The Cooper Union for the Advancement of Science and Art

Albert Nerken School of Engineering

Improving Semantic Water Segmentation by Fusing Sentinel-1 Intensity and Interferometric Synthetic Aperture Radar (InSAR) Coherence Data

By

Ernesto Colon

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Engineering

Advisor

Prof. Sam Keene

The Cooper Union for the Advancement of Science and Art

Albert Nerken School of Engineering

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

[illegible] 4.22.2022

Barry L. Shoop, Ph.D., P.E. - Date

Dean, Albert Nerken School of Engineering

[illegible] 4.22.2022

Prof. Sam Keene, Ph.D. - Date

Candidate's Thesis Advisor

Acknowledgment

I would first like to thank my advisor, Sam Keene, for his guidance, insights, and support as I pursued my graduate studies.

I would like to thank my family for their endless love and support in all my endeavors regardless of how much I wander.

I would like to thank Jade Da Silva for her zealous support and encouragement.

I would like to thank Krishna Karra for introducing me to the discipline of Earth Observation and sparking an interest that would ultimately become a passion. I would also like to thank Yuval Ofek for his help as we both navigated remote sensing research.

I would like to thank my professors and mentors for their influence in the person I am today.

I would like to thank Beth Tellman, Rohit Mukherjee, Hannah Friedrich, and the extended University of Arizona, Columbia University and THP teams for their partnership, guidance, and inspiration.

I would like to thank Batu Osmanoglu, MinJeong Jo, Elodie Macorps, and Rustem Albayrak from NASA Goddard Space Flight Center for their generosity and guidance as I researched the world of Synthetic Aperture Radar.

I would like to thank all my friends and peers for constantly helping me grow as a person and as a professional.

Abstract

Several studies have demonstrated the advantages of using Interferometric Synthetic Aperture Radar (InSAR) coherence and SAR backscatter intensity data for change detection and flood mapping. Most of these works, however, are limited to a few case studies or use high resolution SAR data not freely available to the public. The purpose of this study was to determine the effectiveness of fusing 10-meter resolution Sentinel-1 intensity and InSAR coherence data across geographically diverse regions for semantic water segmentation. We fused Sentinel-1 intensity and InSAR coherence as inputs to uni- and bi-temporal classification models cross-trained using optically derived Sentinel-2 water masks. We trained Attention U-Net convolutional neural network models and XGBoost pixel-wise classifiers to assess the relative improvements gained by adding the coherence data to a bi-temporal model relative to a uni-temporal intensity-only model. We found that the bi-temporal intensity and coherence fusion models improve the water intersection over union by over 3% when aggregated over all geographical regions studied. We also found that the bi-temporal intensity and coherence fusion models improve the water-class recall by over 4%, systematically reducing the false negative rate across all the geographic regions studied. The reduction in the water-class false negative rate comes at the expense of the false positive rate, however. We found that the uni-temporal intensity only models outperform the bi-temporal intensity and coherence fusion models by 1 to 2% in terms of water-class precision. With a 6-day repeat cycle, the Sentinel-1 platform democratizes high resolution SAR data access that is complementary to its optical counterparts. This is especially important as we continue to develop, improve, and operationalize methods to manage and respond to natural disasters such as extreme flooding events.

Contents

List of Figures

List of Tables

1 Introduction

Natural disasters constantly pose risks to human livelihood, our environment and infrastructure, and have significant socioeconomic impacts. It is imperative that we develop robust means to respond to natural disasters, and most importantly, to respond in a timely manner. We cannot prevent natural disasters from occurring, but it is within our power to take actions in order to prepare and adapt to mitigate damages associated with extreme events.

Earth Observation (EO) satellites have become an integral part of natural disaster emergency management and response. Satellites provide a unique perspective prior, during, and after a natural disaster occurs that we cannot get with in situ observations. The list of applications of EO satellites for natural disaster management is very extensive. Some of these applications include monitoring climate change, wildfires, volcanic, seismic and hurricane activity [2].

The latest Intergovernmental Panel on Climate Change (IPCC) state of the climate report projects that the frequency of extreme weather events is in an upward trajectory. There is strong evidence that human-induced climate change is one of the main driving forces behind this trend. The IPCC report estimates that "50 to 75% of the global population could be exposed to periods of life-threatening climatic conditions due to extreme heat and humidity by 2100." The report also projects that climate change "will increasingly put pressure on food production and access, especially in vulnerable regions, undermining food security and nutrition" [3] [4].

1.1 Statement of Problem

Extreme flooding events are the cause of many lives, homes, economic and infrastructure losses every year. Flood extent mapping with EO satellites enable emergency response teams and policy makers to prepare with proactive response mechanisms. With accurate flood proxy maps, better evacuation plans and routes can be planned and assistance can be directed where needed in a time-critical manner. Without accurate estimations of hazardous flooded zones, emergency response bodies may misroute assistance and lose critical time.

The World Bank estimates that close to 1.5 billion people are exposed to the risk of flooding worldwide. Moreover, up to 89% of the exposed population live in low- and middle-income countries as shown in Figure 1. This disproportionate exposure means that any time an extreme flooding event occurs, the disaster can potentially wipe decades of infrastructure development and perpetuate economic hardships.

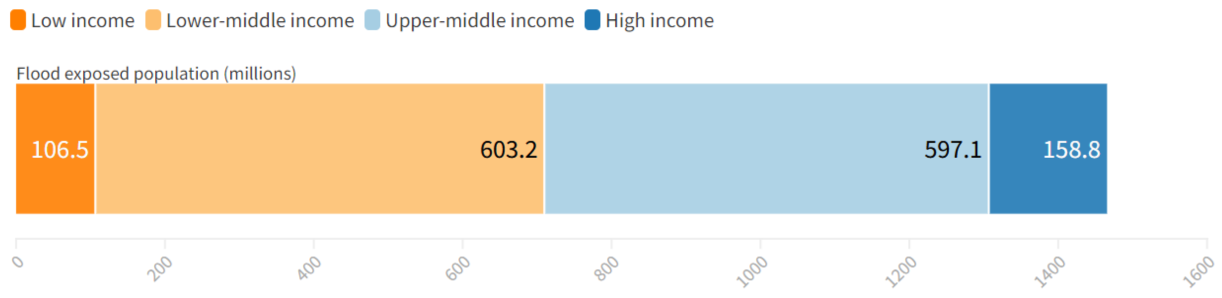


Figure 1: Flood exposed population (millions), by income group. Adapted from [5] [6]

Earth Observation applications using Synthetic Aperture Radar (SAR) satellites have gained increased popularity recently [7]. As active microwave remote sensors, SAR platforms boast the advantage of day and night and all-weather imaging capabilities. The typical wavelengths (e.g., L-, C-, and X-band) used for SAR measurements are relatively unobstructed by earth's atmosphere and can penetrate through clouds. These advantages make SAR data a key enabler for flood emergency response management [8].

Standing water bodies tend to reflect the radar signals away from the satellite and are characterized by low backscatter intensity values. The low backscatter intensity typically results in a bi-modal pixel distribution in the SAR intensity images [7]. Well known image thresholding techniques can be used to segment this bi-modal backscatter distribution (e.g., Otsu's method [9], k-Means clustering [10]), thus segmenting water pixels from non-water pixels.

Interferometric Synthetic Aperture Radar (InSAR) coherence quantifies the phase noise or quality of the phase signal in the complex-valued radar return. In repeat-pass interferometry, the coherence of a SAR scene is computed from two SAR acquisitions obtained at different times (i.e., with a known temporal baseline). The interferometric coherence is extracted as the normalized cross-correlation between the two complex SAR images [7] [11].

In this study, we exploit freely accessible SAR intensity and coherence data from the European Space Agency's Sentinel-1 platform [12]. We consider two time steps: before flooding (pre-event) and during flooding (co-event) similar to [13]. A pre-event coherence map sets a benchmark used to compare against the co-event coherence maps. When flooding occurs, the spatial distribution and electric properties of scatterers on the surface is modified by the presence of standing water. Thus, the coherence maps during the flood event tend to exhibit a loss of coherence relative to their pre-event benchmarks. We use the information from the pre- and co-event coherence maps and combine them in a bi-temporal intensity and coherence machine learning classifier to improve water mapping accuracy. The improvement of our bi-temporal SAR intensity and coherence models are relative to uni-temporal SAR intensity-only models.

1.2 Previous Work

Traditional image processing techniques such as Otsu thresholding and its variants have been used in past studies to classify water bodies using SAR imagery. Some of these algorithms have been published and operationalized as full-fledged software packages that enable widespread public use. The Hydrologic Remote Sensing Analysis for Floods (HYDRAFloods), for example, is an open source Python package that relies on Google Earth Engine for deriving surface water maps from remote sensing data and includes algorithms such as edge-Otsu [14] [15].

HydroSAR is another surface water mapping algorithm implemented in Python that uses a dynamic threshold followed by fuzzy-logic and a Height Above Nearest Drainage (HAND) layer [16] to post-process and classify flood water using Sentinel-1 imagery. The HydroSAR project is intended to be used as a near real-time flood water detection algorithm by exploiting on-demand Sentinel-1 products readily available through the Alaska Satellite Facility [17] [18] [19] [20].

Scotti et al. (2020) combine satellite imagery, hydraulic models, and markers from social media to improve post-event flood maps [21]. In addition to traditional image processing techniques, machine learning techniques have also gained increased popularity for SAR image classification tasks. In [22], Katiyar et al. (2021) use the Sen1Floods11 data set [1] and train popular convolutional neural network (CNN) architectures like the SegNet [23] and U-Net [24] using different backscatter intensity band combinations as inputs. In [25], Peng et al. (2020) train a self-supervised auto-encoder network with bi-temporal multi-spectral imagery. The bi-temporal nature allows the authors to generate change maps and extract flooded areas. Mayer et al. (2021) use an edge-Otsu dynamic threshold algorithm to automate surface water label generation to train and tune deep learning classification models [26].

Past studies have also exploited the use of coherence data for flood mapping. Pulvirenti et al. (2016) use COSMO-SkyMed imagery to analyze multi-temporal coherence trends and improve flood mapping accuracy [27]. Chini et al. (2019) use Sentinel-1 InSAR coherence to detect floodwater [28]. The authors use coherence data to extract building footprints in urban areas (Houston, Texas) and use co-event imagery to detect changes in both intensity and coherence. Chaabani et al. (2018) fuse TerraSAR-X and TanDEM-X backscatter intensity and InSAR data to train a random forest classifier and extract flood extent maps using the Richelieu River flooding in 2011 as a case study [29]. In [13], Martinis et al. combine TerraSAR-X intensity and coherence data to train an active self-learning convolutional neural network (CNN) for flood water classification. The study uses Hurricane Harvey as a case study and focuses on flood water classification in urban areas.

1.3 Objective and Methods

This study improves surface water mapping accuracy by combining Sentinel-1 backscatter intensity and interferometric coherence data at 10-meter resolution. We cross-train using Sentinel-1 and Sentinel-2 data using the publicly available Sen1Floods11 data set as our backbone [1] [30]. We augment the Sen1Floods11 intensity data with pre-event intensity data from Google Earth Engine, and on-demand InSAR products offered by the Alaska Satellite Facility [20]. The augmented Sentinel-1 intensity and interferometric coherence data set is used as input to an XGBoost and Attention U-Net binary classifier trained on Sentinel-2-derived water labels. A co-event intensity, a bi-temporal (pre- and co-event) intensity, and bi-temporal intensity and coherence model is trained and benchmarked with one another. Each model is validated against a hand-labeled data set across geographically diverse regions. The hand-labeled data set is never used during training and serves as an independent means to validate our models' relative improvements.

As outlined above, previous studies have fused SAR intensity and coherence data for flood mapping applications. However, most of the studies have focused on well-documented flood events such as Hurricane Harvey and other urban flood events imaged with high resolution SAR platforms. This study seeks to validate whether Sentinel-1 coherence data can systematically improve semantic water segmentation at 10-meter resolution across diverse geographical regions. InSAR product processing is extremely computationally expensive and requires specialized understanding of the processing pipeline. For this study, we take advantage of on-demand Sentinel-1 InSAR products freely available through the Alaska Satellite [20] in an attempt to understand whether fusing SAR backscatter intensity and InSAR coherence data can be democratized. Moreover, we note that the models trained were not designed to optimize absolute classification performance relative to some state of the art benchmark. Instead, the experiments were set up to assess the relative improvement gained by fusing intensity and interferometric coherence data when compared against an intensity data-only model.

This report is organized as follows: Section 2 presents the fundamentals of microwave remote sensing, synthetic aperture radar and InSAR applicable to our study. Section 3 describes the data processing used to train the machine learning models described in Section 4. Section 5 outlines our results and discussion including trade-offs. We conclude with Section 6 where potential areas of future work are explored. Finally, Appendix A presents code samples used to train and evaluate the classification models. The entire code base used for this study can be found in [31].

2 Background

This section describes fundamental concepts of microwave remote sensing. We start by defining electromagnetic radiation and its properties relevant to remote sensing applications. The field of microwave remote sensing is very vast and spans many applications. In this study, we are interested in defining the fundamental concepts with a focus on Synthetic Aperture Radar (SAR). The information exposed in this section is adapted from [2] [7] [8] [11] [32] [33] [34] [35].

2.1 Microwave Remote Sensing

Electromagnetic Radiation Wave Interpretation

The wave nature of light describes electromagnetic (EM) radiation as energy in the form of time varying electric and magnetic fields propagating through space. Electromagnetic radiation propagating through vacuum travels at the speed of light ($c = 3 \times 10^8 m/s$) and spans a broad spectrum of wavelengths as shown in Figure 2. The electromagnetic spectrum ranges from radio waves with wavelengths in the order of 10^2 meters, visible light in the optical range, up to gamma waves with wavelengths in the order of 10^{-12} meters [36].

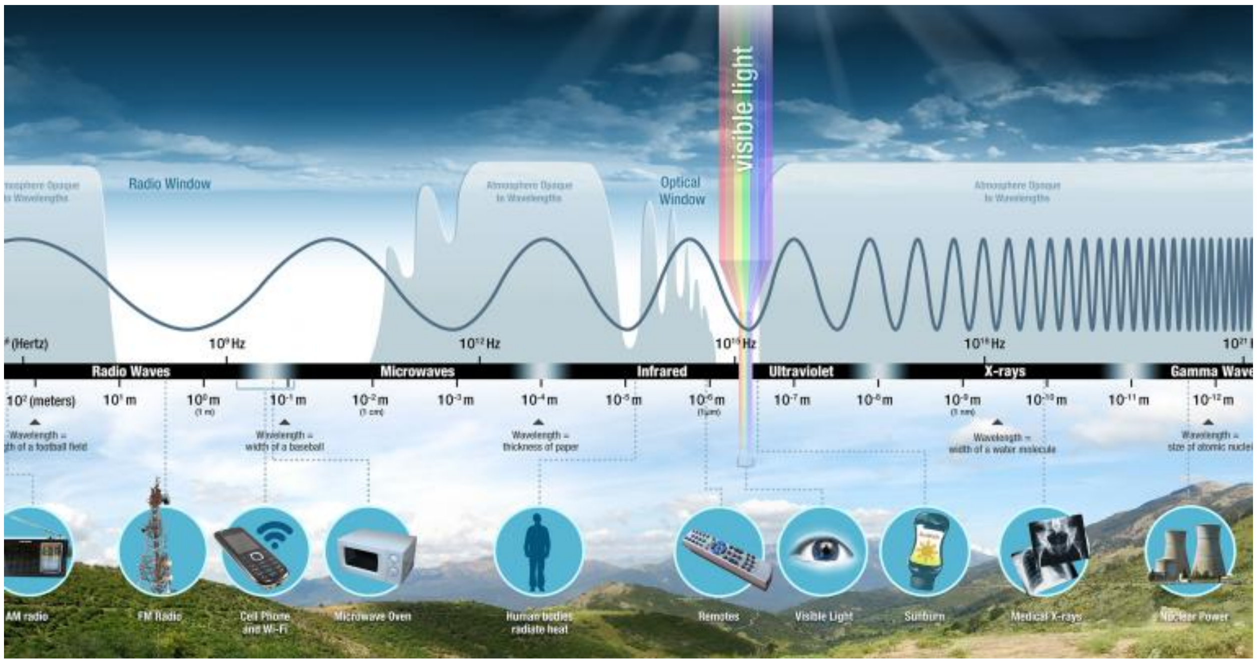


Figure 2: Electromagnetic Spectrum. Adapted from [36]

Complex EM Wave Description

Mathematically, an electromagnetic wave can be described by an amplitude component, a phase component, and a time varying spatial component (i.e., a direction of travel). The amplitude of the electromagnetic wave is related to the energy or power carried by the radiation. The phase component describes the wave's phase evolution as it propagates in time and space.

Equation 1 describes a complex electromagnetic wave propagating along the z-axis in three-dimensional space. The amplitude component of the wave is denoted by E, ω is the angular frequency in radians per second, k is the wave number describing the spatial frequency of the wave, and ϕ_0 is an arbitrary initial phase component.

$$\Psi(z,t) = Ee^{j(\omega t - kz + \phi_0)} \tag{1}$$

The oscillatory nature of complex electromagnetic waves as described in Equation 1 is illustrated in Figure 3. On the left-hand side, a phasor rotates counterclockwise as a function of the angular frequency ω and time t. At any given time, the phasor subtends an angle ϕ with the a-axis. As the phasor rotates, its amplitude vector traces the sinusoidal waveform seen on the right. The time varying phase angle, ϕ, determines the phase of the sinusoidal oscillations as they propagate through space. This vector representation will be useful in later sections when we consider the combination of multiple waves, or interference.

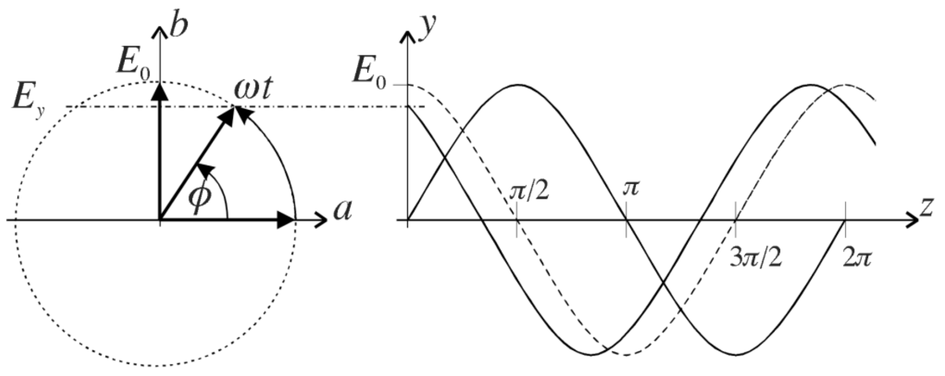


Figure 3: Phasor representation of an EM wave. Adapted from [11]

Energy and Power of EM Radiation

The amplitude of an electromagnetic wave defines the energy it carries. In remote sensing applications, the sensors measure the incident power, P, of the impinged radiation at the antenna. Power is defined as energy per unit time (Watts) and is mathematically described as the square of the wave's amplitude divided by the impedance η of the propagating medium as outlined in Equation 2.

$$P = |E(z,t)|^2 = \frac{E_0^2}{2\eta} \quad [W] \tag{2}$$

In Earth observation, we are usually interested in characterizing targets over distributed areas (e.g., geographical regions). Thus, it is best to work with the power density defined as power per unit area $[W/m^2]$. We are also interested in differentiating between incident radiation *on* a given area and radiation emanating *from* the target area. We define the total amount of power incident on a unit area as the irradiance, E, and the total amount of power emitted or reflected from a unit area as the exitance, M.

Moreover, because we are often interested in characterizing the electromagnetic power arriving from different directions and different frequencies, we define the *spectral radiance* as the quantity of interest in many remote sensing applications. *Spectral radiance* is also known as *brightness* or *intensity* and is defined as a measure of the amount of energy incident from a given direction at a specific frequency of radiation. The spectral radiance has units of $[Wm^{-2}sr^{-1}Hz^{-1}]$, where sr are steradians for a solid angle defining direction. This intensity measurement will be key in our understanding of the operation of a synthetic aperture radar platform in later sections.

Wave Polarization

Electromagnetic waves oscillate orthogonal to the direction of propagation (i.e., they are transversal). A transverse wave can be defined as the superposition of two orthogonal waves (x- and y-direction) as outlined in Equation 3. Each of the orthogonal waves is said to have a polarization associated with them. Figure 4 shows the orthogonal components having displacements in the x- or y-direction are said to be horizontally or vertically polarized respectively.

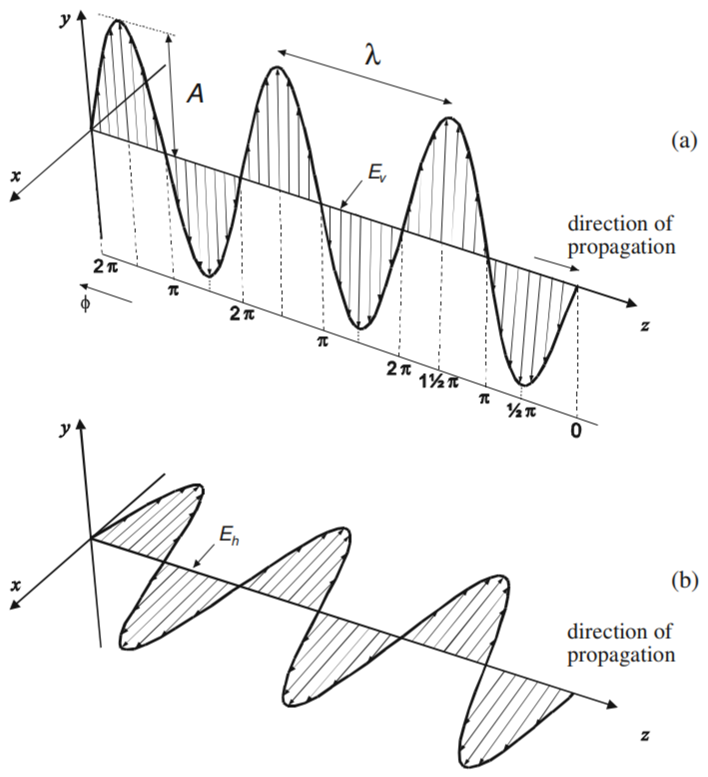


Figure 4: Horizontal and vertical polarizations in EM waves. Adapted from [11]

$$E(z, t) = E_x(z, t) + E_y(z, t) \tag{3}$$

Interference and Coherence

In remote sensing applications, we typically work with multiple electromagnetic waves at the same time. Coherent waves are defined as two waves with a phase difference that remains constant over time. When multiple waves are combined or superimprosed in space, we get *interference* between them. For example, if two waves with identical phase and amplitude, A, are superimposed, the result will be a wave with double the amplitude, $2A$. This additive nature of in-phase waves is referred to as *constructive* interference. On the flip side, if two waves that are out of phase (i.e., a phase difference of π radians) interfere, the resulting amplitude will be zero. This latter case is referred to as *destructive* interference. More generally, the superposition of waves will result in wave amplitudes between 0 and $2A$ depending on the relative phase difference between the waves. Figure 5 depicts constructive interference on the left-most plot, destructive interference in the middle plot, and interference when the phase difference between two waves is $\frac{\pi}{2}$ radians.

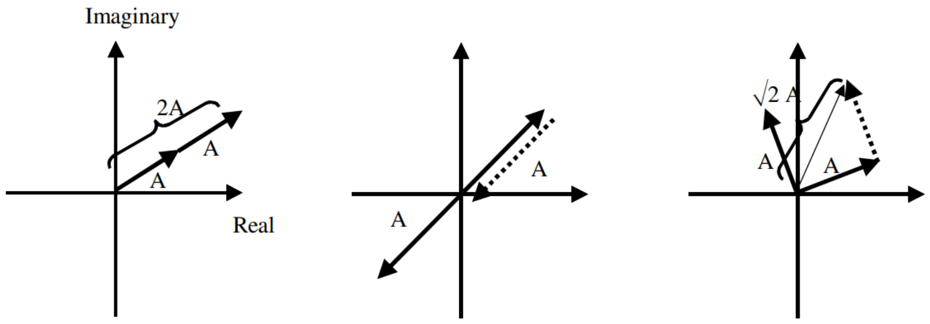


Figure 5: Constructive interference (left), destructive interference (middle), interference for $\frac{\pi}{2}$ phase difference. Adapted from [11]

Coherence

As described above, two waves are coherent if their phase difference is constant over time. Another way to define coherence is two waves with the same frequency of oscillation. Physically, coherence measures the degree of similarity between the waves over an interval of time or space.

Mathematically, the complex coherence between two waves is defined by the complex cross-correlation outlined in Equation 4.

$$\Gamma_{12} = < E_1 E_2^* > \tag{4}$$

Equation 4 defines coherence as an ensemble average over different realizations of the same electromagnetic waves, E_1 and E_2. However, when using satellites for Earth observation, we have a limited amount of time to make measurements as the platform orbits the earth. This means that we cannot gather different realizations of the same measurements. Coherence is then estimated by making multiple measurements that are either close in time or close in space. That is, the average is calculated over a series of measurements taken at different times or measurements taken over a number of locations. Our coherence measurement is then defined as shown in Equation 5 where the complex coherence of Equation 4 has been normalized to account for time varying amplitudes of the repeated measurements.

$$\gamma = \frac{\sum_N E_1 E_2^*}{\sqrt{\sum_N |E_1|^2 \sum_N |E_2|^2}} \tag{5}$$

In Equation 5, the subscript N means we are averaging over N observations (e.g., a collection of pixels). Since we are normalizing by the magnitudes, coherence will vary from 0 to 1, or from *incoherent* to *coherent*. The topic of coherence between waves is at the heart of this study and will be revisited in later sections.

Propagation of EM Radiation

Next, we consider the propagation of electromagnetic waves through different media. The media of propagation affects a wave's properties such as speed of propagation, attenuation, and phase shift. The electric permittivity, ϵ, the magnetic permeability, μ, and the electric conductivity, g are used to characterize the electromagnetic properties of media.

The conductivity of a medium is related to the mobility or lack of mobility of electrons in the material. In metals, the electrons are not bound to the atoms and are free to move through the metal. For an ideal conductor, the electric field is zero inside the conductor, and therefore, conductors will reflect any incident EM radiation.

In microwave remote sensing, we are concerned with the relative permittivity of media and targets. Permittivity describes how an electric field propagates through a material. The permittivity of media is defined in Equation 6 where ϵ_r is the relative permittivity and ϵ_0 is the permittivity of free space.

$$\epsilon = \epsilon_r \epsilon_0 \tag{6}$$

The relative permittivity is a complex quantity and can be expressed as outlined in Equation 7. The real part is also known as the dielectric constant of the dielectric medium. In this representation we can think of the real part as the lossless dielectric constant while the imaginary part describes the energy losses through the medium.

$$\epsilon_r = \epsilon_r^{'} - j\epsilon_r^{''} \tag{7}$$

The relative permittivity also defines the penetration depth of EM radiation into a medium as shown in Equation 8. Penetration depth is defined as the distance at which the power is reduced by a factor e. Note the linear dependence of penetration depth on the wavelength. This means that longer wavelengths will be better at penetrating deeper into materials.

$$\delta_p \approx \frac{\lambda\sqrt{\epsilon_r^{'}}}{2\pi\epsilon_r^{''}} \tag{8}$$

2.2 Why Microwave Remote Sensing?

Earth observation from space is a key player in our ability to study and understand our world. The applications of EO satellites are vast and include activities such as monitoring environmental changes and earth surface dynamics, disaster management, and ship routing to name a few. Earth observation using microwave remote sensing entails using the microwave region of the electromagnetic spectrum to measure the scattering properties of the earth's surface. Unlike their optical counterparts which rely on external sources of illumination (e.g., the sun) and cannot image through clouds, microwave remote sensing satellites can penetrate through clouds and image the earth at any time of day. These advantages makes microwave remote sensing techniques complementary to their optical counterparts and expand our abilities to understand our world.

We can categorize microwave remote sensing instruments as either passive or active. Passive sensors, or radiometers, measure the microwave energy that is radiated (by thermal emission) or reflected by Earth's surface or the atmosphere. Active sensors such as radar systems, have their own illumination source and transmit EM energy that is reflected off objects on Earth's surface and measured by the sensor's receiver. Active microwave sensors typically use frequencies below 10 GHz, operating in the so-called microwave window. Figure 6 shows the atmospheric attenuation over the range of the EM spectrum. Note that for wavelengths in the 3 to 30 *cm* range, the EM radiation can travel relatively unobstructed through the atmosphere with very low attenuation.

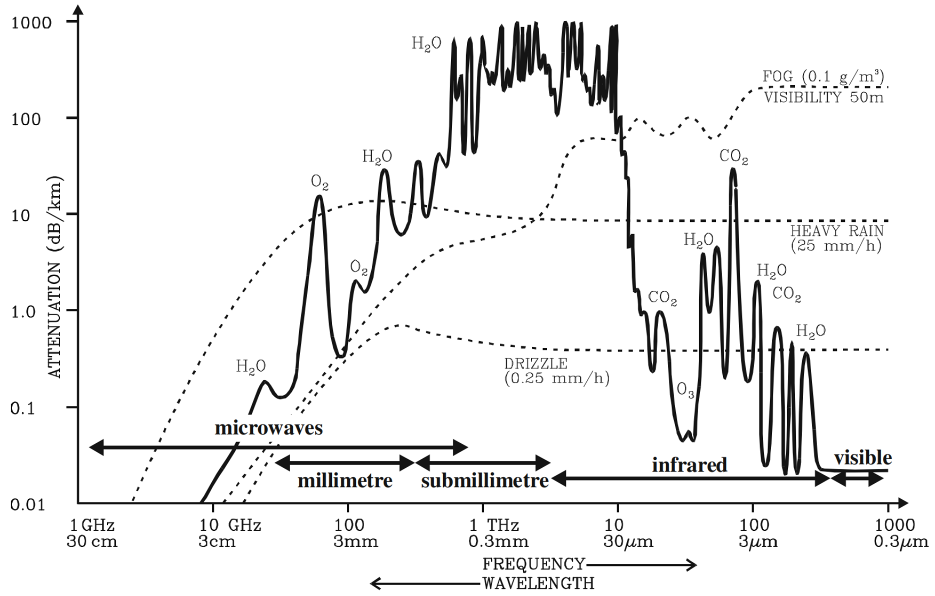


Figure 6: Atmospheric attenuation for a clear atmosphere as a function of wavelength. Adapted from [11]

Active Microwave Sensing

Our study uses active imaging radars for remote sensing applications and will be the main focus in subsequent sections. There are many active microwave remote sensing applications and platforms. For example, airborne altimeters send microwave pulses in the nadir-direction (perpendicular to the direction of travel) and measure distance from targets by analyzing the echos scattered back. Scatterometers, on the other hand, measure echo characteristics such as the radar cross-section of the targets. Most practical systems use a combination of the different types of measurements to analyze targets. In the next subsection, we explore the fundamental principles of radar remote sensing.

2.3 What is RADAR?

The altimeter system described briefly in Section 2.2 is an example of a radar system. A Radar (RAdio Detection and Ranging) system relies on the transmission of microwave energy and the measurement of the echoes returned from its targets to extract range or distance information. This is the principle of *echolocation*. The range information is calculated by measuring the time of the return signal. Sonar, LIDAR, and ultrasound are examples of echo-based remote sensing. Radar systems sense the environment in terms of surface roughness, signal reflectivity, and relative motion (e.g., Doppler effects) between the targets and the radar platform. This is what makes microwave remote sensing platforms complementary to optical remote sensing platforms. With microwave remote sensors, we extract surface properties and characteristics that are not available from optical sensors.

2.3.1 Basic Radar Operation

In this section, we introduce the radar equation, the radar cross-section, and the range resolution relationships that will be useful in our study and application of imaging radar systems.

Radar echoes tend to be very low-powered signals returned from the targets. This is especially true in remote sensing applications where the radar platform orbits the earth a few hundred kilometers above the surface. Additionally, the return signals are also hindered by noise which makes detecting the echoes more challenging. The radar equation describes what portion of the transmitted signal is scattered from a target as will be discussed next.

The Radar Equation

For a radar system with transmit power, P_t, antenna gain, G, range to target R, and target's radar cross-section, σ, the power scattered back from the target is given by Equation 9.

$$\text{power scattered by target} = P_t G \frac{\sigma}{4\pi R^2} \quad [W] \tag{9}$$

We are interested in calculating the power density at the radar's receiver, P_r. The effective area of the receiving antenna relative to a sphere of radius R is denoted by A_e. The antenna's effective area is equivalent to its cross-section. Moreover, because the return signals travel an additional distance R back to the receiver, the signal power received at the antenna is reduced by an extra factor of $4\pi R^2$. We can then write the received power as shown in Equation 10.

$$P_r = (P_t G \frac{\sigma}{4\pi R^2}) \frac{A_e}{4\pi R^2} \quad [W] \tag{10}$$

It is useful to express the radar equation in terms of its wavelength dependence. We can relate the effective area of the antenna to its gain by: $A_e = \frac{G\lambda^2}{4\pi}$ and re-write Equation 10 as shown in Equation 11.

$$P_r = \frac{P_t G^2 \lambda^2 \sigma}{(4\pi)^3 R^4} \quad [W] \tag{11}$$

From Equation 11, we note that the received signal power has a $1/R^4$ dependence on the range to the target. If we double the range to the target, our received power drops by a factor of sixteen, for example. Since our radar equation has a linear dependence on transmit power, we can in theory counteract the $1/R^4$ roll-off by increasing the transmit power. However, there are practical limits (e.g., size and cost) to how much power a radar system can generate. Increasing the receive antenna's gain is another way to improve our received signal power, but this increase gain would come at the expense of a heavier, larger antenna. Lastly, we focus on the received power's dependence to wavelength. The way that Equation 11 is written masks the true relationship between P_r and λ. Since the antenna gain, G, is proportional to $1/\lambda^2$, the received power will also have a $1/\lambda^2$ dependence. This means that shorter wavelengths result in higher received powers.

Radar cross-section (RCS)

The radar cross-section of a target describes the equivalent area seen by the radar. We can rearrange Equation 11, and arrive at the target's radar cross-section as shown in Equation 12.

$$\sigma = P_r \frac{(4\pi)^3 R^4}{P_t G^2 \lambda^2} \tag{12}$$

In remote sensing, we are typically working with distributed area targets. Thus, we define normalized radar cross-section as shown in Equation 13, where A represents the target's area.

$$\sigma_0 = \frac{\sigma}{A} = P_r \frac{(4\pi)^3 R^4}{A P_t G^2 \lambda^2} \tag{13}$$

Radar Range Resolution

A radar system's ability to distinguish between two target points is referred to as the range resolution. Following the principle of echolocation, if the two targets are physically close to one another such that their return echoes overlap, the radar will not be able to resolve one echo from the other. This implies that the range resolution depends on the radar's transmitted pulse duration, τ_p. Mathematically, the range resolution can be written as shown in Equation 14, where the factor of 2 accounts for the round-trip nature of the signal and c is the speed of light in vacuum.

$$\rho_R = \frac{c \cdot \tau_p}{2} \quad [m] \tag{14}$$

Given the linear relationship between range resolution and the pulse duration, we might be tempted to decrease the pulse duration as much as possible and improve our range resolution. However, practical systems cannot simultaneously shorten the microwave pulse duration and generate high transmit peak powers needed to increase the received power at the antenna. Frequency chirping is often used as a technique to balance the trade-off between high transmit powers and range resolution. Pulse chirping uses frequency modulation (FM) to encode the transmitted signal with a unique signature. At the receiver, the return signal can be correlated or match-filtered and decoded to separate the pulses from one another. The premise is that even if the frequency modulated pulses overlap with one another, a unique frequency modulated encoding will be sufficient to differentiate one pulse from another at the receiver.

The frequency modulation or chirping sweeps the signal over a predefined bandwidth, B_p. At the receive end, the sharpness of the pulse is determined by the *effective pulse length*, τ_e, which is related to the chirp's bandwidth as shown in Equation 15.

$$\tau_e = \frac{1}{B_p} \quad [s] \tag{15}$$

The match filtering of the encoded FM chirp is also known as *pulse compression.* Substituting the effective pulse length, τ_e into Equation 14, the range resolution for the radar system is given by Equation 16.

$$\rho_R = \frac{c}{2B_p} \quad [m] \tag{16}$$

Note that the range resolution is now dependent on the inverse of the chirp's bandwidth and not the individual pulses' length. The wider the FM bandwidth used, the finer the range resolution of our radar system. Moreover, note that the range resolution has no dependence on distance to the target. In other words, a radar's range resolution will be the same regardless of the flying altitude.

2.4 Radar Image Formation

In this section, we consider radar imaging systems and lay the foundation for microwave remote sensing using imaging radars.

2.4.1 Side-Looking Airborne Radar Systems (SLAR)

The radar altimeters described in Section 2.2 and most airborne or space-borne optical sensors image the earth by pointing their sensors towards nadir (perpendicular to the azimuth direction). In contrast, imaging radars point their antenna at a look angle, θ_l, away from nadir as depicted in Figure 7. This side-looking geometry allows the imaging platform to differentiate objects that are at the same distance from the imaging sensor as shown in Figure 8. If the antenna were pointed in the nadir direction, as in Figure 8, the return signals from objects A and B arrive at the radar system at the same time and the radar would not be able to differentiate object A from B.

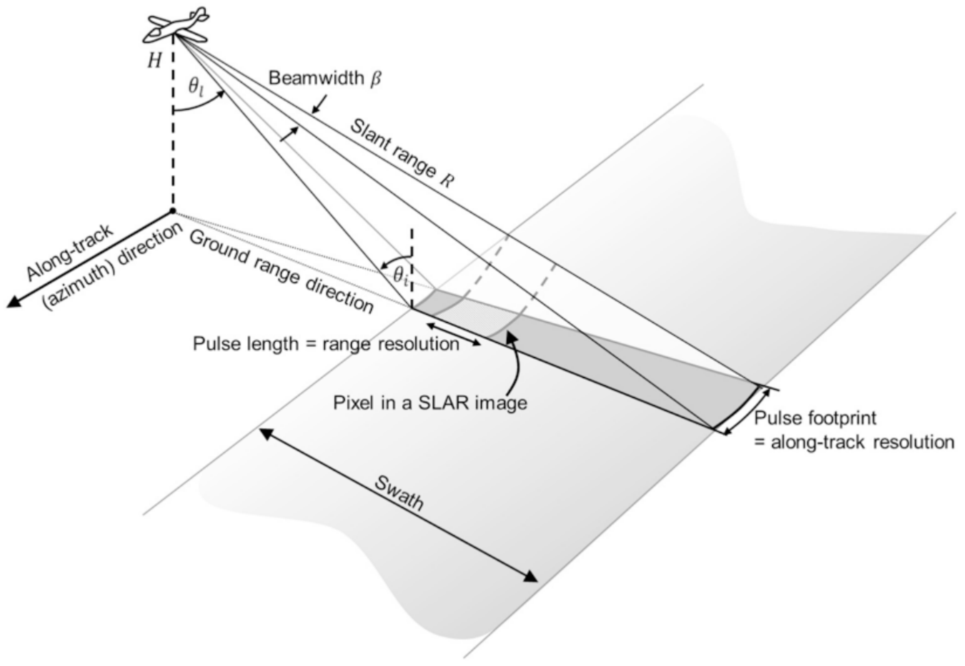

H
θ_l
Beamwidth β
Slant range R
Along-track (azimuth) direction
Ground range direction
θ_i
Pulse length = range resolution
Pixel in a SLAR image
Pulse footprint = along-track resolution
Swath

Figure 7: Side-Looking Airborne Radar (SLAR) viewing geometry. Adapted from [32]

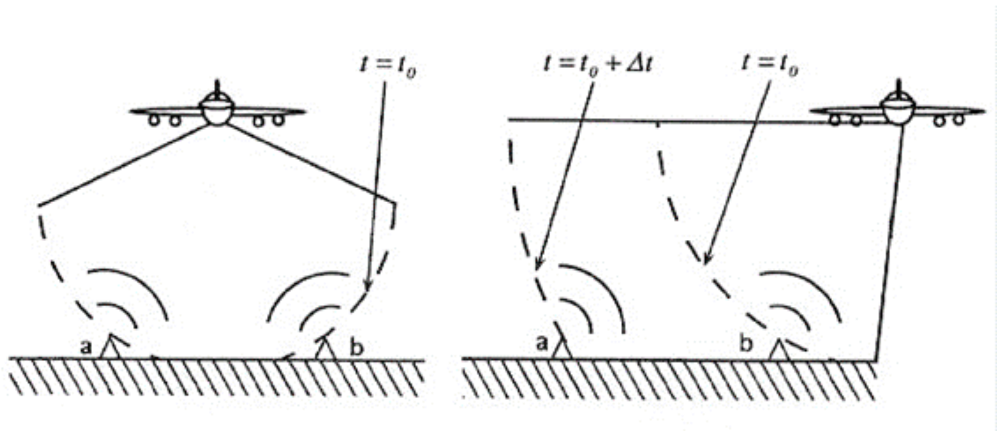


Figure 8: Different airborne radar viewing geometries. Nadir-direction (left) and side-looking direction (right). Adapted from [7]

Figure 7 depicts the geometry of a SLAR imaging platform. The radar system points its antenna at an angle to the surface as it flights along the azimuth or along-track direction.

As the SLAR transmits a series of microwave pulses of length, τ_p, the pulses illuminate an instantaneous area on the ground denoted as the *swath* in the figure. The slant range, R, refers to the round-trip distance the microwave pulses travel from the sensor to the target surface and back. Note that the slant range is different from the true ground range projected on the surface. In order to derive the true ground range from the slant range, we need to account for the surface characteristics (e.g., curvature, elevation, etc.) of the target and calibrate the radar return signal.

The size of the radar footprint on the ground depends on the wavelength, λ, the size of the radar's antenna, L, and the distance of the sensor from the ground, R. Defining the antenna beam width as $\beta = \lambda/L$, we can define the radar's footprint as shown in Equation 17.

$$S \approx \frac{\lambda}{L} R = \beta \cdot R \;\; [m] \tag{17}$$

As the SLAR platform flies along the azimuth direction, the return signals are processed by arrival time in azimuth and range direction. This process allows us to form a two-dimensional image of the surface. Note from Figure 7 that echoes received from the near-range of the platform arrive earlier than those from the far-range edge of the swath. This introduces challenges when interpreting and extracting information from the SLAR images. Another challenge introduced by the side looking geometry is that some of the transmitted energy is reflected away (in the specular direction) from the targets. This results in smaller radar cross-sections or backscatter intensity at the receiver.

