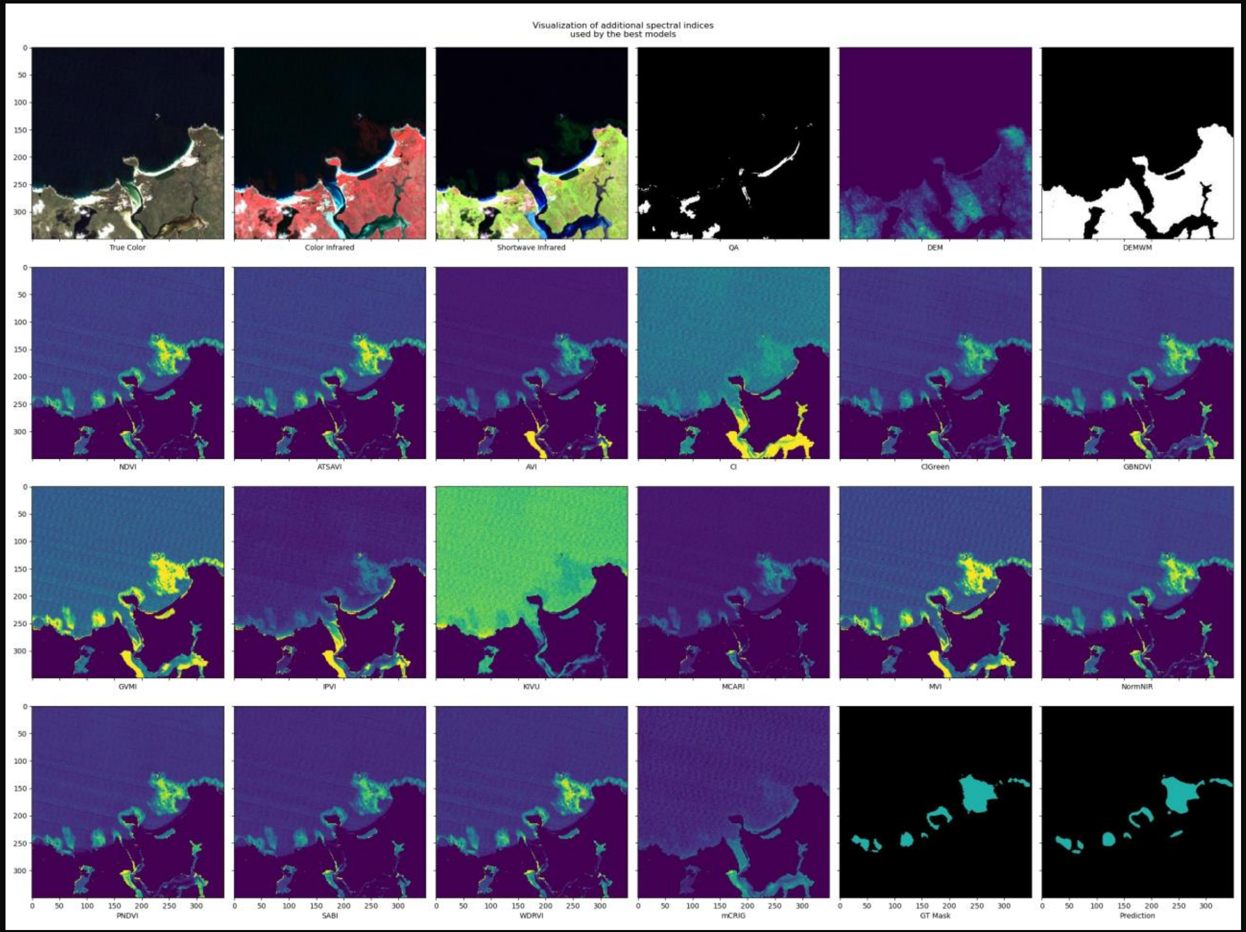


- Adding more advanced augmentations did not improve performance
- Avoid destructive operations:
  - Random crop-resize
  - Gaussian noise
  - Hue & Saturation
  - Contrast
- Keep spectral characteristics intact when working with Multispectral Imagery
- Augmentations on multi-channel tensors are expensive to compute on CPU
- Use GPU to apply them on whole batch at the time – kornia library

