

Detecting plume-driven polynyas from dual-pol SAR imagery

Jelle Zitman

Technische Universiteit Delft

Detecting plume-driven polynyas from dual-pol SAR imagery

by

Jelle Zitman

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on August, 25, 2022 at 14:30.

Student number: 4376048
Project duration: September 1, 2021 – August 25, 2022
Thesis committee: Dr. Ir. S. I. M. Lhermitte, TU Delft, daily supervisor
Dr. Ir. B. Wouters, TU Delft, supervisor
Ir. M. Izeboud, TU Delft, supervisor

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



Abstract

Antarctic ocean temperatures are rising due to climate change, causing land ice to melt at increasingly higher rates. Ice shelf bottom melt is a key factor responsible for Antarctic ice mass loss and as such understanding melt processes in the Antarctic is therefore key to more accurately predict how the global sea level will respond to climate change in the foreseeable future. Basal melt results in the formation of both basal melt channels underneath an ice shelf and persistent sea ice wakes (named plume-driven polynyas) at the ice shelf shoreline. The goal of this research is to develop a method that can help to automatically infer basal melt locations along the Antarctic shoreline with significantly increased spatio-temporal resolution compared to previously researched basal melt detection methods. We infer basal melt locations by detecting plume-driven polynyas. We used dual-pol (HH/HV) Sentinel-1 EW SAR data (40x40m resolution) in combination with GLCM textural features as input for a random forest classification that differentiates images as water or ice in four sub-classes: undisturbed 'open' water, disturbed 'rough' water, sea ice and (floating) land ice. We assessed what the advantages and limitations of this approach were for plume-driven polynya detection by performing water-ice (sub-class) classifications and examining which GLCM features proved most useful, what GLCM window size is preferred, and how classification can be aided by post-processing classified images. We computed GLCM textures for window sizes $w = [5, 11, 21]$ and created a classifier for each choice (GLCM5, GLCM11 and GLC21) and compared results to a classifier based on original dual-pol SAR data (BASE). Via cross validated recursive feature elimination we determined that 'sum average' (HH and HV polarization) and 'difference variance' (HV polarization) were most useful for separation of water and ice classes (HH_savg, HV_savg and HV_dvar). Our results have shown that using GLCM texture based dual-pol classifiers improves water-ice classification significantly compared to dual-pol only classifiers, although using HH_savg and HV_savg instead of original dual-pol data comes at a cost of reduced spatial resolution. Water-ice classification accuracy of BASE was 92.2% (kappa = 84.4%) was increased to 95.9% (kappa = 91.5%) for GLCM5, 96.3% (kappa = 92.7%) for GLCM11 and to 96.5% (kappa = 93.0%) for GLC21. From a spatial context, GLCM21 showed an insufficient ability to detect small-scaled bodies of water at a sub-kilometer scale. GLCM5 showed unsatisfactory results in terms of sea ice classification. GLCM11 showed highest robustness in both these performance aspects and proved to be most successful classifier for the application of polynya detection. Using an area filter as a post-processing step proved successful when a classifier is based on GLCM data with a window size no larger than $w=11$. Noise output (small regions of falsely classified open water pixels) was heavily reduced via this form of post-processing and significantly increased polynya detection performance. The final classified product however still contained too many incorrectly classified water regions of similar spatial scales as plume-driven polynyas to be able to apply this algorithm as a reliable automated polynya detection method. We urge to build upon this SAR-based detection method, by using additional non-GLCM input features or using extra post-processing steps, such as temporally filtering water body presence, until results are satisfactory for a fully automated plume-driven polynya detection algorithm. The method presented here has the potential to make detection significantly faster, easier and more accessible than the current methods available. Lastly, in its current state, this method can already be used to validate predicted locations of basal melt by ocean-ice sheet models and DEM-based methods.

Preface

This year has been extremely educational and challenging, but, for a multitude of reasons, ultimately very tough for me. I'd therefore like to take this opportunity to express my gratitude for all the people that have helped me throughout the year.

First of all, a big thank you to my supervisors. Stef, it has been a joy to work together. It has been a chaotic year in many aspects, but I always found you a very kind person to work with, which I value greatly. Your advise was always spot on and I have learned a lot from your way of working. We may unfortunately have not seen each other as much as is normally the case, but the (online) meetings we've had were always helpful. Bert, thank you for being available as the final supervisor in relatively short notice. Your feedback was honest and I highly appreciate the time that you've put in for the supervision. Finally, a massive thanks to Maaike. I'm extremely grateful for all the personal contact that we've had, especially during the second half of the past year. You were always available when I asked to meet, you always gave quick and thorough feedback on intermediate results, and were glad to answer the many questions I threw at you. Your help has been one of the reasons why I've been able to work as hard as I have done over the last couple of months and I thank you immensely for it. Thank you all for your time, effort and honesty. I appreciate it greatly and I wish you all the best.

Secondly, I have to thank my family! Mom, dad, thank you so much for your continuous support throughout the past 8 years. Your support has made me the person I am today. Dad, I want to thank you personally for the help you offered. Even though I in the end haven't made much use of it I've always kept it in the back of my mind. Mom, thanks for the conversations we've had about graduating. I can express whatever I want to you and our conversation in May is one of the reasons my work went much better from then on. Camee, the moments we've had talking about my thesis were sparse given our different interests in our studies, but you've helped me a great deal with the mental side of things. Know that I'll help you in any way I can for the continuation of your own studies, you only need to ask. Thanks, I love you all.

Lastly, a HUGE thanks to all my friends! Your support has helped me tremendously. Derek, thanks for an amazing time as room mates. Some moments were really tough this year and without asking you helped me and lifted my spirits. I cannot ask for a better friend, thank you. Sjonne, we've had similar struggles this year and I appreciate the talks we've had on how to deal with it. You helped me a great deal by supporting my approach and I hope I helped you as well in a similar fashion. Koen, Nienke and Romy, thanks to you as well for having my back and letting me work at your place when I wanted to. Iris, thank you a great deal for listening to me when we talked about this project. I needed it and you were there for me. Paula, Luuk, Tim, thanks for an amazing adventure in the Swedish mountains. Kim, Mitchel, Niek, thanks for working together during the end of the year and during the summer months. I furthermore have to thank you as Soepel for the amazing holiday we had together in the Austrian Alps. I hopped aboard the train pretty exhausted, but you guys fixed me. Lastly, my biggest thanks has to go to Martine. I'm not kidding when I say I could have not have achieved what I have done the past year without you. Your continuous support, your personal interest, working together at IDE or at home, I could always count on you. Thank you so much.

Thanks as well to everyone who helped me that I've not mentioned personally, I love the backing you gave me. It feels amazing to have this many special people around me.

*Jelle Zitman
Utrecht, August 18, 2022*

Contents

1	Introduction	1
1.1	Basal Melt of Antarctic Ice Shelves	1
1.2	Plume-driven Polynyas	1
1.3	Synthetic Aperture Radar	2
1.3.1	Polarization	3
1.3.2	Surface Backscatter Characteristics	3
1.3.3	Textural Features	3
1.4	Study Goal	4
1.5	Approach	4
1.5.1	Google Earth Engine	4
1.5.2	Classification	4
1.5.3	Thesis Outline	5
2	Satellite Imagery & Study Area	6
2.1	Optical Reference Imagery	6
2.1.1	Quality Filter	6
2.2	Synthetic Aperture Radar Imagery	6
2.2.1	Pre-Processing	6
2.2.2	Reference Data Filter	7
2.3	Study Region	7
2.4	GLCM Features	8
3	Method	10
3.1	Data Sampling	10
3.2	GLCM Textures	10
3.2.1	Feature Selection	10
3.2.2	Window Size	11
3.3	Classification	11
3.3.1	Classifiers	11
3.3.2	Random Forest Classification	11
3.3.3	Accuracy Assessment	12
3.4	Post-Processing	12
4	Results	13
4.1	Feature Selection	13
4.2	Feature Characteristics	14
4.3	Classifier Performance	16
4.3.1	Accuracy Assessment	16
4.3.2	Confusion Matrices	17
4.4	Window Size Analysis	17
4.5	Post-Processing	21
5	Discussion	23
5.1	Classification	23
5.2	Post-Processing	25
5.3	Comparison to Other Detection Methods	26
5.4	Recommendations	29

6 Conclusion	30
A Tables & Figures	33
B Code	38
B.1 Ice Shelf Coastlines	38
B.2 (Reference) Image Metadata	41
B.3 Overpasses	45
B.4 Match Data Set	47
B.5 Training & Validation Data	54
B.6 Feature Selection	100
B.7 Visualization of Feature Data	106
B.8 Classification & Accuracy Assessment	108
B.9 Export of Classification & Post-Processing Results	114

List of Figures

1.1 Optical imagery of Pine Island Glacier, showing calving fronts due to basal melt in blue. At the end of basal channels, adjacent to the ice shelf shoreline, plume-driven polynyas are visible in the sea ice. Courtesy of Alley et al. [2019].	2
2.1 Number of dual-pol S1 images acquired within 4 hours of a L8 scene.	7
4.1 Feature Selection for <code>GLCM11</code> . Left: In blue, RFECV to select the optimal number of features and in red the student-t test to assess statistical independence for each number of features. Right: Feature importance for the chosen number of features (n=6).	13
4.2 Boxplots showing the distribution of original backscatter intensities (left) and GLCM features (right) for <code>GLCM11</code> training data	14
4.3 HH, HV and <code>GLCM21</code> texture features for V2. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.	15
4.4 HH, HV and <code>GLCM21</code> texture features zoomed in at small scale water bodies for V2. A water mask from the optical reference image is overlaid in pink for convenience.	16
4.5 Confusion matrices of each classifier for water-ice classification.	17
4.6 Confusion matrices of each classifier for sub-class classification.	18
4.7 Window size influence on water-ice classification. On top of an optical reference image, classification outputs of each classifier are presented as water outlines. <code>BASE</code> : Green, <code>GLCM5</code> : Yellow, <code>GLCM11</code> : Orange, <code>GLCM21</code> : Red.	19
4.8 Classification output for different window sizes, zoomed in at regions of interest.	20
4.9 Overview of impacts of filtering smaller classified bodies of water (750 pixels) from the classification output	22
5.1 Comparison of polynya detection methods zoomed in at regions of interest.	27
A.1 Feature Selection for the <code>GLCM5</code> classifier. Left: In blue, recursive Feature Elimination with Cross Validation to select the optimal number of features and in red a student-t test to assess statistical independence for each number of features. Right: feature importance for the chosen number of features (n=6).	34
A.2 Feature Selection for the <code>GLCM21</code> classifier. Left: In blue, recursive Feature Elimination with Cross Validation to select the optimal number of features and in red a student-t test to assess statistical independence for each number of features. Right: feature importance for the chosen number of features (n=5).	34
A.3 HH, HV and <code>GLCM21</code> texture features for V1. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.	34
A.4 HH, HV and <code>GLCM21</code> texture features for V3. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.	35
A.5 HH, HV and <code>GLCM21</code> texture features for V4. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.	35
A.6 Boxplots showing the distribution of backscatter intensities (left) and textural features (right) for <code>GLCM5</code> training data	36
A.7 Boxplots showing the distribution of backscatter intensities (left) and textural features (right) for <code>GLCM21</code> training data	36

A.8 Post-processing filtering step where bodies of water larger than 750 pixels are distinguished from those smaller than 750 pixels.	37
---	----

List of Tables

2.1	Metadata of Sentinel-1 and LandSat-8 images used as training and validation images. Images are index based on the coordinates of each Sentinel-1 image's footprint. Image IDs of each S1 and L8 image are given in Table A.1.	8
2.2	Names and abbreviations of GLCM features, originally described by Haralick et al. [1973] and Conners et al. [1984].	9
4.1	Classifier performance scores and comparative performance to <code>BASE</code> for water-ice clas- sification.	16
A.1	Image IDs of each Senintel-1 and LandSat-8 match.	33

Introduction

Ocean temperatures are rising due to climate change [IPCC, 2014]. Warmer ocean temperatures have great effects on Arctic and Antarctic environments, where land ice is melting at increasingly higher rates [e.g. IPCC, 2014; Jourdain et al., 2017; Alley et al., 2019]. Antarctic ice mass loss accelerates global sea level rise [IPCC, 2014], but its contribution is still one of the least understood contributing factors [Patty and Morlighem, 2020; IPCC, 2014]. Understanding melt processes in the Antarctic is therefore key to more accurately predict how global sea level will respond to climate change in the foreseeable future.

1.1. Basal Melt of Antarctic Ice Shelves

Most mass loss of the Antarctic Ice Sheet is due to melt of Antarctic Ice Shelves. Many factors are responsible for ice shelf melt, but bottom melt of an ice shelf due to warm ocean currents is one of its most important drivers. Research by Alley et al. [2019]; Lazeroms et al. [2018]; Dutrieux et al. [2014]; Rignot and Steffen [2008]; Lhermitte et al. [2020] has shown that it significantly affects the stability of ice shelves and is responsible for accelerations in sea level rise.

Basal melt occurs when warmer circulatory ocean layers, e.g. Circumpolar Deep Water (CDW), flow upwards along the bottom of ice shelves [Brocq et al., 2013; Alley et al., 2016] and can be responsible for creating 1-10km wide trenches in the ice [Sergienko, 2013; Gourmelen et al., 2017; Alley et al., 2019]. These trenches usually form along the ice shelf's outer edges, though such trenches have also been observed along ice shelf center lines as well. This is the case for one of the west Antarctic ice sheet's glaciers, Pine Island Glacier [Mankoff et al., 2012; Alley et al., 2019]. This glacier shows clear signs of calving due to basal melt, shown in Figure 1.1 for reference.

1.2. Plume-driven Polynyas

As Figure 1.1 shows, apart from melting the bottom of an ice shelf, warmer oceanic waters also melt part of the sea ice where this current reaches the ocean surface. These sea ice wakes are usually confined to the shoreline of the ice shelf at the end of a basal melt channel as they are caused by buoyant meltwater plumes [Lazeroms et al., 2018]. In this study we call these wakes *plume-driven polynyas*. *Polynya* is the preferred nomenclature for regions of open water or thin ice, found in polar sea ice zones where thicker sea ice would be expected based on climatological conditions [Barber and Massom, 2007]. Polynyas can form due to various external factors, but generally fall in two categories: polynyas formed by mechanical forces such as wind, or polynyas formed by convective forces such as warm buoyant water plumes [Williams et al., 2007]. Wind-driven polynyas are generally very large in size and can span up to thousands of kilometers [Barber and Massom, 2007]. Their thermal signature is low, as such water bodies are only ice-free due to harsh meteorological conditions, and can show seasonal variability if climatological conditions show large seasonal variations. On the other hand, plume-driven polynyas are characterized by strong thermal signatures due to warm buoyant water plumes [e.g. Alley et al., 2019; Hellmer et al., 2012] being the cause of melt. These polynyas therefore do not freeze easily if the ocean currents do not vary seasonally. Furthermore, plume-driven polynyas

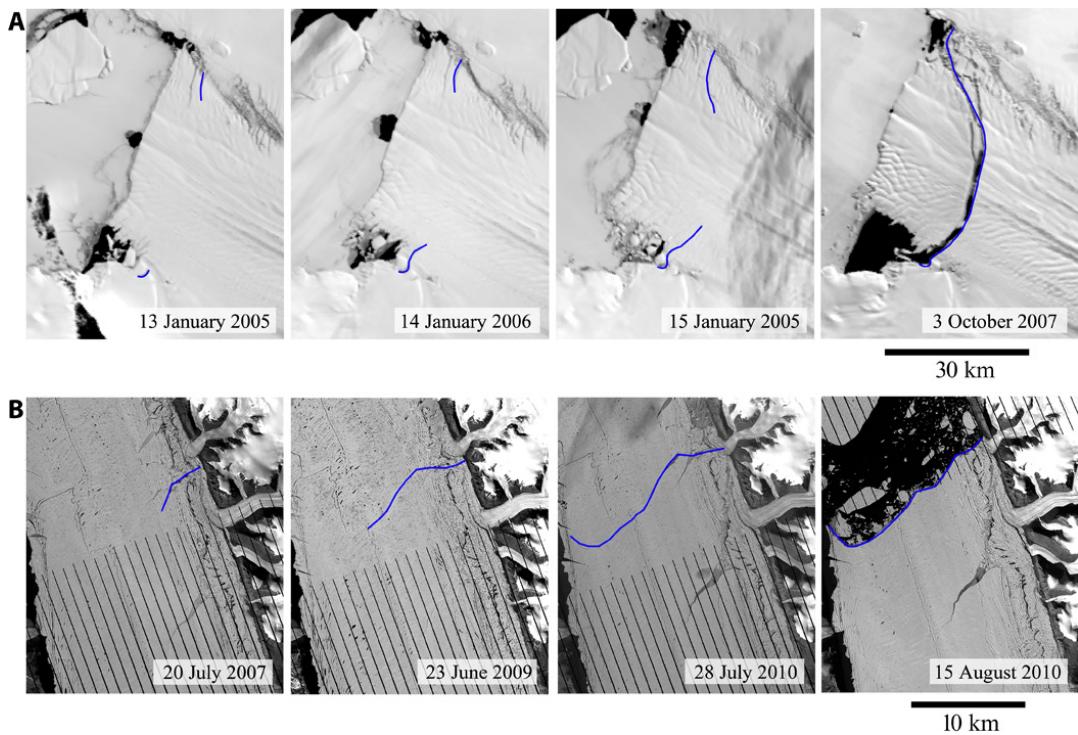


Figure 1.1: Optical imagery of Pine Island Glacier, showing calving fronts due to basal melt in blue. At the end of basal channels, adjacent to the ice shelf shoreline, plume-driven polynyas are visible in the sea ice.
Courtesy of [Alley et al. \[2019\]](#).

are considerably smaller in size than wind polynyas, having diameters roughly in the same order as the width of basal melt channels [e.g. [Alley et al., 2019](#)].

Research has been done to predict where basal melt is occurring. [Gladish et al. \[2012\]](#); [Lazeroms et al. \[2018\]](#); [Sergienko \[2013\]](#), among others, have modelled basal melt distribution and influence on ice shelf stability using ocean models. Others, such as [Dutrieux et al. \[2014\]](#) have used in-situ measurements to describe basal melt characteristics. Another means is to use satellite imagery in combination with an ice sheet model to look at ice shelf dynamics directly, performed by [Lhermitte et al. \[2020\]](#), or to combine satellite imagery with a Digital Elevation Model (DEM) to detect basal melt pathways [\[Shean et al., 2019\]](#). Consequently, research into basal melt via different methods aids in increasing the rate at which we understand Antarctic basal melt, and it allows us to draw more accurate conclusions on its consequences. However, these methods have their own drawbacks. Ice sheet models are computationally expensive, in-situ observations are often sparse and not easily obtained, and the use of DEMs results in a low temporal resolution of basal melt predictions. It would therefore be beneficial to use satellite data to (indirectly) study basal melt, such that the temporal and spatial resolution of predictions remains reasonably high.

In this light, we propose a new method to research Antarctic basal melt. We do not detect basal melt directly, but use remote sensing to detect plume-driven polynyas, which occur as a direct consequence of basal melt channels. The goal of this method is to develop a method to infer basal melt locations along the Antarctic shoreline with significantly increased spatio-temporal resolution compared to previously researched basal melt detection methods.

1.3. Synthetic Aperture Radar

A variety of satellite data could be used for polynya detection. Ideally, we use data that are independent on weather conditions, sun elevation (due to long periods of polar winter), have a spatial resolution higher than 1km, and a high enough temporal resolution to revisit locations multiple times per year. For this purpose, C-band Synthetic Aperture Radar (SAR) data is well suited [e.g. [Dierking, 2010](#); [Karvonen and Hallikainen, 2009](#)]. C-band SAR is an active sensor that sends radar waves, which

penetrate clouds, to measure surface characteristics, meaning it can operate during day and night time [Xu and Li, 2015; Karvonen and Hallikainen, 2009].

1.3.1. Polarization

Depending on the type of sensor SAR data is polarized in different ways, meaning that the plane in which radar waves and the surface interact is defined by the sensor. Each sensor can pick up different surface characteristics as surface depolarization is dependent on surface roughness and on surface composition [Mermoz et al., 2009]. Therefore, SAR data is able to reveal surface characteristics in different ways, depending on the sensor. The polarization of SAR data can be VV (vertically transmitted, vertically received), VH (vertically transmitted, horizontally received), HH (horizontally transmitted, horizontally received) or HV (horizontally transmitted, vertically received). VV and HH are co-polarization backscatter intensities, whereas VH and HV are cross-polarization backscatter intensities. SAR data can be single-polarized (single-pol) by measuring VV or HH or dual-polarized (dual-pol) by measuring VV and VH or HH and HV.

1.3.2. Surface Backscatter Characteristics

Though SAR has many advantages on an operational level, detecting water bodies and sea ice from it is challenging [Karvonen et al., 2005]. Backscatter intensities of water and sea ice have very similar ranges and values are very much dependent on local wind speeds, wind directions and the incidence angle of the sensor. Wind has an immediate effect on the sea state. In very calm conditions, water is a highly specular surface and acts as a mirror, leading to low co-polarization backscatter intensities (≤ -30 [dB]) at an incidence angle of 30° . Conversely, when a sea state is rougher due to higher wind speeds, ripple waves or large waves can significantly increase the backscatter intensity of the water body [Karvonen et al., 2005] ($\simeq -20$ [dB]). HH is a useful polarization as it is not very sensitive to wind roughness [Long et al., 1996], which is helpful to distinguish smooth sea ice from water in windy conditions. The range of backscatter intensity of sea ice is large as well. Sea ice can be very smooth, thereby mimicking calmer open water conditions in terms of backscatter intensity. However, co-polarization backscatter intensities of smooth or wet sea ice are still higher than very calm open bodies of water, due to (more) volumetric scattering and higher surface roughness, but intensities can be comparable to rougher water conditions ($\simeq -20$ [dB]). In the case that sea ice is ridged, or when sea ice is covered in a thick snow layer, surface roughness increases significantly. Furthermore, radar waves are able to penetrate such surfaces more easily due to higher porosity of ice and snow compared to water and are scattered volumetrically, leading to much higher backscatter intensities (≥ -10 [dB]). Water and ice surfaces have somewhat different characteristics for cross-polarization data. Calm, flat water bodies still show very low backscatter intensities (≤ -30 [dB]) as surface roughness and volumetric scattering are minimal. For rougher sea states, the backscatter intensities are more similar to those of calm water ($\simeq -30$ [dB]), as reflection is mostly due to surface scattering, so the incoming radar signal experiences little depolarization [Freeman and Durden, 1998]. For smooth sea ice surfaces, a significantly larger portion of the incoming radar waves is scattered volumetrically, leading to higher cross-polarization backscatter intensities ($\simeq -25$ [dB]) than rough water. In this way, cross-polarized data is helpful for the distinction of sea ice and water, as subtle differences between rougher water and smooth ice are better distinguishable [de Roda Husman et al., 2021], albeit at the expense of a poorer signal to noise ratio than co-polarization data [ESA, 2012]. Cross-polarization backscatter intensities for older ridged ice and snow covered ice remain high as volumetric scattering is dominant for this surface type and as surface roughness is large as well.

1.3.3. Textural Features

Overall, the use of SAR data to classify water and ice has a lot of potential, despite the inherent weaknesses related to similar surface characteristics of rougher waters and smoother ice. Research done by Karvonen and Hallikainen [2009]; de Roda Husman et al. [2021]; Dierking [2013] has proven that SAR can be well-suited for ice type classification. For ice-water classification, Karvonen et al. [2005]; Lohse et al. [2021]; de Roda Husman et al. [2021] showed that dual-pol SAR data in particular can be useful, but that additional textural information is needed for more accurate classification. Spatial information can be derived from co- and cross-polarization backscatter intensities by computing statistical texture metrics around each pixel of a SAR image. This is called a Gray Level Co-occurrence Matrix (GLCM) and is a tabulation of different combinations of grey levels on a specified area surrounding any pixel in

the input image [Haralick et al., 1973; Conners et al., 1984]. The extent of this area is determined by a prescribed window size w . Research done by de Roda Husman et al. [2021], has shown the advantageous use of GLCM textures in the context of river ice classification. Other studies, such as Lohse et al. [2021], have used a similar approach for sea ice detection and concluded that specific texture metrics can be useful for sea ice classification. This is relevant for plume-driven polynya detection, as there is a need to distinguish any ice type from bodies of water as accurately as possible. For one, Lohse et al. [2021] have shown that window size influences classification accuracy. Generally, larger windows are recommended for sea ice state detection, although this comes at a price of spatial resolution. We are therefore interested in the (dis)advantages of dual-pol SAR data in combination with GLCM texture features to detect plume-driven polynyas.

1.4. Study Goal

As basal melt and its extent under Antarctic Ice Shelves is still not fully understood, it is valuable to analyze its dynamics. Basal melt observations are still sparse and difficult to acquire and detecting plume-driven polynyas can give more insight in the presence of basal melt channels. Using SAR satellite data can be a valuable asset in detecting such polynyas as the spatio-temporal coverage of SAR data is significantly higher than in-situ measurements and DEMs. Differentiating water and ice from SAR data can be challenging as both surface types show similarities in cross- and co-polarized SAR images. Furthermore, the use of GLCM textural features has been proven effective in improving water-ice classification accuracy. Studies so far have used SAR data to detect sea ice types and general open water bodies, instead of plume-driven polynyas in particular and have mostly focused on local regions. With this in mind, we focus on an approach to automatically detect plume-driven polynyas, following the research question below:

What are the strengths and limitations of classifying and post-processing dual-pol SAR data and GLCM features for the detection of plume-driven polynyas?

To answer this question, we ask the following sub-questions:

1. *How is open water classification affected when basing a classifier on dual-pol GLCM features, compared to classification where only dual-pol SAR data is used?*
 - (a) *Which GLCM features based on dual-pol polarized SAR data prove most useful for water detection?*
 - (b) *How does the GLCM window size affect classification performance of water bodies?*
2. *In what way do post-processing steps improve plume-driven polynya detection from ice-water classified images?*

1.5. Approach

1.5.1. Google Earth Engine

A disadvantage of using satellite data on continent-wide scales is that data sets become extremely large. Big data analysis requires large computational storage capacity and calculations can quickly become computationally expensive. For this problem, the Google Earth Engine (GEE) provides a solution. The GEE is a cloud-based earth observation platform that allows any user to perform geospatial analyses on a planetary scale through cloud computing [Gorelick et al., 2017]. The GEE comes with a large variety of pre-processed data sets ready for use and is well-suited to analyze Antarctic climate change dynamics. For exactly this reason the GEE is a useful tool for Antarctic polynya detection and will be used as computational platform in this study.

1.5.2. Classification

Many classification algorithms are available, each with their own strengths and shortcomings. First of all, one can choose to use unsupervised or supervised classification. Via unsupervised classification, an image will be classified in a prescribed number of classes, but this method does not provide a means to assign labels to each class beforehand. This means that images have to be analysed using expert

knowledge by comparing it to reference data to interpret the classification. Supervised classification does provide the option to classify the SAR image in prescribed classes, by training data beforehand. No extra step is needed after classification is done and thus supervised classification is preferred, as we want to automate the detection process.

1.5.3. Thesis Outline

To study what GLCM features affect the classification of open water bodies, we developed and analysed multiple classifiers. To analyse the detection accuracy of plume-driven polynyas, we have post-processed the classified images. In [chapter 2](#) we describe which data sets were used, how they were pre-processed and filtered, and define a study region. In [chapter 3](#) we show what textural features were selected and which supervised Machine-Learning (ML) algorithm is used for classification. Furthermore, we show how we have chosen GLCM window sizes and which classifiers we have created and from the selected textural features, ML algorithm and window size. Moreover, we lay out how each classifier was tested on their performance. In [chapter 4](#) we then present classification results and assess how different classifiers compare to each other. Furthermore we apply post-processing steps to the classified data set. Moving to [chapter 5](#), we discuss to what ends the classifier can be used, whether post-processing steps aid in the detection of plume-driven polynyas and provide recommendations for future studies regarding this topic. Lastly, we conclude on the efficacy of adding textural features to dual-pol SAR data for open water classification and the use of post-processing steps on classified images for plume-driven polynya detection in [chapter 6](#).

2

Satellite Imagery & Study Area

This study uses SAR imagery to detect open water bodies, after which plume-driven polynyas are distinguished from other bodies of water in post-processing. As we perform a supervised classification, optical reference data is needed to first manually prescribe feature data to predefined classes. We use data available on the GEE and define a study region based on their spatio-temporal distribution. The presence of optical imagery determines which SAR images are suited as training or validation images. We apply several pre-processing and filtering steps to create this combined data set.

2.1. Optical Reference Imagery

For optical imagery we use LandSat 8 (L8) scenes, which are open-source available. Specifically, we use raw images from Collection 1, Tier 2, as these are available over the Antarctic continent and adjacent waters. Tier 2 raw images display at-sensor radiance that did not meet the quality requirements of Tier 1 images. Factors that determine this quality check are, among others, significant cloud coverage and a lack of ground control. This lack in quality is overcome by manually selecting images of the highest quality during scene selection.

2.1.1. Quality Filter

We have selected optical images based on the following criteria: 1) the footprint of the image should intersect with the coast line of any Antarctic Ice Shelf, 2) the cloud coverage in the image should be lower than 20%, and, to discard images that are too dark, 3) the image should have a maximum brightness above a predetermined threshold. The selection of optical images that we obtained from these filtering steps was matched with SAR images and separated into training and validation images.

2.2. Synthetic Aperture Radar Imagery

When wanting to use SAR data, the GEE provides access to ESA's Sentinel-1 (S1) SAR Level-1 products. The Level-1 product is acquired from a dual-pol C-band SAR sensor [ESA, 2012] and is calibrated and ortho-corrected. Along the Antarctic coast the sensor in operation is mostly single-pol HH, but dual-pol HH and HV is also available. For this reason we use C-band SAR data and we perform our analysis on HH and HV polarized backscatter intensities. The data have resolutions specific to the type of sensor used for acquisition. An advantage of working in the GEE is that most of the other usual pre-processing steps (e.g. SLC processing, radiometric calibration, terrain correction and thermal noise removal) are already accounted for using the Sentinel-1 Toolbox [ESA, 2012].

2.2.1. Pre-Processing

For our study, we used scenes in HH and HV polarization acquired from the Extra Wide Swath instrument mode (EW). These scenes have a footprint of 400x400km and a spatial resolution of 40x40m [ESA, 2012]. Next to backscatter intensity in HH and HV polarization, Sentinel-1 images come with an incidence angle band, whose values range from 20 to 47 degrees [ESA, 2012]. We pre-processed

both image bands by applying an angle correction presented by [Topouzelis and Singha \[2016\]](#) on all images to normalize HH and HV backscatter intensities to an effective angle of (roughly) 30 degrees.

2.2.2. Reference Data Filter

For the selection of training and validation images, we filtered S1 images acquired from 2014-10-01 to 2022-04-01 in several criteria. 1) The footprint of a scene should intersect with the shoreline of any ice shelf, 2) the scene should contain both HH and HV polarized data, and 3) the scene should intersect with the footprint of an optical image from the already filtered L8 scenes, whilst having been acquired within 4 hours of each other. A spatial overview of amount of S1-L8 matches along the Antarctic coast is presented in [Figure 2.1](#) for reference.

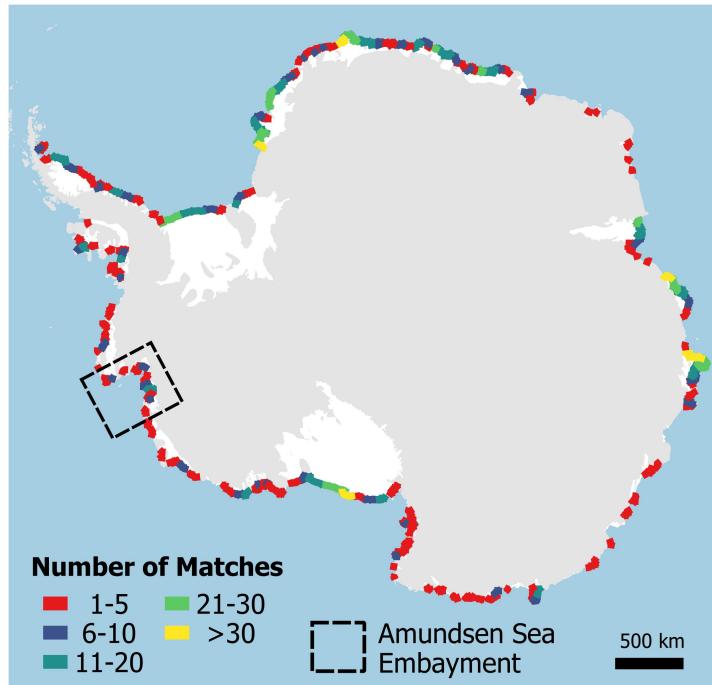


Figure 2.1: Number of dual-pol S1 images acquired within 4 hours of a L8 scene.

2.3. Study Region

Based on [Figure 2.1](#) and on prior knowledge of the presence of plume-driven polynyas [[Bindschadler et al., 2011](#); [Alley et al., 2019](#); [Maqueda et al., 2004](#); [Stewart et al., 2019](#)], we chose a study region for classification. Observations from optical imagery [e.g. [Alley et al., 2019](#)] have shown that most plume-driven polynyas are present in the Amundsen Sea Embayment and as such we choose this coastal region as our region of interest. Although in this region does not contain most S1-L8 matches, enough matches (~ 20) are available to create training and validation data from. We extend the study area some distance west-ward to include the coast near Coney Island. We do this to increase the number of available images, and to make sure that the training and validation data are not acquired in one secluded region of the Antarctic Coast. In this manner, we aim make the classifier more robust and applicable along the entire coast, without the need to resample training data. We filtered the match data set on the study region and performed a final visual quality check where we discarded S1 images that had too little overlap with their L8 match. In the end, the S1 image collection used for this study consists of 13 images, spanning from 2018-2021, and contain at least one L8 match each. We have separated this collection in a training collection of 9 images and a validation collection of 4 images to ensure that the ratio of number of training samples to number of validation samples is roughly 0.7. An overview of the training and validation data, including useful metadata, is given in [Table 2.1](#).

Table 2.1: Metadata of Sentinel-1 and LandSat-8 images used as training and validation images. Images are index based on the coordinates of each Sentinel-1 image's footprint. Image IDs of each S1 and L8 image are given in [Table A.1](#).

Match ID	Data Set	Satellite	Local Date	Local Time	Types of Water Bodies in Image
T1	Training	Sentinel-1	2017/12/26	20:52:24	
		LandSat-8	2017/12/26	23:28:03	Open Sea, Water Leads, Wind Polynya, Plume Polynya
		LandSat-8	2017/12/26	23:28:27	
T2	Training	Sentinel-1	2017/12/21	20:44:13	Open Sea
		LandSat-8	2017/12/22	00:09:32	
T3	Training	Sentinel-1	2018/12/18	20:28:00	Open Sea
		LandSat-8	2018/12/18	23:46:17	
T4	Training	Sentinel-1	2021/12/26	20:28:17	Open Sea
		LandSat-8	2021/12/26	23:46:42	
T5	Training	Sentinel-1	2018/01/11	20:19:41	Wind Polynya, Plume Polynya
		LandSat-8	2018/01/11	23:27:57	
T6	Training	Sentinel-1	2018/12/13	20:19:48	Open Sea, Water Leads, Wind Polynya
		LandSat-8	2018/12/13	23:28:08	
T7	Training	Sentinel-1	2017/12/13	21:28:20	Open Sea
		LandSat-8	2017/12/13	23:37:50	
T8	Training	Sentinel-1	2018/01/07	20:31:08	Open Sea
		LandSat-8	2018/01/07	23:31:35	
T9	Training	Sentinel-1	2018/01/08	20:33:39	Open Sea, Wind Polynya
		LandSat-8	2018/01/08	23:36:21	
V1	Validation	Sentinel-1	2017/12/19	21:00:36	Wind Polynya, Plume Polynya
		LandSat-8	2017/12/19	23:21:54	
V2	Validation	Sentinel-1	2017/12/23	20:27:53	Open Sea, Water Leads, Wind Polynya, Plume Polynya
		LandSat-8	2017/12/23	23:57:10	
		LandSat-8	2017/12/23	23:57:34	
V3	Validation	Sentinel-1	2020/01/06	20:28:05	Open Sea, Water Leads
		LandSat-8	2020/01/06	23:46:37	
V4	Validation	Sentinel-1	2017/12/18	20:58:33	Open Sea, Water Leads
		LandSat-8	2017/12/18	23:17:55	

2.4. GLCM Features

To introduce extra information to the original HH and HV polarized SAR data, we compute GLCM textural features. Through these computations, a prescribed amount of pixels around a center pixel are statistically interpreted in various ways, each resulting in a unique texture metric. In the GEE, 17 GLCM output features are computed per input band, resulting in 36 unique input features (including HH and HV) for each classifier. As stated by [Haralick et al. \[1973\]](#), GLCM features are computed from entries in three types of gray-tone spatial-dependence matrices: $p(i,j)$, $p_{x+y}(k)$ or $p_{x-y}(k)$. $p(i,j)$ is the (i,j)th entry in the original normalized gray-scale matrix. Entries of the other adjusted gray-scale matrices are described via

$$p_{x+y}(k) = \sum_{\substack{i=1 \\ i+j=k}}^{N_g} \sum_{j=1}^{N_g} p(i,j), \quad k = 2, 3, \dots, 2N_g \quad (2.1)$$

used for computing 'summed' features, and

$$p_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{\substack{j=1 \\ |i-j|=k}}^{N_g} p(i,j), \quad k = 0, 1, \dots, N_g - 1 \quad (2.2)$$

which is the notation for computing 'difference' features. In both equations, N_g represents the number of distinct grey levels in the quantized image. 13 unique features are extracted by computing different statistics from each matrix. In a similar but more elaborate fashion, [Conners et al. \[1984\]](#), present six extra unique features, of which four are computed in the GEE. For clarity, we present an overview of names and descriptions of these 17 GLCM features in [Table 2.2](#).

Table 2.2: Names and abbreviations of GLCM features, originally described by [Haralick et al. \[1973\]](#) and [Conners et al. \[1984\]](#).

Index	Feature Name	Abbreviation	Matrix	Authors	Description
1	Angular Second Moment	ASM	$p(i,j)$	Haralick	Number of repeated grey-level pairs (energy)
2	Contrast	CONT	$p(i,j)$	Haralick	Local contrast of grey-levels
3	Correlation	CORR	$p(i,j)$	Haralick	Correlation between pairs of grey-levels
4	Variance	VAR	$p(i,j)$	Haralick	Spread in grey-levels
5	Inverse Difference Moment	IDM	$p(i,j)$	Haralick	Homogeneity of grey-levels
6	Sum Average	SAVG	$p_{x+y}(k)$	Haralick	Mean of 'summed' grey-level matrix
7	Sum Variance	SVAR	$p_{x+y}(k)$	Haralick	Variance of 'summed' grey-level matrix
8	Sum Entropy	SENT	$p_{x+y}(k)$	Haralick	Entropy of 'summed' grey-level matrix
9	Entropy	ENT	$p(i,j)$	Haralick	Randomness of grey-levels
10	Difference Variance	DVAR	$p_{x-y}(k)$	Haralick	Variance of 'difference' grey-level matrix
11	Difference Entropy	DENT	$p_{x-y}(k)$	Haralick	Entropy of 'difference' grey-level matrix
12	1st Correlation Measure	IMCORR1	$p(i,j)$	Haralick	Extra information measure of cluster correlation (1)
13	2nd Correlation Measure	IMCORR2	$p(i,j)$	Haralick	Extra information measure of cluster correlation (2)
14	Dissimilarity	DISS	$p(i,j)$	Conners	Distinction between grey-levels (named Local Homogeneity in Conners et al. [1984])
15	Inertia	INERTIA	$p(i,j)$	Conners	Periodicity of grey-levels
16	Cluster Shade	SHADE	$p(i,j)$	Conners	Uniformity of grey-levels
17	Cluster Prominence	PROM	$p(i,j)$	Conners	Proximity of grey-levels

For a full description of the formulas behind these features, we urge to consult [Haralick et al. \[1973\]](#) for features 1-13, and [Conners et al. \[1984\]](#) for features 14-17.

3

Method

In this study we compare different classifiers on their ability to separate open water from ice from dual-pol SAR imagery. We created multiple classifiers using different selections of input features. For this we computed texture features from backscatter intensity in HH and HV polarization for various window sizes. After classification, we applied post-processing steps to translate the classified open water bodies to polynyas. Every step in this process is discussed in detail below.

3.1. Data Sampling

We sampled data in different (sub-)classes to use for classification. We divided data in two main classes, *Water* and *Ice*, each consisting of two sub-classes: *Open Water*, *Rough Water*, *Sea Ice* and *Floating Land Ice* respectively. S1 images were manually assigned to a (sub-)class, based on several criteria from L8 images (*I*) and S1 HH (*II*) and HV (*III*) polarized backscatter intensities. Part of an S1 image is assigned as *Open Water* if *I* clearly shows a water body, while *II* and *III* are both near minimum intensity values. It is seen as *Rough Water* if *I* shows water, but either *II* or *III* has low to medium values, while the other has low to very low values. For *Sea Ice* the criteria of *II* and *III* are equal to that of *Rough Water*, but now *I* shows a presence of ice instead of water. Finally, part of the image is considered as *Floating Land Ice* when *I* shows ice and *II* and *III* both show high backscatter intensities.

Training and validation data were created by extracting data from 1,000 randomly selected pixels for every sub-class in each image. Sampling 1,000 pixels was an imposed computational upper limit. This resulted in a training data set consisting of 9,000 samples per feature per sub-class (i.e. 18,000 per class) and a validation data set of 4,000 samples per feature per sub-class (8,000 per class).

3.2. GLCM Textures

3.2.1. Feature Selection

As mentioned in [chapter 1](#), spatial information is derived from HH and HV backscatter intensities by computing various GLCM features. GLCM texture calculations provide 17 output features per input band, given in [Table 2.2](#). To analyse the added value of each feature (both for HH and HV), we performed a cross validated recursive feature elimination (RFECV). This method tests the accuracy of a classifier for a chosen number of features, ranging from 1 to all features available, on a subset of the training data set. In our case we divide the training data in three folds, and then compute the average accuracy over all folds. We can then apply a normal recursive feature elimination (RFE) on the full training data set to determine the n most important features. The algorithm determines the least important feature for each iteration and discards that feature until the desired number of features in order of highest importance is computed.

Apart from feature elimination, we also tested how statistically dependent training labels were from classified labels. We did this by performing a student t-test on each classification outcome. The statistical dependency is expressed in a p-value, where values lower than a threshold (usually 0.05) indicate that two variables are likely independent or unrelated. Conversely, higher p-values indicate that two

variables are likely dependent or closely related.

We use the accuracy and p-values to assess which features are best for the classifier. We do this by first looking for the number of features where accuracy is significantly high, and the p-value has approached or reached its maximum. Then, we also looked at how much a feature correlates to any other feature in each data set. Features that were correlated more than 90% to each other were considered highly correlated and of these features only the feature with the highest importance score was used as input data for the classifier. We chose to do this to minimize the number of input features, so that we were able to more closely look at the influence that a feature has on the classification output. We have made the assumption that excluding highly correlated features does not impact the classification result significantly, and that for this reason such omissions are allowed. We discuss the eventual selected features in more detail in [chapter 4](#).

3.2.2. Window Size

For the detection of different sea ice states, [\[Lohse et al., 2021\]](#) recommends the use of relatively large window sizes, as they lead to more accurate classifications overall. They suggest an optimal window size of $w = 51$ pixels for S1 EW imagery (i.e. 2040x2040m) and mention that window sizes of 21 and 11 pixels (i.e. 840x840m and 440x440m respectively) can be used as well, even though they lead to less accurate sea ice classification. Another study by [\[de Roda Husman et al., 2021\]](#) suggested the use of $w = 11$ for river ice classification on S1 IW imagery (a window size of 110x110m).

These studies were, however, not cloud-based, and therefore encountered different computational restrictions than in this study. It proved particularly tedious to sample large GLCM texture data sets for window sizes over $w=21$. As such our choice of w is a compromise between improving expected accuracy (large window size) and minimizing computational effort (small window size). Moreover, a downside of larger window sizes is the reduced spatial resolution that is introduced by the statistical computations over each GLCM window [\[Lohse et al., 2021\]](#). As we aim to detect plume-driven polynyas that are mostly in the order of 1-10km² [e.g. [Gourmelen et al., 2017](#); [Alley et al., 2019](#)], we expect that a window size of more than 21x21 pixels (0.71 km²) would reduce the spatial resolution too much for it to be a useful window size for the purpose of this study.

In the end we computed GLCM textures for $w = [5, 11, 21]$. The addition of $w = 5$ is to assess how successful the choice of a relatively small, but computationally efficient window size to base a classifier on proves to be. As a GLCM feature is dependent on the window size w we compute these features for three different values of w and create different classifiers for each. In this way we eventually assess which choice of w is optimal for plume-driven polynya detection in [chapter 4](#).

3.3. Classification

3.3.1. Classifiers

Multiple classifiers were developed. We start of with a base classifier, consisting of original HH and HV backscatter intensity as its input features. Three more classifiers were constructed, each using one of the three GLCM texture data sets as their input features. For clarity we will name the base classifier BASE. The classifiers for the GLCM data sets with window size $w = [5, 11, 21]$ are named GLCM5, GLCM11 and GLCM21 respectively. We primarily focus on how well the classifiers perform at classifying the input images in water and ice. However, we also compare how they classify the images in their respective sub-classes to get an indication of how well they are able to distinguish rougher water from smoother ice.

3.3.2. Random Forest Classification

Several supervised classification algorithms are applicable for the detection of water bodies in sea ice, but research by [Hoekstra et al. \[2020\]](#); [Shelestov et al. \[2017\]](#) has concluded that a Random Forest (RF) classifier is most useful for water-ice detection. [Hoekstra et al. \[2020\]](#) mentions that RF classification can be done with a multitude of input features, which is useful as we combine dual-pol SAR and GLCM features. Furthermore, an aspect of an RF classifier that is especially well-suited for this study is that the classifier is simultaneously able to assess feature importance [\[Hoekstra et al., 2020\]](#). This allows for an in-depth analysis of optimal feature selection to improve classification accuracy. [Shelestov et al. \[2017\]](#) has analysed supervised machine learning algorithms in the GEE specifically and concluded that the GEE RF algorithm generally performs well and can be used as a reliable tool for classification.

In an RF classifier several hyper-parameters need to be specified. First of all, the classifier creates a prescribed number of decision trees, which are used predict a label for each pixel based on randomly sampled subsets of training data. For this, we manually define the desired number of classification trees (*numberOfTrees*). Furthermore, we define the minimal number of variables per split in a tree (*variablesPerSplit*), the minimum number of points in a training subset (*minLeafPopulation*) and the maximum number of nodes per tree (*maxNodes*). In Python, *GridSearchCV* from the *sklearn* package was used to test what configuration of these input parameters results in the highest accuracy score. This approach uses cross validation to determine the accuracy score of each combination of parameters and thus ensures that the validation data remains independent from the classification algorithm. When comparing 'optimized' hyperparameter choices from the *GridSearchCV* algorithm with 'standard' hyperparameter choices, we however concluded that the optimized hyperparameter choices resulted in similar or poorer classification results than the standard hyperparameter choices. We believe that the optimized values resulted in a classifier that was over-fitted on the training data and chose to use standard values as they resulted in a classifier that performed equally well, and was less prone to over-fitting. The following hyperparameter values were used: *numberOfTrees*=10, *variablesPerSplit*=2, *minLeafPopulation*=1, *maxNodes*= ∞ .

3.3.3. Accuracy Assessment

To assess how well each classifier performs, we compute each classifier's Cohen's kappa score [Cohen, 1960]. Cohen's kappa score is an accuracy metric that takes into account the full confusion matrix of a classifier. Where a normal accuracy metric is only dependent on true positives and true negatives, Cohen's kappa also depends on false positives (ice incorrectly classified as water) and false negatives (water incorrectly classified as ice), meaning that it gives an more complete view of a classifier's performance. Furthermore, the False Negative Ratio (FNR) and False Positive Ratio (FPR) of each classifier were computed. The FNR is the ratio of True Negatives to False Negatives, and is a measure to tell what fraction of water pixels are incorrectly classified as ice. Conversely, the FPR is the ratio of False Positives to True Positives and relates to the number of ice pixels incorrectly classified as water. As we wish to create a classifier that is specialized to detect isolated water bodies in frozen seas, we primarily aim to create a classifier with a FNR as low as possible.

To assess which classifier performed best, the kappa score of each classifier was compared to that of BASE. In this comparison, the comparative performance was expressed as an improvement score. This score is a fraction of the original kappa score, which was positive if the classifiers performs better and negative if the classifier produces worse results. The best classifier was chosen by looking at its improvement score, FNR, and by visually comparing classification results from each classifier in regions where polynyas, water leads, and smooth sections of sea ice are present.

Finally, we compared each classifier's confusion matrix for classification in main classes (*Water*, *Ice*) and for classification in sub-classes (*Open Water*, *Rough Water*, *Sea Ice* and *Floating Land Ice*) to be able to analyze which (sub-)class distinctions prove easy and difficult to accurately predict.

3.4. Post-Processing

The choice of smaller or larger window sizes will result in different strength and weaknesses of each classifier. Smaller window sizes will lead to a higher spatial resolution of the classification output [Lohse et al., 2021], but are expected to be less robust than one based on a larger window size. Plume-driven polynyas are characterised by an area of open water (1-10km², [Alley et al., 2019]). Classified images of each classifier are expected to detect and classify pixels in these regions as relatively large and uniform bodies of water (>1km²). However, a fraction of incorrectly classified water pixels as sea ice is expected to occur (represented by each classifier's FPR). For these cases, we expect that the bodies of water detected are more irregular and, in general, small in size. We use this assumption to apply a filter based on the area of each detected body of water in the classified image. As small polynyas of approximately 1 km² in size can occur, we apply a threshold of 1.2km² (750 pixels of 40x40m) to filter out incorrectly classified water pixels.

To assess how post-processing affected classification, we analyzed the filtered images on a spatial basis. We compared each classifier result with and without applying an area filter and assessed which combination of classification and post-processing resulted in the optimal strategy for detecting plume-driven polynyas.

4

Results

4.1. Feature Selection

We applied RFECV to each texture set. An example for GLCM textures of $w=11$ is shown in Figure 4.1. The figure shows the accuracy of RF classification per n number of features. Plotted in red are the p-values that show how statistically dependent the classified labels are from the training labels. Using 16 features results in the best classifier for this data set. As the p-value for this choice of feature number is sufficiently high and shows no significant increase when using higher feature numbers, 6 features were chosen as a more practical choice. Feature importance scores of the top 6 features are given in Figure 4.1b and are in descending order of importance: Sum Average (HV polarization), Backscatter Intensity (HH polarization), Difference Variance (HV polarization), Dissimilarity (HV polarization), Difference Entropy (HV polarization) and Sum Average (HV polarization). We will from now on refer to these features as `HH_savg`, `HH_dvar`, `HV_diss`, `HV_dent` and `HV_savg` respectively.

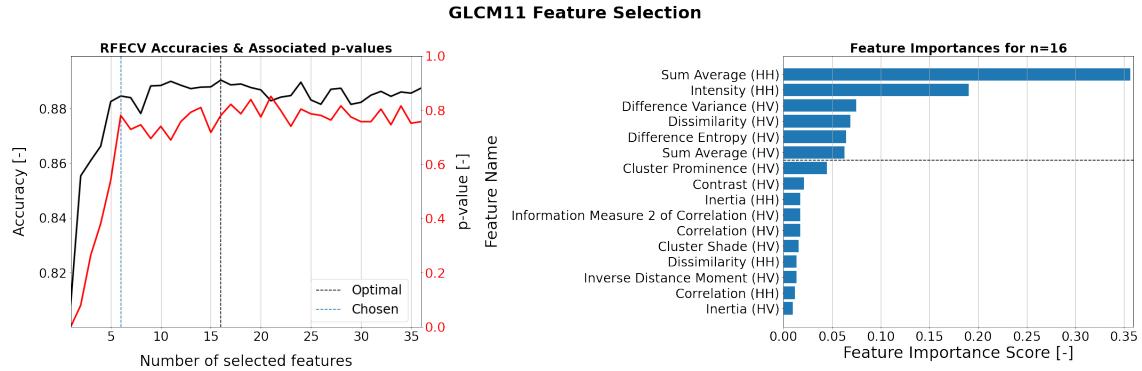


Figure 4.1: Feature Selection for GLCM11. Left: In blue, RFECV to select the optimal number of features and in red the student-t test to assess statistical independence for each number of features. Right: Feature importance for the chosen number of features ($n=6$).

Finally, the final choice of features was based on these features and their correlations. As `HV_dvar` was highly correlated to both `HV_diss` (96.0%) and `HV_dent` (90.1%) and as Figure 4.1b shows that `HV_diss` and `HV_dent` are of less importance than `HV_dvar`, `HV_diss` and `HV_dent` are both discarded. Similarly, `HH` was dropped as it correlates for 97.5% with `HH_savg` and the importance score of `HH_savg` is significantly higher. This left us with the final selection of features: `HH_savg`, `HV_savg` and `HV_dvar` to base GLCM11 on. We repeated this process for the texture data sets of other window sizes and found that for both the data sets of GLCM5 as well as GLCM21, `HV_dvar`, `HH_savg` and `HV_savg` were the optimal features to select. Results for these data sets are presented in Figure A.1 and A.2.

4.2. Feature Characteristics

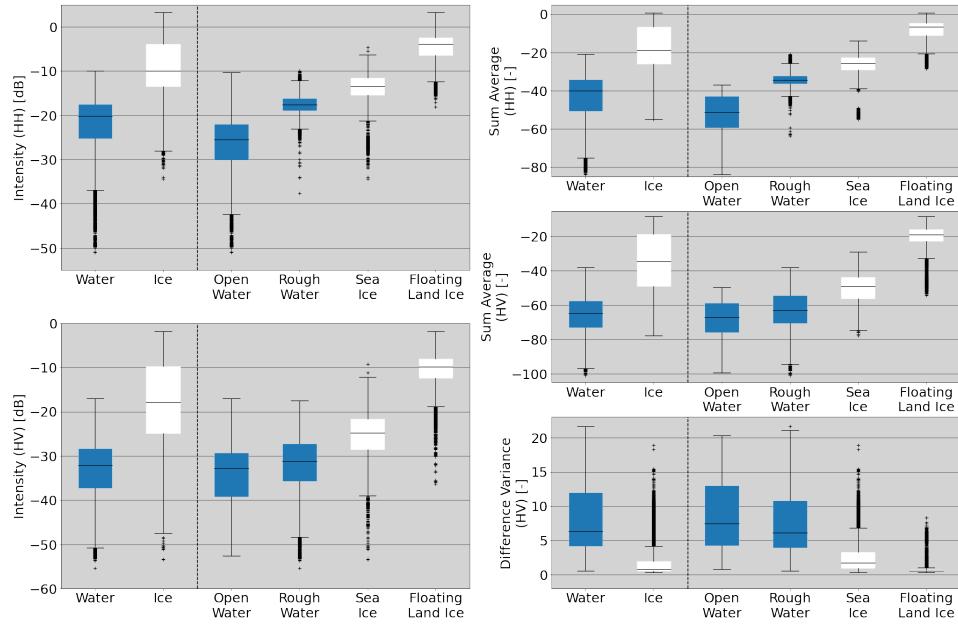


Figure 4.2: Boxplots showing the distribution of original backscatter intensities (left) and GLCM features (right) for GLCM11 training data

[Figure 4.2](#) shows a spectral distribution of the training data per (sub-)class. The left boxplots show the input features of BASE, the original HH and HV backscatter intensities. The right boxplots show the input features of GLCM11. Boxplots of GLCM5 and GLCM21 are similar to results in [Figure 4.2](#) and are presented in [Figure A.6](#) and [A.7](#).