Ground Range Resolution

The ground range resolution is defined as the minimum ground distance needed to differentiate one object from another. From Figure 9, we can define the ground range resolution as a function of the range resolution from Equation 16 and the local incidence angle, θ_i, as outlined in Equation 18.

$$\rho_G = \frac{\rho_R}{\sin\theta_i} \quad [m] \tag{18}$$

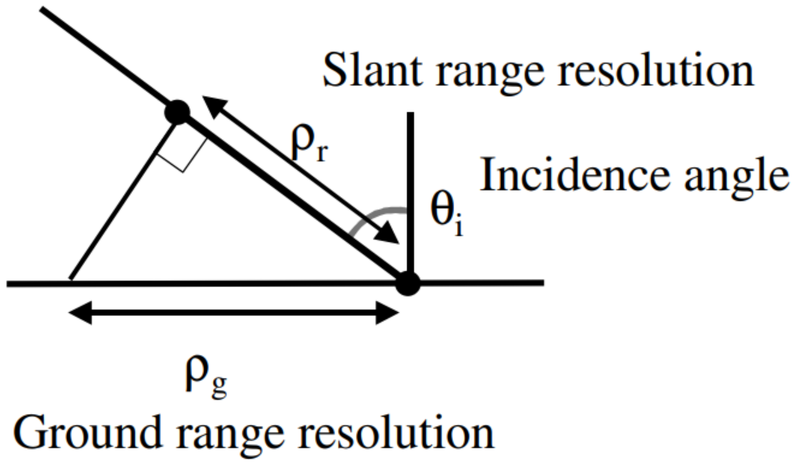


Figure 9: Slant range resolution and ground range resolution relationship. Adapted from [11]

Note that the ground resolution depends on the local incidence angle, and therefore, is not constant across the swath. As the incidence angle increases, the ground resolution gets progressively better. The near edge of an image has poorer ground resolution than the far edge of the swath.

Azimuth Resolution

The azimuth resolution is defined as the ability of the radar to distinguish two points within the same swath, but at different azimuth angles. The azimuth resolution depends on the antenna's beam width in the azimuth direction as outlined in Equation 19.

$$\rho_{AZ} = S_{AZ} \approx \frac{\lambda}{L_{AZ}} R = \beta \cdot R \quad [m] \tag{19}$$

From Equation 19, note that the azimuth resolution decreases as the distance from the imaging system to the ground, R, increases. This result poses a challenge for space borne imaging platforms where the distance between the sensor and the ground is in the order of hundreds of kilometers. We could in theory increase the antenna's size, but we quickly run into impractical antenna lengths for space-borne systmes. The synthetic aperture principle was introduced in the 1950's to alleviate these drawbacks of SLAR systems as will be described in Section 2.5.

2.5 Synthetic Aperture Radar (SAR) Imaging Systems

In order to overcome the azimuth range resolution challenges outlined above, a synthetic aperture radar imaging system synthesizes a larger antenna aperture from a shorter physical antenna. Figure 10 shows a diagram of a SAR imaging platform. Similar to the SLAR systems, as the space-borne or airborne SAR platform moves in the along-track direction, the short antenna emits microwave pulses that are reflected back from objects on the earth's surface. Each pulse illuminates a footprint of size S on the ground, which is typically in the order of several kilometers for space-borne SAR systems. The aperture synthesis concept relies on the fact that subsequent transmitted pulses have overlapping footprints for a given target, P, on the surface. The overlapping echoes are post-processed to create a resulting image as if it was acquired with a larger physical antenna. The length L_{SA} of this synthesized antenna can be calculated by Equation 20. Figure 2.5 shows an example of a C-band SAR image acquired by the Sentinel-1 mission over New York City.

$$L_{SA} = \frac{\lambda}{L} \cdot R_0 = \beta \cdot R_0 \quad [m] \tag{20}$$

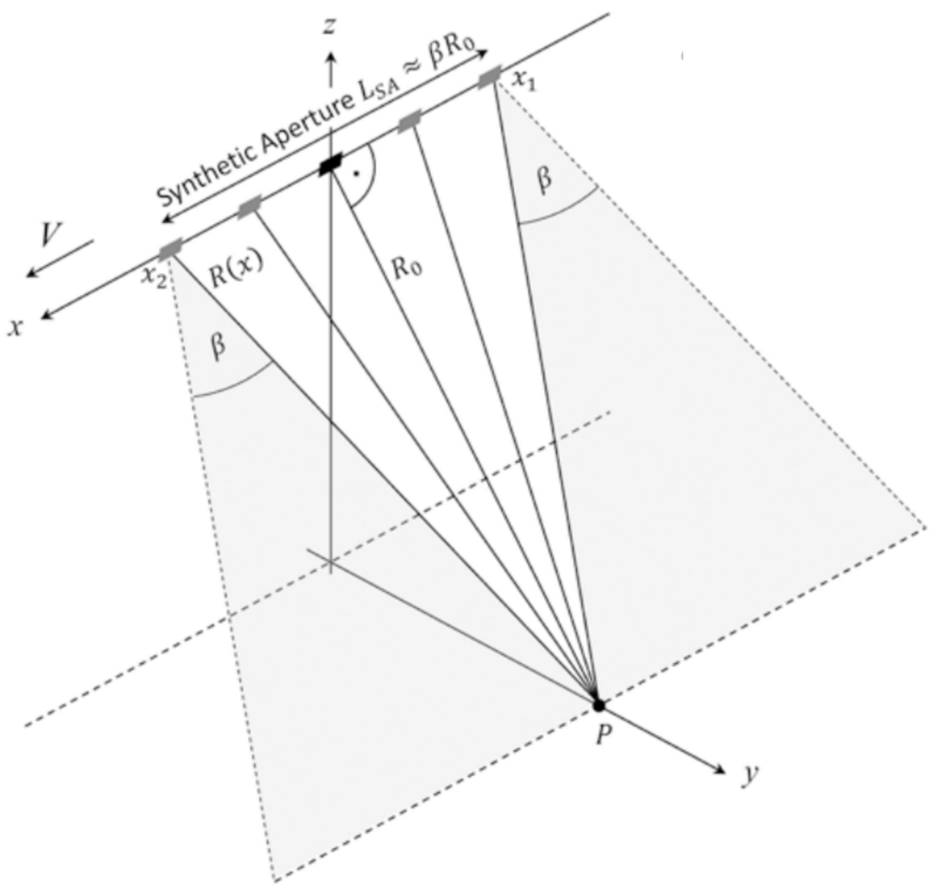


Figure 10: Synthetic Aperture Radar (SAR) platform viewing geometry. Adapted from [32]

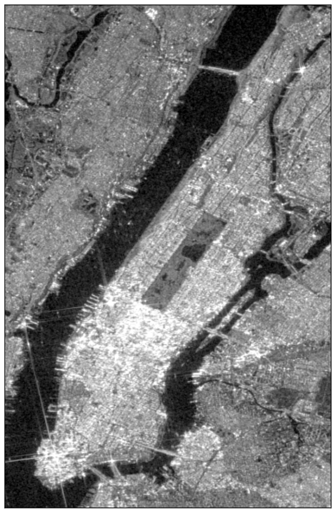


Figure 11: Sentinel-1 SAR image over New York City on February 3rd, 2022. Sentinel-1 GRD data processed by ASF [20]

2.5.1 A Doppler Interpretation of Synthetic Aperture Radar (SAR)

This section provides a Doppler interpretation of the synthetic aperture radar concept adapted from [11].

Figure 12 depicts another look at the geometry of a space-borne synthetic aperture radar system. The antenna footprints are designed wide enough in the azimuth direction so that consecutive pulses overlap. As the satellite moves in the azimuth direction, the radar returns from the front part of the beam are Doppler-shifted to higher frequencies relative to the transmit signal's frequency. On the other hand, the radar returns from the tail part of the beam are shifted to lower frequencies. These frequency shifts result in a per pulse frequency spread similar to a chirp or frequency modulation. At the receiver, the chirps are match-filtered to determine which pulse the echo originated from. This technique is known as *azimuth* compression. Instead of cross-correlating the chirp from a single pulse echo as in pulse compression, the cross-correlation is now performed over a collection of returned overlapping signals. This collection of overlapping signals is post-processed to synthesize a larger virtual antenna.

There are challenges with azimuth compression, however. In range compression, the radar transmitter controls the FM chirp transmitted. In azimuth compression, the radar system lacks precise transmit signal control. There is no definite or repeatable reference to cross-correlate with since the frequency shifts depend upon the imaging geometry and structure of the targets on the surface. In practice, the reference signal is usually determined by a combination of known platform parameters and correction functions that are constantly updated.

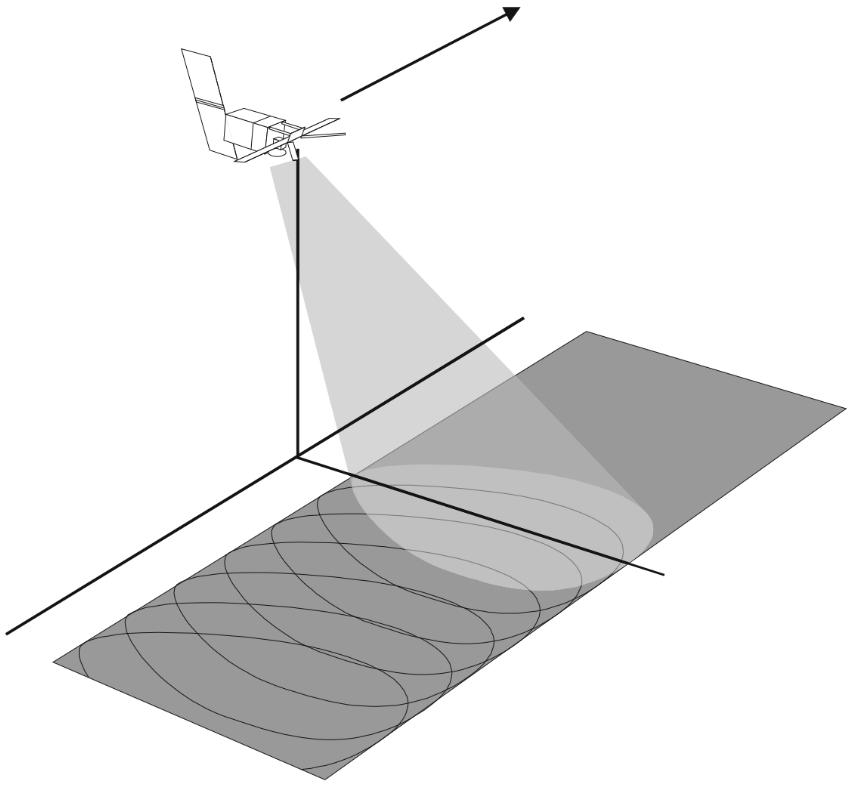


Figure 12: Geometry of a Synthetic Aperture Radar (SAR) platform. Adapted from [11]

Nonetheless, with the azimuth compression interpretation described above, the azimuth resolution for a SAR system can be calculated. Similar to the range compression equations, the equivalent pulse width for the azimuth resolution is related to the inverse of the chirp's bandwidth. The effective azimuth pulse length, ρ_t is given in Equation 21, where B_D is the Doppler shift bandwidth.

$$\rho_t = \frac{1}{B_D} \quad [s] \tag{21}$$

Given the sensor's flight speed, the azimuthal spatial resolution can be calculated as shown in Equation 22. The only unknown is the Doppler bandwidth, B_D.

$$\rho_a = \frac{V_s}{B_D} \quad [m] \tag{22}$$

Figure 13 depicts the SAR platform's geometry from the sensor's perspective. That is, the satellite is stationary and the target is moving relative to the satellite. The relative velocity of the target as observed by the sensor is given by $V_{rel} = V_s \sin\theta_a$. The maximum Doppler shift is obtained when the target enters or leaves the radar beam. The minimum Doppler shift occurs when the target is perpendicular to the sensor at position x_0. Because of symmetry, we can define f_D as the maximum Doppler frequency shift, and thus our range of echo frequencies will be $(f_0 - f_D)$ to $(f_0 + f_D)$ where f_0 is the transmit signal's center frequency.

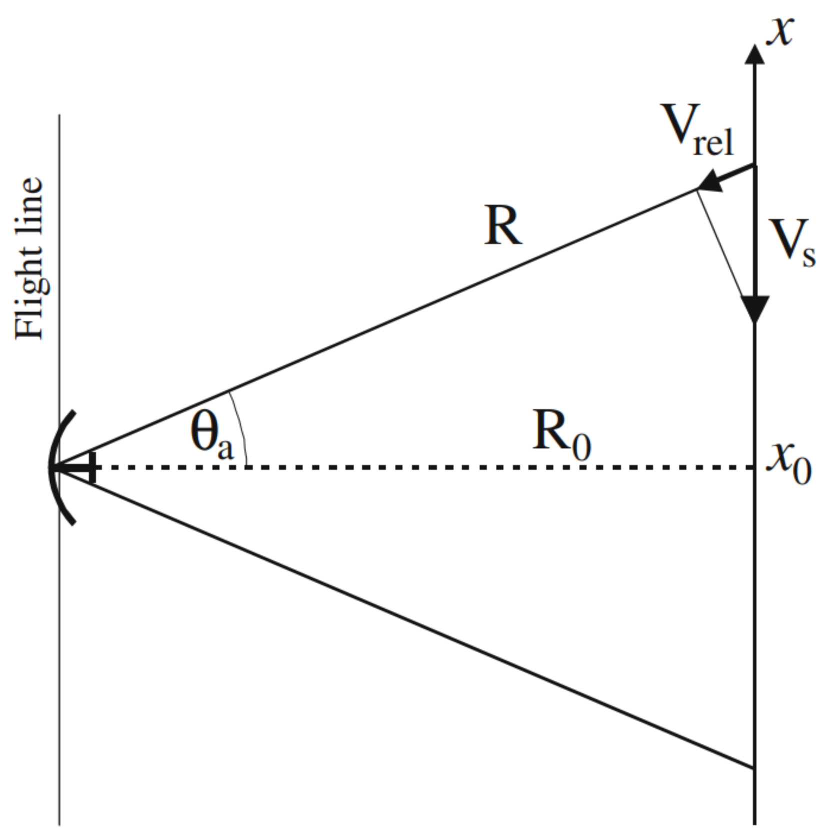


Figure 13: Geometry of a Synthetic Aperture Radar (SAR) platform from the sensor's perspective. Adapted from [11]

The Doppler frequency shift is outlined in Equation 23 where the factor of 2 is due to the fact that the signal is Doppler-shifted twice.

$$f_D = 2\frac{V_{rel}}{\lambda} \quad [Hz] \tag{23}$$

From Equation 23, we can calculate the Doppler bandwidth as shown in Equation 24 below.

$$\begin{aligned} B_D &= (f_0 + f_D) - (f_0 - f_D) \\ &= 2f_D \\ &= \frac{4V_{rel}}{\lambda} \\ &= \frac{4V_s \sin\theta_a}{\lambda} \end{aligned} \tag{24}$$

Because the angle, θ_a is half the beam width of the antenna of length, D, it follows from Equation 25 and the small angle approximation:

$$\sin\theta_a \approx \theta_a = \frac{1}{2}\frac{\lambda}{D} \tag{25}$$

Substituting Equations 24 and 25 into Equation 22, the azimuth range resolution is given by Equation 26.

$$\begin{aligned} \rho_a &= \frac{V_s}{B_D} \\ &= \frac{2\lambda D V_s}{4\lambda V_s} \\ &= \frac{D}{2} \end{aligned} \tag{26}$$

The result summarized by Equation 26 implies that a SAR system has an azimuth resolution equal to half the length of the antenna. Moreover, the smaller the antenna, the better the azimuth resolution. Note that the azimuth resolution does not depend on the distance of the sensor to the target nor does it depend on the wavelength of the signal. This result may seem counter-intuitive, but recall that in a SAR system, the azimuth compression is what determines the resolution, and not the actual angular beam width of the antenna. In fact, the larger the range of Doppler frequencies, the better the azimuthal resolution. These are promising results, but there are practical limitations in real-world systems that degrade the theoretical predictions

It is important to note that in the Doppler interpreation of SAR systems, we've used the frequency shift to define the Doppler bandwidth. However, in practice, the radar system does not measure the frequency directly. Instead, the relative phase differences between transmit and received signals are measured.

2.5.2 SAR Radar Equation

The radar equation presented in Equation 11 in Section 2.3.1 describes the received power at the antenna of the radar system. If we rewrite the radar equation as a signal to noise ratio (SNR), where N_0 is the instrument noise, we arrive at the SAR radar equation outlined in Equation 27. However, this equation is only relevant for a single received echo. For a practical SAR system, the aperture synthesis process relies on the coherent addition of multiple received echoes. If we consider n overlapping echoes, the SNR gets multiplied by a factor of n such that $SNR_{SAR} = n \cdot SNR_{single}$.

$$\frac{P_r}{N_0} = \frac{P_t G^2 \lambda^2 \sigma}{(4\pi)^3 R^4 N_0} \tag{27}$$

For a given target, the number of echoes received, n, is the length of time the target remains illuminated by the radar beam multiplied by the pulse repetition frequency (PRF) as outlined in Equation 28.

$$n = \frac{R\lambda}{2V_s \rho_a} PRF \tag{28}$$

Substituting Equation 28 into Equation 27 yields Equation 29, where the radar cross section, σ, has been normalized by the antenna's footprint area such that $\sigma = \sigma_0 \rho_R \rho_a$.

$$\frac{P_r}{N_0} = \frac{P_t G^2 \lambda^3 \sigma_0 \rho_R PRF}{(4\pi)^3 R^3 N_0 2 V_s} \tag{29}$$

Equation 29 implies that to maintain a high signal to noise ratio, we need a high pulse repetition frequency and a low platform velocity. This results in increasing the number of echoes received for any target within the scene.

2.5.3 SAR Distortions - Radiometric Distortions

Syntehtic aperture radar images exhibit a number of distortions that need to be accounted for in order to extract meaningful information from them. Some of the distortions are caused by the side-looking geometry of the platforms while others are caused by the way that electromagnetic radiation interacts with the targets on the surface. This section will summarize the main radiometric and geometric distortions that impair SAR systems and some of the techniques used to address them.

Speckle Noise

SAR images are characterized by a salt and pepper-like noise appearance known as speckle noise. Figure 14 shows a C-band Sentinel-1 SAR image before and after speckle noise filtering. When a SAR system illuminates the surface, scatterers within a SAR resolution cell (pixel) reflect energy with random phase shifts back to the sensor. The receiver superimposes all the individual returns within the resolution cell. Because of the heterogeneous distribution of scatterers from one pixel to the next, the amplitude and phase of the radar returns will experience random fluctuations from pixel to pixel. These random fluctuations cause the salt-and-pepper appearance of SAR images.

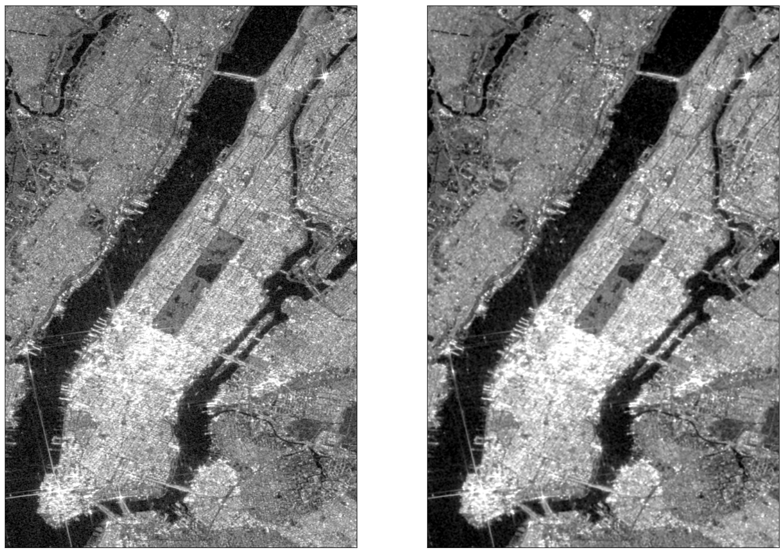


Figure 14: SAR image with speckle noise (left) and SAR image after Lee Filter with 7x7 window (right). Sentinel-1 GRD data processed by ASF [20] and post-processed with Python (right)

In order to understand the nature of speckle noise, let's consider a vegetated surface. The distribution of scatterers (e.g., trees, crops, etc.) is highly unlikely to be uniform throughout the distributed area and the variations from patch to patch induces an interference pattern in the image. Moreover, the side-looking geometry of the SAR system implies that the relative position of the instrument for each patch is different from pixel to pixel. This causes the non-deterministic fluctuations of amplitude and phase of the returned signals described above.

Speckle can also be understood using the vector diagram shown in Figure 15. Each vector in the diagram represents the return from the individual scatterers within a SAR pixel. The random amplitude and phase fluctuations of each vector results in the interference pattern causing the speckled appearance.

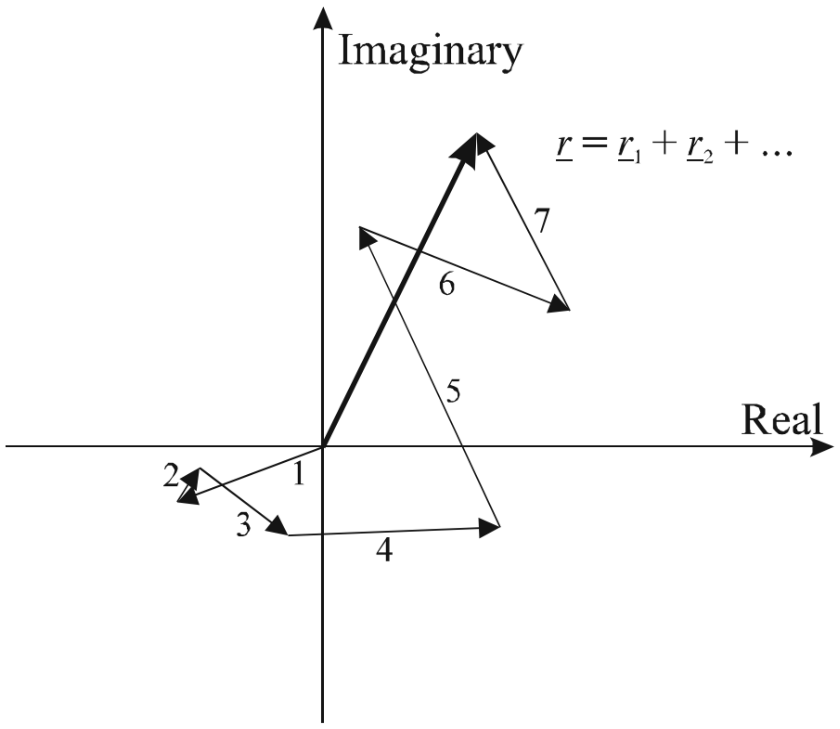

Imaginary
$\underline{r} = \underline{r}_1 + \underline{r}_2 + \ldots$
7
6
5
Real
2
1
3
4

Figure 15: Vector representation of speckle. Adapted from [11]

Speckle Distribution

For a more detailed treatment of the mathematics behind speckle, the reader is referred to [11]. The intensity distribution of a SAR acquisition can be modeled by an exponential distribution as outlined in Equation 30. This distribution is sometimes called the speckle distribution.

$$pdf(I|\sigma_0) = \frac{1}{\sigma_0}\, exp(-\frac{1}{\sigma_0}) \tag{30}$$

Note that the speckle distribution is a conditional distribution. That is, it depends on the normalized radar cross-section of the target. As the radar cross-section of the target increases, the speckle distribution starts to approach a uniform distribution. A larger radar cross-section results in the target appearing brighter in the radar image. Because of its dependence on radar cross-section and its multiplicative nature, speckle noise is difficult to deal with for practical applications. If we wanted to eliminate speckle noise completely from a radar image, the true radar cross section of the target must be known. Estimating the true radar cross section of targets is a difficult endeavor.

Dealing with Speckle with Multi-looking

Given the close resemblance of speckle noise to salt-and-pepper noise, we may be tempted to smooth out the image using spatial averaging. Median filters are typically used to smooth salt-and-pepper noise, for example [10]. However, in order to preserve the information contained in a SAR image, we must ensure that any averaging is done only over pixels that are of the same target. Otherwise, we could be averaging measurements from different targets and masking the true physical information contained in the radar return of each individual target. In other words, we would compute a false radar cross section measurement.

How can we deal with speckle noise? A technique used to reduce speckle entails making independent measurements of the distributed targets as we make the SAR acquisitions. As the SAR platform illuminates the targets, the azimuthal beam is split into L sub-beams similar to the aperture synthesis concept. However, we now have L smaller apertures to work with. Figure 16 shows this so-called multi-look geometry. Each sub-beam creates an image with a smaller azimuthal resolution relative to the full beam. This multi-look process basically makes L measurements of the *same* SAR pixel, and thus, ensures that we are averaging the radar cross-section over the same target area. The higher the number of looks over a given target, the lower the intensity variance within the pixel cell. However, this multi-look process comes at the expense of reduced spatial resolution.

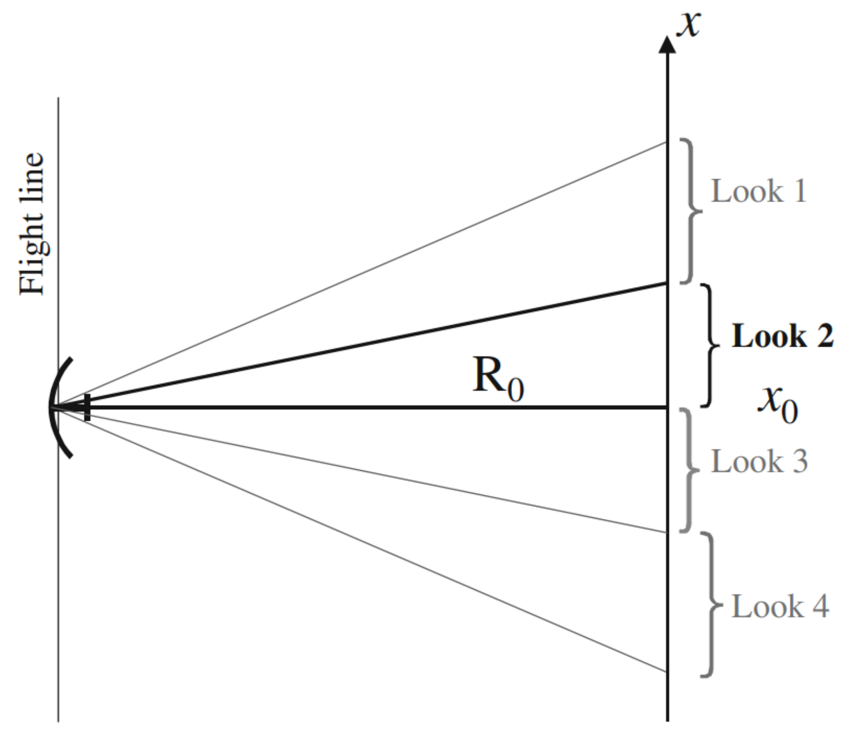


Figure 16: SAR multi-look geometry. Adapted from [11]

2.5.4 Radiometric Distortions

SAR images are also prone to radiometric or intensity distortions originating from surface topography. Recall that the side-looking geometry of a SAR platform induces a local incidence angle with the surface. This means that different topographical inclinations will be illuminated with different incidence angles. For example, inclined terrain facing the sensor will have an overexposed appearance relative to areas that reflect the incoming radar energy away from the sensor. This radiometric distortion can be corrected if the surface topography is known.

Consider the radar cross section, σ, of a pixel in a calibrated SAR image as described by Equation 31. The normalized radar cross-section, σ_0, and the surface area covered by a SAR pixel are expressed as a function of the local incidence angle, θ_i.

The process of radiometric terrain correction entails accounting for the pixel area dependence in Equation 31. If the pixel area is known, we can obtain an unbiased radar cross-section. A digital elevation model (DEM) is used to calculate the equivalent area A_σ covered by each SAR pixel and then normalizing the radar cross section by A_σ to extract a terrain normalized cross-section.

$$\sigma = \sigma_0(\theta_i) \cdot A_\sigma(\theta_i) \tag{31}$$

2.5.5 Geometric Distortions

Radiometric distortions are not the only impairments introduced by the side-looking geometry of SAR systems. Topographic features and surface terrain properties introduce geometric distortions as well. The most common types of geometric distortions are foreshortening, layover, and radar shadow as outlined in Figures 17 and 18.

Layover and Foreshortening

From Figure 17 we see that foreshortening results in the sensor-facing side of the mountain appearing compressed in the SAR image. This occurs because the radar signal arrives at the top of the mountain (point B) shortly after arriving at the base of the mountain (point A).

Layover is the extreme case of foreshortening when the radar signal arrives at the top of the mountain before the signal reaches the base of the mountain. In the SAR image, the mountain appears as if leaning towards the satellite.

Radar Shadow

Radar shadows are regions where there is no radar return. The lack of radar return is caused by targets such as mountains or buildings blocking the shadow regions behind them. Figure 18 also depict the radar shadow distortions.

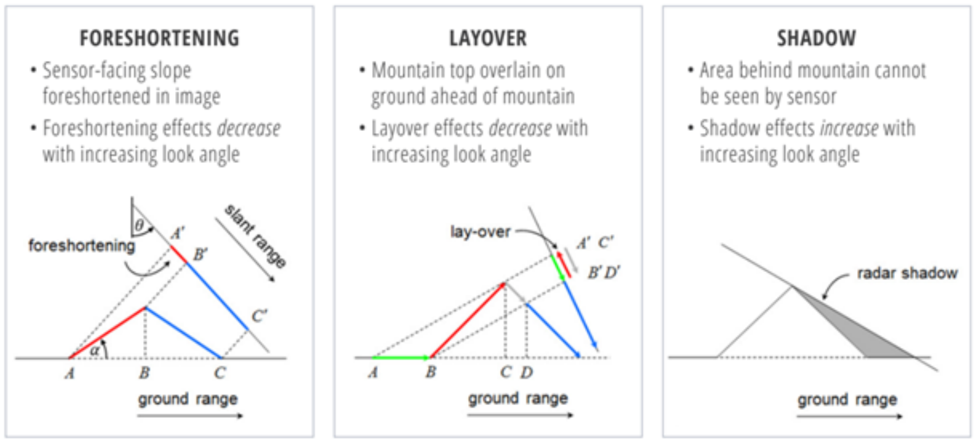

FORESHORTENING
• Sensor-facing slope foreshortened in image
• Foreshortening effects *decrease* with increasing look angle
foreshortening
slant range
θ
α
A'
B'
C'
A
B
C
ground range
LAYOVER
• Mountain top overlain on ground ahead of mountain
• Layover effects *decrease* with increasing look angle
lay-over
A' C'
B' D'
A
B
C D
ground range
SHADOW
• Area behind mountain cannot be seen by sensor
• Shadow effects *increase* with increasing look angle
radar shadow
ground range

Figure 17: Foreshortening, layover, and shadow geometric distortions in side-looking radar systems. Adapted from [32]

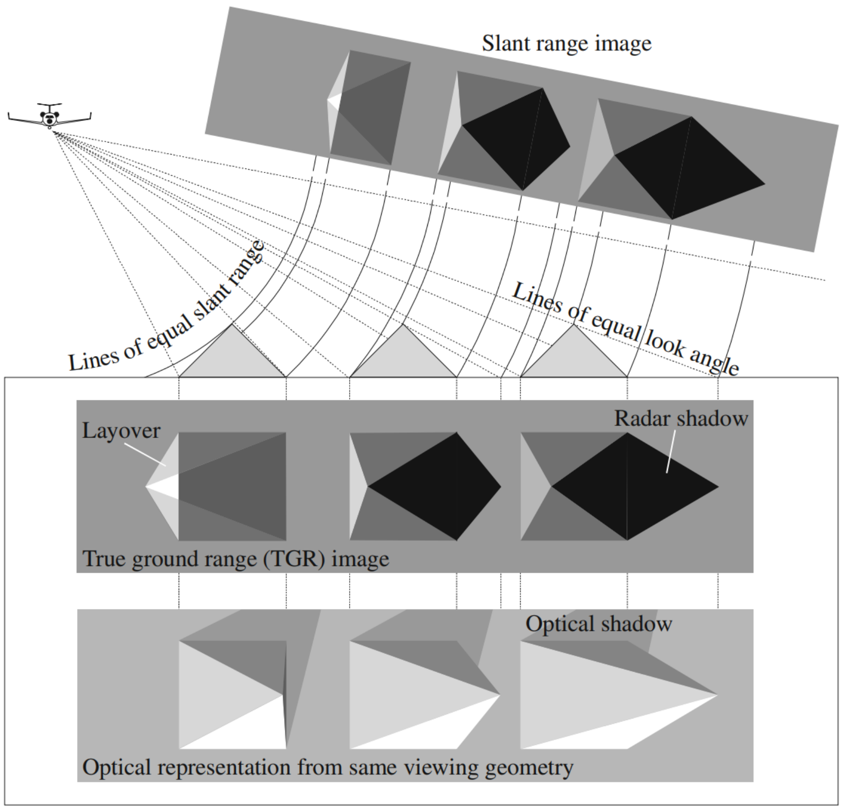

Slant range image
Lines of equal slant range
Lines of equal look angle
Layover
Radar shadow
True ground range (TGR) image
Optical shadow
Optical representation from same viewing geometry

Figure 18: Geometric distortions in side-looking radar systems. Adapted from [11]

2.5.6 Dielectric properties and penetration depth of radar signals

In Section 2.1, we discussed some of the physical fundamentals of electromagnetic radiation propagation. In this section, we revisit some of these fundamental topics from a SAR system perspective.

Recall that the radar cross-section of a target quantifies how much energy is scattered back from the target to the sensor. The radar cross-section depends on the target's properties such as the size, shape, dielectric properties, and its roughness.

Dielectric Properties

The dielectric properties of a scattering surface directly impact how much of the incident electromagnetic energy penetrates through the surface, how much energy is reflected back, and how much energy is absorbed by the medium. Equation 8 outlines the linear dependence between penetration depth and the radiation's wavelength. That is, longer wavelengths (e.g., L and C-band) penetrate deeper through surfaces than shorter wavelength radars (e.g., X-band). This implies that different SAR systems are more suitable for certain types of missions than others. For example, longer wavelengths (e.g., L and P band) are better suited to map inundation under tree canopies since they can penetrate deeper through vegetation than shorter wavelengths (e.g., X-band).

Figure 19 depicts penetration depth for X, C, and L-band system. We see that as the wavelength increases, the penetration depth also increases. Note, however, that the density of the medium and arrangement of the tree canopies or scattering surface also play a role in the penetration depth. Moreover, moisture content also impacts the dielectric properties of media as we will see in subsequent sections. As the moisture content increases, so does the relative permittivity discussed in Section 2.1, and thus, the penetration depth decreases.

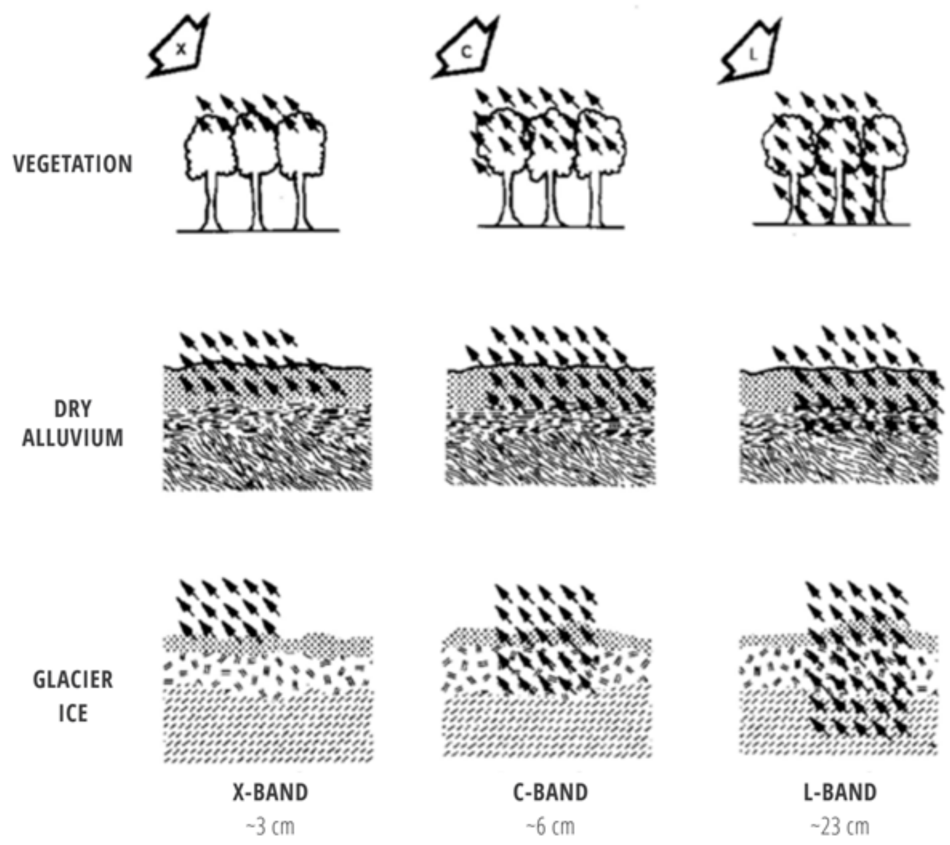


Figure 19: Scattering mechanisms by wavelength. Adapted from [32]

Surface Roughness

Surface roughness influences how much of the incident electromagnetic radiation is scattered back to the sensor. How rough a particular surface appears to a radar system depends on the size of the wavelength relative to local surface properties. Mathematically, we can quantify surface roughness as the standard deviation of the height deviation h from some mean height $\bar{h}$ of the surface. A surface is defined as rough if the height deviations exceed the value h_{rough}, as shown in Equation 32 [32].

$$h_{rough} > \lambda/(32 \cdot \cos\theta_i) \tag{32}$$

The implications of Equation 32 are that a surface considered rough at X-band ($\approx$ 3cm wavelength), may not be considered rough for C-band ($\approx$ 6cm wavelength). Figure 20 shows surface roughness from smooth to progressively rougher surfaces.

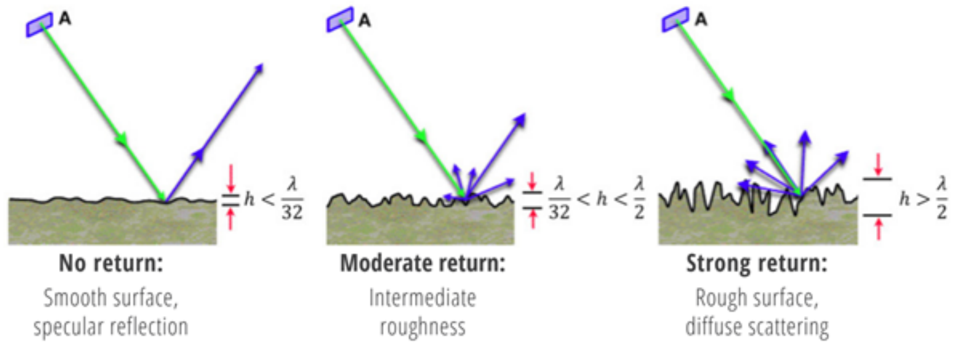

A
A
A
$h < \frac{\lambda}{32}$
$\frac{\lambda}{32} < h < \frac{\lambda}{2}$
$h > \frac{\lambda}{2}$
No return:
Smooth surface, specular reflection
Moderate return:
Intermediate roughness
Strong return:
Rough surface, diffuse scattering

Figure 20: EM waves reflection types. Adapted from [32]

Polarization

As described in Section 2.1, the polarization of electromagnetic radiation describes the orientation of the plane of oscillation of a propagating signal. The wave's polarization also influences the radar cross-section measured by the SAR system. Because SAR platforms are active microwave systems, the supported transmit and receive polarizations can be controlled. Most modern SAR systems are linearly polarized. In dual or quad-polarized systems, the transmitter alternates between transmitting H- and V-polarized pulses and receiving both H and V simultaneously.

2.5.7 Scattering Mechanisms

Different surface types or scatterer arrangements result in different scattering mechanisms as we will discuss next.

Specular reflection occurs when the incident signal is reflected away from the sensor in the so-called specular reflection. Specular reflection is caused by smooth surfaces as illustrated in Figure 20. In SAR images, specular reflection is characterized by dark regions as shown in zone 1 of Figure 22.

Rough surface scattering occurs when the incident signal is scattered in multiple directions as depicted in Figures 20 and 21. Rough surface scatterers are typically low-vegetation fields, bare soils, and roads. Zone 2 of Figure 22 shows an example of rough surface scattering characterized by mid-tones in the gray-scale SAR image.

Volume scattering occurs when the signal bounces multiple times through the target. Vegetation canopies are an example of volume scatterers as shown in zone 3 of Figure 22.

Double-bounce scattering occurs when the incoming radar signal reflects from smooth surfaces oriented at 90°. Double bounce scatterers include buildings, tree trunks, light poles and other vertical structures as shown by the red signal trajectories in Figure 21. Zone 4 of Figure 22 shows double-bounce scattering characterized by very bright intensities.

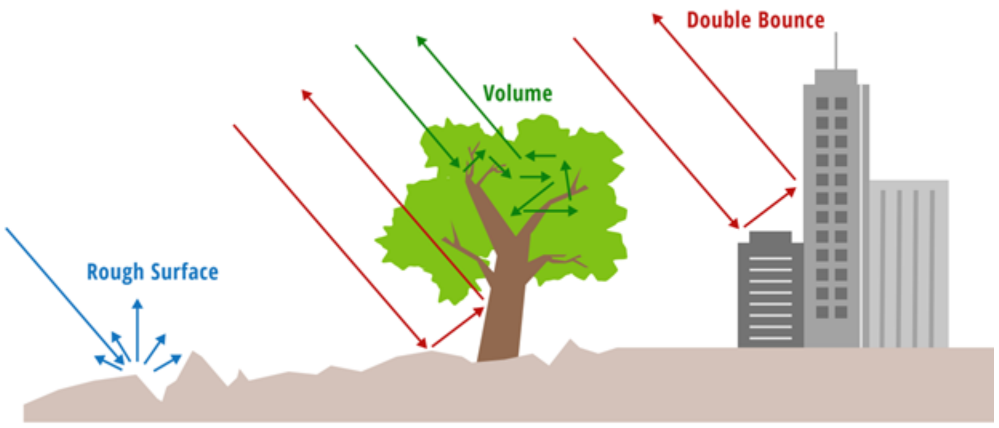


Figure 21: SAR scattering mechanisms. Adapted from [7]

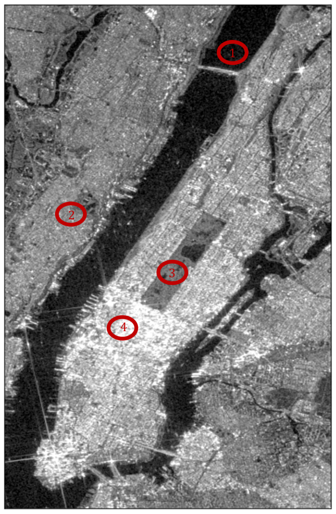


Figure 22: Scattering mechanisms on New York City Sentinel-1 image. Sentinel-1 GRD data processed by ASF [20]

2.6 Interferometric Synthetic Aperture Radar (InSAR)

The phase difference between two coherent signals is related to the path length difference to within a fraction of a wavelength. Thus, by measuring the phase difference between two radar return signals from the same target we can extract distance information. Interferometric synthetic aperture radar systems rely on this principle for applications such as detecting surface motion, measuring topography, studying land deformation, and monitoring volcanic and seismic activity.