Image source: http://dx.doi.org/10.3390/rs12030417

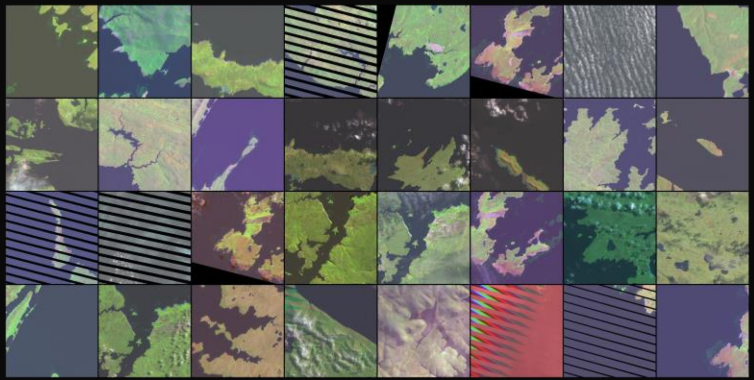
Feature Engineering

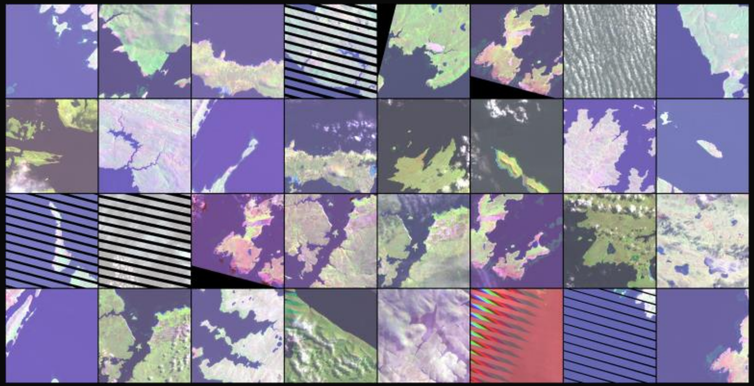
- Append spectral indices to help the model
- Use Spectral Index DB to see what's available for your sensor
- Extra indices:
    - DEMWM,
    - NDVI,
    - ATSAVI,
    - AVI,
    - CI,
    - ClGreen,
    - GBNDVI,
    - GVMI,
    - IPVI,
    - KIVU,
    - MCARI,
    - MVI,
    - NormNIR,
    - PNDVI,
    - SABI,
    - WDRVI,
    - mCRIG

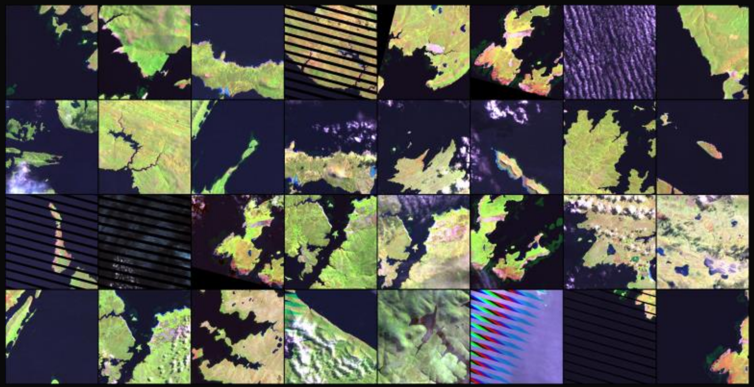


Data Normalization

- Data was already pre-processed by competition organizers
- Calculate per-band statistics to be used for normalization
- Ignore NaNs and corrupted pixels
- Start with z-score
- Min-max only if outliers are removed e.g., clip (0; 20,000)
- Best results were obtained with quantile normalization (q01-q99)
- Add some domain knowledge - mask values with QA mask and Land mask – use stats for water pixels only