In general, *Water* is characterized by low median values for HH (~-20 [dB]), HV (~-32 [dB]), HH_savg (~-40 [-]) and HV_savg (~-65 [-]). For calmer open water low intensities in HH and HV polarization are expected due to surface scattering and low surface roughness. On the other hand, *Water* shows high median HV_dvar values (~6 [-]), even though the spread in values is large (~0.1 to 22 [-]). Conversely, spectral characteristics for ice are in a general sense opposite to those of water for each feature. Rough(er) surfaces of types of *Ice* and more volumetric scattering compared to *Water*, result in higher median values for HH (~-10 [dB]), HV (~-18 [dB]), HH_savg (~-20 [-]) and HV_savg (~-35 [-]). *Ice* has a very low median value for HV_dvar, at ~0.1 [-]. HV backscatter intensity is extremely low for *Water* and its sub-classes *Open Water* and *Rough Water*. Their range goes down to less than -50 [dB], which is well below the noise floor of the cross-polarization sensor (-40 [dB], [ESA \[2012\]](#)). Values lower than the noise floor are considered as outliers, but are kept in the training data as we expect RF classification to handle the presence of outliers well.

When looking at the spectral distribution of *Water* and *Ice*, the difference between the two classes can appear to be quite distinct for both HH and HV as median values are significantly different. However, [Figure 4.2](#) also shows that this is due to the influence of the more easily distinguishable sub-classes of water and ice: *Open Water* (median HH value ~-25 [dB], median HV value ~-33 [dB]) and *Floating Land Ice* (median HH value ~-5 [dB], median HV value ~-10 [dB]). *Open Water* is characterized by low values of HH_savg and HV_savg, due to its smooth specular surface. *Floating Land Ice*, shows the opposite behaviour. Values in HH_savg and HV_savg are higher than those of *Water* due to its rough surface and its composition of mostly snowy surfaces, leading to higher backscatter intensities. The sub-classes *Rough Water* and *Sea Ice* are a lot more ambiguous regarding their median intensities in both HH (~-18 [dB] and ~-14 [dB] respectively) and HV (~-31 [dB] and ~-26 [dB] respectively). This ambiguity is still present in HH_savg (~-35 [-] and ~-25 [-] respectively) and HV_savg (~-65 [-] and ~-50 [-] respectively). The HH_savg and HV_savg values of *Sea Ice* have similar standard deviations as those of *Rough Ice*, but its median values are higher. The spread in intensities is considerable as HH_savg and HV_savg intensities are dependent on the particular water and sea ice states at the

moment of data acquisition. Some sea ice states are smoother, less snowy, or wetter than others. These sea ice states reflect few radar waves back to the sensor, since more surface scattering takes place, and surface roughness is more comparable to that of *Rough Water*.

`HV_dvar` shows its strength in its ability to separate the *Rough Water* from the *Sea Ice*, as median values for both sub-classes differ significantly (~ 6 [-] and ~ 1 [-] respectively). This is reflected in [Figure 4.2](#), as `HV_dvar` achieved third highest importance score behind `HH_savg` and `HH`. Opposed to `HH_savg` and `HV_savg` the median value of `HV_dvar` for *Sea Ice* is relatively low. The spread in values however is much higher than that of *Floating Land Ice*, as state of sea ice for *Sea Ice* can become near as smooth as some (rougher) water surfaces. Still, all texture features show overlap for each sub-class (*Rough Water* and *Sea Ice* in particular) outside of the 1st and 3rd quantile, meaning that fully accurate classification remains a challenge and classification errors are still likely to occur.

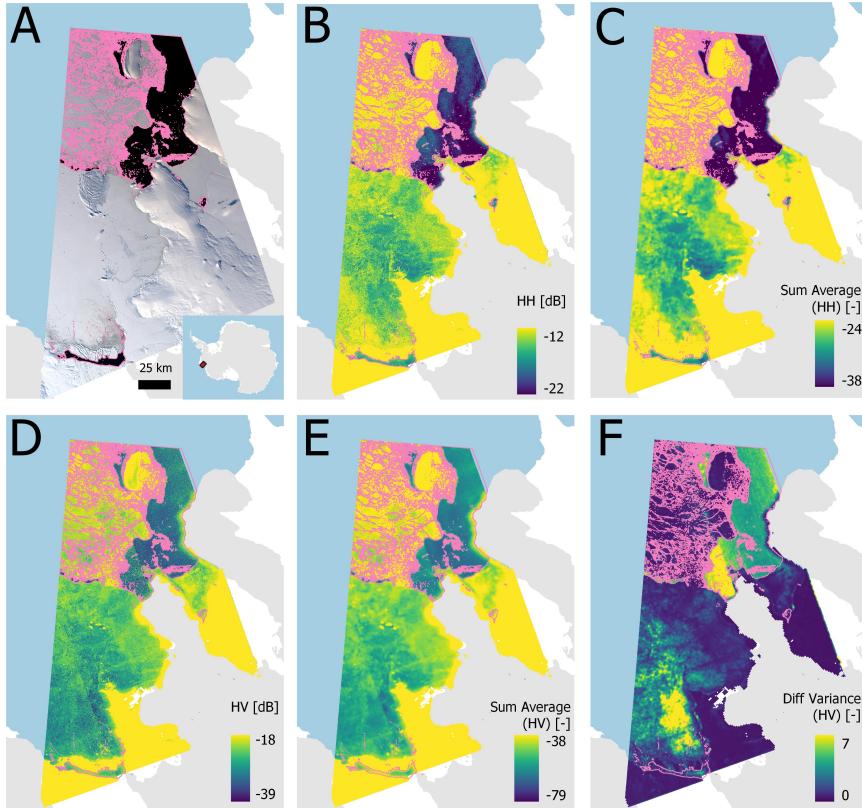


Figure 4.3: `HH`, `HV` and `GLCM21` texture features for *V2*. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.

We visualized each `GLCM21` texture feature for *V2* in [Figure 4.3](#) for visual interpretation. For reference, texture features for *V1*, *V3*, *V4* are presented in [Figure A.3](#), [A.4](#), [A.5](#). We show `GLCM21` features as these show the largest contrast with the original `HH` and `HV` features, making them best-suited for visual comparisons. First off, when comparing `HH` (4.3B) and `HV` (4.3D) to `HH_savg` (4.3C) and `HV_savg` (4.3E), we notice the smoothing effect of the statistical matrix computations on original input features. In large sea ice or open water regions, the feature data of `HH_savg` and `HV_savg` are much more uniform and show significantly less variation in magnitude within such regions. In [Figure 4.3E](#) we see that large open water bodies are associated with higher values of `HV_dvar` (in accordance with [Figure 4.2](#), although we see similar pattern in specific regions of sea ice. It appears that smaller bodies of water are not necessarily associated with higher `HV_dvar` values. To highlight this, we zoom in at three specific regions of small scaled water bodies in [Figure 4.4](#). Smaller bodies of water do not consistently show high values of `HV_dvar` as the water in bodies in [4.4V2-1](#) and [4.4V2-3](#) both show lower ranged `HV_dvar` values, whilst [4.4V2-2](#) is associated with higher values. For bodies of water, the spread in `HV_dvar` is therefore significant (see also [Figure 4.2](#)). Furthermore, we see an edge smoothing effect of `GLCM21` features in [4.4V2-1](#) and [4.4V2-3](#). Where *Floating Land Ice* is present next to *Water*, high

values of HH_savg and HV_savg are found in the original open water body with a width comparable to window size $w=21$. In 4.4V2-2 an edge pattern is also present. 4.3V2-2D (and 4.3V2-2E, but slightly less apparent) shows that a low intensity edge extends outwards from the water body on the western side, while a high intensity edge protrudes inwards into water body's eastern edge. GLCM5 and GLCM11 data show similar results, but the spatial extent of edge smoothing is less significant.

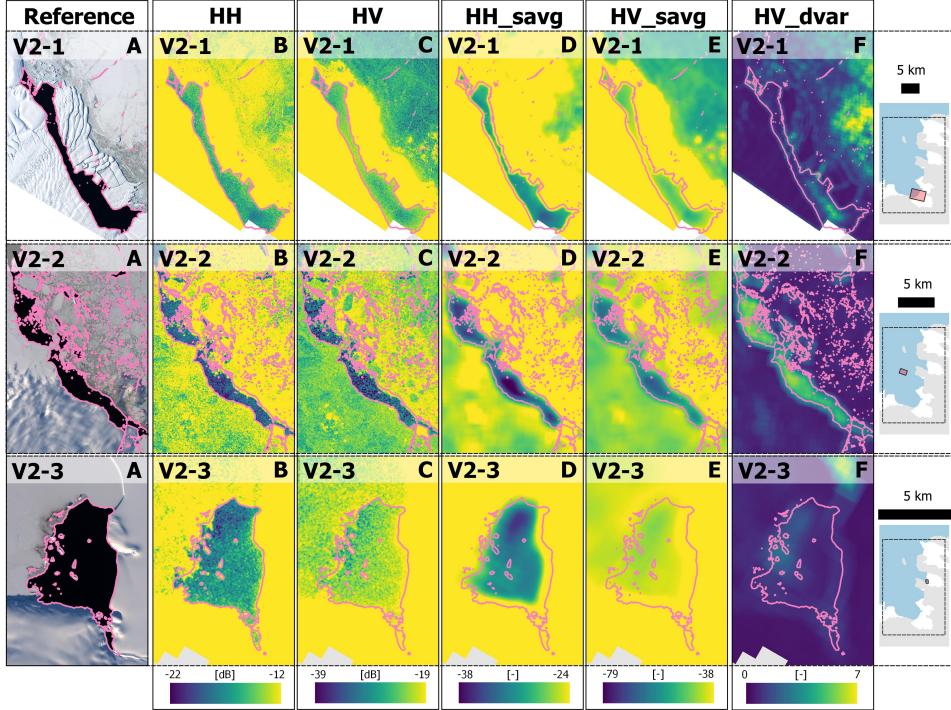


Figure 4.4: HH, HV and GLCM21 texture features zoomed in at small scale water bodies for V2. A water mask from the optical reference image is overlaid in pink for convenience.

4.3. Classifier Performance

4.3.1. Accuracy Assessment

Table 4.1 shows how each classifier performed. To assess each classifiers ability to classify polynyas correctly as water, we denote FNR, FPR and the kappa score apart from overall accuracy. A high FNR indicates a poor ability to correctly classify water, whereas a high FPR indicates an inability to detect ice. A low FNR is thus essential to detect plume polynyas, as they are isolated bodies of water surrounded by ice. The kappa score takes both ratios into account for its accuracy, but we also evaluate whether a low kappa score corresponds to the lowest FNR.

Table 4.1: Classifier performance scores and comparative performance to BASE for water-ice classification.

Classifier	Spatial		Results			Improvement from BASE [%]
	Resolution [m]	Accuracy [-]	Kappa [-]	FNR [%]	FPR [%]	
BASE	40	0.922	0.844	9.39	6.22	-
GLCM5	200	0.959	0.915	4.74	3.72	+8.46
GLCM11	440	0.963	0.927	4.49	2.83	+9.81
GLCM21	840	0.965	0.930	4.25	2.78	+10.2

Each classifier has a water-ice classification accuracy above 92%, but the kappa score of `BASE` of 84% is significantly lower than any of the kappa scores of the GLCM classifiers, which are all higher than 91%. We can see that the accuracy of `BASE` gives a false sense of high quality classification as both FNR and FPR are around 2 times higher than those of the GLCM classifiers. Therefore, each GLCM classifier is a significant improvement compared to `BASE`, also denoted by their respective improvement scores. `GLCM21` performs best, with a kappa score of 93.0% (+10.2% improvement from `BASE`). `GLCM11` has the second-highest kappa score, at 92.7% (+9.81% improvement from `BASE`). `GLCM5` scores lowest, with a kappa score of 91.5% (+8.46% improvement from `BASE`). In terms of FNR and FPR, we see a similar pattern. Of the GLCM classifiers, `GLCM21` has lowest FNR and FPR scores at 4.25% and 2.78% respectively, then `GLCM11` with 4.49% and 2.83% and lastly `GLCM5`, which has a FNR of 4.74% and a FPR of 3.72%. Kappa scores, FNRs and FPRs of `GLCM11` and `GLCM21` are almost equal, whereas those of `GLCM5` deviate relative to these classifiers.

4.3.2. Confusion Matrices

Confusion matrices provide another means to visualize classifier accuracy and portray how accurately each (sub-)class is predicted. In [Figure 4.5](#) we present the confusion matrices of each classifier per class and in [Figure 4.6](#) confusion matrices for sub-classes are given.

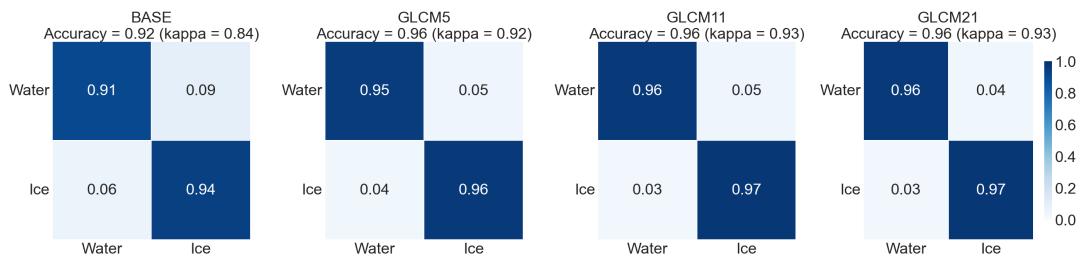


Figure 4.5: Confusion matrices of each classifier for water-ice classification.

[Figure 4.5](#) shows the same performance pattern as the values of FNR and FPR in [Table 4.1](#). `Water` and `Ice` pixels are classified correctly for >91% for each classifier, with percentages reaching 96% and 97% respectively for `GLCM11` and `GLCM21`. Comparing the GLCM classifiers, we can again see that `GLCM5` performs slightly worse than both `GLCM11` and `GLCM21`, as true positives and true negatives are marginally lower, whilst false negatives and false positives are marginally higher.

In [Figure 4.6](#) we take a closer look at each classifier to see which sub-classes prove most difficult to accurately detect. For sub-class classification, the accuracy pattern found in [Table 4.1](#) and [Figure 4.5](#) is repeated again. `BASE` performs significantly worse than the GLCM classifiers (accuracy = 0.81, kappa = 0.74), with `GLCM21` being the most accurate classifier (accuracy = 0.87, kappa = 0.83). Performance of `GLCM21` equalled by `GLCM11` (accuracy = 0.87, kappa = 0.83). Somewhat less accurate is `GLCM5` with an accuracy of 0.86 and kappa of 0.81. When looking at each confusion matrix it is clear that each classifier has most difficulty with accurately distinguishing the `Water` sub-classes. Accurate detection of `Ice` sub-classes proves more effective. Finally, as expected, each classifier experiences relatively high difficulty with accurate distinction of `Rough Water` and `Sea Ice`. Every GLCM classifier shows the same performance in incorrectly classifying `Rough Water` as `Sea Ice` (False Negatives), each having omission errors of 10%. Although substantial in magnitude, omission errors for these sub-classes are significantly reduced compared to `BASE` (omission error = 15%). For the incorrect classification of `Sea Ice` as `Rough Water` (False Positives), `GLCM5` is marginally less accurate (omission error = 8%) compared to both `GLCM11` and `GLCM21` (omission error = 5%).

4.4. Window Size Analysis

In [Figure 4.7](#) we compare the effect of window size choices to the classification outputs on all four validation images. [Figure 4.7](#), [4.8a](#) and [4.8b](#) show that every classifier, `BASE` as well, is able to distinguish very large bodies of water from ice regions. However, `BASE` classification also leads to very high number of False Positives and False Negatives within classified regions of `Water` and `Ice`. Large portions of sea ice show regions of classified water bodies, seen in [Figure 4.7V1-4](#). Similarly, the figure shows

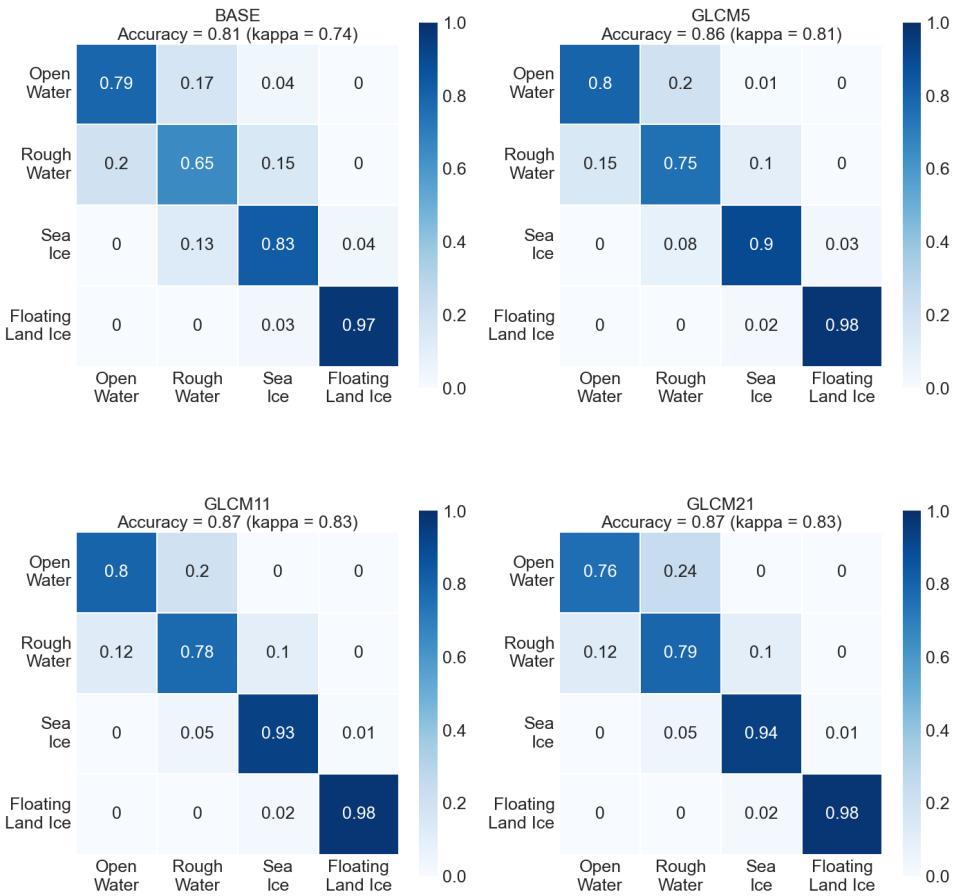


Figure 4.6: Confusion matrices of each classifier for sub-class classification.

a multitude of smaller classified ice bodies within the larger open water regions. Compared to BASE output, all the GLCM classifiers perform significantly better. Each classifier is able to pick up bodies of water whilst keeping the FPR and FNR substantially lower.

For an in-depth comparison of each GLCM classifier, we zoom in to examples of open water ([Figure 4.8a](#)) and sea ice ([4.8b](#)) respectively to better illustrate the performance of the different GLCM classifiers. When looking at [Figure 4.8a](#), we notice that GLCM21 performs relatively poorly in classifying small bodies of water, as the classifier misses water near edges of open water bodies. GLCM5 and GLCM11 perform better at detecting the edge of water bodies and can detect smaller bodies of water. In [Figure 4.8a](#) (row A), near edges and in the smallest water leads, we primarily see yellow masks, meaning that these regions of water are only picked up by GLCM5 (most clearly visualized in [Figure 4.8aV4A](#)). Slightly further away from the sea ice edges, we primarily see green masks, meaning that at this distance GLCM11 also detects water (e.g. [Figure 4.8aV2-2A](#), [4.8aV2-3A](#) and [4.8bV1-2A](#)). Only when we are significantly farther away from a sea ice edge, the mask is purple, meaning the GLCM21 also detects water ([Figure 4.8aA](#)). From visual interpretation, GLCM5 therefore clearly has the best performance for this specific aspect of classification, which should be reflected in a low FNR. However, the FNR does still decrease from GLCM5 (FNR = 4.74%) to GLCM21 (FNR = 4.25%), even though the difference is relatively minor (-0.25% and -0.49% respectively). The FNR therefore cannot be used on its own as a reliable assessment of a classifier's ability to detect water.

Conversely, GLCM5 performs poorest at classifying sea ice correctly, resulting in a high number of pixels falsely detected as open water (higher FPR). The FPR increases significantly for GLCM5 (FPR = 3.72%) compared to both GLCM11 (FPR = 2.83%) and GLCM21 (FPR = 2.78%). [Figure 4.8b](#) shows that GLCM21 indeed leads to the least amount False Positives pixels. In [Figure 4.8bV2](#) and [4.8bV4](#) differences in sea ice detection are clear. Comparing each classifier in [4.8bV2-A](#) and [4.8bV4-A](#) shows that GLCM5 has a much higher FPR for sea ice regions than GLCM21 (large portions of sea ice are

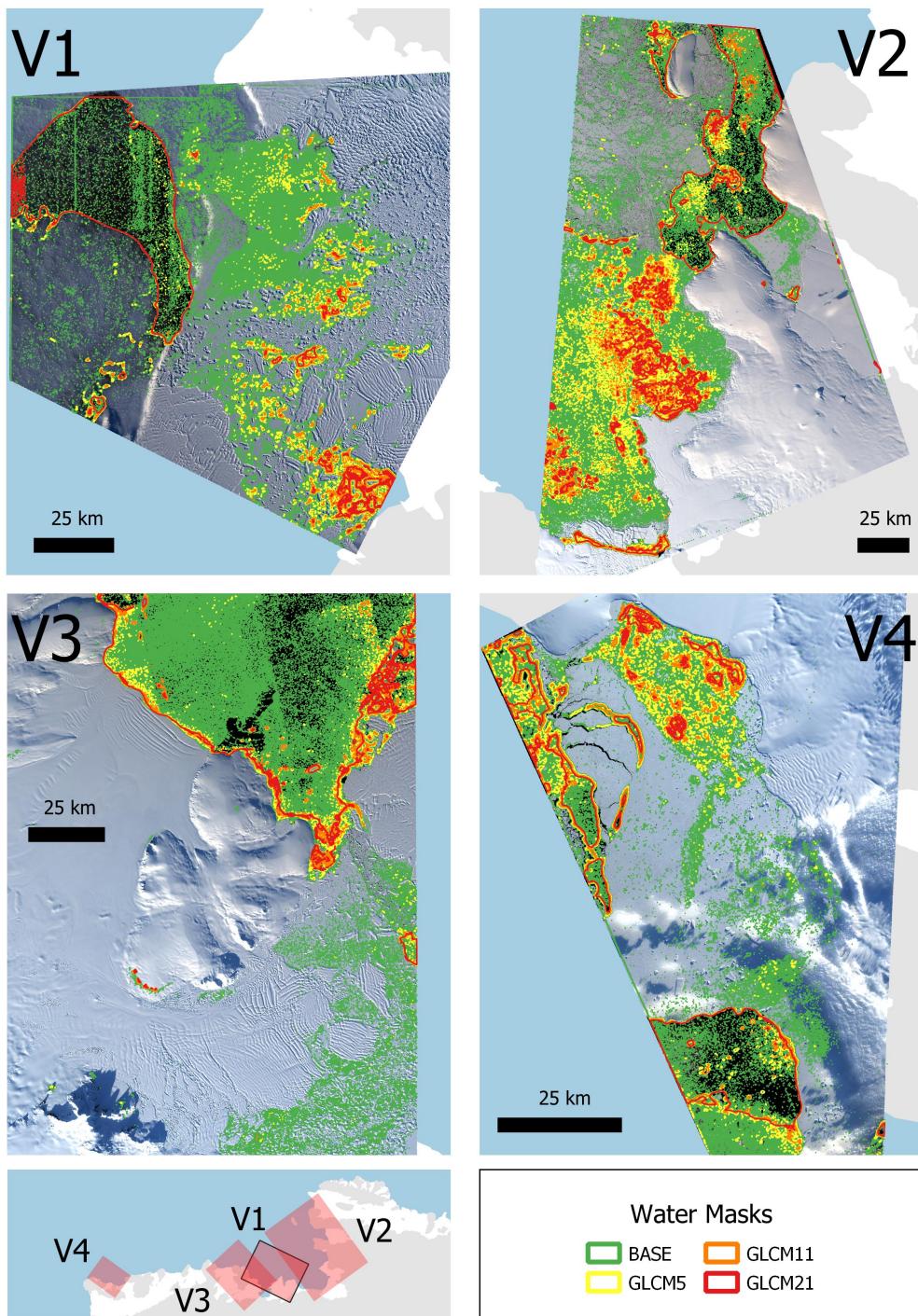


Figure 4.7: Window size influence on water-ice classification. On top of an optical reference image, classification outputs of each classifier are presented as water outlines. BASE: Green, GLCM5: Yellow, GLCM11: Orange, GLCM21: Red.

overlaid by the yellow mask). GLCM11 shows substantially less False Positives than GLCM5 (orange and green masks), but shows some extra False Positive regions compared to GLCM21 (red and purple masks). In terms of sea ice classification, FNR values presented in [Table 4.1](#) are in accordance with spatial patterns found in [Figure 4.8b](#).

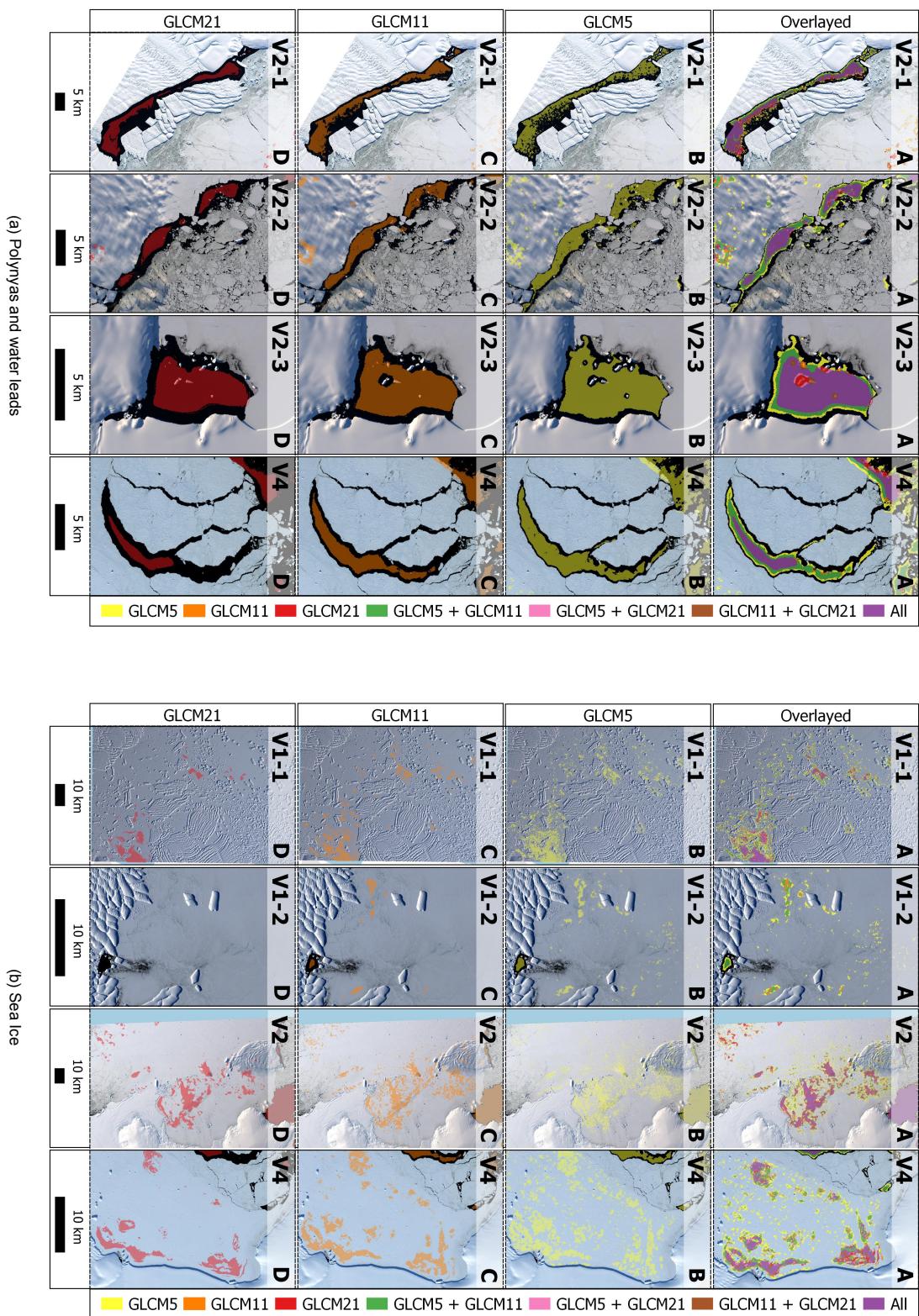


Figure 4.8: Classification output for different window sizes, zoomed in at regions of interest.

4.5. Post-Processing

[Figure 4.5](#), [Figure 4.8a](#) and [Figure 4.8b](#) have shown that increasing GLCM window size leads to a significantly lower FPR, but leads to poorer performance in accurate small water body and water edge detection. Ideally, we merge the ability of `GLCM21` in minimizing FPR with the ability of `GLCM5` to detect water bodies up to small spatial scales. We try to achieve this by applying an area filter on classified bodies of water.

To analyse the influence of filtering the classified open water bodies by a minimum area of 1.2km^2 (750 pixels of $40\times 40\text{m}$), we look at classified water bodies in regions of sea ice and regions where polynyas and smaller water leads occur. We visualize water bodies that are included and excluded by this filter for each classifier in [Figure 4.9a](#) and [4.9b](#). For a complete overview of the filtering step on each validation image we have visualized this filter for each classifier in [Figure A.8a](#), [A.8b](#) and [A.8c](#).

Looking at [Figure 4.9b](#), we see very little effect of filtering on `GLCM21` classification outputs. For `GLCM21` not many incorrectly classified bodies of water are smaller than 1.2km^2 , meaning that few of these water bodies are filtered out. Only in [Figure 4.9bV4-C](#) a significant positive effect is visible on removing False Negative pixels. The effect for `GLCM11` is more pronounced. We see a positive effect in [Figure 4.9bV4-B](#), where a large fraction of False Positives are filtered out. However, the negative effect of this filter step is visible in [Figure 4.9bV1-2B](#). When applying this filtering step, the output of `GLCM11` does not contain a small scaled polynya anymore. When filtering the output of `GLCM5`, we again see that a significant amount of False Positive pixels are filtered from the results in [Figure 4.9bV4-A](#). But, even after filtering, the amount of False Positives is significantly higher than `GLCM11` and `GLCM21` especially, although the False Positives are now mostly found in the same regions of sea ice, where they were first much more spread out. Finally, the filtering step does not filter out the small polynya from the `GLCM5` result, seen in [Figure 4.9bV1-2A](#), meaning that filtering `GLCM5` classification outputs does not impact its ability to detect water bodies below 1.2km^2 as significantly as for `GLCM11` and `GLCM21`.

When comparing effects of filtering on classification outputs near sea ice regions in [Figure 4.9a](#), we see similar results. In [4.9aV2-3C](#) and [4.9aV4C](#) no bodies of water are filtered from the `GLCM21` outputs, whereas in [4.9aV2-1C](#) and [4.9aV2-2C](#) the number of removed bodies of water is minimal (roughly 3-5). We do however see that those bodies of water that are removed, correspond to relatively large water leads or polynyas. The fact that only a small portion of such water bodies is detected by `GLCM21` is due to the reduced spatial resolution by choosing a larger window size, as described before. We see that `GLCM21` already has difficulty in detecting smaller scale water features (especially in [4.9aV4C](#)), and applying this filter step slightly increases the already relatively poor performance of the classifier. The effect of applying the filter step on `GLCM11` images is slightly more pronounced, but still marginal. A couple of water bodies in [4.9aV2-1B](#), [4.9aV2-2B](#) and [4.9aV4B](#) are removed, but they do not influence the extent of classified water bodies significantly. As `GLCM11` has a higher spatial resolution, it is able to better detect water edges and smaller wakes than `GLCM21`. As such, single water bodies are not detected as multiple bodies of water ([4.9aV2-2C](#) compared to [4.9aV2-2B](#)) and filtering does not take place. Finally, `GLCM5` shows the most filtering of small water bodies, as it is able to detect the smallest scale water bodies. Especially in [4.9aV4A](#) we see that two smaller water leads that are correctly classified as water are removed with this filtering step. Still, despite filtering such bodies of water, `GLCM5` provides more accurate water body detection than `GLCM11` and in particular compared to `GLCM21`.

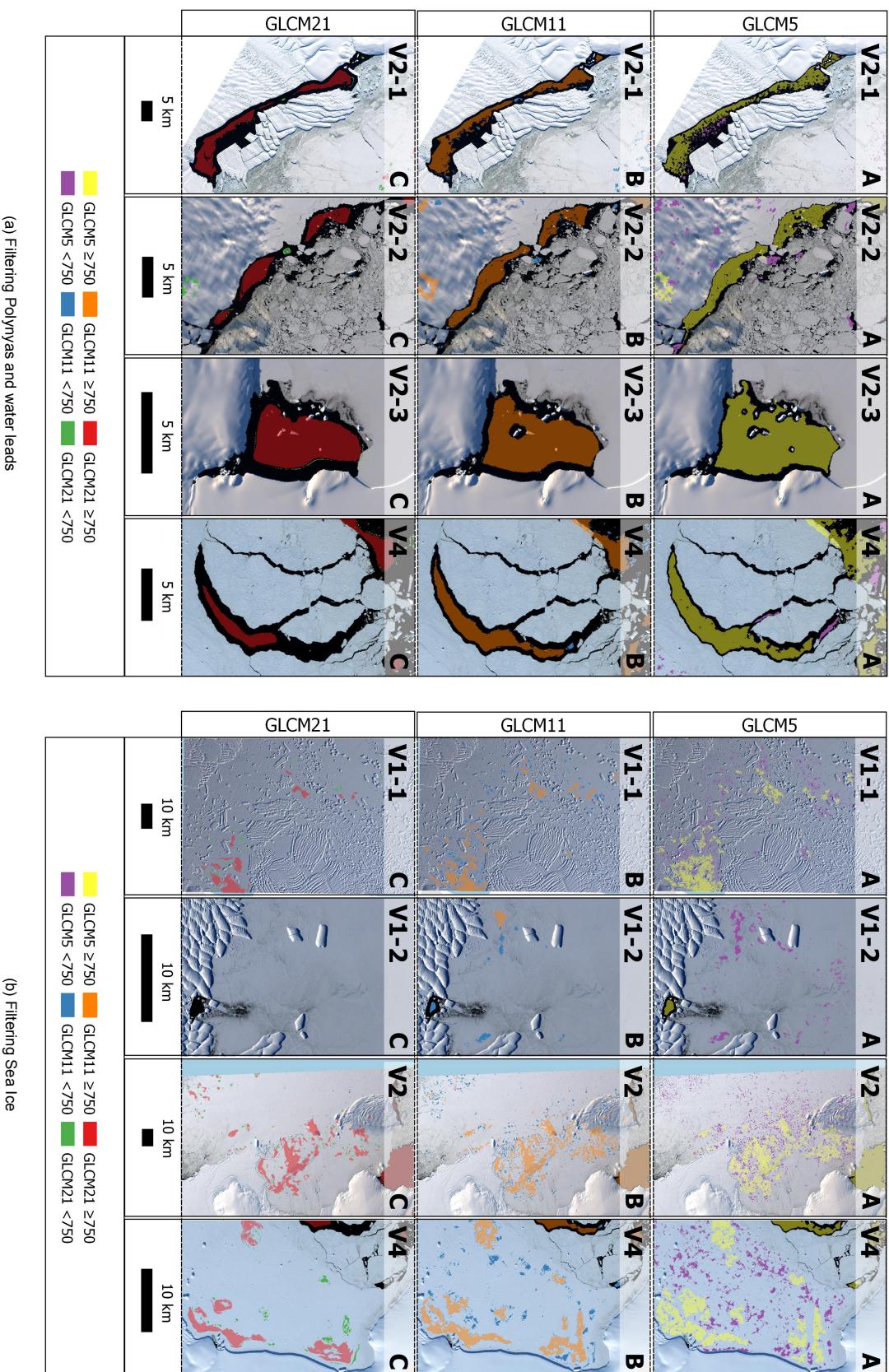


Figure 4.9: Overview of impacts of filtering smaller classified bodies of water (750 pixels) from the classification output

5

Discussion

5.1. Classification

In both [Table 4.1](#) and [Figure 4.7](#) we compared the ability of each classifier to detect open water bodies. We could see that `BASE` performs significantly poorer than any GLCM classifier, as its kappa score is significantly lower, and both its FNR and FPR are higher by roughly factor two to three. [Figure 4.7](#) has shown the effect of this lower accuracy in a spatial context. The water classification of `BASE` is much less uniform compared to `GLCM5`, `GLCM11` and `GLCM21`. Large numbers of False Negative pixels are visible even in greater bodies of water, indicating that `BASE` is significantly less robust in its ability to distinguish open water regions from sea ice. False Positive pixels are much more prevalent across sea ice regions in all validation images, showing a similar reduced robustness for accurate distinction of sea ice compared to rough water. Overall, `BASE` results in a noisy water mask, while sea ice is detected much too inaccurately to be able to use this classifier effectively. Our results confirm that `HH` and `HV` alone are ill-suited for accurate water-ice classification and plume-driven polynya detection.

[Figure 4.2](#) (plus [Figure A.6](#) and [Figure A.7](#)) have shown us that, after filtering out features that are correlated for more than 90% to each other, `HH_savg`, `HV_savg` and `HV_dvar` are the unique features that have the largest potential for accurate water body detection based on their importance scores. [Figure 4.2](#) and [Figure 4.3](#) respectively show the added value of each feature from a data perspective and from a spatial context. In [Figure 4.2](#) we saw that the importance of `HH_savg` and `HV_savg` is always higher than that of `HH` and `HV`. These importance scores are likely attributed to the fact that textural features pick up spatial features in the GLCM window: `HH_savg` and `HV_savg` (and `HV_dvar`) make use of extra information compared to the original backscatter intensity features `HH` and `HV`. This, however, is at the expense of spatial resolution. Optionally, instead of three, the six 'best' GLCM texture features (see [Figure 4.1](#)) could have been used to, in this case, base `GLCM11` on. We argue that this would have likely only improved classification accuracy marginally, as removed texture features do not provide a significant amount of extra information due to their high correlation with `HH_savg` or `HV_dvar`. Despite high correlations, we do however argue that the classifier would likely have been fractionally more robust, as extra information of texture features with high importance improves the training process. However, by only selecting `HH_savg`, `HV_savg` and `HV_dvar`, we were able to do a more in-depth analysis of the influence that each texture feature has on classification, through which we were able to understand the classification process more thoroughly.

In [Figure 4.3](#) we saw the effect of our choice for using the GLCM features rather than original backscatter intensity values. In regions of open water this choice aids accurate classification, as values of `HH_savg` and `HV_savg` are more uniformly low. In sea ice regions this smoothing effect can have an adverse effect if too many pixels occur within the GLCM window with a significantly low intensity. The smoothing effect in `HH_savg` and `HV_savg` can then lead to patches of sea ice with consistently and uniformly low intensity values, which can explain the larger regions of False Positives. Even though False Positives are introduced in this manner, the overall effect of using GLCM features is positive: it leads to significantly lower FNR and FPR for each GLCM classifier with respect to `BASE` ([Table 4.1](#)). A downside of using GLCM features is the smoothing or translation of sharp contrasts in `HH` and `HV` values at the water-ice shoreline (see [Figure 4.4](#)). A sharp transition from high to low (or low to high) `HH`

and HV intensities within the GLCM window, leads to a displacement of this transition in GLCM features of, at most, the width of the GLCM window. This impacts classification accuracy negatively, as original backscatter intensity transitions will always be adjusted. In some cases, the water-ice boundary will be extended outwards, increasing the classified water body area, while in others, the boundary was translated into water bodies, thereby decreasing its classified area. When *Floating Land Ice* borders *Water*, as is often the case for plume-driven polynyas, very high GLCM feature intensities (HH_savg and HV_savg) for *Floating Land Ice* are directly adjacent to low *Water* feature intensities. During GLCM computation, this results in a translation of the original water-ice boundary already when only a few *Floating Land Ice* pixels are present within the GLCM window (as seen by an extension of high intensity values roughly 21 pixels (2040m) wide in 4.4D&E) within the body of water. Consequently, using GLCM features introduces a bias towards the detection of sea ice in such regions.

As mentioned before, classification results are heavily dependent on training data. Sampling is therefore a crucial aspect of this research. Given the nature of this research, sampling was a random selection process of S1 input data, with labels being assigned manually. The training process could have been improved by having in-situ measurements of the surface for each (sub-)class. As this was not the case, we believe that, in this way, errors were introduced in the classification output.

As the added value of creating a GLCM classifier using HH_savg , HV_savg and HV_dvar as input features has been made apparent, the most important variable to assess next is the GLCM window size w . Table 4.1 shows a correlation between combined kappa, FNR and FPR scores and window size. By decreasing spatial resolution and increasing the GLCM computation window kappa increases consistently (GLCM5: 91.5%, GLCM11: 92.7%, GLCM21: 93.0%), while FNR (GLCM5: 4.74%, GLCM11: 4.49%, GLCM21: 4.25%) and FPR (GLCM5: 3.72%, GLCM11: 2.83%, GLCM21: 2.78%) decrease consistently. Using a window size of $w=5$ results in an improvement score of 8.46%. Roughly doubling the window size $w=11$ increases the improvement score to 9.81% (+1.35%), which is considerable. Quadrupling the window size to $w=21$ leads to a marginal extra increase of the improvement score (+10.2%, which is +1.74% compared to GLCM5 and +0.39% compared to GLCM11). Increasing w is therefore beneficial for accurate classification, but as the increase in accuracy between GLCM11 and GLCM21 is marginal, we argue that increasing $w>11$ is not worth the extra decrease in spatial resolution (440m to 840m). To confirm this, we refer to the spatial context of the classification results in Figure 4.8a and 4.8b.

From spatial interpretation, increasing window size shows a clear increased performance in the detection of open water. GLCM5 performs best at detection small scale water bodies, GLCM11 and GLCM21 both perform worse. GLCM21 has most difficulty with detecting the full extent of small scaled water bodies. Significant water sections are not detected by the classifier, most prevalent for water leads with sub-kilometer widths (Figure 4.8aD). This effect is repeated for GLCM11 (4.8aC) and GLCM5 (4.8aB), but at a much lesser extent than for GLCM21. While GLCM5 clearly outperforms GLCM21 in terms of minimizing False Negatives for small water bodies, this is not reflected in Table 4.1 (GLCM5: FNR = 3.72%, GLCM21: FNR = 2.78%). Furthermore, looking at Figure 4.7, we see no significant difference between the two classifiers in False Negatives within large bodies of water (as is the case for BASE). This shows that the overall FNR computation in Table 4.1 is somewhat biased towards large open water bodies and does not accurately represent the behaviour we see in Figure 4.8a. This bias is most likely introduced due to the random selection of sample points. Perhaps by chance, significantly less water pixels are sampled near the ice-water edge and within sub-kilometer scaled water leads, meaning that incorrect detection is not picked up.

With respect to sea ice detection, we see the opposite correlation between w and detection performance. By increasing window size, GLCM11 and GLCM21 are much better at classifying large sections of sea ice correctly than GLCM5, as the standard deviation of HH_savg , HV_savg and HV_dvar intensities are significantly lower than those of HH and HV for both water and ice regions. This is reflected in the fact that considerable portions of Figure 4.8bA are masked in yellow (GLCM5 outputs). Most of these classified bodies of water are small in size. They are detected as water as the window size is not large enough to smooth out high intensity values from (predominantly) low intensity values. Similar patterns are visible for GLCM11 by looking at green and orange masks (4.8bA). Note that for the output of this classifier, areas of False Positives are larger than those of GLCM5, due to a larger window size. Exactly due to this effect, GLCM21 detects the least amount of water bodies on sea ice, but the water bodies that are detected are considerably large in size (4.8bA, red and purple masks). We thus note that, although significantly better than BASE, every GLCM classifier shows an inability to classify several (considerably large) sections of sea ice in Figure 4.8bV1-1, 4.8bV2 and Figure 4.8bV4 correctly.

We thus argue that this inherent disadvantage to using dual-pol SAR data for water-ice classification cannot be overcome by using GLCM texture based classifiers alone.

Overall, a pattern emerges. Increasing window size leads to lower overall FNR and FPR, at the cost of spatial resolution. What spatial analysis clearly shows though, is that this reduction in spatial resolution has significant impacts on what type of water body a classifier is able to detect. Applying a window size w for GLCM computation leads to an inability to detect water edges and bodies of water at spatial scales that are smaller than this reduced spatial resolution. `GLCM21` performs best in identifying sea ice, showing minimal False Negatives. `GLCM11` performs comparable to `GLCM21` on this front. However, its spatial resolution is twice as high, meaning that simultaneously, it is better able to detect small scale water bodies. Thus, `GLCM11` outperforms `GLCM21` in open water detection. `GLCM5` results in the most accurate open water detection. With a spatial resolution of only 200m this classifier can detect significantly smaller water leads and can detect the water-ice edge of polynyas much more accurately than both `GLCM11` and `GLCM21` in particular. `GLCM5` performs however extremely poorly in sea ice detection, introducing significant numbers of False Positives in the classification output. Its FPR is significantly higher than that of `GLCM11` and `GLCM21` and is so high that despite its accurate open water detection, it is overall ill-suited for plume-driven polynya detection in its current form. The same holds for `GLCM21` due to its low spatial resolution. As such, we argue that `GLCM11` is best suited for accurate plume-driven polynya detection, based purely on classifier performance.

5.2. Post-Processing

In chapter 4 we show effects of filtering each classifier's output on the minimum water body size of 1.2km² (750 pixels of 40x40m) and assess whether False Positives can be substantially and accurately reduced for any of the classifiers, while minimizing the amount of True Positives that are removed. Filtering particularly affects the `GLCM21` output in a negative sense, as smallest regions of correctly detected water (True Positives) in its output are removed. For `GLCM21` this removal is a substantial impact as `GLCM21` already has difficulty with detecting smaller water bodies. For `GLCM5`, sections of classified water are filtered out as well. However, these detected water bodies are of a much smaller scale than those of `GLCM21` and we therefore argue that these True Positive removals are acceptable losses. From the output of `GLCM11` fewer True Positives are removed compared to `GLCM5` and `GLCM21`, as almost all of the classified water bodies in each region are larger than 1.2km². The exception to this is the small (possibly plume-driven) polynya in 4.9bV1-2. When filtering the `GLCM11` output, detection of this polynya is removed, while in filtered `GLCM5` results this polynya is preserved.

Over sea ice regions, filtering small scale water bodies shows a considerable positive impact. Here, False Positives in classification outputs of every classifier are present and are removed systematically by the filter. The effect is most prevalent when filtering `GLCM5` output, as lower window sizes result in higher spatial resolution of the classification result, leading to many False Positive regions which are smaller than 1.2km² (4.9bV4-A). A significant amount of False Positive regions are removed in `GLCM11` as well, although the increased window size of classification results in a lower impact of this filter on the amount of False Positives removed (4.9bV4-B). Some False Positives are removed from `GLCM21` results, but the effect is far less significant than for `GLCM5` in particular, as the use of a larger window size already smoothed out most small scale features (<1.2km²) during classification. In every filtered classification result, multiple regions of False Positives of considerable size still remain, which, irrespective of window size, cannot be removed with an area filter.

Overall, we suggest to avoid filtering `GLCM21` results to prevent True Positive removal from classification output that already suffers from poorer water detection. We suggest to filter both `GLCM11` and `GLCM5` outputs for areas over 1.2km² as large amounts of False Positive pixels are removed from sea ice regions, whilst impact on water detection is considered acceptable. For both classifiers, open water detection is comparable to the original classification outputs and is an improvement compared to original `GLCM21` results. Choosing to filter `GLCM11` outputs means we accept the inability to detect the smallest polynyas, as seen in Figure 4.9bV1-2B. Open water detection in other regions is still sufficient (Figure 4.9aV2-2B and 4.9aV4B), while 4.9bV4B shows that sea ice detection of `GLCM11` is improved significantly in this way.

Finally, the filtered classification results are summarized in Figure 5.1a and 5.1b to analyze which detection method of each classifier is preferred. For clarity, we name the filtered outputs of `GLCM5` and `GLCM11` `GLCM5F` and `GLCM11F` respectively. First, we compare each method in regions where

polynyas and water leads are present ([Figure 5.1a](#)). It is evident that GLCM21 performs very poorly at water edge detection and at thin water lead detection. This is made even more apparent when comparing GLCM21 results directly to GLMC11F and GLCM5F in [5.1aV2-3A](#) and [5.1aV4A](#). Here we see that GLCM21 is outperformed by both GLCM11F and GLCM5F as the purple mask is significantly smaller and less accurate than the green and yellow masks of GLCM5F and GLCM11F respectively. GLCM5F outperforms GLCM11F ([5.1aV4A](#), yellow versus green), but not as considerable as it outperforms GLCM21 results ([5.1aV4A](#), yellow and green versus red and purple masks). GLCM5F and GLCM11F results are considered to be roughly equal in terms of small scale open water detection accuracy. Second, we compare detection methods in [Figure 5.1b](#) on their ability to accurately detect sea ice, thus minimizing False Positives. Here, we see that regions of False Positives ([5.1bV1-1](#), [5.1bV2](#), [5.1bV4](#)) are confined to roughly equal locations for every detection method (purple mask). Upon further inspection, these figures also show that GLCM5F, despite an impressive reduction in False Positives, is still most prone to detect sea ice incorrectly as substantial parts of these regions are only detected via this method (yellow mask). We argue that, even though the inherent draw-back of SAR-based classification for water-ice classification is not completely overcome, applying a water body area filter is an effective manner to increase polynya detection performance considerably for GLCM-based classifiers with window sizes up to $w = 11$. Given that GLCM21 output underperforms compared to GLCM11F and GLCM5F for open water detection, and that GLCM5F underperforms compared to GLCM21 and GLCM11F, the proposed method to classify open water bodies for plume-driven polynya detection is to use GLCM11F: classify dual-pol SAR data using GLCM11, then post-process classification results with an water-area filter ($>1.2\text{km}^2$).

5.3. Comparison to Other Detection Methods

The method presented in this study is able to identify small scaled water bodies ($<440\text{m}$) with high accuracy for both water-ice classification ($\kappa = 93\%$) and water-ice sub-class classification ($\kappa = 83\%$). Overall (sub-classification) accuracy is higher than the automated SAR-based river ice detection method presented by [de Roda Husman et al. \[2021\]](#) ($\kappa = 80\%$), despite choosing the same window size ($w=11$). Comparing overall accuracy to results from other studies, such as [[Hollands and Dierking, 2016](#)] and [[Karvonen and Hallikainen, 2009](#)] is more difficult as the authors do not present accuracy metrics. Given that our results are comparable to that of [de Roda Husman et al. \[2021\]](#), we do however believe that the accuracy of our results are on par with results from [Hollands and Dierking \[2016\]](#) and [Karvonen and Hallikainen \[2009\]](#). It should however be noted, that each of these studies have made more detailed water-ice detection algorithms than our (sub-class) classification algorithm. [Hollands and Dierking \[2016\]](#) have used a decision tree to classify 8 types of ice besides a water class. They however used both L-band and C-band SAR data, as well as brightness temperatures from passive microwave radar imagery to achieve this. As such, temporal resolution of input data is much lower compared to our input data, as images from all three data sets are needed for a classified image. As we want to keep temporal resolution as high as possible, the use of other data sets was omitted, meaning that ability to detect the amount of ice sub-classes presented by [Hollands and Dierking \[2016\]](#) was diminished. [Karvonen and Hallikainen \[2009\]](#) based their detection algorithm on edge-features alone. As expected, their results show high performance for edge detection and show a good ability to classify thin leads of ice. For the scope of this research we only looked at GLCM-based features, and comparing them to those of [Karvonen and Hallikainen \[2009\]](#), we argue that results of GLCM5 (a window size of 5×5 pixels of $40\times 40\text{m}$) are comparable to their results, although the method by [Karvonen and Hallikainen \[2009\]](#) does detect the thinnest wate leads more easily. Our method loses performance compared to [Karvonen and Hallikainen \[2009\]](#) when increasing GLCM window size to $w=11$ and above . It could therefore be worthwhile to add edge features to the feature selection procedure in a future study to see whether such features are of added value for mitigating false edge detection ([chapter 5](#)).

When comparing classification results of [de Roda Husman et al. \[2021\]](#) to those in this study, we believe that there are two reasons for the improved classification performance. For one, the difference in performance is due to a different choice of classification classes. While class choices are comparable in both studies ([de Roda Husman et al. \[2021\]](#) classify SAR images as *Open Water*, *Rouble Ice* and *Sheet Ice*), the ice sub-classes in this study are more different from a physical standpoint than *Rubble Ice* and *Sheet Ice*, thereby making it easier to classify images correctly. As the main objective of this study is to distinguish water from any type of ice, instead of classifying specific types of

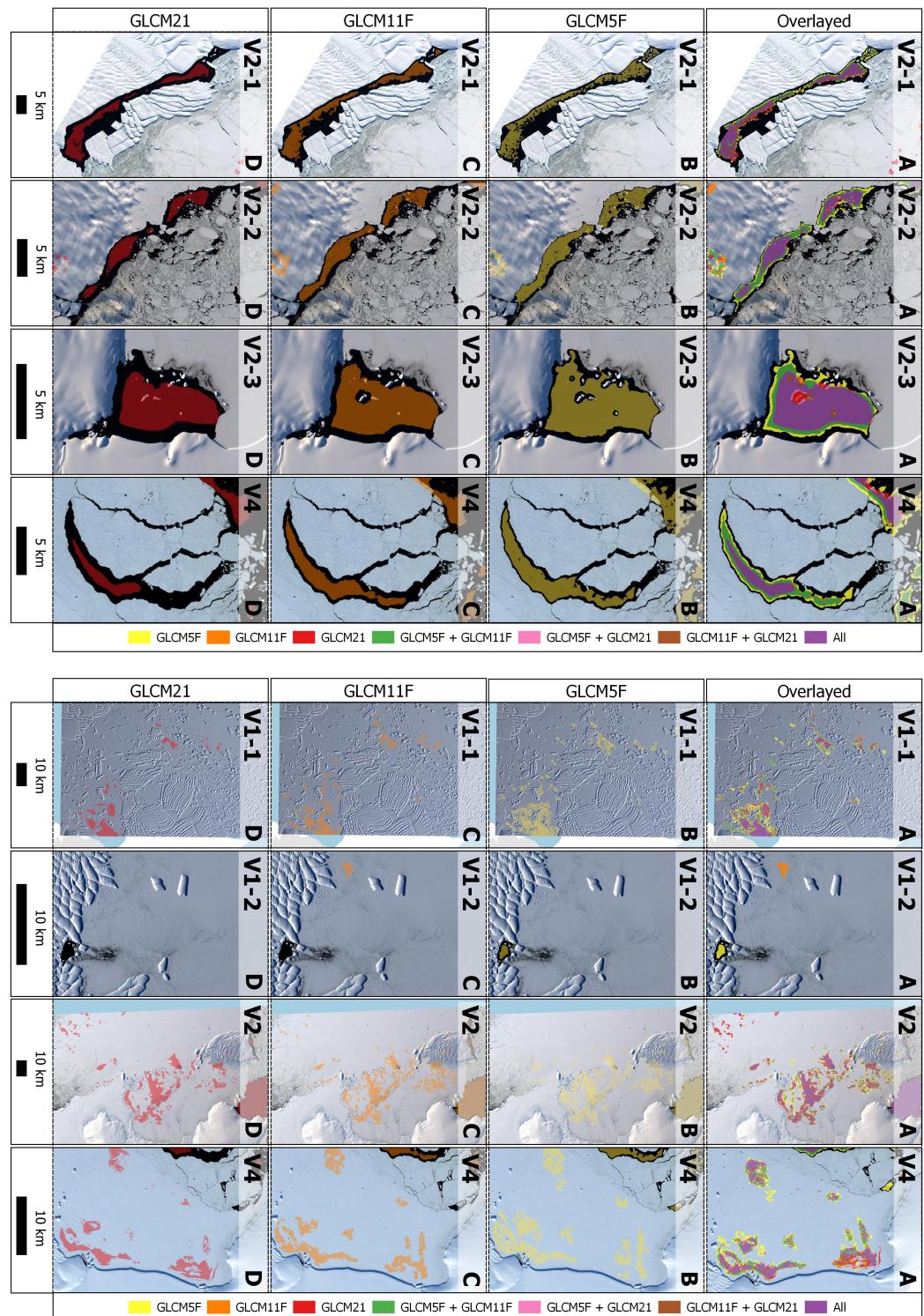


Figure 5.1: Comparison of polynya detection methods zoomed in at regions of interest.

ice correctly as well, we believe that the class choice used here is valid. Another reason for improved classification could be the slightly different choice of GLCM features. In this study we propose to base the classifier on 'sum average' (HH and HV polarization) and 'difference variance' (HV polarization, whereas [de Roda Husman et al. \[2021\]](#) proposes the use of original cross-polarization backscatter intensity (VH in their case), the co-polarization GLCM average (VV) and a polarimetric based feature (not used in this study). Evidently, feature choice between both studies is relatively similar HH_savg relates to VV GLCM average, while HV_savg , given its high correlation with HV (94.8%), is comparable to VH. [de Roda Husman et al. \[2021\]](#) used a polarimetric feature to improve water-river ice distinction, where we show that a GLCM variance feature based on cross-polarized data leads to similar results for (small-scaled) water-sea ice detection.

