



Deep Learning for Segmentation of Hyperspectral Satellite Images

Author: Arya Raeesi

Supervisor: Roger Birkeland

Problem Owner: NTNU SmallSat Lab

Date: 2025-05-02

Contents

1	Problem Statement	1
2	Solution	2
2.1	Hyperspectral Images	2
2.2	Neural Networks	2
2.2.1	Parts of a Neural Network	2
2.2.2	Convolutional Neural Networks	3
2.3	1D-JustoLiuNet	4
2.4	Preprocessing	6
2.5	Dataset and Format	6
2.6	Model Training Setup	6
2.6.1	Optimizer - AdamW	7
2.6.2	Scheduler - StepLR	7
2.6.3	Loss Function - CrossEntropyLoss	8
2.7	Classification Metrics	8
2.7.1	Accuracy	8
2.7.2	Precision	8
2.7.3	Recall	9
2.7.4	F1-Score	9
2.7.5	Macro Average	9
2.7.6	Weighted Average	9
2.7.7	Confusion Matrix	9
2.8	Evaluation Methodology	10
2.9	GPU Acceleration	10
2.10	Code Implementation Diagrams	10
3	Verification and Test	14
3.1	Tests	14
3.1.1	T01	14
3.1.2	T02	16
3.1.3	T03	17
3.1.4	T04	18
3.1.5	T05	20
3.1.6	T06	21
3.2	Sources of Error	23
3.3	Why the 1D-JustoLiuNet Doesn't Work as Intended	24
3.4	Future Work and Recommendations	26
4	Conclusion	27
5	Acknowledgments	28
A	Code	30
A.1	libraries.py	30
A.2	cnn_1d.py	31
A.3	processing.py	33
A.4	train_functions.py	36
A.5	dataset.py	41
A.6	manage_data.py	43

A.7	train.py	47
B	Complete Class Diagram	50
C	Images Used in Testing	51
C.1	T01	51
C.2	T02	51
C.3	T03	51
C.4	T04	51
C.5	T05	52
C.6	T06	52
D	Used Images and Their Labels	54

1 Problem Statement

The problem addressed in this project concerns the adaptation of an existing convolutional neural network, the 1D-JustoLiuNet, from the HYPSO-1 satellite to the HYPSO-2 satellite at the SmallSat Lab at NTNU. The network was originally trained on data from HYPSO-1 and now needs to be adjusted and retrained on new hyperspectral images from HYPSO-2 to support continued onboard use during operational deployment.

Hyperspectral satellite images are datasets that capture information across a large number of narrow wavelength bands, providing a rich spectral representation for each pixel in the image. This spectral information enables accurate classification of surface types such as sea, land, and cloud, based on their unique spectral signatures.

To automate this classification, a convolutional neural network (CNN) is employed, a type of machine learning model particularly well suited for image analysis. The goal of the project is to train and evaluate this network using the new HYPSO-2 images, and to assess the model's ability to generalize to diverse and challenging imaging conditions.

2 Solution

2.1 Hyperspectral Images

Hyperspectral images are images that capture information across contiguous spectral bands, typically ranging from the visible to the near-infrared spectrum. Unlike traditional RGB images that contain only three channels (red, green and blue), hyperspectral data provides a full reflectance spectrum for each pixel in the image, enabling fine-grained discrimination between materials based on their spectral signatures. This spectral information makes hyperspectral imaging particularly useful for our task of segmenting sea, land and clouds from HYPSON-2 satellite images. Figure 1 shows an illustration of a satellite with a hyperspectral camera.

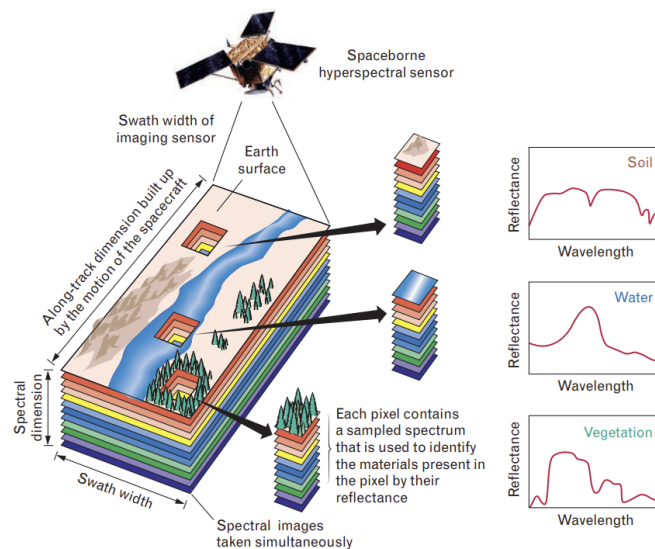


Figure 1: Illustration of a satellite with a hyperspectral camera [1].

2.2 Neural Networks

Neural networks are computational models inspired by the structure of the human brain, consisting of interconnected layers of artificial neurons [2, page 15-16]. Each neuron applies a mathematical function to its input and passes the result to the next layer. These networks are trained using large datasets and learn to recognize patterns, classify data, or make predictions [2, page 15-16].

2.2.1 Parts of a Neural Network

A neural network consists of three main components: an input layer, one or more hidden layers, and an output layer, as illustrated in Figure 2.

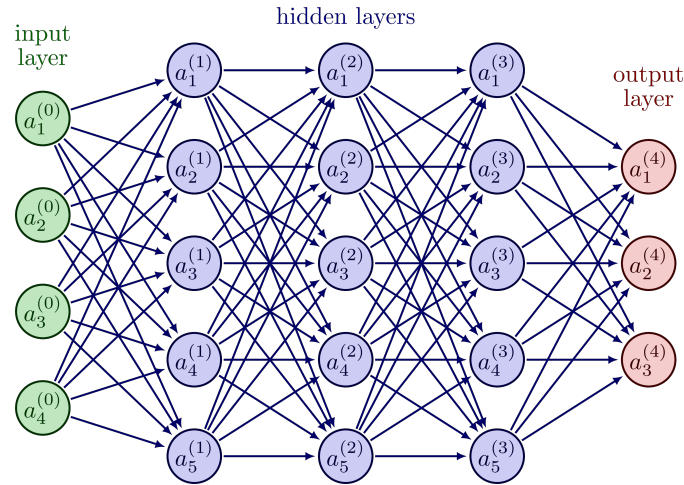


Figure 2: Example of a neural network [3].

Each neuron in the network receives input values x_i , each associated with a corresponding weight w_i , which determines the relative importance of that input. Additionally, a bias term b is added to the weighted sum, allowing the neuron to adjust its output independently of the input values. The resulting value z is then passed through an activation function ϕ , which introduces non-linearity and enables the network to model complex relationships. Mathematically, this process can be described in Equation 1 [2, page 168-172]:

$$z = \sum_{i=1}^n w_i x_i + b \implies \text{output} = \phi(z) \quad (1)$$

Together, these components allow neural networks to model data patterns and solve machine learning tasks.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network designed for analyzing data with spatial or temporal structure, such as images or time series [2, page 254]. Unlike fully connected networks, CNNs apply convolutional filters that move across the input data, allowing the network to detect local patterns and spatial hierarchies [2, page 330]. Figure 3 illustrates a 2D convolution operation.

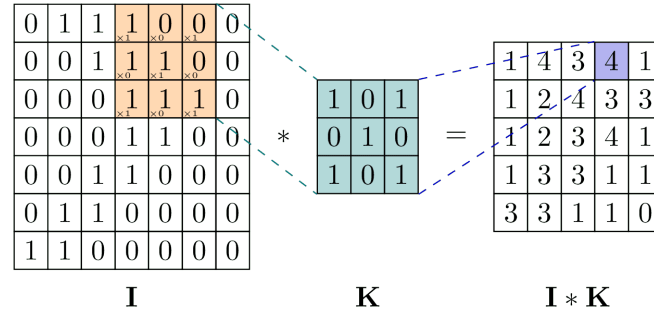


Figure 3: Example of a 2D convolution operation. A small filter (K) slides across the input matrix (I), computing a weighted sum at each location. This produces a feature map (I * K) where local patterns, such as edges or textures, are highlighted. Convolution enables the network to extract spatial features while keeping the number of parameters manageable [4].

CNNs are especially well-suited for hyperspectral images because these images contain both spatial and spectral dimensions. In our case, the 1D-JustoLiuNet is used to process the spectral signature of each pixel independently, focusing solely on the spectral axis.

2.3 1D-JustoLiuNet

1D-JustoLiuNet is a lightweight one-dimensional convolutional neural network designed for processing hyperspectral image data [5, page 3-4]. The network consists of four sequential convolutional layers, each followed by a ReLU activation function. The ReLU operation is illustrated in algorithm 1 [2, page 175].

Algorithm 1: ReLU Activation Function

Input: Feature vector $x = [x_1, x_2, \dots, x_n]$

Output: Activated vector $y = [y_1, y_2, \dots, y_n]$

```

1 for  $i \leftarrow 1$  to  $n$  do
2    $y_i \leftarrow \max(0, x_i)$ ;
3 return  $y$ 

```

After the ReLU activation, a 1D MaxPooling operation is initialized to reduce the dimensionality of the feature maps while preserving important spectral features. This operation is illustrated in algorithm 2 [2, page 339-342].

Algorithm 2: 1D MaxPooling

Input: Feature vector $x = [x_1, x_2, \dots, x_n]$, pool size k

Output: Pooled vector y

```

1 Initialize  $y \leftarrow []$ ;
2 for  $i \leftarrow 0$  to  $n - k$  do
3    $window \leftarrow [x_i, x_{i+1}, \dots, x_{i+k-1}]$ ;
4    $max \leftarrow \max(window)$ ;
5   Append  $max$  to  $y$ ;
6 return  $y$ 

```

After the convolutional stages, the feature maps are flattened into a single vector, by

flattening it. The flattening algorithm is illustrated in algorithm 3 [2, page 207].

Algorithm 3: Flattening a 3D Tensor

Input: 3D tensor T with shape (B, C, L)

Output: 2D matrix M with shape $(B, C \cdot L)$

```

1 for each sample  $T_b$  in batch do
2    $M_b \leftarrow \text{Flatten}(T_b)$ ;
3   Add  $M_b$  as row in  $M$ ;
4 return  $M$ 

```

Lastly, the vector is passed through a fully connected layer that outputs the final class predictions [2, page 203]. This algorithm is illustrated in algorithm 4 [2, page 203].

Algorithm 4: Fully Connected Layer

Input: Vector $x = [x_1, \dots, x_m]$, weight matrix W of shape $(n \times m)$, bias vector b of length n

Output: Output vector $y = [y_1, \dots, y_n]$

```

1 for  $i \leftarrow 1$  to  $n$  do
2    $y_i \leftarrow W_i \cdot x + b_i$ ;
3 return  $y$ 

```

With all the algorithms now defined, the forward pass of the 1D-JustoLiuNet can be defined in algorithm 5 [5, page 11].

Algorithm 5: Forward pass of the 1D-JustoLiuNet

Input: Input spectral vector $x \in \mathbb{R}^{1 \times N}$

Output: Predicted class probabilities $y \in \mathbb{R}^C$

```

1  $x \leftarrow \text{Conv1d}(x, \text{kernel}_1)$  ; // First convolution layer
2  $x \leftarrow \text{ReLU}(x)$ 
3  $x \leftarrow \text{MaxPool1d}(x)$ 
4  $x \leftarrow \text{Conv1d}(x, \text{kernel}_2)$  ; // Second convolution layer
5  $x \leftarrow \text{ReLU}(x)$ 
6  $x \leftarrow \text{MaxPool1d}(x)$ 
7  $x \leftarrow \text{Conv1d}(x, \text{kernel}_3)$  ; // Third convolution layer
8  $x \leftarrow \text{ReLU}(x)$ 
9  $x \leftarrow \text{MaxPool1d}(x)$ 
10  $x \leftarrow \text{Conv1d}(x, \text{kernel}_4)$  ; // Fourth convolution layer
11  $x \leftarrow \text{ReLU}(x)$ 
12  $x \leftarrow \text{MaxPool1d}(x)$ 
13  $x \leftarrow \text{Flatten}(x)$ 
14  $y \leftarrow \text{Linear}(x)$  ; // Fully connected layer
15 return  $y$ 

```

2.4 Preprocessing

Before feeding the hyperspectral input to the neural network, the raw spectral values are normalized to ensure numerical stability and to bring all input features onto a common scale. In this project, min-max normalization is applied to each pixel. Equation 2 defines the min-max normalization.

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min} + \epsilon}, \quad \epsilon \ll 1 \quad (2)$$

The normalization happens band-by-band, meaning that for instance band 29 in pixel X is normalized for all band 29 in the whole image. This transformation maps the values to a range close to $[0, 1]$. To avoid division by zero, a small constant $\epsilon \ll 1$ is added in the denominator.

2.5 Dataset and Format

The dataset used in this project consists of raw hyperspectral images stored in Band Interleaved by Pixel (BIP) format. In this structure, the complete spectral profile of each pixel is stored sequentially, which is ideal for spectral analysis on a per-pixel basis. These `.bip` files are preprocessed and loaded into memory as tensors for input to the neural network.

In addition to the raw data, label data is stored in `.dat` files, where each entry corresponds to a pixel-wise class annotation (sea, land or cloud). These `.dat` files were created semi-automatically using [ENVI](#). The labels are used in training to guide the network in learning class-specific spectral patterns.

2.6 Model Training Setup

The training pipeline follows the sequence shown in Figure 4.

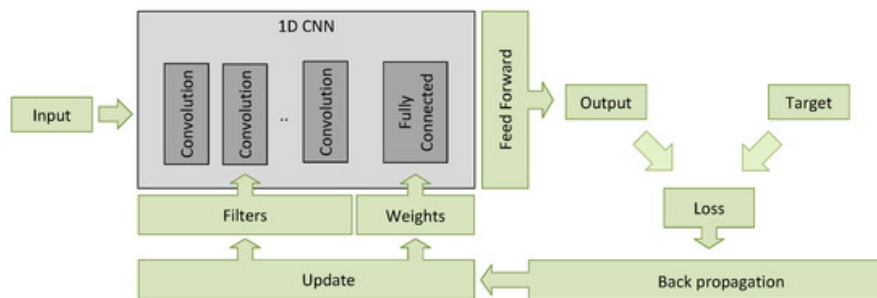


Figure 4: The training pipeline utilized in this project [6].

As illustrated in Figure 4, the CNN gets an input, and uses that input, compares with the target and updated the filters and weights of the trained CNN. By saving the filters and weights, inference can be done with the CNN. Inference is the process of the CNN classifying an image based on its trained model. In simpler terms, good filters and weights are the parameters needed for the 1D-JustoLiuNet in the HYPSO-2 satellite.

The training pipeline relies on several key hyperparameters that control how the model learns. Batch size defines how many training samples are processed before the model updates its internal parameters. It affects memory usage and the stability of gradient estimates. Learning rate determines how much the model's weights are adjusted in response to the estimated error each time the weights are updated. It plays an important role in convergence speed and stability. Label smoothing is a regularization technique that slightly adjusts the target labels, preventing the model from becoming overly confident in its predictions and improving generalization. Epochs refer to the number of complete passes through the entire training dataset during training, allowing the model to iteratively refine its performance. Table 1 has these variables and its values for this project.

Table 1: Variables used in training pipeline and its values.

Variable	Value
Batch size	128
Learning rate	0.001
Label smoothing	0.1
Epochs	10

In addition, the pipeline uses **AdamW** as the optimizer, **StepLR** as the scheduler, and **CrossEntropyLoss** as the loss function. Section 2.6.1, 2.6.2 and 2.6.3 explain these parts of the pipeline.

2.6.1 Optimizer - AdamW

The optimizer is responsible for updating the model's parameters during training to minimize the loss function. **AdamW** is a variant of the **Adam** optimizer that decouples weight decay from the gradient update, improving generalization performance. It's needed because without an optimizer, the network cannot learn. **AdamW** was chosen because it performs well on sparse gradients and allows for weight decay regularization. Algorithm 6 illustrates the **AdamW** optimizer [7].

Algorithm 6: AdamW Optimizer

Input: Gradient g_t , parameters w_t , learning rate α , momenta β_1, β_2 , weight decay λ

Output: Updated weights w_{t+1}

- 1 $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
 - 2 $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
 - 3 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
 - 4 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
 - 5 $w_{t+1} \leftarrow w_t - \alpha(\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda w_t)$
-

2.6.2 Scheduler - StepLR

A learning rate scheduler dynamically adjusts the learning rate during training to help the model converge better. **StepLR** reduces the learning rate by a fixed factor every

few epochs. This was beneficial to the pipeline because using a static learning rate can cause overshooting or slow convergence. **StepLR** helps refine the training as it progresses. Algorithm 7 shows how the **StepLR** scheduler works [7].

Algorithm 7: StepLR Scheduler

Input: Initial learning rate α_0 , step size s , gamma γ , epoch t

Output: Learning rate α_t

```

1 if  $t \% s == 0$  then
2    $\alpha_t \leftarrow \alpha_0 \cdot \gamma^{\lfloor t/s \rfloor}$ 
3 else
4    $\alpha_t \leftarrow \alpha_{t-1}$ 

```

2.6.3 Loss Function - CrossEntropyLoss

The loss function measures how far the model's predictions are from the true labels. **CrossEntropyLoss** is commonly used for multi-class classification problems. It's needed because the loss function drives the learning. **CrossEntropyLoss** is ideal for classification since it heavily penalizes incorrect confident predictions. Algorithm 8 shows how **CrossEntropyLoss** works [7].

Algorithm 8: CrossEntropyLoss

Input: Predicted vector $p = [p_1, \dots, p_C]$, true class label y

Output: Loss $\mathcal{L}$

```

1  $\mathcal{L} \leftarrow -\log(p_y)$ 

```

2.7 Classification Metrics

To evaluate the model's performance, several standard classification metrics are used. These metrics quantify different aspects of the prediction quality and are based on the concepts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

2.7.1 Accuracy

Accuracy measures the overall proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

It indicates how often the model correctly predicts the class compared to the total number of predictions.

2.7.2 Precision

Precision measures how many of the predicted positive instances are actually correct:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4)$$

High precision means that when the model predicts a certain class, it is usually correct.

2.7.3 Recall

Recall measures how many of the actual positive instances were correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (5)$$

High recall means the model successfully finds most instances of the class.

2.7.4 F1-Score

The F1-score is the harmonic mean of precision and recall:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6)$$

It balances precision and recall, especially useful when there is class imbalance.

2.7.5 Macro Average

Macro averaging computes the metric independently for each class and then takes the unweighted mean:

$$\text{Macro Avg} = \frac{1}{N} \sum_{i=1}^N \text{Metric}_i \quad (7)$$

where N is the number of classes. Each class contributes equally, regardless of its support.

2.7.6 Weighted Average

Weighted averaging computes the metric for each class and weights it by the number of instances (support) in each class:

$$\text{Weighted Avg} = \frac{\sum_{i=1}^N \text{Support}_i \times \text{Metric}_i}{\sum_{i=1}^N \text{Support}_i} \quad (8)$$

This ensures that classes with more instances have a greater impact on the final score.

2.7.7 Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model. It compares the predicted labels against the true labels and is structured as follows:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

2.8 Evaluation Methodology

To evaluate the performance of the model, the dataset was selected to include both relatively easy and more challenging images. Easy images are those that share similar characteristics with the training data. For instance, images from the same general geographic region. These help assess how well the model performs under familiar conditions.

Challenging images, on the other hand, include examples that differ more noticeably from the rest. This could involve, for example, spectral data from different climates or terrain types. Such variation introduces a moderate level of difficulty and helps test the model's ability to generalize beyond the conditions it was trained on.

This approach ensures that the evaluation captures both the model's learning effectiveness and its robustness, without being overly dependent on similarity to the training data.

2.9 GPU Acceleration

GPUs (Graphics Processing Units) are essential for accelerating deep learning workloads due to their massively parallel architecture, allowing thousands of operations to be executed concurrently [8]. In this project, GPU acceleration was leveraged to reduce training time for the 1D-CNN model by parallelizing matrix operations such as convolutions, activations, and gradient computations during backpropagation.

The model was trained using the NVIDIA GeForce RTX 3080 GPU with CUDA (Compute Unified Device Architecture), a parallel computing platform and API developed by NVIDIA. CUDA allows developers to write code that runs directly on the GPU, providing access to hundreds or thousands of GPU cores for simultaneous computation. In frameworks like PyTorch, CUDA handles tensor operations by distributing the work across multiple threads, optimizing both memory access and computational throughput [8].

CUDA-enabled training not only speeds up the forward and backward passes of the CNN, but also ensures efficient utilization of GPU memory and compute resources. This is particularly important for hyperspectral data, which involves high-dimensional inputs that benefit greatly from parallel processing.

2.10 Code Implementation Diagrams

After presenting the components and flow of the system, it is useful to include diagrams that provides an overview of the code structure. Firstly, a component diagram was created to illustrate the main functional parts of the system and how they are interconnected. This diagram highlights the overall structure of the codebase, showing the key modules responsible for data management, model training, evaluation, and utility functions. The diagram is provided in Figure 5.

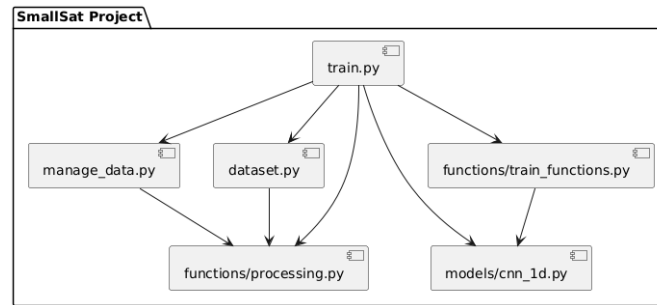


Figure 5: Component diagram of code.