In Section 2.5.3, we saw that the radar cross-section of a target within a SAR resolution cell is the superposition of all the echoes with random amplitude and phase components. This implies that the phase of a return signal can be decomposed into a range-dependent deterministic component and a random phase component. The goal of interferometry is to extract the range-dependent phase component and reject the stochastic component. We can express the measured phase as outlined in Equation 33, where ψ_{scatt} is the random phase component.

$$\psi = \psi(R) + \psi_{scatt} \tag{33}$$

InSAR platforms make two acquisitions and combine them to cancel out the random phase component. There are two common approaches for an InSAR system to make two measurements of the same target: *single-pass* interferometry and *repeat-pass* interferometry. In single-pass interferometry, a single SAR platform carries two antennas separated by a known baseline distance. However, carrying two antennas on a single platform is not always feasible or practical. In repeat-pass interferometry, a SAR platform collects two measurements during different overpasses of the instrument. The time difference between passes is known as the temporal baseline and can be as short as a few hours, to the span of months or years. The temporal baseline depends both on the SAR platform's orbital repeat cycle or the type of measurements being made. For slow-moving processes such as land deformation, observations over the span of months or years are typical to detect changes with interferometric approaches.

Figure 23 depicts the viewing geometry of a dual-system interferometry platform. Antenna 1 and Antenna 2 are separated by a baseline distance, B, and collect measurements of the same target area from different vantage points in space. In repeat-pass interferometry with a single antenna platform, there is an inherent spatial baseline pertaining to orbital variations between overpasses. Figure 24 shows the different ways to quantify the baseline separation between the two viewing positions. Typically, the perpendicular baseline, $B_{\perp}$ is used to define the viewing geometry.

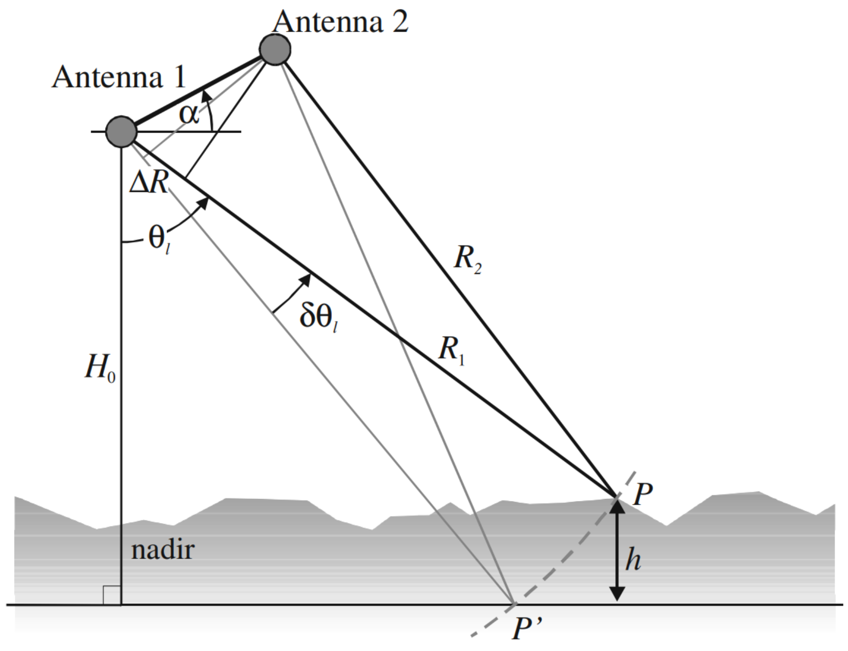


Figure 23: Geometry for across-track interferometry. Adapted from [11]

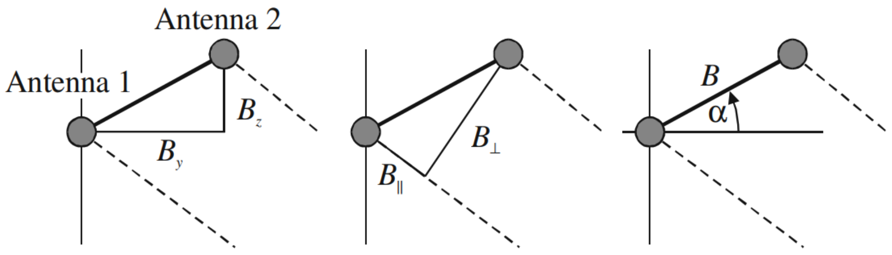


Figure 24: Baseline geometries for across-track interferometry - horizontal and vertical component (left), perpendicular and parallel component (middle), baseline angle and baseline length (right). Adapted from [11]

For a given look angle, θ_l, as shown in Figure 23, the phase difference measured between A_1 and A_2 is related to the path length difference as outlined in Equation 34. The factor of 2 accounts for the round-trip nature of the radar signal.

$$\Delta R = R_2 - R_1 = 2(R_2 - R_1) \;\; [m] \tag{34}$$

From Equation 34, the phase difference, $\delta\phi$, is computed as the fraction of wavelengths as outlined in Equation 35.

$$\delta\phi = 2\pi\frac{\Delta R}{\lambda} \;\; [rad] \tag{35}$$

From Figure 23, note that the slant ranges R_1 and R_2 are much larger that the baseline separation between the antennas ($B << R$). This magnitude differences allows us to approximate the path length difference with the parallel baseline distance depicted in the middle of Figure 35. Following this geometric train of thought, we can approximate the phase difference as shown in Equation 36.

$$\begin{aligned} \Delta R &\approx B_{\|} = B\sin(\alpha - \theta_l), \quad B << R \\ \delta\phi &= kB\sin(\alpha - \theta_l) \;\; [rad] \end{aligned} \tag{36}$$

Equation 36 expresses the interferometric phase difference as a function of B, k, α, θ_l. However, B, k, and α are constants, which means that the the absolute phase difference is only a function of the look angle, θ_l. That is, the interferometric phase difference provides information about the look angle of a signal. This additional knowledge of the pixel's look angle allows us to compute a true ground range and a height (altitude). This is one of the key applications of InSAR.

The InSAR measurements produce a phase interferogram. It is a plot of the relative phase differences for each SAR pixel. Figure 25 shows an example interferogram computed from two Sentinel-1 acquisitions over Sri-Lanka in 2017. The lines of equal color in the interferofram represent contours of equal look angle. However, given the cyclical nature of phase, there is an ambiguity in defining an absolute look direction. In order to remove the ambiguity, an interferogram that accounts for the reference surface alone is computed and subtracted from the measured interferogram. This calibration step is known as 'Flat-Earth' correction. Once the flat-earth component has been subtracted out, a measure of height above the reference surface is computed to extract topographic information.

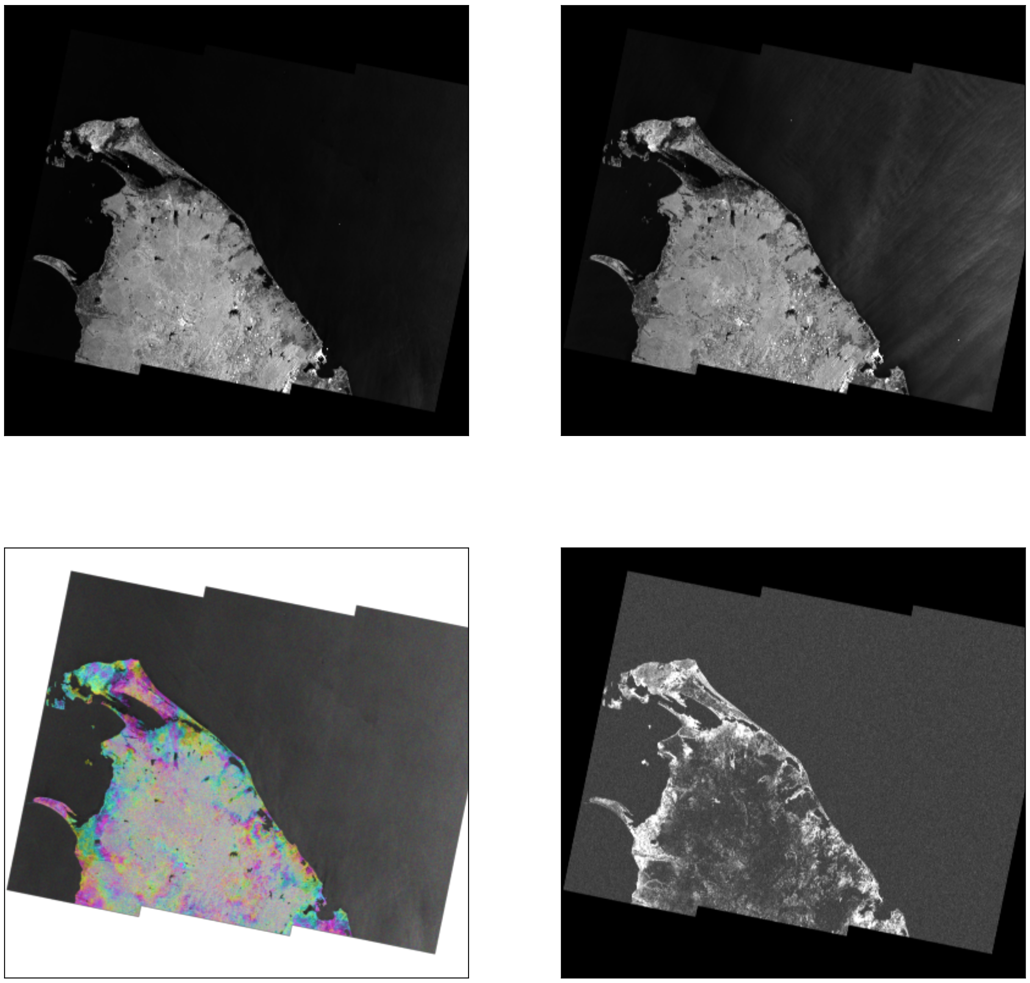


Figure 25: Interferogram over Sri-Lanka. Primary image date: May 12th (top-left), secondary image date (top-right): May 24th 2017, interferogram (bottom-left), and coherence map (bottom-right). Sentinel-1 data processed by ASF [20]

Interferometric Coherence Magnitude

In Section 2.1, we defined coherence as a measure of similarity between two waves. In other words, with knowledge of one wave, can we estimate the other? We also saw that interferometric coherence is calculated as a complex cross-correlation between two SAR acquisitions. The process of correlation ensures that only SAR pixels with valid radar cross-section measurements will be treated as confident measurements and reject noisy measurements. That is, pixels with low backscatter intensity will inherently yield low correlation values.

Equation 37 shows the normalized complex coherence between two SAR acquisitions. The brackets refer to a spatial average as opposed to a temporal average for the two SAR images. Similar to our discussion on multi-look filtering, the satellite platform has a limited time span over the image targets, and thus, cannot obtain many realizations of the same measurement with a single overpass. Thus, under an ergodic process assumption, we replace the time average with a spatial average over a small spatial window. The unerlying assumption is that imaged target properties remain constant. Equation 38 outlines the N-sample windows normalized coherence, where u_1 and u_2 are the single-look complex pixel values and N is the number of pixels.

$$\gamma = \frac{< E(t_1)E(t_2)^* >}{\sqrt{< |E(t_1)|^2 > \cdot < |E(t_2)|^2 >}} \tag{37}$$

$$|\gamma[i,k]| = \frac{|\sum_N u_1[i,k] \cdot u_2^*[i,k]|}{\sqrt{\sum_N |u_1[i,k]|^2 \cdot \sum_N |u_2[i,k]|^2}} \tag{38}$$

Note that γ is a complex number. The phase angle of γ gives us the phase difference between acquisitions, whereas the magnitude, $|\gamma|$, gives us a measure of 'meaningfulness' of the measurement. Coherence values range from 0 to 1. In other words, if the coherence magnitude equals 1, the averaged pixels are fully coherent. On the flip side, a coherence magnitude of 0 represents uncorrelated pixels. Coherence measurements are at the heart of our study and will be the topic of subsequent sections.

2.6.1 Decorrelation

What happens when the target surface changes between InSAR acquisitions? Our InSAR measurements decorrelate. That is, they become less similar. System noise is the first source of decorrelation since it is intrinsic to the SAR platform and not the imaged targets. Given its random nature, noise will not be correlated between SAR acquisitions, and it is therefore an impairment that cannot be avoided. Noise decorrelation is directly proportional to the signal to noise ratio. In areas of low backscatter energy, decorrelation will be greater. For example, areas of radar shadow lack cross-section measurements, and thus, the coherence magnitude will approach zero. This yields any phase difference calculations over these areas meaningless. Similarly, other targets with low signal measurements (e.g., specular reflection) are hindered by SNR and will exhibit noise decorrelation.

Baseline decorrelation occurs when the baseline between acquisitions starts to increase. For non-zero baseline geometries, as the baseline separation increases, the return signals become less similar and start to decorrelate.

Temporal decorrelation occurs when the targets change between repeat passes. If the targets are not changing much with time, the coherence will remain high. Urban environments exhibit high temporal correlation values, for example. On the flip side, vegetated terrain exhibits low temporal correlation values due to the random motion of leaves, crop and vegetation growth, etc.

In remote sensing applications, our goal is very often to detect changes in the imaged landscapes. Decorrelation is therefore a key enabler for change detection. Natural processes such as rain and wind affect the environment between satellite passes. Anthropogenic activity is another source of potential landscape change. In all of these cases, if the physical changes are larger than the size of the wavelengths used, they will impact the coherence measurements.

We can exploit the coherence changes over time to monitor changes on the surface.

2.6.2 InSAR Limitations

As summarized in Section 2.6, InSAR techniques have many useful remote sensing applications. However, InSAR techniques run into a number of limitations that make their use and applications challenging.

The main source of limitations for InSAR is phase decorrelation. Shorter wavelength exhibit more rapid decorrelation than longer wavelengths. This means that we need to consider the platform's design when applying InSAR techniques. Another major limitation of InSAR techniques is the relatively low coherence values in highly vegetated areas. Volume scatterers tend to reflect signals in multiple directions and are more likely to depolarize and decorrelate the radar signals. Future SAR missions, like NISAR [7] [37] will operate at L-band which is more robust to decorrelation over vegetated terrain due to the longer wavelengths.

Temporal phase decorrelation is another limitation of InSAR techniques. As the temporal baseline increases, the more prone an area is to experience change, and thus, decorrelate between acquisitions. Temporal decorrelation makes studying slow-changing phenonema more challenging. Surface deformation is an example of a typically slow-moving process that is noticeable only over extended periods of time (e.g., months or years).

Temporal decorrelation can also be caused by surface disturbances cause by weather events. Rain water temporarily changes the soil moisture content of the terrain and leads to temporal decorrelation. Likewise, flooding events change the structure of the target terrain resulting in temporal decorrelation.

Lastly, atmospheric distortions are another source of decorrelation. The vacuum to atmosphere boundary imposes a change in refractive index which results in a phase delay in the signal. This phase delay can distort our interferometric measurements.

2.7 Sentinel-1 Platform

This section provides an overview of the Copernicus Sentinel-1 SAR platform adapted from [12] [38] and [39].

The Sentinel-1 mission operated by the European Space Agency (ESA) is comprised of twin synthetic aperture radar satellites (Sentinel-1A and Sentinel-1B) orbiting the earth 180° apart. The SAR mission generates products that are openly accessible to the public. Each of the twin satellites carries a C-band SAR sensor with dual-polarization capabilities (VV+VH, HH+HV, HH, VV). The Sentinel-1 satellites have a 12-day repeat cycle at Equator which brings the revisit times down to 6 days between passes for a given scene or region on earth.

The Sentinel-1 mission acquires earth-observation products in a few different modes as outlined in Figure 26 and summarized in Figure 27. Of particular importance to our study are the Single Look Complex (SLC) products and Ground Range Detected Products (GRD) from the Sentinel-1 mission. The SLC products preserve the phase information and enable interferometry calculations including coherence estimations. The GRD products enable accurate geo-referenced radar cross-section measurements in the form of Sentinel-1 intensity images. For a more detailed treatment of the Sentinel-1 products, the reader is referred to [12] and [39].

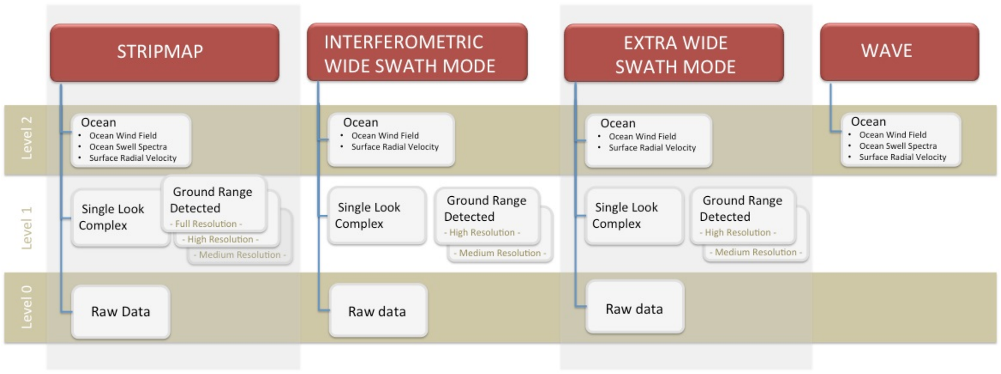
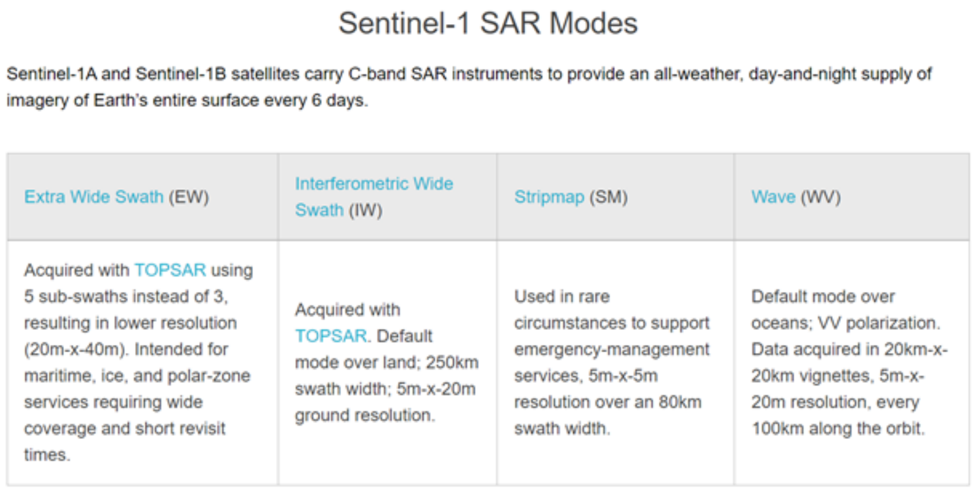


Figure 26: Copernicus Sentinel-1 core products. Adapted from [38]



Sentinel-1 SAR Modes

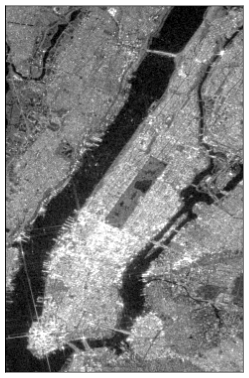
Sentinel-1A and Sentinel-1B satellites carry C-band SAR instruments to provide an all-weather, day-and-night supply of imagery of Earth's entire surface every 6 days.

Extra Wide Swath (EW)	Interferometric Wide Swath (IW)	Stripmap (SM)	Wave (WV)
Acquired with TOPSAR using 5 sub-swaths instead of 3, resulting in lower resolution (20m-x-40m). Intended for maritime, ice, and polar-zone services requiring wide coverage and short revisit times.	Acquired with TOPSAR. Default mode over land; 250km swath width; 5m-x-20m ground resolution.	Used in rare circumstances to support emergency-management services, 5m-x-5m resolution over an 80km swath width.	Default mode over oceans; VV polarization. Data acquired in 20km-x-20km vignettes, 5m-x-20m resolution, every 100km along the orbit.

Figure 27: Sentinel-1 acquisition modes. Adapted from [39]

2.8 Surface Water Mapping with SAR Intensity

Synthetic aperture radar intensity data is useful for surface water mapping. Standing water bodies reflect the incoming radar signals away from the sensor resulting in low backscattered energy. The result is a SAR image with a bi-modal intensity distribution. Traditional image processing techniques can then be used to segment the water pixels. Figure 28 shows our New York City Sentinel-1 example along with its intensity distribution. Note that the histogram exhibits a bi-modal distribution that can be easily segmented to extract the water pixels in the image. Next, we consider three flooding scenarios and discuss the SAR backscatter signature associated with each scenario. The discussion that follows is adapted from [7] and [35].



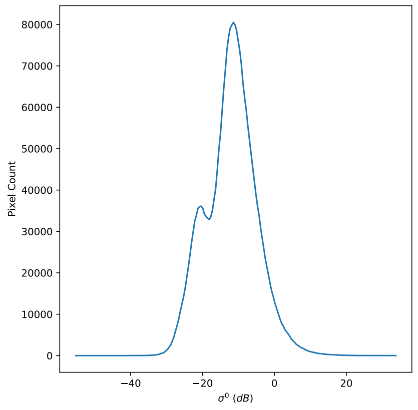


Figure 28: Sentinel-1 bi-modal distribution. Sentinel-1 GRD data processed by ASF [20]

Open Lands

The first environment we consider are open lands. Figure 29 depicts the relative soil moisture as a function of time on the left-hand plot and the radar cross-section as a function of time on the right-hand plot. As precipitation increases with time, the soil moisture increases monotonically until it reaches a saturation point. Once the soil is saturated, flooding occurs and water starts to accumulate over the surface. On the right-hand plot, we see that the radar cross-section or intensity initially increases in tandem with the increase in relative soil moisture. The increase in radar cross-section is due to the temporary increase in electrical conductivity associated with the moist soil. However, as the soil saturates and water starts to accumulate, we see a sudden drop in the radar cross section attributed to the specular reflection of the microwave signals.

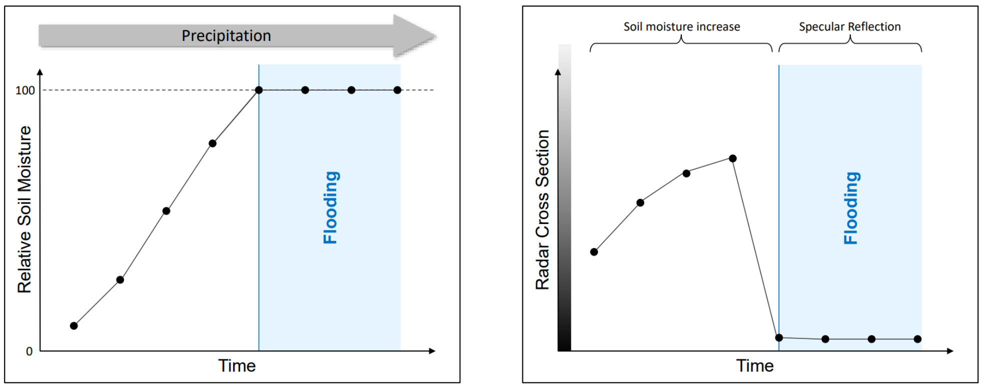


Figure 29: Surface water mapping in open lands. Adapted from [7]

Flooding under Vegetation Canopies

The second scenario is flooding under vegetation canopies as shown in Figure 30. The left-hand diagram shows dry terrain before the onset of rain. We get the typical volume scattering from tree canopies and rough surface scattering from dry soil. As water starts to accumulate, double-bounce scattering from the water-trunk boundary starts to dominate the backscattered energy.

Correlating our pictorial observations to Figure 31, we see a monotonic increase in soil moisture up to a saturation point when flooding occurs. The radar intensity also increases monotonically due to the change in electrical properties in the soil. For this scenario, however, when the double bounce scattering between the water-tree boundary starts to occur, the radar return is enhanced and appears very bright in our SAR image.

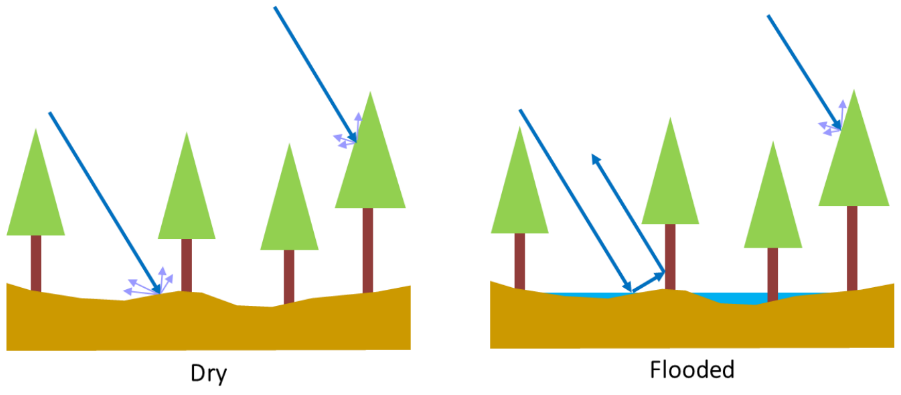


Figure 30: Surface water mapping under vegetation canopies. Adapted from [7]

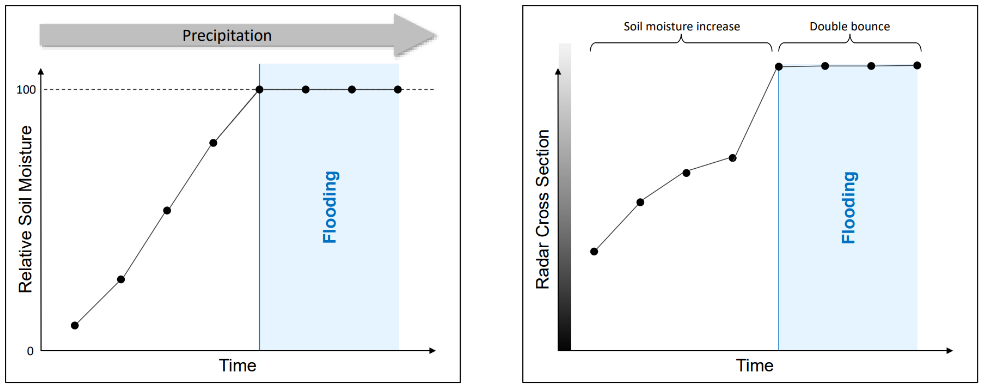


Figure 31: Surface water mapping under vegetation canopies. Adapted from [7]

Flooding in Crop Lands

The last scenario is flooding in crop lands. Figure 32 shows a progression of events as rain starts to flood crop lands. As shown in diagram B, as rain starts to accumulate but not completely submerge the vegetation, we get a double bounce effect that improves the radar backscatter intensity. As the flooding onsets and the crops are submerged, specular reflection dominates and the radar cross section drops as shown in Figure 33.

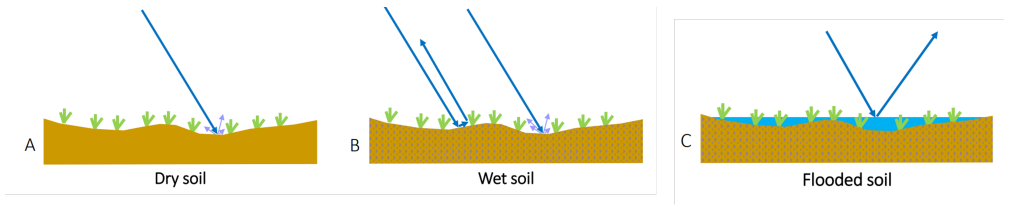


Figure 32: Surface water mapping in crop lands. Adapted from [7]

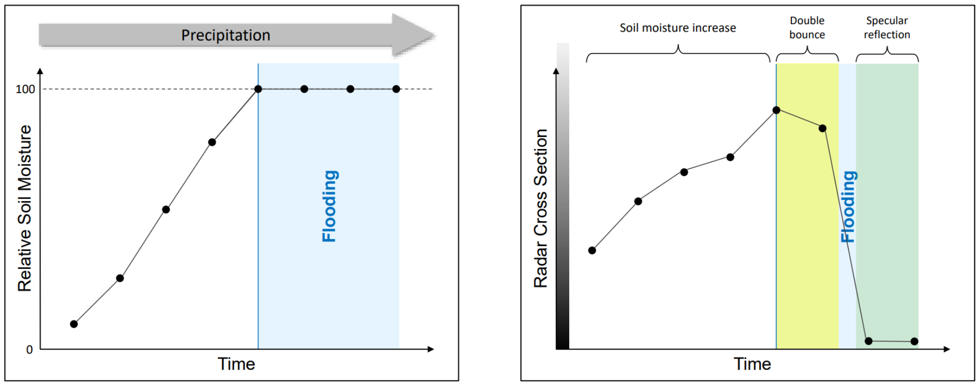


Figure 33: Surface water mapping in crop lands. Adapted from [7]

3 Data Processing and Preparation

This section describes the data set used in our study. Section 3.1 provides an overview of the Sen1Floods11 public data set used as the backbone for our experiments. Section 3.2 describes our methodology for augmenting the Sen1Floods11 data set with bi-temporal intensity and interferometry Sentinel-1 data.

3.1 Sen1Floods11 Data set

Sen1Floods11 is a georeferenced surface water data set for eleven flood events generated by Cloud to Street, a Public Benefit Corporation [40]. The flood events occurred between 2016 and 2019 and are spread around the world. The Sen1Floods11 data set contains classified permanent water, flood water, and Sentinel-1 backscatter intensity data (VV- and VH-polarized at 10-meter spatial resolution). In total, the data set is comprised of 4,831 chips (512 x 512) with a 446 chip subset quality controlled and hand-labeled with surface water from the flood events. The remaining 4,385 chips are labeled with two threshold-based methods and are considered weakly-labeled. The first weak label modality is extracted from the Sentinel-1 intensity imagery by applying an Otsu variance maximization threshold in the VV-band. The second weak label modality is extracted by thresholding the Sentinel-2-derived Normalized Difference Vegetation Index (NDVI) and the Modified Normalized Difference Water Index (MNDWI) [1] [30].

3.2 Data Set Augmentation

The Sen1Floods11 data set described in Section 3.1 serves as the backbone for our study's data set. We consider a subset of seven of the flood events included in the Sen1Floods11 data set. Figure 34 outlines the geographical distribution of the flood events considered in this study while Table 1 summarizes the geographical regions along with pertinent metadata.

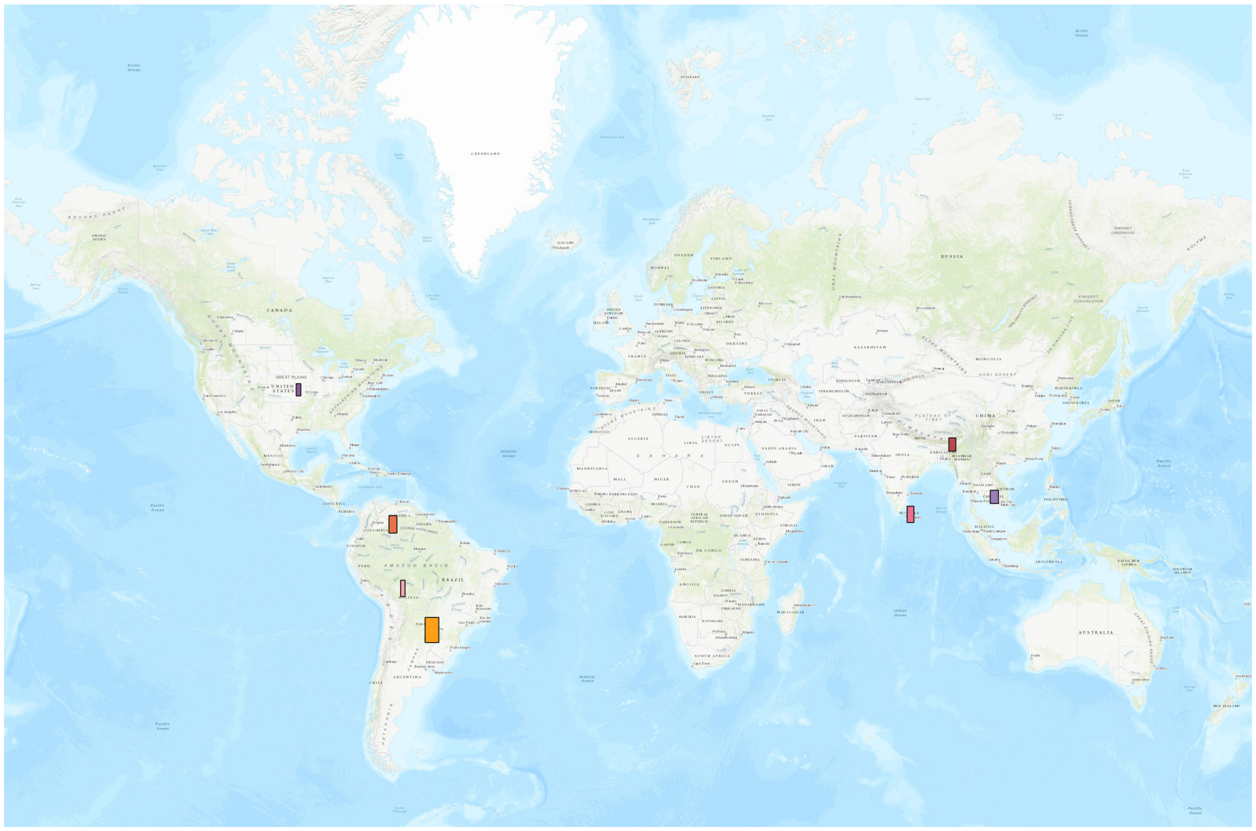


Figure 34: Geographical regions from where flood data was sampled [1]

Country	S2 date	S1 Co-event date	S1 Pre-event date	Rel. Orbit	Orbit
USA	2019-05-22	2019-05-22	2019-04-16	136	Ascending
KHM	2018-08-04	2018-08-05	2018-07-24	26	Ascending
BOL	2018-02-15	2018-02-15	2018-02-03	156	Descending
IND	2016-08-12	2016-08-12	2016-06-01	77	Descending
PRY	2018-10-31	2018-10-31	2018-10-19	68	Ascending
COL	2018-08-23	2018-08-22	2018-07-17	106	Ascending
LKA	2017-05-28	2017-05-39	2017-05-12	19	Descending

Table 1: Flood event metadata for regions used in this study [1]

Figure 35 shows six different Sen1Floods11 scenes sampled across different regions. The left-most column shows the true color Sentinel-2 scene followed by the VV-polarized Sentinel-1 co-event intensity. The false color composite shown in the figure combines the VV- and VH-polarized channels to highlight the water bodies in dark shades of blue similar to the approach used in [41]. Lastly, hand-labels for each scene are shown on the right-most column.

The Sen1Floods11 imagery provides the co-event intensity chips used to train our models. We augment the co-event intensity data by downloading pre-event intensity chips from Google Earth Engine [14] overlapping the spatial extent of the co-event chips. Specifically, Sentinel-1 Interferometric Wide (IW) Swath Ground Range Detected (GRD) prodcuts are downloaded to augment the co-event intensity chips. The Level-1 GRD products available from Google Earth Engine are processed to backscatter coefficients (σ_0) in decibels (dB). Google Earth Engine uses the Sentinel-1 Toolbox [42] to carry the sequence of processing steps outlined in Algorithm 1 to produce GRD products [43]. Table 1 summarizes the pre-event and co-event Sentinel-1 scene dates used in our experiments and Figure 36 shows an example of a pre- and co-event Sentinel-1 intensity chip at 10-meter resolution.

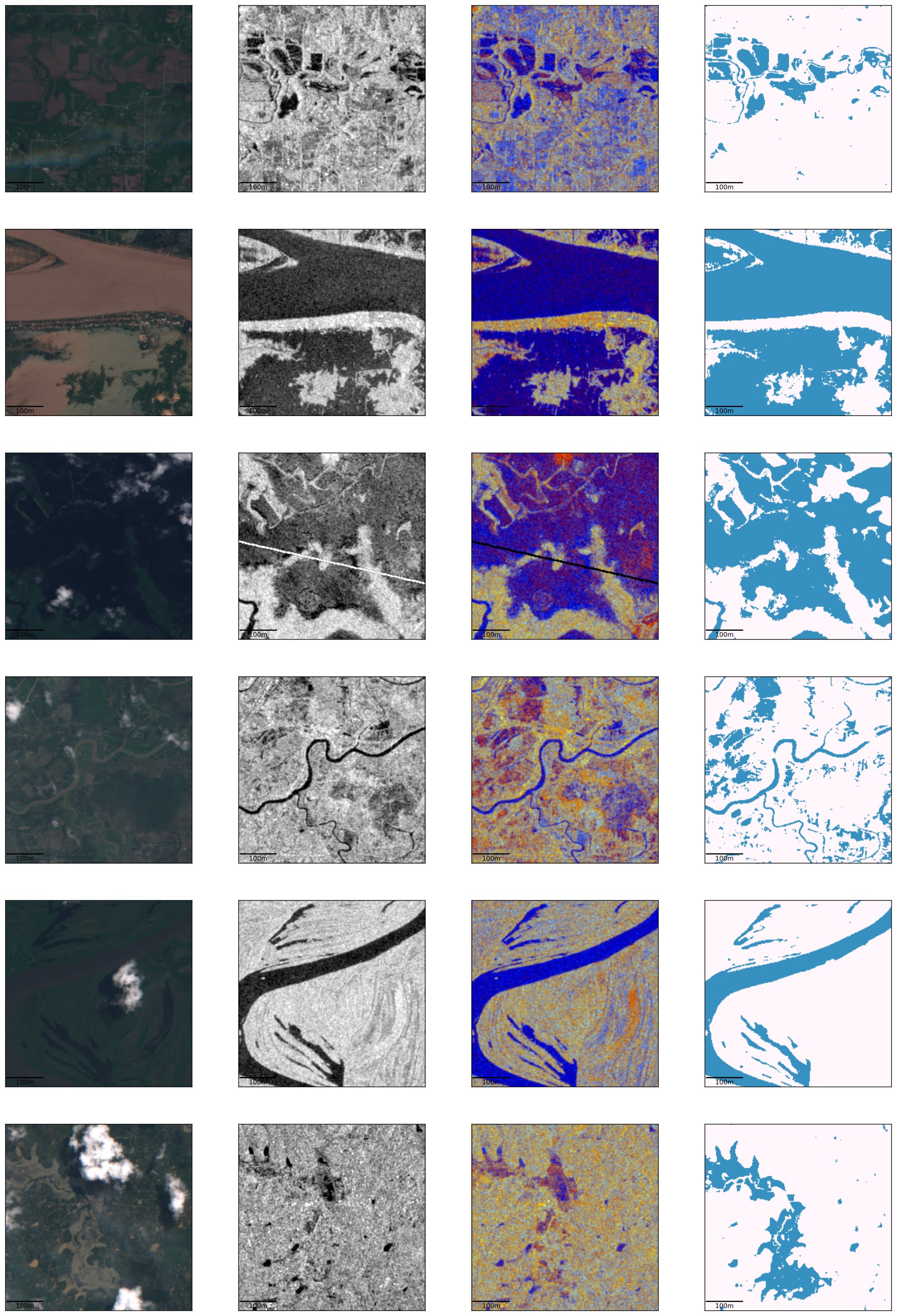


Figure 35: Sentinel-2 true color composite, Sentinel-1 (VV) intensity, Sentinel-1 false color composite (R: VV , G: VH, B: VV / VH), Hand-label. Sentinel-1 and Sentinel-2 data processed by GEE [14] and post-processed with Python

Algorithm 1 Google Earth Engine Sentinel-1 GRD Processing [43]

1. Apply orbit file to update orbit data

2. GRD border noise removal to remove low intensity noise and invalid data on the scenes

3. Thermal noise removal to mitigate additive noise in the merging of Sentinel-1 sub-swaths

4. Radiometric calibration to compute backscatter intensity using sensor calibration parameters

5. Terrain correction (orthorectification) to convert data from ground range geometry to geocoded σ_0 using a DEM (e.g., SRTM 30 meter or ASTER DEM)

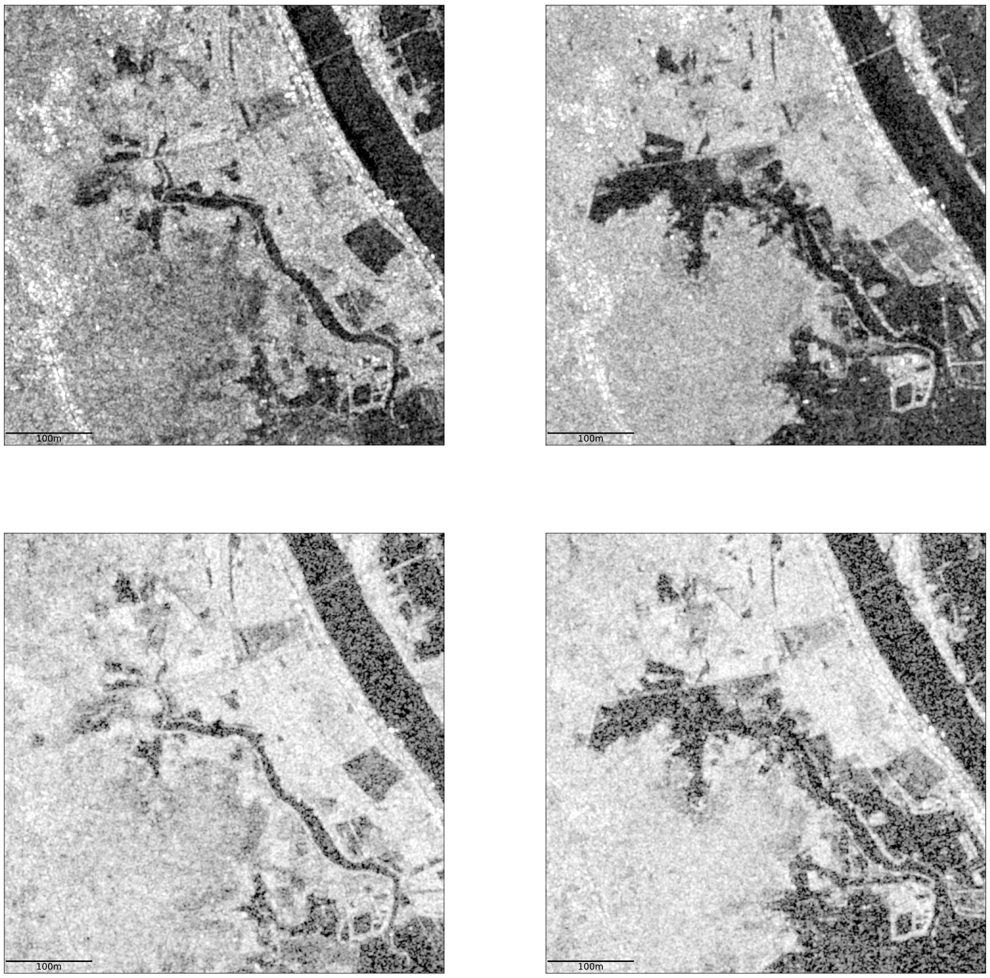


Figure 36: Pre-event intensity (left) and co-event intensity (right) pairs (VV-polarized top, VH-polarized bottom) [30]. Sentinel-1 data processed by GEE [14].