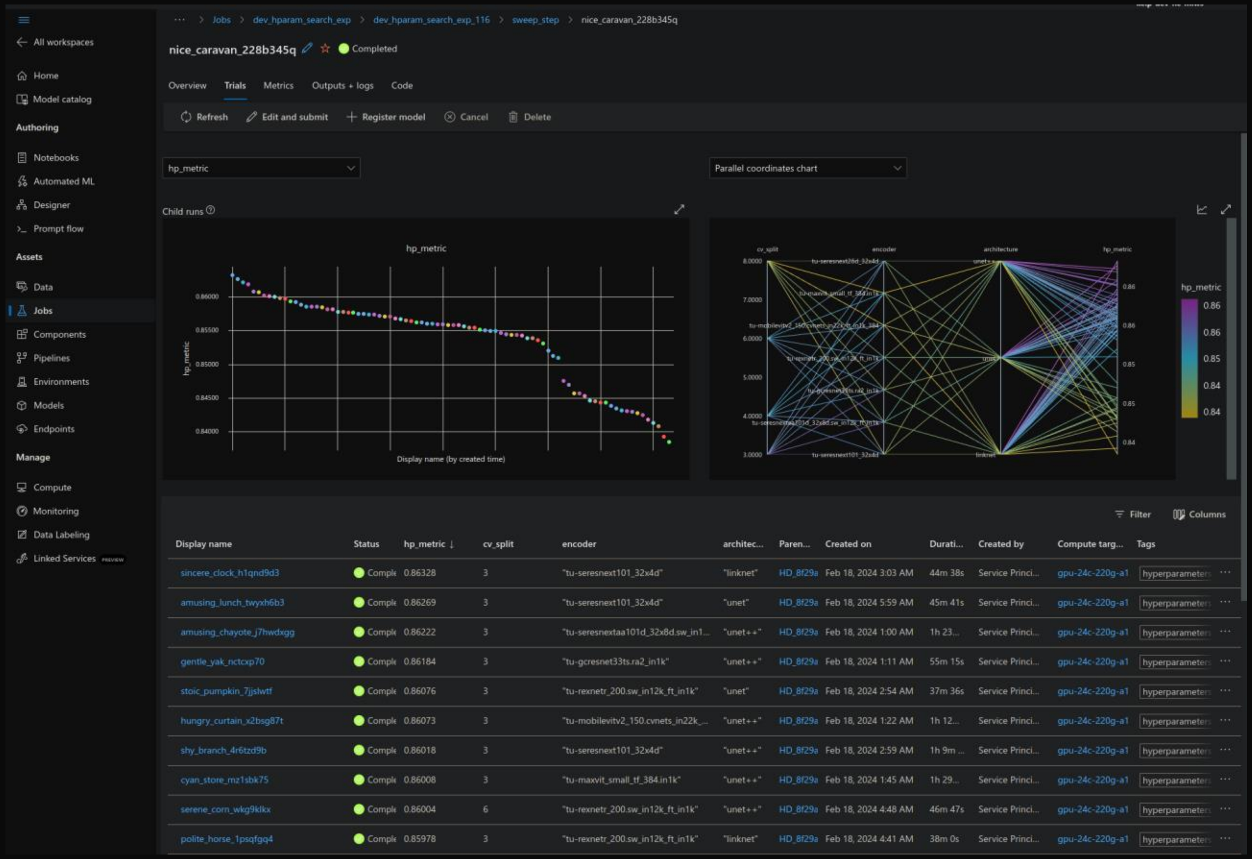


Model Hyperparameters

- Use hparam tuning lib: RayTune, Optuna, HyperOpt
- Grid:
  - Architecture: Unet, Unet++, LinkNet, DeepLab, DeepLabv3+, EfficientUnet++, ResUnet etc.
  - Encoder: ResNet, EfficientNet, VGG, VIT etc.
  - Learning rate: 1e-5 – 1e-1
  - Learning rate scheduler: Constant, StepLR, OneCycleLR, etc.
  - Optimizer: Adam, SGD, AdamW etc.
  - Loss functions: Cross Entropy, DICE, Tversky, Jaccard, Lovasz, Focal etc.
  - Epochs: 5 – 50
  - Samples per epoch: 5120 – 10240
  - Apply Land Masking – Yes/No
  - Pre-trained – Yes/No
  - Bands to use: R, G, B, SWIR, NIR, QA, DEM
  - Band order
  - Spectral indices to append: DEMWM, NDVI, ATSAVI, AVI, CI, ClGreen etc.
- Keep batch size and precision constant – maximize GPU utilization
- Use FP16 mixed precision if possible