Secondly, a sequence diagram was developed to describe the typical runtime behavior of the system during model training. This diagram captures the flow of operations between the main classes and functions, from data loading to training loop iterations and evaluation steps. It shows how control moves dynamically through the system during execution. The diagram is provided in Figure 6.

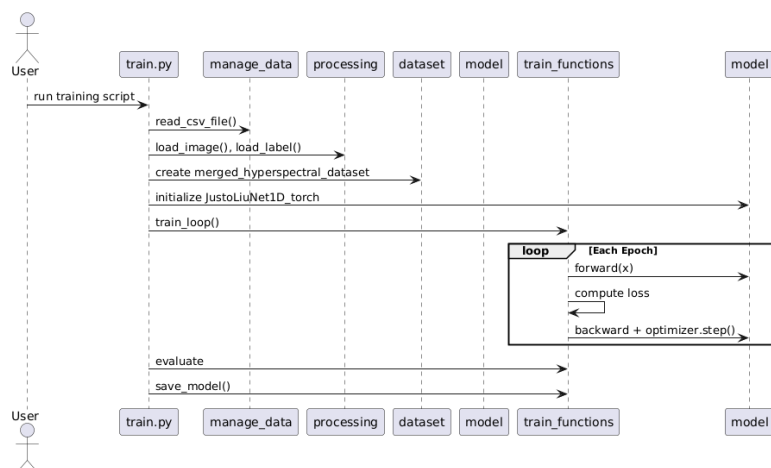


Figure 6: Sequence diagram of code.

Thereafter, an activity diagram was designed to present the logical workflow of the entire training and evaluation process. It focuses on the decision points, data transformations, and actions performed throughout the pipeline. The activity diagram complements the previous diagrams by providing a detailed view of the operational logic and conditional flows in the project. The diagram is provided in Figure 7.

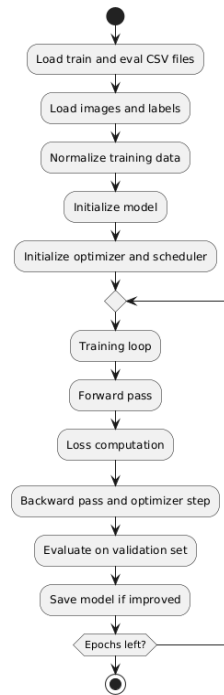


Figure 7: Activity diagram of code.

Lastly, the class diagram summarizes how the different parts of the codebase are organized and how they interact with each other. Figure 8 shows the structure of the implemented codebase.

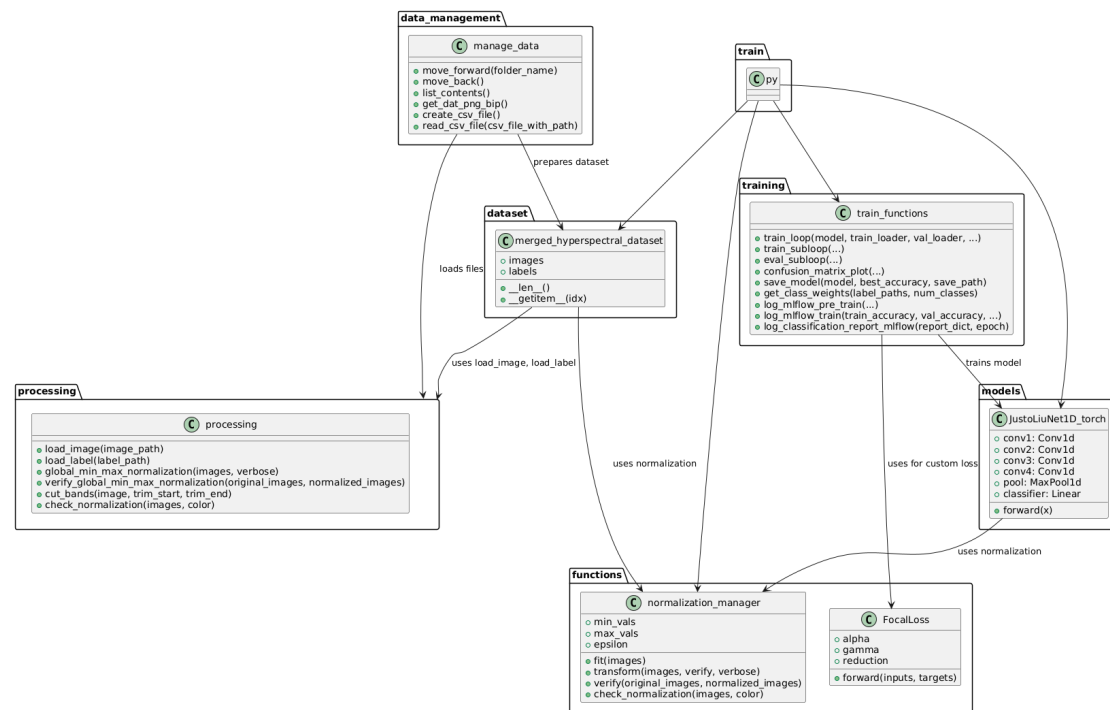


Figure 8: Class diagram of code. Note that some elements have been shortened to keep the diagram understandable (see Figure 22 for complete image). In addition, `FocalLoss` has been implemented in the code, but isn't used, that's why it's present here, but isn't noted in Section 2.6.

The code itself can be found in appendix A.

3 Verification and Test

3.1 Tests

Table 2 outlines the tests presented in this report. There are several variables set in Table 2, their meanings are:

1. **Sanity test:** A basic check to verify that the model and data pipeline function correctly.
2. **Similarity test:** A test where the evaluation images are similar to the training data, assessing how well the model generalizes under low variation.
3. **Diversity test:** A test where both the training and evaluation sets are sampled from a pool of highly varied images, meaning the images within and between the sets have low visual similarity, testing the model's ability to generalize across diverse inputs.
4. **Difficulty level:** An indicator of how hard a test is, based on the similarity between training and evaluation data.
5. **Similarity index:** A measure indicating whether the images within the dataset are visually similar to each other, where 1 means high internal similarity and 0 means high diversity.

Table 2: Overview of defined tests, difficulty levels, and similarity indices.

Test ID	Test type	Difficulty level	Images in test	Similarity index
T01	Sanity test	Easy	6	1
T02	Sanity test	Easy	8	0
T03	Sanity test	Easy	25 ¹	1
T04	Similarity test	Medium	9	1
T05	Diversity test	Hard	12	0
T06	Diversity test	Hard	27	0

The results from the tests in Table 2 are presented in the subsections of Section 3.1. In addition, Appendix C contains the images chosen for each test.

3.1.1 T01

The train- and evaluation accuracy per epoch is plotted in Figure 9.

¹There's 1 image for both training and evaluating, but during training the image creates 24 copies of itself. In simpler terms, the model gets trained on 25 exact copies of one image, and gets evaluated on that same image.

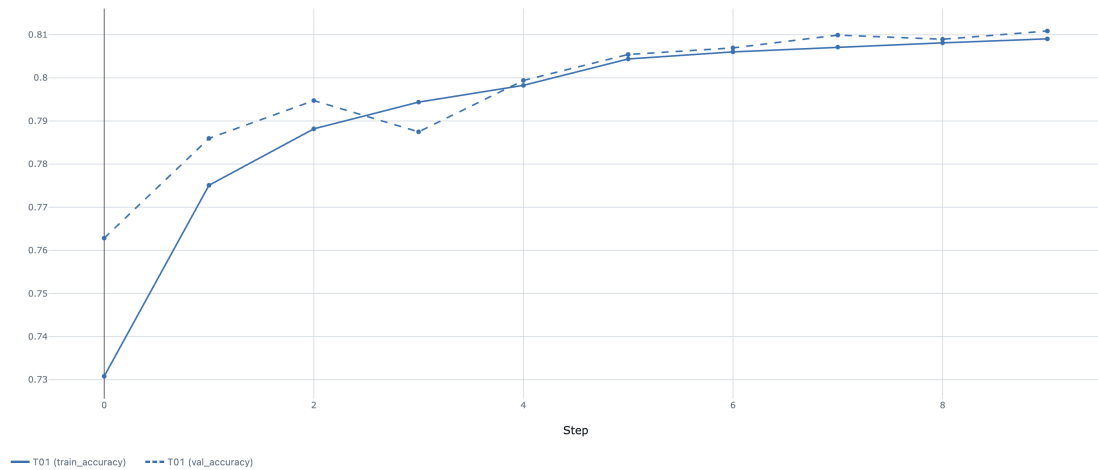


Figure 9: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T01.

In addition, the classification report from the test's final epoch has been illustrated in Table 3.

Table 3: Classification report from the final epoch. Test ID: T01.

Class	Precision	Recall	F1-Score	Support
Cloud	0.61	0.01	0.01	128413
Land	0.85	0.89	0.87	2562136
Sea	0.74	0.74	0.74	1227547
Accuracy			0.81	3918096
Macro Avg	0.73	0.54	0.54	3918096
Weighted Avg	0.80	0.81	0.80	3918096

Lastly, the last epoch's confusion matrix has been illustrated in Figure 10.

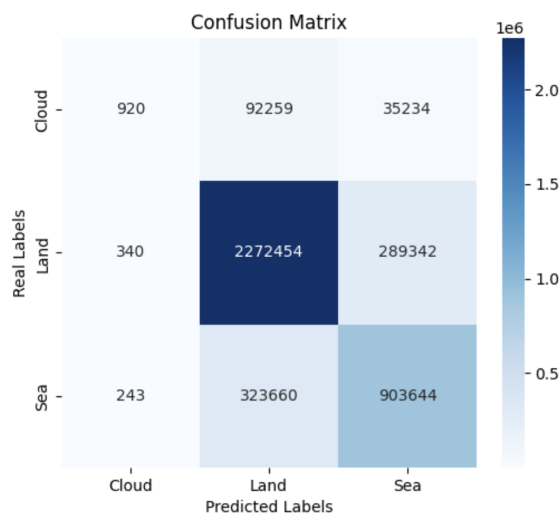


Figure 10: Confusion matrix from the final epoch. Test ID: T01.

3.1.2 T02

The train- and evaluation accuracy per epoch is plotted in Figure 11.

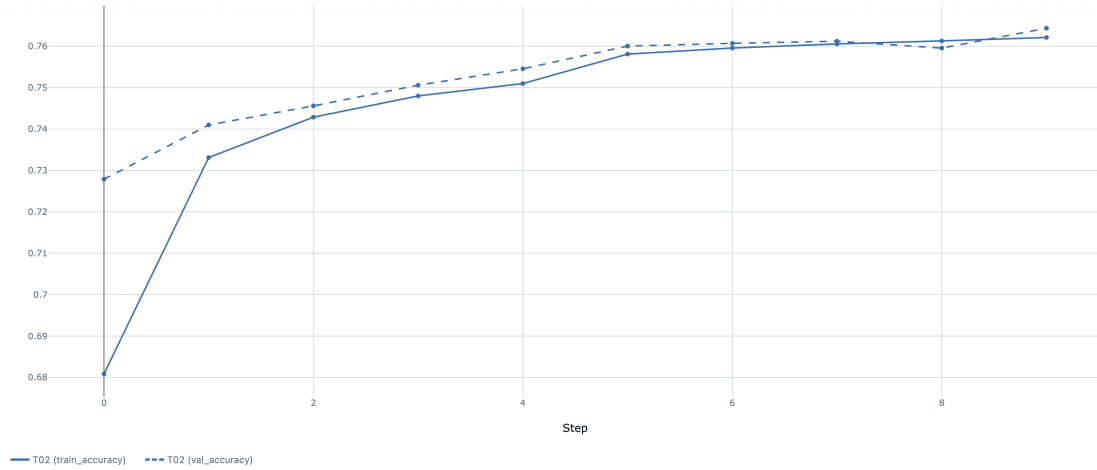


Figure 11: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T02.

In addition, the classification report from the test's final epoch has been illustrated in Table 4.

Table 4: Classification report from the final epoch. Test ID: T02.

Class	Precision	Recall	F1-Score	Support
Cloud	0.73	0.73	0.73	1019113
Land	0.82	0.79	0.81	2687515
Sea	0.69	0.73	0.71	1517500
Accuracy			0.76	5224128
Macro Avg	0.75	0.75	0.75	5224128
Weighted Avg	0.77	0.76	0.77	5224128

Lastly, the last epoch's confusion matrix has been illustrated in Figure 12.

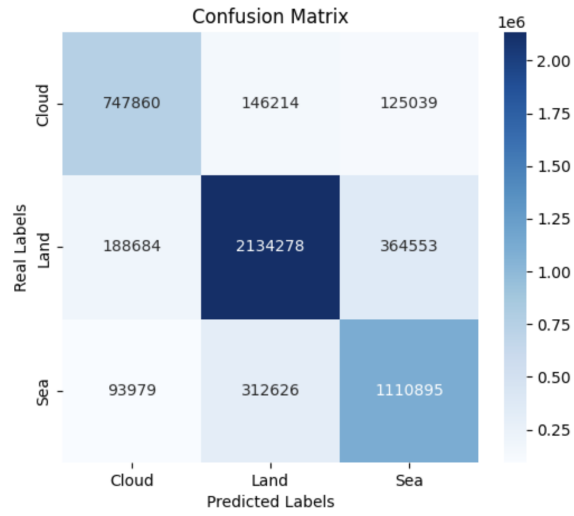


Figure 12: Confusion matrix from the final epoch. Test ID: T02.

3.1.3 T03

The train- and evaluation accuracy per epoch is plotted in Figure 13.

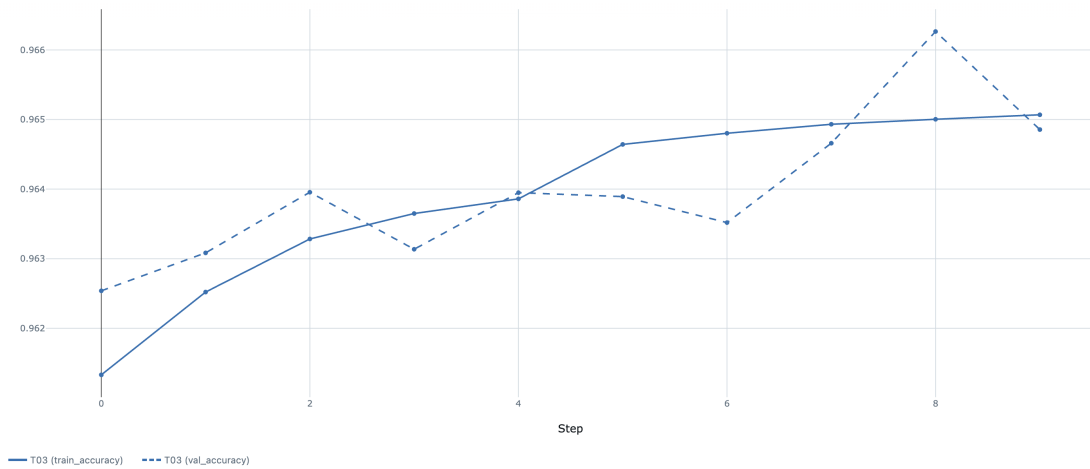


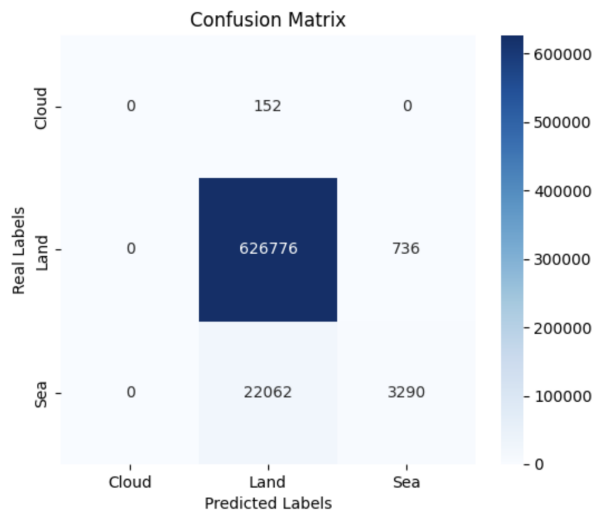
Figure 13: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T03.

In addition, the classification report from the test's final epoch has been illustrated in Table 5.

Table 5: Classification report from the final epoch. Test ID: T03.

Class	Precision	Recall	F1-Score	Support
Cloud	0.00	0.00	0.00	152
Land	0.97	1.00	0.98	627512
Sea	0.82	0.13	0.22	25352
Accuracy			0.96	653016
Macro Avg	0.59	0.38	0.40	653016
Weighted Avg	0.96	0.96	0.95	653016

Lastly, the last epoch's confusion matrix has been illustrated in Figure 14.

**Figure 14:** Confusion matrix from the final epoch. Test ID: T03.

3.1.4 T04

The train- and evaluation accuracy per epoch is plotted in Figure 15.

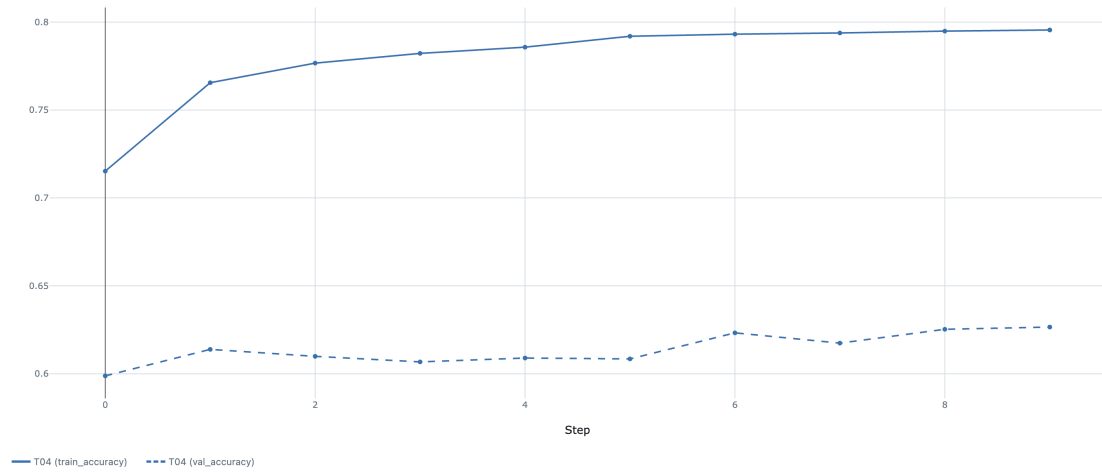


Figure 15: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T04.

In addition, the classification report from the test's final epoch has been illustrated in Table 6.

Table 6: Classification report from the final epoch. Test ID: T04.

Class	Precision	Recall	F1-Score	Support
Cloud	0.00	0.00	0.00	46921
Land	0.68	0.76	0.72	753582
Sea	0.53	0.49	0.51	505529
Accuracy			0.63	1306032
Macro Avg	0.40	0.42	0.41	1306032
Weighted Avg	0.60	0.63	0.61	1306032

Lastly, the last epoch's confusion matrix has been illustrated in Figure 16.

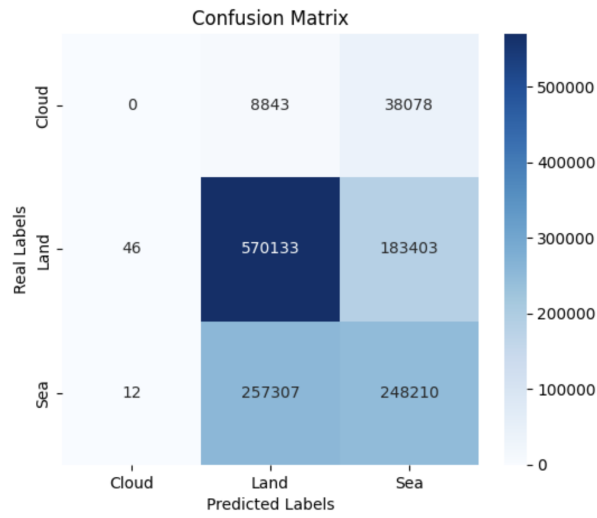


Figure 16: Confusion matrix from the final epoch. Test ID: T05.

3.1.5 T05

The train- and evaluation accuracy per epoch is plotted in Figure 17.

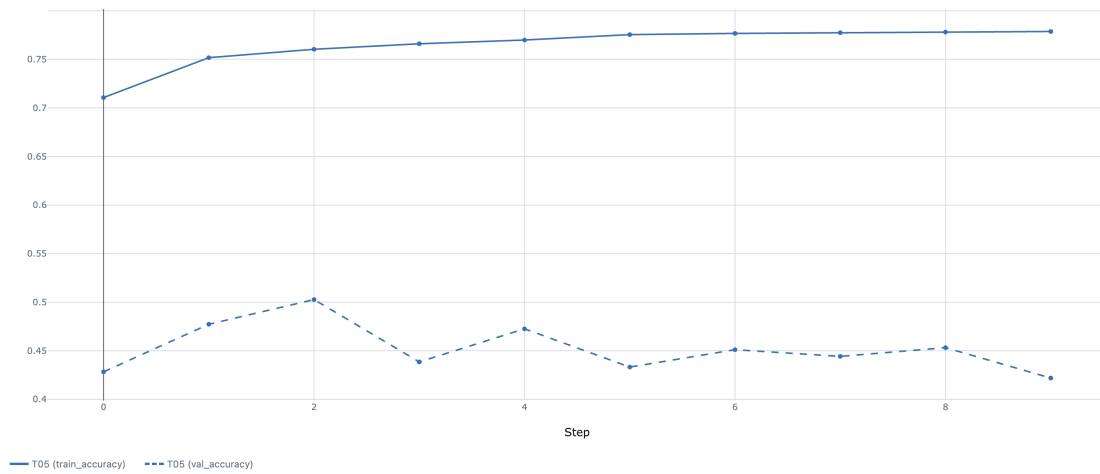


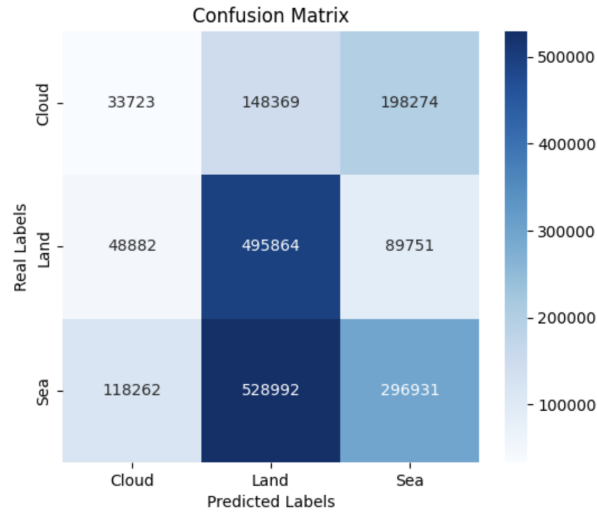
Figure 17: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T05.