[Lohse et al. \[2021\]](#) have proposed another feature choice for sea ice classification. They proposed the use of original dual-pol intensities and co-polarization Dissimilarity, Contrast and Energy (features 14, 2 and 1 in [Table 2.2](#) respectively). They do not use HV-based features, stating that HV is more problematic than HH as signals are often close to the noise floor. Our results prove that that in the context of small-scaled water detection, cross-polarization data is very useful, when necessary pre-processing steps such as thermal noise correction are applied [[Lohse et al., 2021](#)]. Despite a different detection goal, namely detecting different types of sea ice instead of small-scaled water detection, the feature choices themselves are again similar to those in this study. Dissimilarity and Contrast are highly correlated to variance (99.5% and 97.0% respectively for HV) and will have similar spectral characteristics as HV_dvar . The difference in feature choice is between both studies is characterized via the use of Energy, as HV_dvar has little correlation with HV_asm (15.2%). Energy is therefore more likely a useful feature for the distinction of different ice types, than for the separation of water and sea ice.

From a spatial context, our results show that window sizes larger than $w=11$ should be avoided. This is in contrast with results presented by [Lohse et al. \[2021\]](#), who state that water-ice classification accuracy is improved by choosing a window size up to $w=51$ (for a 40x40m pixel size). In a general sense, our results are in accordance with these findings ([Table 4.1](#)). GLCM21 led to the highest overall accuracy, but the loss in spatial resolution resulted in a poorer ability to detect water features on a spatial scale of small plume-driven polynyas. [Lohse et al. \[2021\]](#) have mentioned as well that the loss of spatial resolution is a consequence for the increase in window size, although it did not have a negative impact on their results, given that spatial resolution could be relatively low for their study goal. For the accurate detection of small-scaled water bodies however, this is not the case, which is why $w=11$ is advised for plume-driven polyna detection.

One of the weaknesses of the method presented in this study is the ability to accurately detect plume-driven polynyas from other bodies of water. The amount of falsely classified small-scaled water bodies is, at the moment, too significant to be able to use an area threshold for plume-driven polynya demarcation. Studies from [Alley et al. \[2016, 2019\]](#), [Gladish et al. \[2012\]](#); [Lazeroms et al. \[2018\]](#); [Sergienko \[2013\]](#), [Shean et al. \[2019\]](#) have had, in this regard, more success, albeit for different reasons. For one, [Alley et al. \[2016, 2019\]](#) have identified plume-driven polynyas manually from MODIS data. The fact that their method is not automated and based on optical imagery means that it is more time consuming and poorer in temporal coverage than the method presented here. [Gladish et al. \[2012\]](#); [Lazeroms et al. \[2018\]](#); [Sergienko \[2013\]](#) have modelled basal melt channels directly using ocean models. The major drawback of their approach is the significant computational effort and expertise needed to compute basal melt patterns and consequently predict melt channels and possible plume-driven polynya locations. The method we present here has extremely low computational effort, given its cloud-based nature, and can be applied by any GEE user. There are other methods to predict basal melt, such as using high resolution DEMs to infer Ice Shelf bottom topography [[Shean et al., 2019](#)]. However, for an accurate prediction of basal melt channels commercial data was used with high enough spatial resolution. This means that high temporal resolution is consequently lower, given that these campaigns are costly and time consuming. For the mapping of basal melt channels, lower temporal resolution is acceptable, but for an accurate overview of spatio-temporal coverage of plume-driven polynyas this is not sufficient.

Overall, despite having poorer applicability for polynya detection than other polynya or basal melt detection methods, there is potential to make detection significantly faster, easier and more accessible than the current methods available. We therefore urge to build upon this SAR-based detection method, until results are satisfactory for a fully automated plume-driven polynya detection algorithm. Based on

our results and comparisons to other research we argue that the lack in accuracy can be overcome by adding some extra post-processing steps, or aiding the classifier with extra SAR-based information, which we will discuss further in the section below.

5.4. Recommendations

Even though `GLCM11F` achieves very high accuracy in the detection of water bodies, it does not yet provide information on what body of water would classify as a plume-driven polynya. A direct way to try and gain this knowledge from `GLCM11F` results can be to apply a second area filter, but now with an upper threshold (<25km² for instance). However, as we can see by comparing results from [Figure 5.1b](#), this will still leave many regions of False Positive in the eventual 'plume polynya' detection. This means some other form of post-processing has to be applied to infer this information more accurately from the `GLCM11F` classification output. Given the physical characteristics of plume-driven polynyas ([chapter 1](#)), it would be interesting to classify dual-pol SAR data using `GLCM11F` in a specific region of interest, such as the Pine Island Glacier Embayment, for a multi-year period. Then, we suggest to create a pixel-based water detection frequency map of the classification output to see what the spatio-temporal distribution of open water detection is like. As we expect that plume-driven polynyas are confined to end points of basal melt channels, these regions should therefore be relatively consistently detected as open water on roughly the same location, resulting in high frequency values. False Positives in sea ice are a result of a process that is much more dependent local meteorological conditions and on the local angle of incidence. This type of water detection is therefore expected to be more irregular (both spatially and temporally). We expect that these detected water bodies have much lower frequency values and could be filtered out by applying a relatively high frequency threshold. This way the detection has the potential to be truly optimized for plume-driven polynya detection.

Another proposed method to extract extra information from dual-pol SAR data to aid polynya-specific detection is by making use of a Conventional Neural Network (CNN). A CNN is specially designed to extract spatial patterns from the input imagery and application on SAR imagery has the potential to directly classify bodies of water as polynyas and water leads. As polynyas and leads are distinctly shaped and have much different spatial features than smooth sea ice (see [Figure 4.3](#)), we assume that a CNN can significantly increase classification reliability and accuracy, whilst directly focusing on polynya detection instead of open water detection.

Even though `GLCM11F` is not yet accurate enough for automated plume-driven polynya detection, the classifier can still serve a useful purpose in its current state. We recommend to evaluate this classifier as a tool to validate basal melt predictions from DEMs and ocean and ice models, as its ability to detect (small scale) bodies of water accurately is sufficient, even though it is not able to directly distinguish plume-driven polynyas from other bodies of water.

6

Conclusion

Using specific texture metrics to improve the detection of water in polar environments leads to promising results. We created a GLCM-based classifier that facilitates the detection of small scaled (1-10km²) open water bodies from dual-pol SAR imagery. The distinction between rougher water states and smoother sea ice states is significantly improved compared to a dual-pol SAR classifier, showing the added value of using GLCM features to detect plume-driven polynyas.

We tested four different classifiers on their ability to classify water and ice. The first classifier was based on the original dual-pol SAR imagery (backscatter intensity in HH and HV polarization) and was used as a base classifier to compare the other classifiers to. We computed GLCM features of various window sizes ($w=[5,11,21]$) and assessed how classifiers for each distinct GLCM feature data set performed. Then, we post-processed the GLCM-based classification outputs in an effort to remove incorrectly classified water regions and to improve the identification of plume-driven polynyas from other open water bodies. In doing so, we tried to answer the question:

What are the strengths and limitations of classifying and post-processing dual-pol SAR data and GLCM features for the detection of plume-driven polynyas?

Below, we go through each sub-question to provide an answer.

- (1) *How is open water classification affected when basing a classifier on dual-pol GLCM features, compared to classification where only dual-pol SAR data is used?*
 - (a) *Which GLCM features based on HH or HV polarized SAR data prove most useful for water detection?*

HH and HV perform reasonably well when looking at accuracy score in BASE, but lead to too many False Positives and False Negatives ([Figure 4.7](#)) for reliable use in polynya detection. The best features, chosen from a selective process based of feature importance scores and feature correlations, are `HH_savg`, `HV_savg` and `HV_dvar`. The choice of these features is independent of window size choice, having highest uncorrelated importance scores for each GLCM-based classifier. We conclude that the application of GLCM features aids in more accurately distinguishing water from ice. GLCM features pick up spatial context around an original HH and HV pixel. In this way, outliers and irregular intensity patterns are smoothed out from HH and HV data at the expense of reduced spatial resolution. Where classification output of BASE shows many False Positives and False Negatives, each of the GLCM-based classifiers are much more robust and show more spatially uniform classification outputs, in accordance with optical reference data. The added value of `HH_savg` and `HV_savg` compared to HH and HV lies in this smoothing effect, thereby reducing the classifier's sensitivity to noise. When looking into feature intensity distributions per (sub-)class, we conclude that `HH_savg` is a useful metric to separate *Rough Water* from *Sea Ice* as overlap between the two subclass is small ([Figure 4.2](#)). `HV_savg` shows a poorer ability to distinguish these sub-classes. For this, `HV_dvar` provides additional useful information, as [Figure 4.2](#) shows that *Water/Ice* distinction is aided by including this feature. Overlap between the (*Rough*) *Water* and (*Sea*) *Ice* is minimal and we conclude that classification without `HV_dvar` leads to less accurate water body detection (see also [Figure 4.7](#)).

(b) *How does the GLCM window size affect classification performance of water bodies?*

Every GLCM-based classifier shows significant improvements compared to *BASE*, where increasing w leads to higher classification accuracy (*BASE*: kappa = 84.4%, *GLCM5*: kappa = 91.5%, *GLCM11*: kappa = 92.7%, *GLCM21*: kappa = 93.0%). FNR and FPR of *BASE* are a factor 2-3 higher than for the GLCM-based classifiers. For the GLCM-based classifiers, FNR decreases consistently with increased w (*GLCM5*: FNR = 4.74%, *GLCM11*: FNR = 4.49%, *GLCM21*: FNR = 4.25%), while showing equal omission errors for classification of *Rough Water* as *Sea Ice* (10%). The FPR of *GLCM5* is higher than *GLCM11* and *GLCM21* which show comparable results (*GLCM5*: FPR = 3.72%, *GLCM11*: kappa = 2.83%, *GLCM21*: kappa = 2.78%), which arises from a lower ability of *GLCM5* compared to *GLCM11* and *GLCM21* to detect *Sea Ice* as *Rough Water* (omission errors are 8%, 5% and 5% respectively). Increasing window size to $w = 21$ is thus optimal from a data analysis standpoint, with $w = 11$ showing nearly equal performance. The choice of $w = 21$ however leads to a significant decrease in spatial resolution: from 40m to 840m. For this reason, spatial interpretation shows that influence of window size is in reality more subtle. While increasing w increases overall accuracy, it also reduces spatial resolution of the classifier. Because of this, the ability of *GLCM21* to detect smaller scaled (<1.2km²) bodies of water is significantly poorer than *BASE* or *GLCM5*. Besides an inability to detect such small scaled spatial features, the application of GLCM-based features also introduces reduced performance in water-ice-edge detection. We conclude that every GLCM-based classifier misinterprets such water edges and introduces so-called False Ice Edge detection. The thickness of these False Ice Edges is directly proportional to the classifier's window size w . In this view, *GLCM5* significantly outperforms *GLCM21* due to its roughly four times higher spatial resolution. As expected, *GLCM11* shows performance that is in between both these classifiers. We conclude that the (small scaled) open water body detection of *GLCM5* and *GLCM11* is sufficient for polynya detection, but consider *GLCM21* performance inadequate. Conversely, for accurate classification of (*Sea*) *Ice*, increasing w reduces the number of False Positives in the classification output. When comparing *GLCM5* to *GLCM21*, results show that *GLCM5* leads to significantly more False Positives than *GLCM21*. Reducing spatial resolution in essence results in a filtering step during classification of smaller regions of sea ice that would otherwise have been detected as *Water*. *GLCM21* is able to minimize False Positives considerably, but for this we consider the performance *GLCM5* to be inadequate. *GLCM11* classification results in a significant amount of False Positives as well. The extent is considerably less than the *GLCM5* output, but we conclude that *GLCM11* is very much prone to misinterpret *Sea Ice* as *Water*.

The size of w therefore is a balance between minimizing False Positives (high w) and minimizing False Negatives (low w). Based on comparisons of open water detection and sea ice detection we conclude that, without any forms of post-processing, a window size of $w = 11$ is the best choice for (small scale) open water detection.

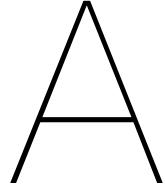
(2) *In what way do post-processing steps improve plume-driven polynya detection from ice-water classified images?*

In this study, we have assessed the use of post-processing classification outputs by applying an area filter on classified bodies of water (>1.2km²). The threshold of 1.2km² was chosen based on minimum areas of plume-driven polynya observations. This additional filter step proved particularly useful for *GLCM5*, as most classified water bodies below this size were False Positives in sea ice regions and removal of True Positives was minimal. The filter step also proved beneficial for *GLCM11*, as many False Positives on sea ice were below this threshold as well. However, filtering did negatively impact the ability of *GLCM11* to detect the smallest of water bodies (~ 1-2km²). Filtering *GLCM21* did not prove to be particularly useful, as most classified bodies of water for this classifier were already larger than 1.2km². Filtering for bodies smaller than this threshold removes some True Positives from the classification output, which is already not a strong point of this classifier. In the end, we conclude that post-processing *GLCM5* and *GLCM11* in this manner is beneficial for plume-driven polynya detection (called *GLCM5F* and *GLCM11F* respectively). For *GLCM21*, polynya detection is not substantially improved and so we conclude that filtering *GLCM21* is not advised. Even though significant numbers of False Positives are removed from the original classification outputs in *GLCM5F* and *GLCM11F*, we still encounter too many False Positives in the filtered result to automatically detect plume-driven polynyas. Both detection methods have their own strengths and weaknesses. *GLCM5F* proves useful for highly accurate polynya detection but, while minimized, False Positives are prevalent. *GLCM11F* minimizes

these False Positives much more efficiently, while being able to classify smaller scaled polynyas, although not down to less than roughly 2km². Both methods have the potential to detect plume-driven polynyas with even higher accuracy in an automatic fashion. However, extra post-processing steps are needed to process these results further in order to achieve this goal. Applying an upper limit could help in distinguishing plume polynyas from larger other water bodies, but this process is still prone to leave in False Positives.

We believe that an extra post-processing step on filtered results via a temporal analysis can help to further filter out these False Positives. Plume-driven polynyas are expected to be spatially confined to end points of basal melt channels, expectedly leading to relatively consistent detections of water in such regions. False Positives in sea ice regions are the result surface characteristics that are more dependent on local meteorological and seasonal conditions, expectedly leading to significantly less temporally consistent detection of water. In this way, a temporal filter can be applied to remove False Positives from classification outputs.

To conclude, our results have shown that using texture based dual-pol classifiers improves classification significantly compared to dual-pol only classifiers. A classifier based on GLCM features with a window size w of 11x11 pixels (440x440m) proved to be most successful in accurate water body detection and sea ice detection. Using an area filter as a post-processing method to improve plume-driven polynya detection accuracy proved successful when a classifier is based on GLCM data with window sizes no larger than 11x11 pixels (440x440m). Although successful in improving polynya detection, False Positives are still prevalent in the filtered classification outputs and additional post-processing steps, such as a temporal analysis, are advised for automatic and unsupervised plume-driven polynya detection, in order to overcome the inherent difficulty of using SAR data for water-ice classification. Nevertheless, in its current form, this method could be used to validate predicted locations of basal melt.



Tables & Figures

Table A.1: Image IDs of each Sentinel-1 and LandSat-8 match.

Match ID	Data Set	Image ID
T1	Sentinel-1	1_S1A_EW_GRDM_1SDH_20171227T045224_20171227T045328_019883_021D66_029C
	LandSat-8	1_LC08_161131_20171227
	LandSat-8	1_LC08_161132_20171227
T2	Sentinel-1	1_S1A_EW_GRDM_1SDH_20171222T044413_20171222T044517_019810_021B27_D74B
	LandSat-8	1_LC08_158131_20171222
T3	Sentinel-1	1_S1A_EW_GRDM_1SDH_20181219T042800_20181219T042904_025089_02C4EA_6C2C
	LandSat-8	1_LC08_164131_20181219
T4	Sentinel-1	1_S1A_EW_GRDM_1SDH_20211227T042817_20211227T042922_041189_04E500_4D0F
	LandSat-8	1_LC08_164131_20211227
T5	Sentinel-1	1_S1A_EW_GRDM_1SDH_20180112T041941_20180112T042056_020116_0224CF_DCC6
	LandSat-8	1_LC08_161131_20180112
T6	Sentinel-1	1_S1A_EW_GRDM_1SDH_20181214T041948_20181214T042052_025016_02C249_1D9D
	LandSat-8	1_LC08_161132_20181214
T7	Sentinel-1	2_S1A_EW_GRDM_1SDH_20171214T072820_20171214T072853_019695_02178E_F273
	LandSat-8	1_LC08_182131_20171214
T8	Sentinel-1	2_S1A_EW_GRDM_1SDH_20180108T063108_20180108T063213_020059_0222EC_D38D
	LandSat-8	1_LC08_181131_20180108
T9	Sentinel-1	2_S1A_EW_GRDM_1SDH_20180109T053339_20180109T053443_020073_02236D_FA92
	LandSat-8	1_LC08_172132_20180109
V1	Sentinel-1	1_S1A_EW_GRDM_1SDH_20171220T050036_20171220T050140_019781_021A4B_C390
	LandSat-8	1_LC08_160131_20171220
V2	Sentinel-1	1_S1A_EW_GRDM_1SDH_20171224T042753_20171224T042857_019839_021C01_4F12
	LandSat-8	1_LC08_156131_20171224
	LandSat-8	1_LC08_156132_20171224
V3	Sentinel-1	1_S1A_EW_GRDM_1SDH_20200107T042805_20200107T042910_030689_03848D_515B
	LandSat-8	1_LC08_164131_2020010
V4	Sentinel-1	2_S1A_EW_GRDM_1SDH_20171219T055833_20171219T055937_019767_0219E2_C668
	LandSat-8	1_LC08_169132_20171219

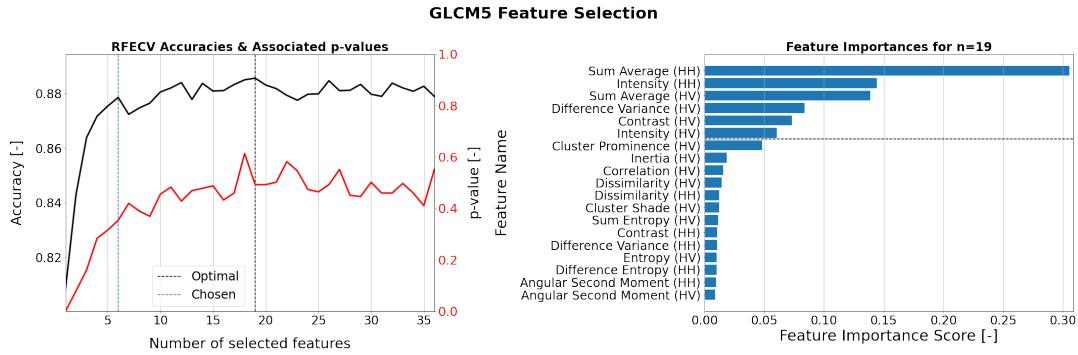


Figure A.1: Feature Selection for the GLCM5 classifier. Left: In blue, recursive Feature Elimination with Cross Validation to select the optimal number of features and in red a student-t test to assess statistical independence for each number of features. Right: feature importance for the chosen number of features (n=6).

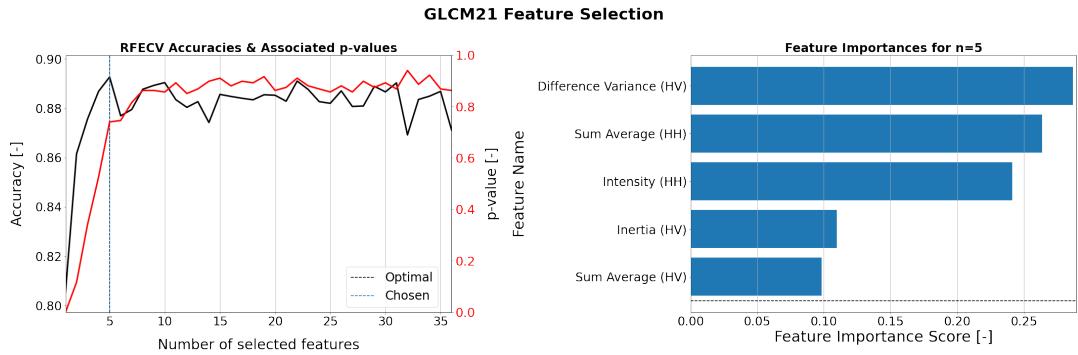


Figure A.2: Feature Selection for the GLCM21 classifier. Left: In blue, recursive Feature Elimination with Cross Validation to select the optimal number of features and in red a student-t test to assess statistical independence for each number of features. Right: feature importance for the chosen number of features (n=5).

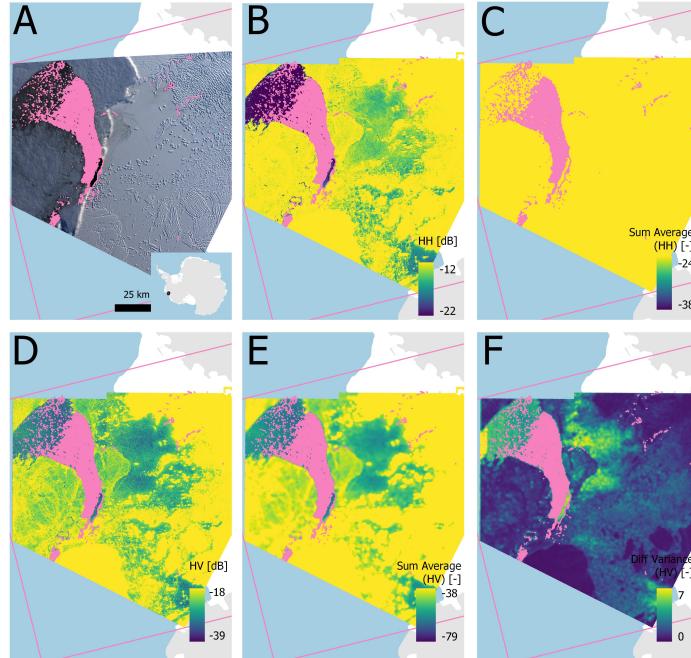


Figure A.3: HH, HV and GLCM21 texture features for V1. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.

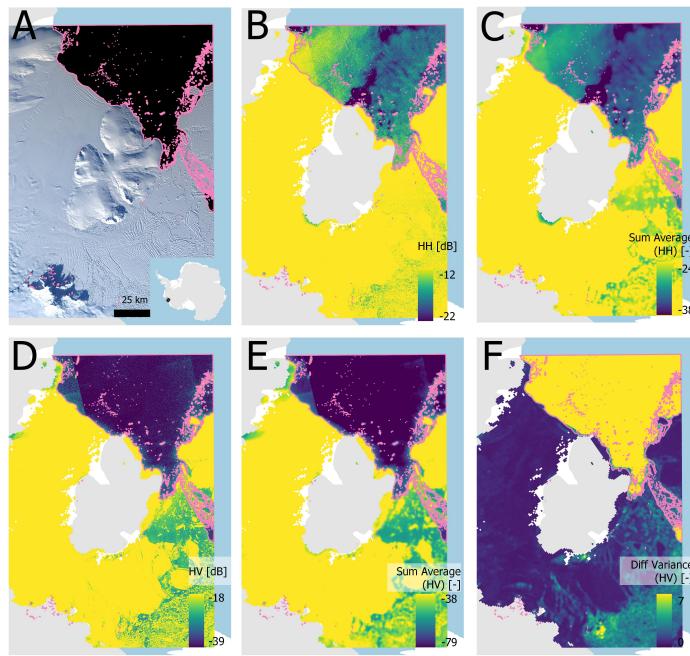


Figure A.4: HH, HV and GLCM21 texture features for V3. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.

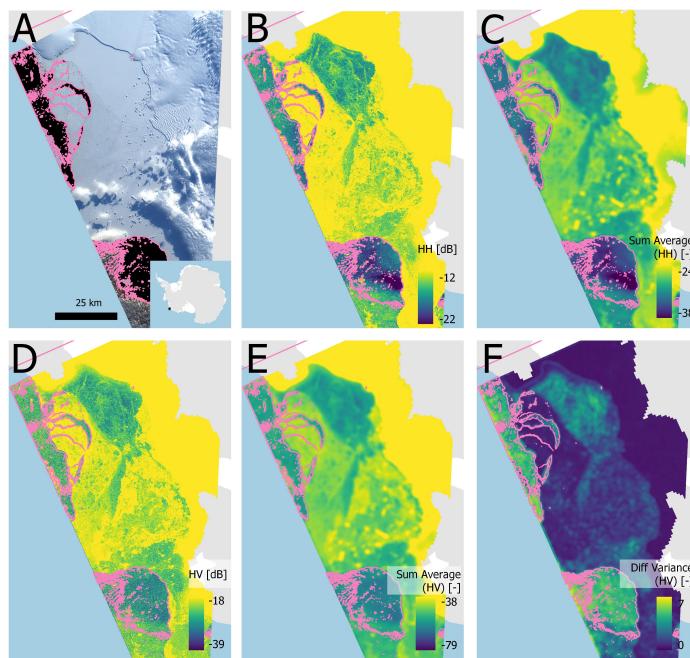


Figure A.5: HH, HV and GLCM21 texture features for V4. Optical image is added for visual reference. For clarity, we highlighted water extent in the optical image by overlaying its water-ice border on top of every image in pink.

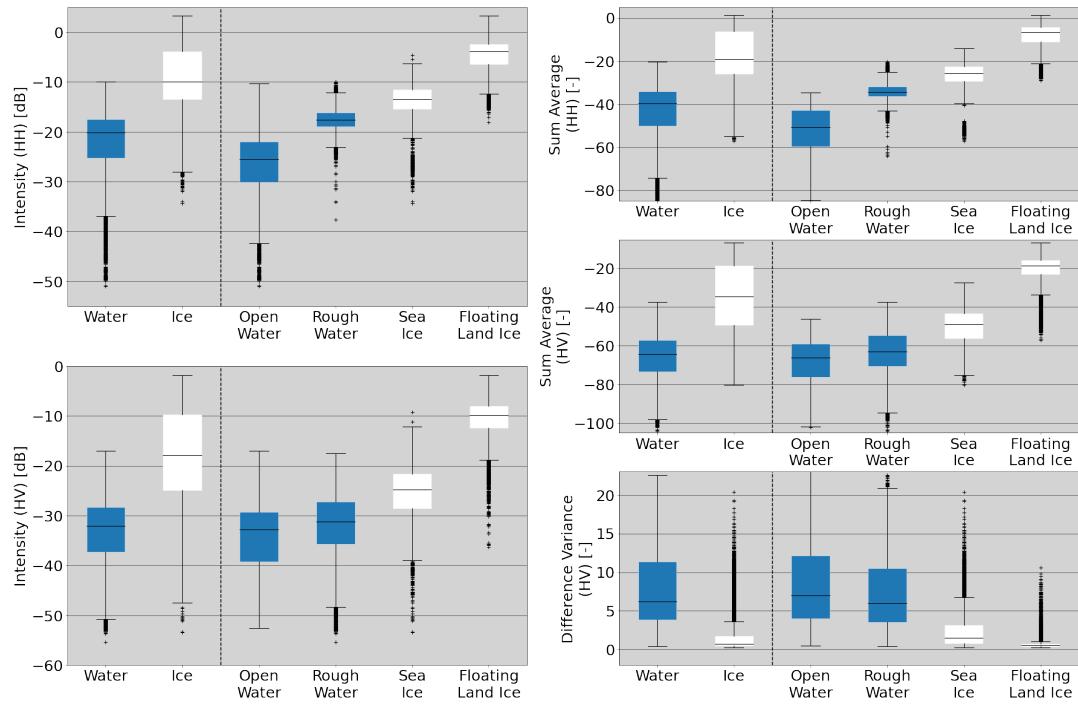


Figure A.6: Boxplots showing the distribution of backscatter intensities (left) and textural features (right) for GLCM5 training data

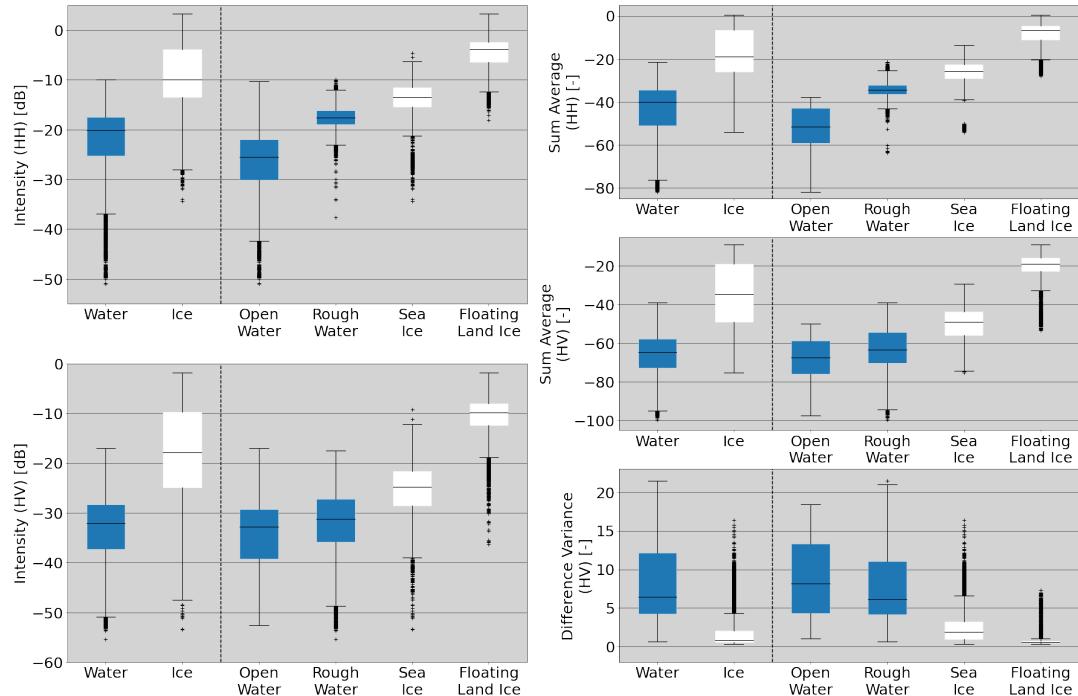


Figure A.7: Boxplots showing the distribution of backscatter intensities (left) and textural features (right) for GLCM21 training data

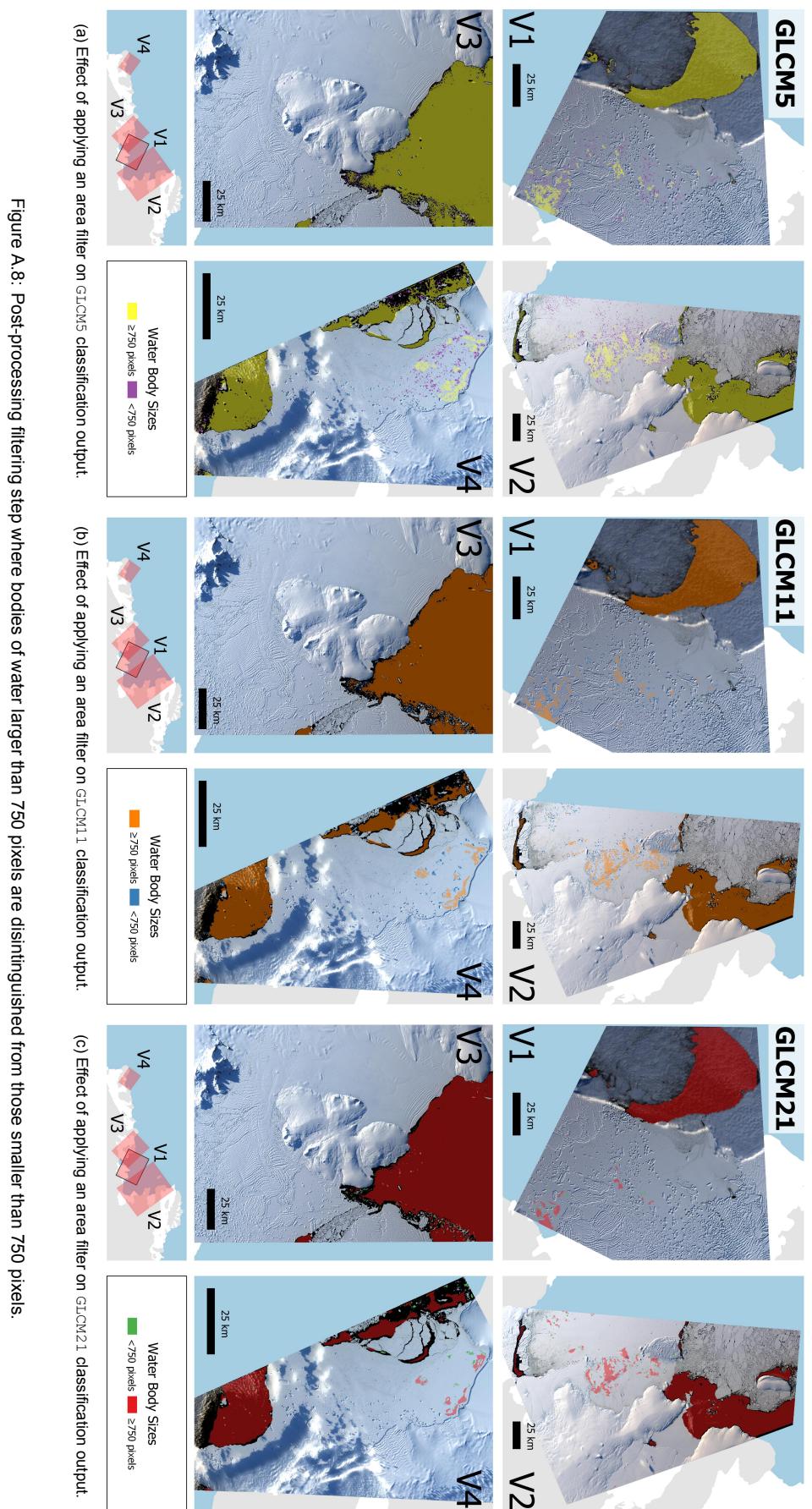
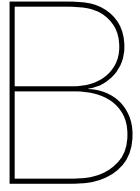


Figure A.8: Post-processing filtering step where bodies of water larger than 750 pixels are distinguished from those smaller than 750 pixels.



Code

B.1. Ice Shelf Coastlines

1_IceShelfCoastLines

```
1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5
6 ## (1.2) Initialize
7 ee.Initialize()
8
9 #%% (2) Create Ice Shelf Coast Line Geometry
10 ## Leave commented!!
11 ## Export was necessary to create feature collections used in section below
12
13 # IceShelves = ee.FeatureCollection("users/sophiederoda/IceShelf_Antarctica")
14 # GroundingLine = ee.FeatureCollection("users/sophiederoda/GroundingLine_Antarctica")
15 # Union = IceShelves.geometry(1e3).union(GroundingLine.geometry(1e3),maxError=1e3)
16 # UnionPoly = ee.Geometry(Union.geometries().get(-1))
17 # CoastLineGeom = ee.Geometry.LineString(ee.List(UnionPoly.coordinates().get(0)))
18 # CoastLine = ee.FeatureCollection([ee.Feature(CoastLineGeom)])
19
20 # geemap.ee_export_vector_to_drive(
21 #     ee_object = CoastLine,
22 #     description = 'AntarcticCoastLine',
23 #     folder = 'CoastLine',
24 #     file_format = 'shp')
25
26 IceShelves = ee.FeatureCollection("users/sophiederoda/IceShelf_Antarctica")
27 CoastLine = ee.FeatureCollection('users/skjeltnmaps/AntarcticCoastLine')
28
29
30 ## (2.1) Divide Coast Line Geometry in Segments
31 def defineSubCoastLines(IceShelfFeature,CoastLineFeatureCollection,maxDistance):
32     IS = IceShelfFeature
33     ISgeom = IS.geometry(1e3)
34     IScoords = ee.List(ISgeom.coordinates().get(0))
35     ISpoly = ee.Geometry.Polygon(IScoords)
36     ISbuffer = ISpoly.buffer(1e2)
37     ISfeat = ee.Feature(ISbuffer).copyProperties(IS)
38
39     CL = CoastLineFeatureCollection.first()
40     CLgeom = CL.geometry(1e3)
41     CLcoords = ee.List(CLgeom.coordinates().get(0))
42     CLmultipoint = ee.Geometry.MultiPoint(CLcoords)
43
44     INTgeom = CLmultipoint.intersection(ISbuffer,maxError=5e3)
45     INTfeat = ee.Feature(INTgeom)
```

```

47     letters = ee.List(['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R',
48     ',S','T','U','V','W','X','Y','Z'])
49     INTcoords = INTfeat.geometry(1e3).coordinates()
50     INTindeces = INTcoords.map(lambda coord: CLcoords.indexOf(ee.List(coord))).removeAll(ee.
51         List([-1]))
52     ISCLcoords = ee.List.sort(INTindeces).map(lambda index: ee.List(CLcoords.get(index)))
53     westbound = ee.List(ISCLcoords.get(0))
54     ITERlist = ee.List([ee.List([0]),westbound,ISCLcoords,maxDistance,ee.List([0])])
55
56     def findInterruptions(coordsIterate,iterations):
57         newCoords = ee.List(coordsIterate)
58         indexList = ee.List(iterations).get(0)
59         oldCoords = ee.List(iterations).get(1)
60         coordinatesList = ee.List(iterations).get(2)
61         maxDistance = ee.Number(iterations).get(3)
62         distanceList = ee.List(iterations).get(4)
63         distance = ee.Geometry.Point(oldCoords).distance(ee.Geometry.Point(newCoords),
64             maxError=1e3)
65
66         return ee.Algorithms.If(
67             condition = distance.gt(maxDistance),
68             trueCase = ee.List([
69                 ee.List(indexList.add(ee.Number(coordinatesList.indexOf(newCoords)))),
70                 ee.List(newCoords),
71                 ee.List(coordinatesList),
72                 ee.Number(maxDistance),
73                 ee.List(distanceList).add(ee.Number(distance))]),
74             falseCase = ee.List([
75                 ee.List(indexList),
76                 ee.List(newCoords),
77                 ee.List(coordinatesList),
78                 ee.Number(maxDistance),
79                 ee.List(distanceList).add(ee.Number(distance))]))
80
81     SUBCLindeces = ee.List(ee.List(
82         function = findInterruptions,
83         first = ITERlist).get(0)).add(-1)
84
85     def sliceSubCoastLines(coords,interruptions,feature):
86         numberofInterruptions = interruptions.size()
87         numberofSlices = numberofInterruptions.subtract(1)
88         indexList = ee.List.sequence(0,numberofSlices.subtract(1))
89         slicedCoords = indexList.map(lambda i: ee.List(coords.slice(
90             ee.Number(interruptions.get(ee.Number(i))),
91             ee.Number(interruptions.get(ee.Number(i).add(1))))))
92         slicedFeatures = slicedCoords.map(lambda coords: ee.Feature(ee.Geometry.LineString(ee.
93             .List(coords)).copyProperties(feature)))
94         name = ee.String(feature.get('NAME'))
95
96         return ee.Algorithms.If(
97             condition = numberofSlices.gt(1),
98             trueCase = indexList.map(lambda i: ee.Feature(slicedFeatures.get(i))\
99                 .set('NAME',name.cat(ee.String('_')).cat(ee.String(letters.get(i))))),
100            falseCase = slicedFeatures)
101
102     ISCLlist = ee.List(sliceSubCoastLines(ee.List(ISCLcoords),ee.List(SUBCLindeces),ee.
103         Feature(ISfeat)))
104     ISCL = ee.FeatureCollection(ISCLlist)
105
106     return ISCL
107
108     def defineSegmentedCoastLines(subCoastCol,maxDistance):
109         def segmentCoastLines(subCoastFeature,maxDistance):
110             feature = ee.Feature(subCoastFeature)
111             coords = feature.geometry().coordinates()
112             westbound = ee.List(coords.get(0))
113             eastbound = ee.List(coords.get(-1))
114             iterationlist = ee.List([ee.List([0]),westbound,coords,maxDistance,ee.List([0])])
115
116             def findSegments(coordsIterate,iterations):
117

```

```

113     newCoords = ee.List(coordsIterate)
114     indexList = ee.List(ee.List(iterations).get(0))
115     oldCoords = ee.List(ee.List(iterations).get(1))
116     coordList = ee.List(ee.List(iterations).get(2))
117     maxDistance = ee.Number(ee.List(iterations).get(3))
118     distanceList = ee.List(ee.List(iterations).get(4))
119     distance = ee.Geometry.Point(oldCoords).distance(ee.Geometry.Point(newCoords),
120             maxError=1e3)
121     cumdist = ee.Number(distanceList.reduce(ee.Reducer.sum())).add(ee.Number(distance
122             ))
123
124     return ee.Algorithms.If(
125         condition = cumdist.gt(maxDistance),
126         trueCase = ee.List([
127             ee.List(indexList.add(ee.Number(coordList.indexOf(newCoords)))),
128             ee.List(newCoords),
129             ee.List(coordList),
130             ee.Number(maxDistance),
131             ee.List([0])]),
132         falseCase = ee.List([
133             ee.List(indexList),
134             ee.List(newCoords),
135             ee.List(coordList),
136             ee.Number(maxDistance),
137             ee.List(distanceList).add(ee.Number(distance))]))
138
139     indeces = ee.List(ee.List(coords.iterate(
140         function = findSegments,
141         first = iterationlist)).get(0))
142
143     segmentIndeces = ee.Algorithms.If(
144         condition = ee.Number(indeces.get(-1)).eq(ee.Number(coords.indexOf(eastbound))),
145         trueCase = indeces,
146         falseCase = indeces.add(coords.indexOf(eastbound)))
147
148     numberofSegments = ee.List(segmentIndeces).size()
149     numberofSlices = numberofSegments.subtract(1)
150     indexList = ee.List.sequence(0, numberofSlices.subtract(1))
151     slicedCoords = indexList.map(lambda i: ee.Algorithms.If(
152         condition = ee.Number(ee.List(segmentIndeces).get(ee.Number(i).add(1))).eq(ee.
153             Number(coords.indexOf(eastbound))),
154         trueCase = ee.List(coords.slice(
155             ee.Number(ee.List(segmentIndeces).get(ee.Number(i))),
156             ee.Number(ee.List(segmentIndeces).get(ee.Number(i).add(1))))).add(eastbound),
157         falseCase = ee.List(coords.slice(
158             ee.Number(ee.List(segmentIndeces).get(ee.Number(i))),
159             ee.Number(ee.List(segmentIndeces).get(ee.Number(i).add(1))))))
160     ))
161
162     segmentlist = indexList.map(lambda i: ee.Feature(ee.Geometry.LineString(ee.List(
163         slicedCoords.get(i))).copyProperties(feature).set('SEG',
164         ee.String(ee.Number(i))))
165
166     featCol = ee.FeatureCollection(segmentlist)
167
168     return featCol
169
170     segmentFeatCol = ee.FeatureCollection(subCoastCol).map(lambda feature: segmentCoastLines(
171         feature, maxDistance))
172
173     return segmentFeatCol
174
175     ## Segment distance was set to 20km
176     maxDistance = 20e3
177     subCoastLines = IceShelves.map(lambda iceshelf: defineSubCoastLines(iceshelf, CoastLine,
178             maxDistance)).flatten()
179     segCoastLines = defineSegmentedCoastLines(subCoastLines, maxDistance).flatten()
180
181     ## Leave commented!!
182     ## Export was necessary to create feature collections used in section below
183
184

```

```

177 # geemap.ee_export_vector_to_drive(
178 #     ee_object = segCoastLines,
179 #     description = 'ISCLSegments'+str(int(maxDistance/1e3))+' km',
180 #     folder = 'Geometries\IceShelfCoastLineSegments',
181 #     file_format = 'shp')

```

B.2. (Reference) Image Metadata

2_Metadata

```

1 %% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5
6 ## (1.2) Initialize
7 ee.Initialize()
8
9 ## (1.3) Parameterization
10 start = '2014-01-01'
11 end = '2022-04-01'
12 cloudcover = 20
13 whiteness = 3e4
14 S1band = 'HV'
15 dt = 4
16
17 L8bands = ['B4','B3','B2']
18 S2bands = ['B4','B3','B2']
19 RGBbands = ['Red','Green','Blue']
20
21 IceShelves = ee.FeatureCollection("users/sophiederoda/IceShelf_Antarctica")
22 ROIs = ee.FeatureCollection('users/skjeltnmaps/ISCLSegments20km')
23
24 %% (2) Optical Data Set
25 ## (2.1) Cloud Mask Functions
26 def cloudMaskL8(image):
27     def getQABits(image, bitStart, bitEnd, name):
28         pattern = 0
29         for i in range(bitStart,bitEnd+1):
30             pattern += 2**i
31
32         return ee.Image(image).select([0], [name]).bitwiseAnd(pattern).rightShift(bitStart)
33
34 QA = ee.Image(image).select('BQA').toInt()
35 cloud = getQABits(QA,4,4,'CLOUD')
36 cloud_confidence = getQABits(QA,5,6,'CLOUD_CONFIDENCE')
37 cloudshadow_confidence = getQABits(QA,7,8,'CLOUDSHADOW_CONFIDENCE')
38 cirrus_confidence = getQABits(QA,11,12,'CIRRUS_CONFIDENCE')
39
40     return ee.Image(image)
41         .updateMask(cloud.eq(0))
42         .updateMask(cloud_confidence.lt(3))
43         .updateMask(cloudshadow_confidence.lt(3))
44         .updateMask(cirrus_confidence.lt(3))
45
46 def cloudMaskS2(image):
47     qa = ee.Image(image).select('QA60').toInt()
48
49     cloudBitMask = 1 << 10
50     cirrusBitMask = 1 << 11
51
52     cloud = qa.bitwiseAnd(cloudBitMask).eq(0)
53     cirrus = qa.bitwiseAnd(cirrusBitMask).eq(0)
54
55     return ee.Image(image)
56         .updateMask(cloud)
57         .updateMask(cirrus)
58
59 def addRGBbands(image,bandnames):
60     renamed = image.select(bandnames,RGBbands)

```

```

61     rgb = image.addBands(renamed).select(RGBbands)
62     return rgb
63
64 def addMaxWhiteness(image):
65     maxvalues = image.select(['B2','B3','B4']).reduceRegion(
66         reducer = ee.Reducer.max(),
67         bestEffort = True,
68         maxPixels = 1e3)
69     blue = ee.Number(maxvalues.get('B2'))
70     green = ee.Number(maxvalues.get('B3'))
71     red = ee.Number(maxvalues.get('B4'))
72     total = blue.add(green).add(red)
73     return image.set('mean',total)
74 ## (2.2) Preprocessing & merging
75 def mergeOpticalData(ROIfeat,start,end,cloudcover,whiteness):
76     #----- General -----
77     roi = ROIfeat.geometry()
78
79     #----- Landsat 8 -----
80     L8 = ee.ImageCollection("LANDSAT/LC08/C01/T2")
81         .filterDate(start,end)
82         .filterBounds(roi)
83         .filterMetadata('CLOUD_COVER', 'less_than', cloudcover)
84         .map(lambda image: image.set('Data','L8'))
85         .map(addMaxWhiteness)
86         .filterMetadata('mean', 'greater_than', whiteness)
87         .sort('system:time_start')
88
89     #----- Sentinel 2 -----
90     S2 = ee.ImageCollection("COPERNICUS/S2")
91         .filterDate(start, end)
92         .filterBounds(roi)
93         .filterMetadata('CLOUDY_PIXEL_PERCENTAGE', 'less_than', cloudcover)
94         .map(lambda image: image.set('Data','S2'))
95         .map(addMaxWhiteness)
96         .filterMetadata('mean', 'greater_than', whiteness)
97         .sort('system:time_start')
98
99     #----- Merged -----
100    Opt = L8.merge(S2).sort('system:time_start')
101
102    return ee.List([L8, S2, Opt])
103
104 %% (3) Sentinel-1 Data Set
105 def preProcessS1(ROIfeat,S1band,start,end):
106     #----- General -----
107     roi = ROIfeat.geometry()
108
109     #----- Preprocessing -----
110     S1_premerged = ee.ImageCollection("COPERNICUS/S1_GRD")
111         .filterDate(start,end)
112         .filterBounds(roi)
113         .filter( ee.Filter.listContains('transmitterReceiverPolarisation',S1band))
114         .sort('system:time_start')
115
116     return S1_premerged
117 %% (4) Function to exclude images that are not within 4 hours
118 def S1OptTempFilter(S1,Opt,hours):
119     #----- Temporal Filter -----
120     temporalFilter = ee.Filter.maxDifference(
121         difference = hours * 60 * 60 * 1000,
122         leftField = 'system:time_start',
123         rightField = 'system:time_start')
124     temporalJoin = ee.Join.saveAll(
125         matchesKey = 'temporalMatches',
126         measureKey = 'timeDiff',
127         ordering = 'system:time_start',
128         ascending = True)
129
130     #----- Matching -----
131     temporalJoined = temporalJoin.apply(S1, Opt, temporalFilter)

```

```

132
133     S1Matched = ee.ImageCollection(temporalJoined)
134
135     return S1Matched
136 #%% (5) Export Metadata Per Coastline Segment
137 ## (5.1) S1-Opt Matches
138 def S1OptMatchFunction(ROIs,S1band,start,end,cloudcover,whiteness,hours):
139     def matchesPerSegment(ROIfeat,S1band,start,end,cloudcover,whiteness,hours):
140         Opt = ee.ImageCollection(mergeOpticalData(ROIfeat,start,end,cloudcover,whiteness).get(
141             (2)))
142         S1processed = preProcessS1(ROIfeat,S1band,start,end)
143         matchCol = S1OptTempFilter(S1processed,Opt,hours)
144         return matchCol
145
146 def saveMatchMetadata(ROIfeat,matchCol):
147     S1Col = matchCol
148     OptList = matchCol.aggregate_array('temporalMatches').flatten()
149     return ee.Algorithms.If(
150         condition = S1Col.size().gt(0),
151         trueCase = ee.FeatureCollection(ee.List([
152             S1Col.toList(S1Col.size()).map(lambda image: ee.Feature(ROIfeat.geometry(),{
153                 'NAME': ee.String(ROIfeat.get('NAME')),
154                 'SEG': ee.String(ROIfeat.get('SEG')),
155                 'S1_UTC': ee.Number(ee.Image(image).get('system:time_start')),
156                 'Opt_UTC': ee.Number(-9999)}),
157             OptList.map(lambda image: ee.Feature(ROIfeat.geometry(),{
158                 'NAME': ee.String(ROIfeat.get('NAME')),
159                 'SEG': ee.String(ROIfeat.get('SEG')),
160                 'S1_UTC': ee.Number(-9999),
161                 'Opt_UTC': ee.Number(ee.Image(image).get('system:time_start'))}))
162         ]).flatten()),
163         falseCase = ee.FeatureCollection([
164             ee.Feature(ROIfeat.geometry(),{
165                 'NAME': ee.String(ROIfeat.get('NAME')),
166                 'SEG': ee.String(ROIfeat.get('SEG')),
167                 'S1_UTC': ee.Number(-9999),
168                 'Opt_UTC': ee.Number(-9999)})
169         ])
170     )
171
172     matchMetadata = ee.FeatureCollection(ROIs      .map(lambda feature: saveMatchMetadata(
173         ee.Feature(feature),
174         ee.ImageCollection(matchesPerSegment(ee.Feature(feature),S1band,start,end,
175             cloudcover,whiteness,hours))))\
176     .flatten())
177
178     return matchMetadata
179
180 S1OptMatchMetadata = S1OptMatchFunction(ROIs,S1band,start,end,cloudcover,whiteness,dt)
181
182 ## Leave commented!!
183 ## Export was necessary to create feature collections used in section below
184
185 # geemap.ee_export_vector_to_drive(
186 #     ee_object = S1OptMatchMetadata,
187 #     description = 'S1Opt_'+S1band+'_'+str(dt)+'hrs_Metadata_'+start[:4]+start[5:7]+start
188 #     [8:10]+'_'+end[:4]+end[5:7]+end[8:10],
189 #     folder = 'Metadata',
190 #     file_format = 'shp')
191
192 ## (5.2) All S1 Images
193 def countS1Function(ROIs,start,end):
194     def saveS1Metadata(ROIfeat,S1):
195         S1HH = S1.filter(ee.Filter.listContains('transmitterReceiverPolarisation','HH'))
196
197         return ee.Algorithms.If(
198             condition = S1HH.size().gt(0),
199             trueCase = ee.FeatureCollection(ee.List([
200                 S1HH.toList(S1HH.size()).map(lambda image: ee.Feature(ROIfeat.geometry(),{
201                     'NAME': ee.String(ROIfeat.get('NAME')),
202                     'SEG': ee.String(ROIfeat.get('SEG'))),
203                     'S1_UTC': ee.Number(ee.Image(image).get('system:time_start')),
204                     'Opt_UTC': ee.Number(ee.Image(image).get('system:time_start'))}))
205             ]).flatten())
206
207         return ee.FeatureCollection([
208             ee.Feature(ROIfeat.geometry(),{
209                 'NAME': ee.String(ROIfeat.get('NAME')),
210                 'SEG': ee.String(ROIfeat.get('SEG')),
211                 'S1_UTC': ee.Number(-9999),
212                 'Opt_UTC': ee.Number(-9999)})
213         ])
214     )
215
216     matchMetadata = ee.FeatureCollection(ROIs      .map(lambda feature: saveS1Metadata(
217         ee.Feature(feature),
218         ee.ImageCollection(countS1Function(ee.Feature(feature),start,end,
219             cloudcover,whiteness,hours))))\
220     .flatten())
221
222     return matchMetadata
223
224 S1OptMatchMetadata = S1OptMatchFunction(ROIs,S1band,start,end,cloudcover,whiteness,dt)