Scaling the compute - Azure ML

- Hparam search does not scale well locally
- Use cloud or HPC if available to speed up the computation
- Use spot instances if possible
- Azure ML offers hosted MLFlow out-of-the-box
- If your dataset is small – download it directly to your compute – don't stream from blob storage
- Make sure your data is in the same region as your compute
- Make sure your remote environment matches your local one – use lock-files!
- Saturate your GPUs!



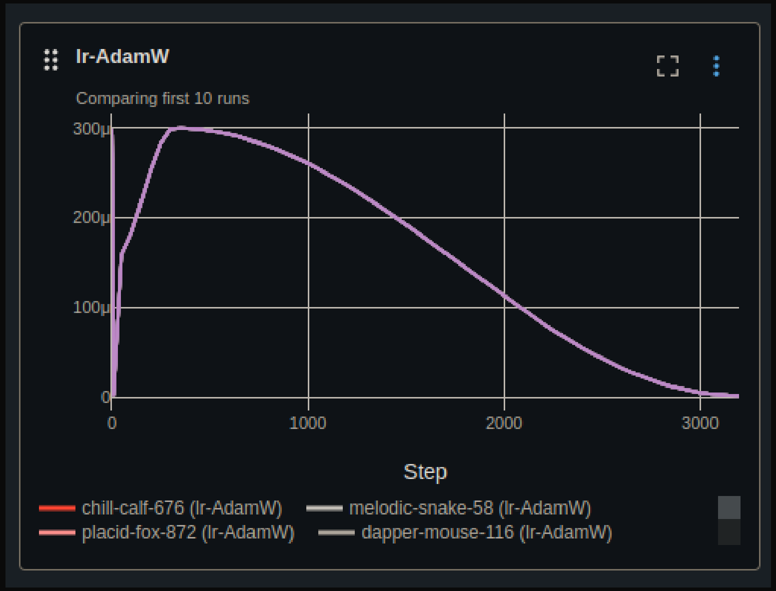
Model architecture

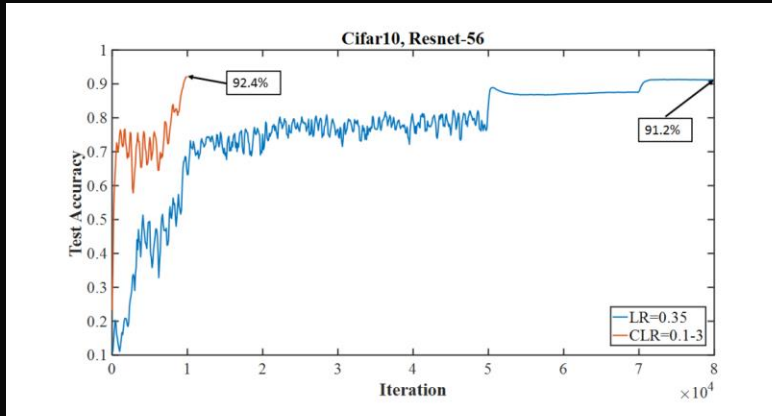
- ResUnet++ often results in NaN loss.
- Unet and Unet++ often were the best.
- Unet++ training was not deterministic even though pl.seed_everything() was used
- Bigger models often resulted in OOM errors during training:
    - Reduce batch size
    - Apply gradient accumulation
- Some models expected the input image to be divisible by 8, 24, 128 etc.
    - Adjust training config on the fly to allow for training those models.
- In general, bigger models worked better, but overfitted quicker
- ConvNext and SWIN Transformer models not supported
- Used DICE instead of cross entropy
- **The best combo was UNet & EfficientNet-B5**

encoder	architecture	val/dice
tu-efficientnet_b5	unet	0.85854
tu-seresnext101_32x4d	unet	0.85807
tu-resnest50d_4s2x40d	unet	0.85787
tu-rexnetr_300	unet	0.85749
tu-seresnext26d_32x4d	unet	0.85728

Learning Rate Scheduling

- Best LR Schedule was OneCycleLR
- Proposed in Super-Convergence paper
- Allows to achieve identical performance in less epochs
- Avoids local minima by allowing the model to explore the loss landscape for longer





(a) Comparison of test accuracies of super-convergence example to a typical (piecewise constant) training regime.