In addition, the classification report from the test's final epoch has been illustrated in Table 7.

Table 7: Classification report from the final epoch. Test ID: T05.

Class	Precision	Recall	F1-Score	Support
Cloud	0.17	0.09	0.12	380366
Land	0.42	0.78	0.55	634497
Sea	0.51	0.31	0.39	944185
Accuracy			0.42	1959048
Macro Avg	0.37	0.39	0.35	1959048
Weighted Avg	0.41	0.42	0.39	1959048

Lastly, the last epoch's confusion matrix has been illustrated in Figure 18.

**Figure 18:** Confusion matrix from the final epoch. Test ID: T05.

3.1.6 T06

The train- and evaluation accuracy per epoch is plotted in Figure 19.

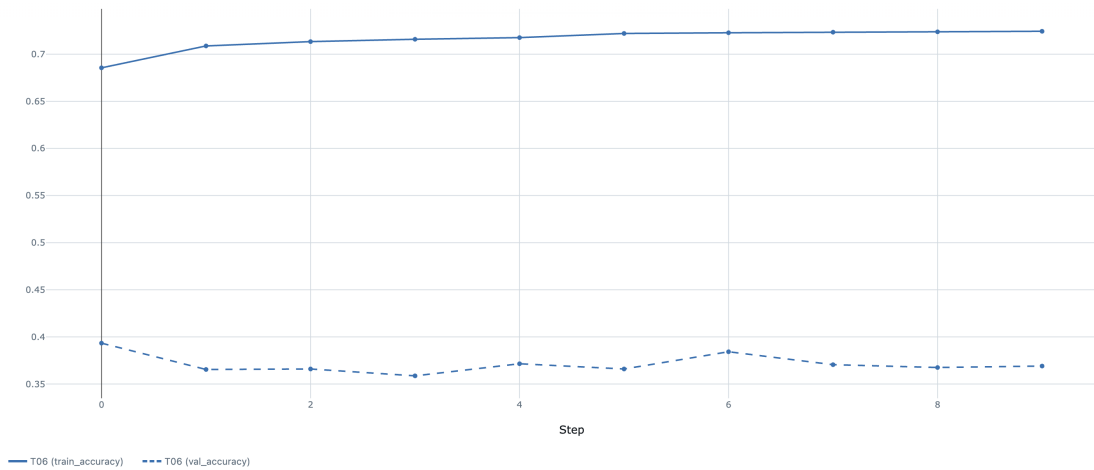


Figure 19: Train- and evaluation accuracy per epoch. Image taken from MLFlow. Test ID: T06.

In addition, the classification report from the test's final epoch has been illustrated in Table 8.

Table 8: Classification report from the final epoch. Test ID: T06.

Class	Precision	Recall	F1-Score	Support
Cloud	0.44	0.34	0.39	587647
Land	0.23	0.29	0.26	526452
Sea	0.44	0.44	0.44	844949
Accuracy			0.37	1959048
Macro Avg	0.37	0.36	0.36	1959048
Weighted Avg	0.38	0.37	0.37	1959048

Lastly, the last epoch's confusion matrix has been illustrated in Figure 20.

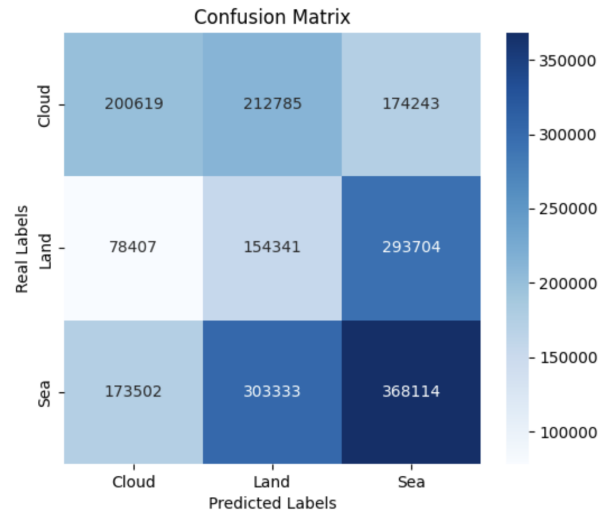


Figure 20: Confusion matrix from the final epoch. Test ID: T05.

3.2 Sources of Error

Now that all tests have been presented, the sources of error can be introduced. One major source is the corruption of the ENVI images. ENVI cannot achieve 100% accuracy when labeling images, as the process is semi-automatic. After plotting the labeled images, several errors were observed. The most notable problem is that some sea pixels, especially near clouds over the sea, have been incorrectly labeled as land when they are actually sea or cloud. An example image illustrating this issue is shown in Figure 21².

²All images are shown in appendix D.

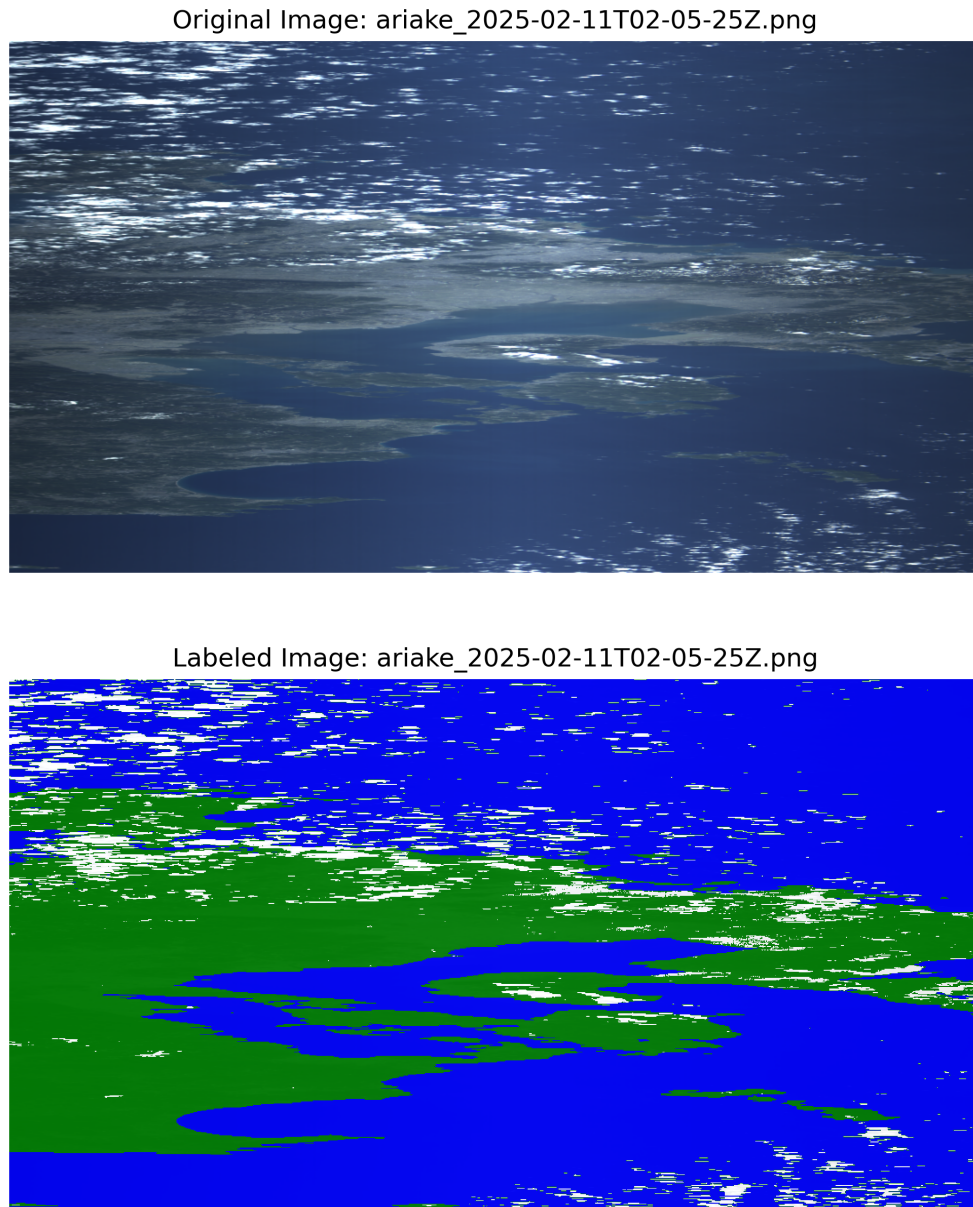


Figure 21: Original image and its plotted labeled image created with ENVI. Blue = sea, green = land, white = cloud. Note the several sea/cloud pixels that have been labeled as land.

Additionally, the original code of 1D-JustoLiuNet is structured in a way that may lead to misunderstandings or overlooked details during development, as it relies heavily on comments rather than a clearly organized codebase [9].

3.3 Why the 1D-JustoLiuNet Doesn't Work as Intended

One important factor contributing to the model's lower-than-expected performance is the use of min-max normalization applied individually to each image, band by band. While this approach ensures that each image is scaled appropriately within its own dynamic range, it can unintentionally reduce the natural variability between images. For example, a sea pixel from one location and a sea pixel from another location may

originally have significantly different spectral signatures due to differences in lighting, vegetation, or environmental conditions (see Appendix D for example images). However, because normalization is performed separately for each image, the relative differences between images can be compressed, making it harder for the model to distinguish meaningful global patterns across the entire dataset. Although per-image normalization is a standard practice in hyperspectral data processing, in this case it may have contributed to the model's difficulty in generalizing across diverse images.

Another important factor is the presence of corrupted images in the training data, as discussed in Section 3.2. These corrupted images may have introduced additional noise into the learning process, confusing the model and forcing it to learn from partially degraded or misleading data. However, although data corruption is certainly a contributing factor, it alone cannot account for the significant gap between the performance observed in this project and the 93% evaluation accuracy reported in the article *Hyperspectral Image Segmentation for Optimal Satellite Operations: In-Orbit Deployment of 1D-CNN* [5, page 4]. It is important to emphasize that the dataset used in that article consisted primarily of relatively "easy" images: scenes without snow, minimal environmental variation, and overall high visual similarity across the dataset [10]. In contrast, the datasets used in this project featured much greater diversity, both geographically and environmentally, placing a far heavier burden on the model's ability to generalize. This fundamental difference in dataset complexity likely explains a large part of the discrepancy in validation accuracy.

A third notable observation throughout all tests is the model's strong bias toward the class with the highest number of samples. Across every experiment, the class with the largest support consistently achieved the highest F1-score, while the class with the fewest examples suffered from very low precision and recall. This imbalance suggests that the model prioritizes majority classes during learning and struggles to correctly classify underrepresented classes, especially when the total amount of training data is relatively small.

Another factor that likely contributed to the limited generalization ability is the model's use of standard ReLU activation functions without incorporating dropout layers. Standard ReLU, while effective in many settings, is prone to neuron "death," where neurons stop activating entirely if they consistently receive negative inputs during training. This can reduce the network's capacity to learn a wide range of features, particularly when the dataset is small or highly varied. Using a Leaky ReLU activation function could have mitigated this issue by allowing small, nonzero gradients even for negative inputs, keeping more neurons active and contributing to the learning process [2, page 193]. Moreover, the complete absence of dropout layers meant that the model was free to memorize specific patterns from the training set without being encouraged to develop more broadly applicable feature representations. Dropout acts as a regularizer that forces the network to spread information across different neurons, improving its robustness to new and unseen inputs [2, page 258-261]. Without these mechanisms, the model was highly prone to overfitting the particular characteristics of the training data. In practice, this led the network to specialize in recognizing patterns from the training images rather than developing the generalization abilities needed to perform well on more diverse and challenging evaluation data.

In summary, the model's limited performance can be attributed to a combination of factors: the compression of natural spectral differences due to per-image min-max nor-

malization, the presence of corrupted and inconsistent training labels, the increased complexity and variability of the evaluation datasets compared to previous studies, and architectural choices that led to specialization rather than generalization. Together, these issues prevented the model from achieving strong validation results on more diverse and challenging data.

3.4 Future Work and Recommendations

One of the primary observations from this project is that relying solely on spectral signatures for classification becomes increasingly difficult when the images exhibit large variations in environmental conditions, such as lighting, atmospheric effects, and geographic features. While spectral information alone can perform well under consistent and controlled conditions, its effectiveness drops significantly when faced with highly diverse and complex datasets.

To address this, future work should consider incorporating spatial information alongside spectral features. Instead of treating each pixel independently, models could be designed to also consider the neighboring pixel context, allowing them to capture local textures and patterns that purely spectral information might miss. This could be achieved by moving from 1D CNNs operating on spectral data, to 2D or even 3D CNN architectures that process spatial and spectral data together.

Another valuable direction would be to directly test the model's performance on the dataset from the article *Hyperspectral Image Segmentation for Optimal Satellite Operations: In-Orbit Deployment of 1D-CNN*, not just compare validation accuracies reported. This would allow for a fairer evaluation of whether the model struggles primarily because of differences in dataset difficulty or due to architectural limitations.

4 Conclusion

This project set out to adapt and evaluate the 1D-JustoLiuNet for the classification of hyperspectral images from the HYPSONO-2 satellite. Through extensive testing, it was found that while the model can learn meaningful patterns under controlled conditions, its performance drops significantly on more diverse and challenging datasets. Several factors contributed to this outcome, including per-image min-max normalization compressing spectral variability, corrupted and inconsistent training labels, increased dataset complexity, and architectural limitations favoring specialization over generalization.

The findings highlight the challenges of relying solely on spectral information for pixel-wise classification when working with real-world satellite data characterized by large environmental variability. Future improvements should focus on combining spectral and spatial features, exploring alternative CNN architectures, and performing fairer benchmarking against previous datasets.

Despite the limitations observed, the work presented here provides valuable insights into the specific difficulties of hyperspectral segmentation for operational satellite use and lays the foundation for future improvements in both methodology and model design.

5 Acknowledgments

I would like to express my sincere gratitude to the NTNU SmallSat Lab team for their continuous support, guidance, and collaboration throughout this project. Their expertise and encouragement were invaluable to the development and completion of this work.

References

- [1] ePortals. *Hyperspectral Imaging*. Version 1.0. 2023.
- [2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [3] Izaak Neutelings. *Neural Networks*. 2025. URL: https://tikz.net/neural_networks/#full_code.
- [4] Janosh Riebesell. *Convolution Operator*. 2025. URL: <https://tikz.net/conv2d/>.
- [5] Jon Alvarez Justo et al. “Hyperspectral Image Segmentation for Optimal Satellite Operations: In-Orbit Deployment of 1D-CNN”. In: *Remote Sensing* 17.4 (2025), p. 642. DOI: [10.3390/rs17040642](https://doi.org/10.3390/rs17040642). URL: <https://doi.org/10.3390/rs17040642>.
- [6] Ilaria Cacciari and Anedio Ranfagni. *Hands-On Fundamentals of 1D Convolutional Neural Networks—A Tutorial for Beginner Users*. Version 1.0. 2024.
- [7] CoderzColumn. *PyTorch: Learning Rate Schedules*. 2022. URL: <https://coderzcolumn.com/tutorials/artificial-intelligence/pytorch-learning-rate-schedules>.
- [8] Jorge Félix Martínez Pazos. *GPU Accelerated Deep Learning with PyTorch on Windows*. 2024. URL: <https://medium.com/@jorgefmp.mle/gpu-accelerated-deep-learning-with-pytorch-on-windows-519898e4c283>.
- [9] Jon Alvarez Justo et al. “Sea-Land-Cloud Segmentation in Satellite Hyperspectral Imagery by Deep Learning”. In: *arXiv preprint arXiv:2310.16210* (2023).
- [10] Jon A Justo et al. “An Open Hyperspectral Dataset with Sea-Land-Cloud Ground-Truth from the Hypso-1 Satellite”. In: *2023 13th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. IEEE, 2023, pp. 1–5.

A Code

The code is divided into multiple files, where each file complements the others, and the final execution is performed in `train.py`. If you want to run the code, clone this [GitHub repository](#). In addition, the code is provided below.

For a better understanding of the code, please see Section 2.10.

A.1 `libraries.py`

```
1  # libraries.py - Common file for all libraries
2
3  # System libraries
4  import os
5  import sys
6  import time
7  import random
8  import warnings
9
10 # Numerical calculations and data processing
11 import numpy as np
12 import pandas as pd
13 import cupy as cp
14 import csv
15 import random as rd
16 from pathlib import Path
17
18 # Visualization
19 import matplotlib.pyplot as plt
20 from matplotlib import image as mpimg
21 import seaborn as sns
22 from PIL import Image
23 import matplotlib.colors as mcolors
24 import matplotlib.patches as mpatches
25 import spectral
26
27 # Machine Learning
28 from sklearn.model_selection import train_test_split
29 from sklearn.preprocessing import StandardScaler, LabelEncoder
30 from sklearn.naive_bayes import GaussianNB
31 from sklearn.linear_model import SGDClassifier
32 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
33     ↪ QuadraticDiscriminantAnalysis
34 from sklearn.metrics import accuracy_score, classification_report,
35     ↪ confusion_matrix
36 from sklearn.decomposition import PCA
37 import joblib
38
39 # Deep Learning (PyTorch, Keras)
```

```

38 import torch
39 import torch.nn as nn
40 import torch.optim as optim
41 import torch.nn.functional as F
42 from torch.utils.data import DataLoader, Dataset
43 from torchvision import transforms
44 from torch.cuda.amp import autocast, GradScaler
45 import torch.multiprocessing as mp
46 from torch.cuda import amp
47 from torch.optim.lr_scheduler import ReduceLROnPlateau
48
49 # Weight Adjusters
50 from collections import Counter
51
52 # Optimizers
53 from ranger_adabelief import RangerAdaBelief
54
55 # HYPSO Package
56 import hypso
57 from hypso import Hypso2
58 from hypso.load import load_l1a_nc_cube # Raw
59 from hypso.load import load_l1b_nc_cube # Radiance
60 from hypso.load import load_l1c_nc_cube # Reflectance
61 from hypso.load import load_l1d_nc_cube # Reflectance
62
63 # Progress Bar
64 from tqdm import tqdm
65
66 # Cache
67 sys.dont_write_bytecode = True
68 torch.cuda.empty_cache()
69
70 # Logger
71 import logging
72 logging.basicConfig(level=logging.DEBUG)
73 logging.getLogger('matplotlib').setLevel(logging.ERROR)
74 from termcolor import colored
75 warnings.filterwarnings("ignore")
76 import mlflow
77 import mlflow.pytorch
78
79 print("Libraries are loaded!")