```

```

200         'S1_UTC': ee.Number(ee.Image(image).get('system:time_start')),
201         'Polarisation': ee.String(ee.List(ee.Image(image).get(
202             'transmitterReceiverPolarisation')).get(-1)),
203         'Overpasses': S1HH.size()})),
204     ]).flatten())),
205     falseCase = ee.FeatureCollection([
206         ee.Feature(ROIfeat.geometry(),{
207             'NAME': ee.String(ROIfeat.get('NAME')),
208             'SEG': ee.String(ROIfeat.get('SEG')),
209             'S1_UTC': ee.Number(-9999),
210             'Polarisation': ee.String('NoData'),
211             'Overpasses': ee.Number(0)})
212     ])
213 )
214
215 S1Metadata = ROIs
216 .map(lambda feature: saveS1Metadata(
217     ee.Feature(feature),
218     preProcessS1(ee.Feature(feature), 'HH', start,end)))\
219     .flatten()
220
221     return S1Metadata
222
223 S1Metadata = countS1Function(ROIs,start,end)
224
225 ## Leave commented!!
226 ## Export was necessary to create feature collections used in section below
227 # geemap.ee_export_vector_to_drive(
228 #     ee_object = S1Metadata,
229 #     description = 'S1_Metadata_'+start[:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+end
230 #     [8:10],
231 #     folder = 'Metadata',
232 #     file_format = 'shp')
233
234 ## (5.3) All Optical Images
235 def countOptFunction(ROIs,start,end,cloudcover,whiteness):
236     def saveOptMetadata(ROIfeat,Opt):
237         OptMetadata = ee.Algorithms.If(
238             condition = Opt.size().gt(0),
239             trueCase = ee.FeatureCollection(ee.List([
240                 Opt.toList(Opt.size()).map(lambda image: ee.Feature(ROIfeat.geometry(),{
241                     'NAME': ee.String(ROIfeat.get('NAME')),
242                     'SEG': ee.String(ROIfeat.get('SEG')),
243                     'Opt_UTC': ee.Number(ee.Image(image).get('system:time_start')),
244                     'Data': ee.String(ee.Image(image).get('Data')),
245                     'Overpasses': Opt.size()})),
246                 ]).flatten()),
247             falseCase = ee.FeatureCollection([
248                 ee.Feature(ROIfeat.geometry(),{
249                     'NAME': ee.String(ROIfeat.get('NAME')),
250                     'SEG': ee.String(ROIfeat.get('SEG')),
251                     'Opt_UTC': ee.Number(-9999),
252                     'Data': ee.String('NoData'),
253                     'Overpasses': ee.Number(0)})
254             ]))
255
256         return OptMetadata
257
258     OptMetadata = ROIs
259     .map(lambda feature: saveOptMetadata(
260         ee.Feature(feature),
261         ee.ImageCollection(mergeOpticalData(ee.Feature(feature),start,end,cloudcover,
262             whiteness).get(2))))\
263         .flatten()
264
265     return OptMetadata
266
267 OptMetadata = countOptFunction(ROIs,start,end,cloudcover,whiteness)
268

```

```

268 ## Leave commented!!
269 ## Export was necessary to create feature collections used in section below
270
271 # geemap.ee_export_vector_to_drive(
272 #     ee_object = OptMetadata,
273 #     description = 'Opt_Metadata_'+start[:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+end
274 #         [8:10],
275 #     folder = 'Metadata',
276 #     file_format = 'shp')

```

B.3. Overpasses

3_Overpasses

```

1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5
6 ## (1.2) Initialize
7 ee.Initialize()
8
9 ## (1.3) Parameterization & Data Import
10 start = '2014-01-01'
11 end = '2022-04-01'
12 S1band = 'HV'
13 dt = 4
14
15 S1OptMetadata = ee.FeatureCollection('users/skjeltmaps/S1Opt_'+S1band+'_'+str(dt)+'_
16     hrs_Metadata_'+start[:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+end[8:10])
16 OptMetadata = ee.FeatureCollection('users/skjeltmaps/Opt_Metadata_'+start[:4]+start[5:7]+
17     start[8:10]+'_'+end[:4]+end[5:7]+end[8:10])
17 S1Metadata = ee.FeatureCollection('users/skjeltmaps/S1_Metadata_'+start[:4]+start[5:7]+start
18     [8:10]+'_'+end[:4]+end[5:7]+end[8:10])
18
19 Opt UTC = S1OptMetadata.aggregate_array('Opt UTC')
20 S1 UTC = S1OptMetadata.aggregate_array('S1 UTC')
21
22 IceShelves = ee.FeatureCollection("users/sophiederoda/IceShelf_Antarctica")
23 ROIs = ee.FeatureCollection('users/skjeltmaps/ISCLSegments20km')
24
25 #%% (2) Reconstruct Match Data Set
26 ## (2.1) General Functions
27
28 def addMetadata(image,utcname,metadata):
29     utc = ee.String(image.get('system:time_start'))
30     filtered = metadata.filter(ee.Filter.inList(utcname,[utc]))
31     name = filtered.aggregate_array('NAME').distinct()
32     segment = name.map(lambda name: filtered.filterMetadata('NAME','equals',name).
33         aggregate_array('SEG'))
34     wb = ee.List(ee.List(image.geometry(1e4).bounds(1e4).coordinates().get(0)).get(0)).
35         get(0)
36     return image.set('NAME',name).set('SEG',segment).set('WB',wb)
36
36 ## (2.2) Reconstruction
37 S1 = ee.ImageCollection("COPERNICUS/S1_GRD")
38     .filter(ee.Filter.inList('system:time_start',S1 UTC))
39     .filter(ee.Filter.listContains('transmitterReceiverPolarisation',S1band))
40     .map(lambda image: addMetadata(image,'S1 UTC',S1OptMetadata))
41     .sort('system:time_start')
42
43 L8 = ee.ImageCollection("LANDSAT/LC08/C01/T2")
44     .filter(ee.Filter.inList('system:time_start',Opt UTC))
45     .map(lambda image: addMetadata(image,'Opt UTC',S1OptMetadata))
46 S2 = ee.ImageCollection("COPERNICUS/S2")
47     .filter(ee.Filter.inList('system:time_start',Opt UTC))
48     .map(lambda image: addMetadata(image,'Opt UTC',S1OptMetadata))
49 Opt = L8.merge(S2).sort('WB')
50
51 ## (2.3) Create S1-Opt Match Data Set

```

```

52 def countTemporalMatches(S1Image, OptCol, hours):
53     image = S1Image
54     utc = image.get('system:time_start')
55     names = ee.List(image.get('NAME'))
56     segs = ee.List(image.get('SEG'))
57     indeces = ee.List.sequence(0, names.size().subtract(1))
58     filtered = ee.FeatureCollection(indeces.map(lambda i: S1OptMetadata.filterMetadata(
59         'NAME', 'equals', names.get(i)).filter(ee.Filter.inList('SEG', segs.get(i)))).flatten())
60     coastlinegeometry = filtered.geometry()
61
62     OptRoi = OptCol.filterBounds(coastlinegeometry)
63
64     temporalFilter = ee.Filter.maxDifference(
65         difference = hours * 60 * 60 * 1000,
66         leftField = 'system:time_start',
67         rightValue = utc)
68     OptFiltered = OptRoi.filter(temporalFilter)
69
70     return image.set('COUNT', OptFiltered.size())
71
72 S1OptDt = S1.map(lambda image: countTemporalMatches(image, Opt, dt))
73
74 #%% (3) Compute Overpasses
75 ## (3.1) General Functions
76 def matchOverpasses(imCol, roi):
77     segmentRoi = roi.geometry()
78     segCol = imCol.filterBounds(segmentRoi)
79     counts = segCol.aggregate_sum('COUNT')
80
81     segmentOverpasses = ee.Feature(segmentRoi, {
82         'NAME': ee.String(roi.get('NAME')),
83         'SEG': ee.Number(roi.get('SEG')),
84         'COUNTS': ee.Number(counts)})
85
86     return segmentOverpasses
87
88 def collectionOverpasses(imCol, roi):
89     segRoi = roi.geometry()
90     segCol = imCol.filterBounds(segRoi)
91     counts = segCol.size()
92
93     segmentOverpasses = ee.Feature(segRoi, {
94         'NAME': ee.String(roi.get('NAME')),
95         'SEG': ee.Number(roi.get('SEG')),
96         'COUNTS': ee.Number(counts)})
97
98     return segmentOverpasses
99
100 def combineCollections(featCol1, featCol2, roi, ids):
101     segRoi = roi.geometry()
102     segCol1 = featCol1.filterBounds(segRoi)
103     segCol2 = featCol2.filterBounds(segRoi)
104
105     counts1 = ee.Number(segCol1.first().get('COUNTS'))
106     counts2 = ee.Number(segCol2.first().get('COUNTS'))
107     countsSum = counts1.add(counts2)
108
109     combined = ee.Feature(segRoi, {
110         'NAME': ee.String(roi.get('NAME')),
111         'SEG': ee.Number(roi.get('SEG')),
112         'COUNTS'+ids[0]: counts1,
113         'COUNTS'+ids[1]: counts2,
114         'COUNTSTOTAL': countsSum})
115
116     return combined
117
118 ## (3.2) Overpass Computation
119 S1OptOverpasses = ROIs.map(lambda roifeat: matchOverpasses(S1OptDt, roifeat))
120
121 L8Overpasses = ROIs.map(lambda roifeat: collectionOverpasses(OptMetadata.filterMetadata('Data
122     ','equals','L8'), roifeat))

```

```

120 S2Overpasses = ROIs.map(lambda roifeat: collectionOverpasses(OptMetadata.filterMetadata('Data
121     ','equals','S2'),roifeat))
122 OptOverpasses = ROIs.map(lambda roifeat: combineCollections(L8Overpasses,S2Overpasses,roifeat
123     ,['L8','S2']))
124
125 S1HHOverpasses = ROIs.map(lambda roifeat: collectionOverpasses(S1Metadata.filterMetadata('
126     Polarisation','equals','HH'),roifeat))
127 S1HVOverpasses = ROIs.map(lambda roifeat: collectionOverpasses(S1Metadata.filterMetadata('
128     Polarisation','equals','HV'),roifeat))
129 S1Overpasses = ROIs.map(lambda roifeat: combineCollections(S1HHOverpasses,S1HVOverpasses,
130     roifeat,['HH','HV']))
131
132 #%% (4) Export Overpass Collections
133 ## (4.1) S1-Opt Overpasses
134 ## Leave commented!!
135 ## Export was necessary to create feature collections used in section below
136
137 # geemap.ee_export_vector_to_drive(
138 #     ee_object = S1OptOverpasses,
139 #     description = 'S1opt_'+S1band+'_'+str(dt)+'hrs_Overpasses_'+start[:4]+start[5:7]+start
140 #         [8:10]+'_'+end[:4]+end[5:7]+end[8:10],
141 #     folder = 'OverpassesPerSegment',
142 #     file_format = 'shp')
143
144 ## (4.2) Optical Overpasses
145 ## Leave commented!!
146 ## Export was necessary to create feature collections used in section below
147
148 # geemap.ee_export_vector_to_drive(
149 #     ee_object = OptOverpasses,
150 #     description = 'Opt_Overpasses_'+start[:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+
151 #         end[8:10],
152 #     folder = 'OverpassesPerSegment',
153 #     file_format = 'shp')
154
155 ## (4.3) S1 Overpasses
156 ## Leave commented!!
157 ## Export was necessary to create feature collections used in section below
158
159 # geemap.ee_export_vector_to_drive(
160 #     ee_object = S1Overpasses,
161 #     description = 'S1_Overpasses_'+start[:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+
162 #         end[8:10],
163 #     folder = 'OverpassesPerSegment',
164 #     file_format = 'shp')

```

B.4. Match Data Set

4_ReferenceData

```

1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5 import numpy as np
6 import os
7
8 ## (1.2) Initialize
9 ee.Initialize()
10
11 ## (1.3) Parameterization & Data Import
12 start = '2014-01-01'
13 end = '2022-04-01'
14 crs = 'EPSG:3031'
15 S1band = 'HV'
16 dt = 4
17 dataset = 'S1Opt'
18
19 S1OptMetadata = ee.FeatureCollection('users/skjeltmaps/S1Opt_'+S1band+'_'+str(dt)+'
20     hrs_Metadata_'+start[:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+end[8:10])

```

```

20 Metadata = eval(dataset+'Metadata')
21
22 Opt_UTC = Metadata.aggregate_array('Opt_UTC')
23 S1_UTC = Metadata.aggregate_array('S1_UTC')
24
25 GroundingLine = ee.FeatureCollection("users/sophiederoda/GroundingLine_Antarctica")
26 DEM = ee.Image('CPOM/CryoSat2/ANTARCTICA_DEM').select('elevation')
27 IceShelves = ee.FeatureCollection("users/sophiederoda/IceShelf_Antarctica")
28 ROIs = ee.FeatureCollection('users/skjeltmaps/ISCLSegments20km')
29
30 AmundsenEmbayment = ee.Geometry.Polygon(
31     coords = [[[-108.406159, -75.468542], [-108.406159, -71.238725], [-96.219667,
32         -71.238725], [-96.219667, -75.468542], [-108.406159, -75.468542]]],
33     geodesic = False,
34     proj = 'EPSG:4326')
35 CarneyCoast = ee.Geometry.Polygon(
36     coords = [[[-146.815157, -74.904569], [-142.211175, -76.343611], [-128.293796,
37         -74.877073], [-120.513741, -75.203774], [-120.513741, -73.617891], [-127.520609,
38         -72.722358], [-146.815157, -74.904569]]],
39     geodesic = False,
40     proj = 'EPSG:4326')
41
42 trainingName = ['AmundsenEmbayment']
43 trainingArea = AmundsenEmbayment
44
45 validationName = ['CarneyCoast']
46 validationArea = CarneyCoast
47
48 trainValArea = trainingArea.union(validationArea,maxError=1e3)
49
50 L8name = 'L8'
51 L8bands = ['B4','B3','B2']
52 L8_params = {
53     'bands': L8bands,
54     'min': 0,
55     'max': 30000}
56
57 S2name = 'S2'
58 S2bands = ['B4', 'B3', 'B2']
59 S2_params = {
60     'min': 0,
61     'max': 0.3*1e4,
62     'bands': S2bands}
63
64 RGBbands = ['Red','Green','Blue']
65 RGB_params = {
66     'bands': RGBbands,
67     'min': 0,
68     'max': 3e4}
69
70 S1name = 'S1'
71 HHmin = -30
72 HHmax = 2.5
73 HVmin = -40
74 HVmax = -3
75
76 S1_HH_params = {
77     'min': HHmin,
78     'max': HHmax}
79 S1_HV_params = {
80     'min': HVmin,
81     'max': HVmax}
82 S1_RGB_params = {
83     'min': [HVmin,HHmin,HHmin],
84     'max': [HVmax,HHmax,HHmax]}
85 angle_params = {
86     'min': 0,
87     'max': 90}
88
89 ## Edit visualization export folder is necessary
90 # html_dir = os.path.join(os.path.expanduser('~'),'Downloads')

```

```

88 # if not os.path.exists(html_dir):
89 #     os.makedirs(html_dir)
90
91 #%% (2) Reconstruct Match Data Set
92 ## (2.1) Cloud Mask Functions
93 def cloudMaskL8(image):
94     def getQABits(image, bitStart, bitEnd, name):
95         # Compute the bits we need to extract.
96         pattern = 0
97         for i in range(bitStart,bitEnd+1):
98             pattern += 2**i
99         return ee.Image(image).select([0], [name]).bitwiseAnd(pattern).rightShift(bitStart)
100
101 QA = ee.Image(image).select('BQA').toInt()
102 cloud = getQABits(QA,4,4,'CLOUD')
103 cloud_confidence = getQABits(QA,5,6,'CLOUD_CONFIDENCE')
104 cloudshadow_confidence = getQABits(QA,7,8,'CLOUDSHADOW_CONFIDENCE')
105 cirrus_confidence = getQABits(QA,11,12,'CIRRUS_CONFIDENCE')
106
107 return ee.Image(image).updateMask(cloud.eq(0))
108 .updateMask(cloud_confidence.lt(3))
109 .updateMask(cloudshadow_confidence.lt(3))
110 .updateMask(cirrus_confidence.lt(3))
111
112 def cloudMaskS2(image):
113     qa = ee.Image(image).select('QA60').toInt()
114
115     cloudBitMask = 1 << 10
116     cirrusBitMask = 1 << 11
117
118     cloud = qa.bitwiseAnd(cloudBitMask).eq(0)
119     cirrus = qa.bitwiseAnd(cirrusBitMask).eq(0)
120
121     return ee.Image(image)
122         .updateMask(cloud)
123         .updateMask(cirrus)
124
125 ## (2.2) General Functions
126 def utcToLocal(image):
127     centroid = ee.Image(image).geometry(maxError=1e3).centroid(maxError=1e3)
128     longitude = centroid.coordinates().get(0)
129     localTime = ee.Image(image).date().advance(ee.Number(longitude).divide(15).ceil().
130         subtract(1),'hour')
131     localMillis = ee.Date(localTime).millis()
132
133     return image
134         .set('utctime',ee.Image(image).get('system:time_start'))
135         .set('localtime',localMillis)
136
137 def addID(image,name):
138     id = ee.String(name+' ').cat(ee.String(ee.Date(ee.Image(image).get('localtime')).format(
139         'YYYY_MM_dd_KK:mm:ss'))))
140     return image.set('id',id).set('data',name)
141
142 def reproject(image):
143     scale = image.select(0).projection().nominalScale()
144     return image.reproject(crs=crs,scale=scale)
145
146 def addRGBbands(image,bandnames):
147     renamed = image.select(bandnames,RGBbands)
148     rgb = image.addBands(renamed).select(RGBbands)
149     return rgb
150
151 def addMetadata(image,utcname,metadata):
152     utc = ee.String(image.get('system:time_start'))
153     filtered = metadata.filter(ee.Filter.inList(utcname,[utc]))
154     name = filtered.aggregate_array('NAME').distinct()
155     segment = name.map(lambda name: filtered.filterMetadata('NAME','equals',name).
156         aggregate_array('SEG'))
157     wb = ee.List(ee.List(image.geometry(1e4).bounds(1e4).coordinates().get(0)).get(0)).

```

```

    get(0)
156     return image.set('NAME',name).set('SEG',segment).set('WB',wb)
157
158 def angleNormalization(image):
159     def degreesToRadians(object):
160         return object.multiply(2*np.pi).divide(180)
161
162     thetaRef = degreesToRadians(ee.Number(30))
163     radians = degreesToRadians(image.select('angle'))
164
165     HH = image.select('HH').multiply(thetaRef.cos().pow(ee.Number(2))).divide(radians.cos() .
166         pow(ee.Number(2)))
166     HV = image.select('HV').multiply(thetaRef.cos().pow(ee.Number(2))).divide(radians.cos() .
167         pow(ee.Number(2)))
168
168     normalized = image.addBands(HH,overwrite=True).addBands(HV,overwrite=True)
169
170     return normalized
171
172 ## (2.3) Reconstruction
173 S1 = ee.ImageCollection("COPERNICUS/S1_GRD")
174     .filter(ee.Filter.inList('system:time_start',S1_UTC))
175     .filter(ee.Filter.listContains('transmitterReceiverPolarisation',S1band))
176     .map(lambda image: addMetadata(image,'S1_UTC',Metadata))
177     .map(utcToLocal)
178     .map(lambda image: addID(image,S1name))
179     .sort('WB')
180     .filterBounds(trainValArea)
181
182 L8 = ee.ImageCollection("LANDSAT/LC08/C01/T2")
183     .filter(ee.Filter.inList('system:time_start',Opt_UTC))
184     .map(lambda image: addRGBbands(image,L8bands))
185     .map(lambda image: addMetadata(image,'Opt_UTC',Metadata))
186     .map(utcToLocal)
187     .map(lambda image: addID(image,L8name))
188     .filterBounds(trainValArea)
189 S2 = ee.ImageCollection("COPERNICUS/S2")
190     .filter(ee.Filter.inList('system:time_start',Opt_UTC))
191     .map(lambda image: addRGBbands(image,S2bands))
192     .map(lambda image: addMetadata(image,'Opt_UTC',Metadata))
193     .map(utcToLocal)
194     .map(lambda image: addID(image,S2name))
195     .filterBounds(trainValArea)
196 Opt = L8.merge(S2).sort('WB')
197 OptList = Opt.toList(Opt.size())
198
199 ## (2.4) Quality Assessment of Optical Images
200 ## Leave commented!!
201 ## Visualization was necessary to assess image quality
202
203 # Map = geemap.Map()
204 # Map.centerObject(Opt.first().geometry(),7)
205 # for i in range(OptList.size(). getInfo()):
206 #     image = ee.Image(OptList.get(i))
207 #     Map.addLayer(image,RGB_params,'Opt'+str(i+1))
208 #     print('check '+str(i+1))
209
210 # html_file = os.path.join(html_dir+'\QualityAssessment\Training_Validation\Optical_'+S1band
211 #     +'_'+str(dt)+'hrs_'+start[:4]+start[5:7]+start[8:10]+'_+'+_end[:4]+end[5:7]+end[8:10]+'_.
211 #     html')
212
213 indexList = ee.List.sequence(0,OptList.size().subtract(1))
214 poorQualityIndeces_HV_4hrs = ee.List([5,14,17,23,24,28,29,30])
215 goodQualityIndeces_HV_4hrs = indexList.removeAll(poorQualityIndeces_HV_4hrs)
216
217 qOptList_HV_4hrs = goodQualityIndeces_HV_4hrs.map(lambda index: OptList.get(index))
218 qOpt_HV_4hrs = ee.ImageCollection(qOptList_HV_4hrs)
219
220 qOpt = qOpt_HV_4hrs
221 qOptList = qOpt.toList(qOpt.size())

```

```

222
223 ## Leave commented!!
224 ## Visualization was necessary to assess image quality
225
226 # Map = geemap.Map()
227 # Map.centerObject(qOpt.first().geometry(), 7)
228 # for i in range(qOptList.size().getInfo()):
229 #     image = ee.Image(qOptList.get(i))
230 #     Map.addLayer(image,RGB_params,'Opt'+str(i+1))
231 #     print('check '+str(i+1))
232
233 # html_file = os.path.join(html_dir+'\QualityAssessment\Training_Validation\OpticalQuality_'+
234 #     S1band+'_'+str(dt)+'hrs_'+start[:4]+start[5:7]+start[8:10]+'_end[:4]+end[5:7]+end
235 #     [8:10]+'.html')
236 # Map.to_html(outfile=html_file, title= 'Opt images', width='100%', height='880px')
237
238 #%% (3) Visualization of Match Data Set
239 # (3.1) Functions
240 def findMatches(image,imCol,hours):
241     utc = image.get('system:time_start')
242     temporalFilter = ee.Filter.maxDifference(
243         difference = hours * 60 * 60 * 1000,
244         leftField = 'system:time_start',
245         rightValue = utc)
246
247     names = ee.List(image.get('NAME'))
248     segs = ee.List(image.get('SEG'))
249     indeces = ee.List.sequence(0,names.size().subtract(1))
250     filtered = ee.FeatureCollection(indeces.map(lambda i: ROIs.filterMetadata('NAME','
251         equals',names.get(i)).filter(ee.Filter.inList('SEG',segs.get(i)))).flatten())
252     geometry = filtered.geometry()
253     filteredCol = imCol.filterBounds(geometry).filter(temporalFilter)
254
255     return image.set('matches',filteredCol.size()).set('utcs',filteredCol.aggregate_array
256         ('system:time_start')).set('iscl',geometry)
257
258 def addMatchesAsBands(image,imCol,data):
259     utcs = ee.List(image.get('utcs'))
260     matches = imCol.filter(ee.Filter.inList('system:time_start',utcs))
261     matchList = matches.toList(matches.size())
262
263     def addMatches(match,iterations):
264         iteration = ee.List(iterations)
265         image = ee.Image(iteration.get(0))
266         namelist = ee.List(iteration.get(1))
267
268         matched = ee.Image(match)
269         id = ee.String(matched.get('id'))
270         utc = ee.String(matched.get('system:time_start'))
271         fp = matched.geometry()
272
273         index = matchList.indexOf(matched)
274         name = ee.String(namelist.get(index))
275         idname = ee.String(data).cat(ee.String('id')).cat(name)
276         utcname = ee.String(data).cat(ee.String('utc')).cat(name)
277         fpname = ee.String(data).cat(ee.String('fp')).cat(name)
278
279         bandnames = matched.bandNames()
280         updatednames = bandnames.map(lambda bandname: ee.String(bandname).cat(name))
281
282         bandsAdded = image
283             .addBands(matched.select(bandnames,updatednames))
284             .set(idname,id)
285             .set(utcname,utc)
286             .set(fpname,fp)
287
288     return ee.List([bandsAdded,namelist])
289
290 S1nameList = ee.List(['_1','_2','_3','_4','_5','_6','_7','_8','_9','_10'])
291 iterList = ee.List([image,S1nameList])

```

```

289     matchesAdded = ee.Image(ee.List(matchList.iterate(
290         function = addMatches,
291         first = iterList
292     )).get(0))
293
294     return matchesAdded
295
296 def filterAllMatches(imList,imCol,hours):
297     dataToMatch = ee.String(imCol.aggregate_array('data').distinct().get(0))
298     matchesCol = ee.ImageCollection(imList)
299         .map(lambda image: findMatches(image,imCol,hours))
300         .filterMetadata('matches','greater_than',0)
301         .map(lambda image: addMatchesAsBands(image,imCol,dataToMatch))
302
303     return matchesCol
304
305 def mapS1Opt(S1Opt,S1band,dt,region):
306     geometry = S1Opt.first().geometry()
307
308     Map = geemap.Map()
309     Map.centerObject(geometry,6)
310
311     S1OptList = S1Opt.toList(S1Opt.size())
312
313     for i in range(S1OptList.size(). getInfo()):
314         image = ee.Image(S1OptList.get(i))
315         utc = ee.Date(image.get('system:time_start'))
316         id = ee.String(image.get('id')).getInfo()
317         bands = image.bandNames()
318         number_of_matches = bands.filter(ee.Filter.stringContains('item','Red')).size().
319             getInfo()
320
321         HHname = str(i+1) + '_S1_HH | Time (local): ' + id[-18:]
322         Map.addLayer(image.select('HH'),S1_HH_params,HHname)
323         print('S1 HH image ' + str(i+1) + ' added')
324
325         for k in range(number_of_matches):
326             utcname = image.propertyNames().filter(ee.Filter.stringEndsWith('item','utc_'
327                 +str(k+1))).get(0).getInfo()
328             Optutc = ee.Date(image.get(utcname))
329             timeDiff = int(np.ceil(utc.difference(Optutc,'minute').getInfo()))
330             name = str(i+1) + '_'+str(k+1) + '_'+utcname[:2] + ' | ' + '\u0394' + 't = ' + str(
331                 timeDiff) + 'min'
332             Map.addLayer(image.select(bands.filter(ee.Filter.stringEndsWith('item',str(k
333                 +1))),RGBbands),RGB_params,name)
334             print(utcname[:2] + ' image ' + str(k+1) + ' added')
335
336         names = ee.List(image.get('NAME'))
337         segs = ee.List(image.get('SEG'))
338         indeces = ee.List.sequence(0,names.size().subtract(1))
339         filtered = ee.FeatureCollection(indeces
340             .map(lambda i: ROIs
341                 .filterBounds(trainValArea)
342                 .filterMetadata('NAME','equals',names.get(i))
343                 .filter(ee.Filter.inList('SEG',segs.get(i))))
344             .flatten())
345         coastlinegeometry = filtered.geometry()
346
347         Map.addLayer(coastlinegeometry,{},str(i+1) + '_IceShelfCoastLine',opacity=0.25)
348
349         Map.addLayer(GroundingLine.filterBounds(trainValArea).geometry(),{},'Grounding Line',
350             opacity=0.15)
351         Map.addLayer(IceShelves.filterBounds(trainValArea).geometry(),{},'Ice Shelves',opacity
352             =0.15)
353
354         html_file = os.path.join(html_dir+'\QualityAssessment\Training_Validation\S1OptMatches_'
355             +region+'_'+S1band+'_'+str(dt)+'hrs_'+start[:4]+start[5:7]+start[8:10]+'_'
356             +end[4:7]+end[8:10]+'.html')
357         Map.to_html(outfile=html_file, title='S1 & Opt Matches', width='100%', height='880px')
358         print('Map finished')
359

```

```

352     return Map
353
354 ## (3.2) Create Reference Data Set
355 S1OptDt = filterAllMatches(S1,qOpt,dt).sort('WB')
356 S1OptList = S1OptDt.toList(S1OptDt.size())
357
358 TrainingData = filterAllMatches(S1,qOpt.filterBounds(trainingArea),dt)
359 ValidationData = filterAllMatches(S1,qOpt.filterBounds(validationArea),dt)
360
361 ## (3.3) Visualize & Select Quality Matches
362 ## Leave commented!!
363 ## Used to visualize data set
364
365 # QualityMap = mapS1Opt(S1OptDt,S1band,dt,'Training&Validation')
366
367 ## Poor quality indeces: [0,3,10,14,15,18,25]
368
369 ## (3.4) Manual Selection of Quality Matches
370 indexList = ee.List.sequence(0,S1OptList.size().subtract(1))
371 poorQualityIndeces_HV_Opt_4hrs = ee.List([0,3,10,14,15,18,25])
372 goodQualityIndeces_HV_Opt_4hrs = indexList.removeAll(poorQualityIndeces_HV_Opt_4hrs)
373
374 qS1OptList_HV_4hrs = goodQualityIndeces_HV_Opt_4hrs.map(lambda index: ee.Image(S1OptList.get(index)))
375 qS1Opt_HV_4hrs = ee.ImageCollection(qS1OptList_HV_4hrs)
376
377 matchIndex = ee.List.sequence(0,qS1Opt_HV_4hrs.size().subtract(1))
378 S1_utc = qS1Opt_HV_4hrs.aggregate_array('system:time_start').flatten().distinct()
379 Opt_utcs = qS1Opt_HV_4hrs.aggregate_array('utcs')
380
381 S1Quality = ee.ImageCollection(
382     matchIndex.map(lambda i: S1\
383         .filter(ee.Filter.inList('system:time_start',ee.List([S1_utc.get(i)]))\
384         .toList(ee.List([S1_utc.get(i)]).size())\
385         .map(lambda image: ee.Image(image).set('match',ee.Number(i).add(1))))
386     ).flatten()
387 )
388 OptQuality = ee.ImageCollection(
389     matchIndex.map(lambda i: Opt\
390         .filter(ee.Filter.inList('system:time_start',ee.List(Opt_utcs.get(i))))\
391         .toList(ee.List(Opt_utcs.get(i)).size())\
392         .map(lambda image: ee.Image(image).set('match',ee.Number(i).add(1))))
393     ).flatten()
394 )
395
396 ## Leave commented!!
397 ## Used to visualize data set
398
399 # RefMap = mapS1Opt(qS1Opt_HV_4hrs,S1band,dt,'Reference_Data')
400
401 #%% (4) Store Training & Validation Data Set
402 S1_FC = ee.FeatureCollection(S1Quality.toList(S1Quality.size()).map(lambda image:
403     ee.Feature(ee.Image(image).geometry(),{
404         'utc': ee.Image(image).get('system:time_start'),
405         'data': ee.Image(image).get('data'),
406         'match': ee.Image(image).get('match')})
407 ))
408 Opt_FC = ee.FeatureCollection(OptQuality.toList(OptQuality.size()).map(lambda image:
409     ee.Feature(ee.Image(image).geometry(),{
410         'utc': ee.Image(image).get('system:time_start'),
411         'data': ee.Image(image).get('data'),
412         'match': ee.Image(image).get('match')})
413 ))
414
415 referenceData = S1_FC.merge(Opt_FC)
416 ## Leave commented!!
417 ## Export was necessary to create feature collections used in section below
418
419 # geemap.ee_export_vector_to_drive(
420 #     ee_object = referenceData,
421 #     description = 'RD_'+S1band+'_'+str(dt)+'hrs_'+start[:4]+start[5:7]+start[8:10]+'_'+end

```

```

422 [:4]+end[5:7]+end[8:10],
423 #     folder = 'TrainingValidation',
424 #     file_format = 'shp')

```

B.5. Training & Validation Data

5_TrainingValidationData

```

1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5 import numpy as np
6 import os
7
8 ## (1.2) Initialize
9 ee.Initialize()
10
11 ## (1.3) Parameterization & Data Import
12 start = '2014-01-01'
13 end = '2022-04-01'
14 crs = 'EPSG:3031'
15 S1band = 'HV'
16 dt = 4
17 elevation_threshold = 200
18
19 ReferenceData = ee.FeatureCollection('users/skjeltnmaps/RD_'+S1band+'_'+str(dt)+'hrs_'+start
20 [:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+end[8:10])
21 S1OptMetadata = ee.FeatureCollection('users/skjeltnmaps/S1Opt_'+S1band+'_'+str(dt)+'
```

```
58 'bands': S2bands}
59
60 RGBbands = ['Red','Green','Blue']
61 RGB_params = {
62   'bands': RGBbands,
63   'min': 0,
64   'max': 3e4}
65
66 S1name = 'S1'
67 HHmin = -35
68 HHmax = 5
69 HVmin = -40
70 HVmax = 0
71
72 S1_HH_params = {
73   'min': HHmin,
74   'max': HHmax}
75 S1_HV_params = {
76   'min': HVmin,
77   'max': HVmax}
78 S1_RGB_params = {
79   'min': [HVmin,HHmin,HHmin],
80   'max': [HVmax,HHmax,HHmax]}
81 angle_params = {
82   'min': 0,
83   'max': 90}
84
85 ## Edit visualization export folder is necessary
86 # html_dir = os.path.join(os.path.expanduser('~'),'Downloads')
87 # if not os.path.exists(html_dir):
88 #     os.makedirs(html_dir)
89
90 #%% (2) Reconstruct Match Data Set
91 ## (2.1) General Functions
92 def utcToLocal(image):
93     centroid = ee.Image(image).geometry(maxError=1e3).centroid(maxError=1e3)
94     longitude = centroid.coordinates().get(0)
95     localTime = ee.Image(image).date().advance(ee.Number(longitude).divide(15).ceil().
96         subtract(1),'hour')
97     localMillis = ee.Date(localTime).millis()
98
99     return image
100    .set('utctime',ee.Image(image).get('system:time_start'))
101    .set('localtime',localMillis)
102
103 def addID(image,name):
104     id = ee.String(name+'_').cat(ee.String(ee.Date(ee.Image(image).get('localtime')).format(
105       'YYYY_MM_dd_KK:mm'))))
106     return image.set('id',id).set('data',name)
107
108 def addRGBbands(image,bandnames):
109     renamed = image.select(bandnames,RGBbands)
110     rgb = image.addBands(renamed).select(RGBbands)
111     return rgb
112
113 def addWB(image):
114     wb = ee.List(ee.List(image.geometry(1e4).bounds(1e4).coordinates().get(0)).get(0)).
115       get(0)
116     return image.set('WB',wb)
117
118 def addMetadata(image,utcname,metadata):
119     utc = ee.String(image.get('system:time_start'))
120     filtered = metadata.filter(ee.Filter.inList(utcname,[utc]))
121     name = filtered.aggregate_array('NAME').distinct()
122     segment = name.map(lambda name: filtered.filterMetadata('NAME','equals',name).
123       aggregate_array('SEG'))
124     wb = ee.List(ee.List(image.geometry(1e4).bounds(1e4).coordinates().get(0)).get(0)).
125       get(0)
126     return image.set('NAME',name).set('SEG',segment).set('WB',wb)
127
128 def addMatchIndex(image,metadata):
```

```

124     utc = image.get('system:time_start')
125     match = metadata.filterMetadata('utc','equals',utc).first()
126     matchIndex = ee.Number(match.get('match'))
127     return image.set('match',matchIndex)
128
129 def angleNormalization(image):
130     def degreesToRadians(object):
131         return object.multiply(np.pi).divide(180)
132
133     thetaRef = degreesToRadians(ee.Number(30))
134     radians = degreesToRadians(image.select('angle'))
135
136     angleCorr = image.select('angle').divide(thetaRef.cos().pow(ee.Number(2))).multiply(
137         radians.cos().pow(ee.Number(2)))
138     HH = image.select('HH').divide(thetaRef.cos().pow(ee.Number(2))).multiply(radians.cos() .
139         pow(ee.Number(2)))
140     HV = image.select('HV').divide(thetaRef.cos().pow(ee.Number(2))).multiply(radians.cos() .
141         pow(ee.Number(2)))
142
143     normalized = image.addBands(HH,overwrite=True).addBands(HV,overwrite=True).addBands(
144         angleCorr)
145
146     return normalized
147
148 ## (2.2) Reconstruction
149 S1 = ee.ImageCollection("COPERNICUS/S1_GRD")
150     .filter(ee.Filter.inList('system:time_start',S1_UTC))
151     .filter(ee.Filter.listContains('transmitterReceiverPolarisation',S1band))
152     .map(utcToLocal)
153     .map(angleNormalization)
154     .map(lambda image: image.updateMask(DEM.unmask()).gt(elevation_threshold).Not())
155     .map(lambda image: addID(image,S1name))
156     .map(lambda image: addMetadata(image,'S1_UTC',S1OptMetadata))
157     .map(lambda image: addMatchIndex(image,ReferenceData))
158     .sort('WB')
159
160 L8 = ee.ImageCollection("LANDSAT/LC08/C01/T2")
161     .filter(ee.Filter.inList('system:time_start',Opt_UTC))
162     .map(lambda image: addRGBbands(image,L8bands))
163     .map(utcToLocal)
164     .map(lambda image: addID(image,L8name))
165     .map(lambda image: addMatchIndex(image,ReferenceData))
166     .map(lambda image: addMetadata(image,'Opt_UTC',S1OptMetadata))
167 S2 = ee.ImageCollection("COPERNICUS/S2")
168     .filter(ee.Filter.inList('system:time_start',Opt_UTC))
169     .map(lambda image: addRGBbands(image,S2bands))
170     .map(utcToLocal)
171     .map(lambda image: addID(image,S2name))
172     .map(lambda image: addMatchIndex(image,ReferenceData))
173     .map(lambda image: addMetadata(image,'Opt_UTC',S1OptMetadata))
174 Opt = L8.merge(S2).sort('WB')
175 OptList = Opt.toList(Opt.size())
176
177 %% (3) Create Training & Validation Data Set
178 ## (3.1) Match functions
179 def matchDataSets(S1,Opt):
180     def addMatchesAsBands(imgS1,Opt):
181         matchIndex = ee.Number(imgS1.get('match'))
182         OptCol = Opt.filterMetadata('match','equals',matchIndex)
183         OptList = OptCol.toList(OptCol.size())
184         GeomList = OptList.map(lambda image: ee.Image(image).geometry())
185
186         def combineGeometries(geom2,geom1):
187             geom = ee.Geometry(geom1).union(ee.Geometry(geom2),maxError=1e3)
188             return geom
189
190         OptGeometry = ee.Geometry(GeomList.iterate(
191             function = combineGeometries,
192             first = ee.Geometry(GeomList.get(0))
193         ))
194         S1Geometry = imgS1.geometry()

```

```

191
192     OptImage = OptCol.median().clip(S1Geometry)
193     S1Image = imgS1.clip(OptGeometry)
194     matchAdded = S1Image.addBands(OptImage)
195
196     return matchAdded
197
198 matchesOfInterest = Opt.aggregate_array('match').distinct().sort()
199 S1Filtered = S1.filter(ee.Filter.inList('match',matchesOfInterest))
200 S1Opt = S1Filtered.map(lambda imgS1: addMatchesAsBands(imgS1,Opt))
201
202     return S1Opt
203
204 ## (3.2) Create Reference Data Set
205 Opt_Amundsen = Opt.filterBounds(AmundsenEmbayment)
206 Opt_Carney = Opt.filterBounds(CarneyCoast)
207
208 AmundsenMatches = Opt_Amundsen.aggregate_array('match').distinct().sort()
209 CarneyMatches = Opt_Carney.aggregate_array('match').distinct().sort()
210
211 S1_Amundsen = S1.filter(ee.Filter.inList('match',AmundsenMatches))
212 Opt_Amundsen = Opt.filter(ee.Filter.inList('match',AmundsenMatches))
213 S1Opt_Amundsen = matchDataSets(S1_Amundsen,Opt_Amundsen)
214
215 S1_Carney = S1.filter(ee.Filter.inList('match',CarneyMatches))
216 Opt_Carney = Opt.filter(ee.Filter.inList('match',CarneyMatches))
217 S1Opt_Carney = matchDataSets(S1_Carney,Opt_Carney)
218
219 trainIDsAmundsen = ee.List([1,2,5,6,7,8])
220 trainIDsCarney = ee.List([1,4,11])
221
222 S1Training = ee.ImageCollection(trainIDsAmundsen.map(lambda index: ee.Image(S1_Amundsen.
223     toList(S1_Amundsen.size()).get(index))))
224     .merge(ee.ImageCollection(trainIDsCarney.map(lambda index: ee.Image(S1_Carney.toList(
225         S1_Carney.size()).get(index)))))
226 OptTraining = Opt.filter(ee.Filter.inList('match',S1Training.aggregate_array('match').
227     distinct().sort()))
228 TrainingData = ee.ImageCollection(trainIDsAmundsen      .map(lambda index: ee.Image(
229     S1Opt_Amundsen.toList(S1Opt_Amundsen.size()).get(index)))
230     .merge(ee.ImageCollection(trainIDsCarney.map(lambda index: ee.Image(S1Opt_Carney.toList(
231         S1Opt_Carney.size()).get(index)))))
232
233 valIDsAmundsen = ee.List([0,3,4])
234 valIDsCarney = ee.List([10])
235
236 S1Validation = ee.ImageCollection(valIDsAmundsen.map(lambda index: ee.Image(S1_Amundsen.
237     toList(S1_Amundsen.size()).get(index)))
238     .merge(ee.ImageCollection(valIDsCarney.map(lambda index: ee.Image(S1_Carney.toList(
239         S1_Carney.size()).get(index)))))
240 OptValidation = Opt.filter(ee.Filter.inList('match',S1Validation.aggregate_array('match').
241     distinct().sort()))
242 ValidationData = ee.ImageCollection(valIDsAmundsen      .map(lambda index: ee.Image(
243     S1Opt_Amundsen.toList(S1Opt_Amundsen.size()).get(index)))
244     .merge(ee.ImageCollection(valIDsCarney.map(lambda index: ee.Image(S1Opt_Carney.toList(
245         S1Opt_Carney.size()).get(index)))))
246
247 #%% (3) Creation of Sampling Polygons
248 ## (3.1) Training Polygons
249 OW1aT = ee.Geometry.LineString(
250     geodesic = False,
251     proj = 'EPSG:4326',
252     coords = [[-114.226429, -73.138338], [-113.743186, -73.071926], [-113.435668,
253         -73.217192], [-113.374165, -73.371302], [-113.330234, -73.543974], [-113.980414,
254         -73.308308], [-114.226429, -73.138338]])
255 )
256 OW1bT = ee.Geometry.LineString(
257     geodesic = False,
258     proj = 'EPSG:4326',
259     coords = [[-105.2167, -74.145339], [-105.01242, -74.206453], [-105.315545, -74.179519],
260         [-105.434159, -74.162138], [-105.649421, -74.163337], [-105.447338, -74.150141],
261         [-105.276007, -74.154341], [-105.330921, -74.138134], [-105.223289, -74.13453],
262         [-105.198141, -74.138134], [-105.198141, -74.145339], [-105.2167, -74.145339]]
```

```

        [-105.2167, -74.145339]]
248 )
249 OW1cT = ee.Geometry.LineString(
250     geodesic = False,
251     proj = 'EPSG:4326',
252     coords = [[-110.213527, -72.875053], [-110.086127, -72.889282], [-110.125665,
253         -72.917704], [-110.222314, -72.940925], [-110.323355, -72.957677], [-110.406824,
254         -72.969263], [-110.42879, -73.000128], [-110.367286, -73.025805], [-110.437576,
255         -73.041193], [-110.49908, -73.005265], [-110.406824, -72.947369], [-110.296997,
256         -72.933187], [-110.213527, -72.875053]]]
257 )
258 DW1aT = ee.Geometry.LineString(
259     geodesic = False,
260     proj = 'EPSG:4326',
261     coords = ee.List([OW1aT,OW1bT,OW1cT]))
262 )
263 DW1aT = ee.Geometry.LineString(
264     geodesic = False,
265     proj = 'EPSG:4326',
266     coords = [[-106.927888, -74.224536], [-107.059681, -74.291298], [-106.796094,
267         -74.369635], [-105.50452, -74.5017], [-105.346367, -74.452303], [-104.915842,
268         -74.454661], [-105.003704, -74.279395], [-105.645099, -74.226921], [-105.8999,
269         -74.262718], [-106.664301, -74.272254], [-106.822453, -74.229313], [-106.927888,
270         -74.224536]]]
271 DW1bT = ee.Geometry.LineString(
272     geodesic = False,
273     proj = 'EPSG:4326',
274     coords = [[-113.177057, -73.116974], [-112.948616, -73.34015], [-113.05405, -73.683134],
275         [-112.53127, -73.927035], [-111.33195, -73.973205], [-111.441778, -73.836756],
276         [-112.149069, -73.660895], [-112.280863, -73.429366], [-112.026062, -73.328806],
277         [-111.551606, -73.235262], [-111.850337, -73.118251], [-113.177057, -73.116974]]]
278 )
279 DW1cT = ee.Geometry.LineString(
280     geodesic = False,
281     proj = 'EPSG:4326',
282     coords = [[-110.442138, -73.215918], [-110.398207, -73.200685], [-110.422369, -73.17908],
283         [-110.464103, -73.191156], [-110.514624, -73.189249], [-110.551966, -73.194969],
284         [-110.53, -73.211477], [-110.442138, -73.215918]]]
285 )
286 DW1T = ee.Geometry.MultiLineString(
287     geodesic = False,
288     proj = 'EPSG:4326',
289     coords = ee.List([DW1aT,DW1bT,DW1cT]))
290 )
291 FI1aT = ee.Geometry.LineString(
292     geodesic = False,
293     proj = 'EPSG:4326',
294     coords = [[-110.395281, -73.919426], [-110.118515, -73.916992], [-110.131695,
295         -73.882869], [-109.920825, -73.809511], [-109.876894, -73.83767], [-109.749494,
296         -73.803383], [-109.841749, -73.773939], [-109.775853, -73.748131], [-109.872501,
297         -73.693931], [-109.96915, -73.70873], [-110.030653, -73.687761], [-110.008688,
298         -73.628409], [-109.947184, -73.588725], [-109.850536, -73.592447], [-109.868108,
299         -73.566361], [-109.929612, -73.571332], [-109.837356, -73.514065], [-109.846142,
300         -73.492849], [-109.95597, -73.512817], [-109.894467, -73.541478], [-109.96915,
301         -73.541478], [-110.021867, -73.582516], [-110.004295, -73.609818], [-110.118515,
302         -73.609818], [-110.329385, -73.625932], [-110.311812, -73.655637], [-110.390888,
303         -73.665528], [-110.346957, -73.702565], [-110.197591, -73.654399], [-110.158054,
304         -73.675413], [-110.250309, -73.691461], [-110.092157, -73.778849], [-110.390888,
305         -73.810737], [-110.29424, -73.840116], [-110.201985, -73.833999], [-110.197591,
306         -73.864559], [-110.329385, -73.873106], [-110.412854, -73.895063], [-110.395281,
307         -73.919426]]]
308 )
309 FI1bT = ee.Geometry.LineString(
310     geodesic = False,
311     proj = 'EPSG:4326',
312     coords = [[-107.364818, -73.575374], [-107.795343, -73.66831], [-107.518577, -73.695475],
313         [-107.584473, -73.781614], [-107.298921, -73.769335], [-107.11441, -73.829412],
314         [-107.022155, -73.785295], [-107.136376, -73.755817], [-107.017762, -73.729983],
315         [-106.66192, -73.84287], [-106.437871, -73.828188], [-106.560878, -73.784068],
316         [-106.679492, -73.800014], [-106.692672, -73.785295], [-106.569665, -73.74352],
```

```

    [-106.727816, -73.679428], [-106.762961, -73.529336], [-107.022155, -73.486921],
    [-107.364818, -73.575374]]]

288 )
289 FI1cT = ee.Geometry.LineString(
290     geodesic = False,
291     proj = 'EPSG:4326',
292     coords = [[-105.619598, -73.844705], [-105.364798, -73.930076], [-105.404336,
293         -73.973812], [-105.38237, -74.000481], [-105.72064, -73.992], [-105.707461,
294         -74.0742], [-106.137986, -74.030734], [-106.076482, -73.998058], [-106.124806,
295         -73.922775], [-106.274172, -73.900857], [-106.287352, -73.847149], [-106.11602,
296         -73.839814], [-105.79093, -73.810432], [-105.619598, -73.844705]]]
297 )
298 FI1dT = ee.Geometry.LineString(
299     geodesic = False,
300     proj = 'EPSG:4326',
301     coords = [[-106.9925, -74.485839], [-107.194583, -74.521063], [-107.273659, -74.602949],
302         [-107.38788, -74.692535], [-106.917817, -74.658853], [-106.983713, -74.609946],
303         [-106.737699, -74.598279], [-106.540009, -74.657692], [-106.966141, -74.749278],
304         [-106.838741, -74.812726], [-106.192954, -74.828837], [-106.197347, -74.635583],
305         [-106.403823, -74.592443], [-106.395037, -74.542157], [-106.9925, -74.485839]]]
306 )
307 FI1eT = ee.Geometry.LineString(
308     geodesic = False,
309     proj = 'EPSG:4326',
310     coords = [[-105.164965, -74.595655], [-104.940916, -74.753615], [-104.391777,
311         -74.858425], [-104.299522, -74.932858], [-103.772348, -74.698046], [-103.750383,
312         -74.586316], [-103.974431, -74.559428], [-104.46646, -74.330813], [-104.690509,
313         -74.214078], [-104.782764, -74.236776], [-104.558715, -74.424305], [-104.699295,
314         -74.550065], [-105.164965, -74.595655]]]
315 )
316 FI1fT = ee.Geometry.LineString(
317     geodesic = False,
318     proj = 'EPSG:4326',
319     coords = [[-109.906178, -73.254106], [-110.042365, -73.249041], [-110.130227,
320         -73.263599], [-110.182944, -73.281306], [-110.160979, -73.309727], [-110.119244,
321         -73.338731], [-110.073116, -73.35951], [-109.947913, -73.365171], [-109.807333,
322         -73.365171], [-109.708488, -73.348808], [-109.717274, -73.31099], [-109.759009,
323         -73.272455], [-109.906178, -73.254106]]]
324 )
325 FI1t = ee.Geometry.MultiLineString(
326     geodesic = False,
327     proj = 'EPSG:4326',
328     coords = ee.List([FI1aT, FI1bT, FI1cT, FI1dT, FI1eT, FI1fT]))
329 )
330 RI1aT = ee.Geometry.LineString(
331     geodesic = False,
332     proj = 'EPSG:4326',
333     coords = [[-109.064168, -73.803383], [-108.831334, -73.713662], [-108.330519,
334         -73.833999], [-108.062539, -73.907249], [-108.365664, -73.913339], [-109.002665,
335         -73.843785], [-109.064168, -73.803383]]]
336 )
337 RI1bT = ee.Geometry.LineString(
338     geodesic = False,
339     proj = 'EPSG:4326',
340     coords = [[-105.550729, -74.827975], [-105.910964, -74.837174], [-105.867033,
341         -74.983609], [-105.818709, -75.055171], [-105.884605, -75.133171], [-105.998826,
342         -75.229828], [-106.077902, -75.271226], [-106.490855, -75.281277], [-106.51282,
343         -75.163583], [-106.561145, -75.055171], [-106.569931, -74.99954], [-106.798373,
344         -74.984746], [-106.789586, -75.029085], [-106.829124, -75.153454], [-106.890628,
345         -75.235428], [-107.004849, -75.326987], [-106.736869, -75.406919], [-106.108654,
346         -75.412454], [-105.45408, -75.279045], [-104.724824, -75.109474], [-105.313501,
347         -74.969937], [-105.550729, -74.827975]]]
348 )
349 RI1cT = ee.Geometry.LineString(
350     geodesic = False,
351     proj = 'EPSG:4326',
352     coords = [[-109.578517, -73.807057], [-109.877248, -73.887746], [-109.956324, -73.96319],
353         [-109.912393, -74.045533], [-110.070545, -74.137081], [-109.815745, -74.137081],
354         [-109.587303, -74.17547], [-109.517013, -74.230502], [-109.209495, -74.208993],
355         [-108.963481, -74.19942], [-108.770184, -74.254371], [-108.480239, -74.24483],
356         [-108.3133, -74.290107], [-108.067286, -74.24483], [-107.724623, -74.069674],
```

```

            [-107.891562, -73.997145], [-108.717467, -73.931594], [-109.112847, -73.877989],
            [-109.578517, -73.807057]]
328 )
329 RI1T = ee.Geometry.MultiLineString(
330     geodesic = False,
331     proj = 'EPSG:4326',
332     coords = ee.List([RI1aT,RI1bT,RI1cT]))
333 )
334 ######
335 OW2T = ee.Geometry.LineString(
336     geodesic = False,
337     proj = 'EPSG:4326',
338     coords = [[-103.522965, -73.814411], [-103.426316, -73.815329], [-103.377992,
339         -73.804914], [-103.340651, -73.793266], [-103.330766, -73.780076], [-103.522965,
340         -73.814411]]]
341 )
342 DW2aT = ee.Geometry.LineString(
343     geodesic = False,
344     proj = 'EPSG:4326',
345     coords = [[-106.489038, -74.329625], [-106.910777, -74.387683], [-105.53134, -74.540696],
346         [-105.329256, -74.492596], [-104.841621, -74.45024], [-104.859193, -74.290406],
347         [-105.048097, -74.175775], [-105.360008, -74.138583], [-105.78614, -74.212879],
348         [-105.799319, -74.291597], [-106.286955, -74.345048], [-106.489038, -74.329625]]]
349 )
350 DW2bT = ee.Geometry.LineString(
351     geodesic = False,
352     proj = 'EPSG:4326',
353     coords = ee.List([DW2aT,DW2bT]))
354 )
355 FI2aT = ee.Geometry.LineString(
356     geodesic = False,
357     proj = 'EPSG:4326',
358     coords = [[-104.511633, -74.550945], [-105.10031, -74.616366], [-104.432557, -74.787959],
359         [-104.344695, -74.909709], [-104.195329, -75.025967], [-103.184914, -74.834014],
360         [-102.824679, -74.861581], [-102.271146, -74.614032], [-103.017975, -74.447597],
361         [-103.184914, -74.250801], [-104.019605, -74.326958], [-103.905384, -74.421652],
362         [-103.606652, -74.445238], [-103.360638, -74.534549], [-104.010819, -74.618694],
363         [-104.511633, -74.550945]]]
364 )
365 FI2bT = ee.Geometry.LineString(
366     geodesic = False,
367     proj = 'EPSG:4326',
368     coords = [[-102.359576, -74.126268], [-102.886749, -74.141887], [-102.741777,
369         -74.255567], [-102.482583, -74.304382], [-102.293679, -74.265104], [-102.100382,
370         -74.279398], [-101.81483, -74.274633], [-101.88512, -74.182663], [-102.183851,
371         -74.129874], [-102.359576, -74.126268]]]
372 )
373 FI2cT = ee.Geometry.LineString(
374     geodesic = False,
375     proj = 'EPSG:4326',
376     coords = ee.List([FI2aT,FI2bT,FI2cT]))
377 )
378 RI2T = ee.Geometry.LineString(
379     geodesic = False,
380     proj = 'EPSG:4326',
381     coords = [[-105.768063, -74.835165], [-105.96136, -74.956546], [-105.205745, -75.290491],
382         [-105.056379, -75.239073], [-104.028391, -75.18523], [-105.276034, -75.040719],
```

```
        [-105.504477, -74.890245], [-105.768063, -74.835165]]  
379 )  
380 ###  
381 OW3aT = ee.Geometry.LineString(  
382     geodesic = False,  
383     proj = 'EPSG:4326',  
384     coords = [[-110.099129, -74.210488], [-110.092539, -74.20092], [-110.046412, -74.199724],  
385     [-110.031036, -74.220052], [-110.068377, -74.224234], [-110.099129, -74.210488]]  
386 )  
386 OW3bT = ee.Geometry.LineString(  
387     geodesic = False,  
388     proj = 'EPSG:4326',  
389     coords = [[-110.150205, -74.383024], [-110.126043, -74.390712], [-110.11506, -74.371781],  
390     [-110.12165, -74.365861], [-110.141419, -74.363196], [-110.150205, -74.357272],  
391     [-110.168876, -74.362603], [-110.139222, -74.373557], [-110.150205, -74.383024]]  
390 )  
391 OW3cT = ee.Geometry.LineString(  
392     geodesic = False,  
393     proj = 'EPSG:4326',  
394     coords = [[-112.035948, -74.211758], [-112.026064, -74.197703], [-112.000803,  
395     -74.197703], [-111.966757, -74.203386], [-111.93271, -74.213551], [-112.003,  
396     -74.210862], [-112.035948, -74.211758]]  
395 )  
396 OW3T = ee.Geometry.MultiLineString(  
397     geodesic = False,  
398     proj = 'EPSG:4326',  
399     coords = ee.List([OW3aT,OW3bT,OW3cT]))  
400 )  
401 DW3aT = ee.Geometry.LineString(  
402     geodesic = False,  
403     proj = 'EPSG:4326',  
404     coords = [[-113.363953, -74.145191], [-112.784062, -74.177574], [-112.520475,  
405     -74.190747], [-112.898283, -74.046442], [-112.819207, -73.98715], [-111.905439,  
406     -74.046442], [-110.908203, -73.939806], [-111.198149, -73.56481], [-113.271697,  
407     -73.936158], [-113.363953, -74.145191]]  
405 )  
406 DW3bT = ee.Geometry.LineString(  
407     geodesic = False,  
408     proj = 'EPSG:4326',  
409     coords = [[-110.448954, -74.118526], [-110.657627, -74.128745], [-110.870693,  
410     -74.138657], [-111.017862, -74.220424], [-110.875086, -74.277089], [-110.727917,  
411     -74.237445], [-110.777339, -74.222217], [-110.765258, -74.172552], [-110.708148,  
412     -74.166258], [-110.660922, -74.183336], [-110.654332, -74.227592], [-110.463232,  
413     -74.244307], [-110.381959, -74.27292], [-110.514851, -74.200693], [-110.312768,  
414     -74.165359], [-110.339127, -74.158762], [-110.329242, -74.147062], [-110.340225,  
415     -74.136555], [-110.426989, -74.135354], [-110.448954, -74.118526]]  
410 )  
411 DW3cT = ee.Geometry.LineString(  
412     geodesic = False,  
413     proj = 'EPSG:4326',  
414     coords = [[-110.098604, -74.299474], [-110.094211, -74.288173], [-110.130454,  
415     -74.284602], [-110.149124, -74.275673], [-110.195252, -74.272993], [-110.199645,  
416     -74.258097], [-110.255657, -74.252134], [-110.26664, -74.266441], [-110.251264,  
417     -74.282818], [-110.207333, -74.2849], [-110.20953, -74.292635], [-110.318259,  
418     -74.287282], [-110.340225, -74.310769], [-110.328144, -74.324429], [-110.294097,  
419     -74.344898], [-110.302883, -74.32799], [-110.263345, -74.318491], [-110.218316,  
420     -74.324725], [-110.19635, -74.318787], [-110.20953, -74.307797], [-110.135945,  
421     -74.292933], [-110.124962, -74.301258], [-110.098604, -74.299474]]  
415 )  
416 DW3T = ee.Geometry.MultiLineString(  
417     geodesic = False,  
418     proj = 'EPSG:4326',  
419     coords = ee.List([DW3aT,DW3bT,DW3cT]))  
420 )  
421 FI3aT = ee.Geometry.LineString(  
422     geodesic = False,  
423     proj = 'EPSG:4326',  
424     coords = [[-107.834207, -74.698189], [-107.950624, -74.703986], [-107.937445,  
425     -74.738148], [-107.821027, -74.752601], [-107.752934, -74.772235], [-107.741951,  
424     -74.795303], [-107.684841, -74.757223], [-107.834207, -74.698189]]  
425 )
```

```

426 FI3bT = ee.Geometry.LineString(
427     geodesic = False,
428     proj = 'EPSG:4326',
429     coords = [[-109.035723, -74.604696], [-109.112602, -74.615777], [-109.099423, -74.62976],
430         [-109.196071, -74.641403], [-109.165319, -74.665828], [-109.092833, -74.651295],
431         [-109.051098, -74.656528], [-108.888553, -74.643732], [-108.921502, -74.620439],
432         [-108.991792, -74.614611], [-109.035723, -74.604696]]
433 )
434 FI3cT = ee.Geometry.LineString(
435     geodesic = False,
436     proj = 'EPSG:4326',
437     coords = [[-109.833072, -74.553429], [-109.995618, -74.555183], [-110.063711,
438         -74.600177], [-109.971455, -74.603677], [-109.804517, -74.622333], [-109.699083,
439         -74.585583], [-109.61122, -74.566882], [-109.433299, -74.54816], [-109.580468,
440         -74.507715], [-109.657348, -74.529416], [-109.743014, -74.521794], [-109.795731,
441         -74.539376], [-109.714458, -74.549916], [-109.789141, -74.584998], [-109.877004,
442         -74.579742], [-109.833072, -74.553429]]
443 )
444 FI3dT = ee.Geometry.LineString(
445     geodesic = False,
446     proj = 'EPSG:4326',
447     coords = [[-108.780922, -74.367562], [-108.875374, -74.373482], [-108.789708,
448         -74.409546], [-108.743581, -74.444349], [-108.706239, -74.437277], [-108.62277,
449         -74.45142], [-108.581036, -74.446707], [-108.695257, -74.402458], [-108.780922,
450         -74.367562]]
451 )
452 FI3eT = ee.Geometry.LineString(
453     geodesic = False,
454     proj = 'EPSG:4326',
455     coords = [[-110.142535, -74.142711], [-110.179876, -74.145712], [-110.161205,
456         -74.171204], [-110.077736, -74.187078], [-110.060164, -74.175998], [-110.014036,
457         -74.173301], [-109.9734, -74.187378], [-109.910798, -74.196356], [-109.880046,
458         -74.192167], [-109.905307, -74.166708], [-110.01184, -74.153214], [-110.07554,
459         -74.152614], [-110.142535, -74.142711]]
460 )
461 FI3fT = ee.Geometry.LineString(
462     geodesic = False,
463     proj = 'EPSG:4326',
464     coords = ee.List([FI3aT,FI3bT,FI3cT,FI3dT,FI3eT,FI3fT])
465 )
466 RI3aT = ee.Geometry.LineString(
467     geodesic = False,
468     proj = 'EPSG:4326',
469     coords = [[-110.220444, -73.854484], [-110.189692, -74.025896], [-110.378596,
470         -74.071789], [-110.361023, -74.11958], [-110.040327, -74.127172], [-109.811885,
471         -74.130775], [-109.76356, -74.19793], [-109.631767, -74.20511], [-109.403325,
472         -74.172778], [-110.220444, -73.854484]]
473 )
474 RI3bT = ee.Geometry.LineString(
475     geodesic = False,
476     proj = 'EPSG:4326',
477     coords = [[-112.262731, -74.260926], [-112.456028, -74.239457], [-112.710829,
478         -74.232292], [-112.912912, -74.208392], [-113.194071, -74.215567], [-113.431299,
479         -74.246618], [-113.457658, -74.640238], [-113.369795, -74.889669], [-112.412097,
480         -74.983317], [-112.719615, -75.195897], [-111.524689, -75.305537], [-111.111736,
481         -75.184665], [-110.083748, -75.17792], [-109.600506, -75.089975], [-108.98547,
482         -74.869032], [-109.328133, -74.800053], [-109.249057, -74.698332], [-109.52143,
483         -74.642563], [-109.934382, -74.656525], [-110.074962, -74.850666], [-111.03266,
484         -74.95825], [-111.840993, -74.898829], [-112.095793, -74.716877], [-112.051862,
485         -74.258545], [-112.262731, -74.260926]]
486 )
487 RI3cT = ee.Geometry.LineString(
488     geodesic = False,
489     proj = 'EPSG:4326',