In addition to augmenting the co-event intensity data, interferometric coherence data is generated for both the pre- and co-event scenarios. As described in Section 2.6, interferometry requires a co-registered pair of single look complex (SLC) Sentinel-1 images. In order to augment the Sen1Floods11 data set, we identify suitable pre- and co-event image pairs using Alaska Satellite Facility's Vertex Data Search platform [20]. Table 2 outlines the temporal, B_t, and perpendicular, B_p baselines for the image pairs used to generate the coherence maps.

The interferometric coherence maps were generated using the on-demand HyP3 platform hosted by the Alaska Satellite Facility (ASF) [44] [20]. HyP3 offers on-demand InSAR products processed using the GAMMA Remote Sensing software [45] at no cost. The InSAR products are offered at both 40-meter and 80-meter resolutions. The resolution is set based on the number of multi-looks used in the HyP3 processing pipeline. The processing steps used to generate the InSAR products are summarized in Algorithm 2.

Country	**Pre-event InSAR** B_t **(days)**	**Pre-event InSAR** B_p **Range (m)**	**Co-event InSAR** B_t **(days)**	**Co-event InSAR** B_p **Range (m)**
USA	12	5 - 6	12	21 - 22
KHM	12	114 - 117	12	37 - 38
BOL	12	24 - 29	12	4 - 6
IND	24	7 - 11	24	9 - 14
PRY	12	21 - 32	12	38 - 40
COL	12	46 - 50	12	0.2 - 1.1
LKA	12	76 - 80	6	71 - 80

Table 2: Temporal and perpendicular baselines for InSAR Products. Data retrieved from ASF [20]

Algorithm 2 InSAR Processing Pipeline [20]

```
    Pre-Processing
 1: Ingest SLC data into GAMMA format
 2: Get DEM file covering the area, apply geoid correction and resample/reproject as re-
    quired
 3: Calculate overlapping bursts for input scenes
 4: Mosaic swaths and bursts together
    InSAR Processing
 5: Prepare initial look-up table and simulated unwrapped image
    - Derive lookup table for SLC/MLI co-registration (considering terrain heights)
    - Simulate unwrapped interferometric phase using DEM height, and deformation rate
    using orbit state vectors
 6: Interferogram creation, matching, refinement
    - Initial co-registration with look-up table
    - Iterative co-registration with look-up table
    - Removal of curved earth and topographic phase
 7: Determine a co-registration offset based on the burst overlap (spectral diversity method)
    - Single iteration co-registration with look-up table
 8: Phase unwrapping and coherence map generation
 9: Generation of displacement maps from unwrapped differential phase
    Post-Processing
10: Generation of geocoded GeoTIFF outputs
```

For this study, we are interested in as high a spatial resolution as possible. Thus, 40-meter coherence maps were ordered from the HyP3 platform for each of the geographical regions in our study. The 40-meter coherence maps were re-sampled to 10-meter resolution using a nearest neighbor interpolation algorithm. The Python bindings for GDAL were used for re-sampling [46]. Since coherence data represents phase noise, nearest neighbor interpolation proved to be the most appropriate re-sampling algorithm for our experiments. Any other non-linear re-sampling algorithm tends to generate smoothed versions of the coherence maps and thereby reduces the spatial context needed to extract meaningful information from the coherence maps.

Figure 37 displays a pre- and co-event intensity and coherence pair from our augmented data set. Coherence values range from 0 to 1, with darker patches in the coherence map corresponding to more incoherent regions relative to the brighter patches. Note that the co-event coherence map exhibits a loss of coherence in the regions where flood water is present. In our case, the loss of coherence can be attributed to the spatial modification induced by the flood water relative to the pre-event benchmark. When flood water stagnates, there is a loss of coherence where the water accumulates, which correlates with what we see from Figure 37. The reference Sentinel-2 weak label for the flood scene shown in Figure 37 is shown in Figure 38. From the water mask, we confirm that the areas corresponding to lower coherence values in the co-event map are indeed caused by the presence of flood water.

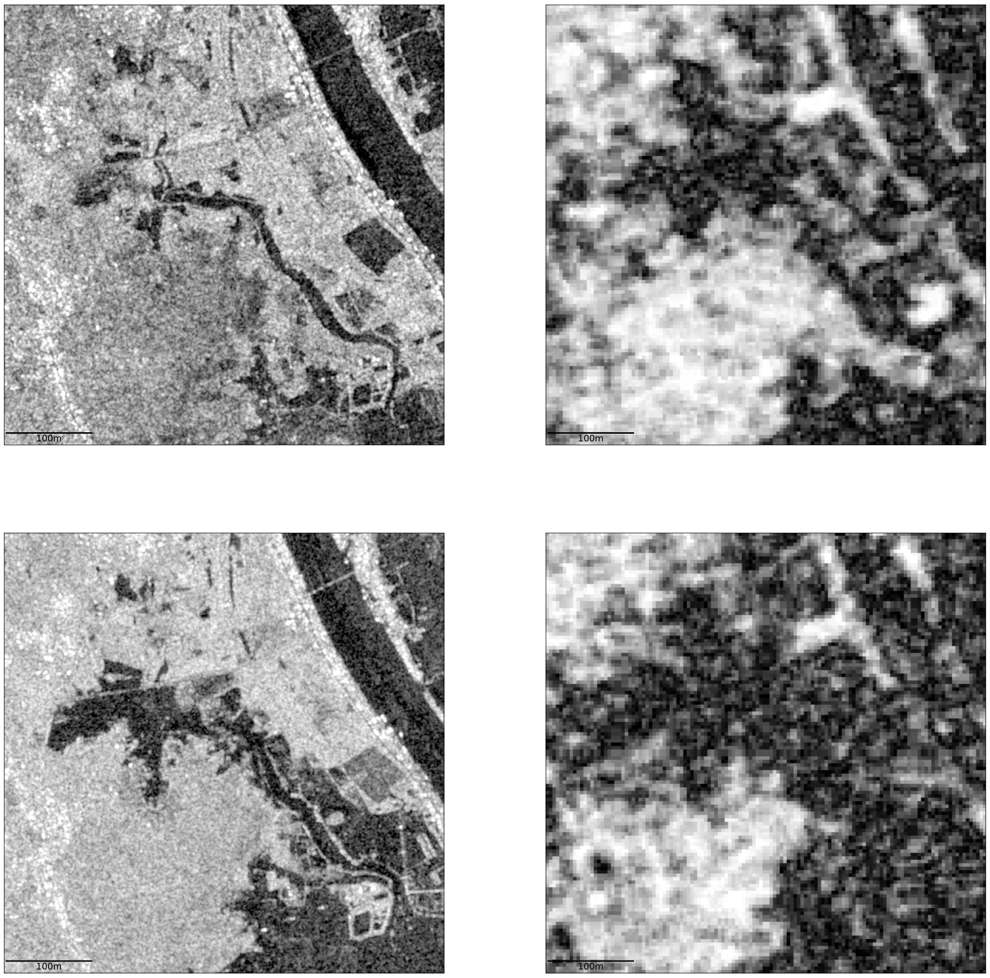


Figure 37: Pre-event intensity (VV) and coherence (top), co-event intensity (VV) and coherence (bottom). Data processed by GEE and ASF [14] [20].

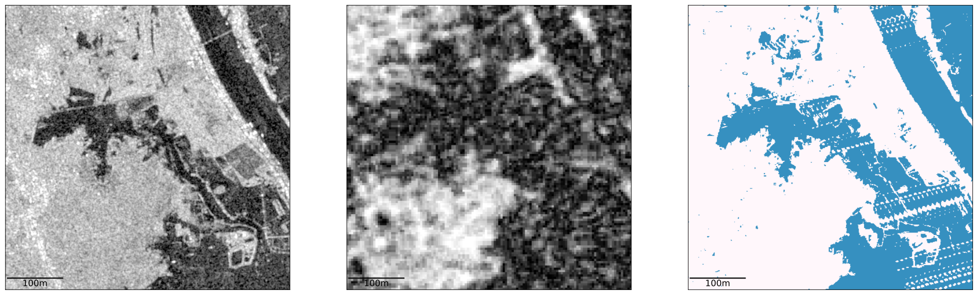


Figure 38: Co-event intensity (VV) (left), co-event coherence (middle), Sentinel-2 weak label (right). Data processed by GEE and ASF, and post-processed with Python [14] [20].

Figure 39 shows a collection of bi-temporal intensity and coherence pairs from our augmented data set. A visual inspection of the co-event intensity chips quickly highlights the presence of flood water as the darker patches in the SAR image. We note that most of the dark patches in the co-event intensity images are not present in the pre-event scenes. The false color composites shown in the image further highlight the flooded areas in cyan tones. The cyan regions represent areas where the co-event intensity is lower than the pre-event intensity. Lastly, we observe a drop in coherence for each scene.

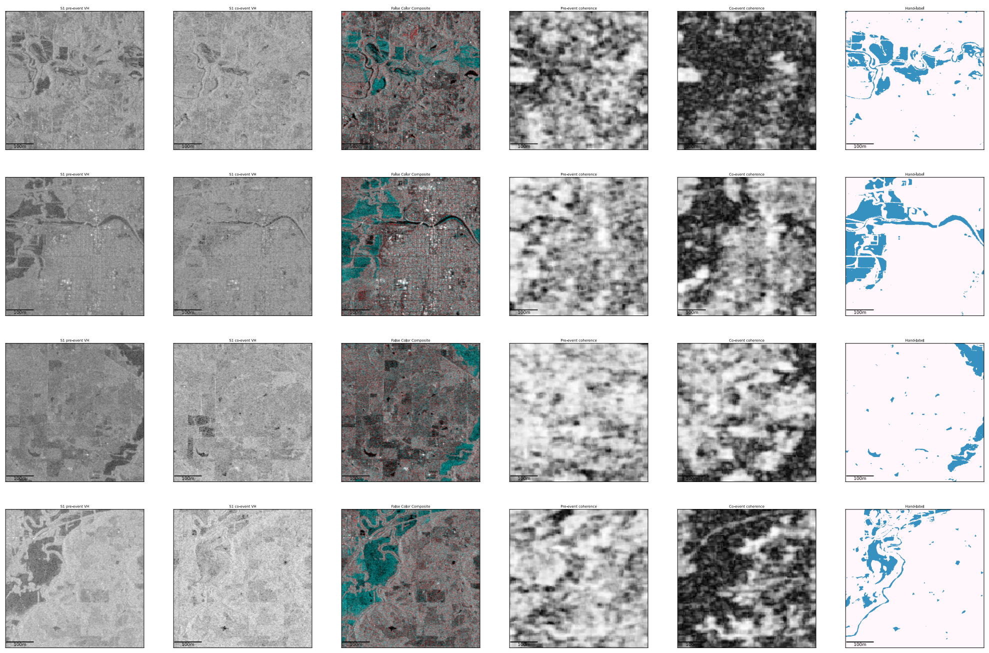


Figure 39: From left to right: co-event intensity (VH), pre-event intensity (VH), false color composite (R: co-event, G: pre-event, B: pre-event, adapted from [13]), pre-event coherence, co-event coherence, hand-label. Sentinel data processed by GEE and ASF and post-processed with Python [14] [20].

4 Methods

This section describes the machine learning models trained for our semantic water segmentation experiments. We note that the models trained were not designed to optimize absolute classification performance relative to some state of the art benchmark. Instead, the experiments were set up to assess the relative improvement gained by fusing intensity and interferometric coherence data when compared against an intensity data-only model. Section 4.1 describes the different scenarios or input data combinations we train. Section 4.2 describes the XGBoost, U-Net, and Attention U-Net architectures. Lastly, Section 4.3 concludes with the training methodology used for each model.

4.1 Model Scenarios

We train three different scenarios with different data combinations for flood water classification. Table 3 outlines the different data combinations along with the number of bands or channels (e.g., co-event VV-polarization, co-event VH-polarization, etc.) for each scenario. These scenarios are similar to three of the scenarios trained by [13] where the authors use TerraSAR-X data and use Hurricane Harvey as a test case. Note that the Sentinel-1 intensity chips contain both VV- and VH-polarized GRD products for both the pre- and co-event data sets. Figures 40, 41, and 42 depict block diagrams for each scenario.

Scenario	**Description**	**Number of channels**
Scenario I	Co-event intensity	2
Scenario II	Pre-event intensity + Co-event Intensity	4
Scenario III	Pre-event intensity and coherence + Co-event intensity and coherence	6

Table 3: Experiment scenarios

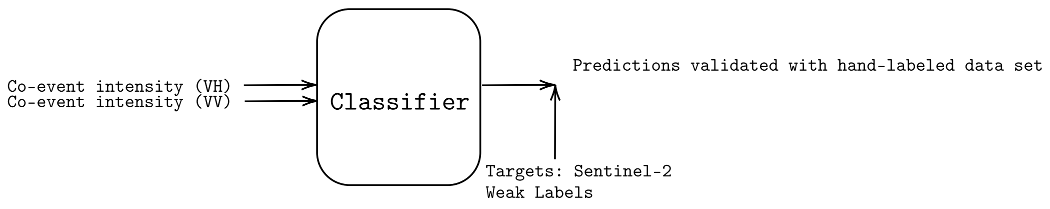


Figure 40: Scenario 1: Uni-temporal intensity scenario block diagram

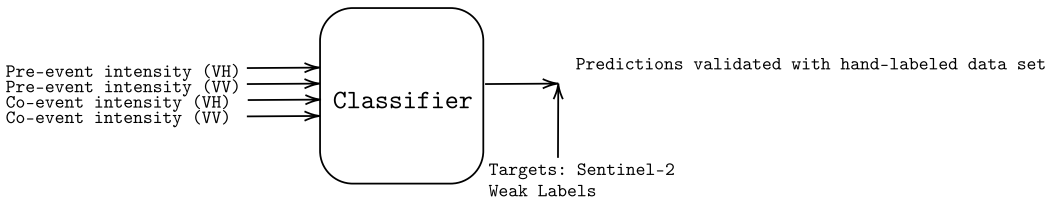


Figure 41: Scenario 2: Bi-temporal intensity scenario block diagram

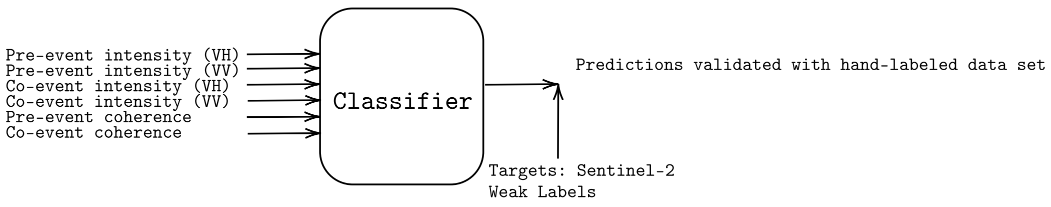


Figure 42: Scenario 3: Bi-temporal intensity and coherence scenario block diagram

4.2 Models

4.2.1 XGBoost

Ensemble Algorithms and Boosting

Ensemble algorithms build a collection of individual statistical predictors or weak learners. The collection of weak learners' predictions are aggregated together to make a final prediction. For classification algorithms, the final prediction can be the majority vote amongst all of the weak learners. Each of the weak learners in the ensemble can be an individual classification tree, for example. These weak learners typically work on a subset of the feature space and are not very good predictors by themselves. The premise behind limiting the feature space is to minimize the potential for overfitting to strong features. For example, if we create an ensemble of decision trees where all of the individual constituent trees are allowed to use all of the features in the data set, any strong predictor in the feature space will dominate and bias the predictions for the entire ensemble [47] [48].

Boosting is a technique that incrementally improves the classification or prediction results of a statistical algorithm. In the context of decision tree ensembles, boosting works by growing the trees sequentially using information from previously grown trees. Each tree is grown with a slightly modified version of the entire data set. The incremental nature of the boosting algorithms means that learning happens slowly and minimizes the potential for overfitting. The increments at each step consider the residuals or errors from the previously grown tree to direct the next increment or iteration. The following discussion of tree algorithms is adapted from [47] [48] and [49]. For a more in-depth treatment of the topic, the reader is referred to these sources.

Tree Algorithms

Regression and classification trees make predictions by splitting the space of predictor variables into disjoint regions $R_j,\ j = 1, 2, ..., J$. A constant γ_j is assigned to each of the disjoint regions and the predictive rule is set as outlined in Equation 39.

$$x \in R_j \implies f(x) = \gamma_j \tag{39}$$

Given the partition of the predictor space into disjoint regions, R_j, and their corresponding decision rules, a decision tree can be expressed as outlined in Equation 40. The decision tree solution is an optimization problem to find the optimal Θ, where $\Theta = \{R_j, \gamma_j\}_1^J$.

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{40}$$

A boosted decision tree model is a sum of M individual decision trees as outlined in Equation 41.

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m) \tag{41}$$

In order to grow the boosted tree model and find the optimal Θ parameters, we must solve the optimization problem outlined in Equation 42.

$$\hat{\Theta} = arg\ min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{42}$$

Numerical methods such as steepest descent or gradient boosting can be used to approximate the solution to Equation 42. For any differentiable loss function, we can pose a gradient descent algorithm with the goal of minimizing the loss in using our tree to predict ground truth.

With gradient boosting, at each step of the tree growing process, the solution is the one that maximally reduces 42 given the current model f_{m-1}. Hastie et al show that using the gradient of our loss function and our tree at step m, we can use squared error to measure closeness. The iterative approach reduces to fitting the three to the residuals between our prediction $f_{m-1}(x_i)$ and the ground truth y_i [47].

4.2.2 U-Net

The U-Net is a convolutional neural network (CNN) architecture used for semantic segmentation (i.e., pixel-wise classification) tasks. It was first introduced for biomedical image segmentation, and has gained increased popularity in many domains outside of the biomedical fields due to its ability to accurately segment images. The U-net architecture follows an encoder-decoder pattern. The encoder network applies subsequent convolutional blocks followed by max-pooling operations to reduce the input feature maps' resolution. The encoder's main objective is to learn a latent feature map from the input images somewhat similar to an object detection network. The encoded feature map is then fed into a decoder network that expands the feature maps back to the input image size. The expansion is accomplished by 2-D transpose convolutional blocks. These convolutional blocks interpolate and upsample our feature maps in a controlled manner. Moreover, the U-Net architecture incorporates skip connections from the high resolution encoder layers to combine them with the upsampled layers in the decoder. The original U-Net architecture published by Ronneberger et al is shown in Figure 43 [24].

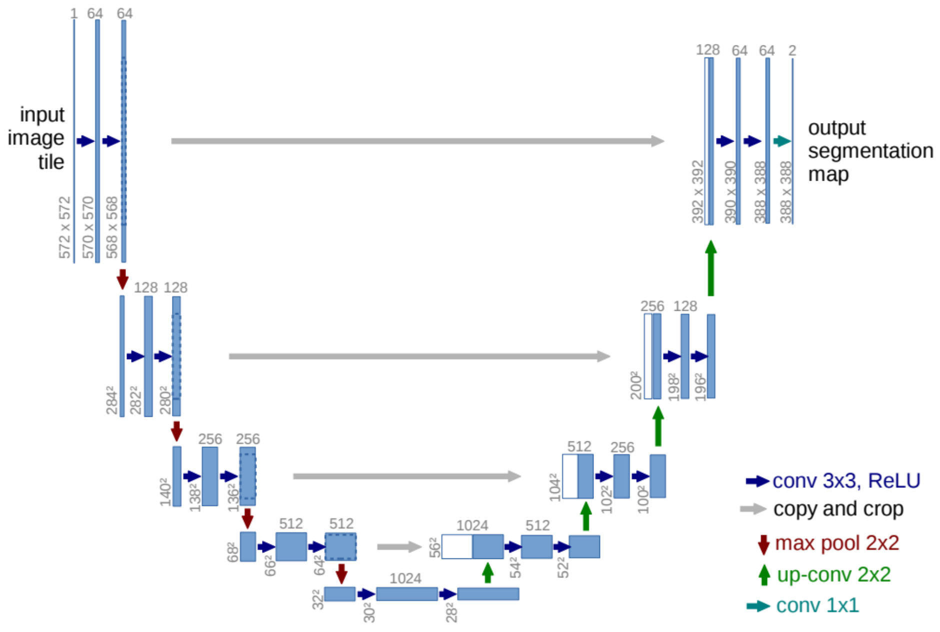


Figure 43: U-Net architecture. Adapted from [24]

4.2.3 Attention U-Net

The Attention U-Net architecture is a modification of the original U-Net architecture described in Section 4.2.2. The new architecture shown in Figure 44 introduces attention gates in the skip connections between the encoder and decoder paths. The attention gates' purpose is to learn how to focus on the important features in the input images to improve the semantic segmentation performance of the model. For the architecture used in this study, the attention gates are trained to focus on local relevant regions as opposed to focusing on global image properties. The inclusion of the attention gates does not drastically increase the training overhead compared to a traditional U-Net architecture and can be optimized with standard gradient descent algorithms. For a more in-depth treatment of the Attention U-Net architecture, the reader is referred to [50].

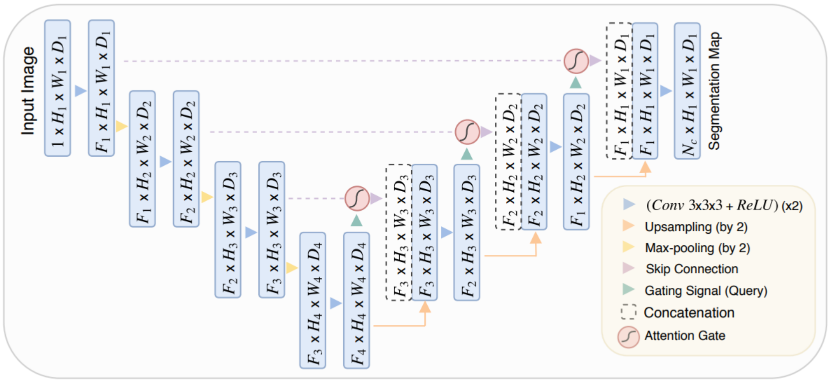

Input Image
$1 \times H_1 \times W_1 \times D_1$
$F_1 \times H_1 \times W_1 \times D_1$
$F_1 \times H_2 \times W_2 \times D_2$
$F_2 \times H_2 \times W_2 \times D_2$
$F_2 \times H_3 \times W_3 \times D_3$
$F_3 \times H_3 \times W_3 \times D_3$
$F_3 \times H_4 \times W_4 \times D_4$
$F_4 \times H_4 \times W_4 \times D_4$
$F_3 \times H_3 \times W_3 \times D_3$
$F_3 \times H_3 \times W_3 \times D_3$
$F_2 \times H_3 \times W_3 \times D_3$
$F_2 \times H_2 \times W_2 \times D_2$
$F_2 \times H_2 \times W_2 \times D_2$
$F_1 \times H_2 \times W_2 \times D_2$
$F_1 \times H_1 \times W_1 \times D_1$
$F_1 \times H_1 \times W_1 \times D_1$
$N_c \times H_1 \times W_1 \times D_1$
Segmentation Map
(Conv 3x3x3 + ReLU) (x2)
Upsampling (by 2)
Max-pooling (by 2)
Skip Connection
Gating Signal (Query)
Concatenation
Attention Gate

Figure 44: Attention U-Net architecture. Adapted from [50]

We focus on training Attention U-Net models in our study. Initial exploratory experiments showed that the Attention U-Net results outperformed the traditional U-Net results.

4.3 Data Set Statistics and Model Training

As outlined in Section 3.2, we subset the Sen1Floods11 data set to train our models. Prior to training, the data set was pruned to remove chips with non-valid data pixels instead of masking them during model training. This allowed us to train more consistently across neural networks and tree-based models. Moreover, the not-valid pixels in our labels are mapped to not-water pixels. This allowed us to train a binary classification model without worrying about a third, ‘not-valid’ class. The pruned training, validation, and test set splits per region are summarized in Table 4. Overall, we augment and train using roughly 50% of the Sen1Floods11 data set. Moreover, we maintain the exact same train, validation, and test set splits across all models trained. **Lastly, the chips from the Sri-Lanka region were not used during training or validation of our models. Instead, we reserved the Sri-Lanka chips for model generalization analyses as will be outlined in Section 5**.

Region	Train Chips	Val Chips	Test Chips	Hand-labeled Chips
USA	300	68	76	60
Mekong	906	155	189	27
Colombia	362	60	77	0
Paraguay	212	38	40	61
India	315	55	59	63
Bolivia	121	16	19	10
Sri-Lanka	0	0	185	41
Total	2216	392	645	262

Table 4: Train, validation, and test data set splits

4.3.1 Data Set Statistics

This section summarizes our data set statistics. Table 5 summarizes our pixel distribution for both the Sentinel-2 weakly labeled data set as well as the hand-labeled data set (including Sri-Lanka). Note that the data set is highly imbalanced with the not-water category far outweighing the water class. We do not assign class weights during model training, however.

Data Set	**Not-Water Pixels**	**Water Pixels**	**Not-Valid Pixels**
S2 Weakly Labeled	82.14%	13.03%	4.83%
Hand-Labeled	82.87%	9.71%	7.42%

Table 5: Pixel class distribution

Table 6 summarizes the average coherence across all chips per region for both the pre-event and co-event time steps. Note that in most regions, the pre-event coherence is higher than the co-event coherence as expected during a flooding event. However, for regions such as Bolivia, the pre-event coherence is not significantly higher than the co-event coherence. Moreover, we note that in Paraguay the co-event coherence is actually higher than the pre-event coherence. One potential explanation for this reversal in coherence magnitude may be due to the presence of vegetation in the scenes. As explained in Section 2, highly vegetated regions tend to exhibit high temporal decorrelation due to the randomness inherent in the scatterers' motion. In fact, most if not all of the scenes contained in the Sen1Floods11 data set used are not urban scenes. This poses a challenge when trying to exploit the coherence data given the intrinsically lower correlation of the scenes. Nonetheless, we move forward with our experiments to assess the relative improvements gained by fusing the coherence and intensity data.

Figure 45 shows the average pre-event and co-event coherence magnitude for 25 sample scenes. We can see a pattern of higher average pre-event coherence for most of the sampled scenes.

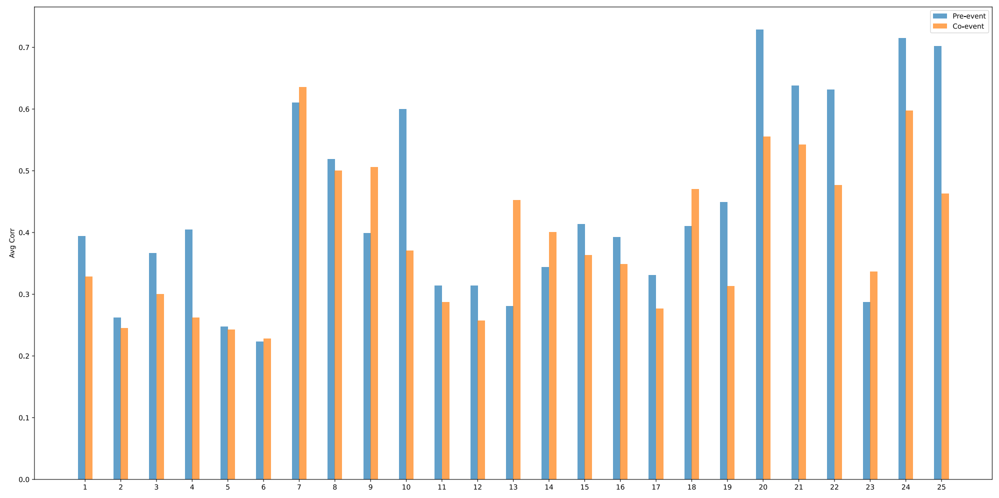


Figure 45: Average coherence for 25 scenes

	USA	Mekong	Colombia	Bolivia	Paraguay	India
Pre-event	64.92%	38.64%	43.80%	24.88%	28.13%	29.59%
Co-event	49.31%	37.09%	37.71%	24.62%	33.04%	27.33%

Table 6: Average coherence by region

4.3.2 Model Training

We train our Attention U-Net models with a batch size of 10 input scenes for 30 epochs using an Adam optimizer [51] with an initial learning rate of $1e^{-4}$, momentum parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and exponential decay rate of 0.96. Each model was trained using TensorFlow [52] on an NVIDIA RTX 3090 GPU. Training time for the 10-meter resolution models took approximately 3 hours per model scenario.

The only data augmentation used with our U-Net models were random 90° flips. We leverage the publicly available Keras U-Net Collection repository published in [53]. Our Attention U-Net models use the VGG-16 [54] architecture as the encoder's backbone, with weights randomly initialized. That is, we do not perform transfer learning and instead train our model weights from scratch. Training our model weights from scratch minimizes our risk of negative transfer given the uniqueness of our input data set. We use batch normalization and ReLU activations for both the convolutional layers and the attention gates and Sigmoid activation at the output.

We train our XGBoost models using the XGBoost python package [55] with GPU acceleration. Each XGBoost classifier is trained with 100 boosting rounds ($n_{estimators}$), a learning rate (η) of 0.3, and maximum tree depth of 6. The training was performed in a serial manner by splitting the data set in two batches that each fit in GPU memory. The first batch was fit to an initial model that is subsequently loaded for training continuation with the second data set batch. No data augmentation was used for the XGBoost models. Moreover, we use the raw intensity and coherence data pixels as inputs to the XGBoost classifiers. In other words, no feature engineering was performed for the XGBoost models. Training time for each XGBoost model averaged around 3 minutes on the same NVIDIA RTX 3090 GPU used to train the Attention U-Net models.

5 Results and Discussion

In this section, we summarize experiment results for the XGBoost and Attention U-Net models trained using the 10-meter resolution intensity and coherence data set described in Section 3. As mentioned in Section 3.2, the on-demand InSAR products offered by the HyP3 platform are available in 40- or 80-meter resolution. Since our objective is to train models with as high a spatial resolution as possible, the results presented below will only consider the 10-meter experiments. **However, it is worth noting that 40-meter resolution models were also trained as part our initial exploratory experiments with coherence data. The relative improvements obtained with 40-meter resolution data are in line with the 10-meter resolution results outlined in this section.**

5.1 Metrics

This section provides definitions for metrics we will use to analyze our classification results. Section 4.3.1 highlighted the fact that our data set is highly imbalanced. This means that the majority class in our data set (not-water class) will mask the water-class pixels, and thus, overall accuracy is not the most informative metric to assess our results. We use intersection over union as our evaluation metric of choice. **Intersection over union is defined as the overlap between the predicted and ground truth labels divided by the union of the predicted and ground truth labels**. Equation 43 outlines the definition mathematically, where TP, FP, and FN are true positive, false positive, and false negative rates respectively. When considering all classes, it is customary to report a mean IoU, which is the average of the individual classes' IoU [56].

$$\text{IoU} = \frac{TP}{TP + FP + FN} \tag{43}$$

Moreover, we also report precision, recall, and f-1 score metrics. Scikit-learn's documentation provides the following intuitive definition of precision and recall: "Precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples" [57].

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. **High precision relates to a low false positive rate**. Precision answers the following question: of all the water pixels labeled, how many are actually true water pixels?

$$\text{Precision} = \frac{TP}{TP + FP} \tag{44}$$

Recall is the ratio of correctly predicted positive observations to the total number of observations in the pertaining class. Recall answers the following question: of all the water pixels, how many did we label as water?

$$\text{Recall} = \frac{TP}{TP + FN} \tag{45}$$

F1-score is the weighted harmonic mean of precision and recall.

$$F_1 = \frac{2\ precision \times recall}{precision + recall} \tag{46}$$

5.2 Classification Results

Classification results are presented for two different test sets. The first test set is a held-out subset of our Sentinel-2 weakly-labeled chips. This weakly-labeled test set is similar to the weak labels used during training (i.e., optical bands thresholds). The second test set is a hand-labeled subset of the 446 chips included with the Sen1Floods11 data set. This report includes the weakly-labeled held-out test set results for completeness. However, we focus most of our results discussions around the hand-labeled test set results. **The hand-labeled test set represents the best way to validate our hypothesis because the labels are completely independent from the training pipeline**. This means that the hand-labeled test set allows us to both assess our water classification improvements as well as to answer the question: how well does the model generalize and agree with independent data? The reader is encouraged to review both test set results.

5.2.1 A note on Sentinel-1-derived labels

Section 3.1 described that the Sen1Floods11 data set contains Sentinel-1-derived weak labels generated with an Otsu threshold of the VV-polarized intensity band. It is worth noting that this study ran exploratory experiments using the Sentinel-1 weak labels from the Sen1Floods11 data set. However, our models were very prone to quickly overfit to the co-event band and the Sentinel-1 weak labels. These results are almost expected since the weak labels were originally generated from the same co-event data used as input to the classification models. In other words, our semantic segmentation models were asked to learn what threshold was used to generate the weak labels.

Similarly, HYDRAFloods [15] and HydroSAR-generated [18] labels were used as part of our exploratory experiments. Both publicly available Python libraries introduced in Section 1 use Sentinel-1 intensity data to segment water pixels. The results obtained with the HYDRAFloods and HydroSAR-generated labels are in line with those obtained with the Sen1Floods11 Sentinel-1 weak labels. Our models quickly learnt the thresholds used to segment the water pixels, and did not exploit the coherence information.

Instead, we shifted our focus and co-trained using the Sentinel-2-derived weak labels in the Sen1Floods11 data set. This approach allowed us to fuse different sensor and data modalities to test our models' robustness for semantic water segmentation and the relative improvements gained by introducing the coherence data.

5.2.2 Attention U-Net Results for all Geographical Regions

This section presents the Attention U-Net model's classification results aggregated over all the geographical regions studied. Intersection over union results for the different model scenarios are summarized in Tables 7 and 8 for the weakly-labeled held-out test set and the hand-labeled test set, respectively. Figure 46 summarizes the IoU results graphically. Tables 9 and 10 summarize overall classification results including precision, recall, and the f1-score metrics.

Considering the co-event and bi-temporal intensity models, our first observation is that adding the pre-event intensity data to the co-event intensity model does not improve the IoU results for the Attention U-Net architecture. This is true for both the weakly-labeled held-out test set and the hand-labeled test set (Tables 7 and 8). In fact, our results show that the co-event intensity model outperforms the bi-temporal intensity model in terms of IoU. This observation is further supported by the water-class precision, recall, and f1-scores presented in Tables 9 and 10. **This indicates that the co-event intensity bands are the most relevant features for mapping surface water**. This is in agreement with what we would expect given that the co-event scenes will exhibit different spatial distributions from the pre-event scenes due to the presence of flood water.

How does interferometric coherence impact our segmentation results?

Considering the hand-labeled data set results for our Attention U-Net models, **with the addition of the multi-temporal coherence information, our overall water IoU increases from** 54.57% **with the co-event intensity model to** 57.86% **with the bi-temporal intensity and coherence model (Table 8); a** 3.29% **improvement**. This represents the single largest IoU improvement across the weakly-labeled and hand-labeled data sets (see Figure 46). This means that our bi-temporal intensity and coherence model performs better with the hand-labeled data set than with the Sentinel-2 weak label modality used for training. **We also see a total mIoU increase from** 70.29% **with the co-event intensity model to** 72.31% **with the bi-temporal intensity and coherence model, further validating our hypothesis.**

In terms of our model's ability to recall the water pixels, Table 10 highlights that our bi-temporal intensity and coherence model outperforms the co-event and bi-temporal intensity models. **Comparing the co-event intensity model to the bi-temporal intensity and coherence model, we see that the water recall increases from** 59.60% **to** 63.93%**, respectively. This means that the addition of bi-temporal coherence data helps our model reduce the false negative rate**. That is, we reduce the percentage of pixels falsely classified as not-water. On the other hand, Table 10 also highlights that our bi-temporal intensity model outperforms the bi-temporal intensity and coherence model in water precision; 88.28% versus 85.90%, respectively. This means that the bi-temporal intensity model is better at minimizing the false positives rate. In other words, **our bi-temporal intensity and coherence model tends to over-estimate the water-class pixels relative to the co-event and bi-temporal intensity models**. Lastly, Table 10 also highlights that our bi-temporal intensity and coherence model outperforms the co-event and bi-temporal intensity models in terms of the f1-score.

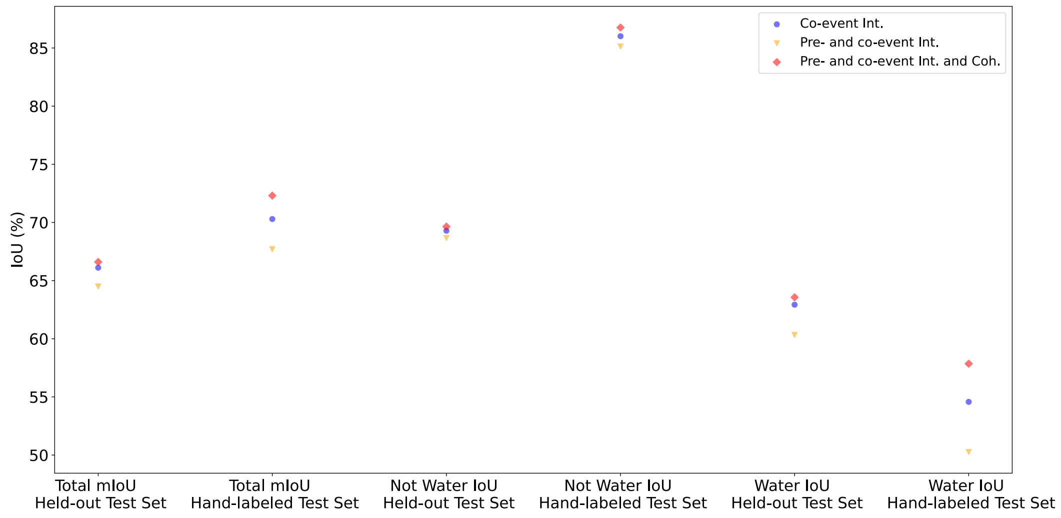


Figure 46: IoU Results for 10-meter Attention U-Net Models

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Total mIoU	66.11%	64.50%	66.60%
Not Water IoU	69.29%	68.66%	69.64%
Water IoU	62.93%	60.33%	63.56%

Table 7: IoU results for Attention U-Net models, held-out test set for all geographical regions

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Total mIoU	70.29%	67.70%	**72.31%**
Not Water IoU	86.01%	85.13%	86.76%
Water IoU	54.57%	50.27%	**57.86%**

Table 8: IoU results for Attention U-Net models, hand-labeled test set for all geographical regions

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Overall Accuracy	93.83%	93.65%	93.94%
Mean IoU	66.11%	64.50%	66.60%
Water Precision	77.66%	80.20%	77.79%
Water Recall	76.84%	70.89%	77.66%
Water f1-score	77.25%	75.26%	77.72%
Not Water Precision	96.35%	95.49%	96.48%
Not Water Recall	96.51%	97.24%	96.50%
Not Water f1-score	96.43%	96.36%	96.49%

Table 9: Overall classification results for Attention U-Net models, held-out test set for all geographical regions

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Overall Accuracy	95.29%	94.94%	95.58%
Mean IoU	70.29%	67.70%	**72.31%**
Water Precision	86.63%	**88.28%**	85.90%
Water Recall	59.60%	53.86%	**63.93%**
Water f1-score	70.61%	66.90%	**73.30%**
Not Water Precision	95.90%	95.35%	96.32%
Not Water Recall	99.04%	99.25%	98.90%
Not Water f1-score	97.44%	97.26%	97.59%

Table 10: Overall classification results for Attention U-Net models, hand-labeled test set for all geographical regions

5.2.3 XGBoost Results for all Geographical Regions

This section presents the XGBoost model's classification results aggregated over all the geographical regions studied. Intersection over union results for the different model scenarios are summarized in Tables 11 and 12 for the weakly-labeled held-out test set and the hand-labeled test set, respectively. Figure 47 summarizes the IoU results graphically. Tables 13 and 14 summarize overall classification results including precision, recall, and the f1-score metrics.

In terms of water IoU performance, in contrast to our Attention U-Net results, our XGBoost's bi-temporal intensity model outperforms our XGBoost co-event intensity model. Table 12 summarizes an incremental water IoU improvement from 52.36% to 53.20% for the hand-labeled data set. **Adding the bi-temporal coherence data improves our water IoU results from** 52.36% **for the co-event intensity model to** 55.87% **for the bi-temporal intensity and coherence model; a** 3.51% **improvement**. This relative water IoU improvement is in line with the results obtained with the bi-temporal instensity and coherence Attention U-Net model.

In terms of our XGBoost models' ability to recall water pixels, Table 14 shows that our bi-temporal intensity and coherence model outperforms the co-event and bi-temporal intensity models. **We see a water recall improvement from** 60.16% **for the co-event intensity model to** 65.57% **for the bi-temporal intensity and coherence model**. On the other hand, our XGBoost co-event and bi-temporal intensity models outperform our bi-temporal intensity and coherence model in the water precision metric. Finally, our bi-temporal intensity and coherence model outperforms both co-event and bi-temporal intensity models in terms of f1-scores. These results are also in line with the results obtained with the Attention U-Net models further validating our hypothesis.