```

A.2 cnn_1d.py

```

1 import sys
2 import os
3

```



```

4 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↪ '..')))
5
6 from libraries import *
7
8 #####
9 #####
10 #####
11
12 class JustoLiuNet1D_torch(nn.Module):
13     def __init__(self, num_features, num_classes=3, kernel_size=6,
↪ starting_kernels=6):
14         super(JustoLiuNet1D_torch, self).__init__()
15
16         self.conv1 = nn.Conv1d(1, starting_kernels,
↪ kernel_size=kernel_size)
17         self.pool = nn.MaxPool1d(kernel_size=2)
18         self.conv2 = nn.Conv1d(starting_kernels, starting_kernels * 2,
↪ kernel_size=kernel_size)
19         self.conv3 = nn.Conv1d(starting_kernels * 2, starting_kernels *
↪ 3, kernel_size=kernel_size)
20         self.conv4 = nn.Conv1d(starting_kernels * 3, starting_kernels *
↪ 4, kernel_size=kernel_size)
21
22         self.classifier = nn.Linear(48, num_classes)
23
24     def forward(self, x):
25         x = F.relu(self.conv1(x))
26         x = self.pool(x)
27
28         x = F.relu(self.conv2(x))
29         x = self.pool(x)
30
31         x = F.relu(self.conv3(x))
32         x = self.pool(x)
33
34         x = F.relu(self.conv4(x))
35         x = self.pool(x)
36
37         #print("DEBUG - Shape before flatten:", x.shape)
38
39         x = x.view(x.size(0), -1)
40
41         if self.classifier is None:
42             raise RuntimeError(f"'classifier' is not defined yet. Set
↪ Linear input dim to {x.size(1)} in __init__.")
43
44         return self.classifier(x)

```

A.3 processing.py

```

1  import sys
2  import os
3
4  sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↵      '..')))
5
6  from libraries import *
7
8  #####
9  #####
10 #####
11
12 def load_image(image_path, HEIGHT=598, WIDTH=1092, BANDS=120):
13     with open(image_path, 'rb') as f:
14         image = cp.fromfile(f, dtype=cp.uint16)
15         image = cp.asnumpy(image)
16         image = image.reshape((BANDS, HEIGHT, WIDTH))
17         image = image.transpose((1, 2, 0))
18         image = image.reshape((-1, BANDS))
19
20     image = cut_bands(image)
21
22     return image
23
24 #####
25
26 def load_label(label_path, HEIGHT=598, WIDTH=1092):
27     label = np.fromfile(label_path, dtype=np.uint8)
28     label = label.reshape((HEIGHT, WIDTH))
29     label = label - 1
30     label = label.flatten()
31     return label
32
33 #####
34
35 def global_min_max_normalization(images, verbose=False):
36     """
37     Normalize the images using global per-band min-max normalization.
↵     That is,
38     the minimum and maximum values are calculated separately for each
↵     spectral band
39     across all pixels in the dataset.
40     """
41     min_values = images.min(dim=0).values
42     max_values = images.max(dim=0).values
43
44     if verbose:

```

```

45     print(colored(f"Checking spectra of the first 10 pixels.",
46         ↪ "red"))
47     check_normalization(images, "red")
48     print("\n")
49
50     new_images = (images - min_values) / (max_values - min_values +
51         ↪ 1e-8)
52
53     verify_global_min_max_normalization(images, new_images)
54
55     if verbose:
56         print("\n")
57         print(colored(f"Checking spectra of the first 10 pixels after
58             ↪ min-max normalization.", "green"))
59         check_normalization(new_images, "green")
60
61     return new_images
62
63 #####
64
65 def verify_global_min_max_normalization(original_images,
66     ↪ normalized_images, epsilon=1e-8):
67     min_vals = original_images.min(dim=0).values
68     max_vals = original_images.max(dim=0).values
69
70     expected = (original_images - min_vals) / (max_vals - min_vals +
71         ↪ epsilon)
72
73     if not torch.allclose(normalized_images, expected, atol=1e-6):
74         diffs = torch.abs(normalized_images - expected)
75         max_diff = diffs.max().item()
76         print(colored(f"Normalization mismatch! Maximum deviation:
77             ↪ {max_diff}", "red"))
78         raise AssertionError("Normalization doesn't follow the
79             ↪ formula.")
80     else:
81         print(colored("All pixels are correctly normalized.", "green"))
82
83 #####
84
85 def check_normalization(images, color):
86     br1 = 0
87     for i in range(images.shape[0]):
88         print(colored(images[i], color))
89         if br1 == 10:
90             break
91         br1 += 1
92
93 #####

```

```

88 def cut_bands(image, trim_start=3, trim_end=117):
89     return image[:, trim_start:trim_end]
90
91 #####
92
93 class normalization_manager:
94     def __init__(self, epsilon=1e-8):
95         self.min_vals = None
96         self.max_vals = None
97         self.epsilon = epsilon
98
99     def fit(self, images):
100         self.min_vals = images.min(dim=0).values
101         self.max_vals = images.max(dim=0).values
102         print(colored("Fitted normalization parameters from training
    ↪ data.", "green"))
103
104     def transform(self, images, verify=False, verbose=False):
105         if self.min_vals is None or self.max_vals is None:
106             raise ValueError("NormalizationManager not fitted. Call
    ↪ fit(images) first.")
107
108         if verbose:
109             print(colored(f"Checking spectra of the first 10 pixels.",
    ↪ "red"))
110             self.check_normalization(images, "red")
111             print("\n")
112
113         norm_images = (images - self.min_vals) / (self.max_vals -
    ↪ self.min_vals + self.epsilon)
114
115         if verify:
116             self.verify(images, norm_images)
117
118         if verbose:
119             print("\n")
120             print(colored(f"Checking spectra of the first 10 pixels
    ↪ after min-max normalization.", "green"))
121             self.check_normalization(norm_images, "green")
122
123         return norm_images
124
125     def verify(self, original_images, normalized_images):
126         expected = (original_images - self.min_vals) / (self.max_vals -
    ↪ self.min_vals + self.epsilon)
127
128         if not torch.allclose(normalized_images, expected, atol=1e-6):
129             diffs = torch.abs(normalized_images - expected)
130             max_diff = diffs.max().item()

```

```

131         print(colored(f"Normalization mismatch! Maximum deviation:
    ↪ {max_diff}", "red"))
132         raise AssertionError("Normalization doesn't follow the
    ↪ formula.")
133     else:
134         print(colored("All pixels are correctly
    ↪ normalized.", "green"))
135
136     def check_normalization(self, images, color):
137         br1 = 0
138         for i in range(images.shape[0]):
139             print(colored(images[i], color))
140             if br1 == 10:
141                 break
142             br1 += 1

```

A.4 train_functions.py

```

1  import sys
2  import os
3
4  sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
    ↪ '..')))
5
6  from libraries import *
7  from functions.processing import load_label
8
9  #####
10 #####
11 #####
12
13 def train_loop(model, train_loader, val_loader, criterion, optimizer,
    ↪ scheduler, device,
14                 save_path="models/best_model.pth", num_epochs=30):
15     best_accuracy = 0.0
16
17     classes = ["Cloud", "Land", "Sea"]
18     print(f"DEBUG - Number of training batches: {len(train_loader)}")
19
20     for epoch in range(num_epochs):
21         model.train()
22         total_loss = 0.0
23         labels_per_epoch = []
24         predictions_per_epoch = []
25
26         loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}",
    ↪ leave=False, colour="red")
27

```

```

28     # Training
29     train_accuracy, total_loss = train_subloop(
30         loop, model, criterion, optimizer, device,
31         predictions_per_epoch, labels_per_epoch, total_loss
32     )
33
34     # Evaluation
35     model.eval()
36     all_labels_eval = []
37     all_preds_eval = []
38
39     with torch.no_grad():
40         all_labels_eval, all_preds_eval, val_loss = eval_subloop(
41             val_loader, model, criterion, device,
42             all_labels_eval, all_preds_eval
43         )
44
45     val_accuracy = accuracy_score(all_labels_eval, all_preds_eval)
46
47     # Print results
48     print("Unique labels in eval-data:",
49         ↪ sorted(set(all_labels_eval)))
49     print("Unique predictions:", sorted(set(all_preds_eval)))
50     print(colored(f"TRAIN - Epoch {epoch+1}, Loss:
51         ↪ {total_loss:.4f}, Accuracy: {train_accuracy:.2f}%",
52         ↪ "magenta"))
51     print(colored(f"EVAL - Epoch {epoch+1}, Accuracy:
52         ↪ {val_accuracy*100:.2f}%", "green"))
52     print(classification_report(all_labels_eval, all_preds_eval,
53         ↪ target_names=classes))
53
54     # Log MLflow
55     log_mlflow_train(train_accuracy, val_accuracy, total_loss,
56         ↪ val_loss, epoch)
56     log_classification_report_mlflow(classification_report(
57         all_labels_eval, all_preds_eval, target_names=classes,
58         ↪ output_dict=True), epoch)
58
59     # Save model
60     if val_accuracy > best_accuracy:
61         best_accuracy = val_accuracy
62         save_model(model, best_accuracy, save_path)
63         print(f"Model saved with accuracy: {best_accuracy:.2f}%")
64
65     # Confusion Matrix
66     confusion_matrix_plot(all_labels_eval, all_preds_eval, classes,
67         ↪ epoch)
67
68     scheduler.step()

```

```

69         print(colored(f"Learning rate:
    ↪     {scheduler.get_last_lr()[0]:.6f}", "yellow"))
70
71     print("\n")
72
73     #####
74
75     def train_subloop(loop, model, criterion, optimizer, device,
76                       predictions_per_epoch, labels_per_epoch, total_loss):
77         correct = 0
78         total = 0
79         for batch in loop:
80             spectrum, labels = batch
81             spectrum = spectrum.unsqueeze(1).to(device, non_blocking=True)
82             labels = labels.view(-1).to(device, non_blocking=True)
83
84             optimizer.zero_grad()
85             output = model(spectrum)
86
87             loss = criterion(output, labels)
88             loss.backward()
89             optimizer.step()
90
91             predictions_per_epoch.append(output.detach().cpu())
92             labels_per_epoch.append(labels.detach().cpu())
93
94             total_loss += loss.item()
95             _, predicted = output.max(1)
96             correct += (predicted == labels).sum().item()
97             total += labels.size(0)
98
99         train_accuracy = 100 * correct / total
100
101         return train_accuracy, total_loss
102
103     #####
104
105     def eval_subloop(val_loader, model, criterion, device,
106                     all_labels_eval, all_preds_eval):
107         val_loss = 0.0
108         for spectrum, labels in tqdm(val_loader, desc="Evaluation",
    ↪     leave=True, colour="blue"):
109             spectrum = spectrum.unsqueeze(1).to(device)
110             labels = labels.view(-1).to(device)
111             output = model(spectrum)
112             _, predicted = torch.max(output, 1)
113             loss = criterion(output, labels)
114
115             all_labels_eval.extend(labels.cpu().numpy())
116             all_preds_eval.extend(predicted.cpu().numpy())

```

```

117         val_loss += loss.item()
118
119     return all_labels_eval, all_preds_eval, val_loss
120
121     #####
122
123 def confusion_matrix_plot(all_labels_eval, all_preds_eval, classes,
124     ↪ epoch):
125     cm = confusion_matrix(all_labels_eval, all_preds_eval)
126     plt.figure(figsize=(6, 5))
127     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
128     ↪ xticklabels=classes, yticklabels=classes)
129     plt.xlabel("Predicted Labels")
130     plt.ylabel("Real Labels")
131     plt.title("Confusion Matrix")
132
133     output_dir = "plots/validation_plots"
134     os.makedirs(output_dir, exist_ok=True)
135     filename = f"confusion_matrix_EPOCH_{epoch+1}.png"
136     filepath = os.path.join(output_dir, filename)
137     plt.savefig(filepath)
138     plt.close()
139
140     mlflow.log_artifact(filepath, artifact_path="confusion_matrices")
141
142     #####
143
144 def save_model(model, best_accuracy, save_path):
145     torch.save({
146         'model_state_dict': model.state_dict(),
147         'best_accuracy': best_accuracy
148     }, save_path)
149
150     #####
151
152 class FocalLoss(nn.Module):
153     def __init__(self, alpha=None, gamma=2, reduction='mean'):
154         super(FocalLoss, self).__init__()
155         self.alpha = alpha # Forventer tensor med shape [n_classes]
156         self.gamma = gamma
157         self.reduction = reduction
158
159     def forward(self, inputs, targets):
160         ce_loss = F.cross_entropy(inputs, targets, reduction='none')
161         p_t = torch.exp(-ce_loss)
162
163         if self.alpha is not None:
164             # Hent riktig alpha for hver klasse i batchen
165             alpha = self.alpha.gather(0, targets) # Shape:
166             ↪ [batch_size]

```



```

164         focal_loss = alpha * (1 - p_t) ** self.gamma * ce_loss
165     else:
166         focal_loss = (1 - p_t) ** self.gamma * ce_loss
167
168     if self.reduction == 'mean':
169         return focal_loss.mean()
170     elif self.reduction == 'sum':
171         return focal_loss.sum()
172     return focal_loss
173
174     #####
175
176 def get_class_weights(label_paths, num_classes):
177     all_labels = []
178
179     for path in label_paths:
180         labels = load_label(path)
181         all_labels.extend(labels)
182
183     labels_tensor = torch.tensor(all_labels, dtype=torch.long)
184     class_counts = torch.bincount(labels_tensor,
185     ↪ minlength=num_classes).float()
186
187     # No zero division
188     epsilon = 1e-6
189     class_counts += epsilon
190
191     print("Using class weights.")
192
193     # Calculate weights
194     alpha = (1.0 / class_counts)
195     alpha = alpha / alpha.sum() # Normalization
196
197     print(colored(f"Successfully calculated class weights: {alpha}",
198     ↪ "green"))
199
200     return alpha
201
202     #####
203
204 def log_mlflow_pre_train(EPOCHS, BATCH_SIZE, LR, LABEL_SMOOTHING,
205     KERNEL_SIZE, STARTING_KERNELS, NUM_FEATURES,
206     ↪ NUM_CLASSES,
207     optimizer, scheduler, criterion, model, train_dataset,
208     ↪ eval_dataset):
209     mlflow.log_param("EPOCHS", EPOCHS)
210     mlflow.log_param("BATCH_SIZE", BATCH_SIZE)
211     mlflow.log_param("LR", LR)
212     mlflow.log_param("LABEL_SMOOTHING", LABEL_SMOOTHING)
213     mlflow.log_param("KERNEL_SIZE", KERNEL_SIZE)

```

```

210     mlflow.log_param("STARTING_KERNELS", STARTING_KERNELS)
211     mlflow.log_param("NUM_FEATURES", NUM_FEATURES)
212     mlflow.log_param("NUM_CLASSES", NUM_CLASSES)
213     mlflow.log_param("optimizer", optimizer.__class__.__name__)
214     mlflow.log_param("scheduler", scheduler.__class__.__name__)
215     mlflow.log_param("loss_function", criterion.__class__.__name__)
216     mlflow.log_param("model", model.__class__.__name__)
217     mlflow.log_param("train_dataset_size", len(train_dataset)/653016)
218     mlflow.log_param("eval_dataset_size", len(eval_dataset)/653016)
219
220     #####
221
222     def log_mlflow_train(train_accuracy, val_accuracy, train_loss,
223     ↪ val_loss, epoch):
224         mlflow.log_metric(f"train_accuracy", train_accuracy/100,
225         ↪ step=epoch)
226         mlflow.log_metric(f"val_accuracy", val_accuracy, step=epoch)
227         mlflow.log_metric(f"train_loss", train_loss, step=epoch)
228         mlflow.log_metric(f"val_loss", val_loss, step=epoch)
229
230     #####
231
232     def log_classification_report_mlflow(report_dict, epoch):
233         for class_name, metrics in report_dict.items():
234             if isinstance(metrics, dict):
235                 for metric_name, value in metrics.items():
236                     if "support" or "accuracy" not in metric_name:
237                         mlflow.log_metric(f"{class_name}_{metric_name}",
238                         ↪ value, step=epoch)
239             else:
240                 if "support" or "accuracy" not in class_name:
241                     mlflow.log_metric(f"{class_name}", metrics, step=epoch)

```

A.5 dataset.py

```

1     import sys
2     import os
3
4     sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
5     ↪ '..')))
6
7     from libraries import *
8     from functions.processing import *
9     from manage_data import read_csv_file
10
11     #####
12     #####
13     #####

```

```

13
14 class merged_hyperspectral_dataset(Dataset):
15     def __init__(self, list_of_images, list_of_labels=None,
16         ↪ normalizer=None):
17         all_images = []
18
19         if list_of_labels is not None:
20             all_labels = []
21
22         for image_path, label_path in tqdm(zip(list_of_images,
23         ↪ list_of_labels),
24             desc="Loading images and
25             ↪ labels",
26             total=len(list_of_images)):
27             image = load_image(image_path) # (pixels, bands)
28             all_images.append(image)
29
30             if list_of_labels is not None:
31                 label = load_label(label_path) # (pixels, )
32                 all_labels.append(label)
33
34         self.images = torch.from_numpy(
35         np.concatenate(all_images)).float().contiguous()
36
37         if list_of_labels is not None:
38             self.labels =
39             ↪ torch.from_numpy(np.concatenate(all_labels)).long()
40         else:
41             self.labels = None
42
43         if normalizer is not None:
44             self.images = normalizer.transform(self.images,
45             ↪ verify=True, verbose=False)
46
47     def __len__(self):
48         return self.images.shape[0]
49
50     def __getitem__(self, idx):
51         #print(self.images[idx].shape)
52         if self.labels is None:
53             return self.images[idx]
54         else:
55             return self.images[idx], self.labels[idx]
56
57     #####
58     """
59     bip_paths, dat_paths, png_paths = read_csv_file("csv/train_files.csv")
60
61     normalizer = normalization_manager()

```

```

58
59 raw_dataset = merged_hyperspectral_dataset(bip_paths, dat_paths)
60 normalizer.fit(raw_dataset.images)
61 dataset = merged_hyperspectral_dataset(
62     bip_paths, dat_paths, normalizer=normalizer
63 )
64
65 dataloader = DataLoader(dataset, batch_size=128, shuffle=True)
66
67 # Test en enkelt batch
68 for pixels, labels in dataloader:
69     print(colored(f"Pixels shape: {pixels.shape}", "light_blue")) #
70     ↪ Forventet: (128, num_bands)
71     print(colored(f"Labels shape: {labels.shape}", "light_green")) #
72     ↪ Forventet: (128,)
73     print(colored(f"Unique labels in batch: {torch.unique(labels)}",
74         ↪ "light_yellow"))
75     break
76 """

```

A.6 manage_data.py

```

1 import sys
2 import os
3
4 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
5     ↪ '..')))
6
7 from libraries import *
8
9 #####
10 #####
11 #####
12
13 """
14 The functions move_forward(), move_back() and list_contents() are used
15 ↪ for retrieving
16 .dat, .bip (/bip@) and .png files for use in CSV creation.
17 """
18
19 current_path = Path(".").resolve()
20
21 def move_forward(folder_name):
22     global current_path
23     new_path = current_path / folder_name
24     if new_path.exists() and new_path.is_dir():
25         current_path = new_path
26         return f"Moved to: {current_path}"

```

```

25     return "Folder does not exist."
26
27 def move_back():
28     global current_path
29     if current_path.parent.exists():
30         current_path = current_path.parent
31         return f"Moved back to: {current_path}"
32     return "Cannot move back further."
33
34 def list_contents():
35     return [item.name for item in current_path.iterdir()]
36
37 #####
38
39 def get_dat_png_bip():
40     """
41     Collects and returns paths of .dat, .png, and .bip files from
42     ↪ specified directories.
43     This function navigates through a directory structure to find and
44     ↪ collect paths of files
45     with specific extensions (.dat, .png, .bip) from labeled and raw
46     ↪ data directories.
47     Returns:
48         tuple: A tuple containing three lists:
49             - dat_paths (list): List of paths to .dat files.
50             - png_paths (list): List of paths to .png files.
51             - bip_paths (list): List of paths to .bip files.
52     """
53
54     dat_paths = []
55     dat_paths_checkup = []
56
57     png_paths = []
58     bip_paths = []
59
60     move_forward("corrected_labeled_data_CLOUD")
61     contents_labeled_data = list_contents()
62     for j in contents_labeled_data:
63         dat_paths.append((str(current_path / j))[35:])
64         dat_paths_checkup.append(j[:j.find("-l1a")])
65         #print(j[:j.find("-l1a")])
66
67     move_back()
68
69     #print(os.getcwd())
70
71     dat_paths_checkup = ([str(path) for path in dat_paths_checkup])
72     dat_paths_checkup = sorted(dat_paths_checkup)
73     #print(len(dat_paths_checkup))

```

```

72     #print(dat_paths_checkup)
73
74     move_forward("raw_data")
75     L1_contents = list_contents()
76     #print(L1_contents)
77
78     print(dat_paths_checkup)
79
80     for folder in L1_contents:
81         move_forward(folder)
82         L2_contents = list_contents()
83         print(colored(L2_contents, "blue"))
84         for folder_gr_2 in L2_contents:
85             if folder_gr_2 in dat_paths_checkup:
86                 move_forward(folder_gr_2)
87                 L3_contents = list_contents()
88                 print(colored(L3_contents, "light_blue"))
89                 for i in L3_contents:
90                     if i.endswith("Z.png"):
91                         png_paths.append((str(current_path / i))[35:])
92                     elif i.endswith(".bip"):
93                         bip_paths.append((str(current_path / i))[35:])
94                     elif i.endswith(".bip@"):
95                         bip_paths.append((str(current_path / i))[35:])
96
97                 move_back()
98             move_back()
99         move_back()
100
101     print("-"*100)
102     print(sorted(png_paths))
103
104     print(len(dat_paths), len(png_paths), len(bip_paths))
105
106     return dat_paths, png_paths, bip_paths
107
108     #####
109
110 def create_csv_file():
111     """
112     Creates CSV files that maps .dat files
113     to their corresponding .bip (/bip@) and .png files.
114
115     - Calls get_dat_png_bip() to retrieve paths.
116     - Writes the file paths into 'train_files.csv' and
117     'evaluate_files.csv' with headers: ["dat_files", "bip_files",
118     ↪ "png_files"].
119     """

```

```

120     dat_files_paths, png_files_paths, bip_files_paths =
121         ↪ get_dat_png_bip()
122
123     dat_files_paths = sorted(dat_files_paths)
124     png_files_paths = sorted(png_files_paths)
125     bip_files_paths = sorted(bip_files_paths)
126
127     if len(dat_files_paths) == 1:
128         with open('csv/train_files.csv', mode='w') as train_file:
129             writer = csv.writer(train_file)
130             writer.writerow(["dat_files", "bip_files", "png_files"])
131             writer.writerow([dat_files_paths[0], bip_files_paths[0],
132                             ↪ png_files_paths[0]])
133     elif len(dat_files_paths) > 1:
134         split_index = int(len(dat_files_paths)*0.9)
135         train_dat_files = dat_files_paths[:split_index]
136         eval_dat_files = dat_files_paths[split_index:]
137
138         train_bip_files = bip_files_paths[:split_index]
139         eval_bip_files = bip_files_paths[split_index:]
140
141         train_png_files = png_files_paths[:split_index]
142         eval_png_files = png_files_paths[split_index:]
143
144         with open('csv/train_files.csv', mode='w') as train_file:
145             writer = csv.writer(train_file)
146             writer.writerow(["dat_files", "bip_files", "png_files"])
147             for i in range(len(train_dat_files)):
148                 writer.writerow([train_dat_files[i],
149                                 ↪ train_bip_files[i], train_png_files[i]])
150
151         with open('csv/evaluate_files.csv', mode='w') as eval_file:
152             writer = csv.writer(eval_file)
153             writer.writerow(["dat_files", "bip_files", "png_files"])
154             for i in range(len(eval_dat_files)):
155                 writer.writerow([eval_dat_files[i], eval_bip_files[i],
156                                 ↪ eval_png_files[i]])
157     else:
158         with open('csv/train_files.csv', mode='w') as train_file:
159             writer = csv.writer(train_file)
160             writer.writerow(["dat_files", "bip_files", "png_files"])
161             for i in range(len(dat_files_paths)):
162                 writer.writerow([dat_files_paths[i],
163                                 ↪ bip_files_paths[i], eval_png_files[i]])
164
165     #####
166
167 def read_csv_file(csv_file_with_path):
168     """
169     Reads a CSV file containing .dat and .bip@ file paths.