Image source: https://arxiv.org/abs/1708.07120

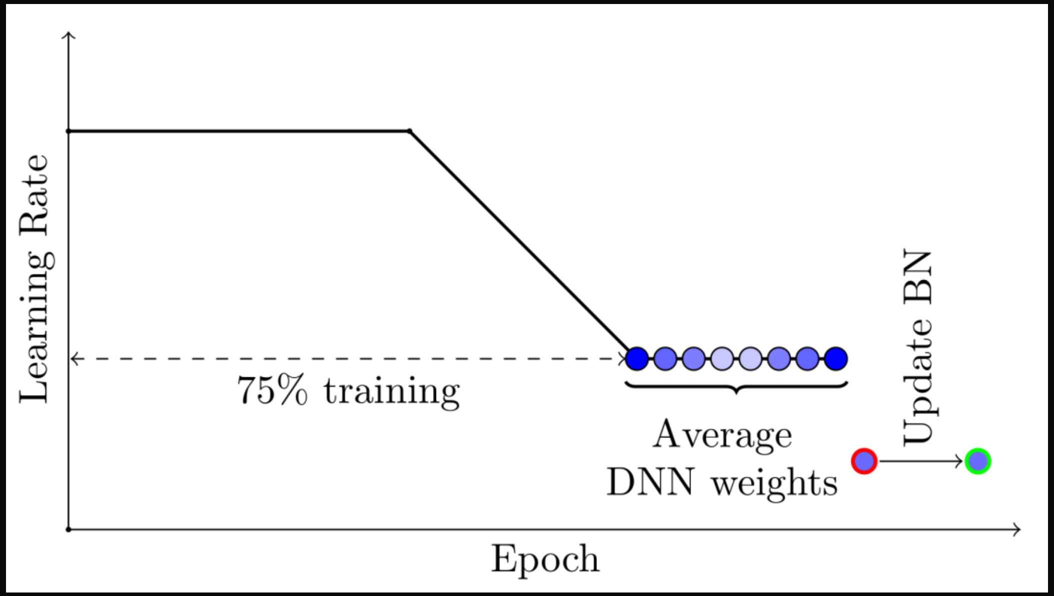
Prediction post-processing

- **Morphological Operations:**
  - Erosion and Dilation to remove noise
  - Opening and Closing for shape refinement
- **Test Time augmentations (TTA):**
  - Apply augmentations (e.g., flips, rotations) during inference
  - Average predictions from augmented inputs for improved accuracy
  - **Did not improve model performance for kelp segmentation**
- **Label Smoothing:**
  - Apply spatial smoothing to reduce noise
  - Ensure smoother transitions between segments
- **Ensemble Methods:**
  - Combine multiple models' predictions for robust output
  - Average or vote-based fusion techniques
  - **Can be used with soft-labels (probabilities instead of labels)**
  - **Adds a lot of complexity and increases prediction times – not suitable for production use unless you have very good use-case for it**
- **Contour Detection:**
  - Use edge detection algorithms to refine segment borders
  - Improve alignment of segmentation boundaries
- **Class-Specific Post-Processing:**
  - Apply targeted techniques for different classes
  - Customize processing based on object characteristics
- **Threshold Adjustment:**
  - Fine-tune threshold values for binary segmentation maps
  - Balance between precision and recall
  - **Using 0.45 instead of the classic 0.5 as decision threshold improved the performance slightly**

What did not work?