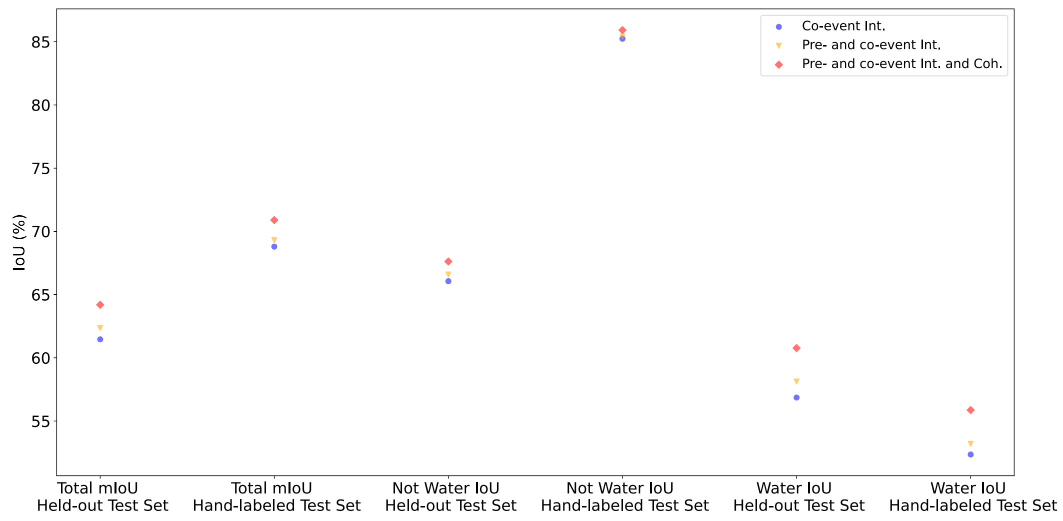


Figure 47: mIoU for 10-meter XGBoost Models

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Total mIoU	61.46%	62.36%	64.20%
Not Water IoU	66.06%	66.59%	67.62%
Water IoU	56.86%	58.13%	60.77%

Table 11: IoU results for XGBoost models, held-out test set for all geographical regions

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Total mIoU	68.80%	69.30%	**70.89%**
Not Water IoU	85.23%	85.39%	85.92%
Water IoU	52.36%	53.20%	**55.87%**

Table 12: IoU results for XGBoost models, hand-labeled test set for all geographical regions

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Overall Accuracy	92.85%	93.02%	93.34%
Mean IoU	61.46%	62.36%	64.20%
Water Precision	76.18%	76.08%	75.43%
Water Recall	69.16%	71.13%	75.77%
Water f1-score	72.50%	73.52%	75.60%
Not Water Precision	95.21%	95.49%	96.18%
Not Water Recall	96.59%	96.47%	96.11%
Not Water f1-score	95.89%	95.98%	96.14%

Table 13: Overall classification results for XGBoost models, held-out test set for all geographical regions

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Overall Accuracy	94.87%	94.93%	95.14%
Mean IoU	68.80%	69.30%	70.89%
Water Precision	**80.15%**	79.85%	79.06%
Water Recall	60.16%	61.45%	**65.57%**
Water f1-score	68.73%	69.45%	**71.69%**
Not Water Precision	95.98%	96.10%	96.50%
Not Water Recall	98.46%	98.40%	98.20%
Not Water f1-score	97.20%	97.24%	97.34%

Table 14: Overall classification results for XGBoost models, hand-labeled test set for all geographical regions

5.2.4 Classification Results Aggregated by Geographical Region

This section outlines classification results aggregated by geographical regions. The classification results encompass the weakly labeled held-out test set and the hand-labeled test set for completeness. Note that the Sen1Floods11 data set does not have hand-labeled data for Colombia. Our discussion will focus on the hand-labeled data set results. We start by summarizing IoU, water precision, recall and f1-score results. We then consider our models' ability to generalize to geographical regions different from the regions used to train our models.

For the Attention U-Net models, Table 16 along with Figure 48 summarize regional IoU results. Tables 18, 20, and 22 along with Figures 49, 50, and 51 summarize the water precision, recall, and f1-score results for the Attention U-Net models.

For the XGBoost models, Table 24 along with Figure 52 summarize regional IoU results. Tables 26, 28, and 30 along with Figures 53, 54, and 55 summarize the water precision, recall, and f1-score results for the XGBoost models.

Attention U-Net: Geographical Results

Table 16 summarizes the geographical water IoU results for the hand-labeled data set. **Our results highlight that our bi-temporal intensity and coherence data models outperform the co-event and bi-temporal intensity models for all regions studied**. Out of all regions, Sri-Lanka exhibits the greatest incremental improvement and will be considered in greater detail in our generalization section. On the other hand, we note that our IoU results over India exhibit the least amount of improvement by adding the bi-temporal coherence data, followed by the Mekong region. One potential explanation for these results over India is that the average interferometric coherence for both pre-event and co-event scenes is relatively low; 29.59% and 27.33% respectively (see Section 4.3). These low coherence values may be indicative that our scenes over India are highly vegetated areas where coherence is intrinsically low. Moreover, recall from Table 2 that the temporal baselines for the InSAR coherence maps over India are 24 days for the pre- and co-event pairs. The relatively longer temporal baselines for India compared to the rest of the geographical regions may induce a temporal decorrelation as described in Section 2.6.1.

Table 18 summarizes the water precision results for the hand-labeled data set aggregated by regions. We note that for most regions, with the exception of Bolivia, our bi-temporal intensity model outperforms our co-event intensity model as well as our bi-temporal intensity and coherence data. In fact, even though the results are mixed (e.g., Sri-Lanka and the USA) the bi-temporal intensity and coherence model seems to under-perform compared to the co-event intensity model in terms of water precision. These results are in line with the aggregated results for all regions outlined above. **Our bi-temporal intensity and coherence model tends to overestimate the water pixels**. These results are further validated visually by the label overlap plots in Section 5.2.10. We see that our Attention U-Net bi-temporal intensity and coherence model tends to lead in terms of false positives (orange regions).

On the flip side, Table 20 highlights that our bi-temporal intensity and coherence model outperforms both the co-event and bi-temporal intensity models in terms of water recall for the hand-labeled data set. **This means that our bi-temporal intensity and coherence model has the lower false negative rate over all regions**. These results can also be visually verified by the label overlap plots in Section 5.2.10. We see that our bi-temporal intensity and coherence model exhibit the least amount of false negative regions (gray regions).

Lastly, we note from Table 22 that our bi-temporal intensity and coherence model outperforms the co-event and bi-temporal intensity models in terms of the f1-score for all regions.

XGBoost: Geographical Results

Table 24 summarizes the geographical water IoU results for the hand-labeled data set for the XGBoost models. **Similar to our Attention U-Net results, the bi-temporal inten-**

sity and coherence XGBoost model outperforms the co-event and bi-temporal intensity model across all regions. We further verify that our IoU results over India exhibit the least amount of improvement by adding the bi-temporal coherence data to our model.

The water precision results of the XGBoost models summarized in Table 26 are a bit more mixed than our Attention U-Net results. We note that the co-event intensity model leads the water precision results over India, Paraguay, and the Mekong regions. The bi-temporal intensity and coherence model outperforms its counterparts over USA and Sri-Lanka. Lastly, the bi-temporal intensity model outperforms water precision over Bolivia. These results can be visualized with the label overlap plots from Section 5.2.11.

Table 28 summarizes the water recall results by region for the hand-labeled data set. **We note that our bi-temporal intensity and coherence model outperforms the co-event and bi-temporal intensity model over all regions**. In Section 5.2.11, we can visually verify that our bi-temporal intensity and coherence model exhibits the lower false negative rate (gray regions) out of the three models considered.

Lastly, as was the case for the Attention U-Net models, we note from Table 30 that our bi-temporal intensity and coherence model outperforms the co-event and bi-temporal intensity model in terms of the f1-score for all regions.

5.2.5 Assessing our model's ability to generalize

Next, we assess our models' ability to generalize to a geographical region different from the regions used during model training. Sri-Lanka was chosen as the generalization region with 41 hand-labeled chips. Our first observation from Table 16 is that our Attention U-Net co-event intensity model provides unsatisfactory generalization capabilities. Adding the pre-event intensity layers improved the water IoU results, but the generalization results are still sub-par at 13.66%. **The largest water IoU improvement gain is obtained with the**

bi-temporal intensity and coherence model obtaining a water IoU of 39.49%.

Our XGBoost models' results summarized in Table 24 also confirm that our largest performance gains are obtained with the bi-temporal intensity and coherence model. **Our bi-temporal intensity and coherence XGBoost model obtains a water IoU of** 48.94% **beating the Attention U-Net generalization capabilities**. In fact, all of the XGBoost models outperform the Attention U-Net models with the generalization data set. If our ultimate goal is to operationalize a flood water detection model, these results pose good news. Training an XGBoost model carries significantly lower overhead compared to a CNN model like the Attention U-Net models trained in this study (assuming the same GPU is used for training).

We note from Tables 18, 20, and 22 for the Attention U-Net models and Tables 26, 28, and 30 for the XGBoost models that our bi-temporal intensity and coherence XGBoost model outperforms all of its counterparts in terms of water recall, and f1-score over Sri-Lanka with the exception of water precision for the Attention U-Net models. The bi-temporal intensity model outperforms the co-event intensity and bi-temporal intensity and coherence model in terms of water precision. Moreover, our XGBoost bi-temporal intensity and coherence model outperforms the Attention U-Net bi-temporal intensity and coherence model in terms of water recall and f1-score. Sections 5.2.8 and **??** compare the Attention U-Net and XGBoost models in more detail.

The label overlap plots shown in Sections 5.2.10 and 5.2.11 validate our quantitative results over Sri-Lanka. If we consider our XGBoost label overlap plot from Figure 67, we see show the improving progression visually. We note that the bi-temporal intensity and coherence model has the highest true positive and true negative rates. This model also boasts the lowest false negative rate.

5.2.6 Attention U-Net: Water IoU, Precision, Recall and F1-Score

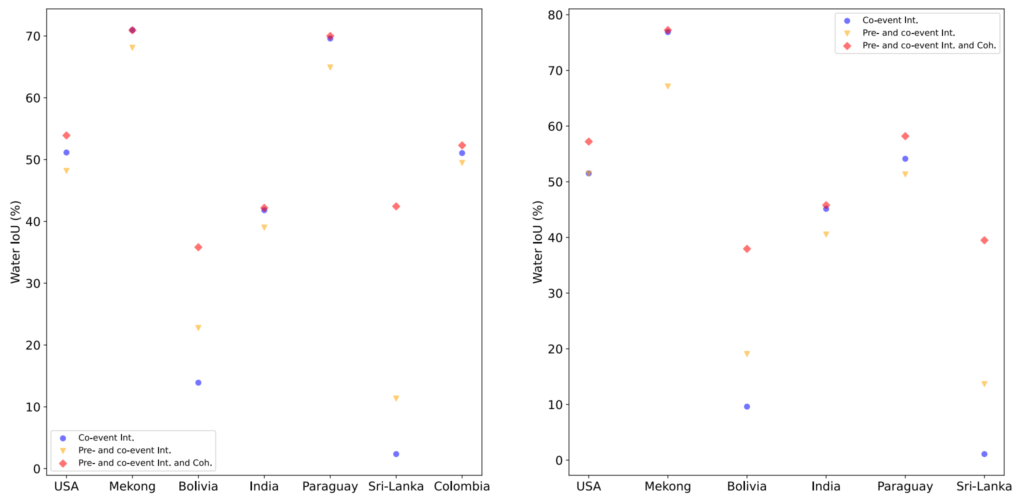


Figure 48: Water IoU for Attention U-Net models aggregated by geographical region, held-out test set (left) and hand-labeled test set (right)

	Co-event Int.	Pre- and co-event Int.	Pre- and co-event Int. and Coh.
USA	51.15%	48.17%	53.91%
Mekong	70.95%	68.09%	70.92%
Bolivia	13.90%	22.76%	35.82%
India	41.81%	38.99%	42.19%
Paraguay	69.59%	64.90%	69.99%
Colombia	51.06%	49.47%	52.30%
Sri-Lanka	2.35%	11.34%	42.43%

Table 15: Water IoU for Attention U-Net models aggregated by geographical region, held-out test set

	Co-event Int.	Pre- and co-event Int.	Pre- and co-event Int. and Coh.
USA	51.51%	51.43%	**57.22%**
Mekong	76.88%	67.13%	**77.25%**
Bolivia	09.61%	19.06%	**37.96%**
India	45.14%	40.52%	**45.80%**
Paraguay	54.13%	51.33%	**58.19%**
Sri-Lanka	1.09%	13.66%	**39.49%**

Table 16: Water IoU for Attention U-Net models aggregated by geographical region, hand-labeled test set

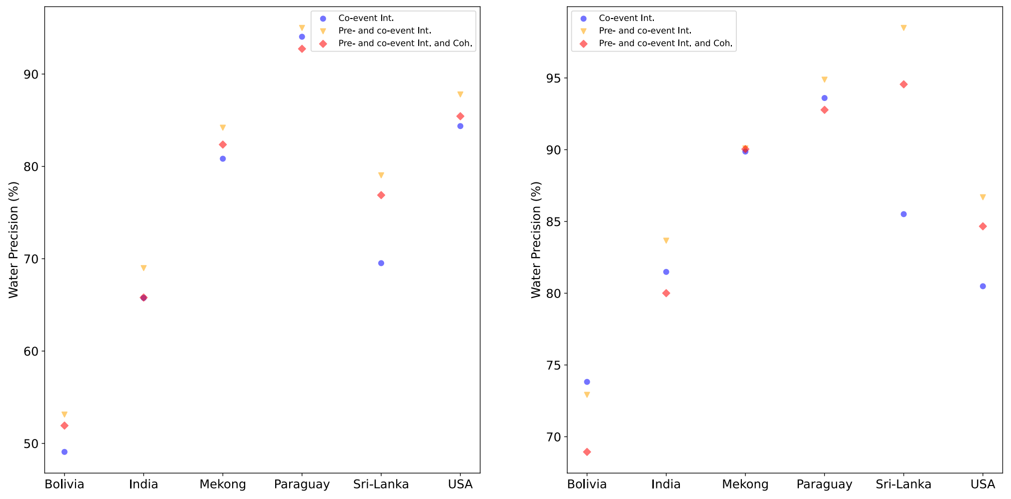


Figure 49: Water Precision for Attention U-Net models aggregated by geographical region, held-out test set (left) and hand-labeled test set (right)

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	49.08%	53.12%	51.93%
India	65.76%	68.99%	65.79%
Mekong	80.83%	84.19%	82.36%
Paraguay	94.04%	95.00%	92.74%
USA	84.37%	87.79%	85.43%
Sri-Lanka	69.52%	79.04%	76.89%

Table 17: Water precision for Attention U-Net models aggregated by geographical region, held-out test set

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	**73.82%**	72.92%	68.94%
India	81.49%	**83.66%**	80.00%
Mekong	89.87%	**90.09%**	90.04%
Paraguay	93.60%	**94.88%**	92.78%
USA	80.49%	**86.69%**	84.66%
Sri-Lanka	85.51%	**98.49%**	94.56%

Table 18: Attention U-Net water precision aggregated by geographical region, hand-labeled test set

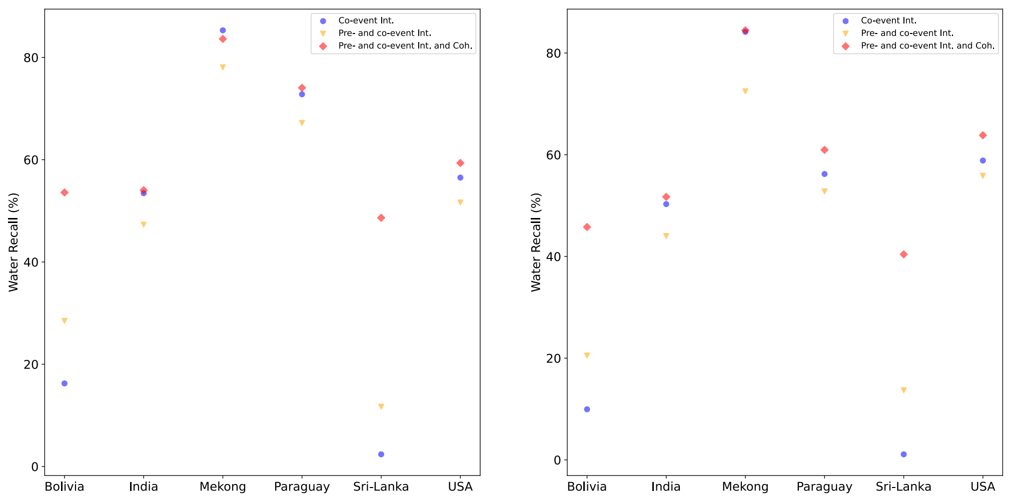


Figure 50: Water Recall for Attention U-Net Models aggregated by geographical region, held-out test set (left) and hand-labeled test set (right)

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	16.25%	28.47%	53.59%
India	53.45%	47.29%	54.04%
Mekong	85.31%	78.07%	83.62%
Paraguay	72.80%	67.20%	74.06%
USA	56.50%	51.63%	59.36%
Sri-Lanka	2.38%	11.69%	48.64%

Table 19: Water recall for Attention U-Net models aggregated by geographical region, held-out test set

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	9.95%	20.51%	**45.78%**
India	50.29%	44.00%	**51.72%**
Mekong	84.18%	72.48%	**84.46%**
Paraguay	56.21%	52.79%	**60.95%**
USA	58.86%	55.84%	**63.83%**
Sri-Lanka	1.09%	13.69%	**40.41%**

Table 20: Water recall for Attention U-Net models aggregated by geographical region, hand-labeled test set

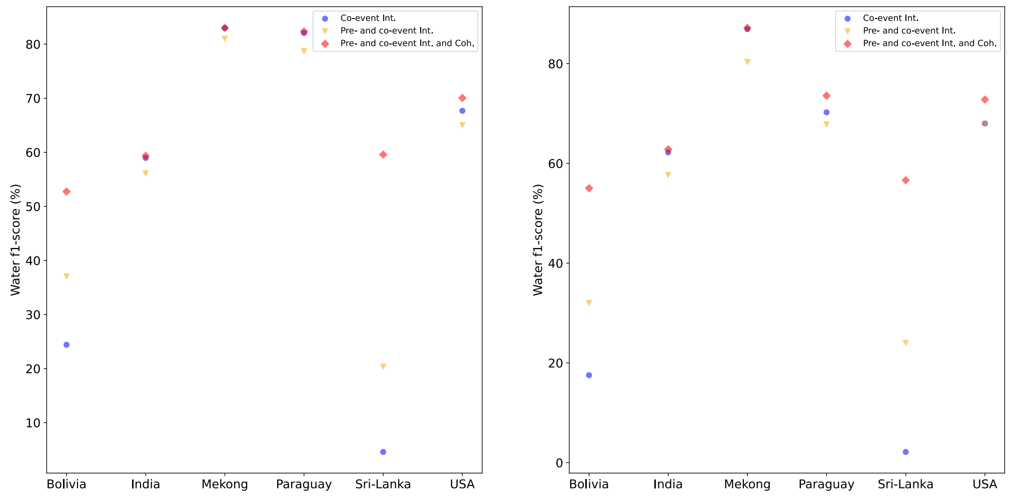


Figure 51: Water F1-Score for Attention U-Net Models aggregated by geographical region, held-out test set (left) and hand-labeled test set (right)

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	24.41%	37.08%	52.75%
India	58.97%	56.11%	59.34%
Mekong	83.01%	81.02%	82.99%
Paraguay	82.07%	78.72%	82.35%
USA	67.68%	65.02%	70.05%
Sri-Lanka	4.60%	20.37%	59.58%

Table 21: Attention U-Net water f1-score aggregated by geographical region, held-out test set

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	17.54%	32.02%	**55.03%**
India	62.20%	57.67%	**62.83%**
Mekong	86.93%	80.33%	**87.16%**
Paraguay	70.24%	67.84%	**73.57%**
USA	68.00%	67.93%	**72.79%**
Sri-Lanka	2.16%	24.03%	**56.62%**

Table 22: Water f1-score for Attention U-Net models aggregated by geographical region, hand-labeled test set

5.2.7 XGBoost: Water IoU, Recall, Precision and F1-Score

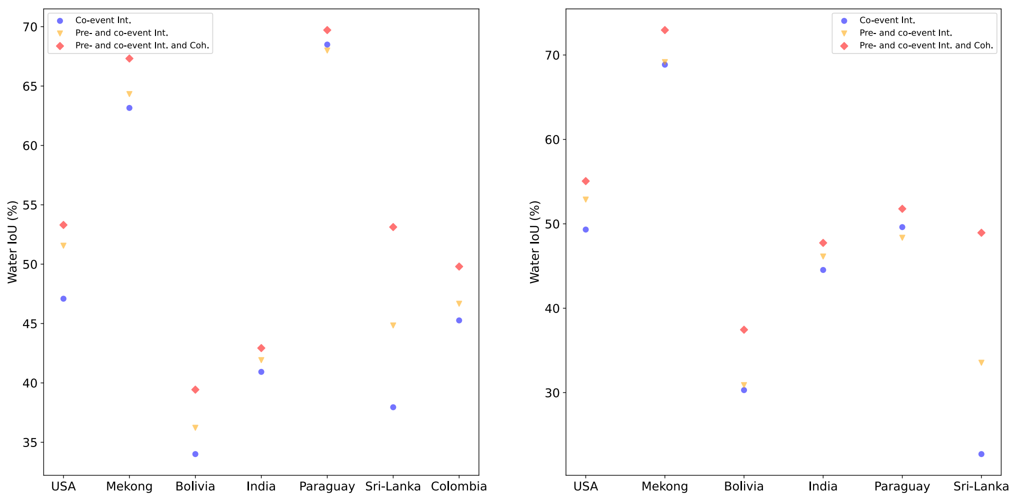


Figure 52: Water IoU for Attention XGBoost models aggregated by geographical regions, held-out test set (left) and hand-labeled test set (right)

	Co-event Int.	Pre- and co-event Int.	Pre- and co-event Int. and Coh.
USA	47.09%	51.56%	53.30%
Mekong	63.16%	64.33%	67.32%
Bolivia	34.01%	36.22%	39.43%
India	40.93%	41.92%	42.93%
Paraguay	68.50%	67.99%	69.72%
Colombia	45.26%	46.67%	49.80%
Sri-Lanka	37.95%	44.84%	53.13%

Table 23: Water IoU for XGBoost models aggregated by geographical region, held-out test set

	Co-event Int.	Pre- and co-event Int.	Pre- and co-event Int. and Coh.
USA	49.32%	52.87%	**55.06%**
Mekong	68.85%	69.16%	**72.95%**
Bolivia	30.30%	30.89%	**37.45%**
India	44.53%	46.13%	**47.74%**
Paraguay	49.61%	48.35%	**51.78%**
Sri-Lanka	22.72%	33.54%	**48.94%**

Table 24: Water IoU for XGBoost models aggregated by geographical region, hand-labeled test set

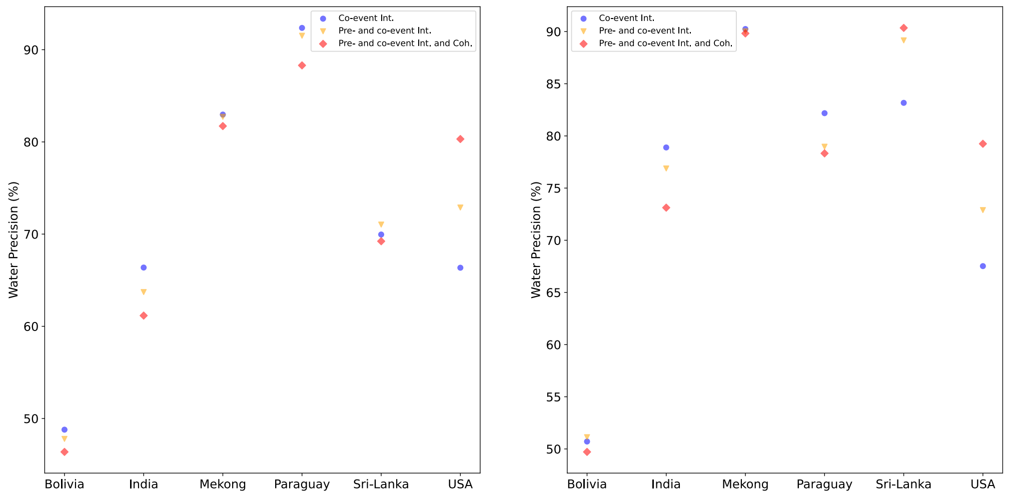


Figure 53: Water Precision for XGBoost models aggreagted by region, held-out test set (left) and hand-labeled test set (right)

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	48.79%	47.79%	46.38%
India	66.37%	63.71%	61.16%
Mekong	82.96%	82.72%	81.72%
Paraguay	92.37%	91.52%	88.30%
USA	66.35%	72.88%	80.31%
Sri-Lanka	69.96%	71.03%	69.23%

Table 25: Water precision for XGBoost models aggregated by geographical region, held-out test set

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	50.71%	**51.13%**	49.72%
India	**78.89%**	76.88%	73.12%
Mekong	**90.24%**	90.03%	89.82%
Paraguay	**82.18%**	78.97%	78.33%
USA	67.52%	72.89%	**79.25%**
Sri-Lanka	83.17%	89.16%	**90.35%**

Table 26: Water precision for XGBoost models aggreagted by geographical region, hand-labeled test set

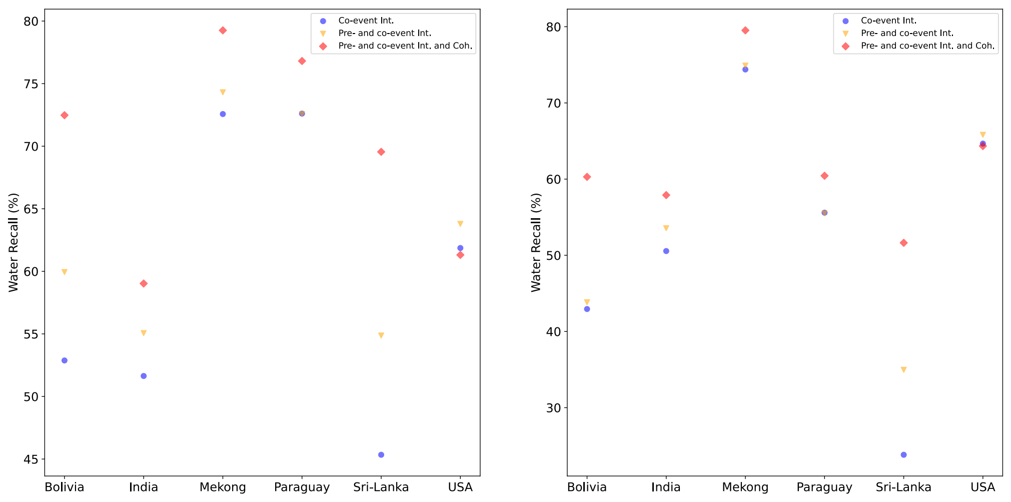


Figure 54: Water Recall for XGBoost models aggregated by geographical region, held-out test set (left) and hand-labeled test set (right)

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	52.88%	59.94%	72.48%
India	51.64%	55.06%	59.03%
Mekong	72.57%	74.31%	79.26%
Paraguay	72.61%	72.57%	76.81%
USA	61.86%	63.79%	61.31%
Sri-Lanka	45.34%	54.87%	69.55%

Table 27: Water recall for XGBoost models aggregated by geographical region, held-out test set

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	42.95%	43.84%	**60.29%**
India	50.56%	53.56%	**57.90%**
Mekong	74.39%	74.89%	**79.52%**
Paraguay	55.59%	55.50%	**60.44%**
USA	64.66%	**65.82%**	64.34%
Sri-Lanka	23.82%	34.97%	**51.64%**

Table 28: Water recall for XGBoost models aggregated by geographical region, hand-labeled test set

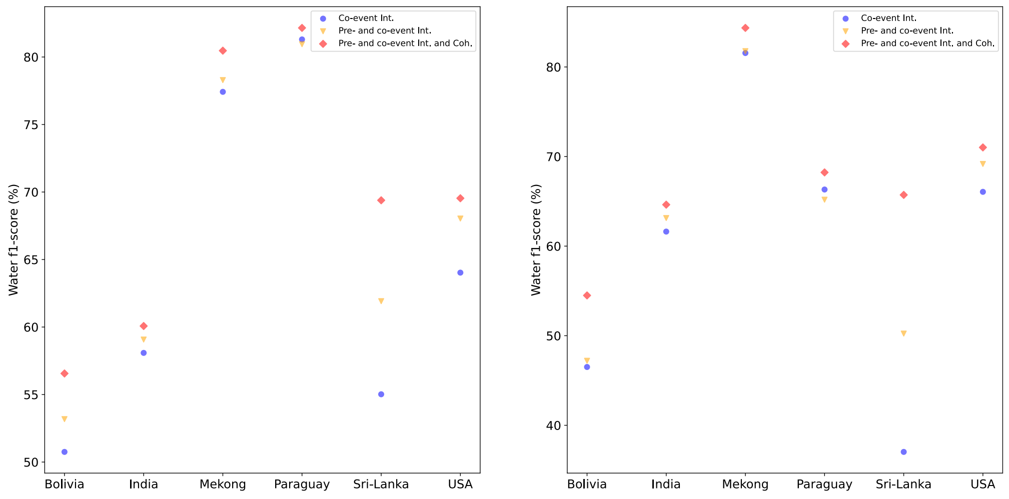


Figure 55: Water F1-Score for XGBoost models aggregated by geographical region, held-out test set (left) and hand-labeled test set (right)

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	50.75%	53.18%	56.56%
India	58.08%	59.07%	60.08%
Mekong	77.42%	78.29%	80.47%
Paraguay	81.30%	80.95%	82.16%
USA	64.03%	68.04%	69.54%
Sri-Lanka	55.02%	61.92%	69.39%

Table 29: Water f1-score for XGBoost models aggreagted by geographical region, held-out test set

	Co-event Intensity	Pre- and co-event Int.	Pre- and co-event Int. and Coherence
Bolivia	46.51%	47.20%	**54.50%**
India	61.62%	63.13%	**64.63%**
Mekong	81.55%	81.77%	**84.36%**
Paraguay	66.32%	65.18%	**68.23%**
USA	66.06%	69.17%	**71.02%**
Sri-Lanka	37.03%	50.24%	**65.72%**

Table 30: Water f1-score for XGBoost models aggregated by geographical region, hand-labeled test set

5.2.8 Attention U-Net and XGBoost Results Comparison

In this section, we compare the Attention U-Net classification results to the XGBoost results. Tables 31, 32, and 33 outline the results aggregated over all geographical regions studied for the hand-labeled data set. The tables summarize overall accuracy, mean IoU, water precision, water recall, and water f1-score metrics. From Table 33, we conclude that the Attention U-Net bi-temporal intensity and coherence model outperforms its XGBoost counterpart in terms of mean IoU, water f1-score and water precision. A higher water precision score implies that the Attention U-Net model is better at reducing the false positive rate than the XGBoost model. We see a 6.84% delta between the Attention U-Net and XGBoost's precision scores. On the other hand, the XGBoost model outperforms the Attention U-Net model in terms of the water recall metric (a 1.74% improvement). This implies that the XGBoost model is better a reducing the false negative rate. Moreover, Table 31 outlines the results for the uni-temporal intensity model. These results align with the results just described for the bi-temporal intensity and coherence model. On the flip side, Table 32 highlights that the XGBoost bi-temporal intensity model outperforms its Attention U-Net counterpart in terms of mean IoU, water recall, and water f1-score. Tables 34, 35, and 36 summarize the IoU results for the Attention U-Net and XGBoost models. From Table **??**, we see that the Attention U-Net bi-temporal intensity and coherence model outperforms its XGBoost counterpart by about 1.99% in terms of water IoU.

	Attention U-Net	**XGBoost**
	Co-event intensity	**Co-event intensity**
Overall Accuracy	95.29%	94.87%
Mean IoU	**70.29%**	68.80%
Water Precision	**86.63%**	80.15%
Water Recall	59.60%	**60.16%**
Water f1-score	**70.61%**	68.73%

Table 31: Attention U-Net versus XGBoost results for co-event intensity models - all geographical regions

	Attention U-Net	**XGBoost**
	Pre- and co-event intensity	**Pre- and co-event intensity**
Overall Accuracy	94.94%	94.93%
Mean IoU	**67.70%**	69.30%
Water Precision	**88.28%**	79.85%
Water Recall	53.86%	**61.45%**
Water f1-score	**66.90%**	**69.45%**

Table 32: Attention U-Net versus XGBoost results for pre- and co-event intensity models - all geographical regions

	Attention U-Net Pre- and	**XGBoost Pre- and**
	co-event Int. and Coh.	**co-event Int. and Coh.**
Overall Accuracy	95.58%	95.14%
Mean IoU	**72.31%**	70.89%
Water Precision	**85.90%**	79.06%
Water Recall	63.93%	**65.57%**
Water f1-score	**73.30%**	**71.69%**

Table 33: Attention U-Net versus XGBoost results for pre- and co-event intensity and coherence models - all geographical regions

	Attention U-Net	**XGBoost**
	Co-event intensity	**Co-event intensity**
Total mIoU	**70.29%**	68.80%
Not Water IoU	86.01%	85.23%
Water IoU	**54.57%**	52.36%

Table 34: Attention U-Net versus XGBoost IoU results for co-event intensity models - all geographical regions

	Attention U-Net Pre- and co-event intensity	**XGBoost Pre- and co-event intensity**
Total mIoU	**67.70%**	**69.30%**
Not Water IoU	85.13%	85.39%
Water IoU	**50.27%**	**53.20%**

Table 35: Attention U-Net versus XGBoost IoU results for pre- and co-event intensity models - all geographical regions

	Attention U-Net Pre- and co-event Int. and Coh.	**XGBoost Pre- and co-event Int. and Coh.**
Total mIoU	**72.31%**	**70.89%**
Not Water IoU	86.76%	85.92%
Water IoU	**57.86%**	**55.87%**

Table 36: Attention U-Net versus XGBoost IoU results for pre- and co-event intensity and coherence models - all geographical regions

5.2.9 Attention U-Net and XGBoost Results Comparison for Generalization Data Set

In this section, we compare the Attention U-Net results to its XGBoost counterparts for the generalization data set over Sri-Lanka. Tables 37, 38, and 39 summarize the water IoU, recall, precision and f1-score for the generalization data set. We conclude that all of the XGBoost models are better than their Attention U-Net counterparts in terms of the water IoU, recall and f1-score metrics. In the case of the bi-temporal intensity and coherence model (see Table 39), the XGBoost models outperform the Attention U-Net by 9 to 11% in terms of water IoU, recall, and f1-score. This means that the XGBoost models are significantly better at reducing the false negative rate than the Attention U-Net models. On the other hand, we conclude that the Attention U-Net model is better at water precision than its XGBoost counterparts. This implies that the Attention U-Net models are better than the XGBoost models at reducing the false positive rate.

	Attention U-Net Co-event intensity	**XGBoost Co-event intensity**
Water IoU	1.09%	**22.72%**
Water Recall	1.09%	**23.82%**
Water f1-score	2.16%	**37.03%**
Water Precision	**85.51%**	83.17%

Table 37: Attention U-Net versus XGBoost IoU results for co-event intensity - generalization data set

	Attention U-Net Pre- and co-event Int.	**XGBoost Pre- and co-event Int.**
Water IoU	13.66%	**33.54%**
Water Recall	13.69%	**34.97%**
Water f1-score	24.03%	**50.24%**
Water Precision	**98.49%**	89.16%

Table 38: Attention U-Net versus XGBoost IoU results for pre- and co-event intensity - generalization data set

	Attention U-Net Pre- and co-event Int. and Coh.	**XGBoost Pre- and co-event Int. and Coh.**
Water IoU	39.49%	**48.94%**
Water Recall	40.41%	**51.64%**
Water f1-score	56.62%	**65.72%**
Water Precision	**94.56%**	90.35%

Table 39: Attention U-Net versus XGBoost IoU results for pre- and co-event intensity - generalization data set