```

```

165
166     Parameters:
167     - csv_file_with_path (str): Path to the CSV file.
168
169     Returns:
170     - bip_files (list): List of .bip (/..bip@) file paths.
171     - dat_files (list): List of .dat file paths.
172     - png_files (list): List of .png file paths.
173     """
174
175     dat_files = []
176     bip_files = []
177     png_files = []
178
179     with open(f'{csv_file_with_path}', mode='r') as file:
180         csvFile = csv.reader(file)
181
182         i = 0
183         for lines in csvFile:
184             if i != 0:
185                 dat_files.append(lines[0])
186                 bip_files.append(lines[1])
187                 png_files.append(lines[2])
188                 i += 1
189
190     return bip_files, dat_files, png_files
191
192     #####
193
194     #create_csv_file()

```

A.7 train.py

```

1  import sys
2  import os
3
4  sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↵      '..')))
5
6  from libraries import *
7  from manage_data import read_csv_file
8  from dataset import merged_hyperspectral_dataset
9  from functions.processing import normalization_manager
10 from functions.train_functions import train_loop
11 from functions.train_functions import get_class_weights
12 from functions.train_functions import FocalLoss
13 from functions.train_functions import log_mlflow_pre_train
14 from models.cnn_1d import JustoLiuNet1D_torch

```



```

15
16 #####
17 #####
18 #####
19
20 # MLflow
21 mlflow.set_experiment("CNN_hyperspectral_v1")
22
23 # Device
24 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
25 print(f"Device: {device}")
26
27 # Hyperparameters
28 EPOCHS = 10
29 BATCH_SIZE = 128
30 LR = 0.001
31 LABEL_SMOOTHING = 0.1
32 KERNEL_SIZE = 6
33 STARTING_KERNELS = 6
34 NUM_FEATURES = 114
35 NUM_CLASSES = 3
36
37 with mlflow.start_run():
38
39     # Data
40     train_bip_paths, train_labels_paths, _ =
41         ↪ read_csv_file("csv/train_files.csv")
42     eval_bip_paths, eval_labels_paths, _ =
43         ↪ read_csv_file("csv/evaluate_files.csv")
44
45     # Normalizer
46     normalizer = normalization_manager()
47     raw_train = merged_hyperspectral_dataset(train_bip_paths,
48         ↪ train_labels_paths)
49     normalizer.fit(raw_train.images)
50     train_dataset = merged_hyperspectral_dataset(train_bip_paths,
51         ↪ train_labels_paths, normalizer)
52     eval_dataset = merged_hyperspectral_dataset(eval_bip_paths,
53         ↪ eval_labels_paths, normalizer)
54
55     # Dataloader
56     train_loader = DataLoader(train_dataset,
57         batch_size=BATCH_SIZE,
58         shuffle=True,
59         num_workers=8,
60         pin_memory=True)
61     eval_loader = DataLoader(eval_dataset,
62         batch_size=BATCH_SIZE,
63         shuffle=False,
64         num_workers=8,

```

```

60         pin_memory=True)
61
62     # Model
63     model = JustoLiuNet1D_torch(num_features=NUM_FEATURES,
64         ↪ num_classes=NUM_CLASSES,
65         ↪ kernel_size=KERNEL_SIZE,
66         ↪ starting_kernels=STARTING_KERNELS).to(device)
67     total_params = sum(p.numel() for p in model.parameters())
68     print(colored(f"Total parameters in {model.__class__.__name__}:
69         ↪ {total_params}", "magenta"))
70
71     optimizer = torch.optim.AdamW(model.parameters(), lr=LR,
72         ↪ weight_decay=1e-4)
73     scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5,
74         ↪ gamma=0.5)
75     alpha = get_class_weights(train_labels_paths, NUM_CLASSES)
76     #criterion = nn.CrossEntropyLoss(weight=alpha.to(device),
77         ↪ label_smoothing=LABEL_SMOOTHING).to(device)
78     criterion =
79         ↪ nn.CrossEntropyLoss(label_smoothing=LABEL_SMOOTHING).to(device)
80     #criterion = FocalLoss(alpha=alpha.to(device), gamma=3,
81         ↪ reduction='sum').to(device)
82
83     # Logging
84     log_mlflow_pre_train(EPOCHS, BATCH_SIZE, LR, LABEL_SMOOTHING,
85         ↪ KERNEL_SIZE, STARTING_KERNELS, NUM_FEATURES,
86         ↪ NUM_CLASSES, optimizer, scheduler, criterion,
87         ↪ model, train_dataset, eval_dataset)
88
89     # Training
90     print("Starting training...")
91     train_loop(model, train_loader, eval_loader, criterion,
92         ↪ optimizer, scheduler, device, num_epochs=EPOCHS)
93     print("Training finished.")

```

B Complete Class Diagram

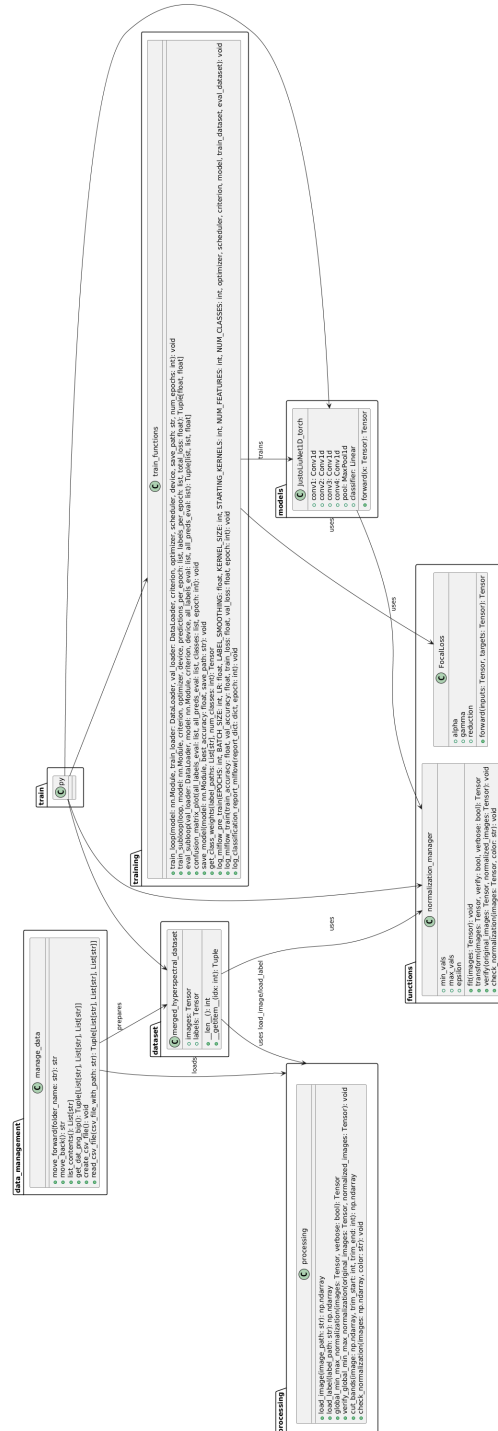


Figure 22: Complete class diagram of code.

C Images Used in Testing

Note here that the "Set" column in the tables below have either the variables: "T", "E" or "TE". "T" indicates that the image was in the training set. "E" indicates the image was in the evaluation set. Lastly, "TE" indicates the image was in both sets.

C.1 T01

Table 9: Images used in test T01.

Image Name	Date	Set
lacrau	2024-12-26	TE
bluenile	2025-01-25	TE
dardanelles	2025-03-08	TE
aquawatchlakehume	2025-03-08	TE
aeronetgalata	2025-01-02	TE
zeebrugge	2025-03-08	TE

C.2 T02

Table 10: Images used in test T02.

Image Name	Date	Set
lacrau	2024-12-26	TE
image61N6E	2025-03-13	TE
menindee	2025-01-02	TE
frohavet	2025-02-25	TE
gulfofcalifornia	2025-01-14	TE
gobabeb	2025-02-02	TE
aquawatchspencer	2025-01-03	TE
straitofgeorgia	2025-01-24	TE

C.3 T03

As stated in Table 2, there's 25 copies of this image during training, and the same image during evaluation.

Table 11: Images used in test T03.

Image Name	Date	Set
bluenile	2025-01-25	TE

C.4 T04

Table 12: Images used in test T04.

Image Name	Date	Set
lacrau	2024-12-26	T
bluenile	2025-01-25	T
dardanelles	2025-03-08	T
aquawatchlakehume	2025-03-08	T
zeebrugge	2025-03-08	T
blanca	2025-02-04	T
aeronetgloria	2025-01-09	T
aeronetgalata	2025-01-02	E

C.5 T05

Table 13: Images used in test T05.

Image Name	Date	Set
lacrau	2024-12-26	T
bluenile	2025-01-25	T
dardanelles	2025-03-08	T
menindee	2025-01-02	T
aeronetgalata	2025-01-02	T
aeronetgloria	2025-01-09	T
aquawatchgrippsland	2025-03-09	T
aquawatchlakehume	2025-03-08	T
image61N6E	2025-03-13	T
falklandsatlantic	2025-03-03	E
frohavet	2025-02-25	E
gobabeb	2025-02-02	E

C.6 T06

Table 14: Images used in test T06.

Image Name	Date	Set
aeronetgalata	2025-01-02	T
aeronetgloria	2025-01-09	T
aquawatchgrippsland	2025-03-09	T
aquawatchlakehume	2025-03-08	T
aquawatchmoreton	2025-01-22	T
aquawatchspencer	2025-01-03	T
ariake	2025-02-11	T
blanca	2025-02-04	T
bluenile	2025-01-25	T
dardanelles	2025-03-08	T
erie	2025-03-13	T
falklandsatlantic	2025-03-03	T

Image Name	Date	Set
frohavet	2025-02-25	T
gobabeb	2025-02-02	T
goddard	2025-01-09	T
grizzlybay	2025-01-22	T
gulfofcalifornia	2025-01-14	T
hypernetEstuary	2024-12-28	T
image61N6E	2025-03-13	T
image65N10E	2025-03-12	T
kemigawa	2025-01-22	T
lacrau	2024-12-26	T
longisland	2025-01-22	T
menindee	2025-01-02	T
moby	2025-01-08	E
straitofgeorgia	2025-01-24	E
zeebrugge	2025-03-08	E

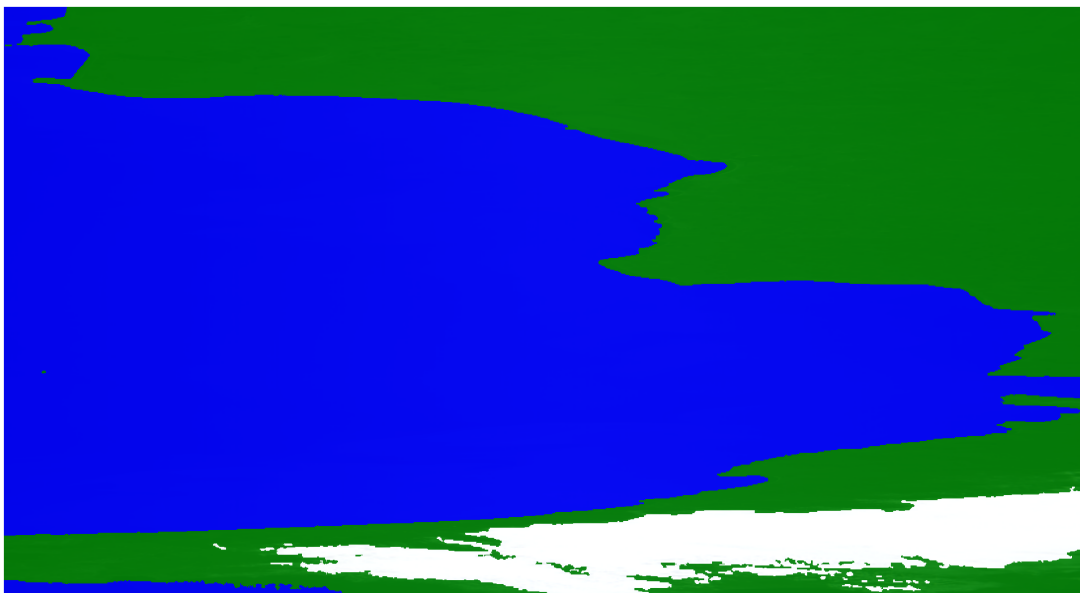
D Used Images and Their Labels

Note here that blue coloring is for sea, green is for land and white is for cloud.

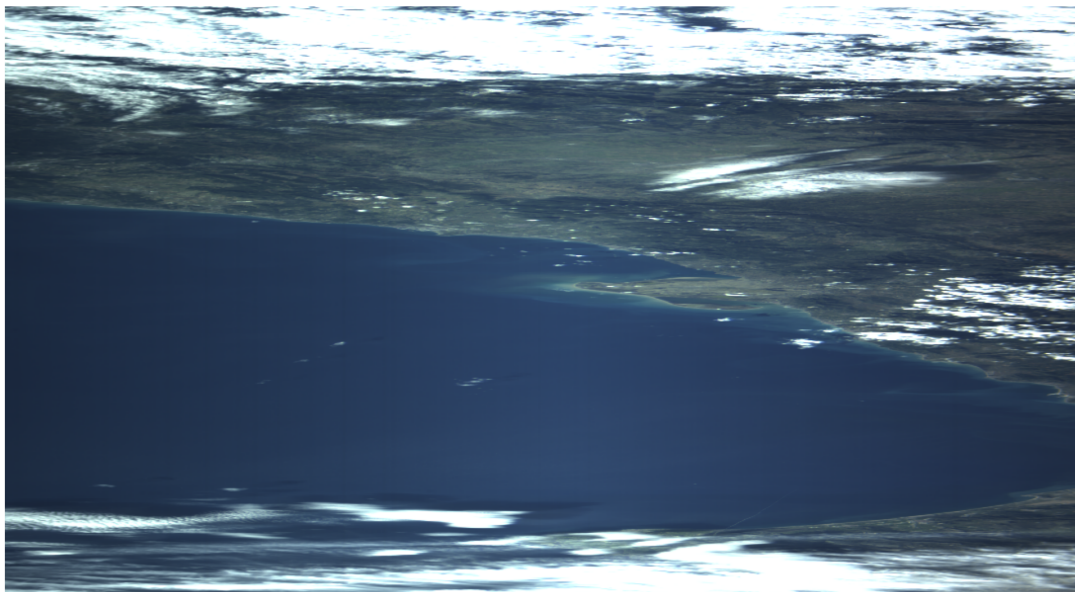
Original Image: aeronetgalata_2025-01-02T08-52-34Z.png



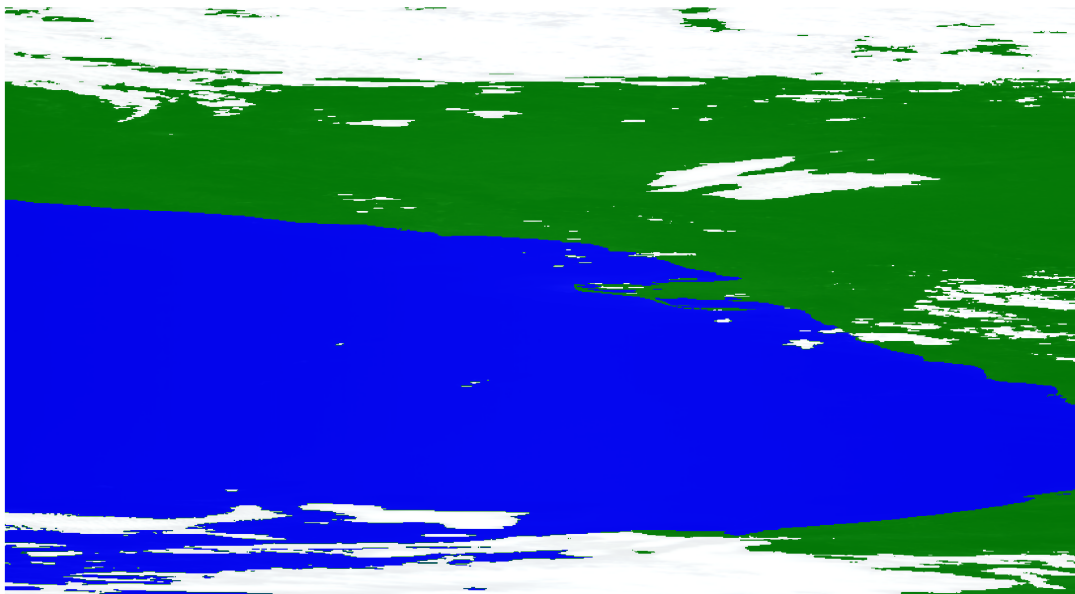
Labeled Image: aeronetgalata_2025-01-02T08-52-34Z.png



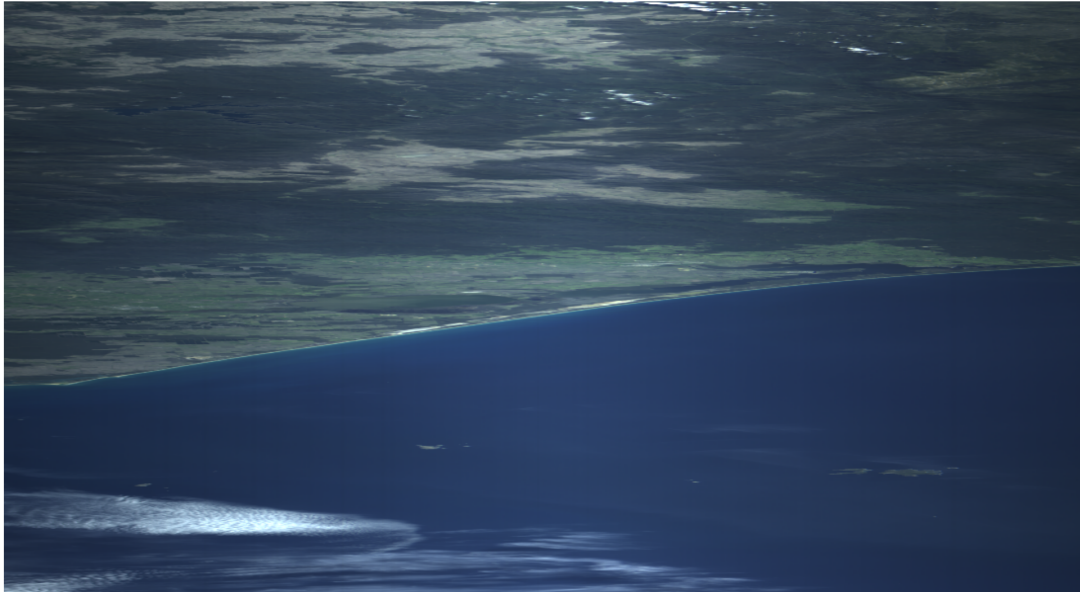
Original Image: aeronetgloria_2025-01-09T11-17-15Z.png



Labeled Image: aeronetgloria_2025-01-09T11-17-15Z.png



Original Image: aquawatchgrippsland_2025-03-09T00-16-17Z.png



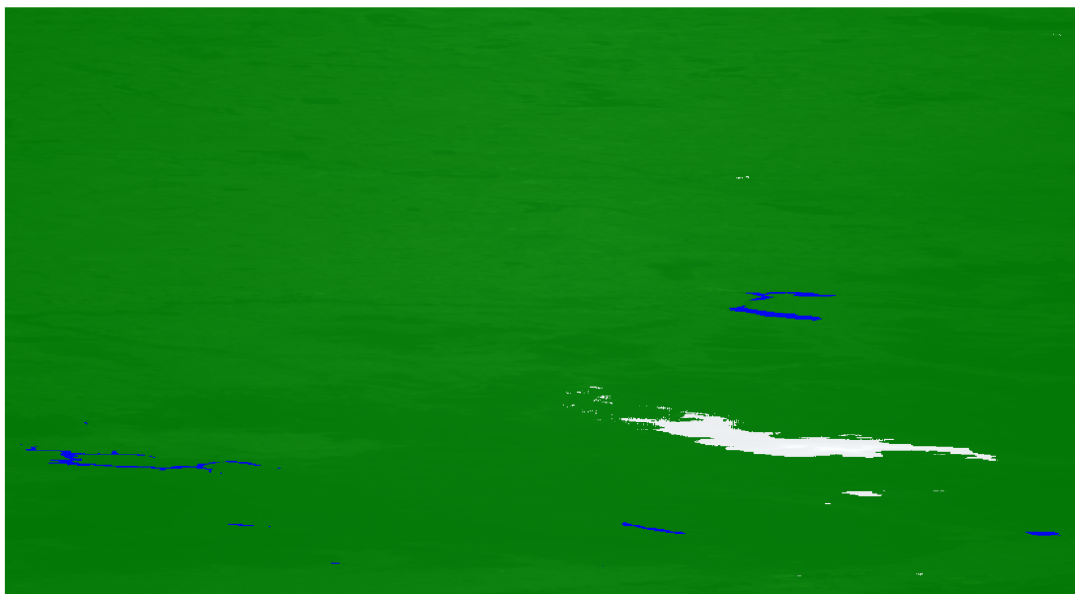
Labeled Image: aquawatchgrippsland_2025-03-09T00-16-17Z.png