```

```

469     coords = [[-108.445819, -74.906555], [-108.705012, -74.929424], [-108.612757, -75.00238],  

470     [-108.454605, -75.063666], [-108.248129, -75.076121], [-108.147087, -75.019429],  

471   RI3T = ee.Geometry.MultiLineString(  

472     geodesic = False,  

473     proj = 'EPSG:4326',  

474     coords = ee.List([RI3aT,RI3bT,RI3cT]))  

475   )  

476  ###  

477 OW4aT = ee.Geometry.LineString(  

478   geodesic = False,  

479   proj = 'EPSG:4326',  

480   coords = [[-111.542252, -73.671708], [-111.357741, -73.629649], [-111.300631,  

481     -73.708731], [-111.107334, -73.759198], [-111.098548, -73.822985], [-111.019472,  

482     -73.880431], [-110.927217, -73.974112], [-111.094155, -74.029825], [-111.089762,  

483     -74.097395], [-111.239127, -74.046744], [-111.072189, -73.924296], [-111.203982,  

484     -73.919428], [-111.300631, -73.88165], [-111.199589, -73.85601], [-111.340169,  

485     -73.821761], [-111.296238, -73.787441], [-111.423638, -73.778851], [-111.533466,  

486     -73.719821], [-111.542252, -73.671708]]  

487 )  

488 OW4bT = ee.Geometry.LineString(  

489   geodesic = False,  

490   proj = 'EPSG:4326',  

491   coords = [[-111.83944, -73.665835], [-111.894354, -73.680968], [-111.840538, -73.682202],  

492     [-111.815278, -73.690535], [-111.851521, -73.726593], [-111.833949, -73.751819],  

493     [-111.852619, -73.769024], [-111.828457, -73.788358], [-111.797705, -73.808897],  

494     [-111.758167, -73.820533], [-111.749381, -73.834303], [-111.691172, -73.830021],  

495     [-111.676895, -73.858145], [-111.656027, -73.830633], [-111.618686, -73.823289],  

496     [-111.627472, -73.789892], [-111.617588, -73.758887], [-111.654929, -73.744132],  

497     [-111.670305, -73.719819], [-111.723022, -73.704413], [-111.83944, -73.665835]]  

498 )  

499 OW4cT = ee.Geometry.LineString(  

500   geodesic = False,  

501   proj = 'EPSG:4326',  

502   coords = [[-109.112746, -74.652494], [-109.112197, -74.655401], [-109.102861, -74.657],  

503     [-109.086936, -74.654966], [-109.112746, -74.652494]]  

504 )  

505 OW4T = ee.Geometry.MultiLineString(  

506   geodesic = False,  

507   proj = 'EPSG:4326',  

508   coords = ee.List([OW4aT,OW4bT,OW4cT]))  

509 )  

510 DW4aT = ee.Geometry.LineString(  

511   geodesic = False,  

512   proj = 'EPSG:4326',  

513   coords = [[-112.456034, -73.784373], [-113.244597, -73.908467], [-113.007369,  

514     -74.031637], [-112.778928, -73.93281], [-112.379155, -73.889577], [-112.346206,  

515     -73.920643], [-112.653724, -74.010467], [-113.242401, -74.081733], [-113.19188,  

516     -74.160488], [-112.752569, -74.108829], [-112.651527, -74.067264], [-112.304472,  

517     -74.011677], [-112.238575, -74.039494], [-112.93708, -74.180266], [-111.922271,  

518     -74.185058], [-111.884929, -74.115446], [-112.113371, -74.08113], [-111.994757,  

519     -73.896282], [-112.456034, -73.784373]]  

520 )  

521 DW4bT = ee.Geometry.LineString(  

522   geodesic = False,  

523   proj = 'EPSG:4326',  

524   coords = [[-112.487978, -73.787131], [-113.226021, -73.910598], [-112.9844, -74.035565],  

525     [-112.791103, -73.923987], [-112.417689, -73.894757], [-112.338612, -73.920338],  

526     [-112.628558, -74.008951], [-113.243594, -74.080225], [-113.18209, -74.155988],  

527     [-112.760351, -74.110332], [-112.690061, -74.069371], [-112.329826, -74.013794],  

528     [-112.255143, -74.035565], [-112.958041, -74.178767], [-112.246357, -74.189545],  

529     [-112.211212, -74.075401], [-112.140922, -73.970165], [-112.022308, -73.897194],  

530     [-112.487978, -73.787131]]  

531 )  

532 DW4cT = ee.Geometry.LineString(  

533   geodesic = False,  

534   proj = 'EPSG:4326',  

535   coords = [[-110.005993, -74.182512], [-110.111428, -74.18311], [-110.07848, -74.213626],  

536     [-110.069694, -74.242294], [-110.027959, -74.264357], [-110.034549, -74.285795],  

537     [-109.94449, -74.284604], [-109.896166, -74.269123], [-109.753389, -74.263165],  

538     [-109.753389, -74.263165]]  

539 )

```

```

[-109.713851, -74.240503], [-109.700672, -74.217809], [-109.716048, -74.191494],
[-109.762176, -74.166334], [-109.957669, -74.199871], [-110.005993, -74.182512]]
511 )
512 DW4dT = ee.Geometry.LineString(
513     geodesic = False,
514     proj = 'EPSG:4326',
515     coords = [[-110.658371, -73.950131], [-110.724267, -73.957419], [-110.588081,
516         -74.031335], [-110.487039, -74.01319], [-110.423339, -74.028312], [-110.423339,
517         -74.075402], [-110.390391, -74.122961], [-110.39698, -74.154789], [-110.616636,
518         -74.206301], [-110.618832, -74.227219], [-110.480449, -74.242741], [-110.344263,
519         -74.217661], [-110.377211, -74.19254], [-110.322297, -74.140984], [-110.225649,
520         -74.120555], [-110.201487, -74.085046], [-110.096052, -74.110332], [-110.089463,
521         -74.09107], [-110.170735, -74.078416], [-110.225649, -74.047648], [-110.258597,
522         -74.02166], [-110.26958, -73.970774], [-110.429929, -73.990783], [-110.53756,
523         -73.988966], [-110.59467, -73.951346], [-110.658371, -73.950131]]
524 )
525 DW4eT = ee.Geometry.LineString(
526     geodesic = False,
527     proj = 'EPSG:4326',
528     coords = [[-110.02278, -74.409028], [-110.018387, -74.414638], [-109.995323, -74.41021],
529         [-109.963473, -74.407552], [-109.948097, -74.403122], [-109.922837, -74.403713],
530         [-109.940409, -74.3981], [-109.967866, -74.399577], [-109.982144, -74.404599],
531         [-110.001913, -74.405485], [-110.02278, -74.409028]]
532 )
533 DW4fT = ee.Geometry.LineString(
534     geodesic = False,
535     proj = 'EPSG:4326',
536     coords = [[-110.132608, -74.361048], [-110.180932, -74.363418], [-110.202897,
537         -74.372299], [-110.228158, -74.385315], [-110.216077, -74.395366], [-110.109544,
538         -74.397139], [-110.131509, -74.380879], [-110.132608, -74.361048]]
539 )
540 DW4gT = ee.Geometry.LineString(
541     geodesic = False,
542     proj = 'EPSG:4326',
543     coords = [[-110.035959, -74.306313], [-110.018387, -74.316116], [-110.013994,
544         -74.331849], [-110.059023, -74.322352], [-110.066711, -74.311364], [-110.035959,
545         -74.306313]]
546 )
547 DW4hT = ee.Geometry.LineString(
548     geodesic = False,
549     proj = 'EPSG:4326',
550     coords = [[-110.34787, -74.311289], [-110.27758, -74.348086], [-110.251222, -74.338301],
551         [-110.264401, -74.331478], [-110.217175, -74.325543], [-110.11174, -74.302969],
552         [-110.167753, -74.272622], [-110.24573, -74.274111], [-110.283072, -74.287505],
553         [-110.242435, -74.296428], [-110.273187, -74.304456], [-110.308332, -74.299105],
554         [-110.337986, -74.302078], [-110.34787, -74.311289]]
555 )
556 DW4T = ee.Geometry.MultiLineString(
557     geodesic = False,
558     proj = 'EPSG:4326',
559     coords = ee.List([DW4aT,DW4bT,DW4cT,DW4dT,DW4eT,DW4fT,DW4gT,DW4hT]))
560 )
561 FI4aT = ee.Geometry.LineString(
562     geodesic = False,
563     proj = 'EPSG:4326',
564     coords = [[-109.630382, -74.133626], [-109.74021, -74.135429], [-109.665527, -74.204358],
565         [-109.623793, -74.197778], [-109.590844, -74.203162], [-109.608417, -74.22707],
566         [-109.654544, -74.246171], [-109.738014, -74.281331], [-109.647955, -74.2968],
567         [-109.593041, -74.316414], [-109.520555, -74.272995], [-109.562289, -74.255117],
568         [-109.496392, -74.238414], [-109.452461, -74.221694], [-109.404137, -74.216914],
569         [-109.347027, -74.257503], [-109.28113, -74.243785], [-109.206447, -74.227668],
570         [-109.27454, -74.200769], [-109.360206, -74.210936], [-109.397547, -74.206152],
571         [-109.34483, -74.187601], [-109.434889, -74.173826], [-109.388761, -74.156438],
572         [-109.441478, -74.133026], [-109.496392, -74.141434], [-109.437085, -74.151638],
573         [-109.430496, -74.163036], [-109.520555, -74.180415], [-109.608417, -74.181015],
574         [-109.645758, -74.164835], [-109.634775, -74.154039], [-109.575469, -74.154638],
575         [-109.575469, -74.139033], [-109.630382, -74.133626]]
576 )
577 FI4bT = ee.Geometry.LineString(
578     geodesic = False,
579     proj = 'EPSG:4326',
580     coords = [[-109.630382, -74.133626], [-109.74021, -74.135429], [-109.665527, -74.204358],
581         [-109.623793, -74.197778], [-109.590844, -74.203162], [-109.608417, -74.22707],
582         [-109.654544, -74.246171], [-109.738014, -74.281331], [-109.647955, -74.2968],
583         [-109.593041, -74.316414], [-109.520555, -74.272995], [-109.562289, -74.255117],
584         [-109.496392, -74.238414], [-109.452461, -74.221694], [-109.404137, -74.216914],
585         [-109.347027, -74.257503], [-109.28113, -74.243785], [-109.206447, -74.227668],
586         [-109.27454, -74.200769], [-109.360206, -74.210936], [-109.397547, -74.206152],
587         [-109.34483, -74.187601], [-109.434889, -74.173826], [-109.388761, -74.156438],
588         [-109.441478, -74.133026], [-109.496392, -74.141434], [-109.437085, -74.151638],
589         [-109.430496, -74.163036], [-109.520555, -74.180415], [-109.608417, -74.181015],
590         [-109.645758, -74.164835], [-109.634775, -74.154039], [-109.575469, -74.154638],
591         [-109.575469, -74.139033], [-109.630382, -74.133626]]]

```

```

550     coords = [[-109.803011, -74.523262], [-109.967753, -74.526778], [-110.057812,
551     -74.573606], [-110.051222, -74.59113], [-109.978736, -74.584123], [-109.866711,
552     -74.596967], [-109.750294, -74.593466], [-109.653645, -74.555477], [-109.423007,
553     -74.51681], [-109.335145, -74.488043], [-109.392255, -74.465107], [-109.541621,
554     -74.457457], [-109.655842, -74.456867], [-109.603125, -74.492743], [-109.651449,
555     -74.505662], [-109.697577, -74.497441], [-109.811797, -74.507423], [-109.671218,
556     -74.523262], [-109.776653, -74.563083], [-109.833763, -74.557233], [-109.803011,
557     -74.523262]]
558 )
559 FI4cT = ee.Geometry.LineString(
560     geodesic = False,
561     proj = 'EPSG:4326',
562     coords = [[-108.824854, -74.358901], [-108.856704, -74.37548], [-108.810576, -74.38465],
563     [-108.807281, -74.399134], [-108.913814, -74.41331], [-108.873178, -74.427769],
564     [-108.775431, -74.428063], [-108.688667, -74.43455], [-108.698552, -74.443392],
565     [-108.611788, -74.442213], [-108.824854, -74.358901]]
566 )
567 FI4dT = ee.Geometry.LineString(
568     geodesic = False,
569     proj = 'EPSG:4326',
570     coords = [[-104.002142, -73.351324], [-104.085611, -73.429207], [-103.793469,
571     -73.445492], [-103.644104, -73.412279], [-104.002142, -73.351324]]
572 )
573 FI4T = ee.Geometry.MultiLineString(
574     geodesic = False,
575     proj = 'EPSG:4326',
576     coords = ee.List([FI4aT,FI4bT,FI4cT,FI4dT]))
577 )
578 RI4aT = ee.Geometry.LineString(
579     geodesic = False,
580     proj = 'EPSG:4326',
581     coords = [[-110.506535, -73.745365], [-110.616363, -73.735521], [-110.664687,
582     -73.789587], [-110.493356, -73.801851], [-110.528501, -73.847147], [-110.598791,
583     -73.843481], [-110.677867, -73.877682], [-110.594397, -73.866697], [-110.506535,
584     -73.913035], [-110.431852, -73.913035], [-110.427459, -73.965315], [-110.199017,
585     -73.939803], [-110.155086, -74.033149], [-109.926644, -74.089866], [-109.737741,
586     -74.048857], [-110.506535, -73.745365]])
587 )
588 RI4bT = ee.Geometry.LineString(
589     geodesic = False,
590     proj = 'EPSG:4326',
591     coords = [[-112.241112, -74.254365], [-112.645278, -74.259133], [-112.970368,
592     -74.242435], [-113.242741, -74.204198], [-113.427252, -74.228105], [-113.462397,
593     -74.354226], [-113.532686, -74.519293], [-113.576618, -74.777567], [-114.13015,
594     -74.965655], [-112.987941, -74.913127], [-112.539843, -75.038437], [-112.451981,
595     -75.180724], [-111.142834, -75.232341], [-110.026983, -75.203188], [-109.245009,
596     -75.221135], [-107.821642, -74.974769], [-108.164304, -74.876479], [-108.919919,
597     -75.092796], [-109.491024, -75.137949], [-109.578886, -75.036165], [-108.867202,
598     -74.929131], [-108.946278, -74.82135], [-109.3153, -74.749845], [-109.271368,
599     -74.638488], [-109.912762, -74.629168], [-110.439936, -74.91999], [-111.547,
600     -74.922273], [-112.232325, -74.661753], [-112.074173, -74.448759], [-112.241112,
601     -74.254365]])
602 )
603 RI4cT = ee.Geometry.LineString(
604     geodesic = False,
605     proj = 'EPSG:4326',
606     coords = [[-108.623786, -74.584562], [-108.887372, -74.60674], [-108.975234, -74.626557],
607     [-108.935697, -74.676575], [-108.830262, -74.691665], [-108.641358, -74.675415],
608     [-108.579854, -74.637041], [-108.623786, -74.584562]])
609 )
610 RI4T = ee.Geometry.MultiLineString(
611     geodesic = False,
612     proj = 'EPSG:4326',
613     coords = ee.List([RI4aT,RI4bT,RI4cT]))
614 )
615 #####
616 OW5T = ee.Geometry.LineString(
617     geodesic = False,
618     proj = 'EPSG:4326',
619     coords = [[-105.683101, -74.317753], [-105.549111, -74.35809], [-105.393155, -74.358683],
620     [-105.305293, -74.330814], [-105.199858, -74.299332], [-105.208644, -74.276721],
621     [-105.208644, -74.276721], [-105.199858, -74.299332], [-105.305293, -74.330814], [-105.683101, -74.317753]])

```

```

[-105.162517, -74.236777], [-105.076851, -74.229015], [-104.993382, -74.242745],
[-104.912109, -74.27136], [-104.868178, -74.304682], [-104.876965, -74.359867],
[-104.826444, -74.380587], [-104.782513, -74.360459], [-104.797889, -74.316566],
[-104.833033, -74.29041], [-104.868178, -74.254078], [-104.916503, -74.214675],
[-104.971417, -74.176376], [-105.070262, -74.163788], [-105.177893, -74.156592],
[-105.149338, -74.127772], [-105.235003, -74.123565], [-105.276738, -74.139786],
[-105.360207, -74.148791], [-105.441479, -74.162589], [-105.564487, -74.197931],
[-105.700673, -74.234986], [-105.727032, -74.289219], [-105.683101, -74.317753]]
592 )
593 DW5aT = ee.Geometry.LineString(
594     geodesic = False,
595     proj = 'EPSG:4326',
596     coords = [[-105.733919, -74.005019], [-105.949182, -74.072692], [-106.107334, -74.15569],
597     [-106.322596, -74.251392], [-106.480748, -74.302599], [-106.90688, -74.42224],
598     [-106.915666, -74.471728], [-106.985956, -74.558548], [-106.665259, -74.642568],
599     [-106.515893, -74.639079], [-106.388493, -74.648387], [-106.274272, -74.524582],
600     [-106.006292, -74.510505], [-105.483512, -74.556209], [-105.281429, -74.536303],
601     [-105.272643, -74.506983], [-104.991484, -74.519889], [-104.815759, -74.487015],
602     [-104.872869, -74.430498], [-105.417615, -74.437572], [-106.195196, -74.372595],
603     [-106.107334, -74.301409], [-105.874499, -74.202417], [-105.378077, -74.10522],
604     [-105.430795, -74.066662], [-105.703167, -74.069074], [-105.698774, -74.042516],
605     [-105.641664, -74.019543], [-105.733919, -74.005019]]
606 )
607 DW5bT = ee.Geometry.LineString(
608     geodesic = False,
609     proj = 'EPSG:4326',
610     coords = [[-107.443372, -73.962811], [-107.510367, -73.993134], [-107.430192,
611     -74.006156], [-107.414816, -73.988286], [-107.384065, -73.987983], [-107.366492,
612     -74.002825], [-107.325856, -74.011906], [-107.337937, -74.029448], [-107.320364,
613     -74.037305], [-107.331347, -74.047876], [-107.325856, -74.070808], [-107.296202,
614     -74.056932], [-107.304989, -74.045461], [-107.297301, -74.035794], [-107.24678,
615     -74.031261], [-107.200652, -74.026425], [-107.15123, -74.027634], [-107.077645,
616     -74.046367], [-107.11938, -74.019167], [-107.195161, -74.020982], [-107.279728,
617     -74.023099], [-107.294006, -74.010998], [-107.304989, -73.979192], [-107.357706,
618     -73.97525], [-107.39944, -73.964935], [-107.443372, -73.962811]]
619 )
620 DW5cT = ee.Geometry.LineString(
621     geodesic = False,
622     proj = 'EPSG:4326',
623     coords = [[-107.064764, -74.095024], [-107.049937, -74.096228], [-107.04829, -74.09111],
624     [-107.037856, -74.087495], [-107.042798, -74.082525], [-107.044995, -74.078457],
625     [-107.045544, -74.073031], [-107.055978, -74.080415], [-107.046093, -74.085387],
626     [-107.052134, -74.088399], [-107.064764, -74.095024]]
627 )
628 DW5T = ee.Geometry.MultiLineString(
629     geodesic = False,
630     proj = 'EPSG:4326',
631     coords = ee.List([DW5aT,DW5bT,DW5cT]))
632 )
633 FI5aT = ee.Geometry.LineString(
634     geodesic = False,
635     proj = 'EPSG:4326',
636     coords = [[-104.582923, -74.242744], [-104.727896, -74.259444], [-104.644427,
637     -74.424305], [-104.736682, -74.537184], [-105.145242, -74.561768], [-105.386863,
638     -74.576968], [-105.048593, -74.669028], [-104.951945, -74.782479], [-104.429164,
639     -74.842348], [-104.71911, -75.027384], [-103.919564, -75.071598], [-104.011819,
640     -75.026247], [-103.550542, -74.986455], [-103.511004, -74.92943], [-103.076086,
641     -74.807838], [-103.770198, -74.550065], [-103.906384, -74.559428], [-104.165578,
642     -74.506691], [-104.411592, -74.531325], [-104.446737, -74.503168], [-104.064536,
643     -74.454957], [-104.152398, -74.405416], [-104.288585, -74.398325], [-104.288585,
644     -74.356905], [-104.582923, -74.242744]]
645 )
646 FI5bT = ee.Geometry.LineString(
647     geodesic = False,
648     proj = 'EPSG:4326',
649     coords = [[-107.188039, -74.564401], [-107.394515, -74.649554], [-107.319832,
650     -74.713407], [-107.126535, -74.663507], [-107.016707, -74.679773], [-107.043066,
651     -74.750438], [-106.920059, -74.834589], [-107.047459, -74.90456], [-106.902487,
652     -74.910281], [-106.775086, -74.833439], [-106.62572, -74.842634], [-106.573003,
653     -74.933145], [-106.49832, -74.898837], [-106.458782, -74.85527], [-106.195195,
654     -74.842634], [-106.199589, -74.815034], [-106.037043, -74.694857], [-106.195195,
655     -74.842634]]

```

```

    -74.644901], [-106.388492, -74.700656], [-106.502713, -74.682094], [-106.647686,
    -74.669317], [-106.82341, -74.707613], [-106.90688, -74.698337], [-106.911273,
    -74.635588], [-107.012314, -74.595948], [-107.188039, -74.564401]]
```

622)

```

623 FI5cT = ee.Geometry.LineString(
624     geodesic = False,
625     proj = 'EPSG:4326',
626     coords = [[-107.313711, -73.77041], [-107.493829, -73.812729], [-107.392787, -73.886073],
627         [-107.159953, -73.926275], [-107.058911, -73.993664], [-106.88758, -73.9973],
628         [-106.845845, -73.964556], [-106.540524, -73.9821491], [-106.465841, -73.945122],
629         [-106.503182, -73.864103], [-106.531738, -73.790662], [-106.716248, -73.792503],
630         [-106.832666, -73.763653], [-106.643762, -73.698404], [-106.623993, -73.637234],
631         [-106.751393, -73.637853], [-106.779948, -73.709502], [-106.911742, -73.738445],
632         [-106.971049, -73.725521], [-106.946887, -73.667537], [-107.089663, -73.694086],
633         [-107.036945, -73.734138], [-107.113825, -73.756281], [-107.197294, -73.724903],
634         [-107.144577, -73.668774], [-107.326891, -73.66692], [-107.408163, -73.705804],
635         [-107.320301, -73.742753], [-107.313711, -73.77041]]
```

627)

```

628 FI5dT = ee.Geometry.LineString(
629     geodesic = False,
630     proj = 'EPSG:4326',
631     coords = [[-108.269213, -73.83905], [-108.576731, -73.89522], [-108.458116, -73.884855],
632         [-108.214299, -73.903142], [-108.093488, -73.918974], [-108.084702, -73.89583],
633         [-107.99684, -73.907405], [-107.959498, -73.894611], [-108.135223, -73.865327],
634         [-108.190137, -73.873264], [-108.240657, -73.863495], [-108.18794, -73.846387],
635         [-108.269213, -73.83905]]
```

632)

```

633 FI5eT = ee.Geometry.LineString(
634     geodesic = False,
635     proj = 'EPSG:4326',
636     coords = [[-107.199489, -74.036628], [-107.269779, -74.04146], [-107.25001, -74.061988],
637         [-107.298334, -74.086105], [-107.291745, -74.135433], [-107.18631, -74.109584],
638         [-107.096251, -74.094537], [-107.0655, -74.060782], [-107.148969, -74.040857],
639         [-107.199489, -74.036628]]
```

637)

```

640 FI5T = ee.Geometry.MultiLineString(
641     geodesic = False,
642     proj = 'EPSG:4326',
643     coords = ee.List([FI5aT,FI5bT,FI5cT,FI5dT,FI5eT]))
642 )
```

643 RI5aT = ee.Geometry.LineString(
644 geodesic = False,
645 proj = 'EPSG:4326',
646 coords = [[-110.850815, -74.921715], [-110.23578, -75.189163], [-109.585599, -75.175674],
647 [-108.627901, -75.211614], [-108.276452, -75.312231], [-107.125457, -75.292158],
648 [-106.571925, -75.405535], [-104.313865, -75.108062], [-105.236419, -75.031073],
649 [-105.596654, -74.832285], [-105.93053, -74.813881], [-106.07111, -75.083193],
650 [-106.633428, -74.996982], [-107.055167, -75.144162], [-107.32754, -75.074143],
651 [-107.353899, -75.001531], [-107.591126, -75.008354], [-107.79321, -75.13965],
652 [-108.179804, -75.13289], [-108.241307, -75.060552], [-107.951361, -74.907988],
653 [-108.452177, -74.832285], [-108.803625, -75.087722], [-109.260509, -75.148667],
654 [-109.295654, -75.092244], [-109.172647, -74.981043], [-108.926632, -74.864449],
655 [-109.286868, -74.81618], [-109.357157, -74.737717], [-109.269295, -74.640244],
656 [-109.401089, -74.595946], [-109.550454, -74.721515], [-109.69982, -74.658853],
657 [-109.954621, -74.647222], [-109.708606, -74.742343], [-110.007338, -74.751589],
658 [-110.191849, -74.772383], [-110.174276, -74.850672], [-110.850815, -74.921715]]

647)

```

649 RI5bT = ee.Geometry.LineString(
650     geodesic = False,
651     proj = 'EPSG:4326',
652     coords = [[-110.042483, -74.156893], [-108.680618, -73.939503], [-107.79321, -74.00744],
653         [-107.801996, -74.120862], [-108.1183, -74.293087], [-108.900274, -74.25497],
654         [-109.084785, -74.209595], [-109.488951, -74.24543], [-109.655889, -74.159289],
655         [-110.042483, -74.156893]]
```

652)

```

656 RI5T = ee.Geometry.MultiLineString(
657     geodesic = False,
658     proj = 'EPSG:4326',
659     coords = ee.List([RI5aT,RI5bT]))
```

657)

658 ## #

```

659 OW6aT = ee.Geometry.LineString(
660     geodesic = False,
661     proj = 'EPSG:4326',
662     coords = [[-110.647432, -73.095821], [-110.625467, -73.088153], [-110.592518,
663         -73.079521], [-110.599108, -73.075044], [-110.579339, -73.071206], [-110.59801,
664         -73.069926], [-110.628762, -73.076004], [-110.649629, -73.084637], [-110.647432,
665         -73.095821]]
666 )
667 OW6bT = ee.Geometry.LineString(
668     geodesic = False,
669     proj = 'EPSG:4326',
670     coords = [[-110.621074, -73.101171], [-110.649629, -73.105323], [-110.628762,
671         -73.114898], [-110.633155, -73.124787], [-110.610091, -73.118089], [-110.607894,
672         -73.110749], [-110.621074, -73.101171]]
673 )
674 OW6T = ee.Geometry.MultiLineString(
675     geodesic = False,
676     proj = 'EPSG:4326',
677     coords = ee.List([OW6aT,OW6bT]))
678 )
679 DW6aT = ee.Geometry.LineString(
680     geodesic = False,
681     proj = 'EPSG:4326',
682     coords = [[-111.641307, -73.088549], [-113.337048, -73.555167], [-112.309059,
683         -74.091374], [-111.694024, -74.072089], [-111.483155, -74.031035], [-110.98234,
684         -74.067265], [-110.815401, -73.953477], [-111.325002, -73.639555], [-111.078988,
685         -73.497848], [-111.281071, -73.390155], [-111.316216, -73.238746], [-111.456796,
686         -73.032213], [-111.641307, -73.088549]]
687 )
688 DW6bT = ee.Geometry.LineString(
689     geodesic = False,
690     proj = 'EPSG:4326',
691     coords = [[-110.346504, -72.879498], [-110.302573, -72.880145], [-110.246561,
692         -72.885965], [-110.19604, -72.890168], [-110.152109, -72.899216], [-110.089507,
693         -72.893076], [-110.097195, -72.882732], [-110.059853, -72.879822], [-110.06864,
694         -72.872058], [-110.048871, -72.863644], [-110.020315, -72.86073], [-109.9665,
695         -72.858139], [-109.956615, -72.851014], [-109.939043, -72.845182], [-109.932453,
696         -72.838376], [-109.981876, -72.846479], [-109.995055, -72.854901], [-110.075229,
697         -72.865909], [-110.067541, -72.857492], [-110.090605, -72.859435], [-110.094998,
698         -72.870117], [-110.168583, -72.872381], [-110.210318, -72.87691], [-110.26633,
699         -72.874322], [-110.304769, -72.874646], [-110.346504, -72.879498]])
700 )
701 DW6cT = ee.Geometry.LineString(
702     geodesic = False,
703     proj = 'EPSG:4326',
704     coords = [[-108.922311, -73.030451], [-108.900345, -73.032375], [-108.861906, -73.02147],
705         [-108.870692, -73.00478], [-108.891559, -73.00478], [-108.911328, -72.997392],
706         [-108.911328, -72.99193], [-108.859709, -72.989681], [-108.905837, -72.981966],
707         [-108.944276, -73.000605], [-108.978323, -73.00221], [-109.050809, -73.005743],
708         [-109.072775, -73.003174], [-109.111215, -73.007349], [-109.071677, -73.010559],
709         [-109.050809, -73.02147], [-109.022254, -73.008312], [-108.993699, -73.009275],
710         [-108.936588, -73.014411], [-108.921212, -73.022433], [-108.922311, -73.030451]]
711 )
712 DW6dT = ee.Geometry.LineString(
713     geodesic = False,
714     proj = 'EPSG:4326',
715     coords = [[-109.256187, -73.009516], [-109.233123, -73.004058], [-109.243008,
716         -72.996991], [-109.22873, -72.992493], [-109.211158, -72.997634], [-109.193585,
717         -72.994099], [-109.173816, -72.995706], [-109.188094, -73.001809], [-109.194684,
718         -73.009516], [-109.221042, -73.01401], [-109.256187, -73.009516]]
719 )
720 DW6eT = ee.Geometry.LineString(
721     geodesic = False,
722     proj = 'EPSG:4326',
723     coords = [[-111.713487, -74.195908], [-111.731059, -74.193215], [-111.732706, -74.18648],
724         [-111.71129, -74.185133], [-111.721724, -74.17465], [-111.714585, -74.169406],
725         [-111.706897, -74.167607], [-111.651434, -74.16506], [-111.64759, -74.168207],
726         [-111.671203, -74.172103], [-111.692619, -74.175099], [-111.701406, -74.185582],
727         [-111.715134, -74.189324], [-111.713487, -74.195908]]
728 )
729 DW6fT = ee.Geometry.LineString(

```

```

700     geodesic = False,
701     proj = 'EPSG:4326',
702     coords = [[-107.836978, -73.822372], [-107.86114, -73.831859], [-107.834781, -73.851426],
703         [-107.780966, -73.851426], [-107.739231, -73.883784], [-107.705185, -73.891407],
704         [-107.68871, -73.915165], [-107.684317, -73.880124], [-107.739231, -73.849287],
705         [-107.697497, -73.837364], [-107.641484, -73.837059], [-107.606339, -73.860896],
706         [-107.59975, -73.879819], [-107.607438, -73.900244], [-107.552524, -73.904508],
707         [-107.508593, -73.94193], [-107.481136, -73.973808], [-107.42073, -73.99442],
708         [-107.352637, -74.000478], [-107.22963, -73.985936], [-107.301018, -73.9641],
709         [-107.297723, -73.937371], [-107.353736, -73.938283], [-107.386684, -73.959547],
710         [-107.4405, -73.95317], [-107.453679, -73.940715], [-107.391077, -73.92977],
711         [-107.376799, -73.879819], [-107.449286, -73.866087], [-107.467956, -73.892016],
712         [-107.56131, -73.884699], [-107.562408, -73.874938], [-107.486627, -73.843785],
713         [-107.563507, -73.838893], [-107.579981, -73.822066], [-107.605241, -73.794801],
714         [-107.646976, -73.804916], [-107.643681, -73.814718], [-107.773278, -73.831553],
715         [-107.836978, -73.822372]]
716 )
717 DW6gT = ee.Geometry.LineString(
718     geodesic = False,
719     proj = 'EPSG:4326',
720     coords = [[-106.792938, -73.671474], [-106.79184, -73.682898], [-106.820395, -73.690922],
721         [-106.814903, -73.697091], [-106.764383, -73.681664], [-106.764383, -73.677033],
722         [-106.732533, -73.668076], [-106.731434, -73.662205], [-106.669931, -73.658496],
723         [-106.641375, -73.652004], [-106.606231, -73.650767], [-106.598543, -73.643343],
724         [-106.645769, -73.643962], [-106.665538, -73.642724], [-106.678717, -73.650457],
725         [-106.699584, -73.652004], [-106.729238, -73.656951], [-106.752301, -73.672092],
726         [-106.767677, -73.672709], [-106.792938, -73.671474]]
727 )
728 DW6T = ee.Geometry.MultiLineString(
729     geodesic = False,
730     proj = 'EPSG:4326',
731     coords = ee.List([DW6aT,DW6bT,DW6cT,DW6dT,DW6eT,DW6fT,DW6gT]))
732 )
733 FI6T = ee.Geometry.LineString(
734     geodesic = False,
735     proj = 'EPSG:4326',
736     coords = [[-108.298148, -73.138021], [-108.3289, -73.432499], [-108.175141, -73.473801],
737         [-107.907161, -73.48005], [-107.718257, -73.539925], [-107.53814, -73.55486],
738         [-107.749009, -73.490044], [-107.626002, -73.473801], [-107.46785, -73.506272],
739         [-107.459064, -73.47005], [-107.450278, -73.389842], [-107.718257, -73.413693],
740         [-107.766582, -73.388585], [-107.687506, -73.379789], [-107.353629, -73.358408],
741         [-107.305305, -73.325654], [-107.489816, -73.315566], [-107.617216, -73.33952],
742         [-107.718257, -73.342039], [-107.683113, -73.297892], [-107.766582, -73.268818],
743         [-107.573285, -73.242232], [-107.463457, -73.228288], [-107.388774, -73.173674],
744         [-107.336057, -73.109952], [-107.511781, -73.070329], [-107.674326, -73.038308],
745         [-107.841265, -73.043434], [-107.80612, -73.157131], [-107.955485, -73.168586],
746         [-108.069706, -73.106124], [-108.298148, -73.138021]]
747 )
748 RI6T = ee.Geometry.LineString(
749     geodesic = False,
750     proj = 'EPSG:4326',
751     coords = [[-108.689137, -74.223929], [-107.898376, -74.09438], [-107.951094, -73.985629],
752         [-108.61006, -73.922463], [-109.418393, -73.84928], [-109.804987, -73.765951],
753         [-110.14765, -73.844391], [-110.14765, -73.963794], [-110.279443, -74.091968],
754         [-109.892849, -74.11123], [-109.655621, -74.197626], [-109.233882, -74.144886],
755         [-108.689137, -74.223929]]
756 )
757 #####
758 OW7aT = ee.Geometry.LineString(
759     geodesic = False,
760     proj = 'EPSG:4326',
761     coords = [[-102.891684, -73.380105], [-102.913649, -73.406478], [-102.891684,
762         -73.449097], [-102.658849, -73.44534], [-102.610525, -73.486611], [-102.557807,
763         -73.415261], [-102.755497, -73.387643], [-102.891684, -73.380105]])
764 )
765 OW7bT = ee.Geometry.LineString(
766     geodesic = False,
767     proj = 'EPSG:4326',
768     coords = [[-142.818282, -74.217513], [-142.805103, -74.228864], [-142.763368,
769         -74.214525], [-142.719437, -74.20376], [-142.704061, -74.19419], [-142.646951,
770         -74.176822], [-142.60302, -74.160639], [-142.690882, -74.175625], [-142.73701,
```

```

734 )
735 OW7cT = ee.Geometry.LineString(
736     geodesic = False,
737     proj = 'EPSG:4326',
738     coords = [[-137.465611, -74.898116], [-137.75336, -74.915853], [-137.731394, -74.885515],
739     [-137.639139, -74.8752], [-137.575439, -74.884371], [-137.472201, -74.887809],
740     [-137.465611, -74.898116]])
741 )
742 OW7T = ee.Geometry.MultiLineString(
743     geodesic = False,
744     proj = 'EPSG:4326',
745     coords = ee.List([OW7aT, OW7bT, OW7cT]))
746 )
747 DW7T = ee.Geometry.LineString(
748     geodesic = False,
749     proj = 'EPSG:4326',
750     coords = [[-137.776459, -75.017583], [-137.797326, -75.029791], [-137.732528,
751     -75.041706], [-137.741314, -75.055311], [-137.62929, -75.040855], [-137.655648,
752     -75.032062], [-137.65455, -75.020423], [-137.594145, -75.020139], [-137.555705,
753     -75.022694], [-137.526052, -75.018719], [-137.460155, -75.019571], [-137.420617,
754     -75.024114], [-137.373391, -75.011333], [-137.374489, -75.000817], [-137.378882,
755     -74.992], [-137.337148, -74.987162], [-137.281136, -74.983748], [-137.244892,
756     -74.986024], [-137.169111, -74.986877], [-137.176799, -74.976914], [-137.153735,
757     -74.974636], [-137.106509, -74.982324], [-137.057087, -74.995698], [-137.030728,
758     -75.017868], [-136.999976, -75.019288], [-136.948357, -74.997689], [-137.038416,
759     -74.987162], [-137.116394, -74.971502], [-137.08674, -74.963524], [-137.071365,
760     -74.94784], [-137.030728, -74.938992], [-136.970323, -74.929569], [-136.925294,
761     -74.928141], [-136.884657, -74.928141], [-136.8572, -74.935281], [-136.830842,
762     -74.92414], [-136.802286, -74.897543], [-136.835235, -74.8752], [-136.891247,
763     -74.874913], [-137.031827, -74.876059], [-137.108706, -74.876633], [-137.241598,
764     -74.874913], [-137.339344, -74.879785], [-137.385472, -74.884943], [-137.38657,
765     -74.894107], [-137.334951, -74.895825], [-137.304199, -74.901836], [-137.245991,
766     -74.908702], [-137.19547, -74.909275], [-137.092232, -74.908702], [-137.05489,
767     -74.909888], [-137.053792, -74.916994], [-137.081249, -74.929283], [-137.09992,
768     -74.935566], [-137.154834, -74.936709], [-137.204256, -74.942133], [-137.235008,
769     -74.950122], [-137.241598, -74.959248], [-137.282234, -74.970648], [-137.332755,
770     -74.975206], [-137.421715, -74.975206], [-137.485415, -74.974636], [-137.523855,
771     -74.978337], [-137.467843, -74.979192], [-137.419519, -74.98204], [-137.414027,
772     -74.988585], [-137.45686, -74.989723], [-137.561196, -74.99456], [-137.657845,
773     -74.998826], [-137.710562, -75.009628], [-137.776459, -75.017583]])
774 )
775 FI7aT = ee.Geometry.LineString(
776     geodesic = False,
777     proj = 'EPSG:4326',
778     coords = [[-141.567056, -74.353642], [-141.865788, -74.40098], [-141.716422, -74.429314],
779     [-141.716422, -74.476429], [-141.791105, -74.545669], [-141.764746, -74.586601],
780     [-141.606594, -74.586601], [-141.479194, -74.557374], [-141.316649, -74.574917],
781     [-141.250752, -74.613445], [-141.220001, -74.569072], [-141.000345, -74.570242],
782     [-140.754331, -74.587777], [-140.44242, -74.590108], [-140.16126, -74.608782],
783     [-139.924033, -74.595946], [-139.928426, -74.555037], [-140.143688, -74.550357],
784     [-140.213978, -74.50111], [-140.389702, -74.464664], [-140.539068, -74.522231],
785     [-140.745544, -74.524579], [-140.833407, -74.498759], [-140.978379, -74.48701],
786     [-141.189249, -74.476429], [-141.206821, -74.454068], [-141.048669, -74.432856],
787     [-141.118959, -74.39507], [-141.395725, -74.390342], [-141.457229, -74.359567],
788     [-141.567056, -74.353642]])
789 )
790 FI7bT = ee.Geometry.LineString(
791     geodesic = False,
792     proj = 'EPSG:4326',
793     coords = [[-138.123355, -74.839469], [-138.093702, -74.846077], [-138.071736,
794     -74.835159], [-138.032198, -74.823371], [-138.002545, -74.813014], [-137.995955,
795     -74.794583], [-138.002545, -74.775842], [-138.005839, -74.758811], [-138.023412,
796     -74.749278], [-138.06295, -74.756212], [-138.081621, -74.76603], [-138.096997,
797     -74.780168], [-138.103586, -74.80956], [-138.123355, -74.839469]])
798 )
799 FI7cT = ee.Geometry.LineString(
800     geodesic = False,
801     proj = 'EPSG:4326',
802     coords = [[-140.046506, -74.81244], [-140.329861, -74.820496], [-140.22223, -74.903124],
803     -74.217513]])

```

```

[-140.444082, -75.023406], [-140.613217, -75.028515], [-140.755993, -74.970505],
[-140.911949, -74.923141], [-140.868017, -74.851533], [-141.021776, -74.846937],
[-141.201894, -74.890528], [-141.133801, -74.956824], [-140.907555, -75.006361],
[-140.720848, -75.111166], [-140.641772, -75.146128], [-140.488013, -75.081779],
[-140.187085, -75.085173], [-140.235409, -75.051203], [-139.868585, -75.010907],
[-139.840029, -74.971075], [-139.82026, -74.883655], [-140.046506, -74.81244]]
```

764)

```

765 FI7dT = ee.Geometry.LineString(
766     geodesic = False,
767     proj = 'EPSG:4326',
768     coords = [[-139.165965, -74.96851], [-139.291169, -75.029365], [-139.227469, -75.07584],
769     [-139.025386, -75.058854], [-138.904575, -75.10467], [-138.673937, -75.113705],
770     [-138.658561, -75.08998], [-138.834285, -75.086021], [-138.840875, -75.066215],
771     [-138.665151, -75.066215], [-138.260984, -75.07584], [-138.307112, -75.058287],
772     [-138.47405, -75.046384], [-138.673937, -75.024257], [-138.636595, -75.012897],
773     [-138.445495, -75.011759], [-138.203874, -75.008348], [-138.083063, -75.015169],
774     [-138.206071, -74.985596], [-138.355436, -74.971359], [-138.533357, -74.9725],
775     [-138.649775, -74.986166], [-138.766192, -75.008348], [-138.873823, -75.004371],
776     [-138.904575, -74.98389], [-139.165965, -74.96851]]]
```

777)

```

778 FI7T = ee.Geometry.MultiLineString(
779     geodesic = False,
780     proj = 'EPSG:4326',
781     coords = ee.List([FI7aT,FI7bT,FI7cT,FI7dT]))
```

782 RI7T = ee.Geometry.LineString(

```

783     geodesic = False,
784     proj = 'EPSG:4326',
785     coords = [[-141.816832, -75.339219], [-142.062846, -75.350338], [-141.948625,
786     -75.438987], [-141.720183, -75.531503], [-141.5181, -75.522719], [-141.33359,
787     -75.551256], [-141.096362, -75.507331], [-140.639478, -75.507331], [-140.336353,
788     -75.420191], [-140.582367, -75.451136], [-140.946996, -75.490826], [-141.294052,
789     -75.50073], [-141.298445, -75.451136], [-140.990927, -75.416874], [-140.617512,
790     -75.400268], [-140.520864, -75.368111], [-140.358319, -75.296897], [-140.244098,
791     -75.219734], [-140.428608, -75.221978], [-140.586761, -75.291319], [-140.828382,
792     -75.323641], [-141.021679, -75.381423], [-141.324803, -75.425721], [-141.307231,
793     -75.374771], [-141.386307, -75.334771], [-141.816832, -75.339219]]]
```

794)

```

795 ##
```

796 OW8aT = ee.Geometry.LineString(

```

797     geodesic = False,
798     proj = 'EPSG:4326',
799     coords = [[-137.126919, -74.797178], [-137.24114, -74.804092], [-137.236747, -74.839761],
800     [-137.166457, -74.878786], [-137.017091, -74.886805], [-136.933622, -74.92114],
801     [-136.929229, -74.97478], [-136.929229, -75.005512], [-136.819401, -74.959965],
802     [-136.84576, -74.93143], [-136.880905, -74.893679], [-137.008305, -74.8501],
803     [-137.021484, -74.821361], [-137.126919, -74.797178]])
```

804)

805 OW8bT = ee.Geometry.LineString(

```

806     geodesic = False,
807     proj = 'EPSG:4326',
808     coords = [[-137.796893, -74.915283], [-137.854003, -74.97122], [-137.750765, -74.986027],
809     [-137.671689, -74.985457], [-137.660706, -74.963242], [-137.430068, -74.926143],
810     [-137.430068, -74.90613], [-137.473999, -74.900981], [-137.649723, -74.93928],
811     [-137.700244, -74.936996], [-137.748568, -74.891246], [-137.796893, -74.915283]])
```

812)

813 OW8T = ee.Geometry.MultiLineString(

```

814     geodesic = False,
815     proj = 'EPSG:4326',
816     coords = ee.List([OW8aT,OW8bT]))
```

817)

818 DW8aT = ee.Geometry.LineString(

```

819     geodesic = False,
820     proj = 'EPSG:4326',
821     coords = [[-136.12529, -74.018634], [-135.760661, -74.137984], [-136.068179, -74.209892],
822     [-136.375697, -74.232596], [-136.402056, -74.190747], [-136.098931, -74.152388],
823     [-136.138469, -74.085045], [-136.287835, -74.082638], [-136.50749, -74.109127],
824     [-136.494311, -74.159588], [-136.560208, -74.214674], [-136.665642, -74.175172],
825     [-136.617318, -74.087457], [-136.301014, -74.023477], [-136.296621, -73.977446],
826     [-136.12529, -74.018634]])
```

827)

```

801 DW8bT = ee.Geometry.LineString(
802     geodesic = False,
803     proj = 'EPSG:4326',
804     coords = [[-142.376688, -74.372], [-142.605129, -74.412199], [-142.504088, -74.438159],
805     [-142.398653, -74.419283], [-141.932983, -74.499349], [-141.20812, -74.618688],
806     [-140.94014, -74.777573], [-140.571119, -74.760255], [-140.720484, -74.66408],
807     [-140.81274, -74.395659], [-140.02198, -74.411022], [-140.030766, -74.582511],
808     [-139.529951, -74.59069], [-139.674924, -74.319827], [-140.531581, -74.248407],
809     [-140.514008, -74.214966], [-139.081854, -74.123862], [-139.248792, -74.012884],
810     [-139.6266, -74.017724], [-140.02198, -73.998351], [-140.342677, -74.061227],
811     [-140.34707, -74.129871], [-141.142223, -74.212578], [-140.913781, -74.313887],
812     [-141.13783, -74.453476], [-141.511245, -74.513433], [-142.19657, -74.380287],
813     [-142.376688, -74.372]]
814 )
815 DW8T = ee.Geometry.MultiLineString(
816     geodesic = False,
817     proj = 'EPSG:4326',
818     coords = ee.List([DW8aT,DW8bT])
819 )
820 FI8aT = ee.Geometry.LineString(
821     geodesic = False,
822     proj = 'EPSG:4326',
823     coords = [[-140.789221, -74.943987], [-141.013269, -74.967372], [-140.685982,
824     -75.080931], [-140.723324, -75.123299], [-140.54101, -75.164987], [-140.461934,
825     -75.145284], [-140.514651, -75.086021], [-140.380661, -75.069613], [-140.161005,
826     -75.026527], [-139.893026, -74.999823], [-139.759036, -74.93771], [-140.037998,
827     -74.961672], [-140.573958, -75.030502], [-140.699162, -74.976488], [-140.789221,
828     -74.943987]]
829 )
830 FI8bT = ee.Geometry.LineString(
831     geodesic = False,
832     proj = 'EPSG:4326',
833     coords = [[-135.912696, -74.692968], [-136.033506, -74.679619], [-136.02472, -74.659869],
834     [-135.939054, -74.636018], [-135.94784, -74.625536], [-136.090617, -74.599884],
835     [-135.941251, -74.605134], [-135.822637, -74.628449], [-135.701826, -74.661032],
836     [-135.873157, -74.649403], [-135.923678, -74.657544], [-135.912696, -74.692968]]
837 )
838 FI8cT = ee.Geometry.LineString(
839     geodesic = False,
840     proj = 'EPSG:4326',
841     coords = [[-137.079067, -74.535568], [-137.197681, -74.540253], [-137.085656,
842     -74.563664], [-136.947273, -74.556643], [-136.962649, -74.593463], [-137.07687,
843     -74.614464], [-137.087853, -74.626119], [-136.929701, -74.637766], [-136.94947,
844     -74.661611], [-136.881377, -74.646495], [-136.920915, -74.616212], [-136.931898,
845     -74.581199], [-136.894556, -74.561325], [-136.989008, -74.542009], [-137.079067,
846     -74.535568]]
847 )
848 FI8dT = ee.Geometry.LineString(
849     geodesic = False,
850     proj = 'EPSG:4326',
851     coords = [[-136.586566, -73.957118], [-136.709574, -73.943753], [-136.797436,
852     -73.980176], [-136.819401, -74.03224], [-136.709574, -74.0564], [-136.617318,
853     -74.03224], [-136.441594, -73.994725], [-136.454773, -73.952261], [-136.503097,
854     -73.90481], [-136.604139, -73.919426], [-136.533849, -73.940108], [-136.586566,
855     -73.957118]]
856 )
857 FI8eT = ee.Geometry.LineString(
858     geodesic = False,
859     proj = 'EPSG:4326',
860     coords = [[-140.683167, -74.854546], [-140.733688, -74.937565], [-140.601894,
861     -74.928428], [-140.417384, -74.874055], [-140.252642, -74.874055], [-139.940731,
862     -74.951265], [-139.725469, -74.886089], [-139.725469, -74.864304], [-139.679341,
863     -74.83904], [-139.758417, -74.814885], [-139.863852, -74.769928], [-139.934142,
864     -74.768195], [-139.868245, -74.806249], [-139.918766, -74.809128], [-140.03079,
865     -74.756067], [-140.085704, -74.75838], [-140.092294, -74.794728], [-140.138421,
866     -74.794728], [-140.180156, -74.776852], [-140.169173, -74.753178], [-140.085704,
867     -74.734678], [-140.120849, -74.705145], [-140.208711, -74.688328], [-140.142814,
868     -74.727154], [-140.215301, -74.757802], [-140.320735, -74.796455], [-140.476691,
869     -74.807976], [-140.669988, -74.838465], [-140.683167, -74.854546]]
870 )
871 FI8T = ee.Geometry.MultiLineString(

```

```

837     geodesic = False,
838     proj = 'EPSG:4326',
839     coords = ee.List([FI8aT,FI8bT,FI8cT,FI8dT,FI8eT])
840   )
841 RI8aT = ee.Geometry.LineString(
842   geodesic = False,
843   proj = 'EPSG:4326',
844   coords = [[-139.545366, -74.210786], [-139.62554, -74.211682], [-139.589297, -74.214372],
845   [-139.557447, -74.219153], [-139.540973, -74.219153], [-139.511319, -74.222739],
846   [-139.473978, -74.225427], [-139.457504, -74.218257], [-139.492649, -74.221245],
847   [-139.521204, -74.211981], [-139.545366, -74.210786]])
848   )
849 RI8bT = ee.Geometry.LineString(
850   geodesic = False,
851   proj = 'EPSG:4326',
852   coords = [[-139.816641, -74.222215], [-139.944041, -74.219228], [-139.95722, -74.224008],
853   [-139.920977, -74.223709], [-139.879243, -74.224905], [-139.833115, -74.224606],
854   [-139.816641, -74.222215]])
855   )
856 RI8cT = ee.Geometry.LineString(
857   geodesic = False,
858   proj = 'EPSG:4326',
859   coords = [[-135.379885, -74.574626], [-135.461158, -74.59682], [-135.048205, -74.616067],
860   [-134.810977, -74.648676], [-134.775832, -74.873336], [-134.424383, -74.899689],
861   [-134.01802, -74.959676], [-133.68634, -74.895108], [-134.253052, -74.694563],
862   [-134.477101, -74.598567], [-134.685773, -74.628301], [-135.028436, -74.587476],
863   [-135.164623, -74.584556], [-135.379885, -74.574626]])
864   )
865 RI8dT = ee.Geometry.LineString(
866   geodesic = False,
867   proj = 'EPSG:4326',
868   coords = [[-136.776895, -74.860572], [-136.803253, -74.775406], [-136.636315,
869   -74.775406], [-136.464984, -74.765596], [-136.352959, -74.738435], [-136.260704,
870   -74.709489], [-136.122321, -74.711806], [-136.388104, -74.777714], [-136.776895,
871   -74.860572]])
872   #
873 OW9aT = ee.Geometry.LineString(
874   geodesic = False,
875   proj = 'EPSG:4326',
876   coords = [[-127.502386, -73.385363], [-127.462848, -73.373737], [-127.447473,
877   -73.360213], [-127.450767, -73.352033], [-127.39146, -73.345106], [-127.412328,
878   -73.340067], [-127.450767, -73.336602], [-127.490305, -73.327464], [-127.528745,
879   -73.3303], [-127.496895, -73.339438], [-127.459554, -73.344162], [-127.487011,
880   -73.360843], [-127.517762, -73.368391], [-127.535335, -73.376565], [-127.559497,
881   -73.383164], [-127.539728, -73.388504], [-127.502386, -73.385363]])
882   )
883 OW9bT = ee.Geometry.LineString(
884   geodesic = False,
885   proj = 'EPSG:4326',
886   coords = [[-127.585372, -74.054969], [-127.635893, -74.054063], [-127.638089,
887   -74.041382], [-127.657858, -74.040475], [-127.653465, -74.052856], [-127.661153,
888   -74.058288], [-127.639188, -74.062211], [-127.616124, -74.0604], [-127.583175,
889   -74.061305], [-127.585372, -74.054969]])
890   )
891 OW9cT = ee.Geometry.LineString(
892   geodesic = False,
893   proj = 'EPSG:4326',
894   coords = [[-128.82776, -74.100929], [-128.868397, -74.099726], [-128.900247, -74.084971],
895   [-128.920016, -74.088585], [-128.894755, -74.094006], [-128.884871, -74.104842],
896   [-128.82776, -74.100929]])
897   )
898 OW9dT = ee.Geometry.MultiLineString(
899   geodesic = False,
900   proj = 'EPSG:4326',
901   coords = ee.List([OW9aT,OW9bT,OW9cT]))

```

```

886 )
887 DW9aT = ee.Geometry.LineString(
888     geodesic = False,
889     proj = 'EPSG:4326',
890     coords = [[-125.158348, -73.26597], [-125.116613, -73.279252], [-124.991409, -73.26154],
891     [-124.90794, -73.241914], [-124.945282, -73.220358], [-124.987016, -73.227334],
892     [-124.995803, -73.240011], [-125.052913, -73.24698], [-125.158348, -73.26597]]
893 )
894 DW9bT = ee.Geometry.LineString(
895     geodesic = False,
896     proj = 'EPSG:4326',
897     coords = [[-123.192613, -73.705801], [-123.192613, -73.692235], [-123.170647,
898     -73.680509], [-123.091571, -73.673715], [-123.192613, -73.666919], [-123.236544,
899     -73.68915], [-123.192613, -73.705801]]
900 )
901 DW9cT = ee.Geometry.LineString(
902     geodesic = False,
903     proj = 'EPSG:4326',
904     coords = [[-126.468774, -73.248166], [-126.43912, -73.258929], [-126.446808, -73.239297],
905     [-126.423744, -73.221229], [-126.422646, -73.197109], [-126.429236, -73.182493],
906     [-126.458889, -73.168502], [-126.497329, -73.162138], [-126.54785, -73.162457],
907     [-126.559931, -73.151634], [-126.585191, -73.139529], [-126.603862, -73.133793],
908     [-126.646695, -73.133793], [-126.660972, -73.142396], [-126.629122, -73.156091],
909     [-126.606059, -73.164048], [-126.615943, -73.188531], [-126.631319, -73.201554],
910     [-126.592879, -73.208539], [-126.576405, -73.229474], [-126.576405, -73.251332],
911     [-126.525884, -73.26241], [-126.500624, -73.269052], [-126.505017, -73.250065],
912     [-126.539064, -73.238979], [-126.526983, -73.223766], [-126.514901, -73.202824],
913     [-126.536867, -73.192344], [-126.551145, -73.175181], [-126.532474, -73.168183],
914     [-126.490739, -73.172001], [-126.449005, -73.18599], [-126.443513, -73.213615],
915     [-126.453398, -73.231374], [-126.474265, -73.238347], [-126.468774, -73.248166]]
901 )
902 DW9dT = ee.Geometry.LineString(
903     geodesic = False,
904     proj = 'EPSG:4326',
905     coords = [[-127.045281, -72.891536], [-127.080426, -72.837159], [-127.286902,
906     -72.905752], [-127.387943, -72.958636], [-127.554882, -72.998515], [-127.502164,
907     -73.109946], [-127.65153, -73.13164], [-127.704247, -73.171127], [-127.616385,
908     -73.228285], [-127.405516, -73.297889], [-127.25615, -73.221943], [-127.225398,
909     -73.130364], [-126.926667, -73.075443], [-127.023315, -73.012639], [-127.177074,
910     -73.083112], [-127.335226, -73.080558], [-127.339619, -73.040865], [-127.032101,
911     -72.963785], [-127.045281, -72.891536]]
906 )
907 DW9T = ee.Geometry.MultiLineString(
908     geodesic = False,
909     proj = 'EPSG:4326',
910     coords = ee.List([DW9aT,DW9bT,DW9cT,DW9dT]))
911 )
912 FI9aT = ee.Geometry.LineString(
913     geodesic = False,
914     proj = 'EPSG:4326',
915     coords = [[-127.493767, -73.341366], [-127.483882, -73.336484], [-127.421281,
916     -73.337429], [-127.391627, -73.335066], [-127.387234, -73.339477], [-127.41524,
917     -73.345303], [-127.425125, -73.353016], [-127.445443, -73.356634], [-127.44709,
918     -73.360725], [-127.453131, -73.368273], [-127.483333, -73.376604], [-127.484981,
919     -73.381632], [-127.486628, -73.387915], [-127.508594, -73.392468], [-127.507495,
920     -73.385873], [-127.502553, -73.380061], [-127.488276, -73.372675], [-127.506397,
921     -73.365757], [-127.489923, -73.360567], [-127.463015, -73.359309], [-127.448738,
922     -73.348137], [-127.465761, -73.346877], [-127.42787, -73.342469], [-127.438853,
923     -73.339791], [-127.493767, -73.341366]]
916 )
917 FI9bT = ee.Geometry.LineString(
918     geodesic = False,
919     proj = 'EPSG:4326',
920     coords = [[-125.864952, -73.246504], [-125.967091, -73.249354], [-125.946224,
921     -73.283832], [-125.931947, -73.307516], [-125.949519, -73.318241], [-125.978074,
922     -73.328331], [-125.979173, -73.340933], [-125.965993, -73.351953], [-125.92316,
923     -73.362651], [-125.881426, -73.358562], [-125.839691, -73.370198], [-125.806743,
924     -73.3768], [-125.747436, -73.389996], [-125.694719, -73.412278], [-125.587087,
925     -73.410397], [-125.607955, -73.391252], [-125.678244, -73.373657], [-125.752927,
926     -73.339674], [-125.811136, -73.31351], [-125.806743, -73.301519], [-125.702406,
927     -73.318557], [-125.699112, -73.306884], [-125.801251, -73.278143], [-125.788072,
928     -73.318557], [-125.699112, -73.306884], [-125.801251, -73.278143], [-125.788072,
```

```

921 )
922 FI9cT = ee.Geometry.LineString(
923     geodesic = False,
924     proj = 'EPSG:4326',
925     coords = [[-126.292246, -73.197983], [-126.353749, -73.253152], [-126.287853,
926     -73.253786], [-126.243921, -73.227808], [-126.164845, -73.182734], [-126.120914,
927     -73.152191], [-126.129701, -73.12032], [-126.096752, -73.103723], [-125.890276,
928     -73.166197], [-125.776055, -73.212585], [-125.571775, -73.213855], [-125.582758,
929     -73.176375], [-125.305992, -73.067286], [-124.961133, -73.046153], [-124.961133,
930     -72.969096], [-125.255471, -72.933023], [-125.422409, -73.016015], [-125.729927,
931     -73.114577], [-125.824379, -73.133075], [-125.86831, -73.07816], [-125.633279,
932     -73.030769], [-125.426803, -72.95944], [-125.505879, -72.905271], [-125.795824,
933     -72.92399], [-125.738714, -73.010877], [-125.914438, -73.017939], [-126.221956,
934     -72.927861], [-126.463577, -72.936247], [-126.371322, -73.027563], [-126.112128,
935     -73.067286], [-126.175828, -73.092225], [-126.331784, -73.07944], [-126.283459,
936     -73.129249], [-126.27028, -73.151553], [-126.342767, -73.149006], [-126.439415,
937     -73.112664], [-126.522884, -73.101169], [-126.536063, -73.130525], [-126.496525,
938     -73.145183], [-126.40427, -73.160469], [-126.309818, -73.16556], [-126.292246,
939     -73.197983]]]
940 )
941 FI9dT = ee.Geometry.LineString(
942     geodesic = False,
943     proj = 'EPSG:4326',
944     coords = [[-128.418073, -73.318952], [-128.433448, -73.339753], [-128.405442,
945     -73.360843], [-128.376887, -73.365876], [-128.367003, -73.360686], [-128.335702,
946     -73.358798], [-128.331309, -73.34306], [-128.308245, -73.332191], [-128.286828,
947     -73.333452], [-128.272551, -73.343532], [-128.280788, -73.322735], [-128.31154,
948     -73.319582], [-128.338447, -73.313906], [-128.38128, -73.320214], [-128.414778,
949     -73.313906], [-128.418073, -73.318952]]]
950 )
951 FI9eT = ee.Geometry.LineString(
952     geodesic = False,
953     proj = 'EPSG:4326',
954     coords = [[-123.719369, -73.440797], [-124.480924, -73.450344], [-124.419421,
955     -73.607339], [-124.149245, -73.636457], [-124.184389, -73.769025], [-124.153638,
956     -73.832778], [-123.885658, -73.829106], [-123.523226, -73.817475], [-123.343108,
957     -73.792348], [-123.257443, -73.721669], [-123.345305, -73.643266], [-123.496867,
958     -73.650691], [-123.470509, -73.717356], [-123.604499, -73.71859], [-123.611088,
959     -73.649452], [-123.890051, -73.580033], [-123.874675, -73.565739], [-123.362877,
960     -73.610439], [-123.325536, -73.556412], [-123.626464, -73.547702], [-123.617678,
961     -73.531514], [-123.393629, -73.516559], [-123.595712, -73.434689], [-123.719369,
962     -73.440797]]]
963 )
964 FI9t = ee.Geometry.MultiLineString(
965     geodesic = False,
966     proj = 'EPSG:4326',
967     coords = ee.List([FI9aT,FI9bT,FI9cT,FI9dT,FI9eT]))
968 )
969 RI9aT = ee.Geometry.LineString(
970     geodesic = False,
971     proj = 'EPSG:4326',
972     coords = [[-127.109144, -73.78468], [-127.267296, -74.002899], [-127.592386, -74.17877],
973     [-127.913083, -74.292193], [-127.170648, -74.457898], [-125.307968, -74.143989],
974     [-126.753302, -73.771175], [-127.109144, -73.78468]]]
975 )
976 RI9bT = ee.Geometry.LineString(
977     geodesic = False,
978     proj = 'EPSG:4326',
979     coords = [[-126.742017, -73.056483], [-126.799127, -73.041428], [-126.811209,
980     -73.040467], [-126.824937, -73.046554], [-126.74037, -73.064485], [-126.742017,
981     -73.056483]]]
982 )
983 RI9cT = ee.Geometry.LineString(
984     geodesic = False,
985     proj = 'EPSG:4326',
986     coords = [[-126.139225, -73.299862], [-126.23807, -73.291021], [-126.386337, -73.291968],
987     [-126.452234, -73.300178], [-126.457725, -73.347153], [-126.119456, -73.351875],
988     [-126.029397, -73.370749], [-125.72737, -73.574827], [-125.439622, -73.640248],
989     [-125.34444, -73.574827], [-125.34444, -73.574827]]]

```

```

[-125.428639, -73.598731], [-125.624132, -73.546223], [-125.699913, -73.509152],
[-125.618641, -73.443222], [-125.664768, -73.433515], [-125.743844, -73.429441],
[-125.776793, -73.414081], [-125.819626, -73.405611], [-125.853672, -73.378293],
[-125.900898, -73.384577], [-125.967893, -73.375463], [-126.048067, -73.348412],
[-126.076623, -73.326045], [-126.123849, -73.3128], [-126.112866, -73.29986],
[-126.139225, -73.299862]]

956 )
957 RI9dT = ee.Geometry.LineString(
958     geodesic = False,
959     proj = 'EPSG:4326',
960     coords = [[[-128.481373, -73.206873], [-128.49016, -73.20973], [-128.446229, -73.213855],
961     [-128.40724, -73.217186], [-128.416026, -73.213062], [-128.481373, -73.206873]]]
962 )
963 RI9T = ee.Geometry.MultiLineString(
964     geodesic = False,
965     proj = 'EPSG:4326',
966     coords = ee.List([RI9aT,RI9bT,RI9cT,RI9dT]))
967
968 ## (3.2) Training Feature Collection
969 TrainingRings = ee.FeatureCollection([
970     ee.Feature(OW1T,{
971         'classes': 1,
972         'subclasses': 1,
973         'name': 'Open Water',
974         'type': 'Water',
975         'image': 'T1',
976         'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(0)).get('system:
977             time_start'),
978     }),
979     ee.Feature(DW1T,{
980         'classes': 1,
981         'subclasses': 2,
982         'name': 'Rough Water',
983         'type': 'Water',
984         'image': 'T1',
985         'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(0)).get('system:
986             time_start'),
987     }),
988     ee.Feature(FI1T,{
989         'classes': 2,
990         'subclasses': 3,
991         'name': 'Flat Sea Ice',
992         'type': 'Ice',
993         'image': 'T1',
994         'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(0)).get('system:
995             time_start')
996     }),
997     ee.Feature(RI1T,{
998         'classes': 2,
999         'subclasses': 4,
1000         'name': 'Floating Land Ice',
1001         'type': 'Ice',
1002         'image': 'T1',
1003         'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(0)).get('system:
1004             time_start')
1005     }),
1006     ee.Feature(OW2T,{
1007         'classes': 1,
1008         'subclasses': 1,
1009         'name': 'Open Water',
1010         'type': 'Water',
1011         'image': 'T2',
1012         'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(1)).get('system:
1013             time_start'),
1014     }),
1015     ee.Feature(DW2T,{
1016         'classes': 1,
1017         'subclasses': 2,
1018         'name': 'Rough Water',
1019     })
1020 ]
1021 )
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
```

```
1015     'type': 'Water',
1016     'image': 'T2',
1017     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(1)).get('system:
1018         time_start'),
1019   },
1020   ee.Feature(FI2T,{
1021     'classes': 2,
1022     'subclasses': 3,
1023     'name': 'Flat Sea Ice',
1024     'type': 'Ice',
1025     'image': 'T2',
1026     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(1)).get('system:
1027         time_start')
1028   },
1029   ee.Feature(RI2T,{
1030     'classes': 2,
1031     'subclasses': 4,
1032     'name': 'Floating Land Ice',
1033     'type': 'Ice',
1034     'image': 'T2',
1035     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(1)).get('system:
1036         time_start')
1037   },
1038   ee.Feature(OW3T,{
1039     'classes': 1,
1040     'subclasses': 1,
1041     'name': 'Open Water',
1042     'type': 'Water',
1043     'image': 'T3',
1044     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(2)).get('system:
1045         time_start'),
1046   },
1047   ee.Feature(DW3T,{
1048     'classes': 1,
1049     'subclasses': 2,
1050     'name': 'Rough Water',
1051     'type': 'Water',
1052     'image': 'T3',
1053     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(2)).get('system:
1054         time_start'),
1055   },
1056   ee.Feature(FI3T,{
1057     'classes': 2,
1058     'subclasses': 3,
1059     'name': 'Flat Sea Ice',
1060     'type': 'Ice',
1061     'image': 'T3',
1062     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(2)).get('system:
1063         time_start')
1064   },
1065   ee.Feature(RI3T,{
1066     'classes': 2,
1067     'subclasses': 4,
1068     'name': 'Floating Land Ice',
1069     'type': 'Ice',
1070     'image': 'T3',
1071     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(2)).get('system:
1072         time_start')
1073   },
1074   ee.Feature(OW4T,{
1075     'classes': 1,
1076     'subclasses': 1,
1077     'name': 'Open Water',
1078     'type': 'Water',
1079     'image': 'T4',
1080     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(3)).get('system:
1081         time_start'),
1082   },
1083   ee.Feature(DW4T,{
```

```
1078     'classes': 1,
1079     'subclasses': 2,
1080     'name': 'Rough Water',
1081     'type': 'Water',
1082     'image': 'T4',
1083     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(3)).get('system:
1084         time_start'),
1085   },
1086   ee.Feature(FI4T,{
1087     'classes': 2,
1088     'subclasses': 3,
1089     'name': 'Flat Sea Ice',
1090     'type': 'Ice',
1091     'image': 'T4',
1092     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(3)).get('system:
1093         time_start')
1094   },
1095   ee.Feature(RI4T,{
1096     'classes': 2,
1097     'subclasses': 4,
1098     'name': 'Floating Land Ice',
1099     'type': 'Ice',
1100     'image': 'T4',
1101     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(3)).get('system:
1102         time_start')
1103   },
1104   ee.Feature(OW5T,{
1105     'classes': 1,
1106     'subclasses': 1,
1107     'name': 'Open Water',
1108     'type': 'Water',
1109     'image': 'T5',
1110     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(4)).get('system:
1111         time_start'),
1112   },
1113   ee.Feature(DW5T,{
1114     'classes': 1,
1115     'subclasses': 2,
1116     'name': 'Rough Water',
1117     'type': 'Water',
1118     'image': 'T5',
1119     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(4)).get('system:
1120         time_start'),
1121   },
1122   ee.Feature(FI5T,{
1123     'classes': 2,
1124     'subclasses': 3,
1125     'name': 'Flat Sea Ice',
1126     'type': 'Ice',
1127     'image': 'T5',
1128     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(4)).get('system:
1129         time_start')
1130   },
1131   ee.Feature(RI5T,{
1132     'classes': 2,
1133     'subclasses': 4,
1134     'name': 'Floating Land Ice',
1135     'type': 'Ice',
1136     'image': 'T5',
1137     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(4)).get('system:
1138         time_start')
1139   },
1140   ee.Feature(OW6T,{
1141     'classes': 1,
1142     'subclasses': 1,
1143     'name': 'Open Water',
1144     'type': 'Water',
1145     'image': 'T6',
1146     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(5)).get('system:
```

```
    time_start'),
1142  )),
1143  ee.Feature(DW6T,{
1144    'classes': 1,
1145    'subclasses': 2,
1146    'name': 'Rough Water',
1147    'type': 'Water',
1148    'image': 'T6',
1149    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(5)).get('system:
      time_start'),
1150  )),
1151  ee.Feature(FI6T,{
1152    'classes': 2,
1153    'subclasses': 3,
1154    'name': 'Flat Sea Ice',
1155    'type': 'Ice',
1156    'image': 'T6',
1157    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(5)).get('system:
      time_start')
1158  )),
1159  ee.Feature(RI6T,{
1160    'classes': 2,
1161    'subclasses': 4,
1162    'name': 'Floating Land Ice',
1163    'type': 'Ice',
1164    'image': 'T6',
1165    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(5)).get('system:
      time_start')
1166  )),
1167
1168  ee.Feature(OW7T,{
1169    'classes': 1,
1170    'subclasses': 1,
1171    'name': 'Open Water',
1172    'type': 'Water',
1173    'image': 'T7',
1174    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(6)).get('system:
      time_start'),
1175  )),
1176  ee.Feature(DW7T,{
1177    'classes': 1,
1178    'subclasses': 2,
1179    'name': 'Rough Water',
1180    'type': 'Water',
1181    'image': 'T7',
1182    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(6)).get('system:
      time_start'),
1183  )),
1184  ee.Feature(FI7T,{
1185    'classes': 2,
1186    'subclasses': 3,
1187    'name': 'Flat Sea Ice',
1188    'type': 'Ice',
1189    'image': 'T7',
1190    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(6)).get('system:
      time_start')
1191  )),
1192  ee.Feature(RI7T,{
1193    'classes': 2,
1194    'subclasses': 4,
1195    'name': 'Floating Land Ice',
1196    'type': 'Ice',
1197    'image': 'T7',
1198    'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(6)).get('system:
      time_start')
1199  )),
1200
1201  ee.Feature(OW8T,{
1202    'classes': 1,
1203    'subclasses': 1,
1204    'name': 'Open Water',
```

```
1205     'type': 'Water',
1206     'image': 'T8',
1207     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(7)).get('system:
1208         time_start'),
1209   )),
1210   ee.Feature(DW8T, {
1211     'classes': 1,
1212     'subclasses': 2,
1213     'name': 'Rough Water',
1214     'type': 'Water',
1215     'image': 'T8',
1216     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(7)).get('system:
1217         time_start'),
1218   )),
1219   ee.Feature(FI8T, {
1220     'classes': 2,
1221     'subclasses': 3,
1222     'name': 'Flat Sea Ice',
1223     'type': 'Ice',
1224     'image': 'T8',
1225     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(7)).get('system:
1226         time_start')
1227   )),
1228   ee.Feature(RI8T, {
1229     'classes': 2,
1230     'subclasses': 4,
1231     'name': 'Floating Land Ice',
1232     'type': 'Ice',
1233     'image': 'T8',
1234     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(7)).get('system:
1235         time_start')
1236   )),
1237   ee.Feature(OW9T, {
1238     'classes': 1,
1239     'subclasses': 1,
1240     'name': 'Open Water',
1241     'type': 'Water',
1242     'image': 'T9',
1243     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(8)).get('system:
1244         time_start'),
1245   )),
1246   ee.Feature(DW9T, {
1247     'classes': 1,
1248     'subclasses': 2,
1249     'name': 'Rough Water',
1250     'type': 'Water',
1251     'image': 'T9',
1252     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(8)).get('system:
1253         time_start'),
1254   )),
1255   ee.Feature(FI9T, {
1256     'classes': 2,
1257     'subclasses': 3,
1258     'name': 'Flat Sea Ice',
1259     'type': 'Ice',
1260     'image': 'T9',
1261     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(8)).get('system:
1262         time_start')
1263   )),
1264   ee.Feature(RI9T, {
1265     'classes': 2,
1266     'subclasses': 4,
1267     'name': 'Floating Land Ice',
1268     'type': 'Ice',
1269     'image': 'T9',
1270     'utc': ee.Image(TrainingData.toList(TrainingData.size()).get(8)).get('system:
1271         time_start')
1272   })
1273 ])
```

```

1268 TrainingPolys = TrainingRings.map(lambda feature: ee.Feature(ee.Geometry.MultiPolygon(feature
1269     .geometry().coordinates())).copyProperties(feature))
1270 
1271 ## (3.3) Validation Polygons
1272 OW1aV = ee.Geometry.LineString(
1273     geodesic = False,
1274     proj = 'EPSG:4326',
1275     coords = [[-106.447642, -74.989868], [-106.446544, -74.984176], [-106.41689, -74.981614],
1276     [-106.396023, -74.987307], [-106.380647, -74.992713], [-106.419087, -74.992429],
1277     [-106.447642, -74.989868]])
1278 )
1279 OW1bV = ee.Geometry.LineString(
1280     geodesic = False,
1281     proj = 'EPSG:4326',
1282     coords = [[-105.9097, -74.317455], [-106.160107, -74.376739], [-106.647743, -74.382655],
1283     [-106.296294, -74.471139], [-105.993169, -74.507571], [-105.672472, -74.536889],
1284     [-105.351775, -74.540404], [-105.659293, -74.469964], [-105.567037, -74.451126],
1285     [-105.334202, -74.45937], [-105.044257, -74.42873], [-105.294664, -74.375557],
1286     [-105.76912, -74.317455], [-105.9097, -74.317455]])
1287 )
1288 OW1V = ee.Geometry.MultiLineString(
1289     geodesic = False,
1290     proj = 'EPSG:4326',
1291     coords = ee.List([OW1aV,OW1bV])
1292 )
1293 DW1aV = ee.Geometry.LineString(
1294     geodesic = False,
1295     proj = 'EPSG:4326',
1296     coords = [[-107.619805, -74.028236], [-107.606626, -74.023701], [-107.580267,
1297     -74.026725], [-107.55281, -74.02491], [-107.497896, -74.022491], [-107.493503,
1298     -74.032165], [-107.54622, -74.033979], [-107.561596, -74.04274], [-107.555006,
1299     -74.051195], [-107.51437, -74.048779], [-107.528648, -74.054515], [-107.593446,
1300     -74.050289], [-107.614314, -74.038813], [-107.619805, -74.028236]])
1301 DW1bV = ee.Geometry.LineString(
1302     geodesic = False,
1303     proj = 'EPSG:4326',
1304     coords = [[-105.299057, -74.13528], [-105.44403, -74.191647], [-105.597789, -74.226326],
1305     [-105.408885, -74.269276], [-105.189229, -74.266891], [-104.965181, -74.271658],
1306     [-104.833387, -74.272848], [-104.947608, -74.211985], [-105.039864, -74.170083],
1307     [-105.299057, -74.13528]])
1308 )
1309 DW1cV = ee.Geometry.LineString(
1310     geodesic = False,
1311     proj = 'EPSG:4326',
1312     coords = [[-107.237293, -74.300521], [-107.3603, -74.299331], [-107.281224, -74.325471],
1313     [-107.193361, -74.353943], [-107.101106, -74.397734], [-106.973706, -74.415453],
1314     [-107.09232, -74.37171], [-107.237293, -74.300521]])
1315 )
1316 DW1V = ee.Geometry.MultiLineString(
1317     geodesic = False,
1318     proj = 'EPSG:4326',
1319     coords = ee.List([DW1aV,DW1bV,DW1cV]))
1320 )
1321 FI1aV = ee.Geometry.LineString(
1322     geodesic = False,
1323     proj = 'EPSG:4326',
1324     coords = [[-107.042918, -74.449355], [-107.262573, -74.511674], [-107.223035, -74.57258],
1325     [-107.38558, -74.651874], [-107.350435, -74.714558], [-106.766151, -74.64373],
1326     [-106.643144, -74.685568], [-107.00338, -74.759677], [-106.832048, -74.825383],
1327     [-106.695862, -74.828834], [-106.265337, -74.854115], [-106.252157, -74.790835],
1328     [-106.094006, -74.712242], [-106.34002, -74.640238], [-106.322447, -74.510499],
1329     [-106.845228, -74.488186], [-107.042918, -74.449355]])
1330 )
1331 FI1bV = ee.Geometry.LineString(
1332     geodesic = False,
1333     proj = 'EPSG:4326',
1334     coords = [[-107.06049, -73.932808], [-106.656324, -74.011676], [-107.104422, -74.11785],
1335     [-106.700255, -74.287728], [-106.502565, -74.230502], [-106.14233, -74.218555],
1336     [-105.940247, -74.237668], [-105.812847, -74.223337], [-105.87435, -74.161087],
1337     [-106.278516, -74.129871], [-106.26973, -74.103413], [-106.032502, -74.101006],
1338     [-106.032502, -74.101006]])

```

```

        [-106.023716, -74.063643], [-105.896316, -74.040702], [-106.036895, -74.017727],
        [-106.217013, -74.03224], [-106.401524, -73.986239], [-106.34002, -73.965614],
        [-106.884766, -73.90603], [-107.06049, -73.932808]]
1315 )
1316 FI1cV = ee.Geometry.LineString(
1317     geodesic = False,
1318     proj = 'EPSG:4326',
1319     coords = [[-109.881836, -74.558843], [-109.996057, -74.564106], [-109.86646, -74.628889],
1320         [-109.767615, -74.623646], [-109.741257, -74.599157], [-109.655591, -74.579889],
1321         [-109.479867, -74.565276], [-109.550156, -74.528391], [-109.651198, -74.499057],
1322         [-109.620446, -74.53308], [-109.73906, -74.534838], [-109.800564, -74.544795],
1323         [-109.701719, -74.556502], [-109.826922, -74.596239], [-109.859871, -74.577552],
1324         [-109.837905, -74.560597], [-109.881836, -74.558843]]
1325 )
1326 FI1V = ee.Geometry.MultiLineString(
1327     geodesic = False,
1328     proj = 'EPSG:4326',
1329     coords = ee.List([FI1aV,FI1bV,FI1cV])
1330 )
1331 RI1aV = ee.Geometry.LineString(
1332     geodesic = False,
1333     proj = 'EPSG:4326',
1334     coords = [[-108.989101, -74.225131], [-107.75903, -74.025591], [-107.785388, -74.215567],
1335         [-108.075334, -74.287132], [-108.391638, -74.303789], [-108.54979, -74.263312],
1336         [-108.866094, -74.268083], [-108.989101, -74.225131]]
1337 )
1338 RI1bV = ee.Geometry.LineString(
1339     geodesic = False,
1340     proj = 'EPSG:4326',
1341     coords = [[-107.763388, -75.13204], [-108.092871, -75.192808], [-107.521767, -75.197301],
1342         [-107.218642, -75.237665], [-106.814476, -75.271221], [-106.546496, -75.275691],
1343         [-106.401523, -75.224222], [-106.550889, -75.170329], [-106.621179, -75.115116],
1344         [-106.53771, -74.997263], [-106.788117, -74.972213], [-106.748579, -75.077818],
1345         [-106.774938, -75.164704], [-106.897945, -75.215253], [-107.218642, -75.204034],
1346         [-107.253787, -75.171455], [-107.376794, -75.133168], [-107.763388, -75.13204]]
1347 )
1348 RI1cV = ee.Geometry.LineString(
1349     geodesic = False,
1350     proj = 'EPSG:4326',
1351     coords = [[-107.391052, -74.986169], [-107.536024, -75.0021], [-107.650245, -75.025962],
1352         [-107.768859, -75.074709], [-107.663424, -75.11766], [-107.386659, -75.112016],
1353         [-107.311975, -75.062256], [-107.224113, -75.03391], [-107.263651, -74.982755],
1354         [-107.391052, -74.986169]]
1355 )
1356 RI1V = ee.Geometry.MultiLineString(
1357     geodesic = False,
1358     proj = 'EPSG:4326',
1359     coords = ee.List([RI1aV,RI1bV,RI1cV])
1360 )
1361 #####
1362 OW2aV = ee.Geometry.LineString(
1363     geodesic = False,
1364     proj = 'EPSG:4326',
1365     coords = [[-102.891684, -73.380105], [-102.913649, -73.406478], [-102.891684,
1366         -73.449097], [-102.658849, -73.44534], [-102.610525, -73.486611], [-102.557807,
1367         -73.415261], [-102.755497, -73.387643], [-102.891684, -73.380105]]
1368 )
1369 OW2bV = ee.Geometry.LineString(
1370     geodesic = False,
1371     proj = 'EPSG:4326',
1372     coords = [[-103.597267, -73.218775], [-103.60825, -73.239697], [-103.559926, -73.254897],
1373         [-103.522584, -73.244764], [-103.597267, -73.218775]]
1374 )
1375 OW2V = ee.Geometry.MultiLineString(
1376     geodesic = False,
1377     proj = 'EPSG:4326',
1378     coords = ee.List([OW2aV,OW2bV])
1379 )
1380 DW2aV = ee.Geometry.LineString(
1381     geodesic = False,
1382     proj = 'EPSG:4326',
1383     coords = [[-106.023716, -74.063643], [-105.896316, -74.040702], [-106.036895, -74.017727],
1384         [-106.217013, -74.03224], [-106.401524, -73.986239], [-106.34002, -73.965614],
1385         [-106.884766, -73.90603], [-107.06049, -73.932808]]]

```

```

1365     coords = [[-101.989228, -75.052905], [-102.033159, -75.071596], [-101.894776,
1366     -75.090263], [-101.758589, -75.063103], [-101.642172, -75.064801], [-101.374192,
1367     -75.028516], [-101.328064, -75.019999], [-101.130374, -75.005225], [-100.893146,
1368     -74.960818], [-100.710832, -74.933426], [-100.710832, -74.917997], [-100.748174,
1369     -74.916854], [-100.609791, -74.867603], [-100.789908, -74.837173], [-100.829446,
1370     -74.860145], [-100.787712, -74.891676], [-100.983205, -74.928857], [-101.035922,
1371     -74.916283], [-101.090836, -74.924285], [-101.038119, -74.937424], [-101.128178,
1372     -74.957397], [-101.169912, -75.000676], [-101.41373, -75.018295], [-101.462054,
1373     -75.029083], [-101.730034, -75.046102], [-101.892579, -75.058004], [-101.989228,
1374     -75.052905]]
1375   )
1376 DW2bV = ee.Geometry.LineString(
1377   geodesic = False,
1378   proj = 'EPSG:4326',
1379   coords = [[-104.336548, -73.920186], [-104.315681, -73.925358], [-104.244293,
1380     -73.912273], [-104.204755, -73.88607], [-104.166315, -73.87753], [-104.111401,
1381     -73.874479], [-104.096026, -73.868375], [-104.117991, -73.862574], [-104.175102,
1382     -73.871732], [-104.21464, -73.876614], [-104.256374, -73.900395], [-104.311288,
1383     -73.905269], [-104.346433, -73.911968], [-104.336548, -73.920186]]
1384   )
1385 DW2cV = ee.Geometry.LineString(
1386   geodesic = False,
1387   proj = 'EPSG:4326',
1388   coords = [[-101.440654, -73.603305], [-101.495568, -73.605786], [-101.486781,
1389     -73.623142], [-101.440654, -73.616325], [-101.427475, -73.650998], [-101.471406,
1390     -73.649142], [-101.477995, -73.660891], [-101.374757, -73.655945], [-101.390133,
1391     -73.626859], [-101.440654, -73.603305]]
1392   )
1393 DW2dV = ee.Geometry.LineString(
1394   geodesic = False,
1395   proj = 'EPSG:4326',
1396   coords = [[-103.827473, -73.478484], [-103.572673, -73.523413], [-103.348624,
1397     -73.651305], [-103.480417, -73.66367], [-103.673714, -73.610436], [-103.603425,
1398     -73.56698], [-103.682501, -73.555788], [-103.792328, -73.586859], [-103.76597,
1399     -73.614155], [-103.845046, -73.614155], [-103.875797, -73.565738], [-103.849439,
1400     -73.51718], [-103.910943, -73.487229], [-103.827473, -73.478484]]
1401   )
1402 DW2eV = ee.Geometry.LineString(
1403   geodesic = False,
1404   proj = 'EPSG:4326',
1405   coords = [[-104.304052, -72.80764], [-104.6555, -72.8647], [-104.541279, -72.904793],
1406     [-104.128327, -72.930611], [-103.891099, -73.075768], [-103.592367, -73.187982],
1407     [-103.592367, -72.943505], [-104.304052, -72.80764]]
1408   )
1409 DW2V = ee.Geometry.MultiLineString(
1410   geodesic = False,
1411   proj = 'EPSG:4326',
1412   coords = ee.List([DW2aV,DW2bV,DW2cV,DW2dV,DW2eV]))
1413   )
1414 FI2aV = ee.Geometry.LineString(
1415   geodesic = False,
1416   proj = 'EPSG:4326',
1417   coords = [[-101.83273, -73.338573], [-102.236896, -73.34361], [-102.245682, -73.406475],
1418     [-102.447765, -73.541481], [-101.973309, -73.588725], [-101.736081, -73.633365],
1419     [-101.62186, -73.576305], [-101.83273, -73.486605], [-101.83273, -73.338573]]
1420   )
1421 FI2bV = ee.Geometry.LineString(
1422   geodesic = False,
1423   proj = 'EPSG:4326',
1424   coords = [[-103.062801, -73.845006], [-103.712982, -73.896284], [-103.563615,
1425     -74.029825], [-103.001298, -74.066058], [-102.772855, -74.167083], [-102.597131,
1426     -74.265104], [-102.043599, -74.345937], [-101.982095, -74.234088], [-101.586715,
1427     -74.243635], [-101.50764, -74.128674], [-101.727295, -74.024989], [-102.149034,
1428     -73.995938], [-102.913435, -73.940108], [-103.062801, -73.845006]]
1429   )
1430 FI2cV = ee.Geometry.LineString(
1431   geodesic = False,
1432   proj = 'EPSG:4326',
1433   coords = [[-105.220969, -73.326285], [-105.225362, -73.385444], [-105.155072,
1434     -73.405535], [-105.317617, -73.440641], [-105.339583, -73.478174], [-105.3923,
1435     -73.531828], [-105.128713, -73.534318], [-105.119927, -73.572889], [-104.957382,
1436     -73.505295]]]

```

```

        -73.567915], [-104.764085, -73.594002], [-104.482926, -73.580342], [-104.491712,
-73.544281], [-104.834375, -73.562943], [-104.829982, -73.520613], [-104.944203,
-73.511882], [-104.878306, -73.481923], [-104.98374, -73.456916], [-104.917844,
-73.438136], [-104.566395, -73.494413], [-104.368705, -73.485672], [-104.40385,
-73.458168], [-104.500498, -73.451911], [-104.430208, -73.416828], [-104.58836,
-73.382931], [-104.865126, -73.410556], [-104.843161, -73.391725], [-104.715761,
-73.367841], [-104.79923, -73.348964], [-104.957382, -73.366585], [-105.220969,
-73.326285]]
```

1406)

1407 FI2dV = ee.Geometry.LineString(
1408 geodesic = False,
1409 proj = 'EPSG:4326',
1410 coords = [[-104.002142, -73.351324], [-104.085611, -73.429207], [-103.793469,
-73.445492], [-103.644104, -73.412279], [-104.002142, -73.351324]]
1411)

1412 FI2eV = ee.Geometry.LineString(
1413 geodesic = False,
1414 proj = 'EPSG:4326',
1415 coords = [[-103.247507, -73.408828], [-103.203576, -73.405691], [-103.196986,
-73.393763], [-103.155251, -73.395647], [-103.183807, -73.418237], [-103.247507,
-73.408828]]
1416)

1417 FI2fV = ee.Geometry.LineString(
1418 geodesic = False,
1419 proj = 'EPSG:4326',
1420 coords = [[-103.243355, -73.602609], [-103.262025, -73.622136], [-103.238962,
-73.647828], [-103.213701, -73.648757], [-103.216996, -73.638237], [-103.141215,
-73.621827], [-103.091792, -73.60478], [-103.092891, -73.595784], [-103.06763,
-73.592992], [-102.931444, -73.589267], [-102.921559, -73.566594], [-103.051156,
-73.548869], [-103.081908, -73.582437], [-103.143412, -73.592992], [-103.211505,
-73.602919], [-103.243355, -73.602609]]
1421)

1422 FI2gV = ee.Geometry.LineString(
1423 geodesic = False,
1424 proj = 'EPSG:4326',
1425 coords = [[-102.84735, -74.759389], [-102.513474, -75.024541], [-102.333356, -74.994988],
[-102.445381, -74.956255], [-102.335553, -74.912849], [-102.355322, -74.868749],
[-102.269656, -74.865881], [-101.913814, -74.912278], [-101.639245, -74.925998],
[-101.643638, -74.826248], [-101.819362, -74.822221], [-101.7315, -74.873337],
[-102.166418, -74.854978], [-102.258674, -74.834297], [-102.194973, -74.795159],
[-102.146649, -74.760544], [-102.84735, -74.759389]]
1426)

1427 FI2V = ee.Geometry.MultiLineString(
1428 geodesic = False,
1429 proj = 'EPSG:4326',
1430 coords = ee.List([FI2aV,FI2bV,FI2cV,FI2dV,FI2eV,FI2fV,FI2gV])
1431)

1432 RI2aV = ee.Geometry.LineString(
1433 geodesic = False,
1434 proj = 'EPSG:4326',
1435 coords = [[-101.314342, -73.419021], [-101.454922, -73.499091], [-101.173763,
-73.616021], [-101.182549, -73.714896], [-100.611444, -73.73706], [-99.882188,
-73.727214], [-99.996409, -73.670473], [-101.314342, -73.419021]]
1436)

1437 RI2bV = ee.Geometry.LineString(
1438 geodesic = False,
1439 proj = 'EPSG:4326',
1440 coords = [[-105.012015, -72.950586], [-105.135022, -72.994981], [-105.108663,
-73.064885], [-104.889008, -73.187976], [-104.71548, -73.203857], [-104.566114,
-73.176533], [-104.678139, -73.110269], [-104.77918, -73.077681], [-104.761608,
-73.047596], [-104.904384, -73.009113], [-105.012015, -72.950586]]
1441)

1442 RI2cV = ee.Geometry.LineString(
1443 geodesic = False,
1444 proj = 'EPSG:4326',
1445 coords = [[-102.276639, -75.192531], [-101.854901, -75.386696], [-100.730264,
-74.961672], [-101.037782, -75.020849], [-101.521024, -75.077536], [-101.802183,
-75.102414], [-102.276639, -75.192531]]
1446)

1447 RI2dV = ee.Geometry.LineString(
1448 geodesic = False,

```

1449     proj = 'EPSG:4326',
1450     coords = [[-100.94074, -74.852968], [-101.063747, -74.858708], [-101.002244, -74.890816],
1451         [-100.986868, -74.913708], [-100.848485, -74.890816], [-100.866057, -74.866741],
1452         [-100.94074, -74.852968]]
1453     )
1454 RI2eV = ee.Geometry.LineString(
1455     geodesic = False,
1456     proj = 'EPSG:4326',
1457     coords = [[-101.799593, -75.002808], [-101.966532, -75.005082], [-101.973121,
1458         -75.037452], [-101.874276, -75.043692], [-101.694159, -75.030075], [-101.593117,
1459         -75.029508], [-101.630458, -75.019288], [-101.751269, -75.013607], [-101.799593,
1460         -75.002808]]]
1461     )
1462 RI2fV = ee.Geometry.LineString(
1463     geodesic = False,
1464     proj = 'EPSG:4326',
1465     coords = [[-101.371265, -74.957253], [-101.509648, -74.965805], [-101.606296,
1466         -74.960104], [-101.665603, -74.964095], [-101.509648, -74.973782], [-101.362479,
1467         -74.988587], [-101.318548, -75.00167], [-101.268027, -74.993707], [-101.338316,
1468         -74.962384], [-101.371265, -74.957253]]]
1469     )
1470 RI2gV = ee.Geometry.LineString(
1471     geodesic = False,
1472     proj = 'EPSG:4326',
1473     coords = [[-101.481712, -74.880646], [-101.500383, -74.878353], [-101.516857,
1474         -74.880359], [-101.500143, -74.882652], [-101.481712, -74.880646]]]
1475     )
1476 RI2V = ee.Geometry.MultiLineString(
1477     geodesic = False,
1478     proj = 'EPSG:4326',
1479     coords = ee.List([RI2aV,RI2bV,RI2cV,RI2dV,RI2eV,RI2fV,RI2gV])
1480     )
1481     ###
1482 OW3aV = ee.Geometry.LineString(
1483     geodesic = False,
1484     proj = 'EPSG:4326',
1485     coords = [[-111.820533, -74.101305], [-111.923771, -74.116346], [-111.840302,
1486         -74.145186], [-111.785388, -74.149988], [-111.805157, -74.170379], [-111.877644,
1487         -74.198524], [-111.80955, -74.183559], [-111.721688, -74.178169], [-111.679954,
1488         -74.154789], [-111.717295, -74.148187], [-111.699723, -74.131975], [-111.631629,
1489         -74.122961], [-111.517408, -74.136781], [-111.475674, -74.134377], [-111.517408,
1490         -74.104917], [-111.589895, -74.105518], [-111.664578, -74.113339], [-111.704116,
1491         -74.098898], [-111.660185, -74.088661], [-111.684347, -74.075402], [-111.686543,
1492         -74.047648], [-111.695329, -74.03073], [-111.655792, -74.003504], [-111.701919,
1493         -73.987147], [-111.737064, -74.004715], [-111.706312, -74.019845], [-111.717295,
1494         -74.084443], [-111.750243, -74.125966], [-111.820533, -74.101305]]]
1495     )
1496 OW3bV = ee.Geometry.LineString(
1497     geodesic = False,
1498     proj = 'EPSG:4326',
1499     coords = [[-112.048975, -73.752127], [-111.974292, -73.774245], [-111.930361,
1500         -73.754587], [-111.934754, -73.726287], [-111.978685, -73.707187], [-112.024813,
1501         -73.731827], [-112.048975, -73.752127]]]
1502     )
1503 OW3cV = ee.Geometry.LineString(
1504     geodesic = False,
1505     proj = 'EPSG:4326',
1506     coords = [[-113.672752, -73.975854], [-113.797956, -73.996009], [-113.775441,
1507         -74.007969], [-113.627173, -73.993132], [-113.552491, -73.998281], [-113.442114,
1508         -73.968574], [-113.514051, -73.948234], [-113.534369, -73.952334], [-113.512953,
1509         -73.963568], [-113.475062, -73.964933], [-113.486594, -73.976916], [-113.583791,
1510         -73.985708], [-113.628272, -73.976461], [-113.672752, -73.975854]]]
1511     )
1512 OW3V = ee.Geometry.MultiLineString(
1513     geodesic = False,
1514     proj = 'EPSG:4326',
1515     coords = ee.List([OW3aV,OW3bV,OW3cV]))
1516     )
1517 DW3aV = ee.Geometry.LineString(
1518     geodesic = False,
1519     proj = 'EPSG:4326',
1520     coords = [[-111.820533, -74.101305], [-111.923771, -74.116346], [-111.840302,
1521         -74.145186], [-111.785388, -74.149988], [-111.805157, -74.170379], [-111.877644,
1522         -74.198524], [-111.80955, -74.183559], [-111.721688, -74.178169], [-111.679954,
1523         -74.154789], [-111.717295, -74.148187], [-111.699723, -74.131975], [-111.631629,
1524         -74.122961], [-111.517408, -74.136781], [-111.475674, -74.134377], [-111.517408,
1525         -74.104917], [-111.589895, -74.105518], [-111.664578, -74.113339], [-111.704116,
1526         -74.098898], [-111.660185, -74.088661], [-111.684347, -74.075402], [-111.686543,
1527         -74.047648], [-111.695329, -74.03073], [-111.655792, -74.003504], [-111.701919,
1528         -73.987147], [-111.737064, -74.004715], [-111.706312, -74.019845], [-111.717295,
1529         -74.084443], [-111.750243, -74.125966], [-111.820533, -74.101305]]]
1530     )

```

```

1496     coords = [[-108.904859, -74.336743], [-108.972952, -74.340895], [-108.964166,
1497         -74.360455], [-108.89168, -74.368745], [-108.860928, -74.385906], [-108.724741,
1498     -74.396546], [-108.904859, -74.336743]]
1499 )
1500 DW3bV = ee.Geometry.LineString(
1501     geodesic = False,
1502     proj = 'EPSG:4326',
1503     coords = [[-111.097739, -73.541475], [-111.308608, -73.589965], [-111.708381,
1504         -73.682821], [-111.774278, -73.802156], [-111.717168, -73.916989], [-111.484333,
1505         -73.991086], [-111.21196, -73.980173], [-110.81658, -74.04674], [-110.856118,
1506         -73.916989], [-110.965946, -73.836446], [-111.01427, -73.732135], [-111.058201,
1507         -73.655634], [-110.952766, -73.557654], [-111.097739, -73.541475]]
1508 )
1509 DW3cV = ee.Geometry.LineString(
1510     geodesic = False,
1511     proj = 'EPSG:4326',
1512     coords = [[-112.894364, -73.852952], [-113.335871, -73.930985], [-113.131592,
1513         -74.047346], [-113.36882, -74.098597], [-113.331478, -74.167682], [-113.188702,
1514         -74.178469], [-113.045926, -74.186255], [-112.925115, -74.177869], [-112.835057,
1515         -74.185056], [-112.753784, -74.198226], [-112.62858, -74.099801], [-112.920722,
1516         -74.064246], [-112.883381, -74.046138], [-112.439676, -74.029824], [-112.50118,
1517         -73.979569], [-112.786732, -73.97047], [-112.777946, -73.941929], [-112.712049,
1518         -73.817472], [-112.894364, -73.852952]]
1519 )
1520 DW3V = ee.Geometry.MultiLineString(
1521     geodesic = False,
1522     proj = 'EPSG:4326',
1523     coords = [[[-109.02567, -74.585142], [-109.170642, -74.633544], [-109.045439, -74.624808],
1524         [-108.975149, -74.63005], [-108.858731, -74.616649], [-108.911449, -74.599737],
1525         [-109.02567, -74.585142]]]
1526 )
1527 FI3aV = ee.Geometry.LineString(
1528     geodesic = False,
1529     proj = 'EPSG:4326',
1530     coords = [[-109.924061, -74.547573], [-109.994351, -74.552256], [-109.948223,
1531         -74.600758], [-109.779088, -74.607174], [-109.748337, -74.577986], [-109.836199,
1532         -74.572141], [-109.812037, -74.537031], [-109.684636, -74.53293], [-109.673654,
1533         -74.563371], [-109.405674, -74.537031], [-109.35735, -74.520033], [-109.346367,
1534         -74.488921], [-109.247522, -74.487746], [-109.247522, -74.454215], [-109.322205,
1535         -74.434179], [-109.412263, -74.458925], [-109.394691, -74.497146], [-109.513305,
1536         -74.503018], [-109.596774, -74.49597], [-109.656081, -74.522378], [-109.838395,
1537         -74.52062], [-109.882326, -74.538204], [-109.924061, -74.547573]]
1538 )
1539 FI3bV = ee.Geometry.LineString(
1540     geodesic = False,
1541     proj = 'EPSG:4326',
1542     coords = [[-109.470442, -74.169631], [-109.571484, -74.182213], [-109.549518,
1543         -74.193589], [-109.406742, -74.195383], [-109.395759, -74.185207], [-109.347435,
1544         -74.18341], [-109.426511, -74.169631], [-109.470442, -74.169631]]
1545 )
1546 FI3cV = ee.Geometry.LineString(
1547     geodesic = False,
1548     proj = 'EPSG:4326',
1549     coords = [[-109.470442, -74.169631], [-109.571484, -74.182213], [-109.549518,
1550         -74.193589], [-109.406742, -74.195383], [-109.395759, -74.185207], [-109.347435,
1551         -74.18341], [-109.426511, -74.169631], [-109.470442, -74.169631]]
1552 )
1553 FI3V = ee.Geometry.MultiLineString(
1554     geodesic = False,
1555     proj = 'EPSG:4326',
1556     coords = [[FI3aV, FI3bV, FI3cV]])
1557 )
1558 RI3aV = ee.Geometry.LineString(
1559     geodesic = False,
1560     proj = 'EPSG:4326',
1561     coords = [[-112.196272, -74.245424], [-113.303337, -74.238262], [-113.443916, -74.58777],
1562         [-113.29455, -74.874766], [-112.213844, -74.941133], [-112.354425, -75.197016],
1563         [-111.229788, -75.277643], [-111.089208, -75.163295], [-109.402253, -75.199263],
1564         [-109.630695, -75.029931], [-109.059591, -74.863295], [-109.542833, -74.768916],
1565         [-109.331964, -74.676283], [-109.903068, -74.65071], [-110.087579, -74.831134],
1566         [-110.685042, -74.959386], [-111.64274, -74.952547], [-112.521362, -74.545669],
1567         [-112.196272, -74.245424]]]
1568 )

```

```

1538 RI3bV = ee.Geometry.LineString(
1539     geodesic = False,
1540     proj = 'EPSG:4326',
1541     coords = [[-110.322649, -73.806139], [-110.384152, -73.824512], [-110.29629, -73.864864],
1542         [-110.305077, -74.003503], [-110.454442, -74.041003], [-110.375366, -74.097693],
1543         [-110.204035, -74.074797], [-109.975593, -74.083238], [-109.8438, -74.166182],
1544         [-109.602179, -74.149388], [-109.567034, -74.09408], [-110.322649, -73.806139]]
1545     )
1546 RI3cV = ee.Geometry.LineString(
1547     geodesic = False,
1548     proj = 'EPSG:4326',
1549     coords = [[-110.836461, -74.084444], [-110.863918, -74.093179], [-110.89467, -74.106422],
1550         [-110.871606, -74.105519], [-110.836461, -74.084444]]
1551     )
1552 RI3dV = ee.Geometry.LineString(
1553     geodesic = False,
1554     proj = 'EPSG:4326',
1555     coords = [[-113.391858, -73.940258], [-113.37099, -73.947247], [-113.346828, -73.962126],
1556         [-113.350123, -73.971229], [-113.375383, -73.958787], [-113.389661, -73.948158],
1557         [-113.391858, -73.940258]]
1558 RI3eV = ee.Geometry.LineString(
1559     geodesic = False,
1560     proj = 'EPSG:4326',
1561     coords = [[-110.447421, -73.871123], [-110.531988, -73.861965], [-110.662683, -73.86746],
1562         [-110.624243, -73.880886], [-110.659388, -73.889425], [-110.604474, -73.904052],
1563         [-110.572624, -73.892778], [-110.515514, -73.893082], [-110.489155, -73.884545],
1564         [-110.447421, -73.871123]]
1565     )
1566 RI3V = ee.Geometry.MultiLineString(
1567     geodesic = False,
1568     proj = 'EPSG:4326',
1569     coords = ee.List([RI3aV, RI3bV, RI3cV, RI3dV, RI3eV]))
1570     )
1571 #####
1572 OW4aV = ee.Geometry.LineString(
1573     geodesic = False,
1574     proj = 'EPSG:4326',
1575     coords = [[-125.742381, -73.197666], [-125.883984, -73.193854], [-125.8291, -73.21671],
1576         [-125.764335, -73.207824], [-125.70945, -73.210998], [-125.68091, -73.209095],
1577         [-125.656761, -73.200523], [-125.631514, -73.19576], [-125.675422, -73.196078],
1578         [-125.699571, -73.201475], [-125.742381, -73.197666]]
1579 OW4bV = ee.Geometry.LineString(
1580     geodesic = False,
1581     proj = 'EPSG:4326',
1582     coords = [[-126.447153, -73.232724], [-126.484475, -73.234626], [-126.452641,
1583         -73.257112], [-126.448251, -73.251413], [-126.447153, -73.232724]]
1584     )
1585 OW4V = ee.Geometry.MultiLineString(
1586     geodesic = False,
1587     proj = 'EPSG:4326',
1588     coords = ee.List([OW4aV, OW4bV]))
1589 DW4aV = ee.Geometry.LineString(
1590     geodesic = False,
1591     proj = 'EPSG:4326',
1592     coords = [[-123.893053, -73.430461], [-123.743687, -73.478644], [-123.589928,
1593         -73.473642], [-123.607501, -73.50237], [-123.42958, -73.536654], [-123.319752,
1594         -73.547238], [-123.251659, -73.520455], [-123.412007, -73.494258], [-123.447152,
1595         -73.506738], [-123.475707, -73.496129], [-123.475707, -73.481768], [-123.539407,
1596         -73.466768], [-123.644842, -73.463015], [-123.805191, -73.436726], [-123.893053,
1597         -73.430461]])
1598 DW4bV = ee.Geometry.LineString(
1599     geodesic = False,
1600     proj = 'EPSG:4326',
1601     coords = [[-124.237912, -73.341252], [-124.222536, -73.354475], [-124.15664, -73.367688],
1602         [-124.092939, -73.382146], [-124.068777, -73.404753], [-124.031436, -73.395337],
1603         [-124.044615, -73.378374], [-124.018257, -73.367688], [-124.064384, -73.361398],
1604         [-124.114905, -73.368317], [-124.145657, -73.352588], [-124.237912, -73.341252]]
```

```

1588 )
1589 DW4cV = ee.Geometry.LineString(
1590     geodesic = False,
1591     proj = 'EPSG:4326',
1592     coords = [[-123.930394, -73.599898], [-123.890856, -73.621597], [-123.829353,
1593     -73.641413], [-123.723918, -73.651313], [-123.666808, -73.642033], [-123.62068,
1594     -73.643269], [-123.658021, -73.630268], [-123.743687, -73.638938], [-123.840336,
1595     -73.627172], [-123.930394, -73.599898]]
1596 )
1597 DW4dV = ee.Geometry.LineString(
1598     geodesic = False,
1599     proj = 'EPSG:4326',
1600     coords = [[-123.322748, -73.70426], [-123.301331, -73.701177], [-123.283759, -73.695318],
1601     [-123.259048, -73.696243], [-123.27662, -73.701948], [-123.283759, -73.705955],
1602     [-123.322748, -73.70426]]
1603 )
1604 DW4eV = ee.Geometry.LineString(
1605     geodesic = False,
1606     proj = 'EPSG:4326',
1607     coords = [[[-125.629844, -73.222259], [-125.761568, -73.26407], [-125.66058, -73.296939],
1608     [-125.537637, -73.307041], [-125.445431, -73.307673], [-125.430063, -73.27166],
1609     [-125.357615, -73.271027], [-125.342247, -73.289361], [-125.243454, -73.28999],
1610     [-125.190764, -73.259006], [-125.076604, -73.233036], [-125.012937, -73.21528],
1611     [-124.927317, -73.204489], [-125.091972, -73.159354], [-125.13588, -73.175894],
1612     [-125.102949, -73.20576], [-125.146857, -73.219088], [-125.232477, -73.200046],
1613     [-125.326879, -73.200681], [-125.390546, -73.257742], [-125.462994, -73.245078],
1614     [-125.425672, -73.219723], [-125.504706, -73.217818], [-125.629844, -73.222259]]
1615 )
1616 DW4fV = ee.Geometry.LineString(
1617     geodesic = False,
1618     proj = 'EPSG:4326',
1619     coords = [[-126.627175, -73.153546], [-126.55363, -73.167868], [-126.534969, -73.191711],
1620     [-126.498745, -73.182178], [-126.485572, -73.189805], [-126.51521, -73.200922],
1621     [-126.518503, -73.214569], [-126.627175, -73.153546]]
1622 )
1623 DW4V = ee.Geometry.MultiLineString(
1624     geodesic = False,
1625     proj = 'EPSG:4326',
1626     coords = ee.List([DW4aV,DW4bV,DW4cV,DW4dV,DW4eV,DW4fV]))
1627 )
1628 FI4aV = ee.Geometry.LineString(
1629     geodesic = False,
1630     proj = 'EPSG:4326',
1631     coords = [[-125.938297, -73.223207], [-126.015136, -73.222572], [-125.986596,
1632     -73.249191], [-125.907562, -73.275137], [-125.903171, -73.318711], [-125.95586,
1633     -73.341402], [-125.821941, -73.360918], [-125.694608, -73.409922], [-125.62216,
1634     -73.413059], [-125.648505, -73.38795], [-125.758275, -73.357142], [-125.821941,
1635     -73.309247], [-125.775838, -73.301672], [-125.685827, -73.312402], [-125.635333,
1636     -73.335102], [-125.608988, -73.309247], [-125.720953, -73.304199], [-125.76047,
1637     -73.28904], [-125.81316, -73.273871], [-125.848286, -73.251724], [-125.879022,
1638     -73.237155], [-125.938297, -73.223207]]
1639 )
1640 FI4bV = ee.Geometry.LineString(
1641     geodesic = False,
1642     proj = 'EPSG:4326',
1643     coords = [[-124.358732, -73.343291], [-124.62218, -73.285248], [-124.481675, -73.407413],
1644     [-124.5651, -73.498774], [-124.270917, -73.630571], [-124.139193, -73.641713],
1645     [-124.12602, -73.811647], [-123.897699, -73.816551], [-123.340068, -73.821449],
1646     [-123.274206, -73.720737], [-123.476183, -73.72197], [-123.783538, -73.677566],
1647     [-123.932825, -73.636762], [-123.994297, -73.575989], [-123.884527, -73.569776],
1648     [-123.722067, -73.610742], [-123.520091, -73.582201], [-123.511309, -73.55236],
1649     [-123.638642, -73.533686], [-123.783538, -73.536179], [-123.954779, -73.473792],
1650     [-123.897699, -73.549875], [-123.985515, -73.55236], [-124.090894, -73.497525],
1651     [-124.17871, -73.393602], [-124.358732, -73.343291]]
1652 )
1653 FI4V = ee.Geometry.MultiLineString(
1654     geodesic = False,
1655     proj = 'EPSG:4326',
1656     coords = ee.List([FI4aV,FI4bV]))
1657 )
1658 RI4aV = ee.Geometry.LineString(

```

```

1630     geodesic = False,
1631     proj = 'EPSG:4326',
1632     coords = [[-125.293972, -73.657177], [-125.495948, -73.713957], [-125.17542, -73.84897],
1633         [-124.644134, -73.817161], [-124.424595, -73.897795], [-124.108457, -73.987748],
1634         [-123.67816, -74.039788], [-123.291769, -73.851417], [-124.029423, -73.873401],
1635         [-124.34556, -73.844081], [-124.613399, -73.745969], [-124.775858, -73.729974],
1636         [-124.789031, -73.694232], [-125.293972, -73.657177]]
1637   )
1638 RI4bV = ee.Geometry.LineString(
1639     geodesic = False,
1640     proj = 'EPSG:4326',
1641     coords = [[-125.255622, -73.425251], [-125.273185, -73.427132], [-125.258366,
1642         -73.430421], [-125.217751, -73.432458], [-125.168355, -73.430421], [-125.139815,
1643         -73.430108], [-125.106884, -73.430578], [-125.144206, -73.426191], [-125.233668,
1644         -73.426661], [-125.255622, -73.425251]]
1645   )
1646 RI4cV = ee.Geometry.LineString(
1647     geodesic = False,
1648     proj = 'EPSG:4326',
1649     coords = [[-124.90381, -73.269408], [-124.916982, -73.27178], [-124.919177, -73.281422],
1650         [-124.899419, -73.297533], [-124.869781, -73.294849], [-124.890088, -73.279052],
1651         [-124.90381, -73.269408]]
1652   )
1653 ## (3.4) Validation Feature Collection
1654 ValidationRings = ee.FeatureCollection([
1655   ee.Feature(OW1V,{
1656     'classes': 1,
1657     'subclasses': 1,
1658     'name': 'Open Water',
1659     'type': 'Water',
1660     'image': 'V1',
1661     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(0)).get('system:
1662       time_start'),
1663   }),
1664   ee.Feature(DW1V,{
1665     'classes': 1,
1666     'subclasses': 2,
1667     'name': 'Rough Water',
1668     'type': 'Water',
1669     'image': 'V1',
1670     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(0)).get('system:
1671       time_start'),
1672   }),
1673   ee.Feature(FI1V,{
1674     'classes': 2,
1675     'subclasses': 3,
1676     'name': 'Flat Sea Ice',
1677     'type': 'Ice',
1678     'image': 'V1',
1679     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(0)).get('system:
1680       time_start')
1681   }),
1682   ee.Feature(RI1V,{
1683     'classes': 2,
1684     'subclasses': 4,
1685     'name': 'Floating Land Ice',
1686     'type': 'Ice',
1687     'image': 'V1',
1688     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(0)).get('system:
1689       time_start')
1690   }),
1691   ee.Feature(OW2V,{
1692     'classes': 1,
1693     'subclasses': 1,
1694   })

```

```
1688     'name': 'Open Water',
1689     'type': 'Water',
1690     'image': 'V2',
1691     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(1)).get('system:
1692         time_start'),
1693   },
1694   ee.Feature(DW2V, {
1695     'classes': 1,
1696     'subclasses': 2,
1697     'name': 'Rough Water',
1698     'type': 'Water',
1699     'image': 'V2',
1700     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(1)).get('system:
1701         time_start'),
1702   },
1703   ee.Feature(FI2V, {
1704     'classes': 2,
1705     'subclasses': 3,
1706     'name': 'Flat Sea Ice',
1707     'type': 'Ice',
1708     'image': 'V2',
1709     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(1)).get('system:
1710         time_start')
1711   },
1712   ee.Feature(RI2V, {
1713     'classes': 2,
1714     'subclasses': 4,
1715     'name': 'Floating Land Ice',
1716     'type': 'Ice',
1717     'image': 'V2',
1718     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(1)).get('system:
1719         time_start')
1720   },
1721   ee.Feature(OW3V, {
1722     'classes': 1,
1723     'subclasses': 1,
1724     'name': 'Open Water',
1725     'type': 'Water',
1726     'image': 'V3',
1727     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(2)).get('system:
1728         time_start'),
1729   },
1730   ee.Feature(DW3V, {
1731     'classes': 1,
1732     'subclasses': 2,
1733     'name': 'Rough Water',
1734     'type': 'Water',
1735     'image': 'V3',
1736     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(2)).get('system:
1737         time_start'),
1738   },
1739   ee.Feature(FI3V, {
1740     'classes': 2,
1741     'subclasses': 3,
1742     'name': 'Flat Sea Ice',
1743     'type': 'Ice',
1744     'image': 'V3',
1745     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(2)).get('system:
1746         time_start')
1747   },
1748   ee.Feature(RI3V, {
1749     'classes': 2,
1750     'subclasses': 4,
1751     'name': 'Floating Land Ice',
1752     'type': 'Ice',
1753     'image': 'V3',
1754     'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(2)).get('system:
1755         time_start')
1756   },
1757 }
```

```
1751     ee.Feature(OW4V, {
1752         'classes': 1,
1753         'subclasses': 1,
1754         'name': 'Open Water',
1755         'type': 'Water',
1756         'image': 'V4',
1757         'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(3)).get('system:
1758             time_start'),
1759     }),
1760     ee.Feature(DW4V, {
1761         'classes': 1,
1762         'subclasses': 2,
1763         'name': 'Rough Water',
1764         'type': 'Water',
1765         'image': 'V4',
1766         'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(3)).get('system:
1767             time_start'),
1768     }),
1769     ee.Feature(FI4V, {
1770         'classes': 2,
1771         'subclasses': 3,
1772         'name': 'Flat Sea Ice',
1773         'type': 'Ice',
1774         'image': 'V4',
1775         'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(3)).get('system:
1776             time_start')
1777     }),
1778     ee.Feature(RI4V, {
1779         'classes': 2,
1780         'subclasses': 4,
1781         'name': 'Floating Land Ice',
1782         'type': 'Ice',
1783         'image': 'V4',
1784         'utc': ee.Image(ValidationData.toList(ValidationData.size()).get(3)).get('system:
1785             time_start')
1786     }),
1787 })
1788
1789 ValidationPolys = ValidationRings.map(lambda feature: ee.Feature(ee.Geometry.MultiPolygon(
1790     feature.geometry().coordinates())).copyProperties(feature))
1791
1792 ## (3.5) Visualization of Polygon Choices
1793 ### (3.5.1) Visualization Function
1794 def mapRDPolys(S1,Opt,polygons,region):
1795     geometry = S1.first().geometry()
1796
1797     Map = geemap.Map()
1798     Map.centerObject(geometry,6)
1799
1800     RDList = S1.toList(S1.size())
1801
1802     for i in range(RDList.size(). getInfo()):
1803         image = ee.Image(RDList.get(i))
1804
1805         matchIndex = ee.Number(image.get('match'))
1806
1807         OptCol = Opt.filterMetadata('match','equals',matchIndex)
1808         OptList = OptCol.toList(OptCol.size())
1809         GeomList = OptList.map(lambda image: ee.Image(image).geometry())
1810
1811         def combineGeometries(geom2,geom1):
1812             geom = ee.Geometry(geom1).union(ee.Geometry(geom2),maxError=1e3)
1813             return geom
1814
1815         OptGeometry = ee.Geometry(GeomList.iterate(
1816             function = combineGeometries,
1817             first = ee.Geometry(GeomList.get(0))
1818         ))
1819         S1Geometry = image.geometry()
1820         OptImage = OptCol.median().clip(S1Geometry)
1821         S1Image = image.clip(OptGeometry)
```

```

1817     Map.addLayer(S1Image.select('HH'),S1_HH_params,'HH'+str(i+1))
1818     print('S1 HH image '+str(i+1)+' added')
1820     Map.addLayer(S1Image.select('HV'),S1_HV_params,'HV'+str(i+1))
1821     print('S1 HV image '+str(i+1)+' added')
1822
1823     Map.addLayer(OptImage.select(RGBbands),RGB_params,'Opt'+str(i+1))
1824     print('Opt image '+str(i+1)+' added')
1825
1826     polyCol = polygons.filter(ee.Filter.eq('utc',S1Image.get('system:time_start')))
1827     classes = polyCol.aggregate_array('name'). getInfo()
1828     for k in range(len(classes)):
1829         polygon = polyCol.filter(ee.Filter.eq('name',classes[k]))
1830         if classes[k] == 'Open Water':
1831             color = 'Orange'
1832         elif classes[k] == 'Disturbed Water':
1833             color = 'Yellow'
1834         elif classes[k] == 'Flat Ice':
1835             color = 'Green'
1836         else:
1837             color = 'Cyan'
1838         Map.addLayer(polygon,{ 'color': color},classes[k]+('+'+color+')_'+str(i+1))
1839         print(classes[k]+ ' polygon added')
1840
1841     html_file = os.path.join(html_dir+'\Polygons\Training_Validation\'+region+'.html')
1842     Map.to_html(outfile=html_file, title='Polygon Choices', width='100%', height='880px')
1843
1844     print('Map finished')
1845
1846     return Map, Map
1847
1848 ## Leave commented!!
1849 ## Used to visualize data set
1850
1851 # PolygonMap, Map = mapRDPolys(S1Training,OptTraining,TrainingRings,'Training')
1852 # PolygonMap, Map = mapRDPolys(S1Validation,OptValidation,ValidationRings,'Validation')
1853 # Map
1854
1855 #%% (4) Sampling
1856 ## (4.1) Functions
1857 def extractPixelValues(data,multiPolygons,numPixels):
1858     dataList = data.toList(data.size())
1859
1860     def samplePerImage(image,multiPolygons,numPixels):
1861         utc = image.get('system:time_start')
1862         multiPolygon = multiPolygons.filterMetadata('utc','equals',utc)
1863         names = multiPolygon.aggregate_array('name')
1864
1865         def samplePerPolygon(image,multiPolygon,name,numPixels):
1866             region = multiPolygon.filterMetadata('name','equals',name).geometry()
1867             polygons = region.geometries()
1868
1869             subSampled = polygons
1870             .map(lambda geometry: image
1871                 .sample(
1872                     region = ee.Geometry(geometry),
1873                     scale = image.select(0).projection().nominalScale(),
1874                     projection = image.select(0).projection().crs(),
1875                     numPixels = ee.Number(numPixels).divide(polygons.size()).ceil(),
1876                     seed = 1,
1877                     geometries = True)\\
1878             .map(lambda feat: feat.copyProperties(multiPolygon.filterMetadata('name',
1879                                         'equals',name).first()))
1880
1881             )
1882
1883             sampledFC = ee.FeatureCollection(subSampled).flatten()
1884
1885             return sampledFC
1886
1887             sampledList = names.map(lambda name: samplePerPolygon(image,multiPolygon,name,
1888                                         numPixels))
1889             sampled = ee.FeatureCollection(sampledList).flatten()

```

```
1886     return sampled
1887
1888
1889     sampleList = dataList.map(lambda image: samplePerImage(ee.Image(image),multiPolygons,
1890                               numPixels))
1890     sampleCol = ee.FeatureCollection(sampleList).flatten()
1891
1892     return sampleCol
1893
1894 def extractPixelValuesSubPolys(data,Polygons,numPixels):
1895     dataList = data.toList(data.size())
1896
1897     def samplePerImage(image,Polygons,numPixels):
1898         utc = image.get('system:time_start')
1899         Polygon = Polygons.filterMetadata('utc','equals',utc)
1900         names = Polygon.aggregate_array('name')
1901
1902         def samplePerPolygon(image,Polygon,name,numPixels):
1903             region = Polygon.filterMetadata('name','equals',name).geometry()
1904
1905             sampledFC = image.sample(
1906                 region = region,
1907                 scale = image.select(0).projection().nominalScale(),
1908                 projection = image.select(0).projection().crs(),
1909                 numPixels = numPixels,
1910                 seed = 1,
1911                 geometries = True)\ \
1912             .map(lambda feat: feat.copyProperties(Polygon.filterMetadata('name','equals',
1913                                                 name).first()))
1914
1915             return sampledFC
1916
1917     sampledList = names.map(lambda name: samplePerPolygon(image,Polygons,name,numPixels))
1918     sampled = ee.FeatureCollection(sampledList).flatten()
1919
1920     return sampled
1921
1922     sampleList = dataList.map(lambda image: samplePerImage(ee.Image(image),Polygons,numPixels
1923                               ))
1924     sampleCol = ee.FeatureCollection(sampleList).flatten()
1925
1926     return sampleCol
1927
1928 def clipToPolygon(image,polys,windowsize):
1929     polygons = polys.filterMetadata('utc','equals',image.get('system:time_start'))
1930     bufferwidth = ee.Number(windowsize).subtract(1).multiply(2)
1931     error = bufferwidth.divide(50)
1932     buffered = polygons.geometry().buffer(bufferwidth,maxError=error)
1933     clipped = image.clip(buffered)
1934
1935     return clipped
1936
1937 def computeGLCM(image,band,windowsize,kernel):
1938     glcm = image.addBands(image.select(band).toInt32().glcmTexture(size=windowsize,kernel=
1939                           kernel))
1940
1941     return glcm
1942
1943 def clipToThresholds(image,band,upperlimit,lowerlimit):
1944     upperMask = image.gt(upperlimit)
1945     lowerMask = image.lt(lowerlimit)
1946
1947     clippedUpper = upperMask.multiply(upperlimit).add(image.select(band).multiply(upperMask.
1948           Not()))
1949     clippedLower = lowerMask.multiply(lowerlimit).add(clippedUpper.multiply(lowerMask.Not()))
1950
1951     return image.addBands(clippedLower.select(band),overwrite=True)
1952
1953 ## (4.2) Computation
1954 ## NOTE: Computation is done during export to reduce computational effort
```

```

1952 #%% (5) Export of Training & Validation Data
1953 ## (5.1) GLCM5
1954 numPixels = 1000
1955 size = 5
1956
1957 ## Leave commented!!
1958 ## Export was necessary to create feature collections used in section below
1959
1960 # geemap.ee_export_vector_to_drive(
1961 #     ee_object = extractPixelValues(
1962 #         TrainingData\
1963 #             .select(['HH','HV'])\
1964 #             .map(lambda image: clipToPolygon(image,TrainingPolys,size))\
1965 #             .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size
1966 # =size,kernel=None)))\
1967 #             .map(lambda image: image.addBands(image.select('HV').toInt32().glcmTexture(size
1968 # =size,kernel=None))),\
1969 #             TrainingPolys,
1970 #             numPixels),
1971 #             description = 'TrainingValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
1972 #             folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
1973 #             file_format = 'shp')
1974
1975 # geemap.ee_export_vector_to_drive(
1976 #     ee_object = extractPixelValues(
1977 #         ValidationData\
1978 #             .select(['HH','HV'])\
1979 #             .map(lambda image: clipToPolygon(image,ValidationPolys,size))\
1980 #             .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size
1981 # =size,kernel=None))),\
1982 #             ValidationPolys,
1983 #             numPixels),
1984 #             description = 'ValidationValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
1985 #             folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
1986 #             file_format = 'shp')
1987
1988 ## (5.2) GLCM11
1989 numPixels = 1000
1990 size = 11
1991
1992 ## Leave commented!!
1993 ## Export was necessary to create feature collections used in section below
1994
1995 # TDList = TrainingData.toList(TrainingData.size())
1996 # for i in range(TrainingData.size(). getInfo()):
1997 #     imCol = ee.ImageCollection(ee.Image(TDList.get(i)))
1998 #     geemap.ee_export_vector_to_drive(
1999 #         ee_object = extractPixelValues(
2000 #             imCol\
2001 #                 .select(['HH','HV'])\
2002 #                 .map(lambda image: clipToPolygon(image,TrainingPolys,size))\
2003 #                 .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(
2004 # size=size,kernel=None)))\
2005 #                 .map(lambda image: image.addBands(image.select('HV').toInt32().glcmTexture(
2006 # size=size,kernel=None))),\
2007 #                 TrainingPolys,
2008 #                 numPixels),
2009 #                 description = 'TrainingValues_w'+str(size)+'_'+str(int(numPixels))+'pixels_Img'+str
2010 # (i+1),
2011 #                 folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
2012 #                 file_format = 'shp')
2013
2014 # geemap.ee_export_vector_to_drive(
2015 #     ee_object = extractPixelValues(
2016 #         ValidationData\
2017 #             .select(['HH','HV'])\
2018 #             .map(lambda image: clipToPolygon(image,ValidationPolys,size))\
2019 #             .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size
2020 # =size,kernel=None)))\

```



```

2083 #                                     geometry = clipped.geometry(),
2084 #                                     scale = image.select(0).projection().nominalScale(),
2085 #                                     crs = image.select(0).projection().crs()
2086 #                                         .get(image.bandNames().get(0)). getInfo()
2087 #                                         print('Pixel number for '+PolyNames[j]+ subpolygon '+str(k+1)+': ',
2088 #                                               pixelCount)
2089 #
2090 #                                         if pixelCount > pixelThreshold:
2091 #                                             shrink = -1*np.sqrt(pixelCount)/4*40
2092 #                                             smallGeometry = Polygon.geometry().buffer(shrink,maxError=1e3)
2093 #                                             smallPolygon = ee.FeatureCollection([ee.Feature(smallGeometry).
2094 #                                         copyProperties(MultiPolygon.first())])
2095 #                                         clipped = clipToPolygon(image,smallPolygon,size)
2096 #
2097 #                                         pixelCount = clipped.select(0).reduceRegion(
2098 #                                             reducer = ee.Reducer.count(),
2099 #                                             geometry = clipped.geometry(),
2100 #                                             scale = image.select(0).projection().nominalScale(),
2101 #                                             crs = image.select(0).projection().crs()
2102 #                                                 .get(image.bandNames().get(0)). getInfo()
2103 #                                             print('Pixel number for '+PolyNames[j]+ subpolygon '+str(k+1)+'
2104 #                                                   shranked): ',pixelCount)
2105 #
2106 #                                         if smallGeometry.type().getInfo() == 'Polygon':
2107 #                                             geemap.ee_export_vector_to_drive(
2108 #                                                 ee_object = extractPixelValuesSubPolys(
2109 #                                                     ee.ImageCollection(clipped) \
2110 #                                                       .select(['HH','HV']) \
2111 #                                                       .map(lambda image: image.addBands(image.select('HH'
2112 # ).toInt32().glcmTexture(size=size,kernel=None))) \
2113 #                                                       .map(lambda image: image.addBands(image.select('HV'
2114 # ).toInt32().glcmTexture(size=size,kernel=None))), \
2115 #                                                       smallPolygon,
2116 #                                                       numPixels),
2117 #                                                       description = 'TrainingValues_w'+str(size)+'_'
2118 #                                                       +str(int(numPixelsOriginal))+'pixels_Img'+str(i+1)+'_'+abbrev[j]+'_'
2119 #                                                       +str(k+1) +' _shranked',
2120 #                                                       folder = 'PixelValues_w'+str(size) +'_'+str(int(
2121 #                                                       numPixelsOriginal))+ 'pixels',
2122 #                                                       file_format = 'shp')
2123 #
2124 #                                             else:
2125 #                                                 geemap.ee_export_vector_to_drive(
2126 #                                                     ee_object = extractPixelValues(
2127 #                                                         ee.ImageCollection(clipped) \
2128 #                                                       .select(['HH','HV']) \
2129 #                                                       .map(lambda image: image.addBands(image.select('HH'
2130 # ).toInt32().glcmTexture(size=size,kernel=None))) \
2131 #                                                       .map(lambda image: image.addBands(image.select('HV'
2132 # ).toInt32().glcmTexture(size=size,kernel=None))), \
2133 #                                                       smallPolygon,
2134 #                                                       numPixels),
2135 #                                                       description = 'TrainingValues_w'+str(size) +'_'
2136 #                                                       +str(int(numPixelsOriginal))+'pixels_Img'+str(i+1) +'_'+abbrev[j]+'_'
2137 #                                                       +str(k+1) +' _shranked',
2138 #                                                       folder = 'PixelValues_w'+str(size) +'_'+str(int(

```

```

2139 #                         file_format = 'shp')
2140 #
2141 #             else:
2142 #                 geemap.ee_export_vector_to_drive(
2143 #                     ee_object = extractPixelValues(
2144 #                         ee.ImageCollection(clipped) \
2145 #                             .select(['HH','HV']) \
2146 #                             .map(lambda image: image.addBands(image.select('HH').toInt32().\
2147 #                                     glcmTexture(size=size,kernel=None))) \
2148 #                             .map(lambda image: image.addBands(image.select('HV').toInt32().\
2149 #                                     glcmTexture(size=size,kernel=None))), \
2150 #                                     MultiPolygon,
2151 #                                     numPixels),
2152 #                                     description = 'TrainingValues_w'+str(size)+'_'+str(int(numPixels))+'\
2153 # pixels_Img'+str(i+1)+'_'+abbrev[j],
2154 #                                     folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
2155 #                                     file_format = 'shp')
2156 #
2157 #             else:
2158 #                 geemap.ee_export_vector_to_drive(
2159 #                     ee_object = extractPixelValues(
2160 #                         ee.ImageCollection(clipped) \
2161 #                             .select(['HH','HV']) \
2162 #                             .map(lambda image: image.addBands(image.select('HH').toInt32().\
2163 #                                     glcmTexture(size=size,kernel=None))) \
2164 #                                     .map(lambda image: image.addBands(image.select('HV').toInt32().\
2165 #                                         TrainingPolys,
2166 #                                         numPixels),
2167 #                                         description = 'TrainingValues_w'+str(size)+'_'+str(int(numPixels))+'pixels_Img\
2168 # '+str(i+1)+'_All',
2169 #                                         folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+'pixels',
2170 #                                         file_format = 'shp')
2171 #
2172 ## Validation Data
2173 # pixelThreshold = 5e6
2174 #
2175 # VDList = ValidationData.toList(ValidationData.size())
2176 # for i in range(VDList.size().getInfo()):
2177 #     if not i == 2:
2178 #         continue
2179 #     numPixels = 1000
2180 #     image = ee.Image(VDList.get(i)).select(['HH','HV'])
2181 #     clipped = clipToPolygon(image,ValidationPolys,size)
2182 #
2183 #     MultiPolygons = ValidationPolys.filterMetadata('utc','equals',image.get('system:\
2184 # time_start'))
2185 #
2186 #     pixelCount = clipped.select(0).reduceRegion(
2187 #         reducer = ee.Reducer.count(),
2188 #         geometry = clipped.geometry(),
2189 #         scale = image.select(0).projection().nominalScale(),
2190 #         crs = image.select(0).projection().crs()
2191 #             .get(image.bandNames().get(0)).getInfo()
2192 #     print('Image ',str(i+1), ', pixel number for all polygons: ',pixelCount)
2193 #
2194 #     if pixelCount > pixelThreshold:
2195 #         PolyNames = ['Open Water','Rough Water','Flat Sea Ice','Floating Land Ice']
2196 #         abbrev = ['OW','RW','FSI','FLI']
2197 #         for j in range(len(PolyNames)):
2198 #             MultiPolygon = MultiPolygons.filterMetadata('name','equals',PolyNames[j])
2199 #             clipped = clipToPolygon(image,MultiPolygon,size)
2200 #
2201 #             pixelCount = clipped.select(0).reduceRegion(
2202 #                 reducer = ee.Reducer.count(),
2203 #                 geometry = clipped.geometry(),
2204 #                 scale = image.select(0).projection().nominalScale(),
2205 #                 crs = image.select(0).projection().crs()
2206 #                     .get(image.bandNames().get(0)).getInfo()
2207 #             print('Pixel number for '+PolyNames[j]+' polygons: ',pixelCount)
2208 #
2209 #             if pixelCount > pixelThreshold:
2210 #                 print('Subpolygons of one class')

```

```

2203 #             subPolys = MultiPolygon.geometry().geometries() \
2204 #                           .map(lambda geom: ee.Feature(ee.Geometry(geom)).copyProperties(
2205 #                               MultiPolygon.first()))
2206 #                           numSubPolys = subPolys.size(). getInfo()
2207 #                           for k in range(numSubPolys):
2208 #                               numPixels = np.ceil(numPixels/numSubPolys)
2209 #                               Polygon = ee.FeatureCollection(ee.Feature(subPolys.get(k)))
2210 #                               clipped = clipToPolygon(image,Polygon,size)
2211 #
2212 #                               pixelCount = clipped.select(0).reduceRegion(
2213 #                                   reducer = ee.Reducer.count(),
2214 #                                   geometry = clipped.geometry(),
2215 #                                   scale = image.select(0).projection().nominalScale(),
2216 #                                   crs = image.select(0).projection().crs()
2217 #                                   ).get(image.bandNames().get(0)).getInfo()
2218 #                               print('Pixel number for '+PolyNames[j] +' subpolygon '+str(k+1)+': ', 
2219 #                                   pixelCount)
2220 #
2221 #                               if pixelCount > pixelThreshold:
2222 #                                   shrink = -1*np.sqrt(pixelCount)/8*40
2223 #                                   smallGeometry = Polygon.geometry().buffer(shrink,maxError=1e3)
2224 #                                   smallPolygon = ee.FeatureCollection([ee.Feature(smallGeometry).
2225 #                                         copyProperties(MultiPolygon.first())])
2226 #                                   clipped = clipToPolygon(image,smallPolygon,size)
2227 #
2228 #                                   pixelCount = clipped.select(0).reduceRegion(
2229 #                                       reducer = ee.Reducer.count(),
2230 #                                       geometry = clipped.geometry(),
2231 #                                       scale = image.select(0).projection().nominalScale(),
2232 #                                       crs = image.select(0).projection().crs()
2233 #                                       ).get(image.bandNames().get(0)).getInfo()
2234 #                                   print('Pixel number for '+PolyNames[j] +' subpolygon '+str(k+1)+'
2235 #                                         shrinked): ',pixelCount)
2236 #
2237 #                               if smallGeometry.type().getInfo() == 'Polygon':
2238 #                                   geemap.ee_export_vector_to_drive(
2239 #                                       ee_object = extractPixelValuesSubPolys(
2240 #                                           ee.ImageCollection(clipped) \
2241 #                                           .select(['HH','HV']) \
2242 #                                           .map(lambda image: image.addBands(image.select('HH
2243 #                                         ').toInt32().glcmTexture(size=size,kernel=None))) \
2244 #                                           .map(lambda image: image.addBands(image.select('HV
2245 #                                         ').toInt32().glcmTexture(size=size,kernel=None))), 
2246 #                                           smallPolygon,
2247 #                                           numPixels),
2248 #                                           description = 'ValidationValues_w'+str(size)+'_'+str(int(
2249 #                                         numPixels))+''pixels_Img'+str(i+1) +'_'+abbrev[j]+'_'+str(k+1) +'_shrinked',
2250 #                                           folder = 'PixelValues_w'+str(size) +'_'+str(int(numPixels))
2251 #                                         +'pixels',
2252 #                                           file_format = 'shp')
2253 #
2254 #                               else:
2255 #                                   geemap.ee_export_vector_to_drive(
2256 #                                       ee_object = extractPixelValues(
2257 #                                           ee.ImageCollection(clipped) \
2258 #                                           .select(['HH','HV']) \
2259 #                                           .map(lambda image: image.addBands(image.select('HH
2260 #                                         ').toInt32().glcmTexture(size=size,kernel=None))) \
2261 #                                           .map(lambda image: image.addBands(image.select('HV

```

```

2262 #                                     .select(['HH','HV'])\ 
2263 #                                         .map(lambda image: image.addBands(image.select('HH')\ 
2264 #                                         .toInt32().glcmTexture(size=size,kernel=None))\ 
2265 #                                         .map(lambda image: image.addBands(image.select('HV')\ 
2266 #                                         .toInt32().glcmTexture(size=size,kernel=None))),\ 
2267 #                                         Polygon,\ 
2268 #                                         numPixels),\ 
2269 #                                         description = 'ValidationValues_w'+str(size)+'_'+str(int(\ 
2270 #                                         numPixels))+pixels_Img+str(i+1)+'' +abbrev[j]+'' +str(k+1),\ 
2271 #                                         folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+\ 
2272 #                                         pixels',\ 
2273 #                                         file_format = 'shp')
2274 #
2275 else:
2276     geemap.ee_export_vector_to_drive(
2277         ee_object = extractPixelValues(
2278             ee.ImageCollection(clipped)\ 
2279             .select(['HH','HV'])\ 
2280             .map(lambda image: image.addBands(image.select('HH').toInt32().\ 
2281                 glcmTexture(size=size,kernel=None)))\ 
2282                 .map(lambda image: image.addBands(image.select('HV').toInt32().\ 
2283                     glcmTexture(size=size,kernel=None))),\ 
2284                 MultiPolygon,\ 
2285                 numPixels),\ 
2286                 description = 'ValidationValues_w'+str(size)+'_'+str(int(numPixels))+\ 
2287                 pixels_Img+str(i+1)+'' +abbrev[j],\ 
2288                 folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+pixels',\ 
2289                 file_format = 'shp')
2290 else:
2291     geemap.ee_export_vector_to_drive(
2292         ee_object = extractPixelValues(
2293             ee.ImageCollection(clipped)\ 
2294             .select(['HH','HV'])\ 
2295             .map(lambda image: image.addBands(image.select('HH').toInt32().\ 
2296                 glcmTexture(size=size,kernel=None)))\ 
2297                 .map(lambda image: image.addBands(image.select('HV').toInt32().\ 
2298                     glcmTexture(size=size,kernel=None))),\ 
2299                     ValidationPolys,\ 
2300                     numPixels),\ 
2301                     description = 'ValidationValues_w'+str(size)+'_'+str(int(numPixels))+\ 
2302                     pixels_Img+str(i+1)+'' _All',\ 
2303                     folder = 'PixelValues_w'+str(size)+'_'+str(int(numPixels))+pixels',\ 
2304                     file_format = 'shp')
2305 
2306 #### (5.3.2) Import Sampled Data Subsets & Export as Full Data Set
2307 ## Convert feature collections per image to full feature collection for training/validation
2308     data sets:
2309 ## NOTE: edit earth engine name to that of personal account if you want to redo this process
2310 ## GLCM11:
2311 # numPixels = 1000
2312 # size = 11
2313 
2314 # TV11_list = []
2315 # for i in range(S1Training.size().getInfo()):
2316 #     fc = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w11_1000pixels_Img'+str(i+1)
2317 # )
2318 
2319 # TV11 = ee.FeatureCollection(TV11_list).flatten()
2320 
2321 # geemap.ee_export_vector_to_drive(
2322 #     ee_object = TV11,
2323 #     description = 'TrainingValues_w11_1000pixels',
2324 #     folder = 'PixelValues_w11_1000pixels',
2325 #     file_format = 'shp')
2326 
2327 ## GLCM21:
2328 # numPixels = 1000
2329 # size = 21
2330 
2331 # trainNames = [
2332     'TrainingValues_w21_1000pixels_Img1_All',

```

```

2321 #     'TrainingValues_w21_1000pixels_Img2_All',
2322 #     'TrainingValues_w21_1000pixels_Img3_All',
2323 #     'TrainingValues_w21_1000pixels_Img4_All',
2324 #     'TrainingValues_w21_1000pixels_Img5_All',
2325 #     'TrainingValues_w21_1000pixels_Img6_FSI',
2326 #     'TrainingValues_w21_1000pixels_Img6_FLI',
2327 #     'TrainingValues_w21_1000pixels_Img6_OW',
2328 #     'TrainingValues_w21_1000pixels_Img6_RW_1_shrinked',
2329 #     'TrainingValues_w21_1000pixels_Img6_RW_2',
2330 #     'TrainingValues_w21_1000pixels_Img6_RW_3',
2331 #     'TrainingValues_w21_1000pixels_Img6_RW_4',
2332 #     'TrainingValues_w21_1000pixels_Img6_RW_5',
2333 #     'TrainingValues_w21_1000pixels_Img6_RW_6',
2334 #     'TrainingValues_w21_1000pixels_Img6_RW_7',
2335 #     'TrainingValues_w21_1000pixels_Img7_All',
2336 #     'TrainingValues_w21_1000pixels_Img8_All',
2337 #     'TrainingValues_w21_1000pixels_Img9_All'
2338 # ]
2339
2340 # TV21_list = []
2341 # for i in range(len(trainNames)):
2342 #     fc = ee.FeatureCollection('users/skjeltmaps/' + trainNames[i])
2343 #     TV21_list.append(fc)
2344 # TV21 = ee.FeatureCollection(TV21_list).flatten()
2345
2346 # geemap.ee_export_vector_to_drive(
2347 #     ee_object = TV21,
2348 #     description = 'TrainingValues_w21_1000pixels',
2349 #     folder = 'PixelValues_w21_1000pixels',
2350 #     file_format = 'shp')
2351
2352 # valNames = [
2353 #     'ValidationValues_w21_1000pixels_Img1_All',
2354 #     'ValidationValues_w21_1000pixels_Img2_All',
2355 #     'ValidationValues_w21_1000pixels_Img3_FSI',
2356 #     'ValidationValues_w21_1000pixels_Img3_FLI',
2357 #     'ValidationValues_w21_1000pixels_Img3_OW',
2358 #     'ValidationValues_w21_1000pixels_Img3_RW',
2359 #     'ValidationValues_w21_1000pixels_Img4_All',
2360 # ]
2361
2362 # VV21_list = []
2363 # for i in range(len(valNames)):
2364 #     featcol = ee.FeatureCollection('users/skjeltmaps/' + valNames[i])
2365 #     VV21_list.append(featcol)
2366 # VV21 = ee.FeatureCollection(VV21_list).flatten()
2367
2368 # geemap.ee_export_vector_to_drive(
2369 #     ee_object = VV21,
2370 #     description = 'ValidationValues_w21_1000pixels',
2371 #     folder = 'PixelValues_w21_1000pixels',
2372 #     file_format = 'shp')

```

B.6. Feature Selection

6_FeatureSelection

```

1 %% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5 import numpy as np
6 import os
7 import matplotlib.pyplot as plt
8 import scipy.stats as stats
9
10 from sklearn.feature_selection import RFECV
11 from sklearn.feature_selection import RFE
12 from sklearn.ensemble import RandomForestClassifier
13 from sklearn.model_selection import StratifiedKFold

```

```

14 from sklearn.model_selection import RandomizedSearchCV
15 from sklearn.model_selection import GridSearchCV
16
17 ## (1.2) Initialize
18 ee.Initialize()
19
20 ## (1.3) Parameterization & Data Import
21 SMALL_SIZE = 24
22 MEDIUM_SIZE = 28
23 BIGGER_SIZE = 32
24
25 plt.rc('font', size=SMALL_SIZE)           # controls default text sizes
26 plt.rc('axes', titlesize=SMALL_SIZE)       # fontsize of the axes title
27 plt.rc('axes', labelsize=MEDIUM_SIZE)      # fontsize of the x and y labels
28 plt.rc('xtick', labelsize=SMALL_SIZE)       # fontsize of the tick labels
29 plt.rc('ytick', labelsize=SMALL_SIZE)       # fontsize of the tick labels
30 plt.rc('legend', fontsize=SMALL_SIZE)        # legend fontsize
31 plt.rc('figure', titlesize=BIGGER_SIZE)     # fontsize of the figure title
32
33 properties = ['', '_asm', '_contrast', '_corr', '_dent', '_diss', '_dvar', '_ent', '_idm', '_imcorr1',
   '_imcorr2', '_inertia', '_prom',
   '_savg', '_sent', '_shade', '_svar', '_var']
34 names = ['Intensity', 'Angular Second Moment', 'Contrast', 'Correlation', 'Difference Entropy',
   'Dissimilarity',
   'Difference Variance', 'Entropy', 'Inverse Distance Moment', 'Information Measure 1 of
   Correlation',
   'Information Measure 2 of Correlation', 'Inertia', 'Cluster Prominence', 'Sum Average',
   'Sum Entropy', 'Cluster Shade',
   'Sum Variance', 'Variance']
35
36 property_names = []
37 feature_names = []
38 for i in range(len(properties)):
39     name = names[i]
40     prop = properties[i]
41
42     feature_names.append(name+' (HH)')
43     feature_names.append(name+' (HV)')
44
45     property_names.append('HH'+prop)
46     property_names.append('HV'+prop)
47
48 TV11 = ee.FeatureCollection('users/skjeltmaps/TrainingValues_w11_1000pixels') .map(lambda
49     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
50     HV_contras')))
51 TV5 = ee.FeatureCollection('users/skjeltmaps/TrainingValues_w5_1000pixels') .map(lambda feat
52     : feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('HV_
53     contras')))
54 TV21 = ee.FeatureCollection('users/skjeltmaps/TrainingValues_w21_1000pixels') .map(lambda
55     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
56     HV_contras')))
57 TV0 = TV11.select(['HH', 'HV', 'classes', 'subclasses', 'name', 'utc', 'type', 'image'])
58
59 VV11 = ee.FeatureCollection('users/skjeltmaps/ValidationValues_w11_1000pixels') .map(lambda
60     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
61     HV_contras')))
62 VV5 = ee.FeatureCollection('users/skjeltmaps/ValidationValues_w5_1000pixels') .map(lambda
63     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
64     HV_contras')))
65 VV21 = ee.FeatureCollection('users/skjeltmaps/ValidationValues_w21_1000pixels') .map(lambda
66     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
67     HV_contras')))
68 VV0 = VV11.select(['HH', 'HV', 'classes', 'subclasses', 'name', 'utc', 'type', 'image'])

#%% (2) Retrieve Sample Data
## (2.1) Function
def SampleDataClientSide(eeTraining,eeValidation,property_names,n_features):
    n_samples_training = eeTraining.size().getInfo()
    train_data = np.empty([n_samples_training,n_features])

    for i in range(n_features):

```

```

69     train_data[:,i] = eeTraining.aggregate_array(property_names[i]).getInfo()
70
71     train_classes = eeTraining.aggregate_array('classes').getInfo()
72     train_subclasses = eeTraining.aggregate_array('subclasses').getInfo()
73
74     return [train_data, train_classes, train_subclasses]
75
76 ## (2.2) Download Data to Client Side
77 sample_data_glc5 = SampleDataClientSide(TV5,VV5,property_names,len(property_names))
78 sample_data_glc11 = SampleDataClientSide(TV11,VV11,property_names,len(property_names))
79 sample_data_glc21 = SampleDataClientSide(TV21,VV21,property_names,len(property_names))
80
81 #%% (3) Correlation
82 ## (3.1) Function
83 def heavilyCorrelated(data,property_names,threshold):
84     corr = np.corrcoef(np.array(data),rowvar=False)
85     pos_corr = np.logical_and(corr > threshold,corr < 0.999)
86     neg_corr = np.logical_and(corr < -threshold,corr > -0.999)
87     boolean = np.logical_or(pos_corr,neg_corr)
88     correlated_names = []
89     correlated = []
90     for i in range(len(data[0,:])):
91         corr_names = [corrs for corrs,check in zip(property_names, boolean[i]) if check]
92         correlated.append(corr_names)
93
94         corr_values = [vals for vals,check in zip(corr[i], boolean[i]) if check]
95         correlated.append(corr_values)
96
97         print(property_names[i]+': '+str(corr_names)+', values: '+str(corr_values))
98     return [correlated_names, correlated]
99
100 ## (3.2) Compute Correlation Between Features
101 correlations = heavilyCorrelated(sample_data_glc11[0],property_names,0.9)
102
103 #%% (4) Hyperparameter Test
104 # Number of trees in random forest
105 n_estimators = np.arange(10,100)
106 # Maximum number of levels in tree
107 max_depth = np.arange(1,30)
108 # Minimum number of samples required to split a node
109 min_samples_split = np.arange(1,10)
110 # Minimum number of leafs required to be at a leaf node
111 min_samples_leaf = np.arange(1,10)
112 # Create the random grid
113 parameters = [n_estimators,max_depth,min_samples_split,min_samples_leaf]
114 parameter_names = ['n_estimators','max_depth','min_samples_split','min_samples_leaf']
115 random_grid = {'n_estimators': n_estimators,
116                 'max_depth': max_depth,
117                 'min_samples_split': min_samples_split,
118                 'min_samples_leaf': min_samples_leaf}
119
120 rf_base = RandomForestClassifier()
121 rf_random = RandomizedSearchCV(estimator = rf_base, param_distributions = random_grid,
122                                 n_iter = 200, cv = 3, verbose = 2, random_state = 1,
123                                 n_jobs = -1)
124 rf_grid = GridSearchCV(estimator = rf_base, param_grid = random_grid,
125                         cv = 3, verbose = 2, n_jobs = -1)
126
127 # Fit the random search model
128 # rf_random.fit(sample_data_glc11[0][:,:2], sample_data_glc11[1])
129 # rf_grid.fit(training_data0, training_labels0)
130 # View the best parameters from the random search
131 # print(rf_random.best_params_)
132 # {'n_estimators': 75, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_depth': 3}
133
134 rf = RandomForestClassifier(
135     n_estimators = 10,
136     max_depth = None,
137     min_samples_split = 2,
138     min_samples_leaf = 1,
139     random_state = 1

```

```
140 )
141
142 %% (5) Recursive Feature Elimination with Cross Validation
143 ## (5.1) Functions
144 def RFE_CV(rf,w,feature_names,classification_label,visualize=True):
145
146     sampledata = eval('sample_data_glc'+str(w))
147     training_data = sampledata[0]
148     if classification_label == 'classes':
149         labels = sampledata[1]
150         classification_type = 'Water/Ice'
151     if classification_label == 'subclasses':
152         labels = sampledata[2]
153         classification_type = 'OW/RW/SI/FLI'
154
155     rfecv = RFECV(estimator=rf, step=1, cv=StratifiedKFold(3), scoring='accuracy')
156     rfecv.fit(training_data,labels)
157
158     features = [f for f,s in zip(feature_names, rfecv.support_) if s]
159     accuracies = np.mean(rfecv.grid_scores_,axis=1)
160
161     if visualize:
162         titlename = '{} Classification, '.format(classification_type)+ 'including GLCM
163             textures (w={})'.format(str(w))
164
165         plt.figure(figsize=(16, 9))
166         plt.title(titlename, fontweight='bold', pad=20)
167         plt.xlabel('Number of selected features', labelpad=20)
168         plt.ylabel('Accuracy [-]', labelpad=20)
169         plt.plot(range(1,len(feature_names)+1), accuracies, color='#303F9F', linewidth=3)
170         plt.xlim(1,len(feature_names))
171         plt.grid()
172         plt.show()
173
174     return features, accuracies
175
176 def FeatureImportanceScores(rf,w,feature_names,best_features,classification_label,visualize=
177     True):
178     sampledata = eval('sample_data_glc'+str(w))
179     training_data = sampledata[0]
180     if classification_label == 'classes':
181         labels = sampledata[1]
182         classification_type = 'Water/Ice'
183     if classification_label == 'subclasses':
184         labels = sampledata[2]
185         classification_type = 'OW/RW/SI/FLI'
186
187     best_indeces = []
188     for i in range(len(best_features)):
189         best_indeces.append(np.argwhere(np.array(feature_names) == best_features[i])[0][0])
190     best_feature_data = training_data[:,best_indeces]
191
192     fitted = rf.fit(best_feature_data,labels)
193     importances = fitted.feature_importances_
194     sorted_indeces = importances.argsort()
195     features_sorted = [best_features[index] for index in sorted_indeces]
196     importances_sorted = np.sort(importances)
197
198     if visualize:
199         titlename = '{} Classification, '.format(classification_type)+ 'including GLCM
200             textures (w={})'.format(str(w))
201
202         fig, ax = plt.subplots(figsize=(16, 9))
203         ax.xaxis.grid(True)
204         ax.barh(features_sorted,importances_sorted)
205
206         plt.title(titlename, fontweight='bold', pad=20)
207         plt.ylabel('Feature name', labelpad=20)
208         plt.xlabel('Feature importance score [-]')
209         plt.show()
```

```

208     return importances
209
210 def StatisticalSignificance(rf,w,feature_names,classification_label,visualize=True):
211     sampledata = eval('sample_data_glcw'+str(w))
212     training_data = sampledata[0]
213     if classification_label == 'classes':
214         labels = sampledata[1]
215         classification_type = 'Water/Ice'
216     if classification_label == 'subclasses':
217         labels = sampledata[2]
218         classification_type = 'OW/RW/SI/FLI'
219
220     predicted_labels = []
221     best_features_list = []
222
223     feature_iterations = np.arange(2,len(feature_names)+1)
224     for _,n_features in enumerate(feature_iterations):
225
226         rfe = RFE(estimator=rf, n_features_to_select=n_features, step=1)
227         fitted_all = rfe.fit(training_data,labels)
228
229         best_features = [f for f,s in zip(feature_names, fitted_all.support_) if s]
230         best_indeces = []
231         for i in range(len(best_features)):
232             best_indeces.append(np.argwhere(np.array(feature_names) == best_features[i])[0][0])
233         best_feature_data = training_data[:,best_indeces]
234
235         fitted_best = rfe.fit(best_feature_data,labels)
236         predicted_labels.append(fitted_best.predict(best_feature_data))
237
238         pvalues = [0]
239         for i in range(len(predicted_labels)):
240             _, pval = stats.ttest_ind(labels,predicted_labels[i])
241             pvalues.append(pval)
242
243     if visualize:
244         titlename = 'T-test p-values of predicted vs actual class labels, including GLCM
245             textures (w={})'.format(str(w))
246
247         plt.figure(figsize=(16, 9))
248         plt.title(titlename, fontweight='bold', pad=20)
249         plt.xlabel('Number of selected features', labelpad=20)
250         plt.ylabel('p-value [-]', labelpad=20)
251         plt.plot(range(1, len(feature_names) + 1), pvalues, color='#303F9F', linewidth=3)
252         plt.hlines(0.05,1,len(feature_names),colors='#303F9F',linestyles='dashed')
253         plt.legend(['p-values','Significance threshold'])
254
255         plt.xlim(1,len(feature_names))
256         plt.ylim(bottom=0)
257         plt.grid(axis='x')
258         plt.show()
259
260     return pvalues
261
262 def FeatureSelectionFigure(w,classification_label,feature_names,rfecv_features,rfecv_scores,
263     importances,pvalues,n_features):
264     if classification_label == 'classes':
265         classification_type = 'Water/Ice'
266     if classification_label == 'subclasses':
267         classification_type = 'OW/RW/SI/FLI'
268
269     fig, axes = plt.subplots(1,2,figsize=(30,10))
270     plt.suptitle('GLCM{} Feature Selection'.format(str(w)),fontweight='bold')
271
272     ymin = 0.99*np.min(rfecv_scores)
273     ymax = 1.01*np.max(rfecv_scores)
274     subplotname = 'RFECV Accuracies & Associated p-values'# and T-test p-values per chosen
275             number of features'
276
277     if len(rfecv_features) < 18:

```

```

275     xbound = 0.98
276     else:
277         xbound = len(rfecv_features)/36*0.95
278     ybound = 0.2
279
280     axes[0].title.set_text(subplotname)
281     axes[0].title.set_fontweight('bold')
282     axes[0].set_xlabel('Number of selected features', labelpad=20, color='black')
283     axes[0].set_ylabel('Accuracy [-]', labelpad=20, color='black')
284     axes[0].plot(range(1,len(feature_names)+1), rfecv_scores, color='black', linewidth=3)
285     optimal_line = axes[0].vlines(len(rfecv_features),ymin,ymax,colors='black',linestyles='dashed')
286     chosen_line = axes[0].vlines(n_features,ymin,ymax,colors='tab:blue',linestyles='dashed')
287     axes[0].set_ylim(ymin,ymax)
288     axes[0].set_xlim(1,len(feature_names))
289     axes[0].grid(axis='x')
290     axes[0].legend([optimal_line,chosen_line],['Optimal','Chosen'],loc='best',bbox_to_anchor=(xbound,ybound))
291
292     axesTwin = axes[0].twinx()
293     axesTwin.set_ylabel('p-value [-]', labelpad=20, color='black')
294     axesTwin.plot(range(1, len(feature_names) + 1), pvalues, color='red', linewidth=3)
295     axesTwin.tick_params(axis='y', labelcolor='red', labelsize=25)
296     axesTwin.set_ylim(0,1)
297
298     sorted_indeces = importances.argsort()
299     features_sorted = [rfecv_features[index] for index in sorted_indeces]
300     importances_sorted = np.sort(importances)
301     importancename = 'Feature Importances for n={}'.format(len(rfecv_features))
302
303     axes[1].title.set_text(importancename)
304     axes[1].title.set_fontweight('bold')
305     axes[1].xaxis.grid(True)
306     axes[1].barh(features_sorted,importances_sorted)
307     chosen = axes[1].hlines(len(features_sorted)-n_features-0.5,0,1,colors='black',linestyles='dashed')
308     axes[1].set_xlim(0,importances_sorted[-1]*1.01)
309     axes[1].set_ylabel('Feature Name', labelpad=20, fontsize=30)
310     axes[1].set_xlabel('Feature Importance Score [-]',fontsize=30)
311
312     plt.tight_layout()
313
314     return fig
315
316 ## (5.2) GLCM5
317 features_glc5, accuracies_glc5 = RFE_CV(rf,5,feature_names,'classes',visualize=False)
318 importances_glc5 = FeatureImportanceScores(rf,5,feature_names,features_glc5,'classes',
319                                         visualize=False)
320 pvalues_glc5 = StatisticalSignificance(rf,5,feature_names,'classes',visualize=False)
321 n = 6
322 featureselection_glc5 = FeatureSelectionFigure(5,'classes',property_names,features_glc5,
323                                                 accuracies_glc5,importances_glc5,
324                                                 pvalues_glc5,n)
325
326 sorted_indeces = importances_glc5.argsort()
327 features_sorted = [features_glc5[index] for index in sorted_indeces][::-1][:n]
328 correlations = heavilyCorrelated(sample_data_glc5[0],property_names,0.9)
329 corr = []
330 for i in range(len(features_sorted)):
331     index = np.argwhere(np.array(feature_names) == features_sorted[i])[0][0]
332     print(features_sorted[i]+': '+str(correlations[0][index]))
333     print(features_sorted[i]+': '+str(correlations[1][index]))
334
335 ## (5.3) GLCM11
336 features_glc11, accuracies_glc11 = RFE_CV(rf,11,feature_names,'classes',visualize=False)
337 importances_glc11 = FeatureImportanceScores(rf,11,feature_names,features_glc11,'classes',
338                                         visualize=False)
339 pvalues_glc11 = StatisticalSignificance(rf,11,feature_names,'classes',visualize=False)
340
341 n = 6

```

```

340 featureselection_glcml1 = FeatureSelectionFigure(11,'classes',property_names,features_glcml1,
341                                         accuracies_glcml1,importances_glcml1,
342                                         pvalues_glcml1,n)
343
344 sorted_indeces = importances_glcml1.argsort()
345 features_sorted = [features_glcml1[index] for index in sorted_indeces][::-1][:n]
346 correlations = heavilyCorrelated(sample_data_glcml1[0],property_names,0.9)
347 corr = []
348 for i in range(len(features_sorted)):
349     index = np.argwhere(np.array(feature_names) == features_sorted[i])[0][0]
350     print(features_sorted[i]+': '+str(correlations[0][index]))
351     print(features_sorted[i]+': '+str(correlations[1][index]))
352
353 ## (5.4) GLCM21
354 features_glcml21, accuracies_glcml21 = RFE_CV(rf,21,feature_names,'classes',visualize=False)
355 importances_glcml21 = FeatureImportanceScores(rf,21,feature_names,features_glcml21,'classes',
356                                         visualize=False)
357 pvalues_glcml21 = StatisticalSignificance(rf,21,feature_names,'classes',visualize=False)
358 n = 5
359 featureselection_glcml21 = FeatureSelectionFigure(21,'classes',property_names,features_glcml21,
360                                         accuracies_glcml21,importances_glcml21,
361                                         pvalues_glcml21,n)
362
363 sorted_indeces = importances_glcml21.argsort()
364 features_sorted = [features_glcml21[index] for index in sorted_indeces][::-1][:n]
365 correlations = heavilyCorrelated(sample_data_glcml21[0],property_names,0.9)
366 corr = []
367 for i in range(len(features_sorted)):
368     index = np.argwhere(np.array(feature_names) == features_sorted[i])[0][0]
369     print(features_sorted[i]+': '+str(correlations[0][index]))
370     print(features_sorted[i]+': '+str(correlations[1][index]))

```

B.7. Visualization of Feature Data

7_DataVisualization

```

1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5 import numpy as np
6 import os
7 import matplotlib.pyplot as plt
8
9 ## (1.2) Initialize
10 ee.Initialize()
11
12 ## (1.3) Parameterization & Data Import
13 properties = ['HH','HV','HH_savg','HV_savg','HV_dvar']
14 classes = ['Water','Ice']
15 subclasses = ['Open Water','Rough Water','Sea Ice','Floating Land Ice']
16 colors = ['C0','white','C0','C0','White','White']
17 fontsize = 25
18
19 TV11 = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w11_1000pixels') .map(lambda
20     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get(
21         'HV_contras')))
22 TV5 = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w5_1000pixels') .map(lambda feat
23     : feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('HV_contras'
24         )))
25 TV21 = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w21_1000pixels') .map(lambda
26     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get(
27         'HV_contras')))
28
29 #%% (2) Boxplots of Data Sets
30 ## (2.1) Functions
31 def makeBoxPlot(subplot,data,propertyname,ylims,fontsize):

```

```
26 plt.rcParams.update({'axes.facecolor':'lightgrey'})
27 plt.subplot(subplot[0],subplot[1],subplot[2])
28
29 bp = plt.boxplot(data, notch=0, sym='+', vert=1, whis=1.5, patch_artist=True,
30                   boxprops={"edgecolor": "white","linewidth": 0.5})
31 for patch, color in zip(bp['boxes'], colors):
32     patch.set_facecolor(color)
33 for median in bp['medians']:
34     median.set_color('black')
35
36 xname = ''
37 plt.xlabel(xname, fontsize=fontsize)
38 yname = propertyname
39 plt.ylabel(yname, fontsize=fontsize)
40
41 plt.xticks(np.arange(1, len(data)+1),['Water','Ice','Open\nWater','Rough\nWater','Sea\nIce',
42                      ',\nFloating\nLand Ice'])
43 plt.tick_params(labelsize=fontsize, labelrotation=0)
44
45 plt.vlines(2.5,ylims[0],ylims[1],'black','dashed')
46 plt.ylim(ylims)
47 plt.grid(axis='y',color='black', linewidth=0.5)
48
49 return _
50
51 ## (2.2) GLCM5
52 sampledata = [[],[],[],[],[]]
53
54 for i in range(len(properties)):
55     prop = properties[i]
56     for j,subname in enumerate(classes):
57         sampledata[i].append(TV5.filterMetadata('classes','equals',j+1).aggregate_array(prop)
58                               .getInfo())
59     for k,name in enumerate(subclasses):
60         sampledata[i].append(TV5.filterMetadata('subclasses','equals',k+1).aggregate_array(
61                               prop).getInfo())
62
63 plt.figure(figsize=(30,20))
64 _ = makeBoxPlot([2,2,1],sampledata[0],'Intensity (HH) [dB]',[-55,5],fontsize)
65 _ = makeBoxPlot([2,2,3],sampledata[1],'Intensity (HV) [dB]',[-60,0],fontsize)
66 _ = makeBoxPlot([3,2,2],sampledata[2],'Sum Average\nn(HH) [-]',[-85,5],fontsize)
67 _ = makeBoxPlot([3,2,4],sampledata[3],'Sum Average\nn(HV) [-],[-105,-5],fontsize)
68 _ = makeBoxPlot([3,2,6],sampledata[4],'Difference Variance\nn(HV) [-],[-2,23],fontsize)
69
70 plt.show()
71
72 ## (2.3) GLCM11
73 sampledata = [[],[],[],[],[]]
74
75 for i in range(len(properties)):
76     prop = properties[i]
77     for j,subname in enumerate(classes):
78         sampledata[i].append(TV11.filterMetadata('classes','equals',j+1).aggregate_array(prop)
79                               .getInfo())
80     for k,name in enumerate(subclasses):
81         sampledata[i].append(TV11.filterMetadata('subclasses','equals',k+1).aggregate_array(
82                               prop).getInfo())
83
84 plt.figure(figsize=(30,20))
85 _ = makeBoxPlot([2,2,1],sampledata[0],'Intensity (HH) [dB]',[-55,5],fontsize)
86 _ = makeBoxPlot([2,2,3],sampledata[1],'Intensity (HV) [dB]',[-60,0],fontsize)
87 _ = makeBoxPlot([3,2,2],sampledata[2],'Sum Average\nn(HH) [-]',[-85,5],fontsize)
88 _ = makeBoxPlot([3,2,4],sampledata[3],'Sum Average\nn(HV) [-],[-105,-5],fontsize)
89 _ = makeBoxPlot([3,2,6],sampledata[4],'Difference Variance\nn(HV) [-],[-2,23],fontsize)
90
91 plt.show()
92
93 ## (2.4) GLCM21
94 sampledata = [[],[],[],[],[]]
95
96 for i in range(len(properties)):
```

```

92     prop = properties[i]
93     for j, subname in enumerate(classes):
94         sampledata[i].append(TV21.filterMetadata('classes', 'equals', j+1).aggregate_array(prop
95             ).getInfo())
96     for k, name in enumerate(subclasses):
97         sampledata[i].append(TV21.filterMetadata('subclasses', 'equals', k+1).aggregate_array(
98             prop).getInfo())
99
100 plt.figure(figsize=(30,20))
101 _ = makeBoxPlot([2,2,1],sampledata[0],'Intensity (HH) [dB]',[-55,5],fontsize)
102 _ = makeBoxPlot([2,2,3],sampledata[1],'Intensity (HV) [dB]',[-60,0],fontsize)
103 _ = makeBoxPlot([3,2,2],sampledata[2],'Sum Average\n(HH) [-]',[-85,5],fontsize)
104 _ = makeBoxPlot([3,2,4],sampledata[3],'Sum Average\n(HV) [-],[-105,-5],fontsize)
105 _ = makeBoxPlot([3,2,6],sampledata[4],'Difference Variance\n(HV) [-],[-2,23],fontsize)
106 plt.show()

```

B.8. Classification & Accuracy Assessment

8_Classification

```

1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5 import numpy as np
6 import os
7 import pandas as pd
8 import matplotlib.pyplot as plt
9 import seaborn as sns
10
11 ## (1.2) Initialize
12 ee.Initialize()
13
14 ## (1.3) Parameterization & Data Import
15 classifier = ee.Classifier.smileRandomForest(
16     number_of_trees = 10,
17     variables_per_split = 2,
18     max_nodes = None,
19     min_leaf_population = 1,
20     seed = 1)
21
22 plt.style.use("seaborn")
23 SMALL_SIZE = 18
24 MEDIUM_SIZE = 20
25 BIGGER_SIZE = 22
26
27 plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
28 plt.rc('axes', titlesize=SMALL_SIZE)      # fontsize of the axes title
29 plt.rc('axes', labelsize=MEDIUM_SIZE)      # fontsize of the x and y labels
30 plt.rc('xtick', labelsize=SMALL_SIZE)      # fontsize of the tick labels
31 plt.rc('ytick', labelsize=SMALL_SIZE)      # fontsize of the tick labels
32 plt.rc('legend', fontsize=SMALL_SIZE)      # legend fontsize
33 plt.rc('figure', titlesize=BIGGER_SIZE)    # fontsize of the figure title
34
35 TV11 = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w11_1000pixels') .map(lambda
36     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
37     HV_contras')))
38 TV5 = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w5_1000pixels') .map(lambda feat
39     : feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('HV_contras'
40     )))
41 TV21 = ee.FeatureCollection('users/skjeltnmaps/TrainingValues_w21_1000pixels') .map(lambda
42     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
43     HV_contras')))
44 TV0 = TV11.select(['HH','HV','classes','subclasses','name','utc','type','image'])
45
46 VV11 = ee.FeatureCollection('users/skjeltnmaps/ValidationValues_w11_1000pixels') .map(lambda
47     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
48     HV_contras')))

```

```

41 VV5 = ee.FeatureCollection('users/skjeltmaps/ValidationValues_w5_1000pixels') .map(lambda
42     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
43     HV_contras'))))
44 VV21 = ee.FeatureCollection('users/skjeltmaps/ValidationValues_w21_1000pixels') .map(lambda
45     feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
46     HV_contras'))))
47 VV0 = VV11.select(['HH','HV','classes','subclasses','name','utc','type','image'])
48
49
50 #%% (2) Accuracy Assessment
51 ## (2.1) BASE
52 TV = TV0
53 VV = VV0
54
55 n_features_training = TV.size(). getInfo()
56 n_features_validation = VV.size(). getInfo()
57
58 classchoices = ['classes','subclasses']
59 bandchoices = [['HH','HV']]
60 dataframes0 = []
61 accuracies0 = []
62 kappas0 = []
63
64 for i in range(len(bandchoices)):
65     bands = bandchoices[i]
66     print('Band choice: ',bands)
67
68     for j in range(len(classchoices)):
69         classchoice = classchoices[j]
70         print('Class choice: ',classchoice)
71         if classchoice == 'classes':
72             classnames = ['Water','Ice']
73         elif classchoice == 'subclasses':
74             classnames = ['Open Water','Rough Water','Sea Ice','Floating Land Ice']
75
76         trained = classifier.train(TV,classchoice,bands)
77         validated = VV.classify(trained)
78
79         confusion_matrix_validation = np.asarray(validated.errorMatrix(classchoice,'
80             classification').getInfo())[1:,1:]
81         dataframes0.append(pd.DataFrame(confusion_matrix_validation/
82             confusion_matrix_validation.sum(axis=1),columns=classnames,index=classnames))
83
84         print('Validation Data Results:')
85         print(dataframes0[j].round(3))
86         accuracies0.append(validated.errorMatrix(classchoice,'classification').accuracy().
87             getInfo())
88         kappas0.append(validated.errorMatrix(classchoice,'classification').kappa().getInfo())
89         print('Overall accuracy: ',np.round(accuracies0[j],4),' (kappa: ',np.round(kappas0[j]
90             ],4),')')
91
92 ## (2.2) GLCM5
93 TV = TV5
94 VV = VV5
95 print('Window size: 5')
96
97 n_features_training = TV.size(). getInfo()
98 n_features_validation = VV.size(). getInfo()
99
100 classchoices = ['classes','subclasses']
101 bandchoices = [
102     ['HH_savg', 'HV_savg', 'HV_dvar']
103 ]
104 dataframes5 = []
105 accuracies5 = []
106 kappas5 = []
107
108 for i in range(len(bandchoices)):
109     bands = bandchoices[i]
110     print('Band choice: ',bands)
111
112     for j in range(len(classchoices)):
113
114         for k in range(len(classnames)):
115             classchoice = classnames[k]
116             print('Class choice: ',classchoice)
117
118             trained = classifier.train(TV,classchoice,bands)
119             validated = VV.classify(trained)
120
121             confusion_matrix_validation = np.asarray(validated.errorMatrix(classchoice,'
122                 classification').getInfo())[1:,1:]
123             dataframes5.append(pd.DataFrame(confusion_matrix_validation/
124                 confusion_matrix_validation.sum(axis=1),columns=classnames,index=classnames))
125
126             print('Validation Data Results:')
127             print(dataframes5[j].round(3))
128             accuracies5.append(validated.errorMatrix(classchoice,'classification').accuracy().
129                 getInfo())
130             kappas5.append(validated.errorMatrix(classchoice,'classification').kappa().getInfo())
131
132             print('Overall accuracy: ',np.round(accuracies5[j],4),' (kappa: ',np.round(kappas5[j]
133                 ],4),')')

```

```

104     classchoice = classchoices[j]
105     print('Class choice: ', classchoice)
106     if classchoice == 'classes':
107         classnames = ['Water', 'Ice']
108     elif classchoice == 'subclasses':
109         classnames = ['Open Water', 'Rough Water', 'Sea Ice', 'Floating Land Ice']
110
111     trained = classifier.train(TV, classchoice, bands)
112     validated = VV.classify(trained)
113
114     confusion_matrix_validation = np.asarray(validated.errorMatrix(classchoice,
115         'classification').getInfo())[1:,1:]
116     dataframes5.append(pd.DataFrame(confusion_matrix_validation /
117         confusion_matrix_validation.sum(axis=1), columns=classnames, index=classnames))
118
119     print('Validation Data Results:')
120     print(dataframes5[j].round(3))
121     accuracies5.append(validated.errorMatrix(classchoice, 'classification').accuracy().
122         getInfo())
123     kappas5.append(validated.errorMatrix(classchoice, 'classification').kappa().getInfo())
124     print('Overall accuracy: ', np.round(accuracies5[j], 4), ' (kappa: ', np.round(kappas5[j],
125         4), ')')
126
127 ## (2.3) GLCM11
128 TV = TV11
129 VV = VV11
130 print('Window size: 11')
131 n_features_training = TV.size().getInfo()
132 n_features_validation = VV.size().getInfo()
133
134 classchoices = ['classes', 'subclasses']
135 bandchoices = [
136     ['HH_savg', 'HV_savg', 'HV_dvar']
137 ]
138 dataframes11 = []
139 accuracies11 = []
140 kappas11 = []
141
142 for i in range(len(bandchoices)):
143     bands = bandchoices[i]
144     print('Band choice: ', bands)
145
146     for j in range(len(classchoices)):
147         classchoice = classchoices[j]
148         print('Class choice: ', classchoice)
149         if classchoice == 'classes':
150             classnames = ['Water', 'Ice']
151         elif classchoice == 'subclasses':
152             classnames = ['Open Water', 'Rough Water', 'Sea Ice', 'Floating Land Ice']
153
154         trained = classifier.train(TV, classchoice, bands)
155         validated = VV.classify(trained)
156
157         confusion_matrix_validation = np.asarray(validated.errorMatrix(classchoice,
158             'classification').getInfo())[1:,1:]
159         dataframes11.append(pd.DataFrame(confusion_matrix_validation /
160             confusion_matrix_validation.sum(axis=1), columns=classnames, index=classnames))
161
162         print('Validation Data Results:')
163         print(dataframes11[j].round(3))
164         accuracies11.append(validated.errorMatrix(classchoice, 'classification').accuracy().
165             getInfo())
166         kappas11.append(validated.errorMatrix(classchoice, 'classification').kappa().getInfo())
167         print('Overall accuracy: ', np.round(accuracies11[j], 4), ' (kappa: ', np.round(kappas11[j],
168             4), ')')
169
170 ## (2.3) GLCM21
171 TV = TV21
172 VV = VV21

```

```
166 print('Window size: 21')
167
168 n_features_training = TV.size().getInfo()
169 n_features_validation = VV.size().getInfo()
170
171 classchoices = ['classes','subclasses']
172 bandchoices = [
173     'HH_savg', 'HV_savg', 'HV_dvar'
174 ]
175 dataframes21 = []
176 accuracies21 = []
177 kappas21 = []
178
179 for i in range(len(bandchoices)):
180     bands = bandchoices[i]
181     print('Band choice: ',bands)
182
183     for j in range(len(classchoices)):
184         classchoice = classchoices[j]
185         print('Class choice: ',classchoice)
186         if classchoice == 'classes':
187             classnames = ['Water','Ice']
188         elif classchoice == 'subclasses':
189             classnames = ['Open Water','Rough Water','Sea Ice','Floating Land Ice']
190
191         trained = classifier.train(TV,classchoice,bands)
192         validated = VV.classify(trained)
193
194         confusion_matrix_validation = np.asarray(validated.errorMatrix(classchoice,
195             'classification').getInfo())[1:,1:]
196         dataframes21.append(pd.DataFrame(confusion_matrix_validation/
197             confusion_matrix_validation.sum(axis=1),columns=classnames,index=classnames))
198
199         print('Validation Data Results:')
200         print(dataframes21[j].round(3))
201         accuracies21.append(validated.errorMatrix(classchoice,'classification').accuracy().
202             getInfo())
203         kappas21.append(validated.errorMatrix(classchoice,'classification').kappa().getInfo())
204
205     print('Overall accuracy: ',np.round(accuracies21[j],4),' (kappa: ',np.round(kappas21[
206         j],4),')')
207
208 %% (3) Visualization
209 ## (3.1) Main Classes Classification (Water-Ice)
210 plt.style.use("seaborn")
211 SMALL_SIZE = 40
212 MEDIUM_SIZE = 50
213 BIGGER_SIZE = 60
214
215 plt.rc('font', size=SMALL_SIZE)           # controls default text sizes
216 plt.rc('axes', titlesize=SMALL_SIZE)       # fontsize of the axes title
217 plt.rc('axes', labelsize=MEDIUM_SIZE)       # fontsize of the x and y labels
218 plt.rc('xtick', labelsize=SMALL_SIZE)       # fontsize of the tick labels
219 plt.rc('ytick', labelsize=SMALL_SIZE)       # fontsize of the tick labels
220 plt.rc('legend', fontsize=SMALL_SIZE)        # legend fontsize
221 plt.rc('figure', titlesize=BIGGER_SIZE)      # fontsize of the figure title
222
223 plt.figure(figsize=(40,10))
224 plt.subplot(1,4,1)
225 heat_map = sns.heatmap(
226     dataframes0[0].round(2),
227     linewidth = 2 ,
228     vmin = 0,
229     vmax = 1,
230     annot = True,
231     cmap = plt.cm.Blues,
232     cbar = False,
233     cbar_kws = {'shrink': 0.8},
234     square = True,
235     xticklabels = ['Water','Ice'],
236     yticklabels = ['Water','Ice'])
```

```

232 plt.yticks(rotation=0)
233 plt.title('BASE\nAccuracy = {} (kappa = {})'.format(np.round(accuracies0[0],2),np.round(
234     kappas0[0],2)))
235
236 plt.subplot(1,4,2)
237 heat_map = sns.heatmap(
238     dataframes5[0].round(2),
239     linewidth = 2 ,
240     vmin = 0,
241     vmax = 1,
242     annot = True,
243     cmap = plt.cm.Blues,
244     cbar = False,
245     cbar_kws = {'shrink': 0.8},
246     square = True,
247     xticklabels = ['Water','Ice'],
248     yticklabels = ['Water','Ice'])
249 plt.yticks(rotation=0)
250 plt.title('GLCM5\nAccuracy = {} (kappa = {})'.format(np.round(accuracies5[0],2),np.round(
251     kappas5[0],2)))
252
253 plt.subplot(1,4,3)
254 heat_map = sns.heatmap(
255     dataframes11[0].round(2),
256     linewidth = 2 ,
257     vmin = 0,
258     vmax = 1,
259     annot = True,
260     cmap = plt.cm.Blues,
261     cbar = False,
262     cbar_kws = {'shrink': 0.8},
263     square = True,
264     xticklabels = ['Water','Ice'],
265     yticklabels = ['Water','Ice'])
266 plt.yticks(rotation=0)
267 plt.title('GLCM11\nAccuracy = {} (kappa = {})'.format(np.round(accuracies11[0],2),np.round(
268     kappas11[0],2)))
269
270 plt.subplot(1,4,4)
271 heat_map = sns.heatmap(
272     dataframes21[0].round(2),
273     linewidth = 2 ,
274     vmin = 0,
275     vmax = 1,
276     annot = True,
277     cmap = plt.cm.Blues,
278     cbar_kws = {'shrink': 0.8},
279     square = False,
280     xticklabels = ['Water','Ice'],
281     yticklabels = ['Water','Ice'])
282 plt.yticks(rotation=0)
283 plt.title('GLCM21\nAccuracy = {} (kappa = {})'.format(np.round(accuracies21[0],2),np.round(
284     kappas21[0],2)))
285
286 plt.tight_layout()
287 plt.show()
288
289 ## (3.2) Sub-class Classification (OW-RW-SI-FLI)
290 plt.style.use("seaborn")
291 SMALL_SIZE = 25
292 MEDIUM_SIZE = 30
293 BIGGER_SIZE = 35
294
295 plt.rc('font', size=SMALL_SIZE)           # controls default text sizes
296 plt.rc('axes', titlesize=SMALL_SIZE)       # fontsize of the axes title
297 plt.rc('axes', labelsize=MEDIUM_SIZE)      # fontsize of the x and y labels
298 plt.rc('xtick', labelsize=SMALL_SIZE)       # fontsize of the tick labels
299 plt.rc('ytick', labelsize=SMALL_SIZE)       # fontsize of the tick labels
300 plt.rc('legend', fontsize=SMALL_SIZE)        # legend fontsize
301 plt.rc('figure', titlesize=BIGGER_SIZE)      # fontsize of the figure title

```

```
299 plt.figure(figsize=(20,20))
300 plt.subplot(2,2,1)
301 heat_map = sns.heatmap(
302     dataframes0[1].round(2),
303     linewidth = 2 ,
304     vmin = 0,
305     vmax = 1,
306     annot = True,
307     cmap = plt.cm.Blues,
308     cbar_kws = {'shrink': 0.8},
309     square = True,
310     xticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'],
311     yticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'])
312 plt.yticks(rotation=0)
313 plt.title('BASE\nnAccuracy = {} (kappa = {})'.format(np.round(accuracies0[1],2),np.round(
314     kappas0[1],2)))
315
316 plt.subplot(2,2,2)
317 heat_map = sns.heatmap(
318     dataframes5[1].round(2),
319     linewidth = 2 ,
320     vmin = 0,
321     vmax = 1,
322     annot = True,
323     cmap = plt.cm.Blues,
324     cbar_kws = {'shrink': 0.8},
325     square = True,
326     xticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'],
327     yticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'])
328 plt.yticks(rotation=0)
329 plt.title('GLCM5\nnAccuracy = {} (kappa = {})'.format(np.round(accuracies5[1],2),np.round(
330     kappas5[1],2)))
331
332 plt.subplot(2,2,3)
333 heat_map = sns.heatmap(
334     dataframes11[1].round(2),
335     linewidth = 2 ,
336     vmin = 0,
337     vmax = 1,
338     annot = True,
339     cmap = plt.cm.Blues,
340     cbar_kws = {'shrink': 0.8},
341     square = True,
342     xticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'],
343     yticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'])
344 plt.yticks(rotation=0)
345 plt.title('GLCM11\nnAccuracy = {} (kappa = {})'.format(np.round(accuracies11[1],2),np.round(
346     kappas11[1],2)))
347
348 plt.subplot(2,2,4)
349 heat_map = sns.heatmap(
350     dataframes21[1].round(2),
351     linewidth = 2 ,
352     vmin = 0,
353     vmax = 1,
354     annot = True,
355     cmap = plt.cm.Blues,
356     cbar_kws = {'shrink': 0.8},
357     square = True,
358     xticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'],
359     yticklabels = ['Open\nnWater','Rough\nnWater','Sea\nnIce','Floating\nnLand Ice'])
360 plt.yticks(rotation=0)
361 plt.title('GLCM21\nnAccuracy = {} (kappa = {})'.format(np.round(accuracies21[1],2),np.round(
362     kappas21[1],2)))
363
364 plt.tight_layout()
365 plt.show()
```

B.9. Export of Classification & Post-Processing Results

9_Visualizations

```

1 #%% (1) Code set-up
2 ## (1.1) Import packages
3 import ee
4 import geemap
5 import numpy as np
6 import os
7
8 ## (1.2) Initialize
9 ee.Initialize()
10
11 ## (1.3) Parameterization & Data Import
12 DEM = ee.Image('CPOM/CryoSat2/ANTARCTICA_DEM').select('elevation')
13 IceShelves = ee.FeatureCollection("users/sophiederoda/IceShelf_Antarctica")
14 ROIs = ee.FeatureCollection('users/skjeltnmaps/ISCLSegments20km')
15
16 AmundsenEmbayment = ee.Geometry.Polygon(
17     coords = [[[[-108.406159, -75.468542], [-108.406159, -71.238725], [-96.219667,
18         -71.238725], [-96.219667, -75.468542], [-108.406159, -75.468542]]],
19     geodesic = False,
20     proj = 'EPSG:4326')
21 CarneyCoast = ee.Geometry.Polygon(
22     coords = [[[[-146.815157, -74.904569], [-142.211175, -76.343611], [-128.293796,
23         -74.877073], [-120.513741, -75.203774], [-120.513741, -73.617891], [-127.520609,
24         -72.722358], [-146.815157, -74.904569]]]],
25     geodesic = False,
26     proj = 'EPSG:4326')
27 PineIslandGlacier = ee.Geometry({
28     'geodesic': False,
29     'type': 'Polygon',
30     'coordinates': [[[[-105.449683, -75.564228], [-105.449683, -73.068832], [-98.639549,
31         -73.068832], [-98.639549, -75.564228], [-105.449683, -75.564228]]]
32 })
33
34 trainValArea = AmundsenEmbayment.union(CarneyCoast,maxError=1e3)
35
36 ## (1.3) Parameterization
37 start = '2014-01-01'
38 end = '2022-04-01'
39 crs = 'EPSG:3031'
40 S1band = 'HV'
41 dt = 4
42 elevation_threshold = 200
43
44 ReferenceData = ee.FeatureCollection('users/skjeltnmaps/RD_'+S1band+'_'+str(dt)+'hrs_'+start
45     [:4]+start[5:7]+start[8:10]+'_'+end[:4]+end[5:7]+end[8:10])
46 S1OptMetadata = ee.FeatureCollection('users/skjeltnmaps/S1Opt_'+S1band+'_'+str(dt)+'
```

```

62 L8name = 'L8'
63 L8bands = ['B4','B3','B2']
64
65 S2name = 'S2'
66 S2bands = ['B4', 'B3', 'B2']
67
68 RGBbands = ['Red','Green','Blue']
69 RGB_params = {
70   'bands': RGBbands,
71   'min': 3000,
72   'max': 15000}
73
74 TV11 = ee.FeatureCollection('users/skjeltemp/TrainingValues_w11_1000pixels') .map(lambda
75   feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
76   HV_contras')))
75 TV5 = ee.FeatureCollection('users/skjeltemp/TrainingValues_w5_1000pixels') .map(lambda feat
76   : feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('HV_contras'
77   )))
76 TV21 = ee.FeatureCollection('users/skjeltemp/TrainingValues_w21_1000pixels') .map(lambda
77   feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
78   HV_contras')))
77
78 VV11 = ee.FeatureCollection('users/skjeltemp/ValidationValues_w11_1000pixels') .map(lambda
78   feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
79   HV_contras')))
79 VV5 = ee.FeatureCollection('users/skjeltemp/ValidationValues_w5_1000pixels') .map(lambda
79   feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
80   HV_contras'))))
80 VV21 = ee.FeatureCollection('users/skjeltemp/ValidationValues_w21_1000pixels') .map(lambda
80   feat: feat.set('HH_contrast',feat.get('HH_contras')).set('HV_contrast',feat.get('
81   HV_contras'))))
81
82 ## Edit visualization export folder is necessary
83 # html_dir = os.path.join(os.path.expanduser('~'), 'Downloads')
84 # if not os.path.exists(html_dir):
85 #     os.makedirs(html_dir)
86
87 #%% (2) Reconstruct Match Data Set
88 #%% (2.1) General Functions
89 def utcToLocal(image):
90     centroid = ee.Image(image).geometry(maxError=1e3).centroid(maxError=1e3)
91     longitude = centroid.coordinates().get(0)
92     localTime = ee.Image(image).date().advance(ee.Number(longitude).divide(15).ceil()).
92       subtract(1,'hour')
93     localMillis = ee.Date(localTime).millis()
94
95     return image
96     .set('utctime',ee.Image(image).get('system:time_start'))
97     .set('localtime',localMillis)
98
99 def addID(image,name):
100     id = ee.String(name+'_').cat(ee.String(ee.Date(ee.Image(image).get('localtime')).format(
100       'YYYY_MM_dd_KK:mm')))
101     return image.set('id',id).set('data',name)
102
103 def addRGBbands(image,bandnames):
104     renamed = image.select(bandnames,RGBbands)
105     rgb = image.addBands(renamed).select(RGBbands)
106     return rgb
107
108 def addWB(image):
109     wb = ee.List(ee.List(image.geometry(1e4).bounds(1e4).coordinates().get(0)).get(0)).
109       get(0)
110     return image.set('WB',wb)
111
112 def addMetadata(image,utcname,metadata):
113     utc = ee.String(image.get('system:time_start'))
114     filtered = metadata.filter(ee.Filter.inList(utcname,[utc]))
115     name = filtered.aggregate_array('NAME').distinct()
116     segment = name.map(lambda name: filtered.filterMetadata('NAME','equals',name).
116       aggregate_array('SEG'))

```

```

117     wb = ee.List(ee.List(image.geometry(1e4).bounds(1e4).coordinates().get(0)).get(0))
118     get(0)
119     return image.set('NAME',name).set('SEG',segment).set('WB',wb)
120
121 def addMatchIndex(image,metadata):
122     utc = image.get('system:time_start')
123     match = metadata.filterMetadata('utc','equals',utc).first()
124     matchIndex = ee.Number(match.get('match'))
125     return image.set('match',matchIndex)
126
127 def angleNormalization(image):
128     def degreesToRadians(object):
129         return object.multiply(np.pi).divide(180)
130
131     thetaRef = degreesToRadians(ee.Number(30))
132     radians = degreesToRadians(image.select('angle'))
133
134     angleCorr = image.select('angle').divide(thetaRef.cos()).pow(ee.Number(2)).multiply(
135         radians.cos().pow(ee.Number(2)))
136     HH = image.select('HH').divide(thetaRef.cos()).pow(ee.Number(2)).multiply(radians.cos().pow(ee.Number(2)))
137     HV = image.select('HV').divide(thetaRef.cos()).pow(ee.Number(2)).multiply(radians.cos().pow(ee.Number(2)))
138
139     normalized = image.addBands(HH,overwrite=True).addBands(HV,overwrite=True).addBands(
140         angleCorr)
141
142     return normalized
143
144 def clipToMatchingImage(image,matchData):
145     matchIndex = ee.Number(image.get('match'))
146     matchCol = matchData.filterMetadata('match','equals',matchIndex)
147     matchList = matchCol.toList(matchCol.size())
148     GeomList = matchList.map(lambda image: ee.Image(image).geometry())
149
150
151     def combineGeometries(geom2,geom1):
152         geom = ee.Geometry(geom1).union(ee.Geometry(geom2),maxError=1e3)
153         return geom
154
155     matchGeometry = ee.Geometry(GeomList.iterate(
156         function = combineGeometries,
157         first = ee.Geometry(GeomList.get(0))
158     ))
159     clipped = image.clip(matchGeometry)
160
161     return clipped
162
163 ## (2.2) Reconstruction
164 S1 = ee.ImageCollection("COPERNICUS/S1_GRD")
165     .filter(ee.Filter.inList('system:time_start',S1_UTC))
166     .filter(ee.Filter.listContains('transmitterReceiverPolarisation',S1band))
167     .map(utcToLocal)
168     .map(angleNormalization)
169     .map(lambda image: image.updateMask(DEM.unmask().gt(elevation_threshold).Not()))
170     .map(lambda image: addID(image,S1name))
171     .map(lambda image: addMetadata(image,'S1_UTC',S1OptMetadata))
172     .map(lambda image: addMatchIndex(image,ReferenceData))
173     .sort('WB')
174
175 L8 = ee.ImageCollection("LANDSAT/LC08/C01/T2")
176     .filter(ee.Filter.inList('system:time_start',Opt_UTC))
177     .map(lambda image: addRGBbands(image,L8bands))
178     .map(utcToLocal)
179     .map(lambda image: addID(image,L8name))
180     .map(lambda image: addMatchIndex(image,ReferenceData))
181     .map(lambda image: addMetadata(image,'Opt_UTC',S1OptMetadata))
182 S2 = ee.ImageCollection("COPERNICUS/S2")
183     .filter(ee.Filter.inList('system:time_start',Opt_UTC))
184     .map(lambda image: addRGBbands(image,S2bands))
185     .map(utcToLocal)
186     .map(lambda image: addID(image,S2name))

```

```

183     .map(lambda image: addMatchIndex(image, ReferenceData))
184     .map(lambda image: addMetadata(image, 'Opt_UTC', S1OptMetadata))
185 Opt = L8.merge(S2).sort('WB')
186 OptList = Opt.toList(Opt.size())
187
188 #%% (3) Create Training & Validation Data Set
189 ## (3.1) Match functions
190 def matchDataSets(S1, Opt):
191     def addMatchesAsBands(imgS1, Opt):
192         matchIndex = ee.Number(imgS1.get('match'))
193         OptCol = Opt.filterMetadata('match', 'equals', matchIndex)
194         OptList = OptCol.toList(OptCol.size())
195         GeomList = OptList.map(lambda image: ee.Image(image).geometry())
196
197         def combineGeometries(geom2, geom1):
198             geom = ee.Geometry(geom1).union(ee.Geometry(geom2), maxError=1e3)
199             return geom
200
201     OptGeometry = ee.Geometry(GeomList.iterate(
202         function = combineGeometries,
203         first = ee.Geometry(GeomList.get(0)))
204     ))
205     S1Geometry = imgS1.geometry()
206
207     OptImage = OptCol.median().clip(S1Geometry)
208     S1Image = imgS1.clip(OptGeometry)
209     matchAdded = S1Image.addBands(OptImage)
210
211     return matchAdded
212
213     matchesOfInterest = Opt.aggregate_array('match').distinct().sort()
214     S1Filtered = S1.filter(ee.Filter.inList('match', matchesOfInterest))
215     S1Opt = S1Filtered.map(lambda imgS1: addMatchesAsBands(imgS1, Opt))
216
217     return S1Opt
218
219 ## (3.2) Create Reference Data Set
220 Opt_Amundsen = Opt.filterBounds(AmundsenEmbayment)
221 Opt_Carney = Opt.filterBounds(CarneyCoast)
222
223 AmundsenMatches = Opt_Amundsen.aggregate_array('match').distinct().sort()
224 CarneyMatches = Opt_Carney.aggregate_array('match').distinct().sort()
225
226 S1_Amundsen = S1.filter(ee.Filter.inList('match', AmundsenMatches))
227 Opt_Amundsen = Opt.filter(ee.Filter.inList('match', AmundsenMatches))
228 S1Opt_Amundsen = matchDataSets(S1_Amundsen, Opt_Amundsen)
229
230 S1_Carney = S1.filter(ee.Filter.inList('match', CarneyMatches))
231 Opt_Carney = Opt.filter(ee.Filter.inList('match', CarneyMatches))
232 S1Opt_Carney = matchDataSets(S1_Carney, Opt_Carney)
233
234 trainIDsAmundsen = ee.List([1,2,5,6,7,8])
235 trainIDsCarney = ee.List([1,4,11])
236
237 S1Training = ee.ImageCollection(trainIDsAmundsen.map(lambda index: ee.Image(S1_Amundsen.
238     toList(S1_Amundsen.size()).get(index)))
239     .merge(ee.ImageCollection(trainIDsCarney.map(lambda index: ee.Image(S1_Carney.toList(
240         S1_Carney.size()).get(index)))))
241     OptTraining = Opt.filter(ee.Filter.inList('match', S1Training.aggregate_array('match').
242         distinct().sort()))
243     TrainingData = ee.ImageCollection(trainIDsAmundsen.map(lambda index: ee.Image(S1Opt_Amundsen.
244         toList(S1Opt_Amundsen.size()).get(index)))
245         .merge(ee.ImageCollection(trainIDsCarney.map(lambda index: ee.Image(S1Opt_Carney.toList(
246             S1Opt_Carney.size()).get(index)))))
247
248 valIDsAmundsen = ee.List([0,3,4])
249 valIDsCarney = ee.List([10])
250
251 S1Validation = ee.ImageCollection(valIDsAmundsen.map(lambda index: ee.Image(S1_Amundsen.
252     toList(S1_Amundsen.size()).get(index)))
253     .merge(ee.ImageCollection(valIDsCarney.map(lambda index: ee.Image(S1_Carney.toList(

```

```

        S1_Carney.size()).get(index)))))

248 OptValidation = Opt.filter(ee.Filter.inList('match',S1Validation.aggregate_array('match')).
    distinct().sort())
249 ValidationData = ee.ImageCollection(valIDsAmundsen.map(lambda index: ee.Image(S1Opt_Amundsen.
    toList(S1Opt_Amundsen.size()).get(index))))
250     .merge(ee.ImageCollection(valIDsCarney.map(lambda index: ee.Image(S1Opt_Carney.toList(
        S1Opt_Carney.size()).get(index)))))

251
252 ## (2.3) Store Match Metadata
253 opttrnnames = OptTraining.aggregate_array('system:index')
254 sltrnnames = S1Training.aggregate_array('system:index')
255 optvalnames = OptValidation.aggregate_array('system:index')
256 s1valnames = S1Validation.aggregate_array('system:index')

257
258 # print(sltrnnames.getInfo())
259 # print(s1valnames.getInfo())
260 # print(opttrnnames.getInfo())
261 # print(optvalnames.getInfo())

262
263 opttrnUTCs = OptTraining.aggregate_array('localtime').map(lambda utc: ee.Date(utc).format(
    'YYYY/MM/dd HH:mm:ss'))
264 sltrnUTCs = S1Training.aggregate_array('localtime').map(lambda utc: ee.Date(utc).format('YYYY
    /MM/dd HH:mm:ss'))
265 optvalUTCs = OptValidation.aggregate_array('localtime').map(lambda utc: ee.Date(utc).format(
    'YYYY/MM/dd HH:mm:ss'))
266 s1valUTCs = S1Validation.aggregate_array('localtime').map(lambda utc: ee.Date(utc).format(
    'YYYY/MM/dd HH:mm:ss'))

267
268 # print(sltrnUTCs.getInfo())
269 # print(opttrnUTCs.getInfo())
270
271 # print(s1valUTCs.getInfo())
272 # print(optvalUTCs.getInfo())
273

274 #%% (3) GLCM Features
275 ## (3.1) Computation
276 windows = [5,11,21]
277
278 for i in range(len(windows)):
279     w = windows[i]
280     training_data = eval('TV'+str(w))

281     features = ValidationData.select(['HH','HV'])
282         .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size=w,
283             kernel=None)))
284         .map(lambda image: image.addBands(image.select('HV').toInt32().glcmTexture(size=w,
285             kernel=None)))
286         .select(['HH','HV','HH_savg','HV_savg','HV_dvar'])

287 ## (3.2) Export
288 ## Leave commented!!
289 ## Export was necessary to store image collections
290
291 # geemap.ee_export_image_collection_to_drive(
292 #     features,
293 #     descriptions=['Val1Features_w'+str(w),'Val2Features_w'+str(w),'Val3Features_w'+str(w),''
294 #         Val4Features_w'+str(w)],
295 #     folder='GLCM',
296 #     scale=40,
297 #     crs=crs
298 # )

299 #%% (4) Classification
300 ## (4.1) Computation
301 classifier = ee.Classifier.smileRandomForest(
302     numberOfTrees = 10,
303     variablesPerSplit = 2,
304     maxNodes = None,
305     minLeafPopulation = 1,
306     seed = 1)
307 classchoice = 'classes'
```

```

308
309 BASE = ValidationData
310     .select(['HH','HV'])
311     .map(lambda image: image.classify(classifier.train(TV5,classchoice,ee.List(['HH','HV']))))
312     .copyProperties(image)
313     .map(lambda image: image.rename('BASE'))
314
315 GLCM5 = ValidationData
316     .select(['HH','HV'])
317     .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size=5,kernel=
318         None)))
319     .map(lambda image: image.addBands(image.select('HV').toInt32().glcmTexture(size=5,kernel=
320         None)))
321     .select(['HH_savg','HV_savg','HV_dvar'])
322     .map(lambda image: image.classify(classifier.train(TV5,classchoice,ee.List(['HH_savg',
323         'HV_savg','HV_dvar']))))
324     .copyProperties(image)
325     .map(lambda image: image.rename('GLCM5'))
326
327 GLCM11 = ValidationData
328     .select(['HH','HV'])
329     .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size=11,kernel
330         =None)))
331     .map(lambda image: image.addBands(image.select('HV').toInt32().glcmTexture(size=11,kernel
332         =None)))
333     .select(['HH_savg','HV_savg','HV_dvar'])
334     .map(lambda image: image.classify(classifier.train(TV11,classchoice,ee.List(['HH_savg',
335         'HV_savg','HV_dvar']))))
336     .copyProperties(image)
337     .map(lambda image: image.rename('GLCM11'))
338
339 GLCM21 = ValidationData
340     .select(['HH','HV'])
341     .map(lambda image: image.addBands(image.select('HH').toInt32().glcmTexture(size=21,kernel
342         =None)))
343     .map(lambda image: image.addBands(image.select('HV').toInt32().glcmTexture(size=21,kernel
344         =None)))
345     .select(['HH_savg','HV_savg','HV_dvar'])
346     .map(lambda image: image.classify(classifier.train(TV21,classchoice,ee.List(['HH_savg',
347         'HV_savg','HV_dvar']))))
348     .copyProperties(image)
349     .map(lambda image: image.rename('GLCM21'))
350
351 ## (4.2) Export
352 ## Leave commented!!
353 ## Export was necessary to create feature collections used in section below
354
355 # geemap.ee_export_image_collection_to_drive(
356 #     BASE,
357 #     descriptions=[
358 #         'Val1_BASE',
359 #         'Val2_BASE',
360 #         'Val3_BASE',
361 #         'Val4_BASE'],
362 #     folder='Classification',
363 #     scale=40,
364 #     crs=crs
365 # )
366
367 # geemap.ee_export_image_collection_to_drive(
368 #     GLCM5,
369 #     descriptions=[
370 #         'Val1_GLCM5',
371 #         'Val2_GLCM5',
372 #         'Val3_GLCM5',
373 #         'Val4_GLCM5'],
374 #     folder='Classification',
375 #     scale=40,
376 #     crs=crs
377 # )
378

```

```

369 # geemap.ee_export_image_collection_to_drive(
370 #     GLCM11,
371 #     descriptions=[ 
372 #         'Val1_GLCM11',
373 #         'Val2_GLCM11',
374 #         'Val3_GLCM11',
375 #         'Val4_GLCM11'],
376 #     folder='Classification',
377 #     scale=40,
378 #     crs=crs
379 # )
380
381 # geemap.ee_export_image_collection_to_drive(
382 #     GLCM21,
383 #     descriptions=[ 
384 #         'Val1_GLCM21',
385 #         'Val2_GLCM21',
386 #         'Val3_GLCM21',
387 #         'Val4_GLCM21'],
388 #     folder='Classification',
389 #     scale=40,
390 #     crs=crs
391 # )
392
393 #%% (5) Post-Processing: Area Filter
394 ## (5.1) Connected Neighbours Computation
395 neighbours5 = GLCM5.map(lambda image: image.updateMask(image.eq(1)).connectedPixelCount(1024,
    False))
396 neighbours11 = GLCM11.map(lambda image: image.updateMask(image.eq(1)).connectedPixelCount
    (1024,False))
397 neighbours21 = GLCM21.map(lambda image: image.updateMask(image.eq(1)).connectedPixelCount
    (1024,False))
398
399 ## (5.2) Export
400 ## Leave commented!!
401 ## Export was necessary to create feature collections used in section below
402
403 # geemap.ee_export_image_collection_to_drive(
404 #     neighbours5,
405 #     descriptions=[ 
406 #         'Val1_NN5',
407 #         'Val2_NN5',
408 #         'Val3_NN5',
409 #         'Val4_NN5'],
410 #     folder='ConnectedNeighbors',
411 #     scale=40,
412 #     crs=crs
413 # )
414
415 # geemap.ee_export_image_collection_to_drive(
416 #     neighbours11,
417 #     descriptions=[ 
418 #         'Val1_NN11',
419 #         'Val2_NN11',
420 #         'Val3_NN11',
421 #         'Val4_NN11'],
422 #     folder='ConnectedNeighbors',
423 #     scale=40,
424 #     crs=crs
425 # )
426
427 # geemap.ee_export_image_collection_to_drive(
428 #     neighbours21,
429 #     descriptions=[ 
430 #         'Val1_NN21',
431 #         'Val2_NN21',
432 #         'Val3_NN21',
433 #         'Val4_NN21'],
434 #     folder='ConnectedNeighbors',
435 #     scale=40,
436 #     crs=crs

```

```
437 # )
438
439 #%% (6) Export Optical Images
440 # geemap.ee_export_image_collection_to_drive(
441 #     OptValidation.select(RGBbands),
442 #     descriptions=['ValOptFull_1','ValOptFull_2','ValOptFull_3','ValOptFull_4'],
443 #     folder='export',
444 #     scale=Opt.first().select(0).projection().nominalScale(),
445 #     crs=crs
446 # )
```

Bibliography

- Alley, K. E., Scambos, T. A., Alley, R. B., and Holschuh, N. (2019). Troughs developed in ice-stream shear margins precondition ice shelves for ocean-driven breakup. Added in review.
- Alley, K. E., Scambos, T. A., Siegfried, M. R., and Fricker, H. A. (2016). Impacts of warm water on antarctic ice shelf stability through basal channel formation. *Nature Geoscience* 2016 9:4, 9:290–293. Added in review.
- Barber, D. G. and Massom, R. A. (2007). Chapter 1 the role of sea ice in arctic and antarctic polynyas.
- Bindschadler, R., Vaughan, D. G., and Vornberger, P. (2011). Variability of basal melt beneath the pine island glacier ice shelf, west antarctica. *Journal of Glaciology*, 57:581–595. Added in review.
- Brocq, A. M. L., Ross, N., Griggs, J. A., Bingham, R. G., Corr, H. F. J., Ferraccioli, F., Jenkins, A., Jordan, T. A., Payne, A. J., Rippin, D. M., and Siegert, M. J. (2013). Evidence from ice shelves for channelized meltwater flow beneath the antarctic ice sheet. *Nature Geoscience* 2013 6:11, 6:945–948. Added in review.
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46.
- Conners, R. W., Trivedi, M. M., and Harlow, C. A. (1984). Segmentation of a high-resolution urban scene using texture operators. *Computer Vision, Graphics, and Image Processing*, 25:273–310.
- de Roda Husman, S., van der Sanden, J. J., Lhermitte, S., and Eleveld, M. A. (2021). Integrating intensity and context for improved supervised river ice classification from dual-pol sentinel-1 sar data. *International Journal of Applied Earth Observation and Geoinformation*, 101:102359.
- Dierking, W. (2010). Mapping of different sea ice regimes using images from sentinel-1 and alos synthetic aperture radar. *IEEE Transactions on Geoscience and Remote Sensing*, 48:1045–1058. Added in review.
- Dierking, W. (2013). Sea ice monitoring by synthetic aperture radar. *Oceanography*, 26. Added in review.
- Dutrieux, P., Stewart, C., Jenkins, A., Nicholls, K. W., Corr, H. F. J., Rignot, E., and Steffen, K. (2014). Basal terraces on melting ice shelves. *Geophysical Research Letters*, 41:5506–5513. Added in review.
- ESA (2012). *Sentinel-1 : ESA's radar observatory mission for GMES operational services*. ESA Communications.
- Freeman, A. and Durden, S. L. (1998). A three-component scattering model for polarimetric sar data. *IEEE Transactions on Geoscience and Remote Sensing*, 36:963–973.
- Gladish, C. V., Holland, D. M., Holland, P. R., and Price, S. F. (2012). Ice-shelf basal channels in a coupled ice/ocean model. *Journal of Glaciology*, 58:1227–1244.
- Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., and Moore, R. (2017). Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 202:18–27. Added in review.
- Gourmelen, N., Goldberg, D. N., Snow, K., Henley, S. F., Bingham, R. G., Kimura, S., Hogg, A. E., Shepherd, A., Mouginot, J., Lenaerts, J. T. M., Ligtenberg, S. R. M., and van de Berg, W. J. (2017). Channelized melting drives thinning under a rapidly melting antarctic ice shelf. *Geophysical Research Letters*, 44:9796–9804. Added in review.

- Haralick, R. M., Dinstein, I., and Shanmugam, K. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-3:610–621.
- Hellmer, H. H., Kauker, F., Timmermann, R., Determann, J., and Rae, J. (2012). Twenty-first-century warming of a large antarctic ice-shelf cavity by a redirected coastal current. Added in review.
- Hoekstra, M., Jiang, M., Clausi, D. A., and Duguay, C. (2020). Lake ice-water classification of radarsat-2 images by integrating irgs segmentation with pixel-based random forest labeling. *Remote Sensing*, 12:1425.
- Hollands, T. and Dierking, W. (2016). Dynamics of the terra nova bay polynya: The potential of multi-sensor satellite observations. *Remote Sensing of Environment*, 187:30–48. Added in review.
- IPCC (2014). *Climate change 2014: Synthesis Report*.
- Jourdain, N. C., Mathiot, P., Merino, N., Durand, G., Sommer, J. L., Spence, P., Dutrieux, P., and Madec, G. (2017). Ocean circulation and sea-ice thinning induced by melting ice shelves in the amundsen sea. *Journal of Geophysical Research: Oceans*, 122:2550–2573.
- Karvonen, J. and Hallikainen, M. (2009). Sea ice sar classification based on edge features. *International Geoscience and Remote Sensing Symposium (IGARSS)*, 3.
- Karvonen, J., Similä, M., and Mäkinen, M. (2005). Open water detection from baltic sea ice radarsat-1 sar imagery. *IEEE Geoscience and Remote Sensing Letters*, 2:275–279.
- Lazeroms, W. M., Jenkins, A., Gudmundsson, G. H., and Wal, R. S. V. D. (2018). Modelling present-day basal melt rates for antarctic ice shelves using a parametrization of buoyant meltwater plumes. *Cryosphere*, 12:49–70. Added to review.
- Lhermitte, S., Sun, S., Shuman, C., Wouters, B., Pattyn, F., Wuite, J., Berthier, E., and Nagler, T. (2020). Damage accelerates ice shelf instability and mass loss in amundsen sea embayment. *Proceedings of the National Academy of Sciences of the United States of America*, 117:24735–24741.
- Lohse, J., Doulgeris, A. P., and Dierking, W. (2021). Incident angle dependence of sentinel-1 texture features for sea ice classification. *Remote Sensing 2021, Vol. 13, Page 552*, 13:552.
- Long, D. G., Collyer, R. S., Reed, R., and Arnold, D. V. (1996). Dependence of the normalized radar cross section of water waves on bragg wavelength-wind speed sensitivity. *IEEE Transactions on Geoscience and Remote Sensing*, 34:656–666.
- Mankoff, K. D., Jacobs, S. S., Tulaczyk, S. M., and Stammerjohn, S. E. (2012). The role of pine island glacier ice shelf basal channels in deep-water upwelling, polynyas and ocean circulation in pine island bay, antarctica. *Annals of Glaciology*, 53:123–128. Added in review.
- Maqueda, M. A. M., Willmott, A. J., and Biggs, N. R. T. (2004). Polynya dynamics: a review of observations and modeling. *Reviews of Geophysics*, 42. Added in review.
- Mermoz, S., Allain, S., Bernier, M., Pottier, E., and Gherboudj, I. (2009). Classification of river ice using polarimetric sar data. *Canadian Journal of Remote Sensing*, 35:460–473.
- Pattyn, F. and Morlighem, M. (2020). The uncertain future of the antarctic ice sheet. *Science*, 367:1331–1335.
- Rignot, E. and Steffen, K. (2008). Channelized bottom melting and stability of floating ice shelves. *Geophysical Research Letters*, 35.
- Sergienko, O. V. (2013). Basal channels on ice shelves. *Journal of Geophysical Research: Earth Surface*, 118:1342–1355. Added in review.
- Shean, D. E., Joughin, I. R., Dutrieux, P., Smith, B. E., and Berthier, E. (2019). Ice shelf basal melt rates from a high-resolution digital elevation model (dem) record for pine island glacier, antarctica. *Cryosphere*, 13:2633–2656. Added in review.

- Shelestov, A., Lavreniuk, M., Kussul, N., Novikov, A., and Skakun, S. (2017). Exploring google earth engine platform for big data processing: Classification of multi-temporal satellite imagery for crop mapping. *Frontiers in Earth Science*, 0:17. Added in review.
- Stewart, C. L., Christoffersen, P., Nicholls, K. W., Williams, M. J. M., and Dowdeswell, J. A. (2019). Basal melting of ross ice shelf from solar heat absorption in an ice-front polynya. *Nature Geoscience* 2019 12:6, 12:435–440.
- Topouzelis, K. and Singha, S. (2016). Incidence angle normalization of wide swath sar data for oceanographic applications. *Open Geosciences*, 8:450–464.
- Williams, W. J., Carmack, E. C., and Ingram, R. G. (2007). Chapter 2 physical oceanography of polynyas.
- Xu, L. and Li, J. (2015). Mapping sea ice from satellite sar imagery. pages 113–135. Added in review.