5.2.10 Label Overlap: Attention U-Net

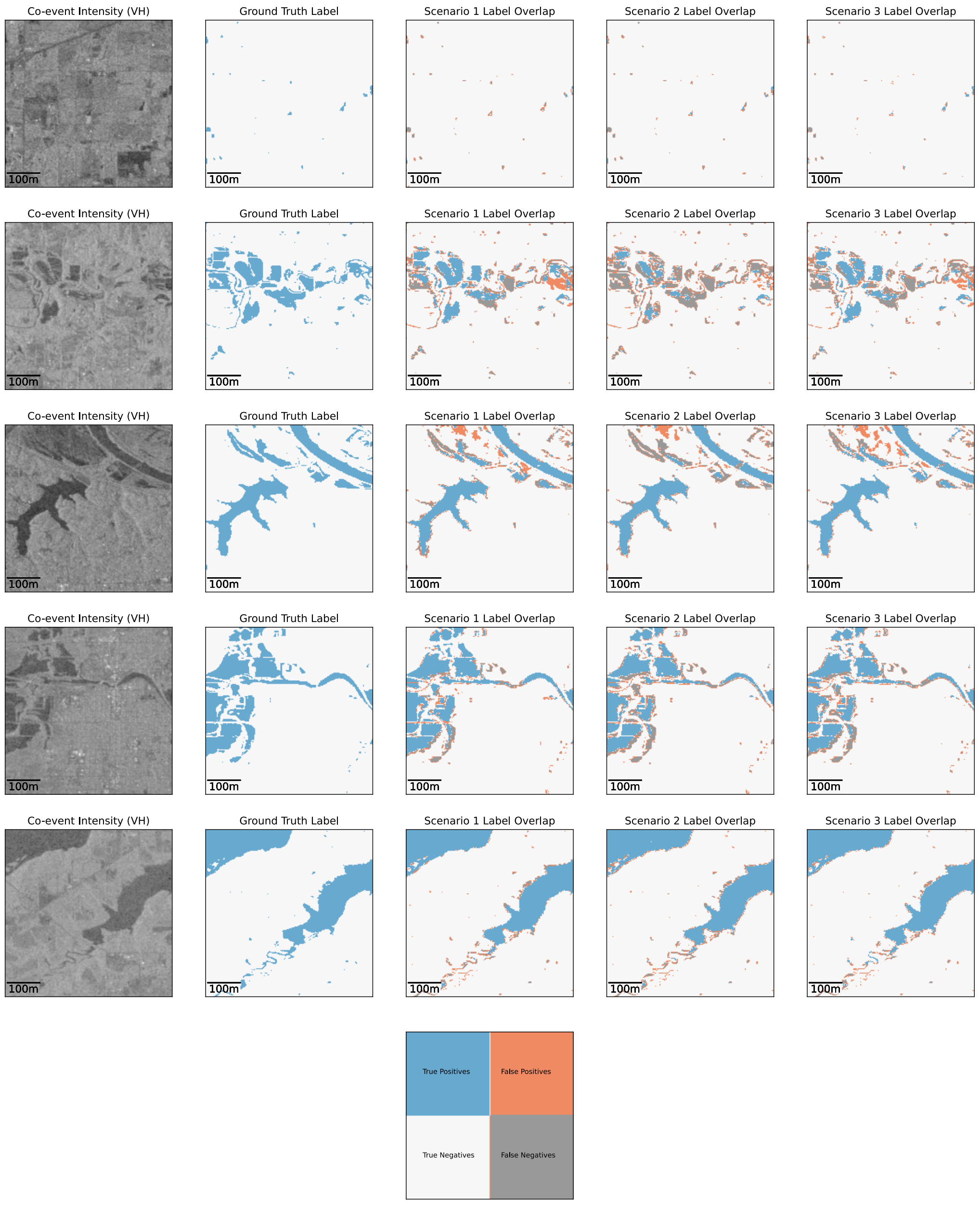


Figure 56: Label Overlap for Attention U-Net Models, USA

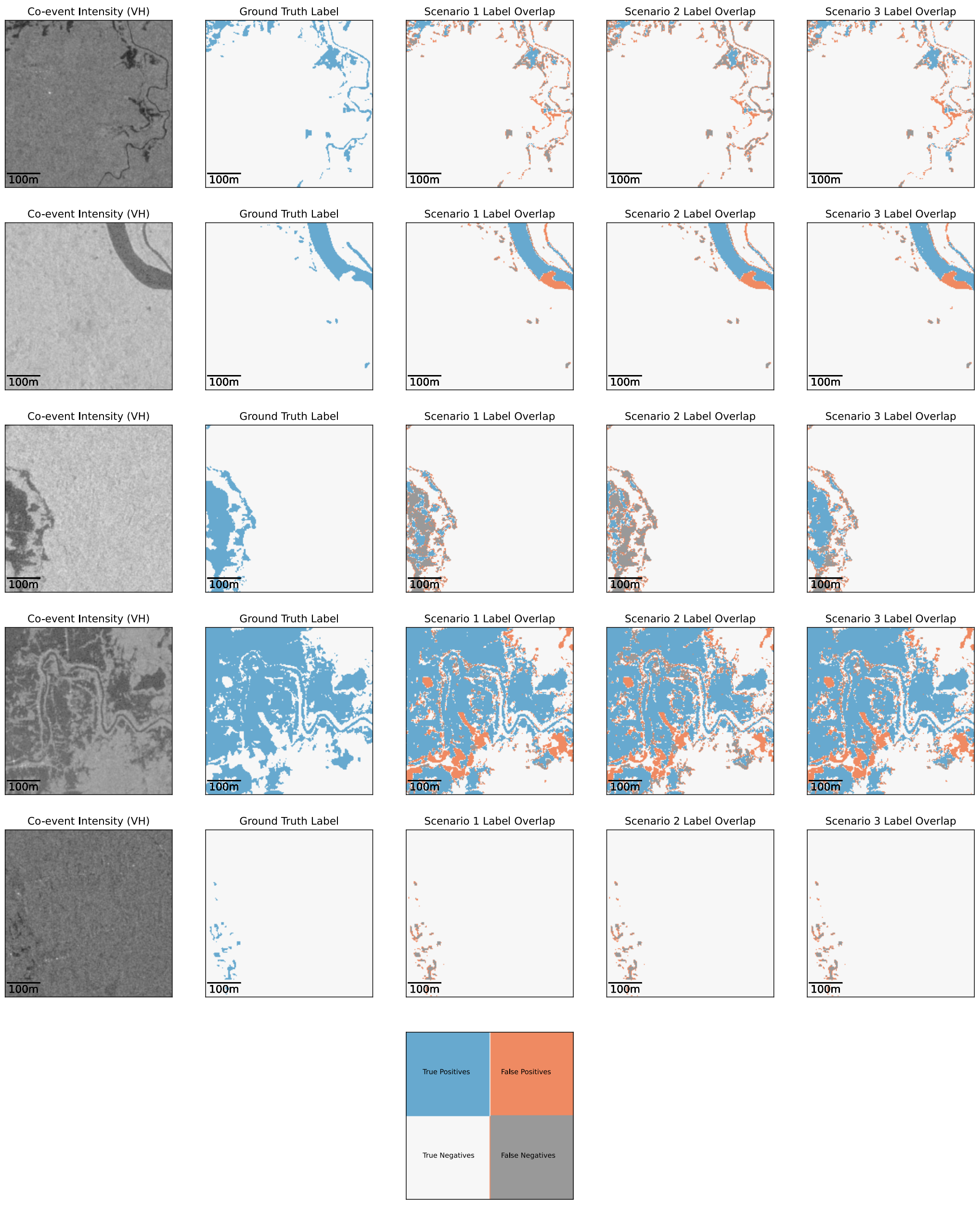


Figure 57: Label Overlap for Attention U-Net Models, Mekong

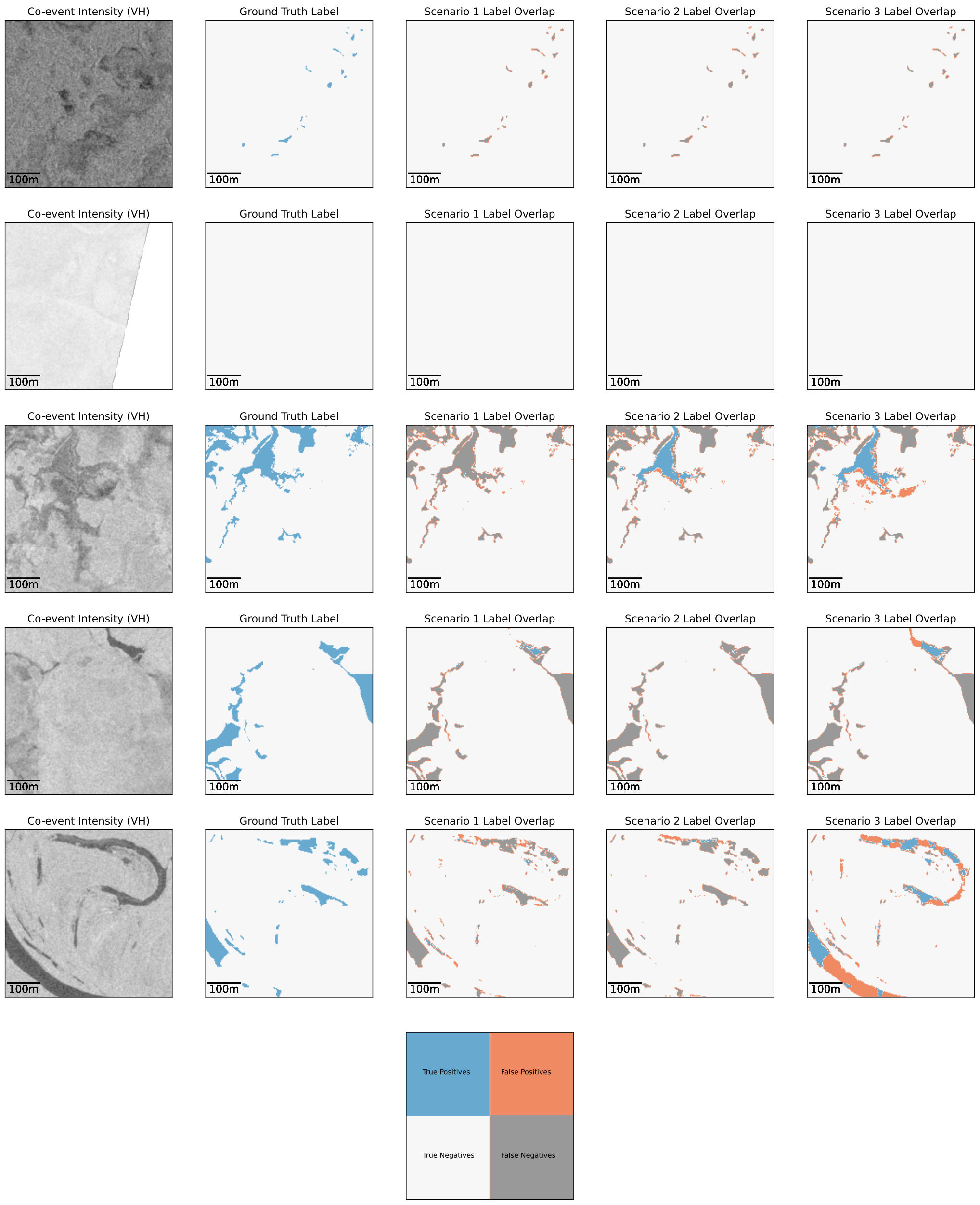


Figure 58: Label Overlap for Attention U-Net Models, Bolivia

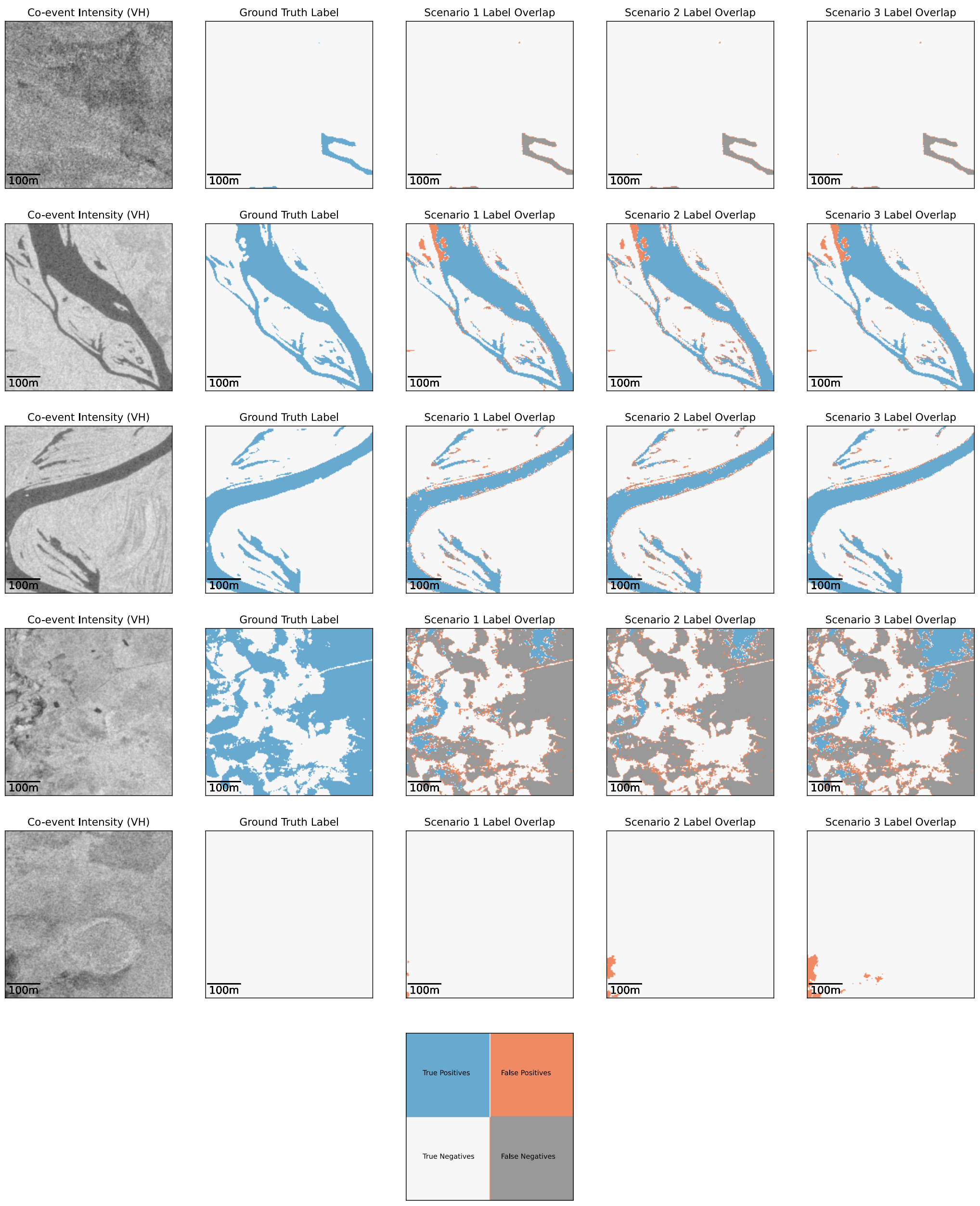


Figure 59: Label Overlap for Attention U-Net Models, Paraguay

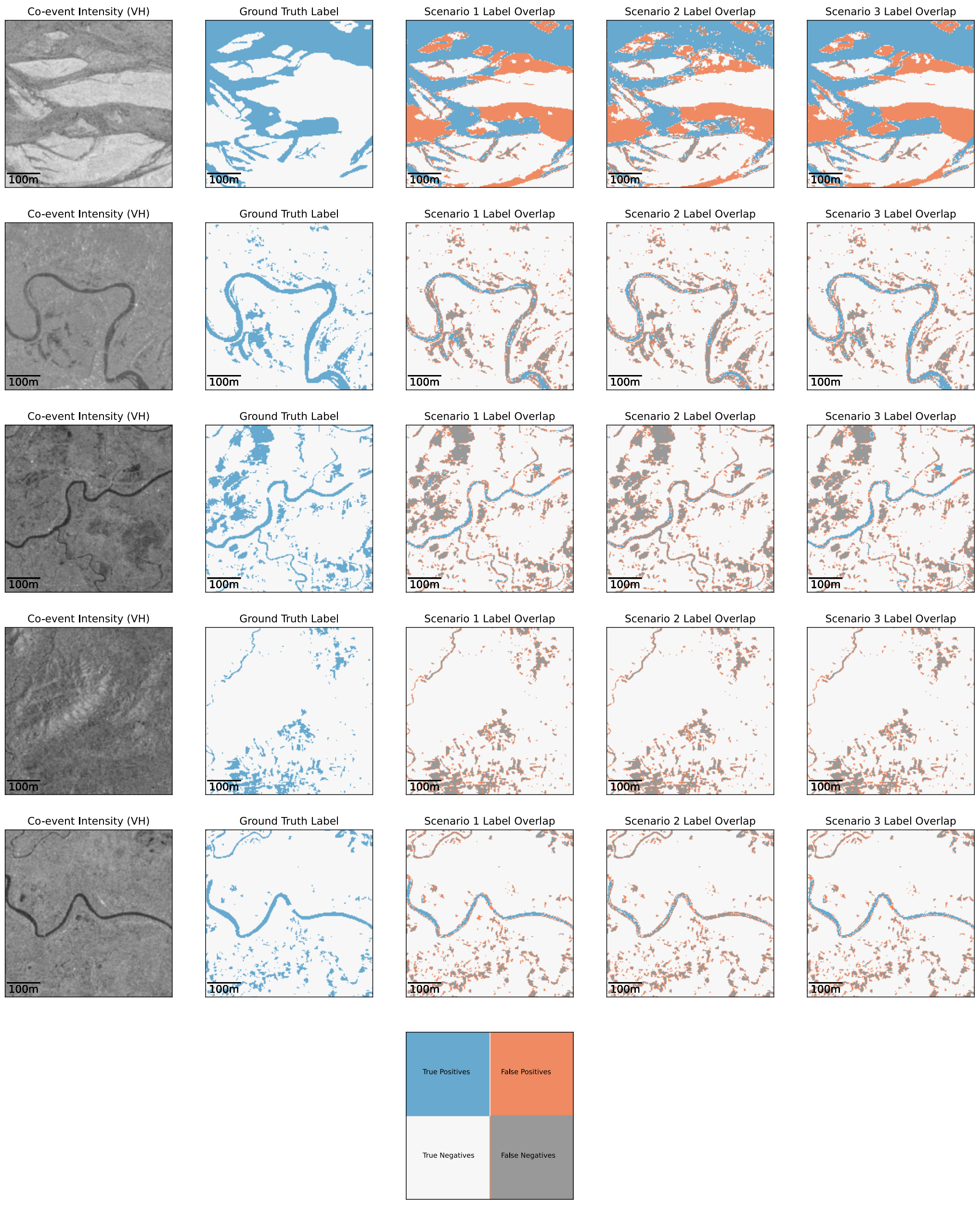


Figure 60: Label Overlap for Attention U-Net Models, India

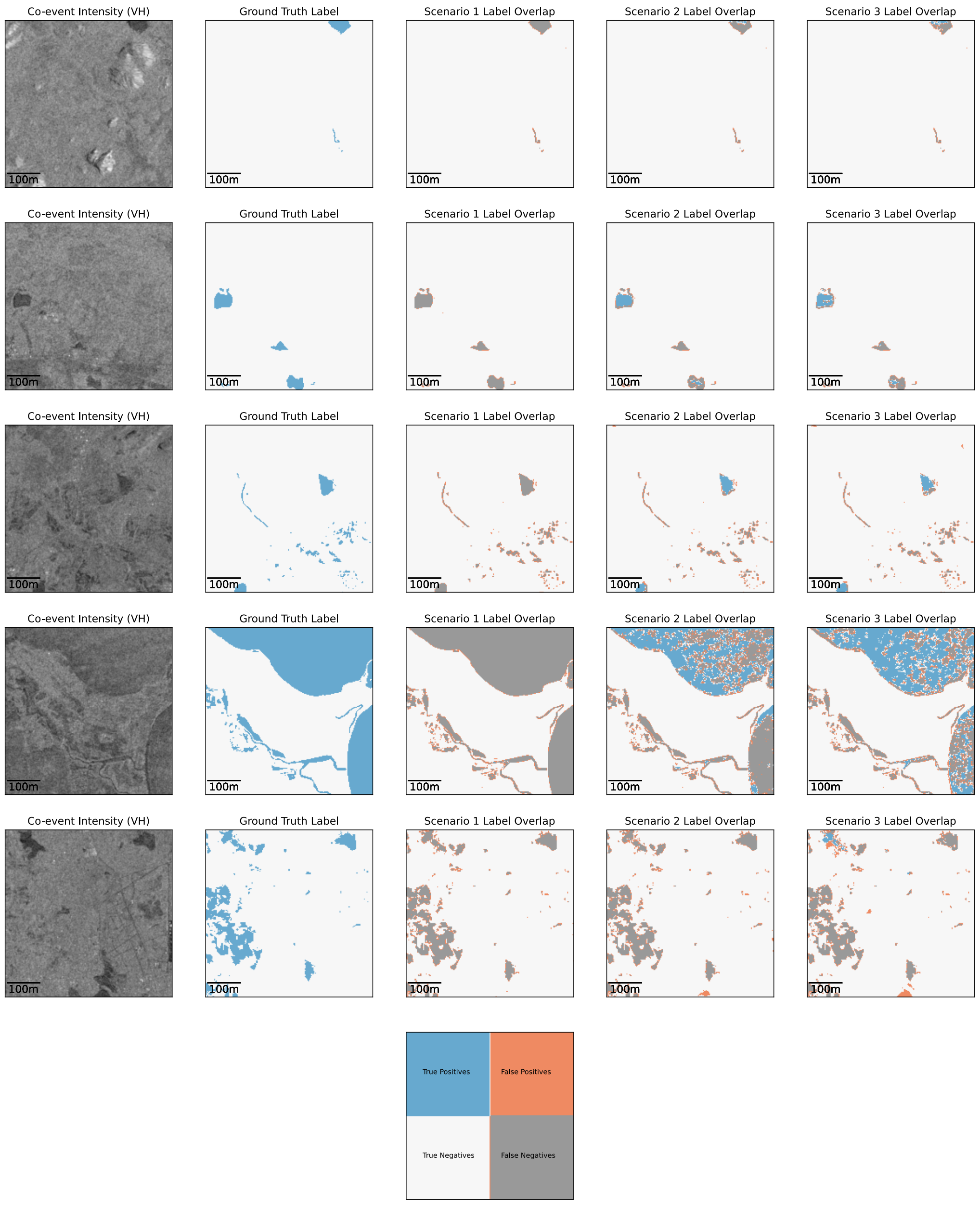


Figure 61: Label Overlap for Attention U-Net Models, Sri-Lanka

5.2.11 Label Overlap: XGBoost

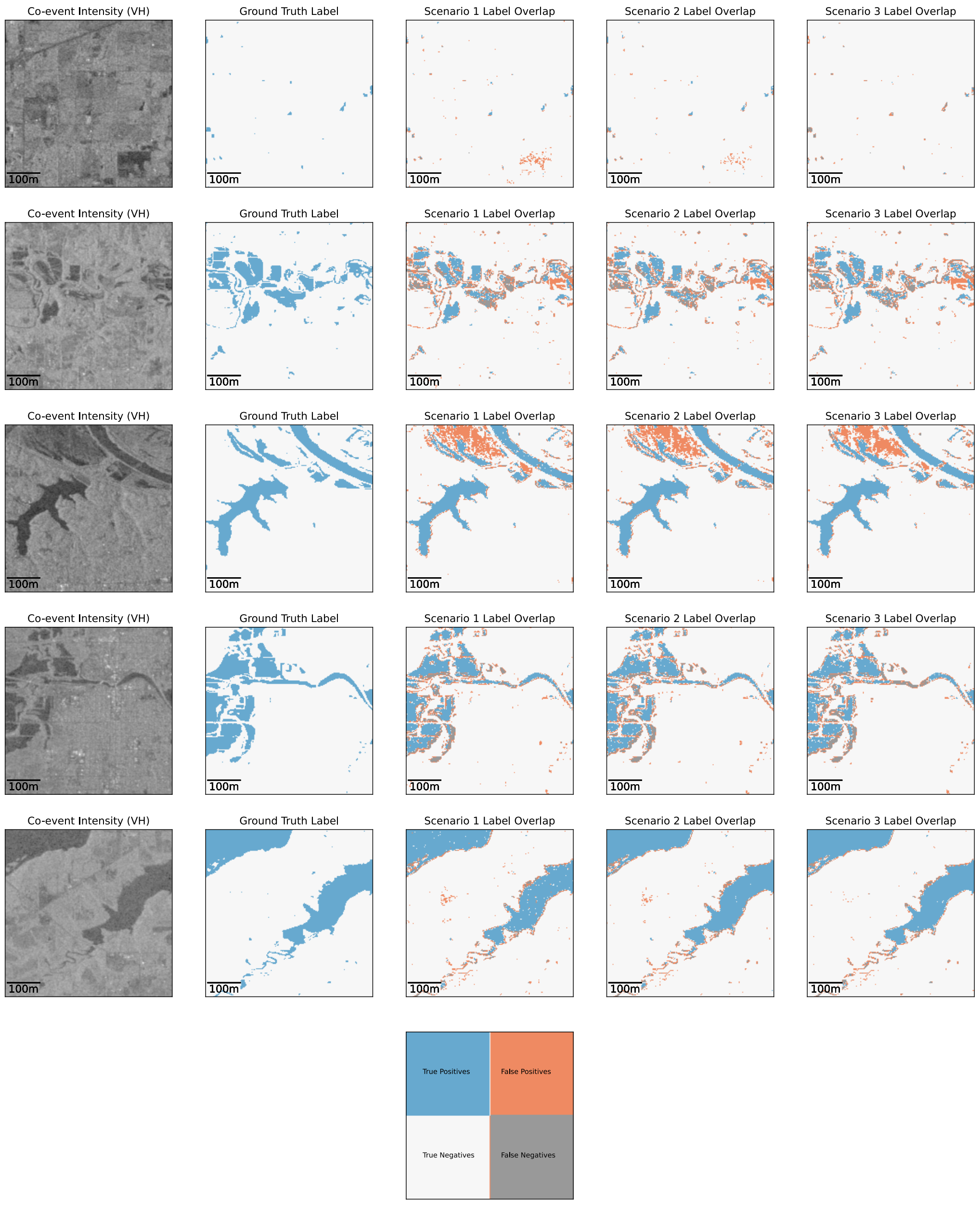


Figure 62: Label Overlap for XGBoost Models, USA

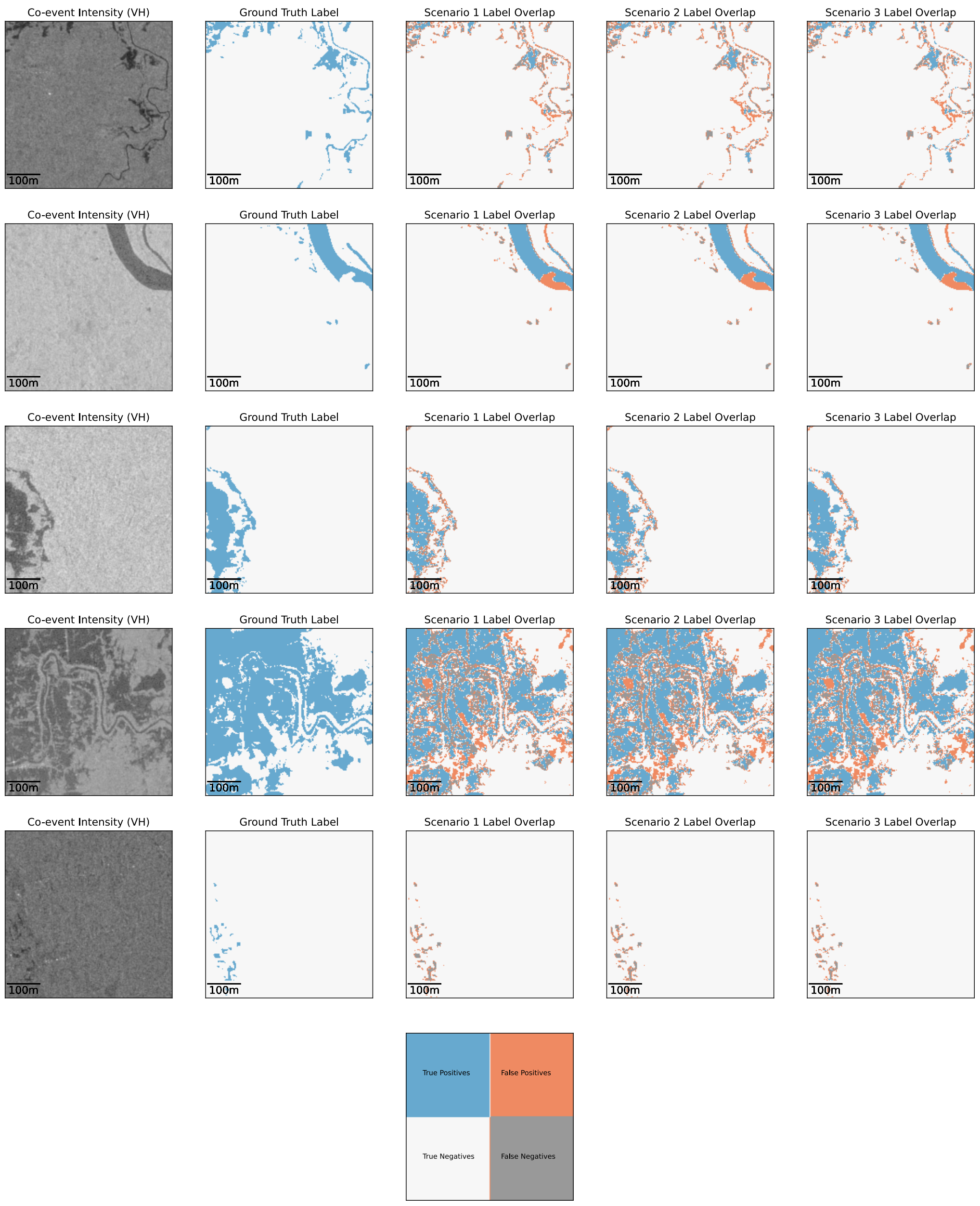


Figure 63: Label Overlap for XGBoost Models, Mekong

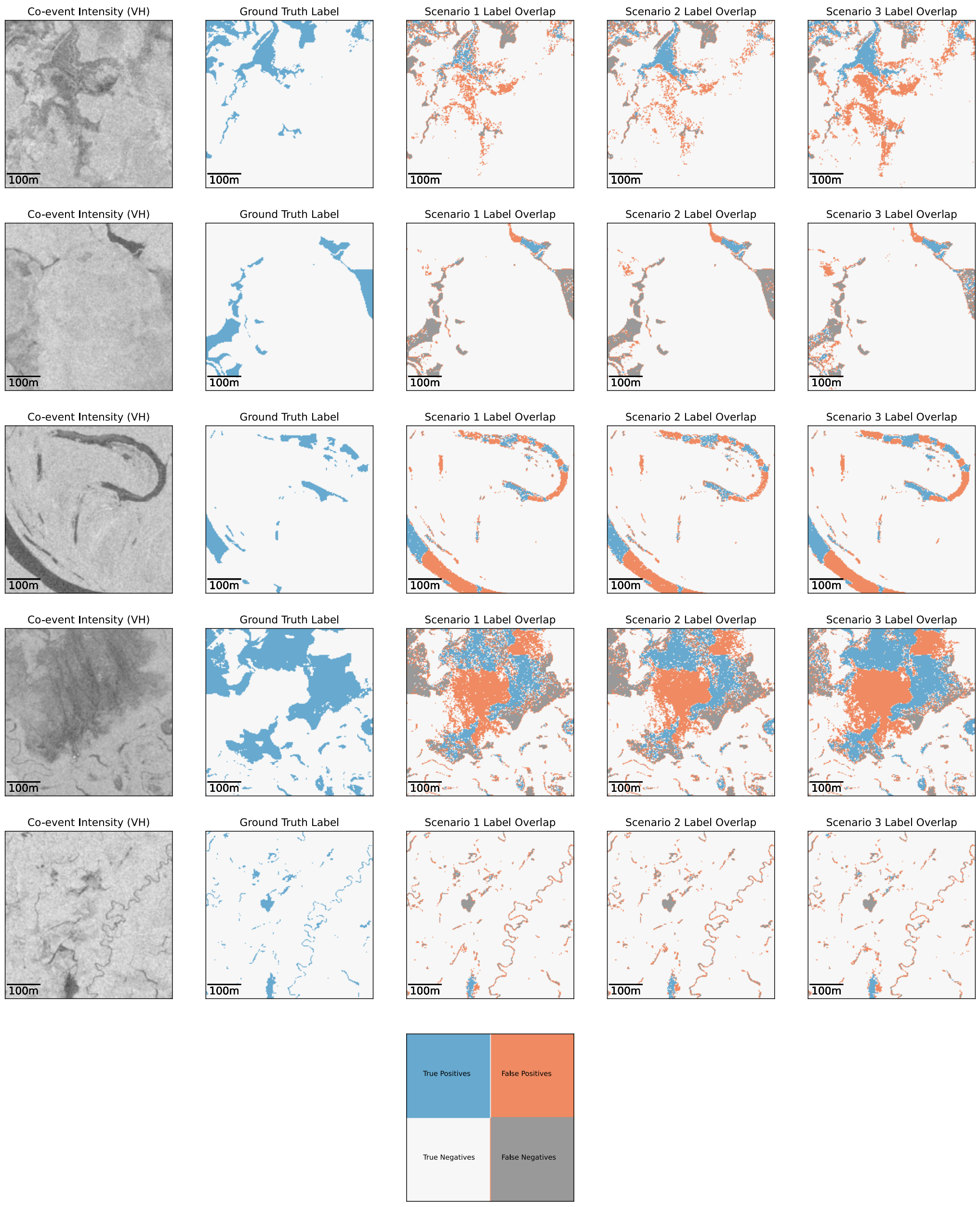


Figure 64: Label Overlap for XGBoost Models, Bolivia

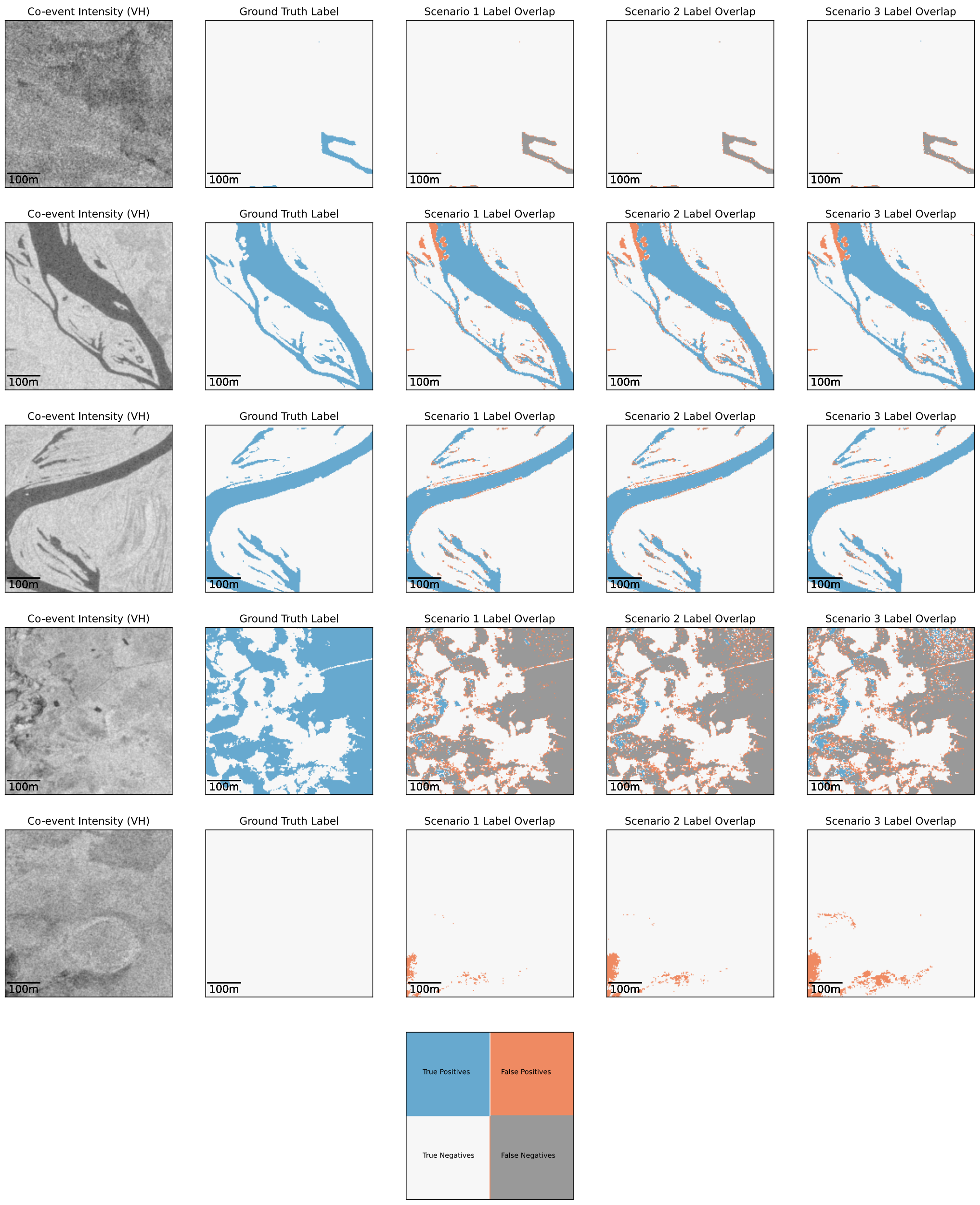


Figure 65: Label Overlap for XGBoost Models, Paraguay

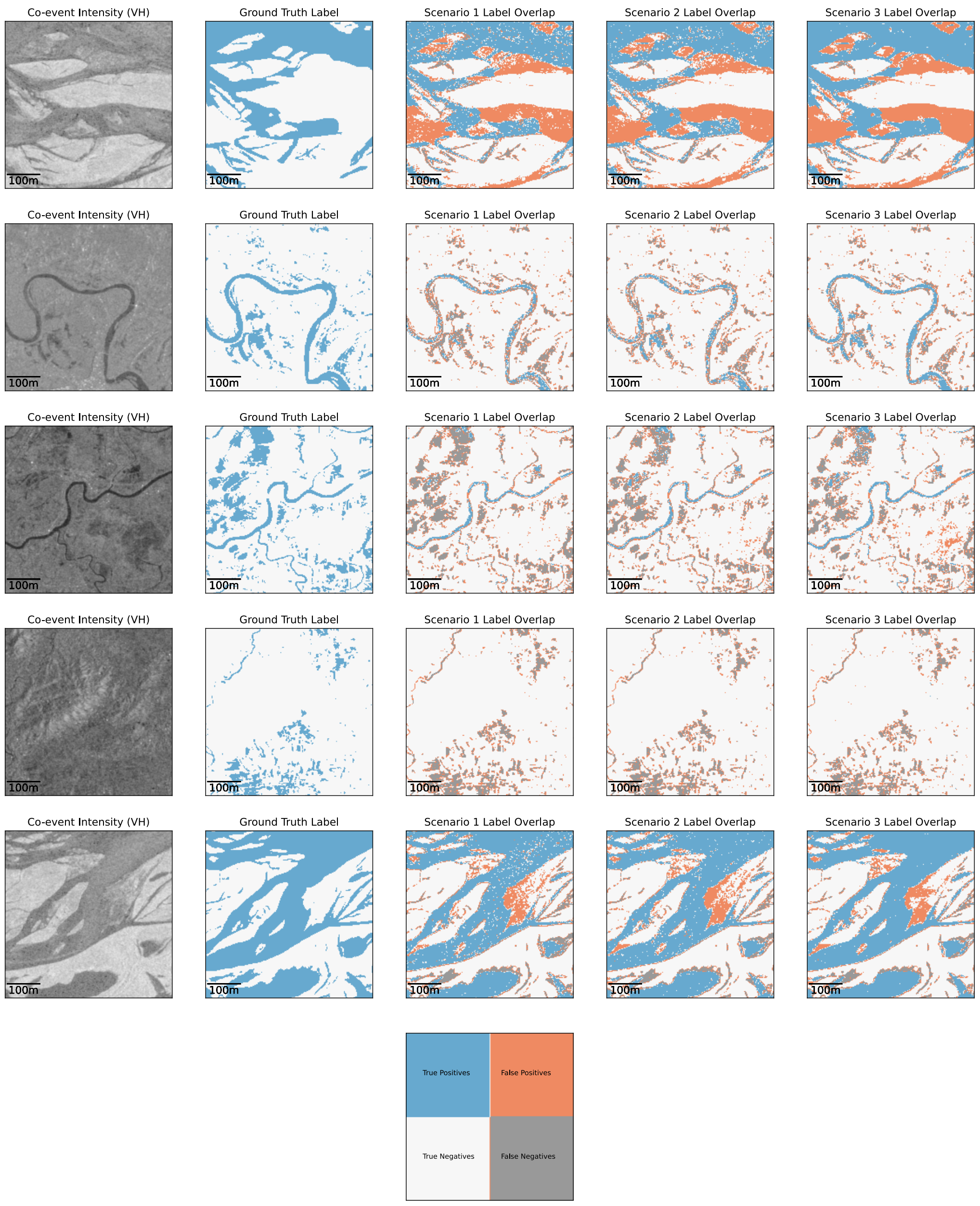


Figure 66: Label Overlap for XGBoost Models, India

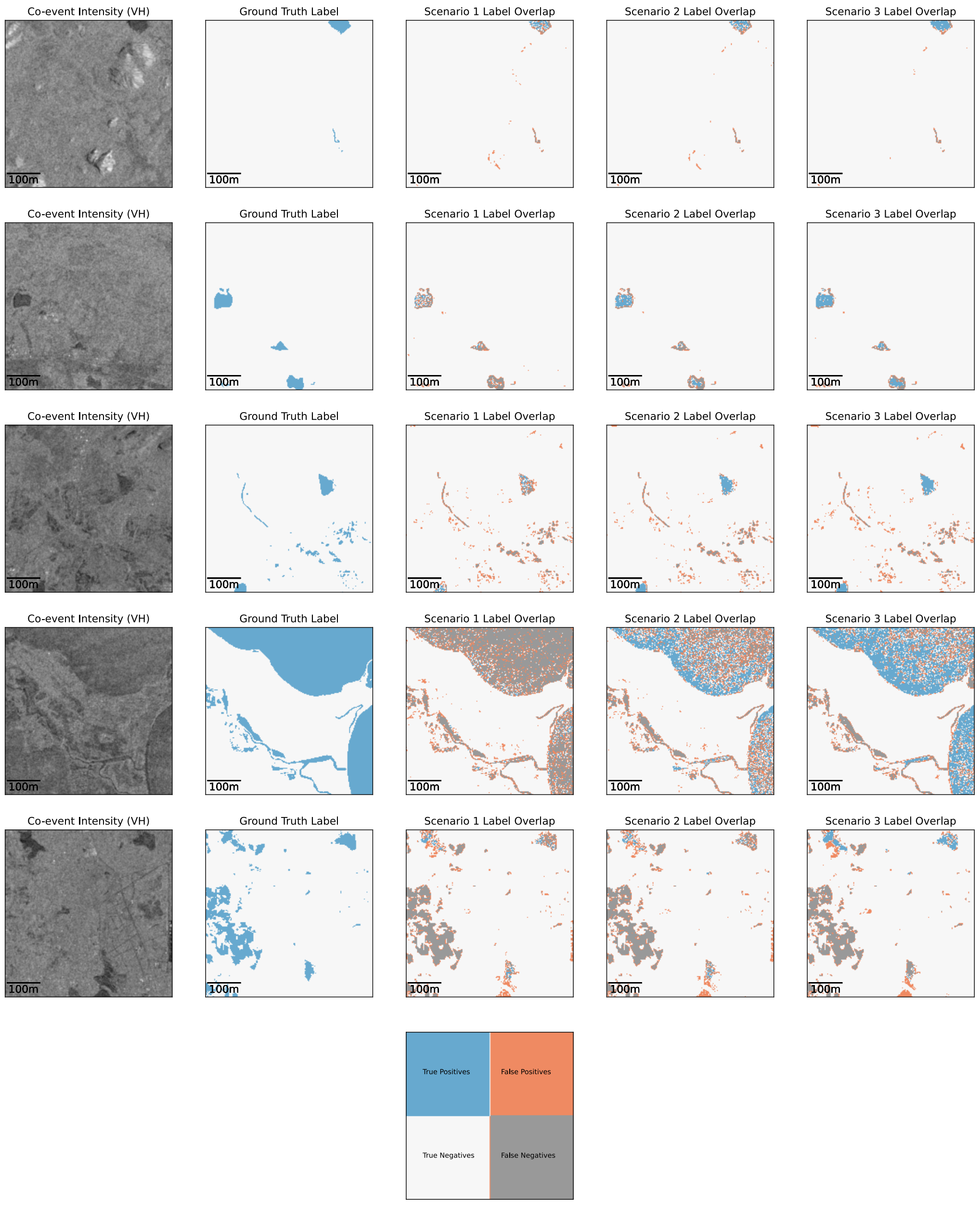


Figure 67: Label Overlap for XGBoost Models, Sri-Lanka

6 Conclusions and Recommendations

In this study, Sentinel-1 synthetic aperture radar intensity and interferometric coherence data were fused in binary classification models to improve semantic segmentation of water pixels at 10-meter spatial resolution. We considered three different model scenarios: a co-event intensity, a bi-temporal intensity, and a bi-temporal intensity and coherence model. For each scenario semantic segmentation models were trained to assess the relative improvement gained by fusing intensity and interferometry data cross-trained with Sentinel-2 derived water masks. The semantic segmentation models include an Attention U-Net model capable of accounting for the spatial distribution of the Sentinel-1 scenes and a pixel-wise XGBoost classifier. By fusing SAR intensity with interferometric coherence, we exploited the spatial decorrelation in the coherence maps during a flooding event compared to the coherence before the flooding event. The data set used in this study leverages the publicly available georeferenced Sen1Floods11 data set [1]. We augmented the co-event Sentinel-1 intensity data provided by the Sen1Floods11 team with pre-event intensity data from Google Earth Engine [14] and InSAR products produced on-demand by the Alaska Satellite Facility [20]. Our experiment results showed that fusing the Sentinel-1 intensity data with the interferometry data improves water intersection over union results by up to 3.29% with the Attention U-Net model and up to 3.51% with the XGBoost model.

Moreover, the results presented in Section 5 highlight that our Attention U-Net and XGBoost models systematically reduce the water-class false negative rate. Water recall improves by 4.33% for the Attention U-Net model and by 5.41% for the XGBoost model relative to the co-event intensity models. Reducing the water-class false negative rate is important in flood mapping applications because this means we are not incorrectly labeling non-flooded pixels. On the other hand, the co-event and bi-temporal intensity models tend to reduce the false positive rate compared to the bi-temporal intensity and coherence models. This means that with the introduction of the coherence data, our bi-temporal intensity and coherence models tend to over-estimate water pixels more than the co-event and bi-temporal intensity models. In practice, over-estimating flooded pixels is less risky than under-estimating them. This could mean that aid may be sent to a potentially flooded area as opposed to no aid being sent at all.

Lastly, our results also highlighted the XGBoost models' ability to outperform the Attention U-Net models with our generalization data set in terms of water IoU, recall, and f1-score. Convolutional neural network models are attractive because the convolutional operations are able to account for the spatial variation in the satellite images. However, the advantages gained by the CNN models come at the expense of loss of feature interpretability. In our study, raw pixel data was used as input features to our XGBoost models. This means that we could potentially assess the relative improvements gained by adding each data modality. Moreover, the training time needed to fit an XGBoost model is orders of magnitude lower than the training time required to train a CNN model. In our experiments, training the biggest XGBoost model took approximately 3 minutes compared to about 3 hours needed to train the Attention U-Net models using the same GPU. Training time can be a differentiating factor when trying to respond quickly to extreme flooding events.

6.1 Future Work

As outlined in Section 1, our study sought to answer whether it is feasible to fuse Sentinel-1 10-meter intensity and coherence data and improve semantic water segmentation for flood mapping. Generating InSAR products is extremely computationally expensive and time consuming. We have shown that exploiting the InSAR on-demand product offerings from the Alaska Satellite Facility, it is feasible to acquire the necessary data modalities needed for our data fusion approach and obtain promising results. Next, we offer potential areas of future work.

- **Model tuning** - This study focused on answering the question: can we get relative improvements by introducing the interferometry data? We were not concerned with designing models that outperformed a state of the art semantic segmentation model. Potential future work may include hyperparameter tuning to improve the water segmentation results.

We could also consider using network architectures that more fully exploit the temporal nature of the bi-temporal data for change detection. Additionally, class weights and other data augmentation techniques may be exploited to address the class imbalance observed in our training data set.

- **Urban environments** - The Sen1Floods11 data set leveraged in this study only includes non-urban flooding events. As outlined in Section 1 past studies have proven the benefits of using interferometry data for urban flood mapping. These studies have mainly used InSAR data from privately-operated SAR platforms. The Sentinel-1 mission democratizes our SAR data access. Extreme flooding events pose major risks to human livelihood. The intersection of freely available Sentinel-1 data and high performance computing InSAR processing through ASF enable a unique advantage for operationalization purposes. Future work may include augmenting our data set to include flood events over urban regions imaged with the Sentinel-1 platform.
- **Conditional models** - Another potential area of future work may include using the interferometry data as a conditioning feature for a segmentation model.

A Code Samples

This section includes code samples for our training and evaluation pipelines. A more comprehensive repository including utility modules and Jupyter notebooks are available in our GitHub repository [31].