Original Image: aquawatchlakehume_2025-03-08T00-09-52Z.png



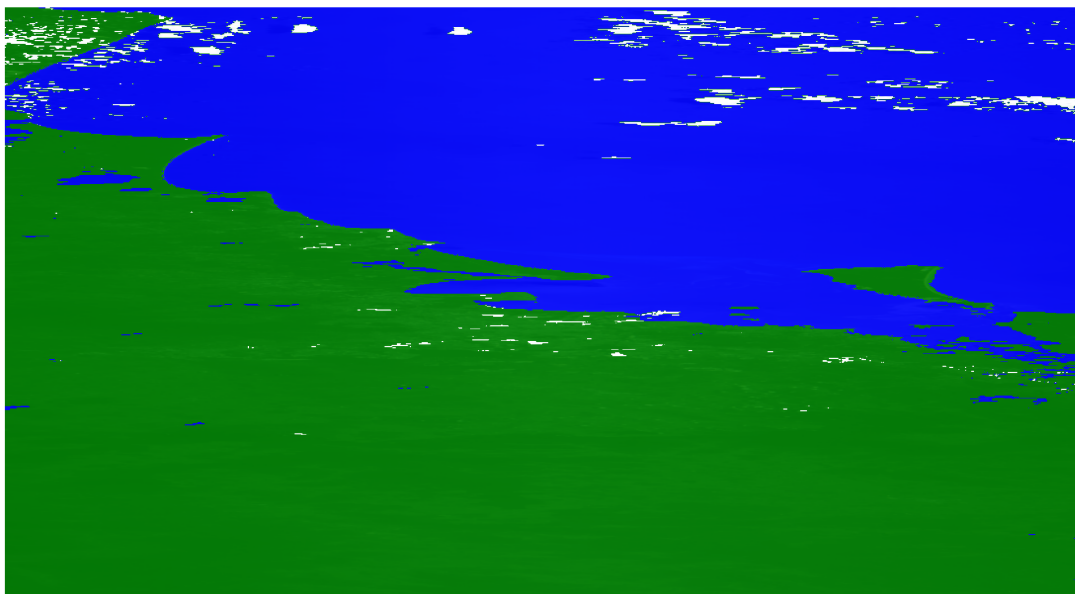
Labeled Image: aquawatchlakehume_2025-03-08T00-09-52Z.png



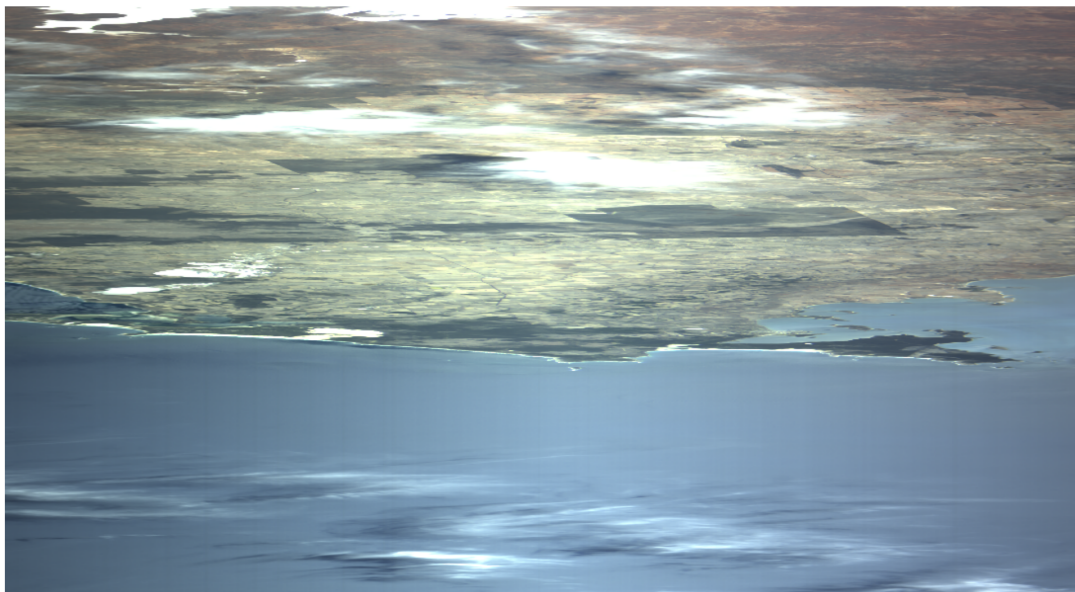
Original Image: aquawatchmoreton_2025-01-22T00-11-33Z.png



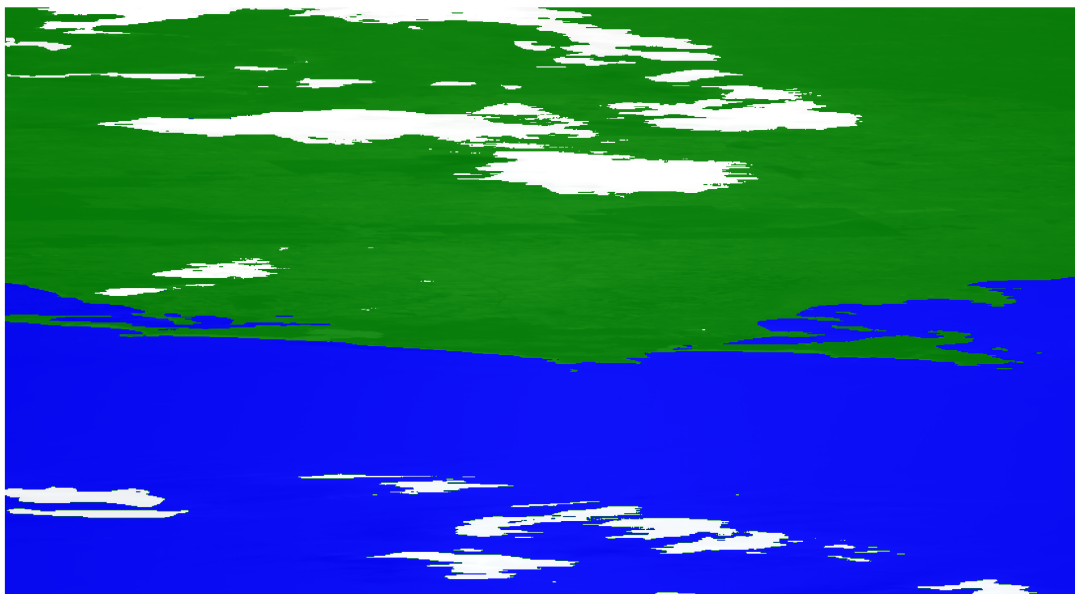
Labeled Image: aquawatchmoreton_2025-01-22T00-11-33Z.png



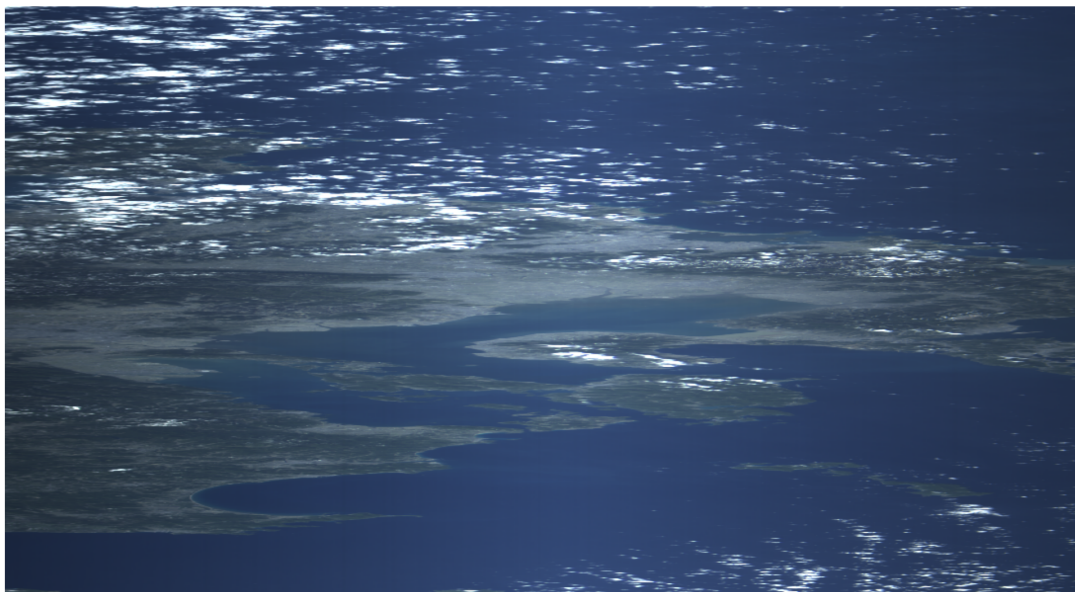
Original Image: aquawatchspencer_2025-01-03T01-17-50Z.png



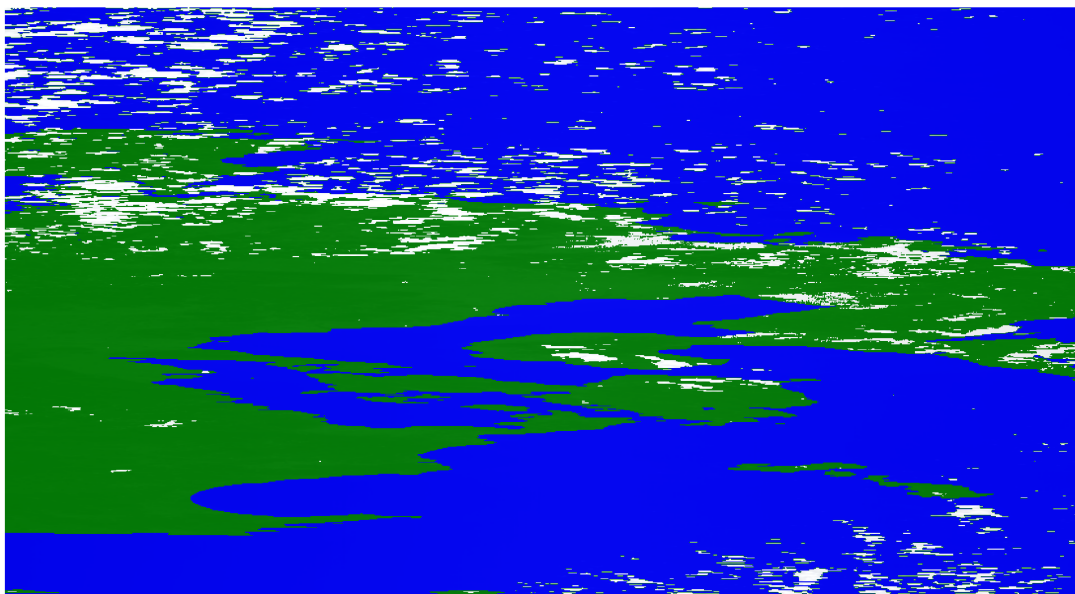
Labeled Image: aquawatchspencer_2025-01-03T01-17-50Z.png



Original Image: ariake_2025-02-11T02-05-25Z.png



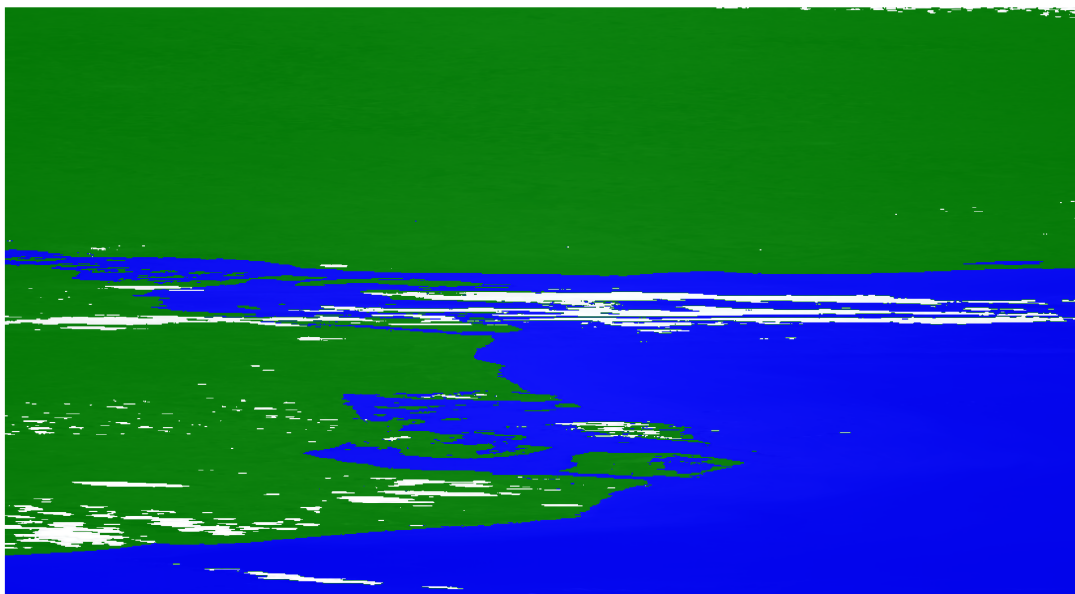
Labeled Image: ariake_2025-02-11T02-05-25Z.png



Original Image: blanca_2025-02-04T14-31-12Z.png



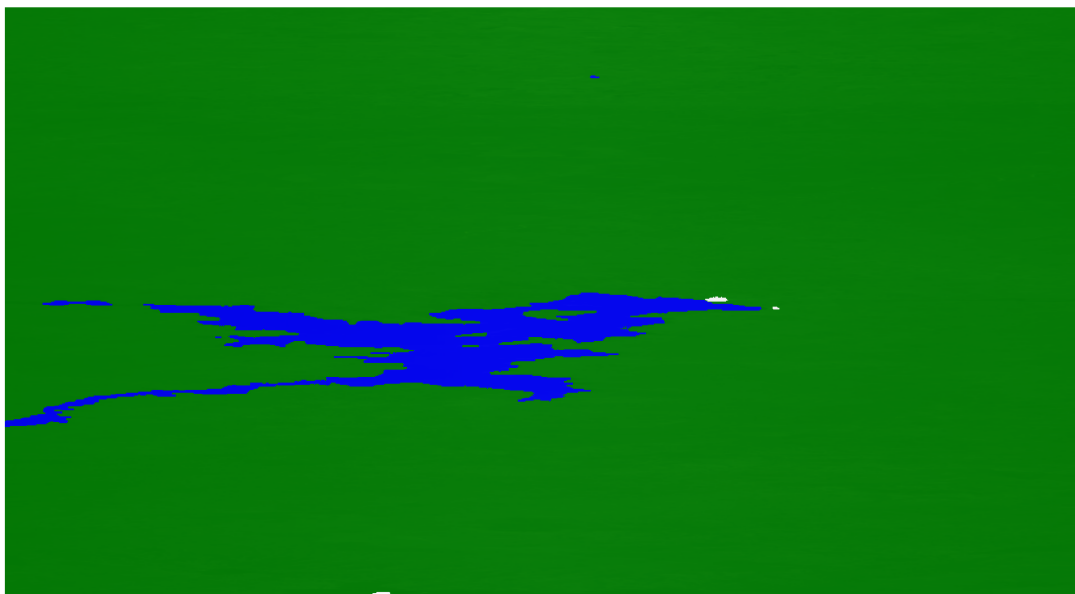
Labeled Image: blanca_2025-02-04T14-31-12Z.png



Original Image: bluenile_2025-01-25T08-23-16Z.png



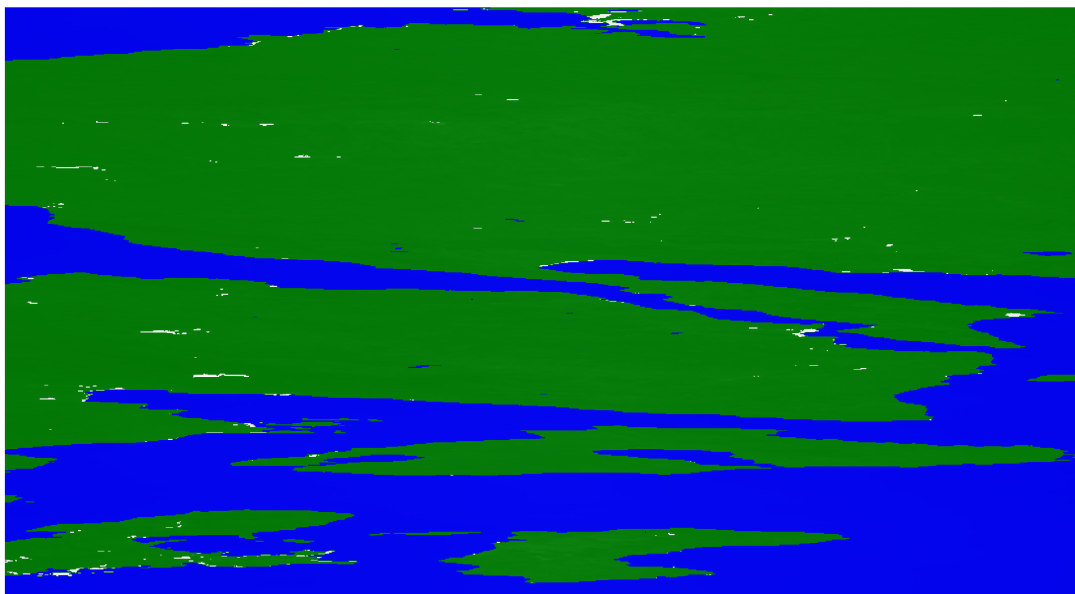
Labeled Image: bluenile_2025-01-25T08-23-16Z.png



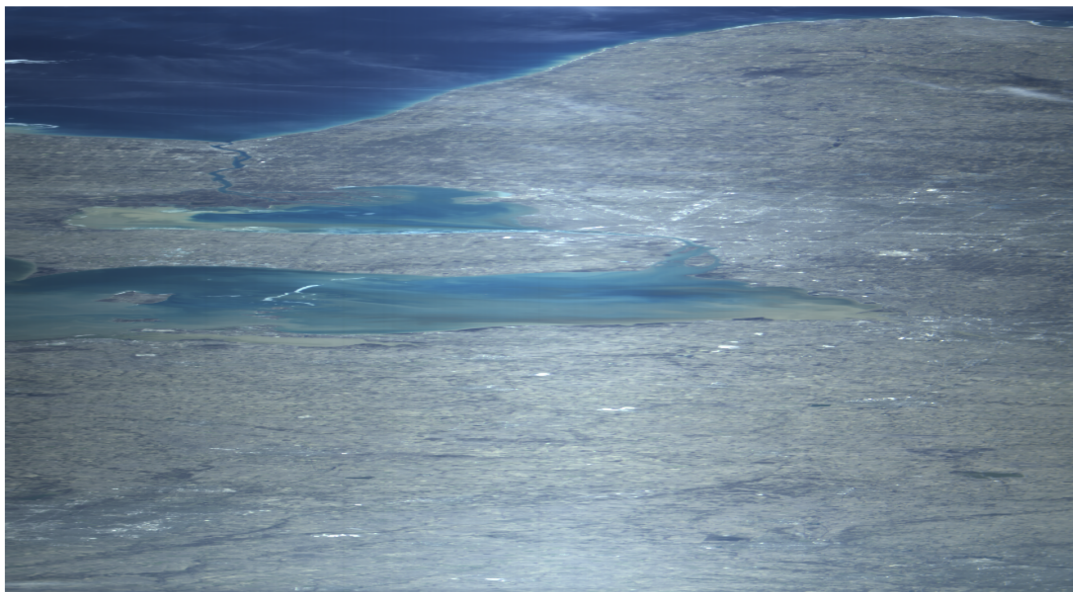
Original Image: dardanelles_2025-03-08T09-27-22Z.png



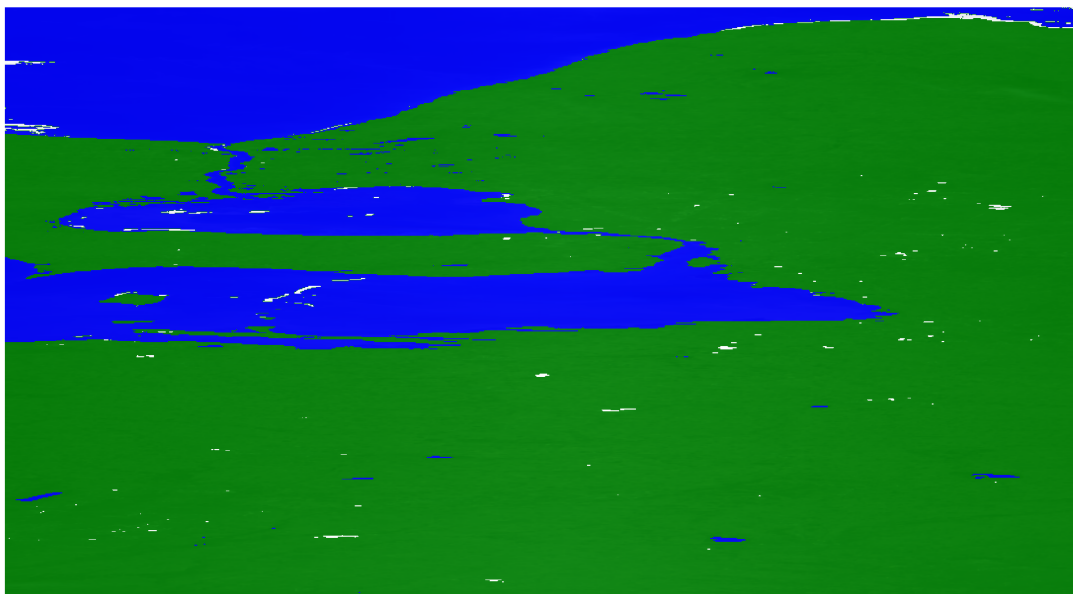
Labeled Image: dardanelles_2025-03-08T09-27-22Z.png