Stochastic Weights Averaging (SWA)

- SWA performs an equal average of the weights traversed by SGD with a modified learning rate schedule
- SWA solutions end up in the center of a wide flat region of loss
- SWA performs an equal average of the weights traversed by SGD with a modified learning rate schedule
- SWA solutions end up in the center of a wide flat region of loss
- While SGD tends to converge to the boundary of the low-loss region

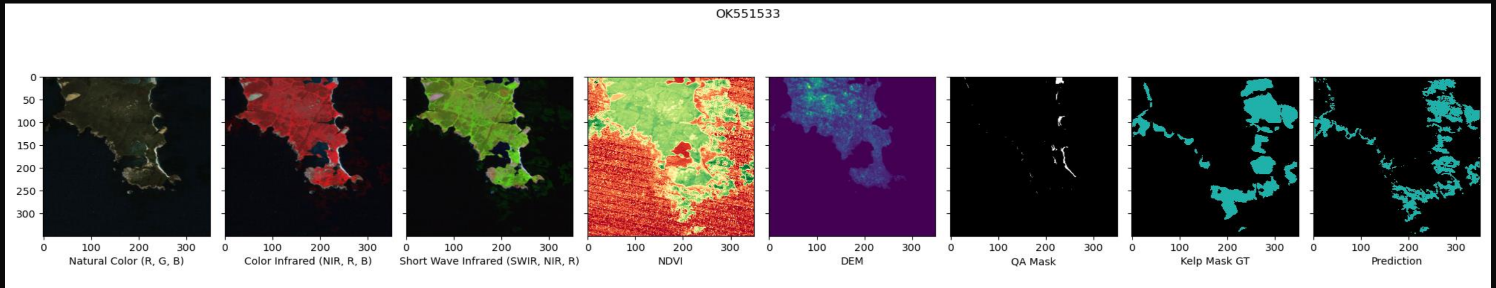


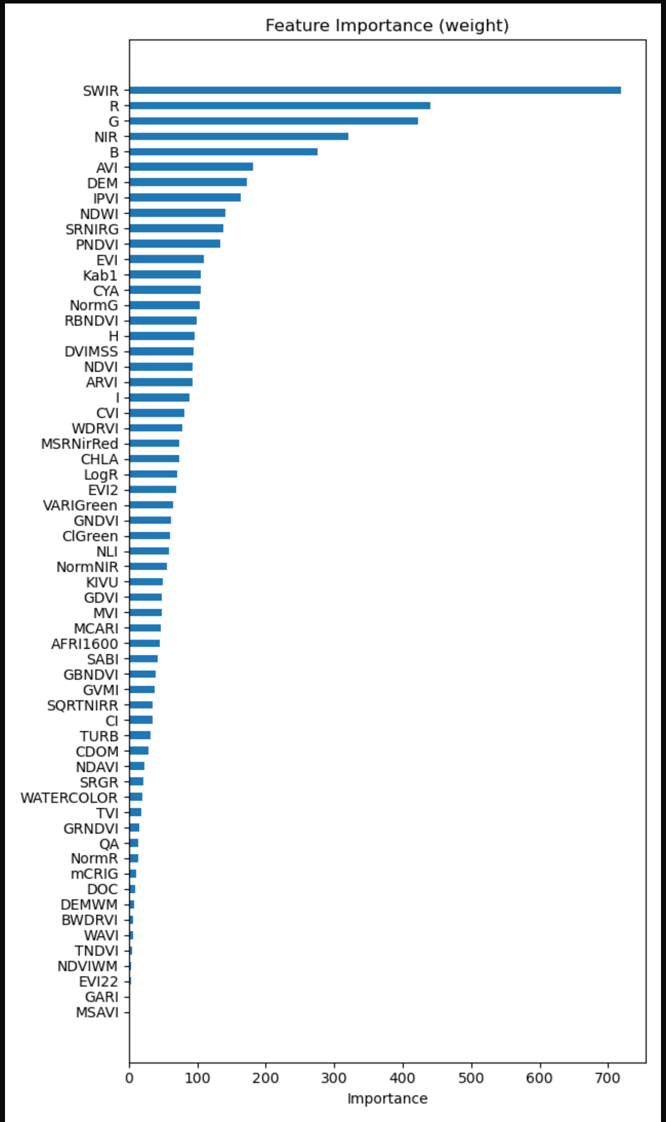
- This makes SGD susceptible to the shift between train and test error surfaces
- Standard decaying schedule is used for the first 75% of the training and then a constant value is used for the remaining 25%.
- The SWA averages are formed during the last 25% of training.

What did not work?

XGBoost

- XGBoost on pixel-level with all spectral indices
- Very long training times on CPU – use GPU when possible
- Hparam search did not improve results compared to CNN
- Segmentation mask often "jagged" with missing pixels
- Public LB score: 0.5125
- Abandoned since improvement of 20 p.p. was unrealistic to achieve
- Feature importance proves the importance of extra spectral indices

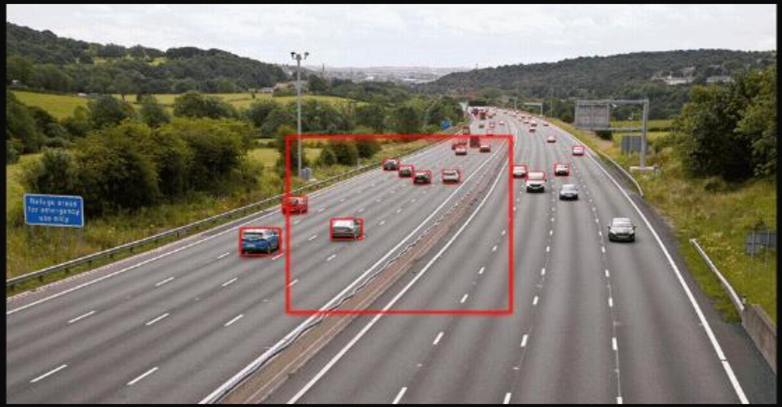




What did not work?

SAHI

- Slicing Aided Hyper Inference - utilizes inference on image slices and prediction merging.
- Slower than running inference on full image
- Usually ends up having better performance, especially for smaller features



The idea was simple:

- Generate sliced dataset of small 128x128 non-overlapping tiles from the bigger 350x350 input images
- Use this dataset to train new model
- During training resize the crops to e.g., 320x320 resolution and train on those
- When running inference generate overlapping tiles, inference on those tiles, and merge the predicted masks by averaging the predictions in the overlapping areas
- Profit?
- **LB score of 0.6848**

Next steps?

Data:

- Revisit kelp labels and re-annotate with agreement between annotators
- Explore sensor fusion and harmonization with Sentinel 2

Models:

- Prithvi-100M - trained on Harmonized Landsat Sentinel 2 (HLS) data
- The Pretrained Remote Sensing Transformer (Presto) - requires additional data and was not directly trained on Landsat - Sentinel 2 as the main data source
- Transformer based models such as Swin transformer or SegFormer

Summary

- Spend a lot of time on EDA
- Identify as much issues with the data as possible
- Make sure there is no data leakage
- Have a reasonable baseline ready fast
- Log everything
- Ensure reproducibility
- When choosing between tweaking inputs (image weights, normalization, augmentations etc.) and model hyperparameters – choose inputs
- Avoid ensembles

Method	Public LB score	Score increase
Baseline UNet + ResNet-50 encoder	**0.6569**	N/A
+ AOI grouping with cosine similarity and more robust CV strategy	0.6648	0.0079
+ Dice loss instead of Cross Entropy	0.6824	0.0176
+ Weighted sampler	0.6940	0.0116
+ Appending 17 extra spectral indices + using water mask to mask land	0.7083	0.0143
+ EfficientNet-B5 instead of ResNet-50	0.7101	0.0018
+ Decision threshold optimization + bf16-mixed inference	0.7132	0.0031
+ 10-fold model ensemble	0.7169	0.0037
+ Training for 50 epochs + weighted average + soft labels	**0.7210**	0.0041

DATA SCIENCE SUMMIT

MACHINE LEARNING EDITION

Thank you!

We encourage you to ask questions and to rate this session.

FEEDBACK

Geospatial Deep Learning: Winning Strategies for Segmenting Kelp Forests from Satellite Imagery

Michał Wierzbiński

http://ml.dssconf.pl/user.html#!/lecture/DSSML24-7292/rate