A.1 XGBoost Training Pipeline

```python
"""
Improving Semantic Water Segmentation by Fusing Sentinel-1 Intensity and Interferometric Synthetic Aperture Radar
(InSAR) Coherence Data

Author: Ernesto Colon
The Cooper Union for the Advancement of Science and Art
Spring 2022

XGBoost Model Training
"""

# Import libraries

import sys
sys.path.append('..')
from utils import dataset_gen
import xgboost as xgb
import time

"""
Define function to train the XGBoost models in two steps or batches. The data set is large (~28GB for scenario 3) and
does not fit in GPU memory. Depending on the GPU memory size, the training pipeline may require training in more than
two stages.
"""


def xgb_batch_train(X_train, Y_train, save_fname):
    """
    Function to serialize the XGBoost training for large data sets. This function only handles two batches
    since the data set we're using can be split in half and fit in the RTX 3090's memory.

    :param X_train: 2D-ndarray with shape (num_pix, num_feat) with input features
    :param Y_train: 2D-ndarray with shape (num_pix,) with labels
    :param save_fname: string with path and filename to save the final model
    :return: None
    """

    start_time = time.time()

    # create first model instance
    model_1 = xgb.XGBClassifier(use_label_encoder=False, tree_method='gpu_hist')

    # fit first model
    model_1.fit(X_train['batch_1'], Y_train['batch_1'])

    # create second model instance
    model_2 = xgb.XGBClassifier(use_label_encoder=False, tree_method='gpu_hist')

    # fit second model
    model_2.fit(X_train['batch_2'], Y_train['batch_2'], xgb_model=model_1)

    print("--- %s seconds ---" % (time.time() - start_time))

    # Save model
    model_2.save_model(save_fname)


if __name__ == "__main__":

    ###############################################################
    #          Load previously saved dataset splits
    ###############################################################

    # Define dictionary with filepaths
    base_dir = "base_dir_path"

    train_val_test_pths = {'train_fn_df': f"{base_dir}\\train_fn_df_fname",
                           'val_fn_df': f"{base_dir}\\val_fn_df_fname",
                           'test_fn_df': f"{base_dir}\\test_fn_df_fname"}

    train_samples, val_samples, test_samples, train_size, val_size, test_size = \
        dataset_gen.xgboost_load_ds_samples(train_val_test_pths['train_fn_df'],
                                            train_val_test_pths['val_fn_df'],
                                            train_val_test_pths['test_fn_df'])

    ###############################################################
    # Create dictionaries to store the training and test data sets
    ###############################################################

    batches = ['batch_1', 'batch_2']
    scenarios = ['scenario_1', 'scenario_2', 'scenario_3']

    X_train_dict = {scenario: {} for scenario in scenarios}
    Y_train_dict = {scenario: {} for scenario in scenarios}

    X_test_dict = {scenario: {} for scenario in scenarios}
    Y_test_dict = {scenario: {} for scenario in scenarios}

    ###############################################################
    # Split the training data set into batches for sequential training
    ###############################################################

    train_split_idx_low = [0, int(len(train_samples) / 2)]
    train_split_idx_high = [int(len(train_samples) / 2), len(train_samples)]

    test_split_idx_low = [0, int(len(test_samples) / 2)]
    test_split_idx_high = [int(len(test_samples) / 2), len(test_samples)]

    ###############################################################
    # Select the scenario to be trained
    train_scenario = 1
    ###############################################################

    # logic to determine whether the current scenario includes coherence data or not
    if train_scenario == 1:
        int_flag = True
    else:
        int_flag = False
    if train_scenario == 3:
        coh_flag = True
    else:
        coh_flag = False

    ###############################################################
    #                  Training Pipeline
    ###############################################################

    current_scenario = train_scenario

    # Generate data sets for the current scenario
    for idx, batch in enumerate(batches):
        X_train_dict[f"scenario_{current_scenario}"][batch],\
        Y_train_dict[f"scenario_{current_scenario}"][batch], _, _ =\
            dataset_gen.rf_xgb_ds_generator(train_samples[train_split_idx_low[idx]: train_split_idx_high[idx]],
                                            coh_flag=coh_flag,
                                            int_flag=int_flag)

        X_test_dict[f"scenario_{current_scenario}"], Y_test_dict[f"scenario_{current_scenario}"], _, _ =\
            dataset_gen.rf_xgb_ds_generator(test_samples[test_split_idx_low[idx]: test_split_idx_high[idx]],
                                            coh_flag=coh_flag,
                                            int_flag=int_flag)

    ###############################################################
    #                   XGBoost training
    ################################################################

    # Define path and file name to save the trained model
    xgboost_model_pth = "xgboost_model_pth"
    fname = f"{xgboost_model_pth}\\scenario_{current_scenario}\\
  xgb_10m_raw_pix_feat_scen_{current_scenario}.model"

    # start the stage-wise training pipeline
    xgb_batch_train(X_train_dict[f"scenario_{current_scenario}"],
  Y_train_dict[f"scenario_{current_scenario}"], fname)
```

A.2 Attention U-Net Training Pipeline

```
"""

Improving Semantic Water Segmentation by Fusing Sentinel-1 Intensity and
    Interferometric Synthetic Aperture Radar
(InSAR) Coherence Data

**Author: Ernesto Colon**
**The Cooper Union for the Advancement of Science and Art**

#### Attention Unet-2D Model Training
"""

################################################################
#                        Import libraries
################################################################

import tensorflow as tf
import time
from utils import dataset_gen
import matplotlib.pyplot as plt
import sys
sys.path.append('..')

################################################################
# Define function to plot train and validation loss
################################################################

def plot_train_val_loss(model_history):
    """
    Function to plot training and validation loss
    """
    loss = model_history.history['loss']
    val_loss = model_history.history['val_loss']

    plt.figure()
    plt.plot(model_history.epoch, loss, 'r', label='Training loss')
    plt.plot(model_history.epoch, val_loss, 'bo', label='Validation loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss Value')
    plt.ylim([0, 1])
    plt.legend()
    plt.show()


if __name__ == "__main__":
    ################################################################
    # check that a GPU is enabled
    ################################################################

    device_name = tf.test.gpu_device_name()
    if device_name != '/device:GPU:0':
        raise SystemError('GPU device not found')
    print('Found GPU at: {}'.format(device_name))

    ################################################################
    # Load the train, validation, and test dataframes
    ################################################################

    # Define dictionary with filepaths
    base_dir = "base_dir_path"

    train_val_test_pths = {'train_fn_df': f"{base_dir}\\ds_train_split_10m
   .csv",
                           'val_fn_df': f"{base_dir}\\ds_val_split_10m.csv
   ",
                           'test_fn_df': f"{base_dir}\\ds_test_split_10m.
   csv"}

    train_val_fn_df, test_fn_df, train_size, val_size, test_size = \
        dataset_gen.unet_load_ds_df(train_val_test_pths['train_fn_df'],
                                    train_val_test_pths['val_fn_df'],
                                    train_val_test_pths['test_fn_df'])

    ################################################################
    #   We generate datasets for the following scenarios:

    #   - Scenario 1: Co-event intensity data only
    #   - Scenario 2: Pre- and co-event intensity data only
    #   - Scenario 3: Pre- and co-event intensity and coherence data
    ################################################################

    # Define dictionaries to hold the datasets - the keys will be the
   different scenarios
    X_train_dict = {}
    Y_train_dict = {}

    X_val_dict = {}
    Y_val_dict = {}

    X_test_dict = {}
    Y_test_dict = {}

    Y_pred_dict = {}

```

```python
# Define scenario number to scenario name mapping
scenario_dict = {1: 'co_event_intensity_only',
                 2: 'pre_co_event_intensity',
                 3: 'pre_co_event_int_coh'}

scenario_num_bands = {1: 2,
                      2: 4,
                      3: 6}

# Define the number of bands per scenario
num_bands_dict = {'co_event_intensity_only': 2,
                  'pre_co_event_intensity': 4,
                  'pre_co_event_int_coh': 6}

IMG_SIZE = 512

# define dictionaries to hold the datasets
train_val_samples_dict = {}
test_samples_dict = {}

# Loop through each scenario and create the tensorflow data loaders
scenarios = [1, 2, 3]

for scenario in scenarios:
    # Create the samples list given the dataframes with file paths as input
    train_val_samples_dict[f"scenario_{scenario}"], test_samples_dict[f"scenario_{scenario}"] = \
        dataset_gen.create_samples_list({'scenario': scenario_dict[scenario],
                                         'test_df': test_fn_df,
                                         'train_val_df': train_val_fn_df})

    # Create data sets dictionary
    X_train_dict[f"scenario_{scenario}"], X_val_dict[f"scenario_{scenario}"], X_test_dict[f"scenario_{scenario}"] = \
        dataset_gen.unet_ds_creation({'train_val_list': train_val_samples_dict[f"scenario_{scenario}"],
                                      'test_list': test_samples_dict[f"scenario_{scenario}"]})

    # Batch the tensorflow train, val, and test data set generators
    X_train_dict[f"scenario_{scenario}"] = \
        X_train_dict[f"scenario_{scenario}"].batch(10).prefetch(tf.data.experimental.AUTOTUNE)

    X_val_dict[f"scenario_{scenario}"] = \
        X_val_dict[f"scenario_{scenario}"].batch(10).prefetch(tf.data.experimental.AUTOTUNE)

    X_test_dict[f"scenario_{scenario}"] = X_test_dict[f"scenario_{scenario}"].batch(1)

```

```python
# ###################################################################################
# Attention U-Net Models

# For this study, we leverage the publicly available Keras UNet Collection linked below.

# https://github.com/yingkaisha/keras-unet-collection
# ###################################################################################

from keras_unet_collection import models

# create dictionary to hold the models by scenario
attn_unet_2d_models = {}

###############################################################
#       Loop through scenarios and generate the models
###############################################################

for scenario in scenarios:
    # Create models for each scenario
    print("\n*******************************************\n")
    print(f"Generating model for scenario: {scenario}")

    attn_unet_2d_models[f"scenario_{scenario}"] = models.att_unet_2d(
        (IMG_SIZE, IMG_SIZE, scenario_num_bands[scenario]),
        filter_num=[64, 128, 256, 512, 1024],
        n_labels=2,
        stack_num_down=2,
        stack_num_up=2,
        activation='ReLU',
        atten_activation='ReLU',
        attention='add',
        output_activation='Sigmoid',
        batch_norm=True,
        pool=False,
        unpool=False,
        backbone='VGG16',
        weights=None,
        freeze_backbone=False,
        freeze_batch_norm=True,
        name='attunet')

    print("*******************************************")

# unet_2d_models['scenario_3'].summary()

# %%

# Define a learning rate schedule
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(0.0001,
    decay_steps=200,
    decay_rate=0.96,
                                                                    staircase
    =True)

 # Define the optimizer
 optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule,
                                       beta_1=0.9,
                                       beta_2=0.999,
                                       epsilon=1e-07,
                                       amsgrad=False,
                                       name='Adam')

 # Create dictionary to store model training history
 attn_unet_2d_train_hist = {}

 ##############################################################
 # Scenario 1 Training- Co-event Intensity Model
 ##############################################################
 # %%

 # Compile the model
 current_scenario = 1
 attn_unet_2d_models[f"scenario_{current_scenario}"].compile(
     optimizer=optimizer,
     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
False),
     metrics=['accuracy'])

 # Start training routine
 train_start_time = time.time()

 EPOCHS = 30
 attn_unet_2d_train_hist[f"scenario_{current_scenario}"] = \
     attn_unet_2d_models[f"scenario_{current_scenario}"].fit(
         X_train_dict[f"scenario_{current_scenario}"],
         validation_data=X_val_dict[f"scenario_{current_scenario}"],
         epochs=EPOCHS)

 print("--- %s seconds ---" % (time.time() - train_start_time))

 ##############################################################
 #       Save the model weights for scenario 1
 ##############################################################

 attn_unet_2d_model_pth = "atten_unet_model_path"
 attn_unet_2d_models[f"scenario_{current_scenario}"].save_weights(
     f"{attn_unet_2d_model_pth}\\scenario_{current_scenario}"+"\\" + f"
unet2d_10m_{scenario_dict[current_scenario]}")

 # Plot training and validation loss for scenario 1

```

```
    plot_train_val_loss(attn_unet_2d_train_hist[f"scenario_{
   current_scenario}"])

    ##############################################################
    # Scenario 2 Training - Pre-event and Co-event Intensity Model
    ##############################################################

    # Compile the model
    current_scenario = 2
    attn_unet_2d_models[f"scenario_{current_scenario}"].compile(
        optimizer=optimizer,
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=
   False),
        metrics=['accuracy'])

    ##############################################################
    #                Start training routine
    ##############################################################
    train_start_time = time.time()

    EPOCHS = 30
    attn_unet_2d_train_hist[f"scenario_{current_scenario}"] = \
        attn_unet_2d_models[f"scenario_{current_scenario}"].fit(
            X_train_dict[f"scenario_{current_scenario}"],
            validation_data=X_val_dict[f"scenario_{current_scenario}"],
            epochs=EPOCHS)

    print("--- %s seconds ---" % (time.time() - train_start_time))

    ##############################################################
    #       Save the model weights for scenario 2
    ##############################################################

    attn_unet_2d_models[f"scenario_{current_scenario}"].save_weights(
        f"{attn_unet_2d_model_pth}\\scenario_{current_scenario}" + "\\" +
   f"unet2d_10m_{scenario_dict[current_scenario]}")

    # Plot training and validation loss for scenario 2

    plot_train_val_loss(attn_unet_2d_train_hist[f"scenario_{
   current_scenario}"])

    ##############################################################
    # Scenario 3 Training - Pre-event and Co-event Intensity Model
    ##############################################################

    # Compile the model
    current_scenario = 3
    attn_unet_2d_models[f"scenario_{current_scenario}"].compile(optimizer=
   optimizer,
                                                                        loss=tf.
   keras.losses.SparseCategoricalCrossentropy(

   from_logits=False),
                                                            metrics=['accuracy'])

    # Start training routine
    train_start_time = time.time()

    EPOCHS = 1
    attn_unet_2d_train_hist[f"scenario_{current_scenario}"] = \
        attn_unet_2d_models[f"scenario_{current_scenario}"].fit(
            X_train_dict[f"scenario_{current_scenario}"],
            validation_data=X_val_dict[f"scenario_{current_scenario}"],
            epochs=EPOCHS)

    print("--- %s seconds ---" % (time.time() - train_start_time))

    ##############################################################
    #       Save the model weights for scenario 3
    ##############################################################

    attn_unet_2d_models[f"scenario_{current_scenario}"].save_weights(
        f"{attn_unet_2d_model_pth}\\scenario_{current_scenario}" + "\\" + f"unet2d_10m_{scenario_dict[current_scenario]}")

    # Plot training and validation loss for scenario 3
    plot_train_val_loss(attn_unet_2d_train_hist[f"scenario_{current_scenario}"])
```

A.3 XGBoost Evaluation Pipeline

```
"""
Improving Semantic Water Segmentation by Fusing Sentinel-1 Intensity and Interferometric Synthetic Aperture Radar
(InSAR) Coherence Data

Author: Ernesto Colon
The Cooper Union for the Advancement of Science and Art**
Spring 2022

XGBoost Model Inference
"""

# Import libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import rasterio
from utils import metrics_utils
from utils import dataset_gen
from utils import general_utils
import time
import xgboost as xgb


# Define helper functions

# Create function to generate the label overlap between ground truth and predictions
def gen_lbl_overlap(y_true, y_pred):
    """
    Function to return a semantic map with a label overlap given ground truth and predicted labels
    :param y_true: ndarray with ground truth labels
    :param y_pred: ndarray with predicted labels
    :return: combined, an ndarray with 4 classes (1: true positive, 2: true negatives, 3: false positives, 4: false neg)
    """

    # allocate space to store the label overlap
    combined = np.zeros(y_pred.shape)

    # true positives are labels that are predicted as water (1)
    tp = np.logical_and(np.where(y_pred == 1, 1, 0), np.where(y_true == 1, 1, 0))

    # true negatives
    tn = np.logical_and(np.where(y_pred == 0, 1, 0), np.where(y_true == 0, 1, 0))

    # false positives are labels that were labeled as 1 but that were 0 in reality
    fp = np.logical_and(np.where(y_pred == 1, 1, 0), np.where(y_true == 0, 1, 0))

    # false negatives are labels that were labeled as 0 but were 1 in reality
    fn = np.logical_and(np.where(y_pred == 0, 1, 0), np.where(y_true == 1, 1, 0))

    # combine all classes
    combined[tp] = 1
    combined[tn] = 2
    combined[fp] = 3
    combined[fn] = 4

    return combined


##############################################################
#       Create a function to plot the label overlap
##############################################################

# Generate color maps for the labels and label overlap

from utils import general_utils

wtr_cmap = general_utils.gen_cmap(['#f7f7f7', '#67a9cf'])
ovrlp_cmap = general_utils.gen_cmap(['#67a9cf', '#f7f7f7', '#ef8a62', '#999999'])
```

```python

from mpl_toolkits.axes_grid1.anchored_artists import AnchoredSizeBar
import matplotlib.font_manager as fm

fontprops = fm.FontProperties(size=15)


def display_lbl_overlap(y_true, lbl_overlap, x_test, num_plot, region,
    indices=None):
    """
    Function to display the label overlap
    :param y_true: ndarray with ground truth labels
    :param lbl_overlap: ndarray with label overlap
    :param x_test: ndarray with Sentinel-1 co-event intensity (VH) raster
    :param num_plot: integer, number of scenes to display
    :param region: string with geographical region to display
    :param indices: list of integer with indices to plot from the entire
    data set
    :return: matplotlib figure handle
    """

    fontprops = fm.FontProperties(size=12)

    num_col = 5
    fig, ax = plt.subplots(num_plot + 1, num_col, figsize=(20, 5 *
    num_plot))
    ax = ax.ravel()

    if indices == None:
        indices = range(num_plot)

    for idx, raster in enumerate(indices):

        ax[num_col * idx].imshow(x_test[region]['scenario_1_hand_lbl'][
    raster, :, :, 0], cmap='gray')
        ax[num_col * idx].set_title(f'Co-event Intensity (VH)')

        # plot ground truth
        ax[num_col * idx + 1].imshow(y_true[region]['scenario_3_hand_lbl'
    ][raster, :, :], cmap=wtr_cmap)
        # ax[num_col * idx + 1].set_title(f'Ground Truth Label, index: {
    raster}')
        ax[num_col * idx + 1].set_title(f'Ground Truth Label')

        # plot scenario 1
        ax[num_col * idx + 2].imshow(lbl_overlap[region]['
    scenario_1_hand_lbl'][raster, :, :], cmap=ovrlp_cmap)
        ax[num_col * idx + 2].set_title('Scenario 1 Label Overlap')

        # plot scenario 2
        ax[num_col * idx + 3].imshow(lbl_overlap[region]['
    scenario_2_hand_lbl'][raster, :, :], cmap=ovrlp_cmap)
        ax[num_col * idx + 3].set_title('Scenario 2 Label Overlap')

        # plot scenario 3
        ax[num_col * idx + 4].imshow(lbl_overlap[region]['
   scenario_3_hand_lbl'][raster, :, :], cmap=ovrlp_cmap)
        ax[num_col * idx + 4].set_title('Scenario 3 Label Overlap')

        for axis in ax[: num_col * num_plot]:
            scalebar = AnchoredSizeBar(
                axis.transData,
                100,
                '100m',
                'lower left',
                pad=0.1,
                color='black',
                frameon=False,
                size_vertical=1,
                fontproperties=fontprops)

            axis.add_artist(scalebar)
            axis.set_yticks([])
            axis.set_xticks([]);

    # Create legend
    checkerboard = np.zeros((512, 512))
    checkerboard[0:256, 0:256] = 1
    checkerboard[256:, 0:256] = 2
    checkerboard[0:256, 256:] = 3
    checkerboard[256:, 256:] = 4

    ax[num_col * idx + 4 + 3].imshow(checkerboard, cmap=ovrlp_cmap)
    ax[num_col * idx + 4 + 3].text(50, 128, "True Positives", fontsize=8.)
   ;
    ax[num_col * idx + 4 + 3].text(50, 384, "True Negatives", fontsize=8.)
   ;
    ax[num_col * idx + 4 + 3].text(290, 128, "False Positives", fontsize
   =8.);
    ax[num_col * idx + 4 + 3].text(290, 384, "False Negatives", fontsize
   =8.);
    ax[num_col * idx + 4 + 3].set_yticks([])
    ax[num_col * idx + 4 + 3].set_xticks([]);

    ind_to_del = [1, 2, 4, 5]
    for ind in ind_to_del:
        fig.delaxes(ax[num_col * idx + 4 + ind])

    return fig


if __name__ == "__main__":

    ###############################################################
    #            Load previously saved dataset splits
    ###############################################################

    # Define dictionary with filepaths
    base_dir = "base_dir"

    train_val_test_pths = {'train_fn_df': f"{base_dir}\\ds_train_split_10m
.csv",
                           'val_fn_df': f"{base_dir}\\ds_val_split_10m.csv
",
                           'test_fn_df': f"{base_dir}\\ds_test_split_10m.
csv"}

    train_samples, val_samples, test_samples, train_size, val_size,
test_size = \
        dataset_gen.xgboost_load_ds_samples(train_val_test_pths['
train_fn_df'],
                                            train_val_test_pths['val_fn_df
'],
                                            train_val_test_pths['
test_fn_df'])

    ###############################################################
    # Define category names and a color mapping for semantic segmentation
    ###############################################################

    # Define category names
    tgt_cat_names = {
        0: 'Not water',
        1: 'Water'
    }

    # Define the colors per category
    wtr_clrs_hex = ['#f7f7f7', '#67a9cf']

    # Generate the labels colormap
    wtr_cmap = general_utils.gen_cmap(wtr_clrs_hex)

    # %% md
    ###############################################################
    # Generate data sets for inference
    ###############################################################
    """
    We generate datasets for the following scenarios:

    - Scenario 1: Co-event intensity data only
    - Scenario 2: Pre- and co-event intensity data only
    - Scenario 3: Pre- and co-event intensity and coherence data
    """

    # Define dictionaries to hold the datasets - the keys will be the
different scenarios
    X_train_dict = {}
    Y_train_dict = {}

    X_test_dict = {}
    Y_test_dict = {}

    Y_pred_dict = {}

    scenarios = ['scenario_1', 'scenario_2', 'scenario_3']

    # Loop through each scenario and generate / load the data sets to memory
    for scenario in scenarios:
        # logic to determine whether the current scenario includes coherence data or not
        if scenario == 'scenario_1':
            int_flag = True
        else:
            int_flag = False
        if scenario == 'scenario_3':
            coh_flag = True
        else:
            coh_flag = False

        # generate data set
        X_test_dict[scenario], Y_test_dict[scenario], _, _ = \
            dataset_gen.rf_xgb_ds_generator(test_samples, coh_flag=coh_flag, int_flag=int_flag)

    ##############################################################
    # Gather dataset parameters we'll need later on
    ##############################################################

    num_train_samp = len(train_samples)
    img_size = 512

    num_feat_dict = {'scenario_1': 2,
                     'scenario_2': 4,
                     'scenario_3': 6,
                     'scenario_1_hand_lbl': 2,
                     'scenario_2_hand_lbl': 4,
                     'scenario_3_hand_lbl': 6}

    ##############################################################
    # Hand Labeled Dataset
    ##############################################################

    # load hand label dataset
    hand_lbl_ds_pth = "hand_lbl_ds_pth"
    hand_lbl_ds_fname = f"{hand_lbl_ds_pth}hand_lbl_ds_10m_res.csv"

    # load csv file to dataframe
    df_hand_lbl_samples = pd.read_csv(hand_lbl_ds_fname)

    # loop through df and append sample paths to a list
    hand_lbl_samples = list()

    for idx, row in df_hand_lbl_samples.iterrows():
        hand_lbl_samples.append((row['s1'],
                                 row['pre_event_grd'],
                                row['pre_event_coh'],
                                row['co_event_coh'],
                                row['hand_lbl']))

    hand_lbl_scenarios = [f"{scenario}_hand_lbl" for scenario in scenarios
]

    # Generate hand-labeled data set
    for scenario in hand_lbl_scenarios:
        # logic to determine whether the current scenario includes
coherence data or not
        if scenario == 'scenario_1_hand_lbl':
            int_flag = True
        else:
            int_flag = False
        if scenario == 'scenario_3_hand_lbl':
            coh_flag = True
        else:
            coh_flag = False

        X_test_dict[scenario], Y_test_dict[scenario], _, _ = \
            dataset_gen.rf_xgb_ds_generator(hand_lbl_samples, coh_flag=
coh_flag, int_flag=int_flag)

    ##############################################################
    #               Visualize some image-target pairs
    ##############################################################

    # Load a number of scenes
    scenes_list = list()

    num_scenes = 5

    for idx in range(num_scenes):
        temp_list = list()

        for j in range(len(train_samples[idx])):

            # Open rasters with rasterio
            with rasterio.open(train_samples[idx][j]) as src:
                src = src.read()
                if j == 4:  # account for labels and map the not-valid
pixels to the not-water category
                    src = np.where(src == -1, 0, src)
                temp_list.append(src)

        scenes_list.append(temp_list)

    ##############################################################
    #                    Display the scenes
    ##############################################################

    num_col = 7

```

```
fig, ax = plt.subplots(num_scenes, num_col, figsize=(80, num_scenes *
10))
ax = ax.ravel()

for i in range(len(scenes_list)):
    # s1 co-event
    ax[num_col * i].imshow(scenes_list[i][0][0, :, :], cmap='gray')
    ax[num_col * i].set_title('S1 co-event VH')

    ax[num_col * i + 1].imshow(scenes_list[i][0][1, :, :], cmap='gray'
)
    ax[num_col * i + 1].set_title('S1 co-event VV')

    # s1 pre-event
    ax[num_col * i + 2].imshow(scenes_list[i][1][0, :, :], cmap='gray'
)
    ax[num_col * i + 2].set_title('S1 pre-event VH')

    ax[num_col * i + 3].imshow(scenes_list[i][1][1, :, :], cmap='gray'
)
    ax[num_col * i + 3].set_title('S1 pre-event VV')

    # pre-event coh
    ax[num_col * i + 4].imshow(scenes_list[i][2][0, :, :], cmap='gray'
)
    ax[num_col * i + 4].set_title('Pre-event coherence')

    # co-event coh
    ax[num_col * i + 5].imshow(scenes_list[i][3][0, :, :], cmap='gray'
)
    ax[num_col * i + 5].set_title('Co-event coherence')

    # s2 label
    ax[num_col * i + 6].imshow(scenes_list[i][4][0, :, :], cmap=
wtr_cmap)
    ax[num_col * i + 6].set_title('S2 Label')

for ax in ax:
    ax.set_yticks([])
    ax.set_xticks([]);

###############################################################
#                       XGBoost Models
###############################################################

# Load previously trained models

xgb_models_dir = {'scenario_1': "model_scen_1_pth",
                  'scenario_2': "model_scen_2_pth",
                  'scenario_3': "model_scen_3_pth"}

xgb_classifier_models = {}

for scenario in xgb_models_dir.keys():
    xgb_classifier_models[scenario] = xgb.XGBClassifier(use_label_encoder=False,
                                                                                                tree_method='gpu_hist')

    print(f"Loading model weights for scenario: {scenario}...")
    xgb_classifier_models[scenario].load_model(xgb_models_dir[scenario])

################################################################
#        Make predictions with the XGBoost models
################################################################

"""
Notes

The held-out test set is comprised of Sentinel-2 weak labels from the Sen1Floods11 data set.

The hand-labeled data set is also provided by the Sen1Floods11 data set, and provides an independent data set not
used during training.
"""

# predict on the held-out test dataset

start_time = time.time()

# loop through each scenario
for scenario in xgb_classifier_models.keys():
    Y_pred_dict[scenario] = xgb_classifier_models[scenario].predict(X_test_dict[scenario])

# Predict on the hand-labeled test dataset

for scenario in scenarios:
    Y_pred_dict[f"{scenario}_hand_lbl"] = xgb_classifier_models[scenario].predict(
        X_test_dict[f"{scenario}_hand_lbl"])

print(f"Inference took: {time.time() - start_time} seconds")

################################################################
#                    Compute Metrics
################################################################

"""
For metrics, we compute:

- Overall accuracy
- Mean intersection over union, mIoU
- Jaccard score
- Water precision
- Water recall
- Water f1-score
- Not-Water precision
- Not-Water recall
- Not-Water f1-score

"""

##############################################################
#                 Held-out Test Dataset
##############################################################

start_time = time.time()

summary_df = metrics_utils.summary_report(Y_test_dict, Y_pred_dict)

print(f"Process took: {time.time() - start_time} seconds")

# save summary to csv file
xgboost_summ_pth = "xgboost_summ_pth"
fname = "xgboost_summary_stats.csv"
summary_df.to_csv(f"{xgboost_summ_pth}\\{fname}")

##############################################################
#    Computing IoU per class (i.e., water and not-water)
##############################################################

miou_per_class = metrics_utils.miou_per_class(Y_test_dict, Y_pred_dict
)

# save to csv file
mIou_fname = "xgboost_10m_mIoU_per_class_stats.csv"
miou_per_class.to_csv(f"{xgboost_summ_pth}\\{mIou_fname}")

##############################################################
# Testing Models Ability to Generalize
##############################################################

# We use data over the Sri-Lanka region (both weakly labeled as well
as hand-labeled) to
# test the models' ability to generalize

##############################################################
# Generate generalization dataset
##############################################################

# create a list with all regions for both the held-out test set and
the hand-labeled test set
regions = ['USA', 'Mekong', 'Colombia', 'Paraguay', 'India', 'Bolivia'
]
regions_w_hand_lbl = [region for region in regions if region != "
Colombia"]

generalization_ds_pth = "generalization_ds_pth"

```

```python
    # Create empty list to store the samples' path
    gen_test_samples = []

    # Grab the number of samples in the data set
    gen_test_fn_df = pd.read_csv(generalization_ds_pth)

    for idx, row in gen_test_fn_df.iterrows():
        gen_test_samples.append(
            (row['s1'], row['pre_event_grd'], row['pre_event_coh'], row['co_event_coh'], row['s2_lbl']))

    num_gen_samp = len(gen_test_samples)

    # create dictionaries to store the data sets
    gener_X_test_dict = dict()
    gener_Y_test_dict = dict()

    # Loop through each scenario
    for scenario in scenarios:
        # logic to determine whether the current scenario includes coherence data or not
        if scenario == 'scenario_1':
            int_flag = True
        else:
            int_flag = False
        if scenario == 'scenario_3':
            coh_flag = True
        else:
            coh_flag = False

        gener_X_test_dict[scenario], gener_Y_test_dict[scenario], _, _ = dataset_gen.rf_xgb_ds_generator(
            gen_test_samples, coh_flag=coh_flag, int_flag=int_flag)

    ##############################################################
    #       Hand-Labeled Generalization Data Set
    ##############################################################

    # load hand label dataset
    gen_hand_lbl_ds_pth = "gen_hand_lbl_ds_pth"

    # read hand-labeled data set into dataframe
    gen_df_hand_lbl_samples = pd.read_csv(gen_hand_lbl_ds_pth)

    # create dict to store the data set
    gen_hand_samples_by_region_dict = {}

    # For now, we only have Sri-Lanka as the generalization region
    regions = ['Sri-Lanka']

    for region in regions:
        # temp list to store file paths
        pths = list()

        # pluck the test sample paths by region
        test_pth_region = gen_df_hand_lbl_samples[gen_df_hand_lbl_samples.s1.str.contains(region)]

        for idx, row in test_pth_region.iterrows():
            pths.append((row['s1'], row['pre_event_grd'], row['pre_event_coh'], row['co_event_coh'], row['hand_lbl']))

        gen_hand_samples_by_region_dict[region] = pths

    # Generate hand-labeled generalization test dataset

    for scenario in hand_lbl_scenarios:
        # logic to determine whether the current scenario includes coherence data or not
        if scenario == 'scenario_1_hand_lbl':
            int_flag = True
        else:
            int_flag = False
        if scenario == 'scenario_3_hand_lbl':
            coh_flag = True
        else:
            coh_flag = False

        gener_X_test_dict[scenario], gener_Y_test_dict[scenario], _, _ = dataset_gen.rf_xgb_ds_generator(
            gen_hand_samples_by_region_dict['Sri-Lanka'], coh_flag=coh_flag, int_flag=int_flag)

    ##############################################################
    # Make Predictions on the generalization data set
    ##############################################################

    start_time = time.time()

    gener_Y_pred_dict = dict()

    for scenario in xgb_classifier_models.keys():
        gener_Y_pred_dict[scenario] = xgb_classifier_models[scenario].predict(gener_X_test_dict[scenario])

    # Predict on the hand-labeled test dataset

    for scenario in scenarios:
        gener_Y_pred_dict[f"{scenario}_hand_lbl"] = xgb_classifier_models[scenario].predict(
            gener_X_test_dict[f"{scenario}_hand_lbl"])

    print(f"Inference took: {time.time() - start_time} seconds")

    ##############################################################
    #   Compute metrics for generalization data set
    ##############################################################

```

```python
    start_time = time.time()

    gener_summary_df = metrics_utils.summary_report(gener_Y_test_dict,
   gener_Y_pred_dict)

    print(f"Process took: {time.time() - start_time} seconds")

    # save the metrics to a csv file for later recall
    gener_summ_fname = "xgboost_10m_generalization_stats.csv"
    gener_summary_df.to_csv(f"{xgboost_summ_pth}\\{gener_summ_fname}")

    ##############################################################
    #   Compute IoU per class for generalization dataset
    ##############################################################

    gener_miou_per_class = metrics_utils.miou_per_class(gener_Y_test_dict,
    gener_Y_pred_dict)

    # save to csv
    gener_miou_fname = "xgboost_10m_generalization_mIoU_stats.csv"
    gener_miou_per_class.to_csv(f"{xgboost_summ_pth}\\{gener_miou_fname}")

    ##############################################################
    # Making Inferences Aggregated by Geographical Region
    ##############################################################

    # Read csv file with the test filepaths
    test_fn_df = pd.read_csv(train_val_test_pths['test_fn_df'])

    test_samples_by_region_dict = {}
    regions = ['USA', 'Mekong', 'Colombia', 'Paraguay', 'India', 'Bolivia'
   ]

    for region in regions:
        pths = list()

        # pluck the test sample paths by region
        test_pth_region = test_fn_df[test_fn_df.s1.str.contains(region)]

        for idx, row in test_pth_region.iterrows():
            pths.append((row['s1'], row['pre_event_grd'], row['
   pre_event_coh'], row['co_event_coh'], row['s2_lbl']))

        test_samples_by_region_dict[region] = pths

    # Generate the data sets per region
    all_scenarios = scenarios + hand_lbl_scenarios

    # Create schemas for the data sets
    X_test_ds_region_dict = {region: {} for region in regions}
    Y_test_ds_region_dict = {region: {} for region in regions}

    for region in regions:
        # Scenario 1
        X_test_ds_region_dict[region]['scenario_1'], Y_test_ds_region_dict[region]['scenario_1'], _, _ = \
            dataset_gen.rf_xgb_ds_generator(test_samples_by_region_dict[region], coh_flag=False, int_flag=True)

        # Scenario 2
        X_test_ds_region_dict[region]['scenario_2'], Y_test_ds_region_dict[region]['scenario_2'], _, _ = \
            dataset_gen.rf_xgb_ds_generator(test_samples_by_region_dict[region], coh_flag=False, int_flag=False)

        # Scenario 3
        X_test_ds_region_dict[region]['scenario_3'], Y_test_ds_region_dict[region]['scenario_3'], _, _ = \
            dataset_gen.rf_xgb_ds_generator(test_samples_by_region_dict[region], coh_flag=True, int_flag=False)

    ################################################################
    #                  Make inferences by region
    ################################################################

    start_time = time.time()

    Y_pred_region_dict = {region: {} for region in regions}

    for scenario in scenarios:

        for region in regions:
            Y_pred_region_dict[region][scenario] = \
                xgb_classifier_models[scenario].predict(X_test_ds_region_dict[region][scenario])

    print(f"Inference took: {time.time() - start_time} seconds")

    ################################################################
    # Compute predictions on hand-labeled dataset aggregated by region
    ################################################################

    # Note: Colombia does not have hand-labeled chips**

    hand_lbl_samples_region_dict = {}

    # Colombia does not have any hand labels
    regions = ['USA', 'Mekong', 'Paraguay', 'India', 'Bolivia']

    for region in regions:
        pths = list()

        # pluck the test sample paths by region
        test_pth_region = df_hand_lbl_samples[df_hand_lbl_samples.s1.str.contains(region)]

        for idx, row in test_pth_region.iterrows():
            pths.append((row['s1'], row['pre_event_grd'], row['
```

```python
pre_event_coh'], row['co_event_coh'], row['hand_lbl']))

        hand_lbl_samples_region_dict[region] = pths

    for region in regions:
        # Scenario 2
        X_test_ds_region_dict[region]['scenario_1_hand_lbl'], Y_test_ds_region_dict[region][
            'scenario_1_hand_lbl'], _, _ = \
            dataset_gen.rf_xgb_ds_generator(hand_lbl_samples_region_dict[region], coh_flag=False, int_flag=True)

        # Scenario 4
        X_test_ds_region_dict[region]['scenario_2_hand_lbl'], Y_test_ds_region_dict[region][
            'scenario_2_hand_lbl'], _, _ = \
            dataset_gen.rf_xgb_ds_generator(hand_lbl_samples_region_dict[region], coh_flag=False, int_flag=False)

        # Scenario 5
        X_test_ds_region_dict[region]['scenario_3_hand_lbl'], Y_test_ds_region_dict[region][
            'scenario_3_hand_lbl'], _, _ = \
            dataset_gen.rf_xgb_ds_generator(hand_lbl_samples_region_dict[region], coh_flag=True, int_flag=False)

    ###############################################################
    # Make inferences with the hand-labeled dataset aggregated by region
    ###############################################################

    start_time = time.time()

    for scenario in scenarios:

        for region in regions:
            Y_pred_region_dict[region][f'{scenario}_hand_lbl'] = \
                xgb_classifier_models[scenario].predict(X_test_ds_region_dict[region][f'{scenario}_hand_lbl'])

    print(f"Inference took: {time.time() - start_time} seconds")

    ###############################################################
    #       Generate prediction summaries by region
    ###############################################################

    start_time = time.time()

    regions = ['USA', 'Mekong', 'Colombia', 'Paraguay', 'India', 'Bolivia']

    xgboost_summ_pth = "xgboost_summ_pth"

    summary_by_region = {}

    for region in regions:
        print(f"Region: {region}\n\n")
        summary_by_region[region] = metrics_utils.summary_report(Y_test_ds_region_dict[region],
                                                                 Y_pred_region_dict[region])
        print("\n\n")

        # save to csv
        summary_by_region[region].to_csv(f"{xgboost_summ_pth}\\{region}_summary_stats.csv")

    print(f"Process took: {time.time() - start_time} seconds")

    ##############################################################
    # Compute IoU per class aggregated by region
    ##############################################################

    # Create dict to store the IoU metrics by region
    regional_miou_per_class = {}

    for region in regions:
        print(f"Region: {region}\n\n")

        regional_miou_per_class[region] = metrics_utils.miou_per_class(Y_test_ds_region_dict[region],
                                                                       Y_pred_region_dict[region])

        print("\n\n")

        # save to csv
        regional_miou_per_class[region].to_csv(f"{xgboost_summ_pth}\\{region}_mIoU_stats.csv")

    ##############################################################
    #       Generate labels and label overlap by region
    ##############################################################

    # Merge the generalization data set with the rest of the data sets
    Y_pred_region_dict['Sri-Lanka'] = gener_Y_pred_dict

    Y_test_ds_region_dict['Sri-Lanka'] = gener_Y_test_dict

    X_test_ds_region_dict['Sri-Lanka'] = gener_X_test_dict

    ##############################################################
    #       Reshape predictions for visualization**
    ##############################################################

    all_regions = list(Y_pred_region_dict.keys())
    all_regions_hand_lbl = [region for region in all_regions if region != 'Colombia']

    # Create dicts to store the ground truth, predicted labels and the intensity rasters for visualization
    Y_pred_hand_lbl_by_region = {}
    Y_true_hand_lbl_by_region = {}
    X_test_hand_lbl_by_region = {}

    # scenarios to pluck
    scen_to_pluck = ['scenario_1_hand_lbl', 'scenario_2_hand_lbl', 'scenario_3_hand_lbl']

    # Create schema to store the predictions and test data
    Y_pred_hand_lbl_by_region = {region: {scen: [] for scen in scen_to_pluck} for region in all_regions_hand_lbl}
    Y_true_hand_lbl_by_region = {region: {scen: [] for scen in scen_to_pluck} for region in all_regions_hand_lbl}
    X_test_hand_lbl_by_region = {region: {scen: [] for scen in scen_to_pluck} for region in all_regions_hand_lbl}

    ################################################################
    #       Copy the predictions and the ground truth labels
    ################################################################

    for region in all_regions_hand_lbl:
        for scen in scen_to_pluck:
            try:
                Y_pred_hand_lbl_by_region[region][scen] = Y_pred_region_dict[region][scen].copy()

                Y_true_hand_lbl_by_region[region][scen] = Y_test_ds_region_dict[region][scen].copy()

                X_test_hand_lbl_by_region[region][scen] = X_test_ds_region_dict[region][scen].copy()
            except:
                continue

    ################################################################
    #            Compute label overlap by region
    ################################################################

    lbl_ovrlap_by_region = {region: {scen: [] for scen in scen_to_pluck} for region in all_regions_hand_lbl}

    for region in all_regions_hand_lbl:
        for scen in scen_to_pluck:
            lbl_ovrlap_by_region[region][scen] = \
                np.reshape(gen_lbl_overlap(
                    Y_true_hand_lbl_by_region[region][scen],
                    Y_pred_hand_lbl_by_region[region][scen]),
                    (-1, img_size, img_size))

    ################################################################
    #     Reshape ground truth and display the label overlap
    ################################################################
```

```
 for region in all_regions_hand_lbl:
     print(region)
     for scen in scen_to_pluck:
         Y_pred_hand_lbl_by_region[region][scen] = np.reshape(
Y_pred_hand_lbl_by_region[region][scen],
                                                              (-1,
img_size, img_size))

         Y_true_hand_lbl_by_region[region][scen] = np.reshape(
Y_true_hand_lbl_by_region[region][scen],
                                                              (-1,
img_size, img_size))

         X_test_hand_lbl_by_region[region][scen] = np.reshape(
X_test_hand_lbl_by_region[region][scen],
                                                              (-1,
img_size, img_size, num_feat_dict[scen]))

 ###############################################################
 #                Label Overlap for Region: USA
 ###############################################################

 ovrlp_lbl_pth = "ovrlp_lbl_pth"
 region = "USA"

 idx_USA = [1, 3, 5, 8, 22]
 fig_USA = display_lbl_overlap(Y_true_hand_lbl_by_region,
                               lbl_ovrlap_by_region,
                               X_test_hand_lbl_by_region,
                               num_plot=len(idx_USA),
                               region='USA',
                               indices=idx_USA)

 # save
 # fig_USA.savefig(fname)

 ###############################################################
 # Label Overlap for Region: Mekong
 ###############################################################

 region = "Mekong"
 idx_Mekong = [1, 2, 5, 7, 8]
 fig_Mekong = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Mekong),
                                  region=region,
                                  indices=idx_Mekong)

 # fname = f"{ovrlp_lbl_pth}\\{region}_lbl_ovrlp.pdf"
 # fig_Mekong.savefig(fname)

 ###############################################################
```

```
    # Label Overlap for Region: Bolivia
    ################################################################

    idx_Bolivia = [1, 2, 3, 4, 5]
    region = "Bolivia"
    fig_Bol = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Bolivia),
                                  region=region,
                                  indices=idx_Bolivia)

    # fname = f"{ovrlp_lbl_pth}\\{region}_lbl_ovrlp.pdf"
    # fig_Bol.savefig(fname)

    ################################################################
    # Label Overlap for Region: Paraguay
    ################################################################

    region = 'Paraguay'
    idx_Paraguay = [0, 1, 2, 6, 7]
    fig_Par = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Paraguay),
                                  region=region,
                                  indices=idx_Paraguay)

    # fname = f"{ovrlp_lbl_pth}\\{region}_lbl_ovrlp.pdf"
    # fig_Par.savefig(fname)

    ################################################################
    # Label Overlap for Region: India
    ################################################################

    region = "India"
    idx_India = [0, 2, 4, 6, 23]
    fig_Ind = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_India),
                                  region=region,
                                  indices=idx_India)

    # fname = f"{ovrlp_lbl_pth}\\{region}_lbl_ovrlp.pdf"
    # fig_Ind.savefig(fname)

    ################################################################
    # Label Overlap for Region: Sri-Lanka
    ################################################################

    region = "Sri-Lanka"
    idx_Sri_Lanka = [8, 9, 11, 16, 21]
    fig_Sri = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Sri_Lanka),
                                  region=region,
                                  indices=idx_Sri_Lanka)

    # fname = f"{ovrlp_lbl_pth}\\{region}_lbl_ovrlp.pdf"
    # fig_Sri.savefig(fname)
```