Original Image: erie_2025-03-13T16-22-46Z.png



Labeled Image: erie_2025-03-13T16-22-46Z.png



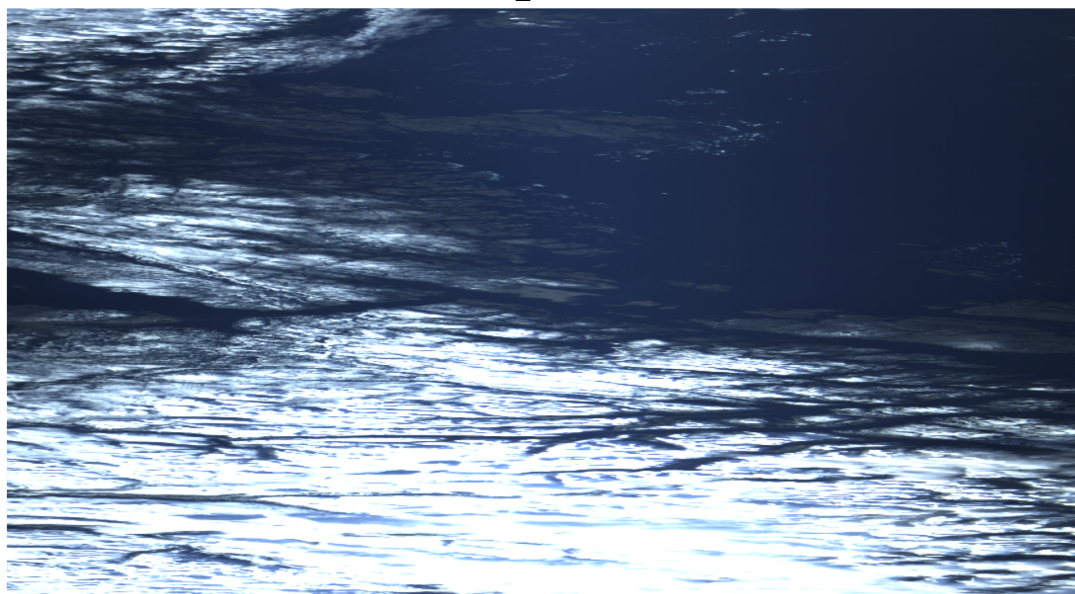
Original Image: falklandsatlantic_2025-03-03T14-11-51Z.png



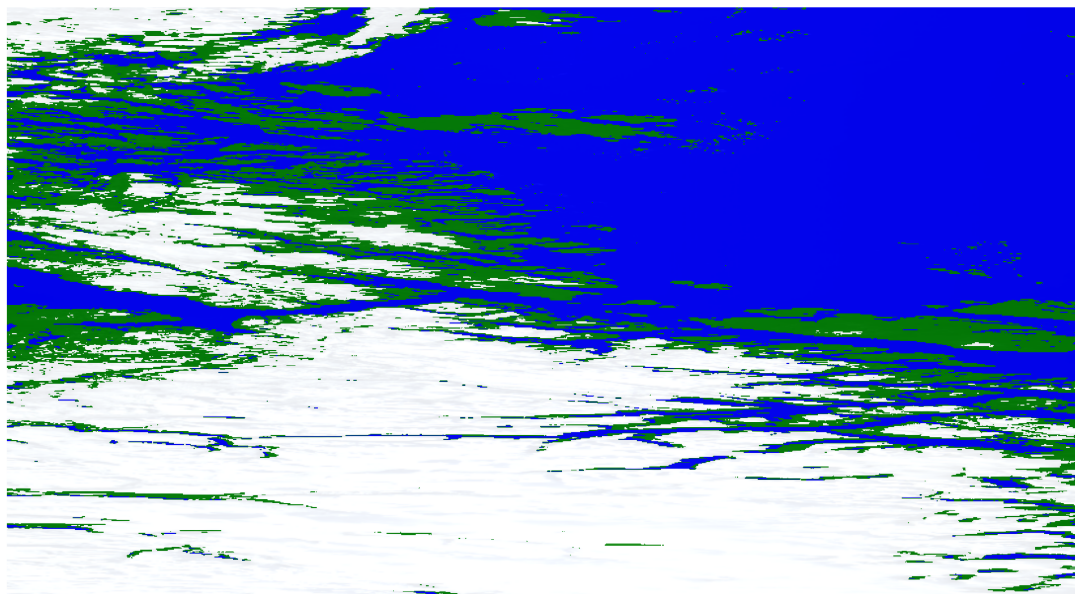
Labeled Image: falklandsatlantic_2025-03-03T14-11-51Z.png



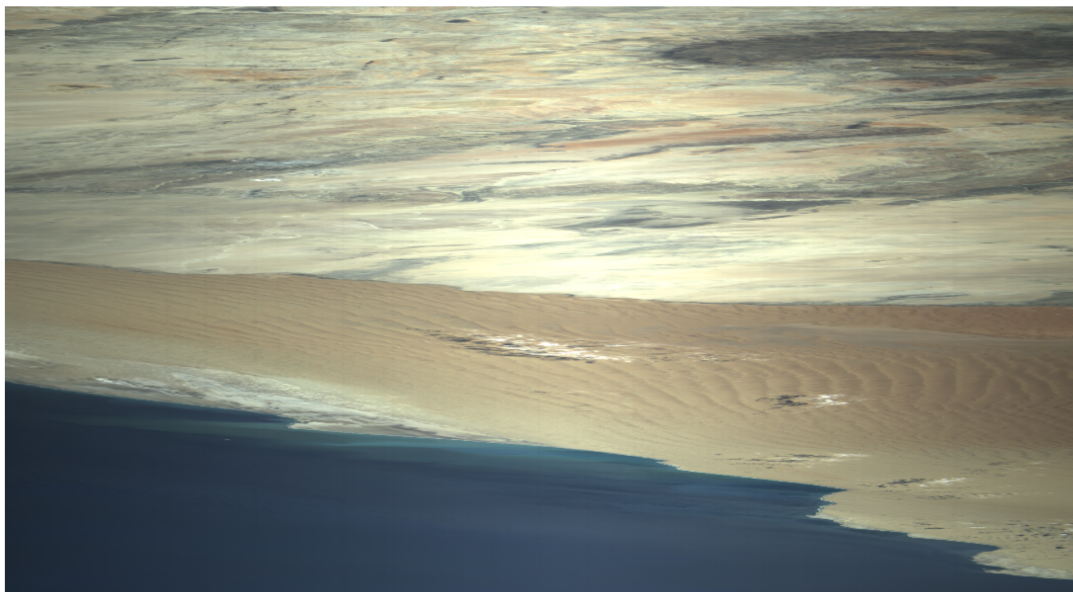
Original Image: frohavet_2025-02-25T11-26-39Z.png



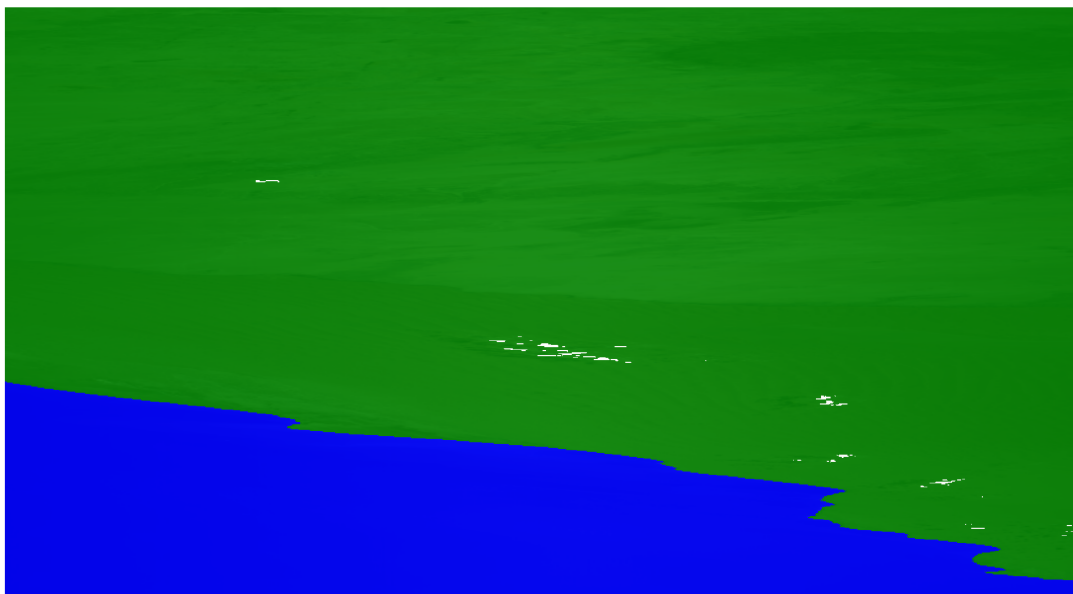
Labeled Image: frohavet_2025-02-25T11-26-39Z.png



Original Image: gobabeb_2025-02-02T09-24-52Z.png



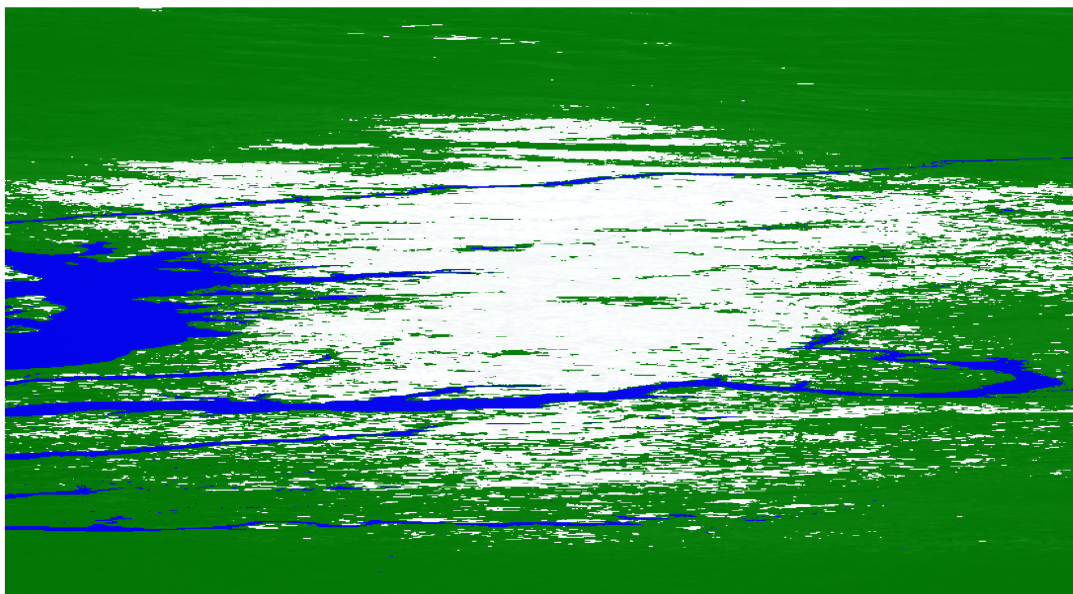
Labeled Image: gobabeb_2025-02-02T09-24-52Z.png



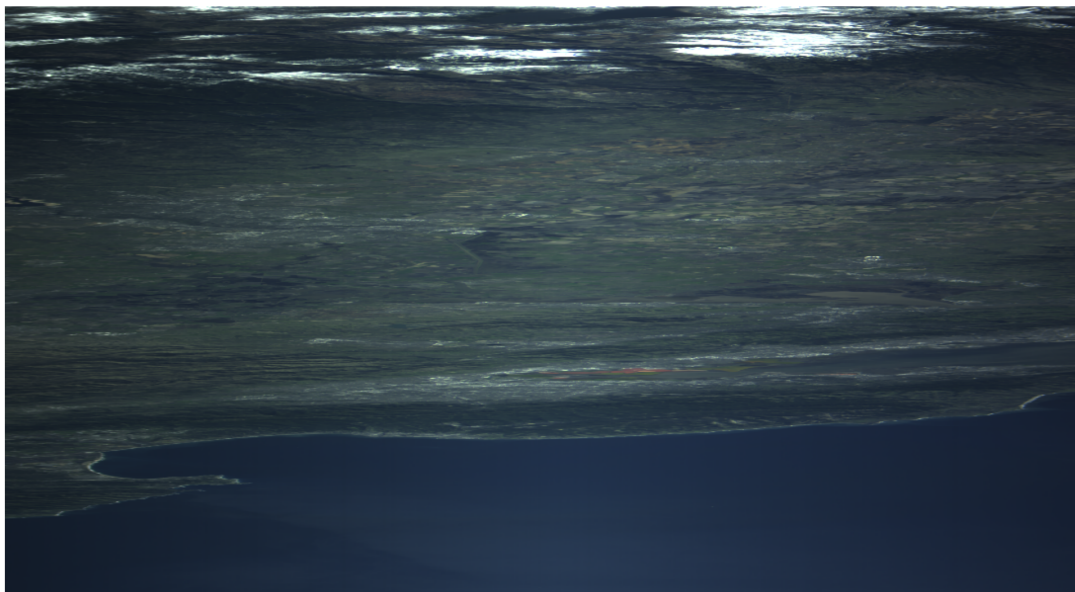
Original Image: goddard_2025-01-09T16-07-16Z.png



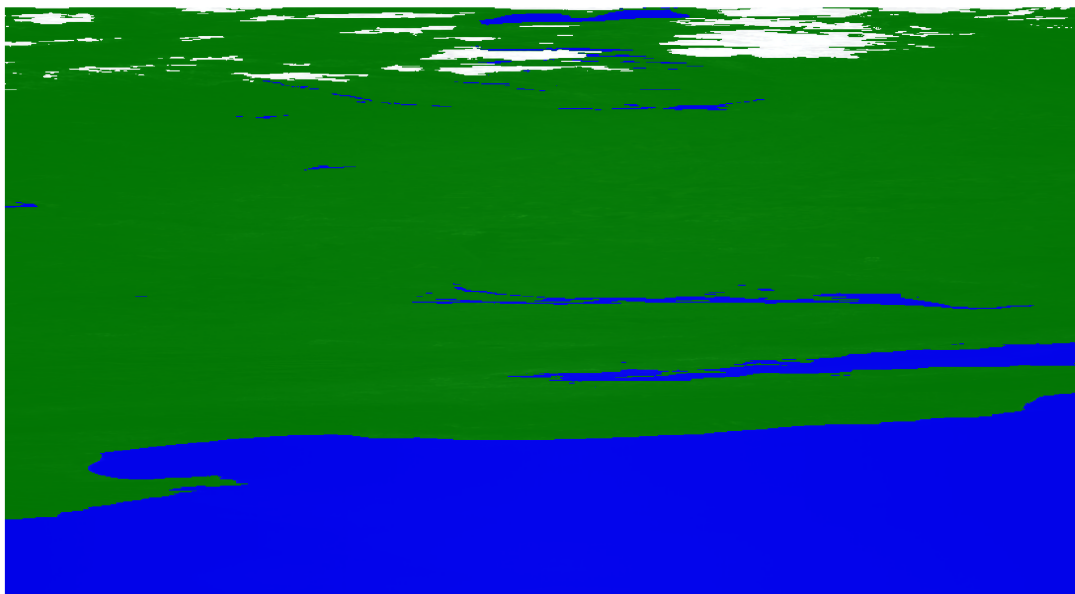
Labeled Image: goddard_2025-01-09T16-07-16Z.png



Original Image: grizzlybay_2025-01-22T19-11-18Z.png



Labeled Image: grizzlybay_2025-01-22T19-11-18Z.png



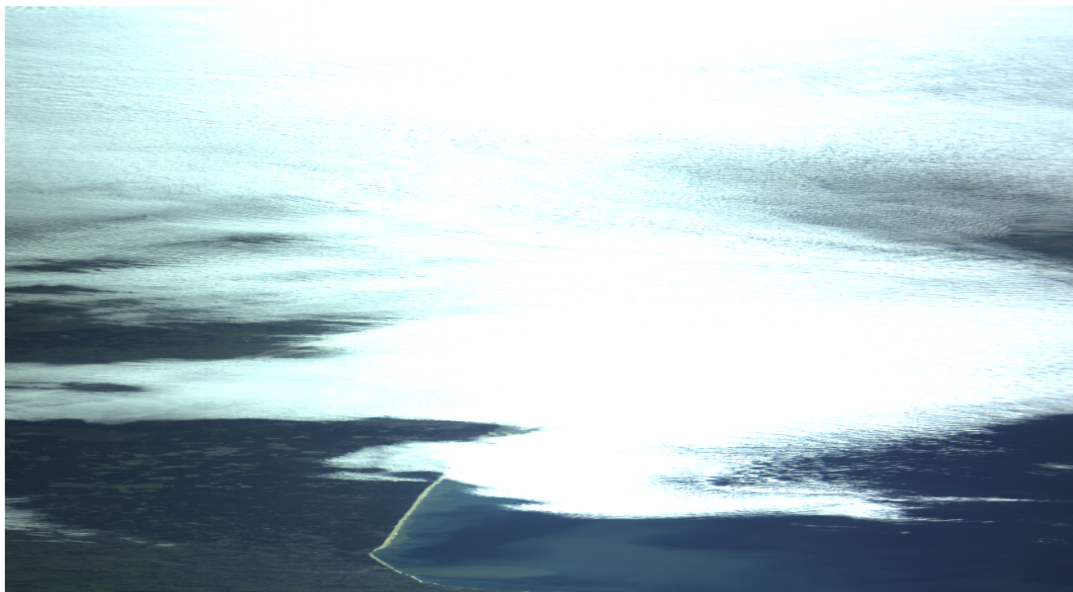
Original Image: gulfofcalifornia_2025-01-14T18-19-49Z.png



Labeled Image: gulfofcalifornia_2025-01-14T18-19-49Z.png



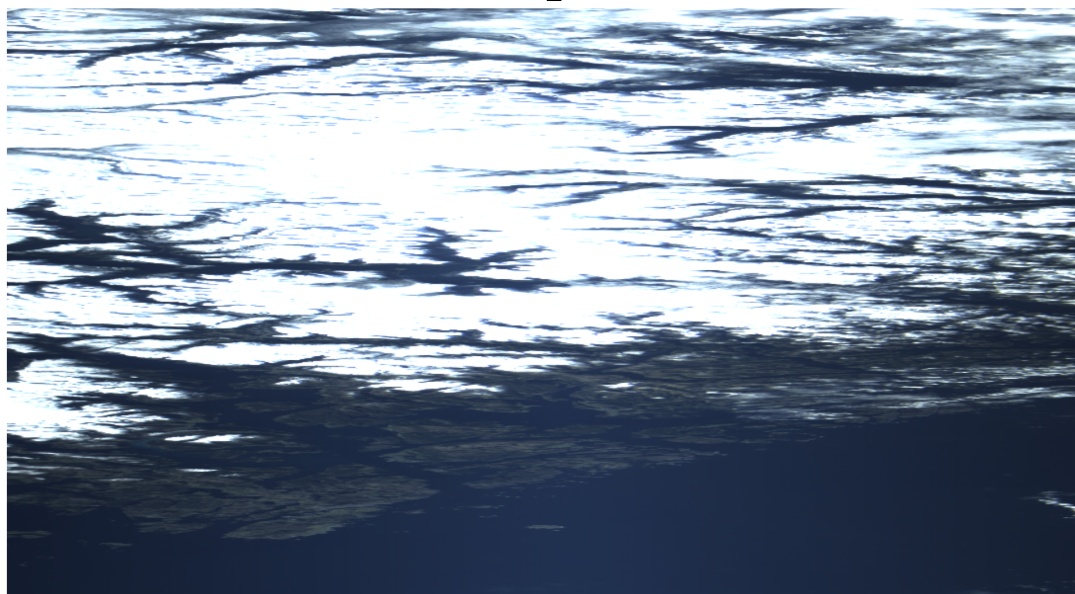
Original Image: hypernetEstuary_2024-12-28T11-29-35Z.png



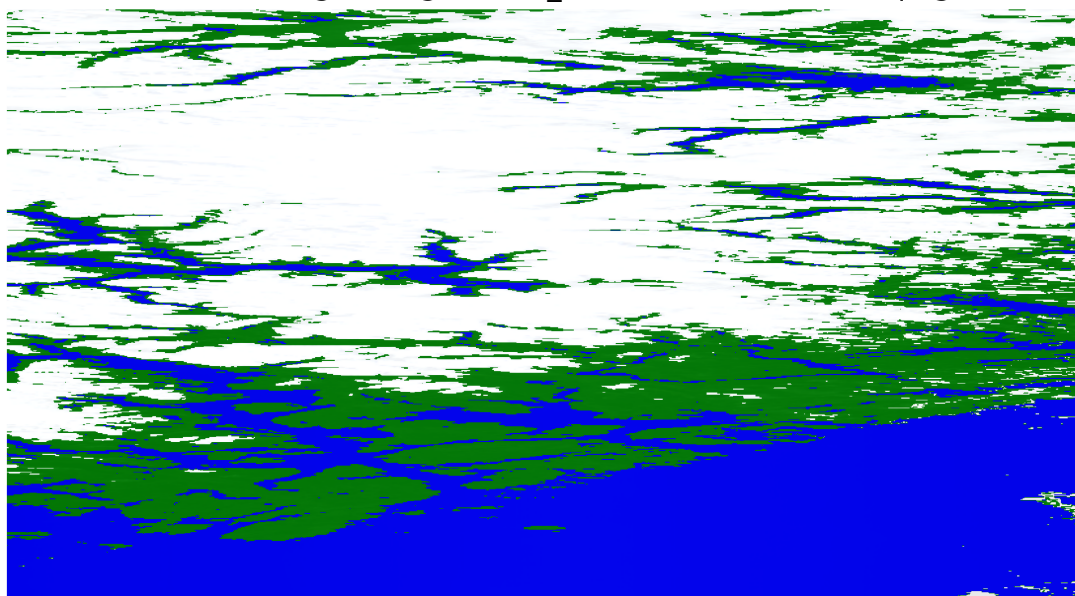
Labeled Image: hypernetEstuary_2024-12-28T11-29-35Z.png



Original Image: image61N6E_2025-03-13T11-27-56Z.png



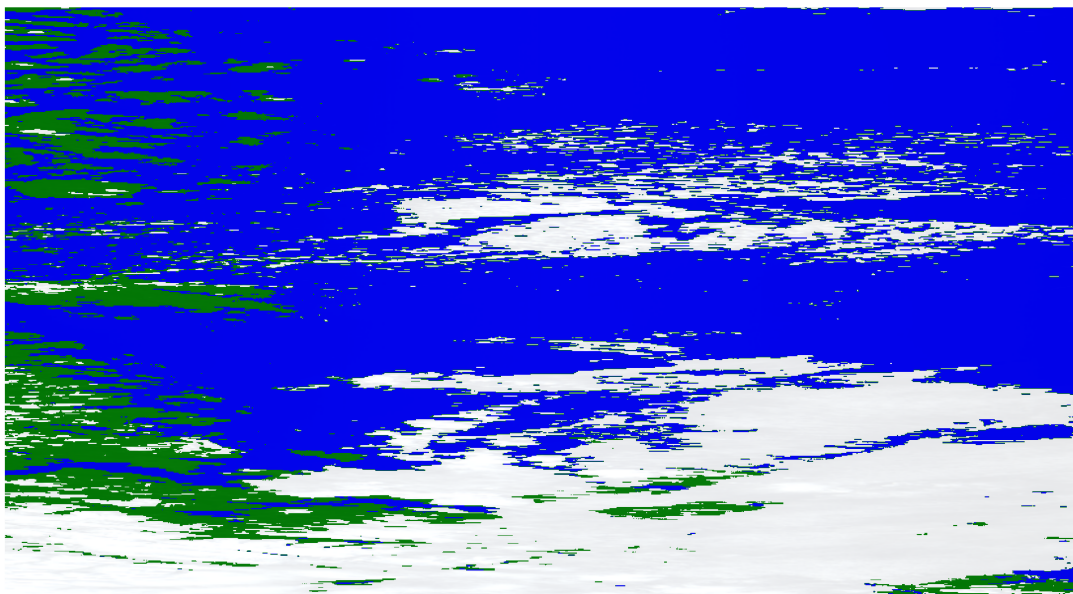
Labeled Image: image61N6E_2025-03-13T11-27-56Z.png



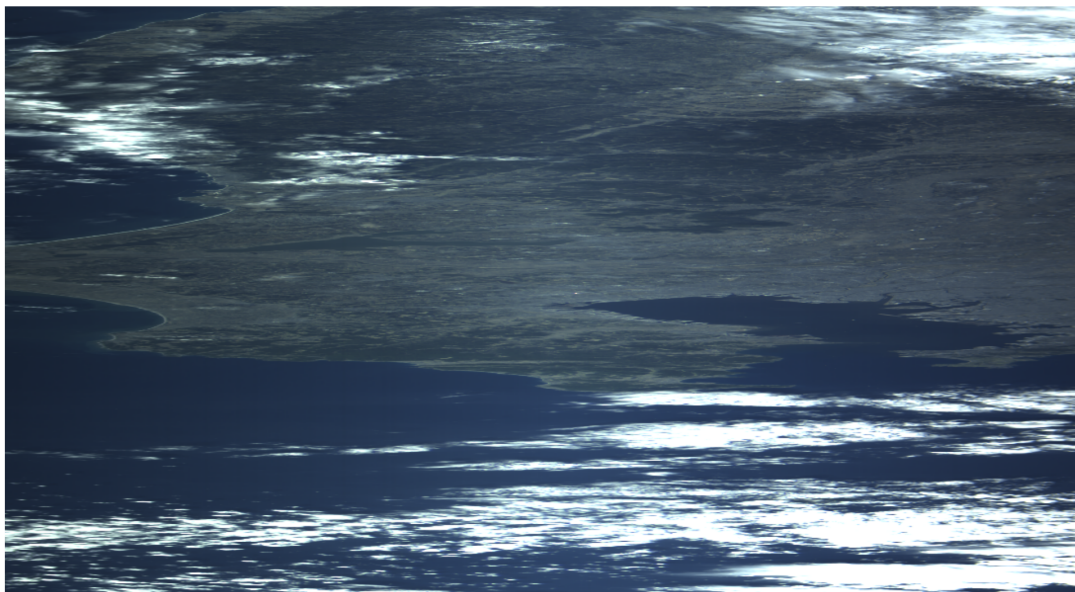
Original Image: image65N10E_2025-03-12T11-20-52Z.png



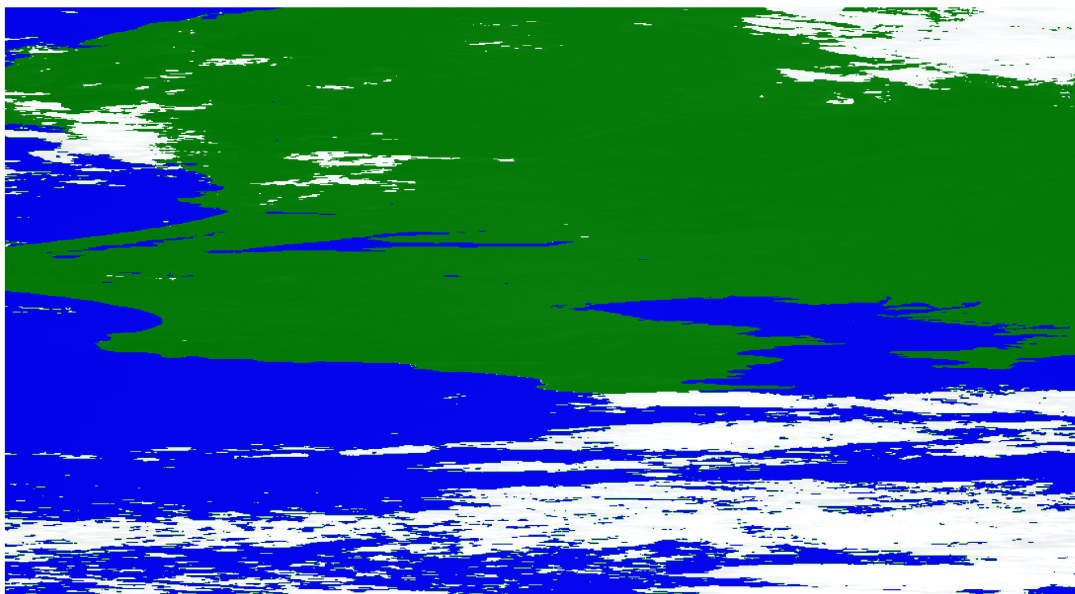
Labeled Image: image65N10E_2025-03-12T11-20-52Z.png



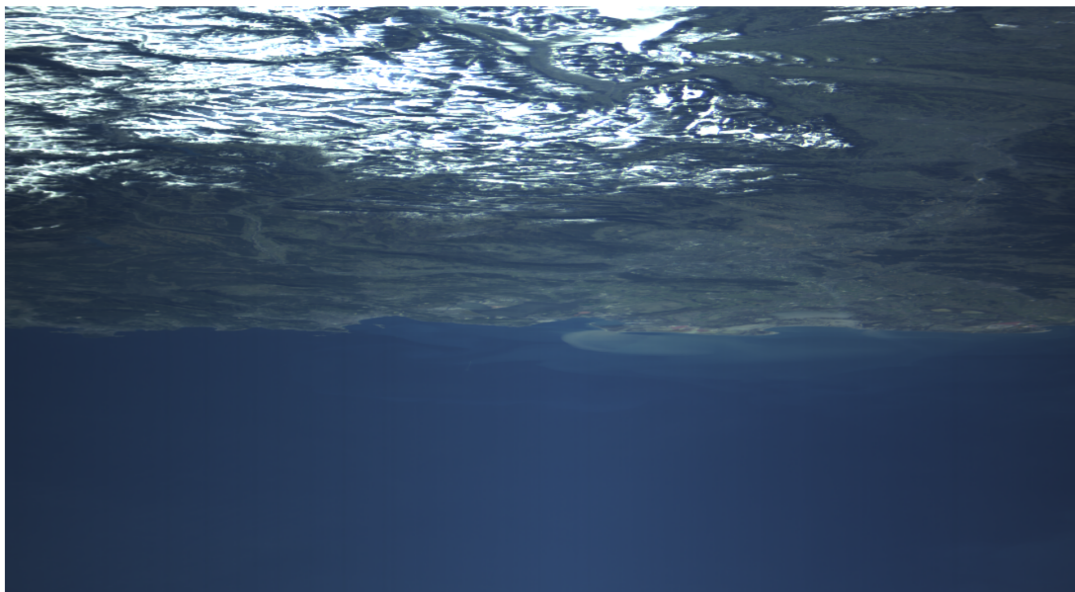
Original Image: kemigawa_2025-01-22T01-31-13Z.png



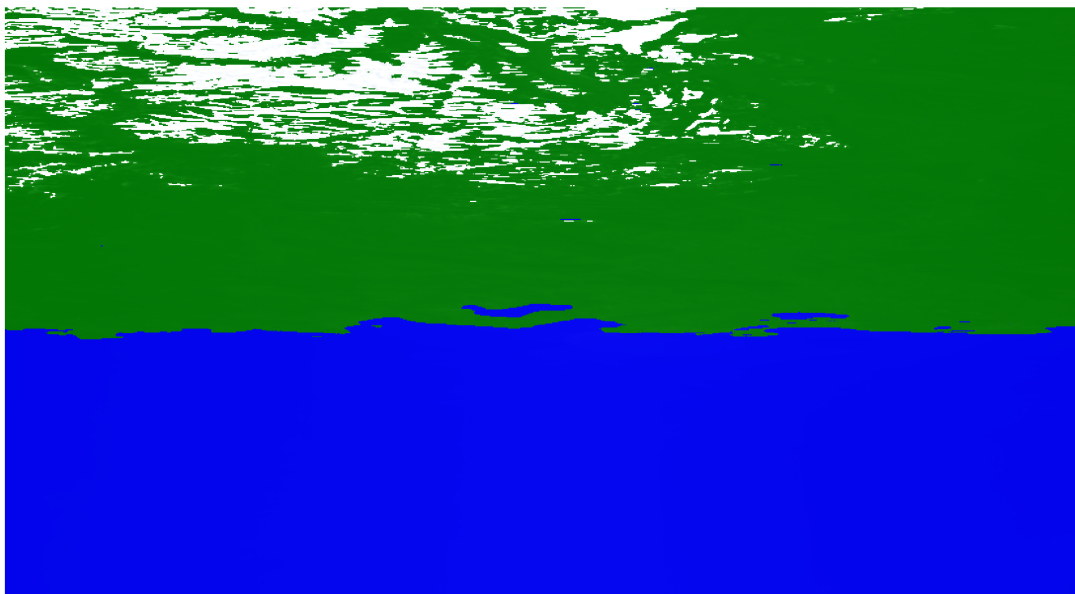
Labeled Image: kemigawa_2025-01-22T01-31-13Z.png



Original Image: lacrau_2024-12-26T11-15-54Z.png



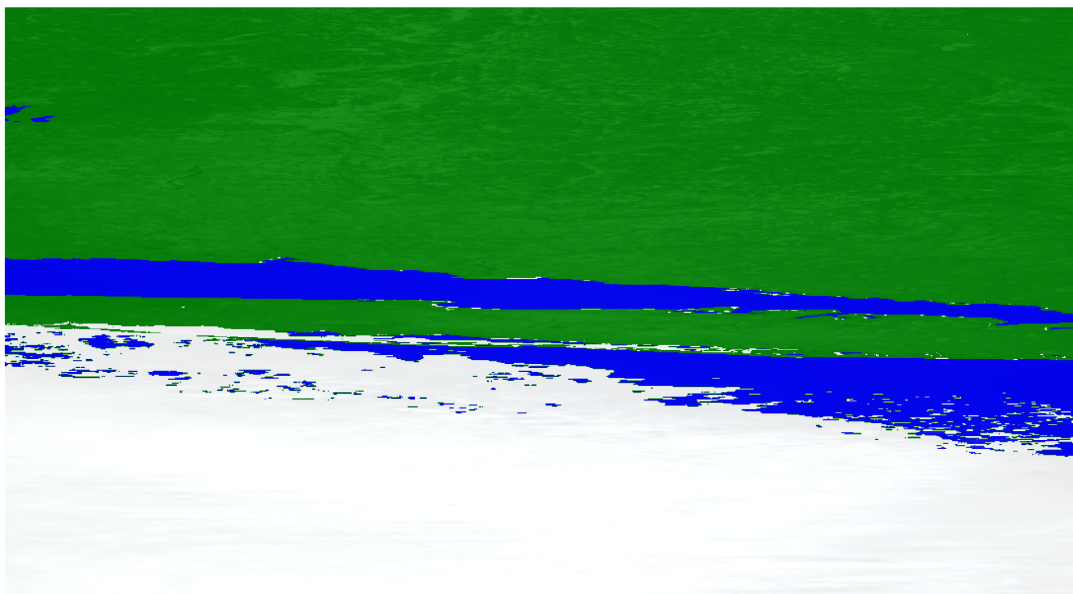
Labeled Image: lacrau_2024-12-26T11-15-54Z.png



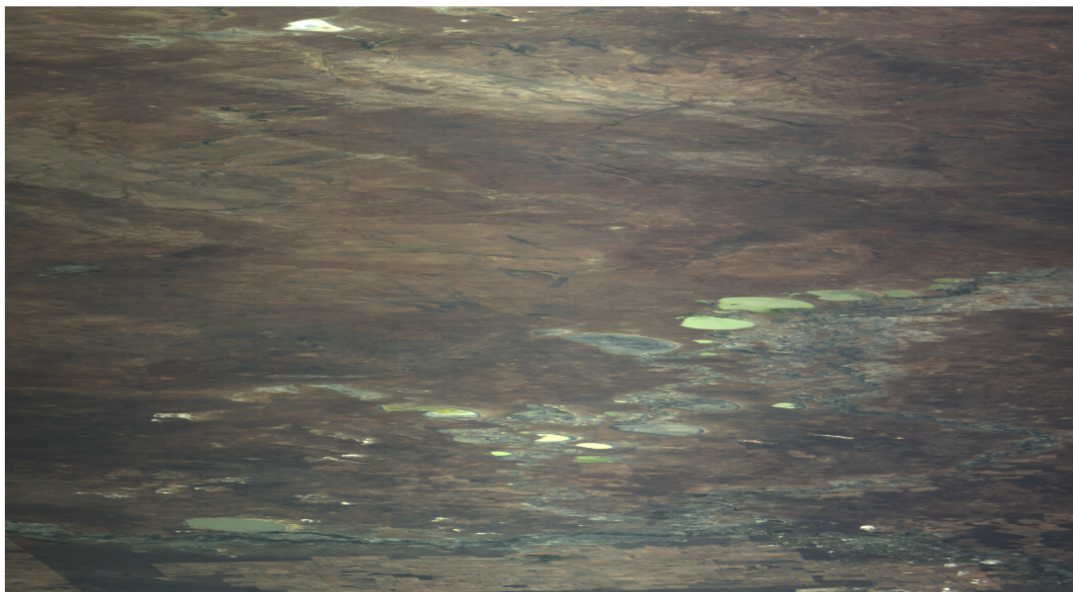
Original Image: longisland_2025-01-22T15-57-39Z.png



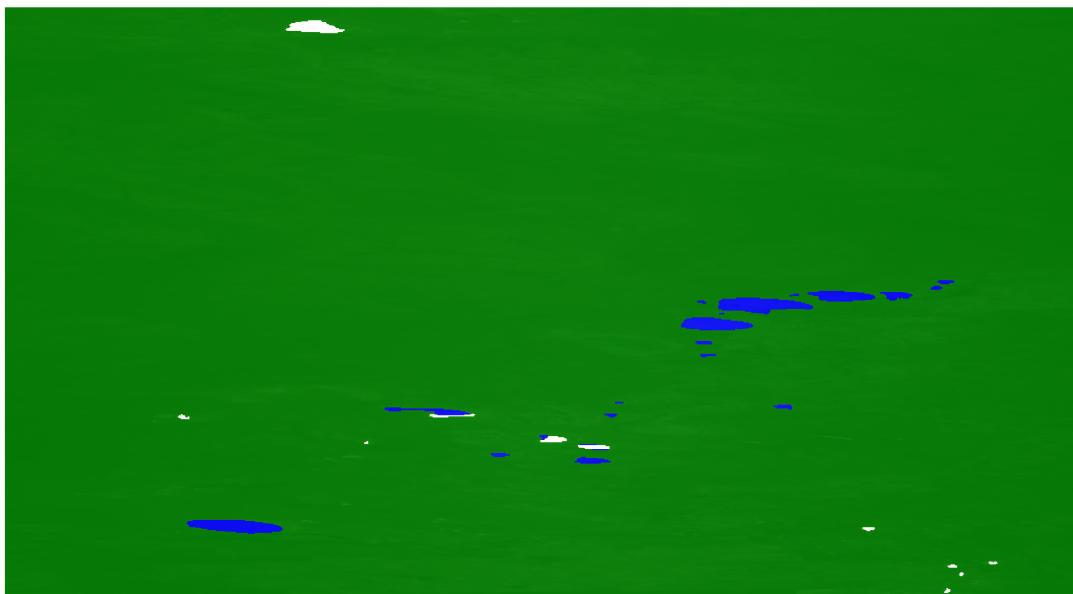
Labeled Image: longisland_2025-01-22T15-57-39Z.png



Original Image: menindee_2025-01-02T01-10-11Z.png



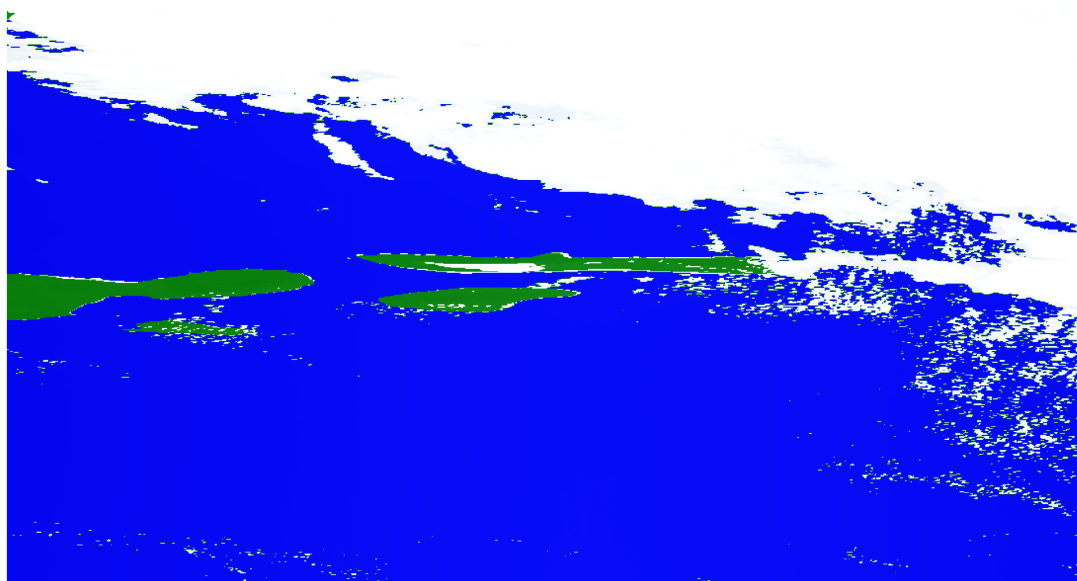
Labeled Image: menindee_2025-01-02T01-10-11Z.png



Original Image: moby_2025-01-08T20-54-59Z.png



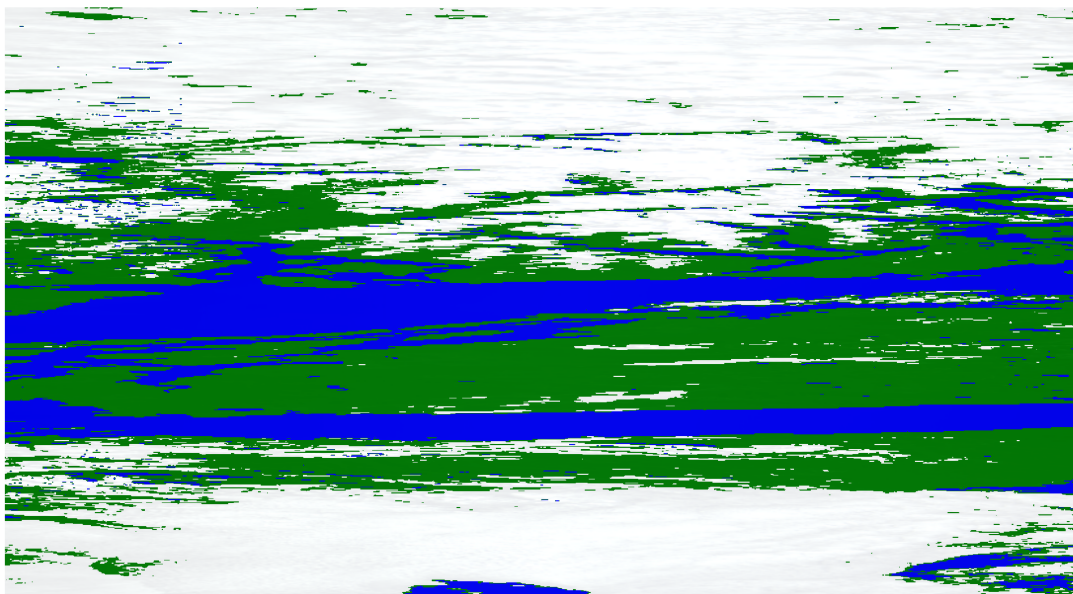
Labeled Image: moby_2025-01-08T20-54-59Z.png



Original Image: straitofgeorgia_2025-01-24T19-21-34Z.png



Labeled Image: straitofgeorgia_2025-01-24T19-21-34Z.png



Original Image: zeebrugge_2025-03-08T11-00-54Z.png



Labeled Image: zeebrugge_2025-03-08T11-00-54Z.png