A.4 Attention U-Net Evaluation Pipeline

```
"""
Improving Semantic Water Segmentation by Fusing Sentinel-1 Intensity and
    Interferometric Synthetic Aperture Radar
(InSAR) Coherence Data

Author: Ernesto Colon**
The Cooper Union for the Advancement of Science and Art**
Spring 2022

Attention U-Net Model Inference
"""

# Import libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from utils import metrics_utils
from utils import dataset_gen
from utils import general_utils
import time
import tensorflow as tf
from keras_unet_collection import models


# Define helper functions

# Create function to generate the label overlap between ground truth and
    predictions

def gen_lbl_overlap(y_true, y_pred):
    """
    Function to return a semantic map with a label overlap given ground
    truth and predicted labels
    :param y_true: ndarray with ground truth labels
    :param y_pred: ndarray with predicted labels
    :return: combined, an ndarray with 4 classes (1: true positive, 2:
    true negatives, 3: false positives, 4: false neg)
    """

    # allocate space to store the label overlap
    combined = np.zeros(y_pred.shape)

    # true positives are labels that are predicted as water (1)
    tp = np.logical_and(np.where(y_pred == 1, 1, 0), np.where(y_true == 1, 1, 0))

    # true negatives
    tn = np.logical_and(np.where(y_pred == 0, 1, 0), np.where(y_true == 0, 1, 0))

    # false positives are labels that were labeled as 1 but that were 0 in reality
    fp = np.logical_and(np.where(y_pred == 1, 1, 0), np.where(y_true == 0, 1, 0))

    # false negatives are labels that were labeled as 0 but were 1 in reality
    fn = np.logical_and(np.where(y_pred == 0, 1, 0), np.where(y_true == 1, 1, 0))

    # combine all classes
    combined[tp] = 1
    combined[tn] = 2
    combined[fp] = 3
    combined[fn] = 4

    return combined

# Generate color maps for the labels and label overlap

from utils import general_utils

wtr_cmap = general_utils.gen_cmap(['#f7f7f7', '#67a9cf'])
ovrlp_cmap = general_utils.gen_cmap(['#67a9cf', '#f7f7f7', '#ef8a62', '#999999'])

from mpl_toolkits.axes_grid1.anchored_artists import AnchoredSizeBar
import matplotlib.font_manager as fm

fontprops = fm.FontProperties(size=15)


def display_lbl_overlap(y_true, lbl_overlap, x_test, num_plot, region, indices=None):
    """
    Function to display the label overlap
    :param y_true: ndarray with ground truth labels
    :param lbl_overlap: ndarray with label overlap
    :param x_test: ndarray with Sentinel-1 co-event intensity (VH) raster
    :param num_plot: integer, number of scenes to display
    :param region: string with geographical region to display
    :param indices: list of integer with indices to plot from the entire data set
    :return: matplotlib figure handle
    """

    fontprops = fm.FontProperties(size=12)

    num_col = 5
    fig, ax = plt.subplots(num_plot + 1, num_col, figsize=(20, 5 *
num_plot))
    ax = ax.ravel()

    if indices == None:
        indices = range(num_plot)

    for idx, raster in enumerate(indices):

        ax[num_col * idx].imshow(x_test[region]['scenario_1_hand_lbl'][
raster, :, :, 0], cmap='gray')
        ax[num_col * idx].set_title(f'Co-event Intensity (VH)')

        # plot ground truth
        ax[num_col * idx + 1].imshow(y_true[region]['scenario_3_hand_lbl'
][:, :, raster], cmap=wtr_cmap)
        ax[num_col * idx + 1].set_title(f'Ground Truth Label')

        # plot scenario 1
        ax[num_col * idx + 2].imshow(lbl_overlap[region]['
scenario_1_hand_lbl'][:, :, raster], cmap=ovrlp_cmap)
        ax[num_col * idx + 2].set_title('Scenario 1 Label Overlap')

        # plot scenario 2
        ax[num_col * idx + 3].imshow(lbl_overlap[region]['
scenario_2_hand_lbl'][:, :, raster], cmap=ovrlp_cmap)
        ax[num_col * idx + 3].set_title('Scenario 2 Label Overlap')

        # plot scenario 3
        ax[num_col * idx + 4].imshow(lbl_overlap[region]['
scenario_3_hand_lbl'][:, :, raster], cmap=ovrlp_cmap)
        ax[num_col * idx + 4].set_title('Scenario 3 Label Overlap')

        for axis in ax[: num_col * num_plot]:
            scalebar = AnchoredSizeBar(
                axis.transData,
                100,
                '100m',
                'lower left',
                pad=0.1,
                color='black',
                frameon=False,
                size_vertical=1,
                fontproperties=fontprops)

            axis.add_artist(scalebar)
            axis.set_yticks([])
            axis.set_xticks([]);

    # Create legend
    checkerboard = np.zeros((512, 512))
    checkerboard[0:256, 0:256] = 1
```

```python
    checkerboard[256:, 0:256] = 2
    checkerboard[0:256, 256:] = 3
    checkerboard[256:, 256:] = 4

    ax[num_col * idx + 4 + 3].imshow(checkerboard, cmap=ovrlp_cmap)
    ax[num_col * idx + 4 + 3].text(50, 128, "True Positives", fontsize=8.);
    ax[num_col * idx + 4 + 3].text(50, 384, "True Negatives", fontsize=8.);
    ax[num_col * idx + 4 + 3].text(290, 128, "False Positives", fontsize=8.);
    ax[num_col * idx + 4 + 3].text(290, 384, "False Negatives", fontsize=8.);
    ax[num_col * idx + 4 + 3].set_yticks([])
    ax[num_col * idx + 4 + 3].set_xticks([]);

    ind_to_del = [1, 2, 4, 5]
    for ind in ind_to_del:
        fig.delaxes(ax[num_col * idx + 4 + ind])

    return fig


if __name__ == "__main__":

    ################################################################
    #           Load previously saved dataset splits
    ################################################################

    # Define dictionary with filepaths
    base_dir = "base_dir_path"

    train_val_test_pths = {'train_fn_df' : f"{base_dir}\\ds_train_split_10m.csv",
                           'val_fn_df' : f"{base_dir}\\ds_val_split_10m.csv",
                           'test_fn_df' : f"{base_dir}\\ds_test_split_10m.csv"}

    # Load csv files with train / val / test splits into dataframes
    train_val_fn_df, test_fn_df, train_size, val_size, test_size =\
        dataset_gen.unet_load_ds_df(train_val_test_pths['train_fn_df'],
                                    train_val_test_pths['val_fn_df'],
                                    train_val_test_pths['test_fn_df'])

    ################################################################
    #   Define category names and a color mapping for semantic segmentation
    ################################################################

    # Define category names
    tgt_cat_names = {
        0: 'Not water',
        1: 'Water'
    }

    # Define the colors per category
    wtr_clrs_hex = ['#f7f7f7', '#67a9cf']

    # Generate the labels colormap
    wtr_cmap = general_utils.gen_cmap(wtr_clrs_hex)


    ##############################################################
    #            Generate data sets for inference
    ##############################################################

    """
    We generate datasets for the following scenarios:

    - Scenario 1: Co-event intensity data only
    - Scenario 2: Pre- and co-event intensity data only
    - Scenario 3: Pre- and co-event intensity and coherence data
    """

    # Define dictionaries to hold the datasets - the keys will be the
   different scenarios
    X_train_dict = {}
    Y_train_dict = {}

    X_val_dict = {}
    Y_val_dict = {}

    X_test_dict = {}
    Y_test_dict = {}

    Y_pred_dict = {}

    # Define scenario number to scenario name mapping
    scenario_dict = {1: 'co_event_intensity_only',
                     2: 'pre_co_event_intensity',
                     3: 'pre_co_event_int_coh'}

    scenario_num_bands = {1: 2,
                          2: 4,
                          3: 6}

    # Define the number of bands per scenario
    num_bands_dict = {'co_event_intensity_only': 2,
                     'pre_co_event_intensity': 4,
                     'pre_co_event_int_coh': 6}

    IMG_SIZE = 512

    # define dictionaries to hold the datasets
    train_val_samples_dict = {}
    test_samples_dict = {}

```

```python
# Loop through each scenario and create the tensorflow data loaders
scenarios = [1, 2, 3]

for scenario in scenarios:

    # Create the samples list given the dataframes with file paths as input
    train_val_samples_dict[f"scenario_{scenario}"], test_samples_dict[f"scenario_{scenario}"] = \
        dataset_gen.create_samples_list({'scenario': scenario_dict[scenario],
                                            'test_df': test_fn_df,
                                            'train_val_df': train_val_fn_df})

    # Create data sets dictionary
    X_train_dict[f"scenario_{scenario}"], X_val_dict[f"scenario_{scenario}"], X_test_dict[f"scenario_{scenario}"] =\
        dataset_gen.unet_ds_creation({'train_val_list': train_val_samples_dict[f"scenario_{scenario}"],
                                      'test_list': test_samples_dict[f"scenario_{scenario}"]})

    # Batch the tensorflow train, val, and test data set generators
    X_train_dict[f"scenario_{scenario}"] =\
        X_train_dict[f"scenario_{scenario}"].batch(10).prefetch(tf.data.experimental.AUTOTUNE)

    X_val_dict[f"scenario_{scenario}"] =\
        X_val_dict[f"scenario_{scenario}"].batch(10).prefetch(tf.data.experimental.AUTOTUNE)

    X_test_dict[f"scenario_{scenario}"] = X_test_dict[f"scenario_{scenario}"].batch(1)



###############################################################
#                    Hand Labeled Dataset
###############################################################

# load hand label dataset
hand_lbl_ds_pth = "hand_lbl_ds_pth"
hand_lbl_ds_fname = f"{hand_lbl_ds_pth}hand_lbl_ds_10m_res.csv"

df_hand_lbl_samples = pd.read_csv(hand_lbl_ds_fname)

# loop through df and append sample paths to a list
hand_samples = list()

for idx, row in df_hand_lbl_samples.iterrows():
    hand_samples.append((row['s1'], row['pre_event_grd'], row['pre_event_coh'], row['co_event_coh'], row['hand_lbl']))


    hand_lbl_test_samples = dict()

    for scenario in scenarios:
        hand_lbl_test_samples[f"scenario_{scenario}_hand_lbl"] =\
            dataset_gen.create_samples_list_hand_lbl({'scenario': 
   scenario_dict[scenario], 'test_df': df_hand_lbl_samples})

        X_test_dict[f"scenario_{scenario}_hand_lbl"] =\
            dataset_gen.ds_creation_hand_lbl({'test_list': 
   hand_lbl_test_samples[f"scenario_{scenario}_hand_lbl"]})

        X_test_dict[f"scenario_{scenario}_hand_lbl"] = X_test_dict[f"
   scenario_{scenario}_hand_lbl"].batch(1)


    ################################################################
    #                    Attention U-Net Models
    ################################################################

    """
    For this study, we leverage the publicly available Keras UNet
   Collection linked below.

    https://github.com/yingkaisha/keras-unet-collection
    """

    #  Load previously saved model weights

    IMG_SIZE = 512

    attn_unet_models_dir = {'scenario_1': "model_scen_1_weights_pth",
                            'scenario_2': "model_scen_2_weights_pth",
                            'scenario_3': "model_scen_3_weights_pth"}

    attn_unet_models_dict = dict()

    for scenario in scenarios:
        attn_unet_models_dict[f"scenario_{scenario}"] =\
            models.att_unet_2d((IMG_SIZE, IMG_SIZE, scenario_num_bands[
   scenario]),
                               filter_num=[64, 128, 256, 512, 1024],
                               n_labels=2,
                               stack_num_down=2,
                               stack_num_up=2,
                               activation='ReLU',
                               atten_activation='ReLU',
                               attention='add',
                               output_activation='Sigmoid',
                               batch_norm=True,
                               pool=False,
                               unpool=False,
                               backbone='VGG16',
                               weights=None,
```

```python
                                freeze_backbone=True,
                                freeze_batch_norm=True,
                                name='attunet')

        # restore the weights
        print(f"Loading weights for scenario: {scenario}...\n")
        attn_unet_models_dict[f"scenario_{scenario}"].load_weights(
    attn_unet_models_dir[f"scenario_{scenario}"])


    ################################################################
    #                    Make predictions
    ################################################################

    # wrapping the predictions routine into a function
    def u_net_inference(x_test_ds, test_size, model):
        """
        Function to make inferences using tensorflow

        :param x_test_ds: tensorflow dataset pipeline for testing
        :param test_size: number of test images in our pipeline
        :param model: tensorflow model for inference
        :return: y_test and y_pred are ndarrays with shape (num_pix,)
        """

        start_time = time.time()

        # Compute predictions at the same time to avoid a random shift
        test_tgts_list = []
        pred_tgts_list = []

        for img, tgt, weights in x_test_ds.take(test_size):
            pred = metrics_utils.create_mask(model.predict(img))
            pred_tgts_list.append(np.squeeze(pred))

            test_tgts_list.append(np.squeeze(tgt.numpy()))

        y_test = np.stack(test_tgts_list, axis=-1).flatten()
        y_pred = np.stack(pred_tgts_list, axis=-1).flatten()

        print(f"Inference took: {time.time() - start_time} seconds\n")

        return y_test, y_pred


    ################################################################
    # Predicting on held-out test set and hand-labeled test set
    ################################################################
    """
    Notes

    The held-out test set is comprised of Sentinel-2 weak labels from the
    Sen1Floods11 data set.

    The hand-labeled data set is also provided by the Sen1Floods11 data
set, and provides an independent data set not
    used during training.
    """

    start_time = time.time()

    # loop through each scenario
    for scenario in scenarios:

        # Held-out dataset predictions
        print(f"Predicting on held-out test dataset for scenario: {
scenario}...\n")
        Y_test_dict[f"scenario_{scenario}"], Y_pred_dict[f"scenario_{
scenario}"] =\
            u_net_inference(X_test_dict[f"scenario_{scenario}"],
                            test_size,
                            attn_unet_models_dict[f"scenario_{scenario}"])

        # Hand-labeled dataset predictions
        print(f"Predicting on hand-labeled test dataset for scenario: {
scenario}...\n")
        Y_test_dict[f"scenario_{scenario}_hand_lbl"], Y_pred_dict[f"
scenario_{scenario}_hand_lbl"] =\
            u_net_inference(X_test_dict[f"scenario_{scenario}_hand_lbl"],
                            test_size,
                            attn_unet_models_dict[f"scenario_{scenario}"])

    print(f"\n\nTotal inference took: {time.time() - start_time} seconds")


    ###############################################################
    # Compute Metrics
    ###############################################################

    """
    For metrics, we compute:

    - Overall accuracy
    - Mean intersection over union, mIoU
    - Jaccard score
    - Water precision
    - Water recall
    - Water f1-score
    - Not-Water precision
    - Not-Water recall
    - Not-Water f1-score
    """

    ###############################################################
    #                    Held-out Test Dataset
    ###############################################################

    start_time = time.time()
```

```python

    summary_df = metrics_utils.summary_report(Y_test_dict, Y_pred_dict)

    print(f"\n\nProcess took: {time.time() - start_time} seconds")

    # save summary to csv file
    attn_unet_summ_pth = "attn_unet_summ_pth"
    fname = "attn_unet_10m_summary_stats.csv"
    summary_df.to_csv(f"{attn_unet_summ_pth}\\{fname}")


    ##############################################################
    # Computing IoU per class (i.e., water and not-water)
    ##############################################################

    miou_per_class = metrics_utils.miou_per_class(Y_test_dict, Y_pred_dict
)

    # save to csv file
    mIou_fname = "attn_unet_10m_mIoU_per_class_stats.csv"
    miou_per_class.to_csv(f"{attn_unet_summ_pth}\\{mIou_fname}")


    ##############################################################
    # Testing Models' Ability to Generalize
    ##############################################################

    # We use data over the Sri-Lanka region (both weakly labeled and hand-
labeled)
    # to test the models' ability to generalize

    ##############################################################
    #         Generate generalization data set
    ##############################################################

    generalization_ds_pth = "generalization_ds_pth"

    # create empty list to store the samples' path
    gen_test_samples = []

    # load csv file into dataframe
    gen_test_fn_df = pd.read_csv(generalization_ds_pth)

    # grab number of samples in the data set
    num_gen_samp = len(gen_test_samples)

    # create dictionaries to store the tensorflow data loaders
    gener_test_samples = {}

    gener_X_test_dict = dict()
    gener_Y_test_dict = dict()


    for scenario in scenarios:
        gener_test_samples[f"scenario_{scenario}"] =\
            dataset_gen.create_samples_list_hand_lbl({'scenario':
scenario_dict[scenario], 'test_df': gen_test_fn_df})

        gener_X_test_dict[f"scenario_{scenario}"] =\
            dataset_gen.ds_creation_hand_lbl({'test_list':
gener_test_samples[f"scenario_{scenario}"]})

        gener_X_test_dict[f"scenario_{scenario}"] = gener_X_test_dict[f"
scenario_{scenario}"].batch(1)


    ################################################################
    # Hand-Labeled Generalization Data Set
    ################################################################

    # load hand-labeled dataset
    gen_hand_lbl_ds_pth = "gen_hand_lbl_ds_pth"

    # read hand-labeled data set into dataframe
    gen_hand_lbl_df = pd.read_csv(gen_hand_lbl_ds_pth)

    # create dict to store tensorflow data loaders
    gen_hand_samples_by_region_dict = {}

    # loop through each scenario and create the tensorflow dat loaders
    for scenario in scenarios:
        gener_test_samples[f"scenario_{scenario}_hand_lbl"] =\
            dataset_gen.create_samples_list_hand_lbl({'scenario':
scenario_dict[scenario],'test_df': gen_hand_lbl_df})

        gener_X_test_dict[f"scenario_{scenario}_hand_lbl"] = dataset_gen.
ds_creation_hand_lbl({'test_list': gener_test_samples[f"scenario_{
scenario}_hand_lbl"]})
        gener_X_test_dict[f"scenario_{scenario}_hand_lbl"] =
gener_X_test_dict[f"scenario_{scenario}_hand_lbl"].batch(1)

    # grab the generalization data set's size
    gener_hand_lbl_test_size = gen_hand_lbl_df.shape[0]

    ################################################################
    # Make Predictions on the Generalization Data Set
    ################################################################

    # create dictionary to store the generalization data set's predictions
    gener_Y_pred_dict = dict()

    # Keep track of how long inference takes
    start_time = time.time()

    for scenario in scenarios:

        # Held-out dataset predictions
        print(f"Predicting on held-out test dataset for scenario: {
```

```
scenario}...\n")
        gener_Y_test_dict[f"scenario_{scenario}"], gener_Y_pred_dict[f"
scenario_{scenario}"] =\
            u_net_inference(gener_X_test_dict[f"scenario_{scenario}"],
                            gener_hand_lbl_test_size,
                            attn_unet_models_dict[f"scenario_{scenario}"])

        # Hand-labeled dataset predictions
        print(f"Predicting on hand-labeled test dataset for scenario: {
scenario}...\n")
        gener_Y_test_dict[f"scenario_{scenario}_hand_lbl"],
gener_Y_pred_dict[f"scenario_{scenario}_hand_lbl"] =\
            u_net_inference(gener_X_test_dict[f"scenario_{scenario}
_hand_lbl"],
                            gener_hand_lbl_test_size,
                            attn_unet_models_dict[f"scenario_{scenario}"])

    print(f"\n\nTotal inference took: {time.time() - start_time} seconds")


    ################################################################
    ### Compute Metrics for Generalization Data Set
    ################################################################

    stat_time = time.time()

    # Generate metrics
    gener_summary_df = metrics_utils.summary_report(gener_Y_test_dict,
gener_Y_pred_dict)

    print(f"\n\nProcess took: {time.time() - start_time} seconds")

    # save the metrics to a csv file for later recall
    gener_summ_fname = "attn_unet_10m_generalization_stats.csv"
    gener_summary_df.to_csv(f"{attn_unet_summ_pth}\\{gener_summ_fname}")


    ################################################################
    # Compute IoU per class for generalization dataset
    ################################################################

    gener_miou_per_class = metrics_utils.miou_per_class(gener_Y_test_dict,
    gener_Y_pred_dict)

    # save to csv
    gener_miou_fname = "attn_unet_10m_generalization_mIoU_stats.csv"
    gener_miou_per_class.to_csv(f"{attn_unet_summ_pth}\\{gener_miou_fname}
")


    ################################################################
    # Making Inferences Aggregated by Geographical Region
    ################################################################

```

```
 for scenario in scenarios:
     gener_test_samples[f"scenario_{scenario}"] =\
         dataset_gen.create_samples_list_hand_lbl({'scenario':
scenario_dict[scenario], 'test_df': gen_test_fn_df})

     gener_X_test_dict[f"scenario_{scenario}"] =\
         dataset_gen.ds_creation_hand_lbl({'test_list':
gener_test_samples[f"scenario_{scenario}"]})

     gener_X_test_dict[f"scenario_{scenario}"] = gener_X_test_dict[f"
scenario_{scenario}"].batch(1)


 test_samples_by_region_dict = {}

 # Define the regions excluding the generalization region
 regions = ['USA', 'Mekong', 'Colombia', 'Paraguay', 'India', 'Bolivia'
]

 # create dictionary schema to hold the tensorflow data loaders
aggregated by region
 X_test_ds_region_dict = {region: {} for region in regions}

 ds_test_size_region = {}

 # loop through each of the regions
 for region in regions:
     # temp list to store the file paths
     pths = list()

     # pluck the test sample paths by region
     test_pth_region = test_fn_df[test_fn_df.s1.str.contains(region)]

     ds_test_size_region[region] = test_pth_region.shape[0]

     for scenario in scenarios:
         region_test_samples = dataset_gen.create_samples_list_hand_lbl
({'scenario': scenario_dict[scenario],
                                'test_df': test_pth_region})

         X_test_ds_region_dict[region][f"scenario_{scenario}"] =\
             dataset_gen.ds_creation_hand_lbl({'test_list':
region_test_samples})

         X_test_ds_region_dict[region][f"scenario_{scenario}"] =\
             X_test_ds_region_dict[region][f"scenario_{scenario}"].
batch(1)


 # Hand-labeled data set aggregated by region
 hand_lbl_ds_test_size_region = {}

 for region in regions:
     # temp list to store the file paths
```

```
        pths = list()

        # pluck the test sample paths by region
        test_pth_region = df_hand_lbl_samples[df_hand_lbl_samples.s1.str.
    contains(region)]

        hand_lbl_ds_test_size_region[region] = test_pth_region.shape[0]

        for scenario in scenarios:
            region_test_samples = dataset_gen.create_samples_list_hand_lbl
    ({'scenario': scenario_dict[scenario],
                                    'test_df': test_pth_region})

            X_test_ds_region_dict[region][f"scenario_{scenario}_hand_lbl"]
     =\
                dataset_gen.ds_creation_hand_lbl({'test_list':
    region_test_samples})

            X_test_ds_region_dict[region][f"scenario_{scenario}_hand_lbl"]
     =\
                X_test_ds_region_dict[region][f"scenario_{scenario}
    _hand_lbl"].batch(1)



     ###############################################################
     #       Compute inferences aggregated by region
     ###############################################################

     # create dicts to store predictions and ground truth
     Y_pred_region_dict = {region : {} for region in regions}
     Y_test_region_dict = {region : {} for region in regions}

     start_time = time.time()

     for region in regions:
         for scenario in scenarios:

             # Held-out dataset predictions
             print(f"Predicting on held-out test dataset for region: {
    region}, scenario: {scenario}...\n")
             Y_test_region_dict[region][f"scenario_{scenario}"],
    Y_pred_region_dict[region][f"scenario_{scenario}"] =\
                 u_net_inference(X_test_ds_region_dict[region][f"scenario_{
    scenario}"],
                                 ds_test_size_region[region],
                                 attn_unet_models_dict[f"scenario_{scenario
    }"])

             # Hand-labeled dataset predictions
             if region == 'Colombia':
                 continue    # The Sen1Floods11 data set does not have hand
    -labels for Colombia
             else:
                 print(f"Predicting on hand-labeled test dataset for region:{region}, scenario: {scenario}...\n")
                 Y_test_region_dict[region][f"scenario_{scenario}_hand_lbl"],\
                 Y_pred_region_dict[region][f"scenario_{scenario}_hand_lbl"] =\
                     u_net_inference(X_test_ds_region_dict[region][f"scenario_{scenario}_hand_lbl"],
                                     hand_lbl_ds_test_size_region[region],
                                     attn_unet_models_dict[f"scenario_{scenario}"])

     print(f"\n\nTotal inference took: {time.time() - start_time} seconds")

     ###############################################################
     #   Generate prediction summaries / metrics by region
     ###############################################################

     start_time = time.time()

     summary_by_region = {}

     for region in regions:
         print(f"Region: {region}\n\n")
         summary_by_region[region] = metrics_utils.summary_report(Y_test_region_dict[region], Y_pred_region_dict[region])
         print("\n\n")

         # save to csv
         summary_by_region[region].to_csv(f"{attn_unet_summ_pth}\\{region}_summary_stats.csv")

     print(f"\n\nProcess took: {time.time() - start_time} seconds")


     ###############################################################
     # Compute IoU per class aggregated by region
     ###############################################################

     regional_miou_per_class = {}

     for region in regions:
         print(f"Region: {region}\n\n")

         regional_miou_per_class[region] =\
             metrics_utils.miou_per_class(Y_test_region_dict[region], Y_pred_region_dict[region])

         print("\n\n")

         # save to csv
         regional_miou_per_class[region].to_csv(f"{attn_unet_summ_pth}\\{region}_mIoU_stats.csv")


     ###############################################################
     #       Generate labels and label overlap by region
     ###############################################################

     # Combine the generalization data set predictions and ground truth with the rest of the data set
     Y_pred_region_dict['Sri-Lanka'] = gener_Y_pred_dict

     Y_test_region_dict['Sri-Lanka'] = gener_Y_test_dict

     X_test_ds_region_dict['Sri-Lanka'] = gener_X_test_dict

     # regions with hand-labels
     all_regions = ['USA', 'Mekong', 'India', 'Bolivia', 'Paraguay', 'Sri-Lanka']
     Y_pred_hand_lbl_by_region = {}
     Y_true_hand_lbl_by_region = {}

     # scenarios to pluck
     scen_to_pluck = ['scenario_1_hand_lbl', 'scenario_2_hand_lbl', 'scenario_3_hand_lbl']

     # create schemas to store predictions and ground truth
     Y_pred_hand_lbl_by_region = {region : {scen : [] for scen in scen_to_pluck} for region in all_regions}
     Y_true_hand_lbl_by_region = {region : {scen : [] for scen in scen_to_pluck} for region in all_regions}

     X_test_hand_lbl_by_region = {}
     X_test_hand_lbl_by_region = {region : {scen : [] for scen in scen_to_pluck} for region in all_regions}


     ###############################################################
     # First, loop through the test data set and grab all the image rasters
     ###############################################################

     for region in all_regions:
         for scen in scen_to_pluck:
             ds_len = X_test_ds_region_dict[region][scen].cardinality().numpy()
             print(f"region: {region}, scen: {scen}, len: {ds_len}")
             img_lst = list()
             for img, tgt, weight in X_test_ds_region_dict[region][scen].take(ds_len):
                 img_lst.append(np.squeeze(img.numpy()))

             X_test_hand_lbl_by_region[region][scen] = np.stack(img_lst, axis=0)


     ###############################################################
     #   Copy the predictions and the ground truth labels
```

```python
    ################################################################

    for region in all_regions:
        for scen in scen_to_pluck:
            Y_pred_hand_lbl_by_region[region][scen] = Y_pred_region_dict[region][scen].copy()

            Y_true_hand_lbl_by_region[region][scen] = Y_test_region_dict[region][scen].copy()


    ################################################################
    #            Compute label overlap by region
    ################################################################

    lbl_ovrlap_by_region = {region : {scen : [] for scen in scen_to_pluck} for region in all_regions}

    for region in all_regions:
        for scen in scen_to_pluck:

            lbl_ovrlap_by_region[region][scen] = np.reshape(
                gen_lbl_overlap(
                    Y_true_hand_lbl_by_region[region][scen],
                    Y_pred_hand_lbl_by_region[region][scen]),
                (img_size, img_size, -1))



    ################################################################
    #        Plot label overlap by region
    ################################################################

    # First, reshape the labels into proper rasters for displaying
    img_size = 512
    num_feat_dict = {'scenario_1_hand_lbl': 2,
                     'scenario_2_hand_lbl': 4,
                     'scenario_3_hand_lbl': 6}

    for region in all_regions:
        for scen in scen_to_pluck:
            Y_pred_hand_lbl_by_region[region][scen] = np.reshape(Y_pred_hand_lbl_by_region[region][scen],
                                                                                    (img_size, img_size, -1))

            Y_true_hand_lbl_by_region[region][scen] = np.reshape(Y_true_hand_lbl_by_region[region][scen],
                                                                                    (img_size, img_size, -1))


    ################################################################
    # Label Overlap for Region: USA
    ################################################################

    ovrlp_lbl_pth = "ovrlp_lbl_pth"
    region = "USA"
    fname = f"{ovrlp_lbl_pth}\\attn_unet_{region}_lbl_ovrlp.pdf"

    idx_USA = [1, 3, 5, 8, 22]
    fig_USA = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_USA),
                                  region='USA',
                                  indices=idx_USA)

    # save
    #fig_USA.savefig(fname)


    ################################################################
    # Label Overlap for Region: Mekong
    ################################################################

    region = "Mekong"
    idx_Mekong = [1, 2, 5, 7, 8]
    fig_Mekong = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                     lbl_ovrlap_by_region,
                                     X_test_hand_lbl_by_region,
                                     num_plot=len(idx_Mekong),
                                     region=region,
                                     indices=idx_Mekong)

    #fname = f"{ovrlp_lbl_pth}\\attn_unet_{region}_lbl_ovrlp.pdf"
    #fig_Mekong.savefig(fname)


    ################################################################
    # Label Overlap for Region: Bolivia
    ################################################################

    idx_Bolivia = [1, 2, 3, 4, 5]
    region = "Bolivia"
    fig_Bol = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Bolivia),
                                  region=region,
                                  indices=idx_Bolivia)

    fname = f"{ovrlp_lbl_pth}\\attn_unet_{region}_lbl_ovrlp.pdf"
    #fig_Bol.savefig(fname)


    ################################################################
    # Label Overlap for Region: Paraguay
    ################################################################

    region = 'Paraguay'
    idx_Paraguay = [0, 1, 2, 6, 7]
    fig_Par = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Paraguay),
                                  region=region,
                                  indices=idx_Paraguay)


    fname = f"{ovrlp_lbl_pth}\\attn_unet_{region}_lbl_ovrlp.pdf"
    #fig_Par.savefig(fname)


    ################################################################
    # Label Overlap for Region: India
    ################################################################

    region = "India"
    idx_India = [0, 2, 4, 6, 23]
    fig_Ind =display_lbl_overlap(Y_true_hand_lbl_by_region,
                                 lbl_ovrlap_by_region,
                                 X_test_hand_lbl_by_region,
                                 num_plot=len(idx_India),
                                 region=region,
                                 indices=idx_India)

    fname = f"{ovrlp_lbl_pth}\\attn_unet_{region}_lbl_ovrlp.pdf"
    #fig_Ind.savefig(fname)

    ################################################################
    # Label Overlap for Region: Sri-Lanka
    ################################################################

    region = "Sri-Lanka"
    idx_Sri_Lanka = [8, 9, 11, 16, 21]
    fig_Sri = display_lbl_overlap(Y_true_hand_lbl_by_region,
                                  lbl_ovrlap_by_region,
                                  X_test_hand_lbl_by_region,
                                  num_plot=len(idx_Sri_Lanka),
                                  region=region,
                                  indices=idx_Sri_Lanka)

    fname = f"{ovrlp_lbl_pth}\\attn_unet_{region}_lbl_ovrlp.pdf"
    #fig_Sri.savefig(fname)
```

B References

[1] D. Bonafilia, B. Tellman, T. Anderson, and E. Issenberg, "Sen1Floods11: a georeferenced dataset to train and test deep learning flood algorithms for Sentinel-1," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 835–845, 2020.

[2] B. Harrison, D. Jupp, M. Lewis, B. Forster, N. Mueller, C. Smith, S. Phinn, D. Hudson, I. Grant, and I. Coppa, *Earth Observation: Data, Processing and Applications. Volume 1A: Data—Basics and Acquisition.* Melbourne: CRCSI, 2021.

[3] ESA, "Satellites support latest IPCC Climate Report," february 2022. [Online]. Available: https://www.esa.int/Applications/Observing_the_Earth/Space_for_our_climate/Satellites_support_latest_IPCC_climate_report

[4] H.-O. Pörtner, D. Roberts, E. Poloczanska, K. Mintenbeck, M. Tignor, A. Alegría, M. Craig, S. Langsdorf, S. Löschke, V. Möller, and A. Okem, "IPCC, 2022: Summary for Policymakers," 2022. [Online]. Available: https://report.ipcc.ch/ar6wg2/pdf/IPCC_AR6_WGII_SummaryForPolicymakers.pdf

[5] "1.47 billion people face flood risk worldwide: For over a third, it could be devastating," Accessed: 2022-03-01. [Online]. Available: https://blogs.worldbank.org/climatechange/147-billion-people-face-flood-risk-worldwide-over-third-it-could-be-devastating

[6] J. Rentschler and M. Salhab, "People in harm's way," Oct 2020, Accessed: 2022-03-01. [Online]. Available: https://openknowledge.worldbank.org/handle/10986/34655?show=full

[7] F. Meyer, "Synthetic Aperture Radar: Hazards [MOOC]," 2021, Accessed: Fall, 2021. [Online]. Available: https://www.edx.org/course/sar-hazards

[8] E. Podest, S. McCartney, E. Fielding, A. L. Handwerger, and N. A. G. Brook, “SAR for Disasters and Hydrological Applications,” 2019. [Online]. Available: https://appliedsciences.nasa.gov/join-mission/training/english/arset-sar-disasters-and-hydrological-applications

[9] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Trans. Sys. Man. Cyber. 9 (1)*, pp. 62–66, 1979.

[10] R. Gonzalez and R. E. Woords, *Digital Image Processing, 4th Ed.* New York, NY: Pearson, 2018.

[11] I. Woodhouse, *Introduction to Microwave Remote Sensing, 1st Ed.* CRC Press, 2006.

[12] ESA, “Sentinel-1 - Missions.” [Online]. Available: https://sentinel.esa.int/web/sentinel/missions/sentinel-1

[13] Y. Li, S. Martinis, and M. Wieland, “Urban flood mapping with an active self-learning convolutional neural network based on TerraSAR-X intensity and interferometric coherence,” *ISPRS Journal of Photogrammetry and Remote Sensing*, 2019.

[14] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, “Google Earth Engine: Planetary-scale geospatial analysis for everyone,” 2017. [Online]. Available: https://doi.org/10.1016/j.rse.2017.06.031

[15] HYDRAFloods, “HYDRAFloods Documentation.” [Online]. Available: https://servir-mekong.github.io/hydra-floods/algorithms/

[16] A. D. Nobre, L. A. Cuartas, M. G. Hodnett, C. D. Rennó, G. Rodrigues, A. Silveira, M. J. Waterloo, and S. R. Saleska, “Height Above the Nearest Drainage – a hydrologically relevant new terrain model,” *Journal of Hydrology*, vol. 404, issues 1-2, pp. 13–29, 2011.

[17] “HydroSAR training: Extracting flood information from SAR,” Accessed: Fall 2021. [Online]. Available: https://servir.icimod.org/events/hydrosar-training-extracting-flood-information-from-sar/

[18] F. Meyer, “HYDROSAR Project. GitHub Repository.”

[19] A. Twele, W. Cao, S. Plank, and S. Martinis, “Sentinel-1-based flood mapping: a fully automated processing chain,” *International Journal of Remote Sensing*, vol. 37, pp. 2990 – 3004, 2016.

[20] ASF, “Alaska Satellite Facility, UAF,” Accessed: Fall 2021, Spring 2022. [Online]. Available: https://asf.alaska.edu

[21] V. Scotti, M. Giannini, and F. Cioffi, “Enhanced flood mapping using synthetic aperture radar (SAR) images, hydraulic modelling, and social media: A case study of Hurricane Harvey (Houston, TX),” *Journal of Flood Risk Management*, vol. 13, no. 4, 2020.

[22] V. Katiyar, N. Tamkuan, and M. Nagai, “Near-Real-Time Flood Mapping Using Off-the-Shelf Models with SAR Imagery and Deep Learning,” *Remote Sensing*, vol. 13, no. 12, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/12/2334

[23] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2481–2495, 2017.

[24] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *MICCAI*, 2015.

[25] B. Peng, Q. Huang, J. Vongkusolkit, S. Gao, D. B. Wright, Z. N. Fang, and Y. Qiang, “Urban Flood Mapping With Bitemporal Multispectral Imagery Via a Self-Supervised Learning Framework,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 2001–2016, 2021.

[26] T. Mayer, A. Poortinga, B. Bhandari, A. P. Nicolau, K. Markert, N. S. Thwal, A. Markert, A. Haag, J. Kilbride, F. Chishtie, A. Wadhwa, N. Clinton, and D. Saah, "Deep learning approach for Sentinel-1 surface water mapping leveraging Google Earth Engine," *ISPRS Open Journal of Photogrammetry and Remote Sensing*, vol. 2, 2021.

[27] L. Pulvirenti, M. Chini, N. Pierdicca, and G. Boni, "Use of SAR Data for Detecting Floodwater in Urban and Agricultural Areas: The Role of the Interferometric Coherence," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, pp. 1532–1544, 2016.

[28] M. Chini, R. Pelich, L. Pulvirenti, N. Pierdicca, R. Hostache, and P. Matgen, "Sentinel-1 InSAR Coherence to Detect Floodwater in Urban Areas: Houston and Hurricane Harvey as A Test Case," *Remote. Sens.*, vol. 11, p. 107, 2019.

[29] C. Chaabani, M. Chini, R. Abdelfattah, R. Hostache, and K. Chokmani, "Flood Mapping in a Complex Environment Using Bistatic TanDEM-X/TerraSAR-X InSAR Coherence," *Remote. Sens.*, vol. 10, p. 1873, 2018.

[30] "Sen1Floods11 GitHub Repository." [Online]. Available: https://github.com/cloudtostreet/Sen1Floods11

[31] E. Colon, "Flood Mapping with SAR Intensity and InSAR Coherence - GitHub Repository." [Online]. Available: https://github.com/ecolon08/Flood-Mapping-with-SAR-Intensity-and-InSAR-Coherence

[32] A. Flores, K. Herndon, R. Thapa, and E. Cherrington, *The SAR Handbook: Comprehensive Methodologies for Forest Monitoring and Biomass Estimation*, 2019.

[33] E. Podest, N. Pinto, and E. Fielding, "Introduction to Synthetic Aperture Radar," 2017. [Online]. Available: https://appliedsciences.nasa.gov/join-mission/training/english/arset-introduction-synthetic-aperture-radar

[34] E. Podest, H. McNairn, X. Jiao, and E. Fielding, “Radar Remote Sensing for Land, Water, Disaster Applications,” 2018. [Online]. Available: https://appliedsciences.nasa.gov/join-mission/training/english/arset-radar-remote-sensing-land-water-disaster-applications

[35] F. Meyer, “GEOS 657 Microwave Remote Sensing. Module II: Imaging Radar Systems [Online Lectures],” 2021, Accessed: Fall, 2021 and Spring, 2022. [Online]. Available: https://radar.community.uaf.edu/module-2-imaging-radar-systems/

[36] NASA, “Introduction to the electromagnetic spectrum,” Accessed: 2022-02-24. [Online]. Available: https://science.nasa.gov/ems/01_intro

[37] NASA, “NASA-ISRO SAR Mission (NISAR),” Accessed: Spring 2022. [Online]. Available: https://nisar.jpl.nasa.gov/

[38] ESA, “Sentinel-1 - Data Products,” Accessed: February 2022. [Online]. Available: https://sentinel.esa.int/web/sentinel/missions/sentinel-1/data-products

[39] ASF, “Sentinel-1 - Data and Imagery,” Accessed: February 2022. [Online]. Available: https://asf.alaska.edu/data-sets/sar-data-sets/sentinel-1/sentinel-1-data-and-imagery/

[40] “Cloud to Street.” [Online]. Available: https://www.cloudtostreet.ai/

[41] T. Glazer, “How to map floodwater from radar imagery using semantic segmentation - benchmark,” Aug 2021, Accessed: February 2022. [Online]. Available: https://www.drivendata.co/blog/detect-floodwater-benchmark/

[42] ESA, “SNAP - ESA Sentinel Application Platform.” [Online]. Available: http://step.esa.int

[43] Google, “Sentinel-1 algorithms, Google Earth Engine,” Accessed: February 2022. [Online]. Available: https://developers.google.com/earth-engine/guides/sentinel1

[44] "HyP3 - Alaska Satellite Facility's Hybrid Pluggable Processing Pipeline." [Online]. Available: https://hyp3-docs.asf.alaska.edu/

[45] GAMMA, "GAMMA Remote Sensing." [Online]. Available: https://www.gamma-rs.ch/

[46] GDAL/OGR contributors, *GDAL/OGR Geospatial Data Abstraction software Library*, Open Source Geospatial Foundation, 2021. [Online]. Available: https://gdal.org

[47] G. M. James, D. D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning, 2nd Ed.[PDF].* Srpinger, 2021.

[48] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction. 2nd Ed.* New York: Springer, 2009.

[49] J. Brownlee, "Extreme gradient boosting (XGBoost) ensemble in Python," Apr 2021, Accessed: 2022-02-16. [Online]. Available: https://machinelearningmastery.com/extreme-gradient-boosting-ensemble-in-python/

[50] O. Oktay, J. Schlemper, L. L. Folgoc, M. J. Lee, M. P. Heinrich, K. Misawa, K. Mori, S. G. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert, "Attention U-Net: Learning Where to Look for the Pancreas," *ArXiv*, vol. abs/1804.03999, 2018.

[51] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2015.

[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow:

Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[53] Y. Sha, "Keras-unet-collection. GitHub repository," 2021. [Online]. Available: https://github.com/yingkaisha/keras-unet-collection

[54] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2015.

[55] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[56] Keras, "Keras Documentation: Image Segmentation Metrics," Accessed: Fall 2021, Spring 2022. [Online]. Available: https://keras.io/api/metrics/segmentation_metrics/#meaniou-class

[57] Scikit, "Metrics and scoring: Quantifying the quality of predictions," Accessed: February 2022. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics