
# A Study on Flood Detection Using Satellite Imaging and Deep Semantic Segmentation

B.Sc. Computer Science

School of Computing and Communication Lancaster University

Name: Luke Van Der Veen

Date of Submission: 21/03/2025

# Declaration of originality

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Name: Luke Van Der Veen
Date: 21/03/2025

# Abstract

This project explores the use of satellite imagery and deep learning techniques to develop an interpretable flood detection model. Leveraging Sentinel-2 hyperspectural satellite images, the study implements and extends previous works to classify flooded areas. Interpretable Deep Semantic Segmentation (IDSS) architecture combines convolutional neural networks (CNNs) with prototype based learning and rule based decision making, and is enhanced by replacing its U-net CNN backbone with a SegNet backbone. SegNet is a CNN architecture optimised for memory efficiency and real time inference and therefore holds potential capabilities for improving flood detection. Using ML4Floods pipeline and WorldFloods dataset, the study evaluates model performance through key segmentation metrics such as Intersection over union (IoU), recall and loss. Results demonstrate that while both CNN backbones achieve reasonable classification performance, SegNet shows potential for faster training with comparable accuracy to U-net. Challenges related to class imbalance, computational limitations and prototype constraints are addressed. This project highlights the feasibility and adaptability of explainable AI in flood detection, laying the groundwork for future research in CNN architecture experimentation and interpretable earth observation models.

# Contents

# 1.Introduction

This project is a study on the use of semantic segmentation and convolution neural networks (CNN) to segment Sentinel 2 satellite images, creating a model to identify areas of flooding.

Flooding accounts for more than $40 billion in damages yearly [1], ensuring it is at the forefront of natural disaster mitigation and prevention. The development of new technologies such as better satellite imagery have helped improve identification of flooded areas. This is crucial as it allows for faster responses to flooding events, leading to more damage being mitigated. However satellites take a lot of images. A sentinel series satellite in the European Space Agency's (ESA) Copernicus project captures a terabyte of images a day [2]. This makes sifting through them and identifying floods very difficult to do manually, so we use AI models. Semantic segmentation separates images into sections identifying key objects in the image. This technique has been applied to a multitude of earth observations: flooding, wild fires, clouds, vegetation and far more.

Satellite images are multiple hyper spectral so one of the key issues is correctly classifying the segments in the image to produce an accurate inference of objects in the satellite image.

## 1.1 Why is Flood Detection necessary?

Flooding, like all natural disasters, are extremely difficult to manage, mitigate and prevent. They cause irrefutable damage to people, the environment and the economy, displacing millions around the world and destroying massive areas of land, rural and urban. Throughout the years flood events have increased drastically, between 1990 and 2020 the number of flood events have risen by 240% with 59 events in 1990 and 201 in 2020 [3]. This could partially be caused by the increase in sea level, 20cm between 1901 and 2018, the largest increase in the past 3000 years. Climate change happens naturally but since the industrial revolution it has been drastically exacerbated due to human activity and is likely the driving force behind the increase in flooding in the past 100 years. Areas in South East Asia are particularly susceptible such as Bangladesh, Indonesia, Pakistan and India. This is driven by monsoon systems and their proximity to the coast. In 2007 Cyclone Sidr displaced millions and caused $1.7 billion in damages, similar patterns were observed in 2005 India Mumbai floods, which paralysed infrastructure [4]. The role climate change plays in these events cannot be understated, changes in weather patterns, melting glaciers and extreme weather conditions are making weather events such as monsoons more unpredictable, dangerous and intense. Increased rainfall and rising sea levels all concatenating into an increase in flood events. With these events becoming far more prevalent in the modern day, the importance of flood detection becomes critical for mitigating social, environmental and economic damage. Emerging technologies such as new early warning systems, observation tools and AI detection are becoming instrumental to handling the mitigation of these events.

The development of new observation techniques and warning systems are at the forefront of mitigation. Satellite imagery is a key earth observation tool crucial for the identification of landforms and weather events within earth as well as events and their causes, such as identifying an approaching strom which could cascade into a flood event. With the European Space Agency's Copernicus project, one of the leading forces in earth observation with their Sentinel line of satellites. However due to the sheer amount of data these satellites capture it becomes especially difficult to sift through this data, especially for crucial images which might help flood identification leading to lives being saved. The use of deep learning techniques to create models have drastically improved weather and climate identification. The Prithvi WxC general purpose AI model created by NASA and IBM [5] which is a groundbreaking foundation model to be repurposed into many different tasks, such as flood detection and hurricane tracking. Used in conjunction with sentinel 2 satellites, models like

this could be built to think like a weather forecaster and identify events within the terabytes of information increasing the identification of time sensitive events such as flooding.

Due to the multiple hyper spectral nature of satellite images they can provide many levels of information, there are 13 hyper spectral bands to a sentinel 2 image which you can see in Table 1. These bands can give us a large amount of data within the image, not just your usual 3 dimensional images (Blue Green Red). The increase in bands allows information on other wavelengths of light such as Near Infrared and Short Wave Infrared. This allows for finer detailing within the image allowing for identification of subtle material presences such as water presence, vegetation health. This makes the data gathered by satellites such as sentinel 2 invaluable for earth observation. However this also increases the processing requirements in deep learning models such as the pair of semantic segmentation and Deep Neural Networks such as U-net or SegNet.

| Name | Description | Resolution (m) |
|---|---|---|
| B01 | Coastal aerosol | 60 |
| B02 | Blue | 10 |
| B03 | Green | 10 |
| B04 | Red | 10 |
| B05 | Vegetation red edge | 20 |
| B06 | Vegetation red edge | 20 |
| B07 | Vegetation red edge | 20 |
| B08 | NIR | 10 |
| B8A | Narrow NIR | 20 |
| B09 | Water vapour | 60 |
| B10 | SWIR | 60 |
| B11 | SWIR | 20 |
| B12 | SWIR | 20 |

Table 1.1 The 13 Hyper Spectral bands of a Sentinel 2 Image

## 1.2 Aims

The aim of this study is to present a method of segmenting an image, correctly identifying areas of water to a relatively high degree. Initially the study will use one neural network and then transition to a different neural network to see if it is compatible with different deep neural network backbones as well as testing with a large dataset to assess the generality of both structures. Publicly available satellite datasets shall be employed for these neural network methods, which shall be prepared in order to run through these pipelines.

The study shall look at strengths and weaknesses of previous works to try and build an effective pipeline for segmenting satellite images, taking aspects from certain other studies in order to accomplish this.

Satellite images are multi hyperspectral in nature and therefore require and in order to properly handle this the bands must be handled to best identify areas of water, especially as the bands allow for more information to help identify segments of the image. Bands such as NIR and SWIR can be especially useful when it comes to identifying bodies of water, providing more information than just visible light allowing for a more precise identification of water bodies. NIR and SWIR are differently absorbed by water and land which can further help distinguish them in multi hyper spectral images such as the sentinel 2 images[6].
The outcome of this study will be a segmentation model which can accurately identify bodies of water from satellite images with an interchangeable backbone.

# 2. Related Works

This section will cover the background research for this project, going into the importance of flood mitigation, deep learning components and closely related studies for this project's development. Exploring foundational work that informs and supports this project. Beginning with an overview of why flood mitigation is important, delving into the global impacts of flooding and the damage it causes to livelihoods. It shall then evolve into a far more technical standpoint, delving into Convolution Neural Networks and their importance and relevance towards semantic segmentation. Finally this section shall delve into closely related pieces of intellectual property that will assist the development of this project, covering databases, architectures and pipelines for development.

## 2.1 The Importance of flood mitigation

Flooding damage is undeniable and extensive. Social, economic and environmental impacts which are vast in every category. They can be caused by many different natural disasters: hurricanes, tsunamis, earthquakes and plenty of others, making them a common secondary impact of an already disastrous event, cascading the effects making them even more disastrous. Social impacts can consist of mass displacement, in 2023 9.4 million people were displaced due to flooding [7]. Destroyed infrastructures prolongs displacement,  forcing displaced people to settle into temporary shelters until the reconstruction is completed, putting strain on the nations and organisations hosting these refugees.

Environmentally flooding can do severe damage, destruction of habitats, erosion of river banks, pollution as floods can burst sewage lines and carry contaminants into fresh water supplies leading to the spread of disease. In 2022 Durban a coastal city in South Africa suffered severe flooding, leading 80% of its water supply to become out of order due to an immense amount of sewage contamination, creating bursts of E coli, killing off a lot of aquatic wildlife[8].

The economic damage flooding causes can’t be denied either, in the US between the years of 1980 and 2024 on average flooding causes $4.3 billion damage per year. For economic giants such as the US this can be handled with relative ease due to its economic prospects, but this could be a lot more difficult for countries with less economic resources. Countries such as Bangladesh which have a much smaller GDP and far fewer financial resources, experience far more flooding events making it significantly harder to mitigate flooding and recover. Monsoons constantly flow through the area, combined with the estuaries that flow through the country make it a flooding hot spot. On average 20-25% of the country is submerged underwater due to flooding[9]. These issues demand mitigation, physical and digital.

With all these global impacts it is paramount that steps be taken to mitigate flood damages. A lot of physical prevention takes place such as sea walls to prevent coastal flooding or early warning systems for tsunamis. This project aims to explore digital methods, such as better weather forecasting and earth observation. Earth observation consists of webs of satellites surrounding Earth’s atmosphere and has become a key tool to modern flood detection and hazard response satellites such as the Sentinel series and Landsats are able to monitor precipitation levels, river flows and land changes providing critical real time data for flood forecasting. These observations can help alert early warning systems, assess post disaster damages and identify areas at the most risk.

With the integration of machine learning, the analysis of the vast datasets of images created by satellites has become a lot faster [10], improving forecasting efficiency and emergency response times. International initiatives such as NASA’s SERVIR [11] program has enabled developing countries to gain access to NASA’s satellite data in order to help with their preparations and mitigations. The combination of larger more established organisations taking on the front of data

capture and processing, allows developing countries to focus their resources on infrastructure resilience and local response efforts rather than investing in expensive satellite systems. By leveraging AI-driven flood mapping, governments and aid organisations can detect high risk areas faster, allocate resources and emergency supplies more effectively and implement preemptive evacuations. Additionally, ongoing advancements in deep learning techniques such as U-net and Prithvi WxC, are further enhancing flood prediction capabilities, bridging the gap between data rich and resource limited nations.

## 2.2 Convolution Neural Networks

Convolution Neural Networks (CNN) are a subset of machine learning and reside at the heart of deep learning algorithms. CNNs are most effective at tasks such as extracting information from image, speech and audio signal inputs. They are typically composed of convolutional layers, pooling layers and a fully connected layer[12]. The convolution layer is the core of CNNs, and contains the majority of computation, aiming to extract features from impetus. Pooling layers, also known as downsampling, reduces the number of parameters in the input, this does lose a lot of information however it reduces complexity and improves efficiency for the model. Finally the fully-connected layer, this layer performs classification for the input based on the features extracted.

### 2.2.1 U-net

U-net is a CNN used primarily for semantic segmentation of images. They're split into 2 parts, the encoder and decoder; the encoder compresses an image to the lower spatial dimensions allowing for features to be extracted, while the decoder upsamples this information back to the original image size allowing each pixel to gain a classification label. Skip connections are built between the corresponding levels of encoder and decoder in order to make faster and better predictions as the model using the backbone learns. Figure 2.1 displays the architecture of the U-net CNN, with encoder on the left and the decoder on the right, as well as skip connections between each layer.

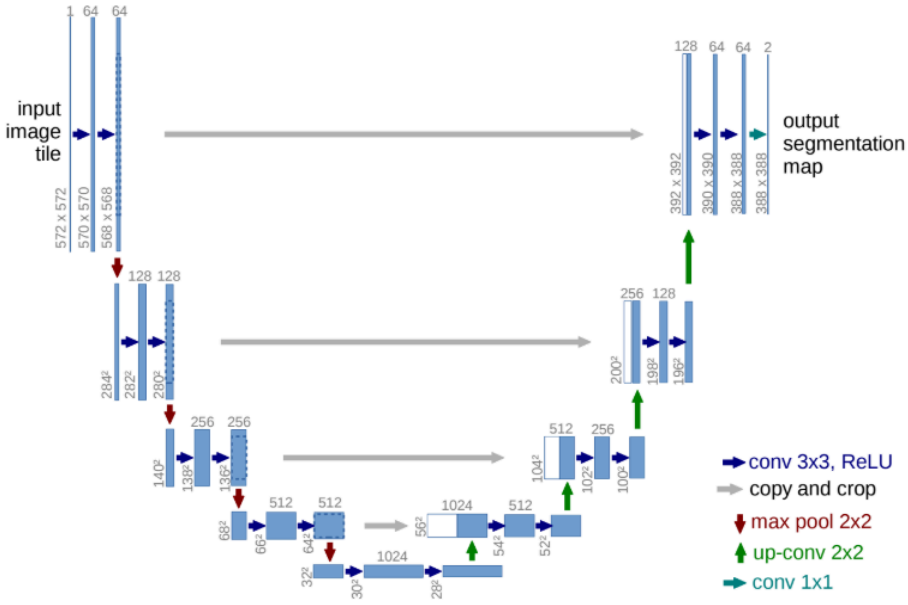



Figure 2.1: U-net architecture diagram [13]

U-net is very effective in semantic segmentation tasks due to its ability to extract features from images with its encoder and applying labels to each pixel in the image with its decoder. The skip connections

allow for faster classifications, as it preserves spatial information lost through the downsampling stages in the encoder, ensuring precise and fast predictions.

### 2.2.2 SegNet

Similarly to U-net, SegNet also has encoder and decoder stages shown in figure 2.2, both with 13 convolution layers, however it has 1 last part, a pixel wise classification layer following the final decoder layer [14]. Each convolution layer performs a convolution function in order to extract features from the image along with batch normalisation and Rectified Linear Unit (ReLU) activation introduced to each element to create non-linearity to improve feature learning. Unlike U-net, which relies on skip connections, SegNet stores the pooling indices from the max-pooling layers in the encoder stage, then reuses them in the decoder stage for upsampling. This eliminates the need of learning complex upsampling filters, making SegNet more memory efficient.

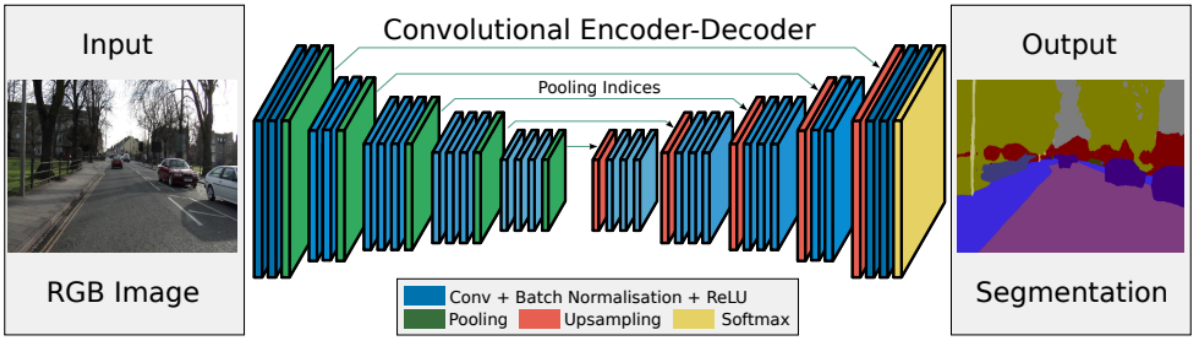



Figure 2.2: SegNet architecture diagram

This is good for semantic segmentation as it allows a less heavy memory toll in the segmentation process, allowing for efficient inference speeds. Due to this speed, when scaled up with powerful hardware it becomes a powerful back bone for real time flood mapping.

## 2.3 Project origins

With flooding rising in frequency, the use of deep learning techniques are essential. Deep learning (DL) techniques, particularly Deep Neural Networks (DNN) are especially effective at identifying objects within a satellite image. Their multilayer architecture is proficient at extracting patterns from images, this is essential for identifying flood affected areas for this project. This technique of extracting patterns and objects from images is known as semantic segmentation, this method classifies pixels and assigns them to classes with labels which represents a known type of object, such as a blue pixel being classed as a water pixel. Flood sensing using satellite imagery has been done for a very long time, dating back to the 1970s when the first ERTS-1[15] (Earth Resources Technology Satellite), later renamed to the Landsat-1 which used similar methods of analysing multi hyper-spectural data to identify water masses within the earth. However with the integration of DL and computer vision techniques such as semantic segmentation in addition to a far better understanding as to how some of the spectrums interact with water, water identification has become far more efficient. Moreover, the use of Synthetic Aperture Radar (SAR) has also contributed to this enhancement. SAR's ability to penetrate through weather conditions and operate independent of daylight, makes it quite invaluable when it comes to identifying flooding[16]. Combining all of these features has led to significant improvements in flood identification.

As useful as AI is, its inferences can often be difficult to interpret as a lot of the decision making is done behind the scenes to ensure ease of use for the user. But in cases such as this project, it could be useful to explain decisions such as the selection of water class for a pixel because it reached the

threshold of NIR. Water can absorb more NIR so it will have a greater positive value [17]. This is where explainable AI comes in (XAI), this is the concept of making AI decision making more interpretable for users. The following paper assesses metrics for Class Activation Maps [18] , a method for determining specific regions of an image using a Convolution Neural Network (CNN), as previous metrics were far too difficult to understand. CAMs take weighted averages of a collection of activation maps to produce an explanation map highlighting the segment of the image that the CAM was trying to find. Previously used metrics such as average drop and increase (measured a % drop or increase in confidence when the model sees the explanation map), as well as deletion and insertion (% change in probability of finding the target segment when important pixels are either deleted or inserted into the image). These metrics were cumbersome to compare due to there being so many of them, they also focused their drops in confidence based on the explanation map, which could be faked to produce more ideal results and was proven in experimentation. The paper proposed new metrics which were combinable into a single % measurement which was far easier to compare CAMs with. These metrics consisted of maximum complexity, minimum coherency and minimum confidence drop which combined into average DCC (ADCC). An experiment was performed using 50,000 images, 6 CNNs and multiple CAMs. Results found that certain CAMs performed better than others (Score-CAM produced the best ADCC). ADCC was able to catch FAKE-CAM and correctly penalise it for its ideal explanation map. The CNN also created variation for example GradCAM was best at ADCC when paired with ResNet. Works such as these encapsulate the need for explainability in complex AI works, this project shall look into aspects of explainability in order to enhance understanding of key decisions when identifying pixels.

### 2.3.1 ML4Floods

In order to undergo this project, it will need an extensively tested architecture that covers a lot of the hidden processing of satellite imagery. ML4Floods is an end-to-end site package for python that has been rigorously tested and covers a lot of the architecture this project requires, from database setup and configurations to inference[19,20,21]. The package creates pipelines for data configurations and preprocessing, image band assignment, model configurations, training, validation and inferencing, providing the perfect structure for our project. Whilst this package provides a strong foundation it doesn't provide everything. A dataset still needs to be sourced, an additional model needs to be integrated and methods need to be extracted from previous works in order to improve what this package already provides. Augmentation to the training structure is the crucial step here, which will require this project to train the model provided by the package, that being U-net. Experimenting with values and configurations will take time, especially as this project involves processing hyper-spectural imagery which can take a significant period of time to perform, based on what processing is occurring during training, validation and test steps, the size of the database and any preprocessing performed on the database. ML4Floods is part of a side project Earth System lab FDL 2019 funded by United Kingdom Space Agency and led byTrillium Technologies as a part of its disaster prevention[24,25]. Aiming to investigate the use of AI to enhance flood forecasting using orbital imagery; exploring the use of low cost satellite imagery especially due to their crucial role in creating flood extent maps, playing a direct role in flood relief.

### 2.3.2 WorldFloods

The WorldFloods database offers 300Gb of sentinel 2 images, 509 pairs of images and masks [19] split into training, validation and testing sets. With this level of diversity in sentinel 2 images and flood events the project will have an excellent foundation for developing an extensively trained model with plenty of representation for water.The size of the dataset does propose a potential issue; building a semantic segmentation model with such a large dataset such as this would likely take far too long. Constantly working on training, testing hyperparameters for the model and epoch lengths, a dataset of

this size would add exponential time to fine tuning and building this model. WorldFloods offers sample datasets which would be more time efficient. These sample datasets are around 12Gb worth of image and ground truth mask pairs, offering a well sized dataset for fine tuning and building models whilst still being split into the same sets, allowing for easy transitioning between the 2 datasets. These images vary in sizing so in order to fit into the U-net pipeline the satellite images will need to be resized to 256 height and 256 width.

### 2.3.3 Interpretable Deep Semantic Segmentation (IDSS)

The works of Zhang et al. (2022) [26] provide a new approach to semantic segmentation. An interpretable approach to semantic segmentation of earth observed based flood detection. Traditional methods, such as purely using a U-net offered high accuracy but didn't offer much interpretability, which is a crucial step for identifying areas of flooding. IDSS enhances the decision making in flood mapping .This proposed method bridged the gap between model accuracy and interpretability by leveraging a prototype based learning method with XAI principles.

Instead of relying on abstract, high dimensional representations like most pure CNN semantic segmentation models, IDSS clusters raw features into human interpretable prototypes. The prototypes are derived from mini-batch K-means clustering method, applied either to the raw feature layer or both the raw and latent feature layer500 prototypes were given to each class land, water and cloud, forming a total of 1500 representative prototypes. These clusters create reference points in the segmentation process allowing for more characteristic feature sets and transparent decision making in classifying flood regions.
Additionally, the IDSS method utilised a decision making layer, shown in figure 2.3, which generates and applies IF THEN linguistic rules to assign classification to unknown pixels. This ensured that each classified pixel is explicitly linked to a hyper spectral band in sentinel 2 images, creating a traceable and interpretable reasoning process. Each prototype corresponded to a key feature value extracted from the sentinel 2 bands for example coastal or SWIR. This results in 500 IF THEN rules per class, which can either be applied separately or combined into single, large disjunctive rules per class, such as the following: IF (SWIR < X) AND (NIR > Y) AND (COASTAL >= Z) THEN “water”.

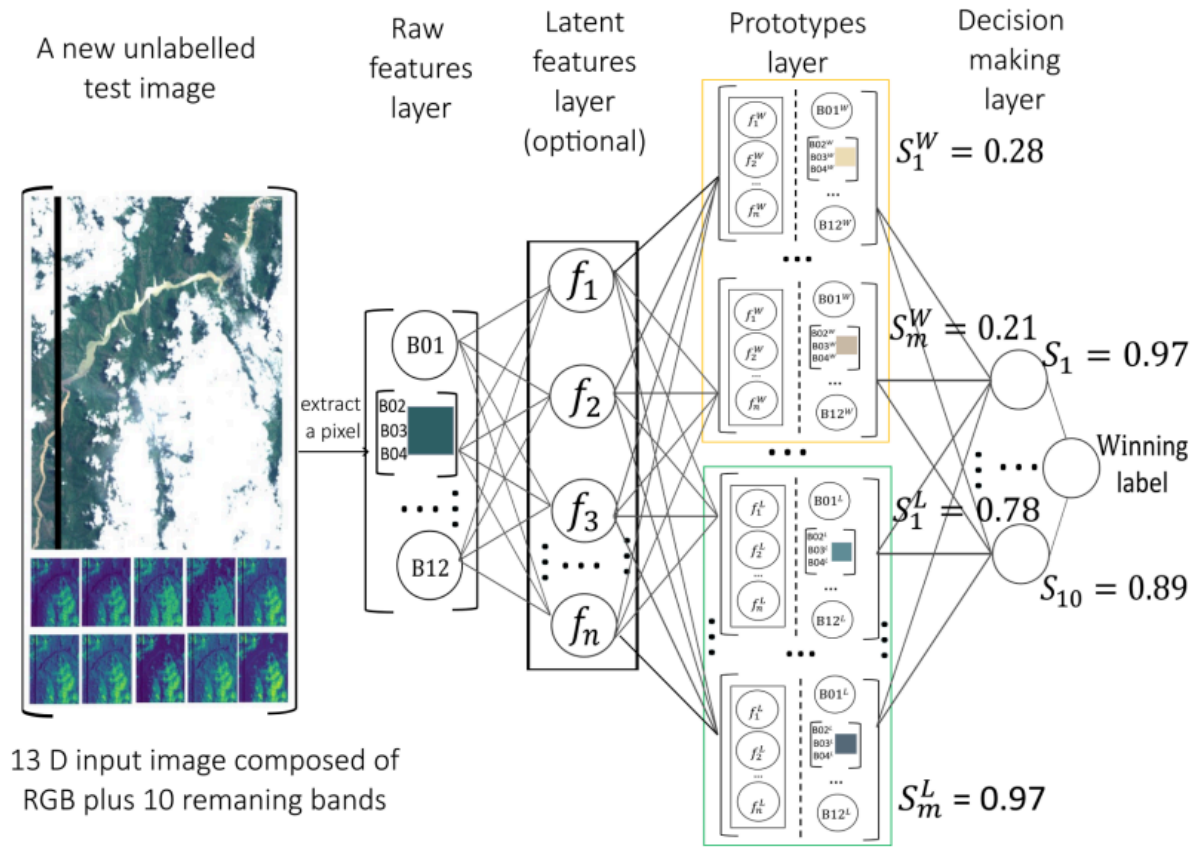



Figure 2.3: IDSS architecture

This method of semantic segmentation provides many advantages over standard approaches. Unlike standard CNN models, improved transparency is established as each classification decision can be traced to specific spectral features and prototype clusters. It provides robustness against noise within the image, by clustering data IDSS reduces sensitivity to outlier pixels that may otherwise mislead deep learning models. Achieving water IOU scores of 73.10% and a water recall of 96.11%. IOU represents the total true positive pixel classification against the true positive, false negatives and false positives, while recall represents the true positives against the true positives and false negatives, these can be seen in figure 2.4 and 2.5. This is whilst using significantly less parameters than U-net, 96K parameters compared to 7.79M, it still managed to achieve higher scores in these metrics whilst introducing an interpretability architecture to understand decision making by the model

$$\text{IOU} = \frac{\text{TP}}{\text{FP} + \text{TP} + \text{FN}}$$

Figure 2.4: IOU calculation, TP = true positive, FP = false positive, FN = false negative

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

Figure 2.5: recall calculation, P = true positive, FN = false negative

The IDSS architecture establishes an effective baseline for this paper, whilst it is very well rounded it only utilises U-net. Exploring how IDSS would work with another CNN backbone such as SegNet or ResNet, could offer insights into adaptability across segmentation models, whilst exploring possible improvements for this model architecture. It also utilises the ML4Floods pipeline, allowing easy

access to the backbone architecture of this model. However this method likely requires a significant amount of processing power or time to train and inference effectively due to the added layers in the architecture; prototype layer and decision making layer. This could be especially demanding if applied to large databases such as the full extent of the WORLDFLOODS database, but with powerful equipment utilised by Zhang such as the HECC (High End Computing Cluster) server this can be done.

# 3. Methodology

This section of the report will discuss the data used to train the model and how the dataset is collected. The model pipeline designed and utilised for processing the dataset. Training, testing and inference frameworks will be outlined. It shall also discuss certain metrics for evaluation such IOU and recall.

## 3.1 Data set

Flood segmentation from satellite imagery requires a large and diverse dataset to train models effectively. This section shall discuss the datasets used and considered for this project, sourced from the WorldFloods series. These datasets composed of sentinel-2 images and annotated segmentation ground truth masks, provide critical input data for deep learning pipelines.

### 3.1.1 WORLDFLOODS

This dataset is a massive 300Gb set of image mask pairs varying in size, split into training, validation and testing sets. Without a powerful machine this dataset is unusable, as the system would run out of memory and would take days to go through a single epoch (1 training step), especially with processing across all 13 sentinel 2 hyper spectral bands. Therefore this dataset can be utilised to perform testing on different datasets if required, grabbing different image and mask pairs in order to extend any other datasets utilised.
Water pixels are heavily underrepresented in this dataset, and unfortunately this is reflected in subsets of this dataset, with 2.7% of the training set being water pixels, opposed to the 43.24% of land and 50.25% of cloud pixels.

### 3.1.2 Worldfloods sample v1

The ML4Floods pipeline grants access to a subset of the WORLDFLOODS dataset called the worldfloods_v1_0_sample, a 12Gb dataset split into training, validation and testing splits with metadata, ground truth masks and sentinel 2 images in each. This dataset was created in 2021 as a part of Mateo-Garcia. G (2021) works of flood mapping using machine learning[19]. This can easily be downloaded, following the “Train models” dataset in step 0 of the ML4FLoods tutorials. These images do vary in size which will lead to issues in the U-net pipeline, they will need to be cropped to 256x256 (height x width) so the encoder can easily downsample the features, same goes for the decoder. Moreover, this dataset is far more realistic for training due to its size. Epoch time would be far faster, therefore increasing the amount of training time. However, with this comes a loss of diversity in a dataset that already has a severe class imbalance with water being the minority class by a substantial amount.
WorldFloods v2 is a newer version that addresses several shortcomings from its predecessor, v1 [20]. Including improved annotation quality, a broader set of flood events and a rearranged classification system. However the classification changes to the original which is what the IDSS architecture utilises, so will be omitted from the database considerations.

## 3.2 Model

This section shall outline the architectural framework used to develop the flood detection system. Beginning by detailing the ML4Floods pipeline, an end-to-end learning framework tailored to satellite image segmentation. The pipeline forms the operational backbone of this study, providing tools for model training, processing and metric tracking. Following this, the section explores how the Interpretable Deep Semantic Segmentation architecture is integrated into the pipeline. IDSS

introduces a prototype-based approach to enhance interpretability in model predictions. Together, these systems enable both high-end segmentation and traceable decision making. The subsections below discuss how these components are configured, modified and extended for use in this project

### 3.2.1 ML4Floods pipeline

ML4Floods, creates a baseline backbone for this project, its end to end architecture is crucial. This python site package, provides tutorials on how to train on datasets, utilise metrics and run inferences. However this pipeline is massive and understanding its nuances will take a significant amount of time, from configuration settings to data preparation. The essential segments of focus are elements within the models directory, this holds dataset setup, worldfloods model which holds the test step and will be the target folder for integrating the IDSS architectures. This directory also holds the architectures, holding the CNNs such as U-net, this is where the SegNet architecture will go for furthering the IDSS architecture to see if its CNN can be adapted and maybe improved. This package makes use of modules such as torch lightning numpy and tensor flow as well as many others in order to build this pipeline.

ML4Floods fortunately also handles the oversized image issue by tiling the images. This divides images into 256x256 tiles, ensuring no image or mask is degraded via other methods such as downsizing which would rearrange class labels and hindering classification or cropping which would take a long amount of time.
The configurations are essential in this pipeline. They grant access to key alterations for the model and are tailored to versions of the WORLDFLOODS database, this model uses version 1. They can grant access to data parameters such as image transformations, batch size, number of classes and number of bands/channels. Furthermore, model parameters can be altered such as learning rates, maximum epochs, metrics, learning rate decay and weighting. Weighting can be altered to ensure that underrepresented classes are rewarded more for correct learning, however this requires a balance as overweighted classes could result in higher training losses and in turn lower IOU scores. By default the configurations set the water weight to a high 36, while land and cloud are at 1.9 and 2.1, this is never stated why in however it is likely due to the class weight and would be calculated based on the class pixel ratios. Variations in the weighting might be helpful based on size of the dataset.
This pipeline doesn’t have a test step, so one will need to be integrated, this includes altering the pipeline in the site-package so that tilling can be used in the configurations and test dataloader. Fortunately this package creates a class for the model, worldfloods_model, a lightning model which will be extended for a test step. The test step will utilise the forward method from the worldfloods model ml4floods provides and utilise it for logging losses, recall and IOU during the step, returning a final set of these metrics.

### 3.2.2 IDSS integration

This project will aim to integrate this architecture into the ML4Floods architectures mentioned above and further it by experimenting with CNN adaptability.
Firstly a method for extracting latent features is required, whilst optional this project has decided to use it in order to create a more saturated feature range. This requires adapting the CNN, typically CNN classes will have a forward method which extracts raw features, for UNet this consists of downward convolutions for the encoder, then upward convolutions for the decoder. To extract the latent features we need to stop at the penultimate decoder convolution, and return these features, this returns a 64 latent feature object. As most CNNs follow this typical structure, performing this on other CNNs such as SegNet should follow the same principles.

Next this architecture utilises a prototype layer after raw and latent features are extracted. This is a critical component of IDSS and serves as the core mechanism for enhancing interpretability. Unlike traditional CNNs which rely on abstract high dimensional feature maps, the prototype layer provides

structured representations of learning features. Leading to greater transparency for the decision making layer, by associating each classification with a set of representative feature prototypes. Utilising the raw and latent features extracted, the prototype layer creates clusters of features using MiniBatch K-Means. This clustering process helps to form class specific prototypes that serve as reference points for the segmentation process.
This process will be built within the worldfloods model class, firstly using a function called update_prototypes, which governs the adaptation of prototypes by iteratively refining them based on new feature observations every 50 batches. First the features are flattened and normalised. The latent features are reshaped and normalised to ensure numerical stability and prevent issues caused by NaN, functions such as torch.nan_to_num() are utilised to do this. Clustering is done per class, iterating through each class and selecting corresponding raw and latent features, the MiniBatch K-means algorithm groups features into representative clusters. If a class has an insufficient number of cluster samples it is skipped to prevent inaccurate prototype information. Updates utilise momentum, prototypes are updated using a weighted combination of old and new cluster centers, this is done using the formula shown in figure 3.1. The same formula is repeated in the raw feature space, ensuring that prototypes remain interpretable in relation to spectral band inputs.

$$P_{new} = \alpha P_{old} + (1-\alpha) P_{clustered}$$

Figure 3.1: update prototypes formula, α represents momentum, P represents Prototype

The final prototype clusters are normalised to maintain numerical stability and improve distance based similarity scores.
Once prototypes are established, they are utilised in a classification phase through the classify function. This function computes prototype distances, using functions such as torch.cdist, which measures the Euclidean distances between latent features and prototypes. Similarity scores are computed using these distances, ensuring that closer prototypes exert a higher influence on final classification. The top 10 nearest prototypes are selected per pixel, each prototype contributes to a class vote, weighted by its similarity score. The final class prediction is determined based on the accumulated votes. This function utilises a fallback mechanism, if a pixel has a high distance to all prototypes, it is assigned to the invalid class, which is processed later on during the decision making layer.

By leveraging clustering rather than raw feature averaging, this layer enhances interpretability by associating each segmented class with a representative prototype. These prototypes not only influence pixel wise classification but also serve as an auditable decision making structure, making it easier to validate and understand predictions. The prototype layer aligns with the interpretability goals of IDSS as it provides a clear mapping between prototypes and class labels.

After the prototype layer comes the decision making layer, this layer is performed during testing and inference, as utilising it for training is not part of the IDSS architecture, furthermore it would take an exponentially long time to use it per training step. This layer is responsible for assigning class labels to unclassified pixels based on extracted raw and latent features, prototype similarity and rule based refinement. Further refining segmentation output by leveraging interpretable rules, prototypical similarity and K-Nearest Neighbors (KNN) refinement. Firstly, the decision making layer uses a function called Interpretable_rules, a rule based classification using IF-THEN logic applied to spectral features. This approach enhances interpretability and ensures that semantic segmentation aligns with human explainable physical properties. Features are extracted from bands such as coastal, shortwave infrared (SWIR), the RGB bands etc, some of which are utilised to make certain indexes. Normalised Difference Water Index (NDWI) [22] and Normalised Difference Vegetation Index (NDVI) are computed using the formula shown in figure 3.2 and 3.3. These are commonly used indexes which help determine water (NDWI) and land (NDVI).

$$NDWI = \frac{(G+NIR)}{(G-NIR)}$$

Figure 3.2: NDWI formula, G stands for Green band, NIR stands for Near Infrared Band

$$NDVI = \frac{(NIR-G)}{(NIR+G)}$$

Figure 3.3: NDVI formula, G stands for Green band, NIR stands for Near Infrared Band

Rules are applied sequentially in the following order to ensure hierarchical decision making; cloud rule, land rule and water rule. Originally, this function iterated over each pixel and each band, but this took exponentially longer, so a transition to a masking technique. A mask is applied over the predictions of pixels with values that follow the IF-THEN rules, this mask labels pixels that are unclassified into either the land, cloud or water mask. This works far better with tensor due to its affinity to batch processing[23].
Following initial rule based classification, stored rules from previous training iterations are applied via the apply_stored_rules function. This step ensures that misclassified or further unclassified pixels are corrected based on learned patterns from training. It uses batch processing to extract latent features and flatten predictions so they can be used… Furthermore, it utilises tensor based operations to apply stored rules in parallel for further efficient batch processing. Utilising prototype based rule matching, each class has a precomputed set of IF-THEN rules based on learned prototype distributions, which is compared to each pixels' feature value.
Finally, a KNN based label assignment is applied, each pixel is assigned to the majority class of its 10 nearest prototypes. If no close prototype exists, the pixel is classified as invalid. This last layer is applied to the test and inference layers, whilst some IF-THEN rules are stored during each training epoch for that added learning comparison in the apply_stored_rules function, they predominantly are utilised as a validation process in these later model steps. This model doesn't utilise a validation step, just training and test steps and a final inference.

## 3.3 CNN

This section shall discuss the Convolution Neural Networks utilised in this model. U-net is the primary backbone architecture for both ML4Floods and IDSS. SegNet has been adapted to fit within the IDSS framework.

### 3.3.1 U-net

ML4Floods implements U-net as the primary backbone architecture for semantic segmentation. U-net is a fully convolutional network (FCN) originally designed for biomedical image segmentation. However, it has been widely adopted in remote sensing applications due to its ability to efficiently capture spatial context and produce high resolution predictions.
The ML4Floods implementation of U-net follows a symmetrical encoder-decoder structure, see figure 2.1 for visualisation, where the encoder downsamples and extracts hierarchical features using

convolutional layers followed by max pooling. The decoder upsamples and reconstructs spatial details using transposed convolutions and skip connections.
This implementation contains the following key components; double convolution blocks, where each encoding and decoding step consists of two consecutive convolutional layers, implemented using layer_factory.double_conv, an ML4Floods utility. Max pooling for downsampling where feature maps are progressively reduced using MaxPool2d(2), reducing by half every time. Skip Connections for information retention, feature maps from the encoder are concatenated to decoder layers at corresponding levels to preserve fine grained spatial detail. Bilinear upsampling for the decoder, instead of transposed convolutions bilinear interpolation is used for smooth upsampling.

In order to extract latent features from U-net for IDSS integration, features need to be extracted from the penultimate layer of the decoder. A function called get_penultimate_layer has been created to perform this. It follows the structure of a forward pass, except it stops the pass at the last convolutional block, before the final segmentation layer. This results in a 64 dimensional feature map, ensuring compatibility with the IDSS prototype layer, enabling further feature saturation in the prototype based clustering decision refinement.

### 3.3.2 SegNet

The SegNet model utilised was adapted from a pytorch based implementation of SegNet [25]. Changes had to be made in order to fit the 13 dimensional feature space, similar to the U-net model it takes in the number of bands as dimensional input, this is acquired from the configurations created by ML4Floods.

While the U-net serves as a baseline, this project explores the adaptability of SegNet as an alternative CNN for segmentation. SegNet is particularly suited for efficient inference as it utilises max pooling indices for upsampling, reducing computation, and eliminates fully connected layers, U-nets skip connections, making it lightweight.
SegNet's architecture for this implementation consists of a five stage encoder decoder structure. The encoder uses multiple convolutional layers per stage, each followed by batch normalisation and ReLU activation. Following this it then downsamples feature maps using max pooling with indices, which are later used in the decoder. The decoder performs max unpooling using stored pooling indices, then reconstructs high resolution features using convolution layers.

Akin to U-net, this SegNet architecture is adapted for latent feature extraction and also implements a get_penultimate_layer function. This function returns feature maps before the final classification layer, ensuring consistency with U-net, extracted latent features are fed into the prototype layer enabling clustering and rule based classification. This further maintains interpretability unlike traditional CNNs, SegNet in this setup is designed to work within the IDSS framework ensuring interpretable, traceable segmentation.

In order to be used by the worldfloods_model in ML4Floods, and in turn the IDSS architecture with it, changes need to be made to the ML4Floods pipeline. Within the worldfloods_model, exists a global method called configure_architecture which accesses the model parameters of the configurations json created by ML4Floods. Within this method is a selection statement which scans the model parameter "architecture" for the type of backbone architecture it shall use (unet, simplecnn, linear, etc). In order for SegNet to be utilised by the worldfloods_model class which implements the IDSS architecture, an additional selection statement needs to be added, scanning the "architecture" configuration for segnet. This then assigns the model as SegNet with the number of channels and classes extracted from the configurations. In order for this to work the SegNet class needs to be in the same directory as the other architectures. As ML4Floods is a site package for python, this needs to be done in the site

packages for python, and is located in the following directory “ml4floods/models/architectures”. Once this is done the worldfloods_model can utilise SegNet with no other integration requirements. This demonstrates ML4Floods’ adaptability, allowing PyTorch compatible CNN that follows this structure to be seamlessly integrated into the pipeline ensuring compatibility with IDSS.

## 3.4 Metrics

This section outlines the performance evaluation metrics used in this project. The segmentation model is assessed using standard metrics for classification and segmentation tasks. These metrics are essential for quantifying the model’s ability to correctly classify pixels into predefined categories such as land, water, cloud and invalid. Key evaluation metrics used include Intersection over Union (IoU), Recall and loss functions.

### 3.4.1 Intersection over Union

Intersection over Union (IoU) is a widely used metric in semantic segmentation to assess the overlap between the predicted segmentation mask and the ground truth. The formula can be seen in figure 2.4. In this project, IoU is computed for each class (land, water, cloud) separately, and the main IoU is reported to summarise overall performance. This project will predominantly report on water IoU as this is the main focus for the model, being able to identify areas of flooding, by definition areas of water, so correct identification of water is crucial.The function calculcate_iou in inference.py ensures that IoU is correctly normalised across batches in inference. IoU is computed at multiple stages during the model pipeline. At the end of each training epoch the average of the top 20 batches with the highest IoU is reported, this is to show a more representative IoU score for each epoch. By default the last batch’s IoU score is reported but this isn’t very representative. During testing, it is calculated per batch to evaluate generalisation performance. During inference it ensures water detection accuracy across real-world flood images.

### 3.4.2 Recall

Recall measures the proportion of actual positives correctly identified by the model. The formula can be seen in figure 2.5. This metric is especially crucial in flood detection since failing to detect water pixels (low recall) could have severe consequences. This is computed at every stage of the model akin to IoU, training, testing and inference. Furthermore, this metric is also computed per class.

### 3.4.3 Loss Functions

The model training process involves multiple loss functions to optimise performance. Binary Cross Entropy (BCE) loss is used for class wise pixel classification, ensuring each class is correctly segmented. Prototype loss ensures that the extracted latent and raw feature cluster closely around their respective class prototypes. Weighted loss, the model applies different weights to each class to compensate for class imbalance, this is acquired and set in the configuration json ML4Floods creates, particularly since water pixels are underrepresented in the dataset. These losses are computed within the compute_losses function within the worldfloods_model class.

### 3.4.4 Confusion matrix

A confusion matrix provides a detailed breakdown of how predictions are distributed across actual classes. It allows the identification of common misclassification and model biases. This is performed during inference, the compute_confusion_matrix constructs this matrix for analysis. The confusion matrix is also visualised using heatmaps for deeper insights into classification performance.

### 3.4.5 Model Evaluation Workflow

Model performance is evaluated in the test step in the floodmodel class, a wrap for worldfloods_model allowing for the test_step to be integrated into the worldfloods model. This step computes the IoU to measure segmentation accuracy, recall to assess model sensitivity and loss metrics. This is performed for each batch of the test data. The best performing model checkpoints are selected based on train_iou_water, ensuring that the model prioritises water detection. Early stopping is applied using train_iou_water to prevent overfitting during training.

# 4. Development

## 4.1 Development Introduction

The development section of this project shall highlight the creation, training, testing and inference of the proposed segmentation model, that integrates the IDSS architecture into ML4Floods, while replacing U-Net with SegNet as the CNN backbone. It delves into a much finer level of detail than the previous methodology section.
All development was written in Python using CUDA ToolKit and Pytorch, with training on a NVIDIA 3070 ti with 8Gb of V-RAM, with 32Gb system memory. Libraries such as TensorFlow and PyTorch Lightning were utilised for developing the deep learning model due to their support for model training, logging and performance tracking. These libraries provide a variety of tools necessary for ML4Floods to run and for the IDSS architecture to be implemented.

## 4.2 Model proposal and justification

The proposed model for this project aims to implement the IDSS architecture and further its architecture by implementing a new backbone CNN architecture, SegNet. SegNet was selected as an alternative CNN backbone due to its efficiency in semantic segmentation and its ability to reduce memory consumption compared to U-Net. Unlike U-Net, which relies on skip connections, SegNet stores max pooling indices from the encoder and reuses them in the decoder. This eliminates the need for complex learned upsampling filters, reducing memory footprint and improving inference speed. As a result, it is suitable for real time flood mapping, as its low memory requirements make it a strong candidate for real time flood detection when deployed on high performance hardware.

## 4.3 Dataset justification

This project utilises worldfloods_v1_0_sample as its final training dataset, a 12Gb dataset consisting of a training set of X image mask pairs utilised in the training of this model, a testing set of 11 image mask pairs utilised in testing and a validation set of 6 image mask pairs used in the inference stage of the model. This dataset was selected over the complete WORLDFLOODS dataset due to its more manageable size, a single training epoch on the WORLDFLOODS dataset took 26 hours. While the WORLDFLOODS dataset is more diverse and expansive, this dataset would not fit within the timeframe of this project, due to how long each training epoch would take. Subsets of the dataset were experimented with, seeing whether slightly larger test sets would be worth training on but ultimately were omitted for the final training cycles of this model due to time constraints. Within IDSS it is explicitly stated that images are either cropped to 256x256 or padded to 256x256. During research and experimentation with ML4Floods a tiling where images are broken down into 256x256 tiles so that the CNNs can break down the images evenly. This is because of the downsampling process, which breaks down features in factors of 2, this is mirrored for the upsampling process. ML4Floods performs this on the training and validation dataloaders by default.

## 4.4 Getting to grips with ML4FLOODS

In order to gain understanding for the ML4Floods package, a considerable amount of time was dedicated to understanding its provided tutorials and nuances of its package before implementing the IDSS architecture. The ML4Floods didn't implement a test step so this had to be integrated, this is done within a wrapper for the model in the model_trainier.py, in a class called floodmodel. Experimentation with different weights, batch sizes, learning rates, maximum epochs and learning

rate decay values were performed during this time, to better gain understanding the effect of each. These changes were made simple to make due to ML4Floods configuration json, these model parameters were set and altered within preprocessing.py During this stage of development a final water IoU score of 0.701 was achieved with a batch size of 16, learning rate of 0.003, and a water weight of 3.75 (other class weights remained at default) after 30 epochs. This was as high as it managed to achieve, and was the only time it ever exceeded 0.7. However this was without IDSS implemented, displaying what pure CNN models can achieve, but it didn't offer any interpretability, the predominant goal of the IDSS architecture.

## 4.5 IDSS implementation

With a foundational understanding of ML4Floods, the next phase involved is implementing IDSS. This required integrating its latent feature extraction, prototype layer and self validating decision making layer, modifying ML4FLoods training flow, and ensuring compatibility with SegNet. Unlike traditional CNN models, IDSS does not use a validation step, as its decision making layer inherently validates classifications. Model validation was performed during the test set where the decision making layer was utilised. The training steps only extended up to the prototype layer. However, IF-THEN rules are stored for later validation in the test step. This is so that the decision making layer can make more informed decisions based on learned prototype distributions.

After further understanding ML4Floods and its overall architecture the development of IDSS was started. IDSS gives an optional latent feature layer, this was added into the model in order to increase the feature saturation on the smaller dataset to help compensate for the decrease in diversity during training. The latent features are acquired within the get_penultimate_layer() functions in both the U-Net and SegNet classes in their respective python files. These are then utilised in the prototype layer in the IDSS architecture. This replaces direct classification with a clustering based approach, improving interpretability and increasing tracing. Firstly features are normalised using torch.nan_to_num, removing NaN features, MiniBatch K-Means is then applied on prototypes for clustering. Finally prototype voting is applied for classification, the top 10 nearest prototypes are selected per pixel, distance based softmax weighting assigns final class labels. IDSS then has a self validating layer, the decision making layer where unclassified pixels are classified based on prelearned prototype distributions, as well as IF-THEN rules extracted from spectral bands for classification. NDWI and NDVI are also calculated to further help these interpretable rule classifications, the formula for these calculations can be seen in figures 3.2 and 3.3. Finally a KNN based label refinement, where each pixel is assigned the majority class of its 10 nearest prototypes.

### 4.5.1 Training and hyperparameters

For training with this architecture the following hyperparameters were utilised. For data parameters a batch size of 8, a number of 4 workers, training transformations of normalisation, random flipping and random brightness were used. For validation and test transformation only normalisation was set to true, this was so that the training dataloader was more diverse, whereas the validation and test dataloaders were kept unchanged so that evaluation wasn't degraded. Model parameters can be seen in table 4.1. Here is a breakdown of the crucial parameters; added parameter latent_dim which represents the expected latent dimensions from the CNN extraction, learning rate is the rate the model learns, learning rate decay gradually reduces learning rate during training, learning rate patience dictates when learning rate decay occurs, early stopping patience dictates after how many epochs where the monitored metric doesn't improve the model stops training, a maximum epochs dictates how many epochs are performed for a complete training loop, model type dictates which CNN is used for the model, batch size represents how many training samples are in a batch, metric monitor is the metric the model bases its learning success on. The rest of the model metrics were set to the

ML4Floods configuration default for the worldfloods v1 dataset, apart from validation metrics such as val bce loss and val loss were omitted as this model doesn't use a validation step. Training had further configurations: a checkpoint callback and early stop callback. These dictate how the checkpoints are defined and by what metrics they are deemed effective checkpoints, as this project's dominant evaluation metric is whether it can effectively determine areas of water, water IoU was selected. Checkpoint callback is set to save the top 3 training cycles with maximum train water IoUs, and records based on every epoch due to this model not having a large quantity of epochs. Metric logging is performed every 10 batches to ensure consistent logging throughout a training epoch without utilising too much storage and slowing down the training cycle as opposed to logging every batch. 45 epochs were selected as training time varied between 7-20 minutes per epoch depending on the CNN utilised, so in order to perform a significant training cycle without exceeding the time frame of this project this was the selected training time for the last few train cycles for the finalised model.
A single training epoch consisted of the following flow. Firstly the image and mask (ground truth) were extracted from the current working batch. Raw and latent features were extracted from the images. Every 50 batches prototypes are updated using update_prototypes, any increase in frequency would lead to massive increases in training time, any decreases in frequency the prototypes would be updated too infrequently, decreasing cluster learning. During prototype updates, raw and latent features are clustered based on class which are flattened for processing. A class mask is created based on the current batch's mask and class IDs for class mappings. Each class is iterated over and features are extracted belonging to that class, mini batch K-means is then performed on these features to cluster them together. Momentum based update for clusters is then performed, see figure 3.1 for the formula, this ensures prototypes evolve gradually rather than changing them abruptly. At the end of this function, an additional function called add_rule is called, parsing in both class_id and the prototype cluster corresponding to it. The add_rule function automatically generates classification rules based on the distribution of learned prototypes, which are then stored in a structured format for later use during the decision making stage. It first extracts feature information from the clusters, by extracting the prototypes belonging to their respective class. For each feature dimension, it calculates the mean feature value across all of the prototypes within that class and determines the standard deviation. Feature ranges are established using the following equation, shown in figure 4.1.

$$\text{Feature Range} = (\text{Mean} - \text{Std Dev}, \text{Mean} + \text{Std Dev})$$

Figure 4.1: Calculation for feature range

This ensures that rules cover the most typical range of feature values without being influenced by extreme outliers. If no existing rules exist for a feature a new entry is initialised in self.rules. Rules are formatted into a structured condition, seen in figure 4.2. This ensures the model can associate learned feature distributions with class specific assignments. Finally rules are stored in self.rule_storage, enabling retrieval and application during the decision making layer.

$$(\text{feature\_0} \sim 0.123 - 0.456) \text{ AND } (\text{feature\_1} \sim -0.789 - 0.654)$$

Figure 4.2: Rule structured condition format

Losses are then calculated using compute_losses, this calculates cross entropy loss, this is for pixel wise classification, and prototype loss, which encourages features of the same class to cluster around prototypes by computing distances between features and their assigned prototypes. The higher the distance the higher the loss. After losses are computed the classify() function is performed in order to obtain predictions. This uses the prototype distances to assign class labels, firstly euclidean distance is computed between features and prototypes, distance is converted to a similarity score. After which,

K-nearest voting is performed on each pixel, which is assigned to the top 10 nearest prototypes, this utilises a weighted voting to determine final classification. Any pixels which don't match any prototypes or are too far from any prototype are assigned to invalid classes. This covers the training architecture of IDSS, which includes everything up to the prototype layer, see figure 2.3, the decision making layer is performed later on.

| **Parameter** | **Value** |
|---|---|
| Batch size | 8 |
| Latent Dimension | 64 |
| Learning Rate | 0.0003 |
| LR decay | 0.1 |
| Maximum Epochs | 45 |
| Early stopping patience | 15 |
| Prototype Count | 600 (200 per class) |
| Metric monitor | train_iou_water |
| Model type | "unet" or "segnet" |

Table 4.1: Table of training model parameters

### 4.5.2 Testing

Testing follows the same core steps as the training, with the addition of the final decision making layer added onto the framework, shown in figure 2.3. The test step is implemented within the floodmodel wrapper that was built during the ML4Floods familiarisation. The purpose of the test step is to evaluate the trained model on the test dataset to measure its performance. It follows similar steps to the train step, where the batch's image and ground truth mask are extracted and processed the same as the train step. Latent features are extracted from their respective backbone. The step forwards the model through the CNN backbone and interpretable_rules() is called. This is where the decision making layer kicks in and unclassified pixels are properly resigned from the invalid class to a valid class.

The CNN parses the features into interpretable_rules, which it uses to run the classify() method to retrieve predictions. The CNN and prototype system classifies most of the pixels correctly, but some remain unclassified. Spatial bands are extracted from sentinel-2 in order to be applied to IF-THEN rules, each band contains unique spatial information for each class, for example in the NIR band vegetation is highly reflective. NDWI and NDVI are calculated here using the green and NIR bands, as they are particularly useful for classifying land and water. The calculations for these can be seen in figures 3.2 and 3.3. A check is performed to scan for unclassified pixels as if there are no unclassified pixels there is no reason to continue with that batch, so it skips this set of interpretable rules by returning the predictions, therefore jumping back to the rest of the test step. Rules are applied using a masking system. Firstly a cloud mask is created based on cloud rules, this mask is applied over predictions, looking for pixels that match the conditions within the band thresholds, this can be seen in figure 4.3. Prediction values that fit this are assigned the value 2, representing the cloud class.

If (SWIR1 > 0.7) OR (SWIR2 > 0.8) AND (Blue > 0.3) AND (Red > 0.3), assign Cloud

Figure 4.3: IF-THEN rule, demonstration example thresholds for cloud rule

This is performed for the land and water class, with their related band thresholds. After these masks are applied to predictions, they are further refined using the stored IF-THEN rules from add_rule function within the prototype layer in training. This is done within apply_stored_rules() which checks previously learned feature distributions and applies feature range classification. This helps refine edge cases that don't fit the original rules and correct missclassified pixels. It makes use of batch processing for efficiency, applying rules in parallel. The Importance of this function lies within flaws within the prototype layer, as classification there may misclassify certain pixels that don't fit perfectly into clusters. If there are no rules it skips this process and returns predictions for the interpretable_rules function to continue its processes. A tensor based rule_mask stores whether a rule applies to each pixel, and a copy of predictions is made so we can modify it with the learned if then rules. Using the rule_mask ensures that only pixels matching a rule are modified, preventing overwriting correct pixels. Each class and its respective stored rules are iterated over to check if rules exist per class, if they don't the class is skipped. For each rule feature constraints are extracted (their minimum and maximum value per feature), then a check is performed to see if each pixel's feature values fall within the prototype range. If a feature value falls within an expected range, the rule is applied. Pixels that match a rule are updated with the corresponding class_id, only the pixels that match are updated in the predictions.

Finally knn_decision_making is applied. KNN voting is used to refine predictions based on prototype similarity, ensuring that pixels not covered by rule classification are assigned to their most similar class. Some pixels remain ambiguous for strict rule classification. Firstly prototypes are checked to see if they exist, preventing runtime errors. If there are available prototypes, KNN is trained using stored prototypes, mapping features to their respective class labels to allow similarity based classification. Once KNN is trained, each test pixel is compared to stored prototypes, and the top-K nearest prototypes are retrieved. The euclidean distance between the pixel's latent feature and each prototype is computed, and the class of the closest prototype is assigned. However before applying these predictions, a validation is performed to ensure that the predicted class exists as a label. Additionally, a confidence threshold is introduced to refine the process further. If the nearest prototype is too far from the pixel, the classification is not updated, as a high distance indicates a lack of similarity between the pixel and its closest prototype. This threshold is set to 0.4 euclidean distance, ensuring that only pixels with strong prototype similarity are reassigned. If the distance is below this threshold, the predicted label from KNN replaces the previous classification. However, if the confidence is low the original prediction remains unchanged to prevent the introduction of uncertainty into well classified regions. KNN voting ensures that pixels not fitting prototype rules are still correctly assigned based on their nearest learned prototypes, improving segmentation robustness, particularly in regions where class boundaries are unclear or misclassified during rule application.

After performing the decision making layer the test step finalises by reshaping the predictions for processing, losses are computed using compute_losses which are then logged for the model. These are the same losses as the ones used in the training step. Confusion matrix for class level analysis is then computed using ML4Floods inbuilt compute_confusions within metrics.py. This function takes in the mask, predictions and number of classes available, this is done so it can be shown which classes are misclassified most often and helps identify common errors. IoU and recall scores are then calculated per class. Finally some added debugging visualisations are implemented to visualise latent features and example images for analysis. Latent feature distributions are logged using TensorBoard SummaryWriter, this helps visualise how feature representations evolve during training and testing.

### 4.5.3 Inference

Inference follows a similar structure to the test step but has some added benefits. This is the final stage of the model, and is the evaluation, where trained the segmentation model is applied to unseen data to assess its generalisation capability.The purpose of inference is to evaluate IDSS architecture with different backbones on unseen data, performing a blind test, this generates the visualisations of segmentation, see figure 4.4. Furthermore IoU and recall are calculated here to measure model performance and generate a confusion matrix heatmap of the inference, to further add visualisation. Inference is performed on the unused validation dataset, and uses the same dataloader initialisations as train and test step, ensuring compatibility with the models architectures. The dominant function within the inference.py file is process_inference(), this function controls the majority of the workflow. It starts with initialising a list of available labels (land, water, cloud, invalid), an array of images to plot for inference, an array for original images, and an array for matrices to be stored. Image and mask pairs are extracted from batches within the dataloader, which are then prepared for processing. It then creates predictions using the trained model, which is initialised as a floodmodel and loaded into the current working cuda device, a . The model is set in evaluation mode. Label encodings are assigned to specific classes, if a prediction is equal to 120 then it is assigned to class 0 (land), if they are equal to 141 then it is assigned to 1 (water), 36 is assigned to 2 (cloud) and 84 is assigned to 3 (invalid). This ensures labels are mapped to the correct label range. In order for visualisation to be effective inference images and original images are appended to image plot array and original images array, all 6 images in the validation dataset are stored for this.

Confusion matrices are calculated per batch, compute_confusions from the metrics class in ML4Floods, to ensure consistency throughout the model, once computed the batch's matrix is appended to the mets array for later use. Mets are transformed to a tensor stack so that all the batches can be aggregated, creating a global confusion matrix. Using this matrix IoU is then calculated per class, this is calculated using another metrics function, calculate_iou. After these are calculated a heat map confusion matrix is calculated using the global confusion matrix, this is done using plot_confusion_matrix. This function utilises modules such as matplot and seaborn to create and plot this confusion matrix, seaborn's sns heatmap function is utilised to generate a clear understanding of more effective values for the confusion matrix. Values in the confusion matrix are normalised to a percentile value to match IoU format. An example of the utilised confusion matrix can be seen in figure 4.5.

After the confusion matrix plot is performed, the image inference and original images are displayed using, process_and_plot for the array of stored image inferences, and visualise_original_images on the original image array. These functions also both use matplot to plot and display images. Process_and_plot takes a list of segmentation predictions as an input, which it then plots using a custom colour map to represent the available classes. Firstly it sets up a plot grid to plot all available images, 3x2 as the validation set holds 6 images. Colours are defined in order of the labels (label order is land, water, cloud, invalid), colours selected are brown, blue, white and gray, representing the class labels in that order. A colour map is then created using these defined colours, with boundary normalisation set ensuring the colours are evenly distributed across class boundaries. The predictions for each image are iterated over, after a validation check to see if predictions and images are available, and are converted to unsigned 8-bit integers for display. Predictions are displayed using imshow(), with the custom colour map and normalisation applied, axis are also set to off as there isn't much need for X and Y axis in an image plot, this also presents a cleaner display of the plot. Finally a colour bar is added as a key to help represent class labels and their corresponding colours. An example of this plot can be seen in figure 4.6.

Following this the original images are plotted in the same format in order to reflect which image inference relates to its corresponding original image. This is done using visualise_original_images, which takes in a list of images, validates if images exist then plots them in a 3x2 format. Original images are plotted in rgb bands (B1,B2,B3), images are then normalised for display and clipping is applied to ensure all pixel values remain within the valid range for correct display. Images are then shown using imshow() in a similar format to the inference images. This can be seen in figure 4.7.

A final set of IoU and recall for each class is outputted to show the final outcome of the model's inference. IoU scores are retrieved from the diagonal of the global confusion matrix, and recall is calculated, then displayed for each class. After which plt.show() is run to display the computed figures (inference, original images and confusion matrix).

## 4.6 Challenges and refinements

Throughout development, different methods were attempted and changes from the IDSS architecture had to be made in order for the model to be run on the utilised machine. This subsection of the Development shall go over these methods, changes and observations made.

During the familiarisation with ML4Floods a large gap in knowledge was discovered. How to implement, train and properly use lightning modules and ML4Floods architecture, whilst the tutorials were useful they only provided so much. In order to get to grips with this package, all the following tutorials were attempted; ML model development “Train models”, “Model Metrics”, “Run inference”. After completing all of these an attempt at a pure CNN model was attempted to test knowledge, whilst building this, it was discovered that ML4Floods doesn’t implement a test step. In order to ensure the test step worked, a floodmodel wrap was created to extend the worldfloods_model and held as mentioned in section 4.5.2 Testing. This proved to still be an issue as ML4Floods performs a lot of advanced preprocessing, providing an even higher knowledge gap, and this was only performed on training and validation dataloaders. Within the ML4Floods package there is a custom lightning module called worldFloodsDataModule, within this the train, val and test dataloaders are set up. In order to use tiling in the test step, within the setup method of this lightning modules image_files needs to be replaced with list_of_windows, this line is shown in the val and train dataset set ups in this method, and can be copied over, all that needs to be changed is adding in the test_files as an argument. Once this was completed the test step worked completely fine. In this phase the test step worked similar to the validation step in the original ML4Floods package, with some added test metrics for evaluation. This laid the groundwork for the IDSS implementation, establishing a code clean flow which can be seen in main.py, however it is now adapted for IDSS and doesn’t reflect the ML4Floods familiarisation phase.

IDSS explicitly describes using 1500 prototypes in total during its prototype layer, 500 per valid class, in order to cluster features extracted. However during experimentation with model configurations and hyperparameters, the implied total prototypes were impossible to use. This was because the GPU utilised in the machine used for this project did not have enough V-RAM to load and process this amount of prototypes during training epochs, leading to termination during the first epoch. The final model uses 600 prototypes, 200 per valid class, as it best fits a balance of feature representation and epoch duration. A total of 900 was trialled, however this greatly extended the epoch team to almost more than double, putting strain on the time frame of this project by greatly increasing the training of model time. Furthermore 300 total prototypes were also trialled, but lead to lower feature clustering so it was discarded and the middle ground of 600 was used.
Due to this change the K value in the KNN decision making layer of the model had to be scaled down. This was done by dividing the original number of prototypes by the current number of prototypes and the value floored, leading to K=2. K=3 was trialled but IoU scores became 0 so was ultimately discarded as a value.

Add in interpretable rule method changes, pixel by pixel, band by band to masking
During the building of interpretable_rules in worldfloods_model, the initial method was scrapped. Initially predictions were gotten using the classify method and prepared for processing, this remains the same. The original section iterated over each pixel in the batch NDWI and NDVI was calculated

and bands were extracted. A rule based override was then applied to the batch's pixel. If it met the conditions, the pixel was classified as the class that fit the prediction, such as high SWIR suggesting clouds. However this was computationally intensive and increased the test and inference steps exponentially in time complexity, so much so that a full run was never completed as it would exceed a day just to run a test step. It also iterated over every pixel, including pixels that were already classified, overwriting classifications that weren't labelled as invalid or 3. The current method has been used in replacement. Not every band of the sentinel 2 images are utilised for the mask applications during the current interpretable rules as not every band provides useful information for land, water or cloud application. However this doesn't mean that all bands aren't considered during the testing and inference validation step, the learned IF-THEN rules from training consider every band, so any feature or band patterns are taken into account in the decision making layer.

This implementation does have differences from Zhang (2022), such as 600 prototypes instead of 1500, a less powerful machine and a significantly smaller dataset. These factors all affect the model's performance, and therefore will likely lessen the results of this study, however this doesn't mean that the results are not useful. They still provide useful information on how the model could perform on the full implementation with a more powerful machine as using these would only improve the model.

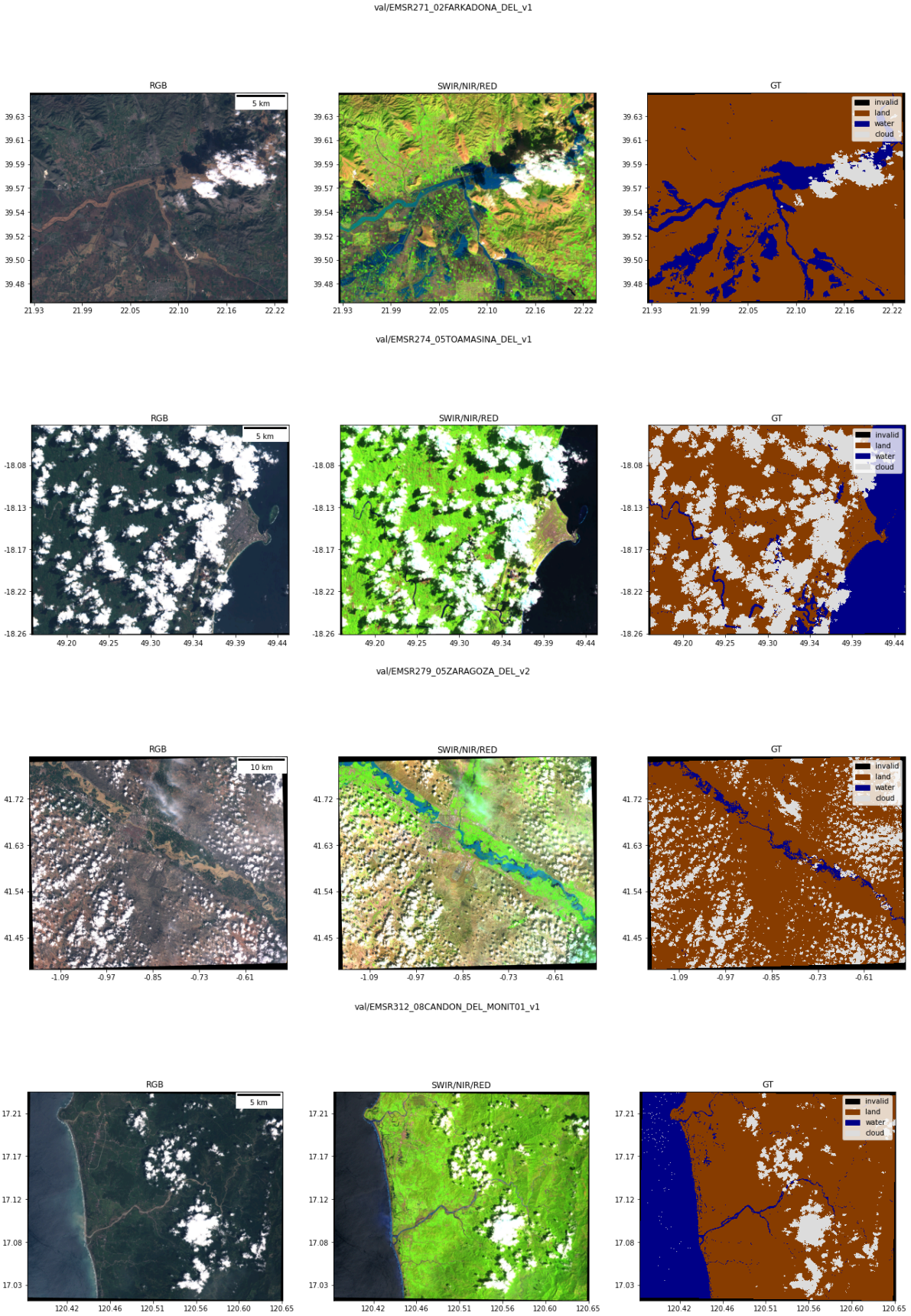



Figure 4.4: Set of inference examples taken from ML4Floods exploring WorldFloods [19, 20, 21], the figure shows RGB bands, SWIR/NIR/RED bands and ground truth inference of sentinel-2 images in the described order

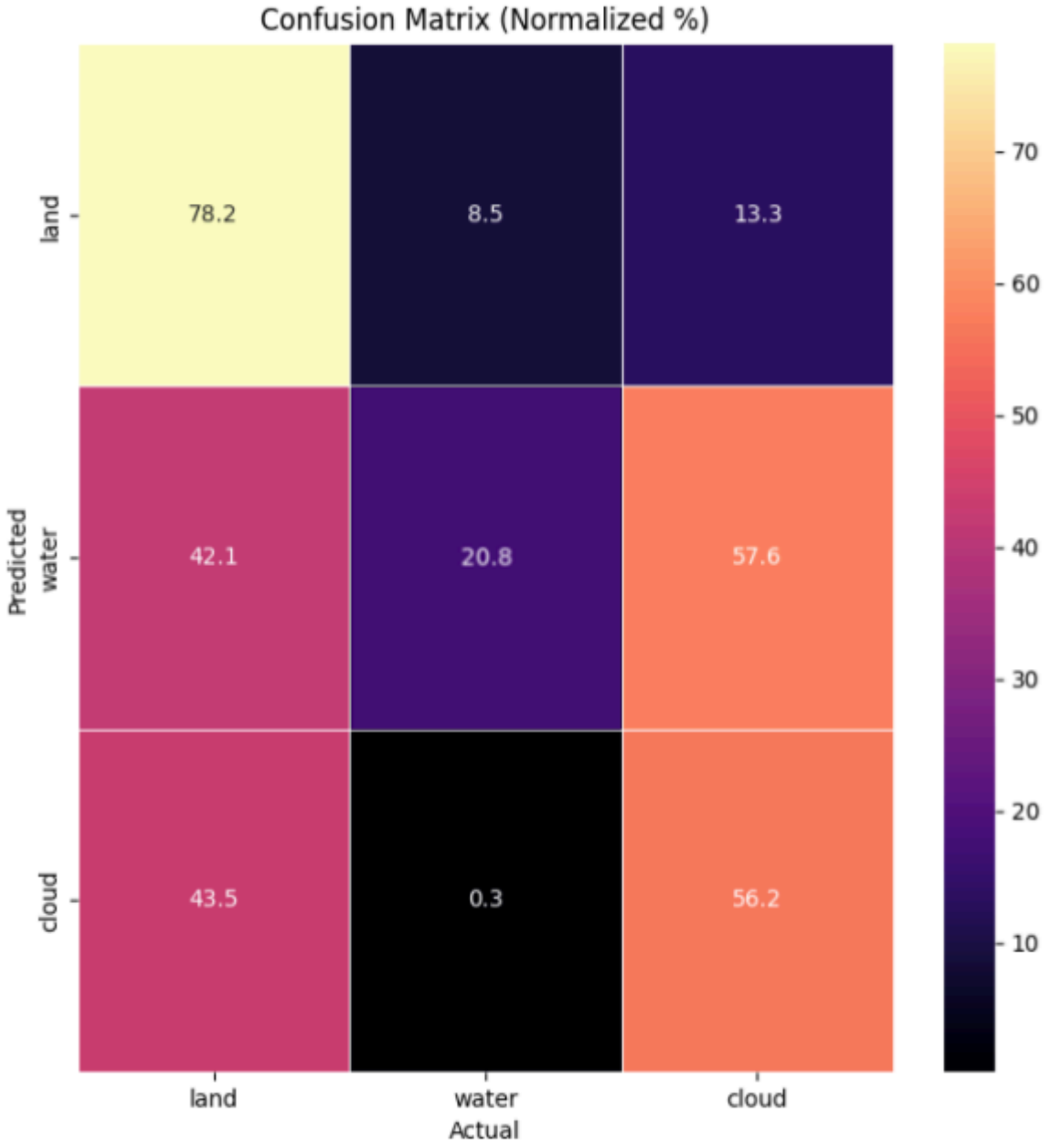



Figure 4.5: an example of the heatmap confusion matrix utilised in the inference stage of the model. X axis represents actual ground truth, Y axis represents model predictions. The diagonal cells starting from top left and ending in bottom right represent correctly classified classes. In this example the top left cell represents correctly classified land pixels, middle represents correctly classified water pixels and bottom right represents correctly classified cloud pixels. Other cells represent misclassifications, for example the cell for row “predicted water” and column “actual land” represents the percentage of land pixels classified as water, these are known as false positives. The bar on the right represents the heatmap colour mapping, the darker the lower the percentage and the lighter the higher the percentage. True 0 values are represented as a blank white cell with no value.

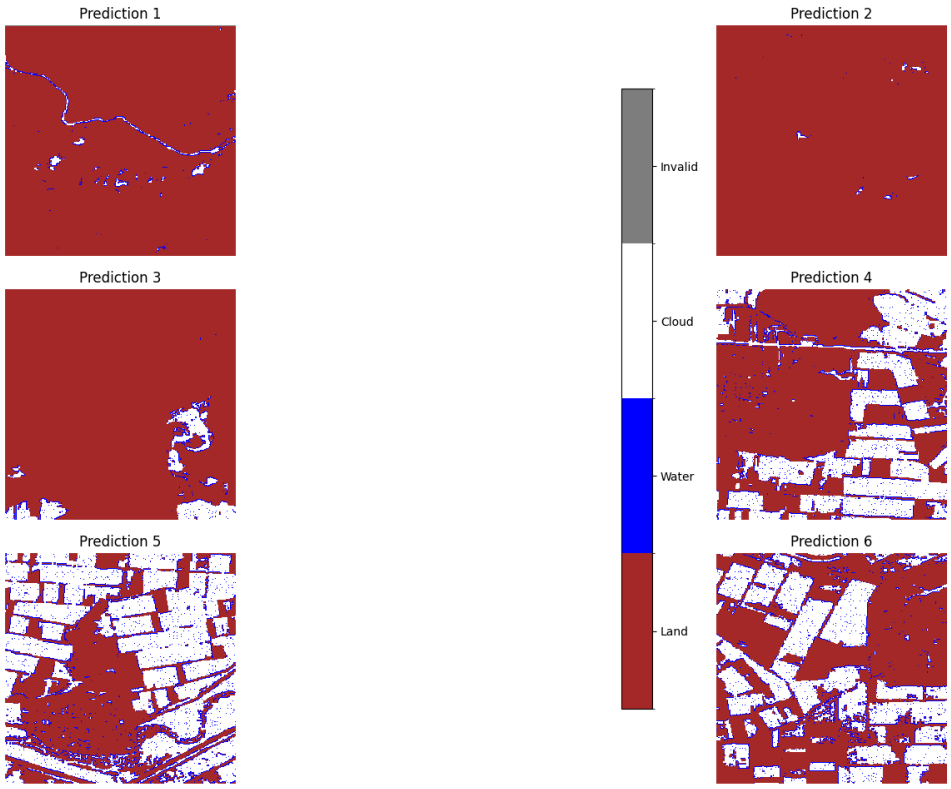



Figure 4.6: an example of inference images from the inference stage of the model


Figure 4.7: The six available inference images, retrieved from the validation set of Worldfloods_v1_0_sample.

# 5. Findings

This section will go over the results of this project and what they might imply for the project. The section is split into two sections, statistical analysis and visual analysis. Statistical analysis will go into the evaluation metrics throughout the model, reporting the findings with IoU, recall and loss metrics throughout each stage of the model, including training, testing and inference. Visual analysis will go into the inference results, looking into the successes and failures of the segmentation process. The model created in this project learns more the more times it is completely run, as the top 3 models with the best water IoU are kept for checkpointing, the more the model is run the more it improves. The following results represent the best runs with the best water IoU scores, whilst there might be a model with a slightly lower loss than these, these are the results that were recorded. This is because water IoU is one of the key evaluative metrics for this model along with recall, but IoU is the set checkpointing metric for this model.

## 5.1 Statistical Analysis

Table 5.1 displays a list of the metrics used in the model and where they are used in the model. This highlights the importance of each metric in each stage of the model. Within each metric's section, a breakdown as to why each metric is utilised in a phase of the model will be described, as well as a description of what the metric does and the findings throughout the model. Both Unet and Segnet models shall be reported in each metric's section.

| **Metric** | **Training** | **Testing** | **Inference** |
|---|---|---|---|
| Water IoU | True | True | True |
| Land IoU | True | True | True |
| Cloud IoU | True | True | True |
| Water Recall | True | True | True |
| Land Recall | True | True | True |
| Cloud Recall | True | True | True |
| Prototype loss | True | True | False |
| BCE loss | True | True | False |

Table 5.1: A table displaying utilised metrics and where they are used in the model

### 5.1.1 Intersection over Union

#### Breakdown of metric

IoU, also known as the Jaccard Index [29], measures the overlap between predicted segmentation and ground truth segmentation. Calculated as the ratio of area of intersection to the area of union, where predicted segmentation represents the intersection (the amount of pixels predicted by the model) and the ground truth represents the union (the total correct pixels). The calculation can be seen in figure 5.1.
IoU is a standard metric for evaluating segmentation models, providing insight into the effectiveness of predictions and how well the predicted segments match the actual segments. Higher IoU values

indicate stronger model performance. IoU is performed on each class, which in this model are land, water and cloud. This ensures that each class is tracked and evaluated, indicating which class is being properly segmented. This is calculated at all stages of this project's models, training, testing and inference.

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

Figure 5.1: calculation for intersection over union used in this project

### U-net

The IoU metric showed steady growth in an almost linear pattern when using U-net with IDSS during training, whilst training was only 45 epochs, IoU did start to show signs of plateau towards the end, water most of all. Figure 5.2 displays the values for IoU per class during the training stage. IoU is reported at the end of each epoch, this is one of the top batches during that training step rather than the default final batch value. This was done so that the best performance of the epoch was represented, which isn't necessarily its final batch.
Land being the dominant class had the highest IoU, of a final score of 0.721 or 72.1% IoU score, showing that the model was good at correctly identifying land throughout the training stage of the model. Water had an IoU of 0.501 or 50.1% by the end of training. Finally Cloud had a final IoU score 0.616 or 61.6%.
These results show the model was fairly good at predicting the land and cloud classes, whilst water being not as easy for it to predict correctly. Being the least represented class in worldfloods dataset, but the highest weighted, it shows that whilst still having a high weighting classification can still be lower than the more dominant classes.
Testing and inference IoU scores are reported at the end of their respective processes. The final IoU scores for the testing step were as follows; land 0.589 or 58.9%, water 0.452 or 45.2% and cloud 0.499 or 49.9%. Inference scores were slightly lower: land 0.526 or 52.6%, water 0.414 or 41.1% and cloud 0.426 or 42.6%. This shows that the evaluation process of the model is slightly weaker in this project's implementation of IDSS. Inference is likely a bit weaker due to it being performed on an unseen dataset.

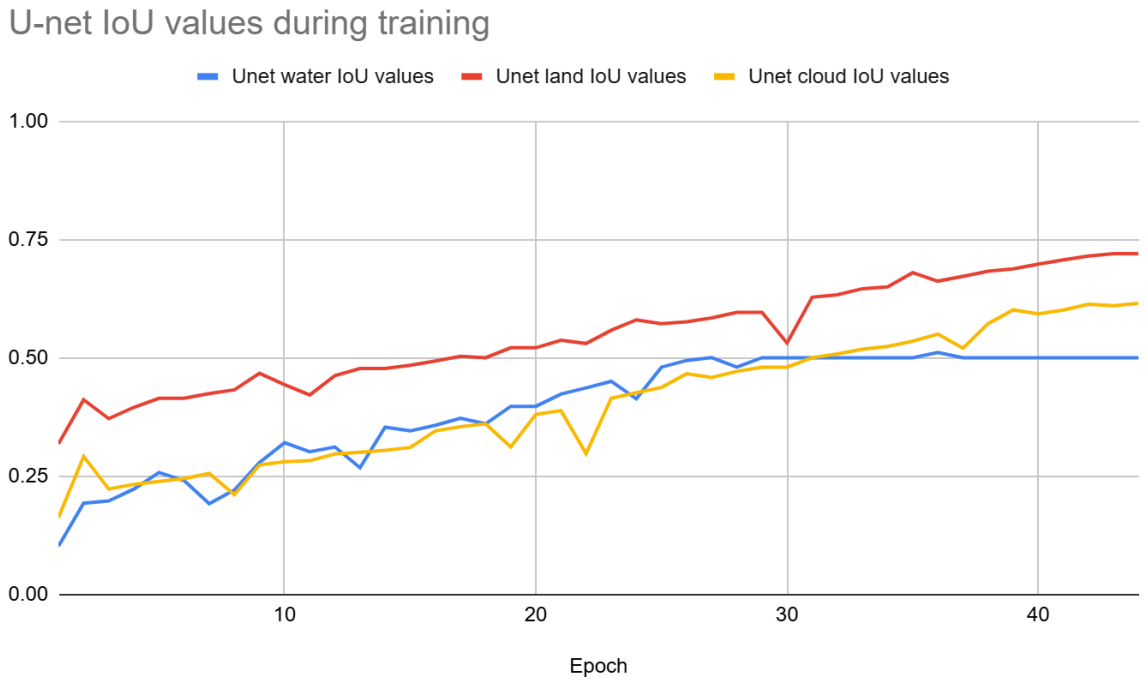



Figure 5.2: A line graph displaying progression of IoU scores over 45 training epochs for U-net IDSS model

## Segnet

Training IoU using SegNet showed similar growth to U-net, a steady growth with occasional fluctuations. There was less stagnation during the end of the epochs for water compared to U-net. The final Iou scores for training were; 0.463 or 46.3% for water, 0.747 or 74.7% for land and 0.509 or 50.9% for cloud. Compared to U-net there is a slightly lower water IoU, showing more of a struggle to classify overall water against the predictions. However the model showed more growth compared to U-net so if the epochs were increased it might've reached a higher IoU score than U-net, but in order for a fair test to occur, epochs are restricted to a maximum of 45. The training values can be seen in Figure 5.3.

Testing showed lower IoU scores, similar to U-net, further suggesting that validation in this model is slightly weaker. The IoU scores were as follows; water was 0.368 or 38.6%, land was 0.529 or 52.9% and cloud was 0.372 or 37.2%.

Inference showed significantly lower water scores, however land was exceptionally identified. The IoU scores were as follows; water was 0.271 or 27.1%, land was 0.785 or 78.5% and cloud was 0.341 or 34.1%. Further suggesting that the validation layers or the decision making layers were struggling with classification predictions against the ground truth values.

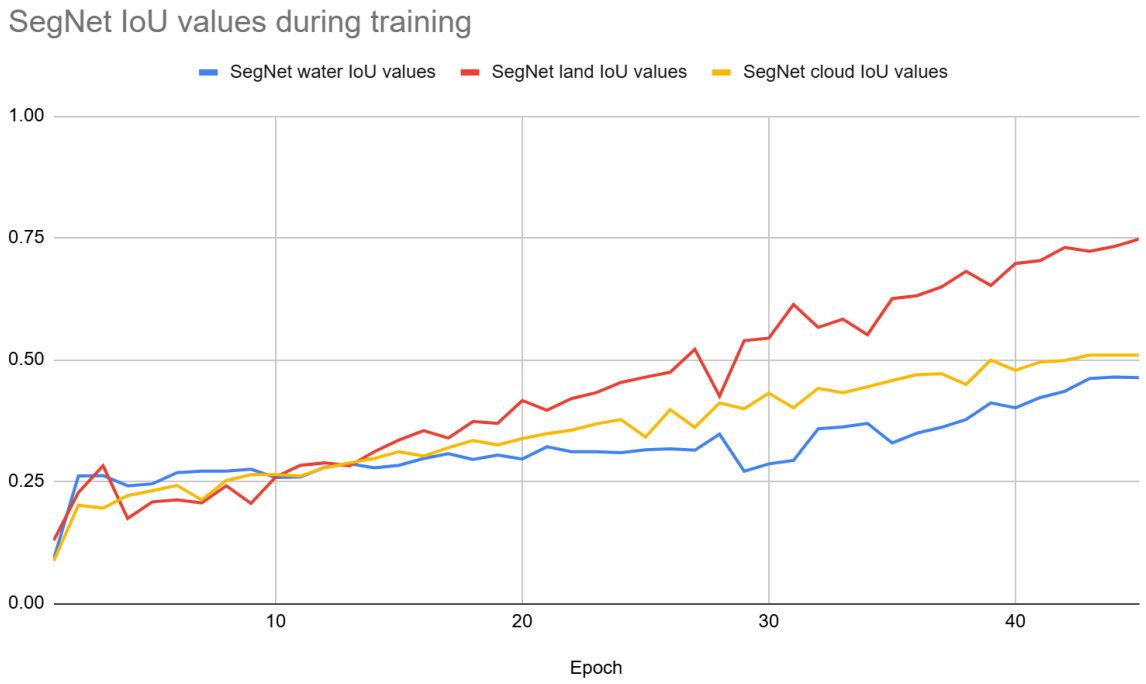



Figure 5.3: A line graph displaying progression of IoU scores over 45 training epochs for SegNet IDSS model

## 5.1.2 Recall

### Breakdown of metric

Recall, also known as sensitivity or true positive rate [30], measures the proportion of true positive pixel predictions correctly identified by the model. The calculation can be seen in figure 5.2.
High recall indicates that the model is successfully identifying most of the positive instances which is crucial in applications where missing positive cases is costly. Akin to IoU this is also measured per class, land, water and cloud, giving detailed information on which classes are being properly identified.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Figure 5.4: Calculation for Recall used in this project

### U-net

Training recall can be seen in figure 5.5, and shows good learning during training, with a fluctuating but steady positive growth rate. This suggests that U-net IDSS is identifying pixels correctly in its predictions during training. The final training epoch recall values were as follows; for water 0.796, for land 0.821 and for cloud 0.761.

However during testing recall was lower, similar to IoUs training to testing pattern in values. The testing values were as follows: 0.489 for water, 0.676 for land and 0.437 for cloud. Further suggesting that the validation section wasn't quite right, or at least didn't have enough training to perform better predictions with the full IDSS architecture including the decision making layer.
Inference also showed similar values for recall, and were as follows; 0.502 for water, 0.689 and 0.520 for cloud, which shows a slight increase in each class recall on blind datasets.

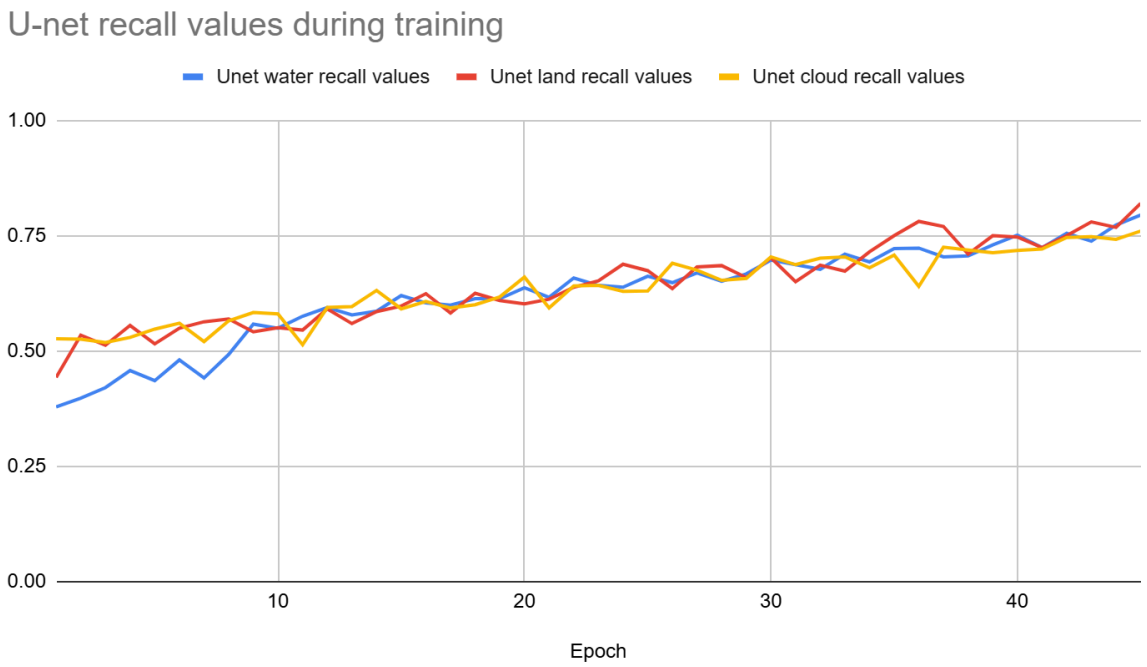



Figure 5.5: A line graph displaying progression of recall values over 45 training epochs for the IDSS model using U-net backbone

## SegNet

Training recall for SegNet displayed a larger range of positive growth than U-net however it did display larger amounts of fluctuation. These patterns can be seen in figure 5.6. The final epoch recall values were as follows; 0.749 for water, 0.803 for land and 0.772 for cloud.
Testing also showed similar results as U-net, which were as follows: 0.496 for water, 0.736 for land and 0.467 for cloud, showing higher scores than U-net but not by a significant amount apart from land recall.
Inference also showed similar values and were as follows; 0.483 for water, 0.809 and 0.502 for cloud. Again having land significantly higher than the other values, suggesting that the model was exceptional at identifying areas of land. This is likely due to its dominance in this dataset, and therefore higher amount of features to learn.

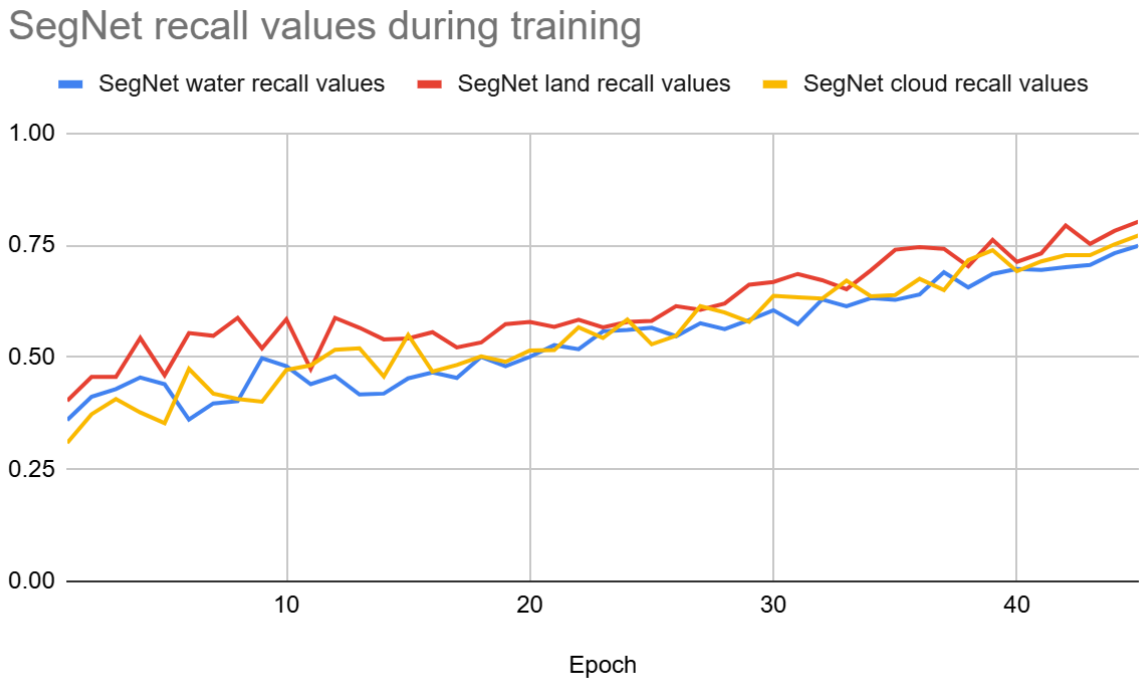



Figure 5.6: A line graph displaying progression of recall values over 45 training epochs for the IDSS model using SegNet backbone

### 5.1.3 Prototype loss

#### Breakdown of metric

Prototype loss aims to enhance the model's discriminative power by minimising the distance between latent features and their corresponding class prototypes. This encourages the model to produce latent representations that are closer to their respective class prototypes. The calculation for prototype loss can be seen in figure 5.3. This loss reaches a maximum of 600.00 prototypes, the total prototypes for this model

$$L_{proto} = \sum_{C=0}^{C-1} \frac{\sum_{b=0}^{B-1} \sum_{i=0}^{HW-1} 1(y_{b,i} = c) \cdot d_{b,i,c}}{\sum_{b=0}^{B-1} \sum_{i=0}^{HW-1} 1(y_{b,i} = c) + \epsilon}$$

Figure 5.7: The calculation used for prototype loss in this project, where C = number of classes, B = batch size, HW = number of pixels in image, $1(y_{b,i} = c)$ = an binary mask (1 if pixel i in batch b belongs to class c, otherwise 0), $d_{b,i,c}$ = the euclidean distance between feature representation of pixel i and its class prototype, $\epsilon = 1e^{-6}$ = a near zero value to prevent division by zero

### U-net

Prototype loss started at 571.12 in the early epochs of training and ended up decreasing as it progressed to values such as 26.22. There were instances where the loss would reach an upper limit of 600.00, however this isn't as high as it can get, there were training instances where loss reached 646.510, suggesting that prototype feature distances became out of bounds. This would often lead to 0.00 in IoU scores and 1.000 in recall scores. Testing showed a value of 389.33 which if divided by the total epochs suggests a loss per prototype of 0.6488833333 with a trained U-net based model

### Segnet

SegNet showed similar results with loss starting at 512.60 during the early stages of training and finishing 19.191. Akin to U-net there were also instances of upper limit prototype loss suggesting that this event is not CNN specific and can happen regardless of architecture and is rather part of the implemented IDSS framework. Testing showed similar values to U-net however they were slightly lower than U-nets showing improvement for distancing in the prototype layer. Prototype loss was 362.22 or 0.6037 per epoch.

## 5.1.4 BCE loss

### Breakdown of metric

BCE loss measures the dissimilarity between the predicted probabilities and the actual binary labels. It's commonly used in the binary classification tasks to penalise incorrect predictions. The calculation for this metric can be seen in figure 5.4. PyTorch provides a cross_entropy function which is utilised to calculate BCE in the compute_losses function in this project's IDSS implementation worldfloods_model.

$$L_{BCE} = \sum_{c=1}^{C} w_c \sum_{b=1}^{B} \sum_{i=1}^{HW} 1(y_{b,i}=c) \cdot \log p_{b,i,c}$$

Figure 5.8: the calculation for BCE loss, where C = number of classes, B = batch size, HW = number of pixels in image, $1(y_{b,i} = c)$ = an indicator function (1 if pixel i in batch b belongs to class c, otherwise 0) and $w_c$ = class weight taken from class weights in the ML4Floods configurations json

### U-net

BCE loss would start incredibly high, at xxx, suggesting a lot of incorrect classifications at the start of training, this is not necessarily a bad thing and is likely influenced by class weights. As training progressed and came to an end loss improved to about 6.71. Whilst this is a significant improvement and showed U-net learned to classify well, optimal BCE values are near zero values such as 0.15. Testing produced a BCE value of 39.01 suggesting that validation on this model was better than early training but still needs to be improved. This may come with more training.
The source of this issue was not found due to time constraints and a gap in knowledge. Furthermore while BCE loss is an important general metric for a model it is not as important as improving IoU or Recall so was prioritised less in this model.

### Segnet

BCE loss would start incredibly high for SegNet, at 295.985, further suggesting that the model struggled with classification in the early stages of training, and decreased to 22.09. This is a significant improvement from the beginning of training, less so than the U-net model, but regardless an improvement. Testing produced a loss result of 33.81, this is an improvement compared to U-net but is still higher than typical model standards.

## 5.2 Visual Analysis

Visual results are entirely centralised to the inference stage of the model. This consists of the displaying of the trained model's predictions against a blind set of 6 images taken from the worldfloods_v1_0_sample validation dataset. In addition to this a heat map confusion matrix is displayed to present a visual representation of correctly predicted pixels and misclassified pixels within the model's inference. The images displayed are not explicitly from the same model iteration as the previous metrics in section 5.1, but are a good representation of the model's visual results.

### 5.2.1 Confusion matrix

The confusion matrix presented in Figure 5.10. Diagonal values starting from top left and going to bottom right represent the IoU of pixels correctly predicted, 78.2% for land, 20.8% for water and 56.2% for cloud. The remaining cells represent the classes misclassified as the corresponding class, for example the middle top cell represents an IoU of 8.5% of land pixels misclassified as water, top right cell represents an IoU of 13.3% of land pixels misclassified as cloud. U-net did exceptionally well when it came to identifying land, was very good at identifying clouds, however water was significantly less identified during inference. This could be due to a multitude of factors, which are discussed in section 5.3.
SegNet showed similar results which can be seen in Figure 5.11. IoU pixels correctly identified were as follows; 86% for land, 15.7% for water and 29.1% for cloud, showing an increase in land classification, but a slight decrease in water classification for water compared to U-net. However, there are a fair few less misclassification cells with high IoU scores across the board. This indicates SegNet was fairly good at not misclassifying predictions during inference.

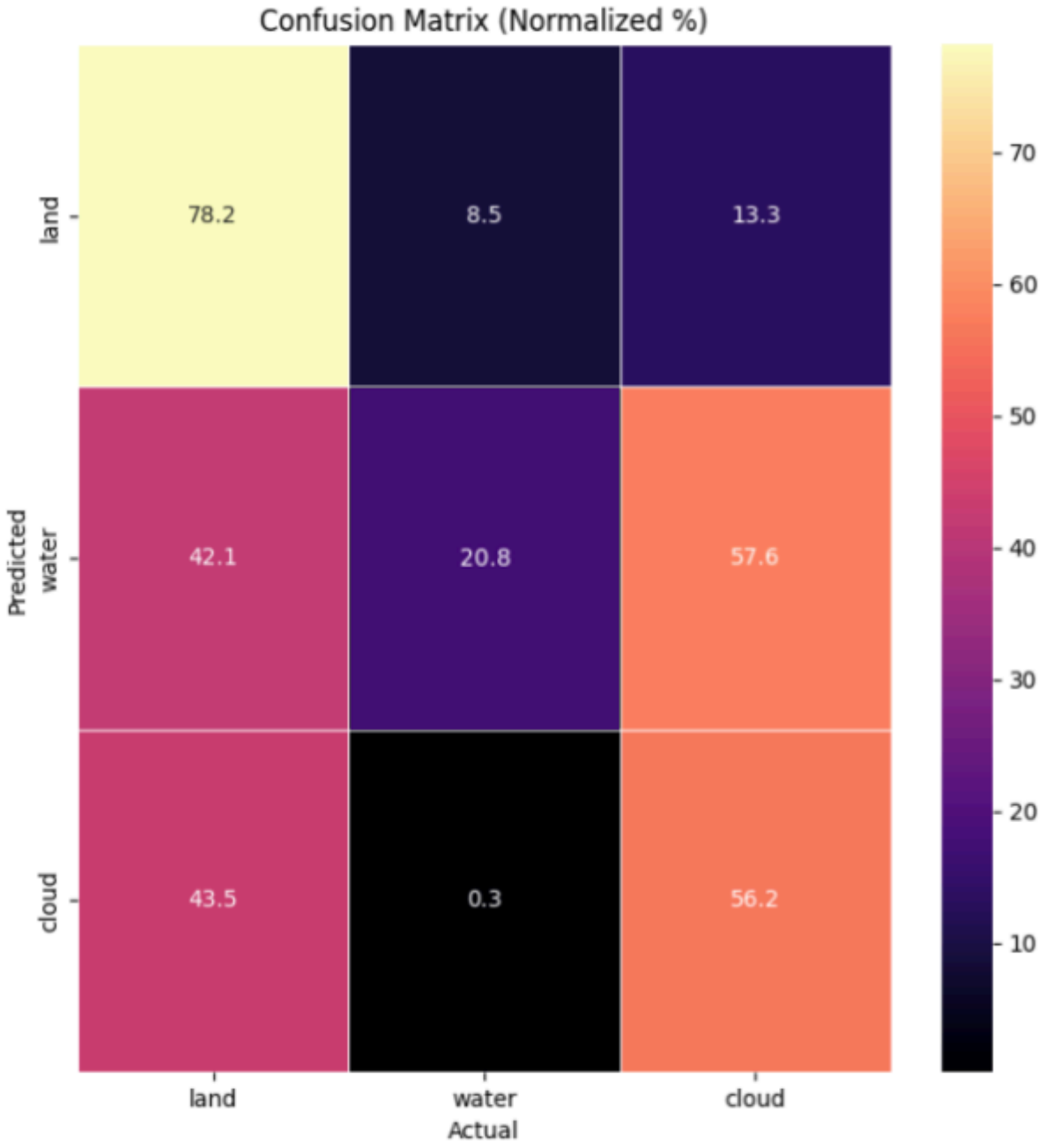



Figure 5.10: U-net heat map confusion matrix

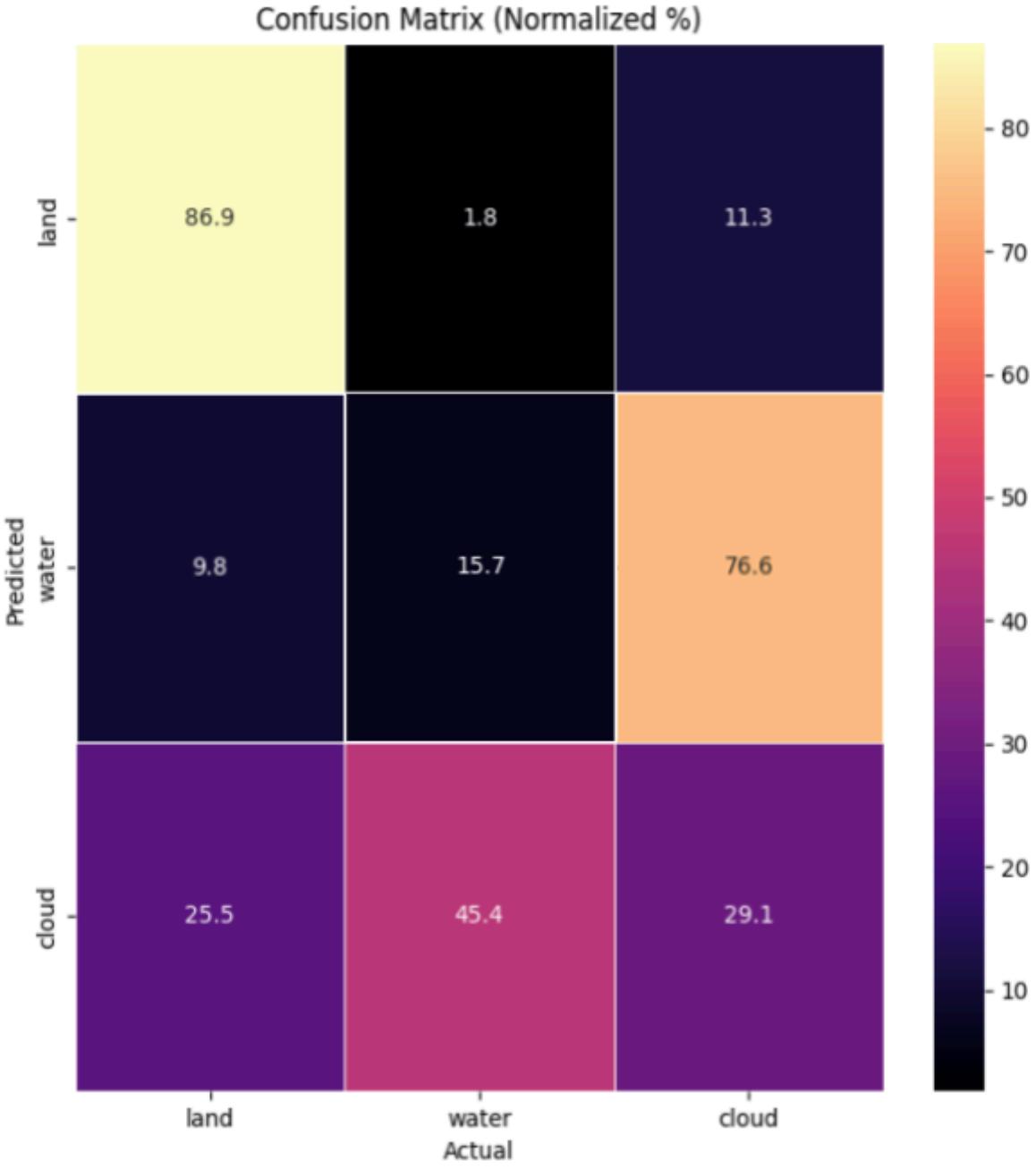



Figure 5.11 SegNet heat map confusion matrix

### 5.2.2 Inference Predictions

These represent the ultimate result of the model. A map of predictions of the segmented image. Figure 5.12 displays the set of original images used for this segmentation. Throughout the model these images haven't been used for training or testing, thus presenting a blind test for the inference process, ensuring unbiased evaluation of the model.
The U-net inference can be seen in figures 5.13 and 5.14. Figure 5.14 presents an early training run, demonstrating how the model improves with the increase in the number of runs performed. 5.13 displays a fairly good level of classification for each class, however there are still some issues, a lot of land and water areas seem to be misclassified as clouds (represented by the frequent patches of white.
The SegNet inferences can be seen in figures 5.15. The inference shows similarity in classification and misclassification to U-net. This shows significant favorability towards land and cloud, with water remaining the weaker classification, likely due to the lack of representation within the database. Whilst the model seemed to identify areas that differed to land such as water regions, it seemed to struggle with consistently classifying them within that region.


Figure 5.12: Validation set images used for inference displayed in RGB bands

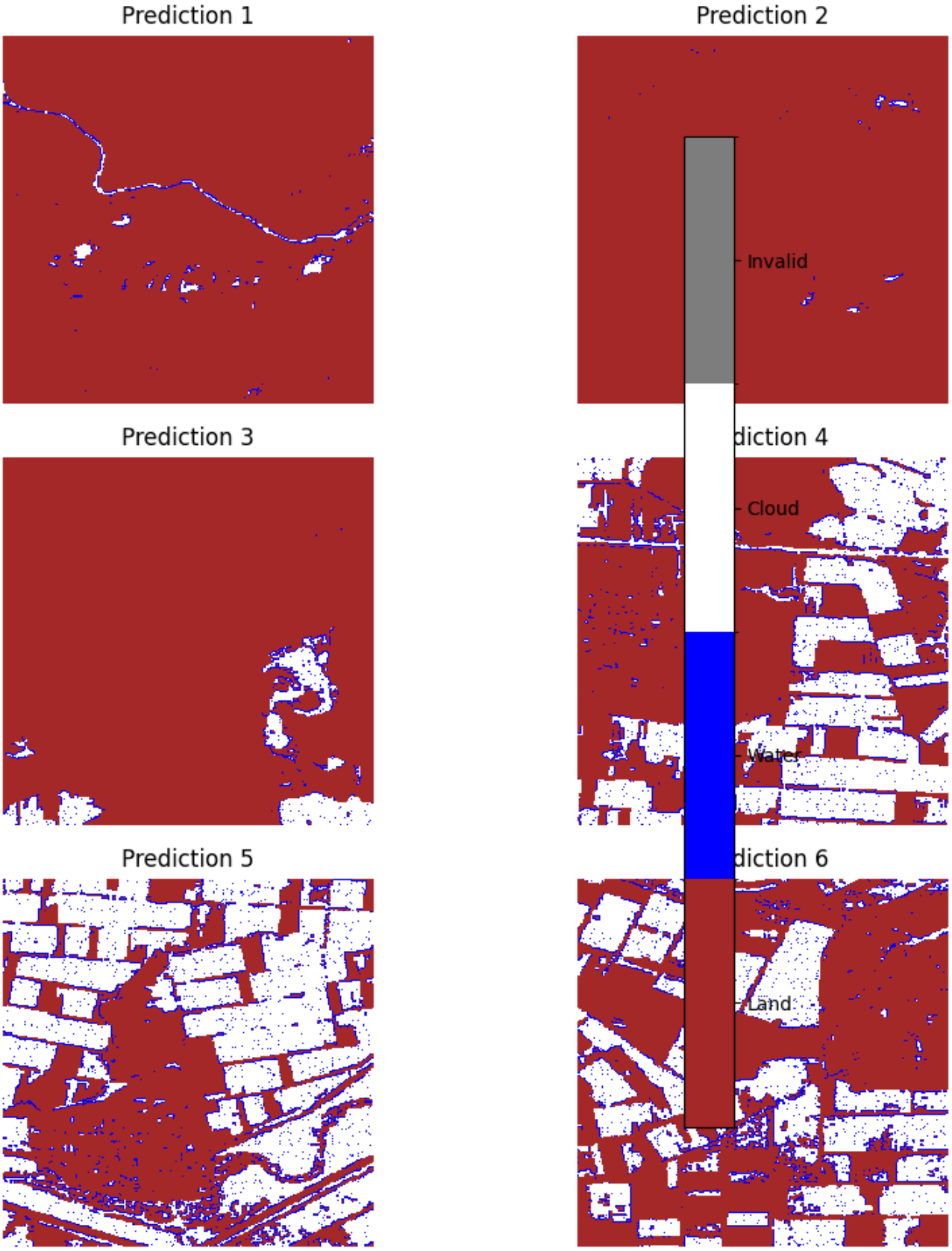



Figure 5.13: U-net inference with colour bar representing class to colour mapping

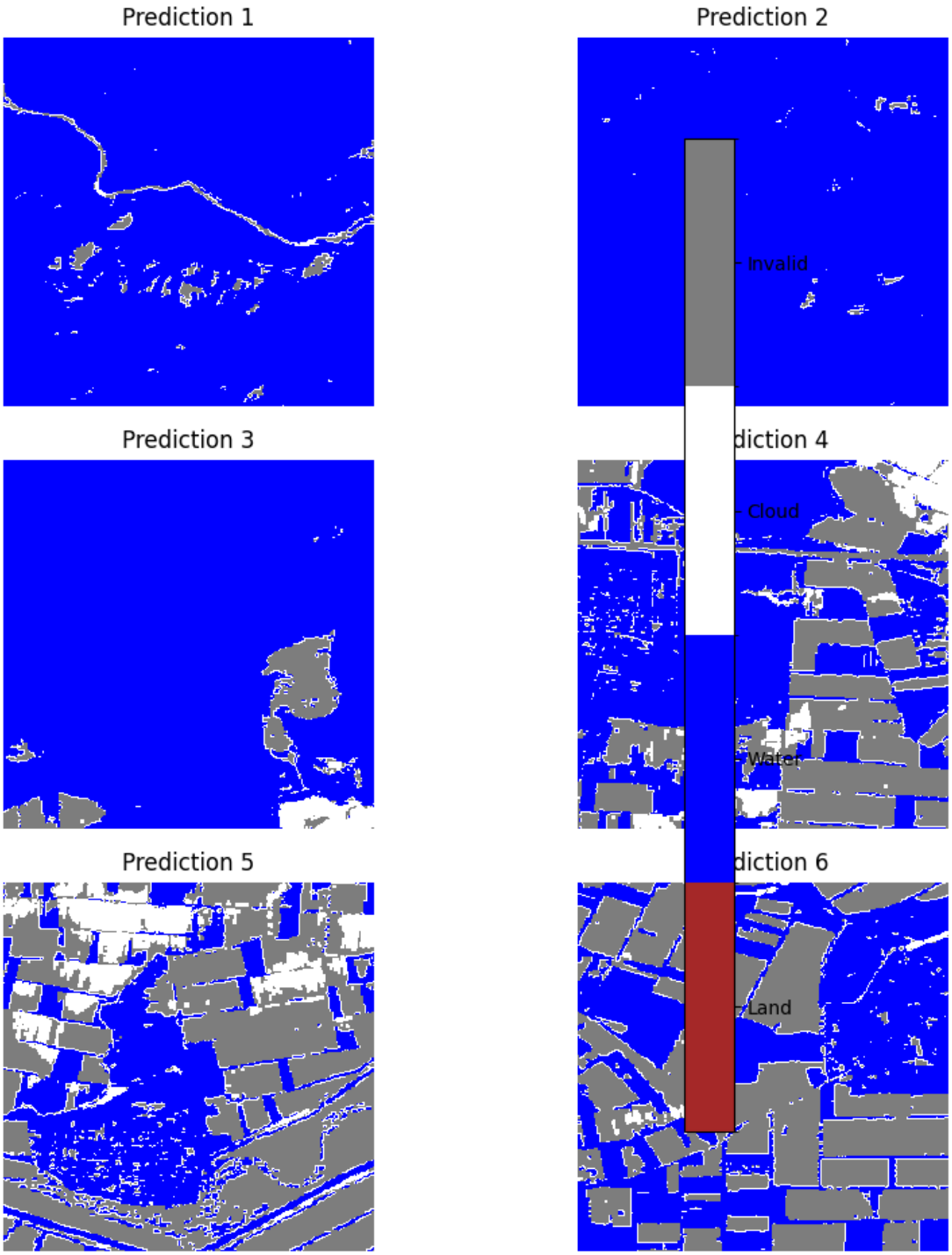



Figure 5.14: a demonstration of early model training inference on U-net, all land pixels have been classified as water or invalid

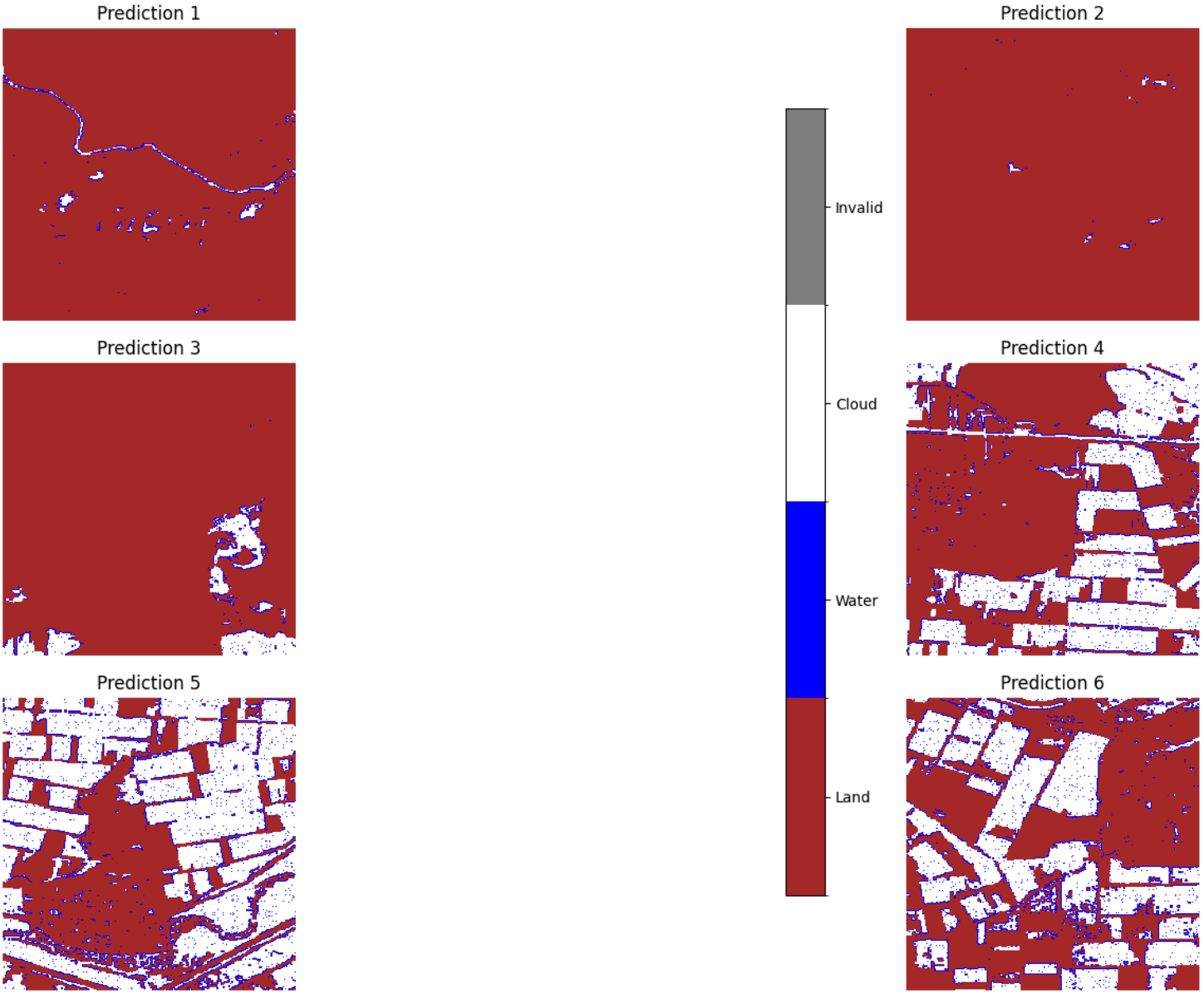



Figure 5.15: SegNet inference

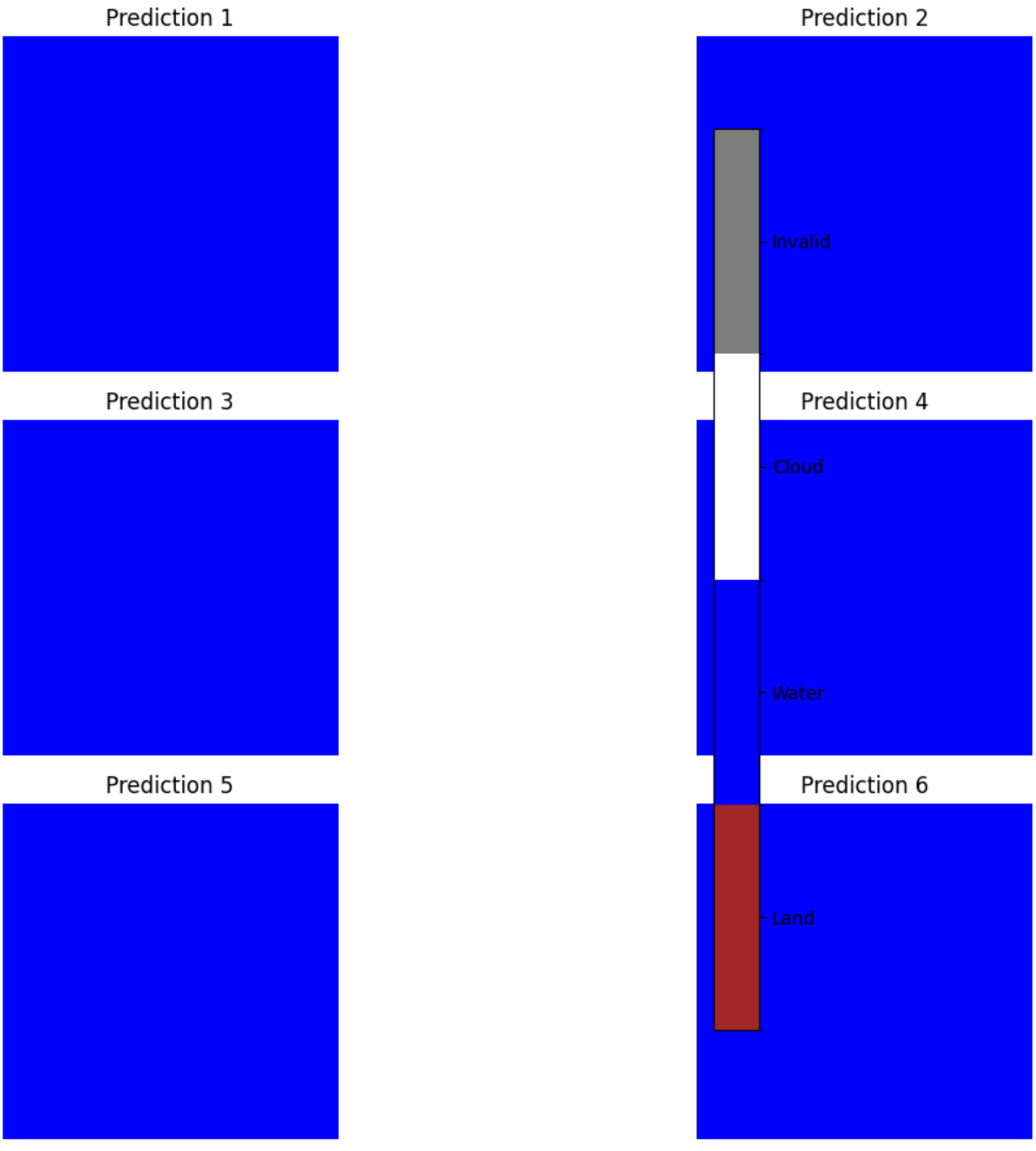



Figure 5.16: U-net model inference when loss became too high (happened on both CNN)

## 5.3 Discussion

Within this section statistical and visual analysis shall be concluded and discussed, inferring meaning from these results. Firstly statistical analysis. IoU and Recall scores weren't as high as the original IDSS paper, however they showed a consistent positive growth rate during training. There were some signs of stagnation in final epochs, but not much that suggests a plateau in growth. This was consistent in all IoUs across both CNN models but the U-net run with IoU water, as a plateau began at epoch 26 and continued till termination. Prototype and BCE loss was high throughout training, but improved the more epochs were performed. There were instances where loss would skyrocket and this caused IoU and Recall scores to plummet significantly, this would impact inference as predictions would present an image of a single class prediction (a completely land square). An example of this can be seen in figure 5.16.

Visual analysis of segmentation results were generally positive. The confusion matrix was an excellent form of display and granted insights into the performance of the model; what it classified well, what it didn't, which class was misclassified more frequently and which class was least misclassified. Cloud and water tended to misclassify more frequently, which suggests that the implemented model's validation ability had a flaw in classification. This could be within the interpretable rules section of the decision making layer, due to its rule masking strategy instead of iterating over the pixels. Inference showed similarity across both U-net and SegNet, having similar issues, with cloud misclassification, hindering water classification, and producing a large amount of noise throughout the image. However U-net did seem to produce a higher level of this noise. The model would also typically mistake man made structures such as roads or buildings, believing them to be clouds, this could be due to a high SWIR reflectivity from these structures.

Training time was also documented. During the first run of U-net each training epoch ran for ~7 minutes, but after a few runs time increased to ~17 minutes. There isn't much indication as to why, it could be due to the use of previous model checkpoints in the later runs to improve training. SegNet did not reflect the same pattern, epochs ran for 15 minutes consistently, which is a shorter time than U-net. This does add up in longer runs as it could be greatly faster than U-net when time is aggregated. This could prove useful in real world applications as it does prove the theory that SegNet is faster than U-net. This is likely due to SegNet's max pooling indices, which are a lighter weight solution that the U-net skip connections, therefore being a bit more efficient in processing.

In conclusion this section provides a comprehensive discussion of both statistical and visual analysis conducted on both the U-net and SegNet model implemented in this project. Statistical analysis for both of these models showed a consistent growth during training, however validation showed signs of struggle when using training and the decision making layer. This suggests a flaw in its implementation or the IDSS model struggles with a lesser model setup such as the 600 prototypes used and the weaker machine. Visual analysis offered high level information as opposed to the strictly measurable insights into model accuracy that statistical analysis provided. This level of analysis helped identify specific class strengths and weaknesses in classification ability, as well as patterns in segmentation quality such as the noise observation. Together these approaches in analysis help infer broader implications for CNN-based flood detection and IDSS model adaptability.

# 6. Conclusion

## 6.1 Review of Project Aims

This project aimed to implement a method of flooding semantic segmentation and improve it by implementing a new CNN backbone. This was completed to full extent, Zhang (2022) IDSS architecture was implemented and a SegNet CNN backbone was implemented into the architecture. However, the model struggled to identify water, a crucial aim for the study, however this is likely due to a multitude of reasons.

## 6.2 Future work

This implementation of IDSS utilises a significantly less powerful machine and in order to compensate, less prototypes, 600 opposed to the 1500. This ultimately hindered the performance of the model by reducing the models ability to cluster dataset features to help classification and produce interpretable rules. The weaker machine also greatly reduced the time for training, as opposed to the High End Computing Cluster used in the original study, which could likely perform far more epochs for training over a longer time with the complete WORLDFLOODS dataset. Whilst the original study didn't explicitly express how many training epochs were utilised, it is likely far greater than the 45 epochs used for training cycles.
In order to gain a better understanding of how the model performs with SegNet CNN a longer training cycle with the complete WORLDFLOODS dataset needs to be completed, providing a far more enriched training. This could also provide good training time recordings, as on the lesser dataset and machine training time for SegNet was faster than U-net, therefore would likely be faster on a larger dataset.
Furthermore other CNNs can be implemented due to the high adaptability of the ML4Floods pipeline, allowing easy access to altering CNN backbones.

## 6.4 Lessons learned

This project adapts past deep learning works, showing that they can be adapted for further insights into their architecture. The adjustment of the CNN backbone in this study opens an avenue for exploring the adaptability of segmentation models in flood detection.
A key takeaway from this study was the complexity of integrating prototype based learning within a flood segmentation model. This method is exceptionally computationally expensive and requires powerful computing ability from the machine training it. Anything less is unable to implement the full extent of the prototypes and will affect the results of the inference.

## 6.5 final conclusion

This project has successfully shown that the CNN backbone for the IDSS model can be adapted with relative ease, improving on existing segmentation models for flood identification. Whilst the model implemented in this project didn't perform to an acceptable standard, due to a lack of computational power, it has opened the door to potential further works on different CNN backbones. Further highlighting the benefits and drawbacks of using SegNet over U-net for prototype based segmentation flood detection, such as its shorter training time and higher rate of growth in learning classification.

# Appendix A: References


1. Floods (World Meteorological Organization (WMO) 2025)
   https://wmo.int/topics/floods#:~:text=Floods%20cause%20more%20than%20%2440,each%20year%20around%20the%20world
   (Accessed 21/01/2025)
2. What you should know about Sentinel-2 climate satellites (Airbus 2025):
   https://www.airbus.com/en/newsroom/stories/2024-08-what-you-should-know-about-sentinel-2-climate-satellites#:~:text=The%20Sentinel-2%20satellites%20capture,swath%20width)%20of%20290%20km.
   (Accessed 21/01/2025)
3. Past and future sea level rise (Met Office 2025)
   https://www.metoffice.gov.uk/weather/climate-change/organisations-and-reports/past-and-future-sea-level-rise#:~:text=Past%20sea%20level%20rise&text=It%20is%20also%20clear%20that,over%20the%20period%202006-2018
   (Accessed 21/01/2025)
4. Mirza, M.M.Q., (2010). Climate change, flooding in South Asia and implications. *Regional Environmental Change*, 11(Suppl 1), pp.S95–S107.
   https://doi.org/10.1007/s10113-010-0184-7.
   (Accessed 21/01/2025)
5. Introducing Prithvi WxC, a new general-purpose AI model for weather and climate (IBM 2024)
   https://research.ibm.com/blog/foundation-model-weather-climate
   (Accessed 25/01/2025)
6. Smith, J., Brown, L. and Taylor, M. (2024) 'Title of the article', *Journal of Environmental Management*, 320(1), pp. 45-60.
   https://www.sciencedirect.com/science/article/pii/S2352938524002313
   (Accessed: 3/2/2025)
7. Climate crisis fuels flooding and deepens displacement (UNHCR 2025)
   https://www.unhcr.org/news/stories/climate-crisis-fuels-flooding-and-deepens-displacement#:~:text=Climate%20change%20exacerbates%20the%20risk,%2C%20onward%2C%20and%20protracted%20displacement.
   (Accessed: 3/02/2025)
8. Durban coastline: sewage polluted beaches pose threat to holiday makers and the environment (The Conversation 2022)
   https://theconversation.com/durban-coastline-sewage-polluted-beaches-pose-threat-to-holiday-makers-and-the-environment-196244#:~:text=After%20the%20floods%2C%2080%25%20of,close%20to%20wastewater%20treatment%20works.
   (Accessed: 3/02/2025)
9. *Tackling flooding in Bangladesh in a changing climate*. (LSE 2023)
   https://www.lse.ac.uk/granthaminstitute/wp-content/uploads/2023/08/Tackling-flooding-in-Bangladesh-in-a-changing-climate.pdf
   (Accessed: 3/2/2025)
10. NASA's AI Use Cases: Advancing Space Exploration with Responsibility (NASA 2025)
    https://www.nasa.gov/general/2024-ai-use-cases/
    (Accessed: 4/02/2025)
11. Bangladesh to use SERVIR satellite-based flood forecasting, warning system (NASA 2015)
    https://science.nasa.gov/earth/water-on-earth/bangladesh-to-use-servir-satellite-based-flood-forecasting-warning-system/
    (Accessed: 4/02/2025)
12. *Convolutional Neural Networks: How they work and their applications*. (IBM 2025).
    Available at: https://www.ibm.com/think/topics/convolutional-neural-networks
    (Accessed: 03/02/2025).
13. Rosebrock, A. (2021). *U-Net: Training image segmentation models in PyTorch*.
    PyImageSearch.
    Available at:
    https://pyimagesearch.com/2021/11/08/u-net-training-image-segmentation-models-in-pytorch/
    (Accessed: 03/02/2025).

14. Badrinarayanan, V., Kendall, A., and Cipolla, R. (2016). *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. arXiv.
Available at: https://arxiv.org/pdf/1511.00561
(Accessed: 03/02/2025).
15. McGinnis, D.F. and Rango, A., (1975). Earth resources satellite systems for flood monitoring. "Geophysical Research Letters," 2(4), pp.132-134.
16. Zhang., Q, et al. (2021). "Unsupervised grnn flood mapping approach combined with uncertainty analysis using bi-temporal sentinel2 msi imageries," International Journal of Digital Earth, vol. 14, no. 11, pp. 1561–1581,
17. Xu, H. (2006). *Modification of Normalized Difference Water Index (NDWI) to enhance open water features in remotely sensed imagery*. International Journal of Remote Sensing, 27(14), pp. 3025–3033.
Available at:
https://www-tandfonline-com.ezproxy.lancs.ac.uk/doi/full/10.1080/01431160600589179#d1e203
(Accessed: 03/02/2025).
18. Poppi, S. *et al.* (2021) 'Revisiting The Evaluation of Class Activation Mapping for Explainability: A Novel Metric and Experimental Analysis', in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, pp. 2299–2304.
Available at: https://doi.org/10.1109/CVPRW53098.2021.00260
(Accessed: 03/02/2025).
19. Mateo-Garcia, G., et al. (2021) Towards global flood mapping onboard low cost satellites with machine learning. Scientific Reports 11, 7249. DOI: 10.1038/s41598-021-86650-z
20. Portalés-Julià. E., et al. (2023). Global flood extent segmentation in optical satellite images. Scientific Reports 13, 20316 DOI: 10.1038/s41598-023-47595-7
(Accessed: 03/02/2025).
21. Mateo-Garcia, G., et al.,(2023) In-orbit demonstration of a re-trainable machine learning payload for processing optical imagery, Scientific Reports 13, 10391 . DOI: 10.1038/s41598-023-34436-w.
(Accessed: 03/02/2025).
22. McFeeters, S.K. (1996). The use of the Normalized Difference Water Index (NDWI) in the delineation of open water features. Remote Sensing of Environment, 57(3), pp.207–217.
Available at: https://www.sciencedirect.com/science/article/pii/S0034425796000673
(Accessed: 03/02/2025).
23. Shaw, J. (2021). Pixel-wise Machine Learning and Deep Learning Methods for Semantic Segmentation. Northern Illinois University.
Available at:
https://huskiecommons.lib.niu.edu/cgi/viewcontent.cgi?article=2467&context=studentengagement-honorscapstones
(Accessed: 03/02/2025).
24. *FDL Europe: AI for Earth Observation Research & Innovation*. ESA & FDL (2019).
Available at: https://eslab.ai/fdl-europe-2019
(Accessed: 03/02/2025).
25. Portalés-Julià, E. *et al.* (2023) 'Global flood extent segmentation in optical satellite images', *Scientific reports*, 13(1), pp. 20316–20316. Available at:
https://doi.org/10.1038/s41598-023-47595-7.
26. Zhang, Z., Angelov, P., Soares, E., Longepe, N., and Mathieu, P.P. (2022). *An Interpretable Deep Semantic Segmentation Method for Earth Observation*. IEEE International Conference on Intelligent Systems.
Available at: https://doi.org/10.1109/IS57118.2022.10019621
(Accessed: 03/02/2025).
27. vinceecws (2021). *SegNet_PyTorch: A PyTorch implementation of SegNet for image segmentation*.
Available at: https://github.com/vinceecws/SegNet_PyTorch
(Accessed: 13/01/2025).
28. Portalés-Julià, E., Mateo-García, G., Purcell, C., & Gómez-Chova, L. (2023). Global flood extent segmentation in optical satellite images. *Scientific Reports*, 13(1), 20316.
Available at: https://doi.org/10.1038/s41598-023-47595-7
(Accessed: 19/03/2025).

29. Rahman, M.A., Wang, Y. (2016). Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation. In: Bebis, G., et al. Advances in Visual Computing. ISVC 2016. Lecture Notes in Computer Science(), vol 10072. Springer, Cham.
Available at: https://doi.org/10.1007/978-3-319-50835-1_22
(Accessed: 19/03/2025).
30. Powers, D. M. (2020). "Evaluation: From Precision, Recall and F-measure to ROC, Informedness, Markedness & Correlation."
Available at: DOI: 10.48550/arXiv.2010.16061
(Accessed: 19/03/2025).

# Appendix B: Proposal

## A Study on Flood Detection Using Satellite Imaging and Deep Semantic Segmentation


### Abstract

This proposed project will study flood detection using satellite images as well as implementing a flood detection method using semantic segmentation. Additionally the project shall adapt the implementation by altering its main driving model and the dataset it uses for training and testing. This is important as it tests whether this vital program can be adaptable and accurate across datasets.

The first stage of this project will involve implementing a deep semantic segmentation program practised on earth observation, specifically satellite imaging. This program will aim to produce results consistent with similar studies. It will then be altered so that the CNN driving the semantic segmentation is changed completely as well as altering the satellite image data set. This project aims to produce correctly labelled features that accurately identify areas prone to flooding based on the input images.


### 1.Introduction

As our planet changes through our activity, climate change brings an exacerbated set of problems, natural disasters. Disasters, including floods, earthquakes, and wildfires, have increased drastically in number over the last four decades. In 1980 there was a total global count of 124 natural disasters and in 2023 a total of 410 of which 166 were floods, that's 40.4% [1]. This is the most frequent of all of the natural disasters, and this increase could be attributed to rising sea levels and climate change. Between the years of 2007 and 2018 sea level has risen approximately 3.7mm a year [2] this means areas closer to tidal rivers and coasts will experience a higher number of flood events.

In order to map flooding, water indexes are often required. An example of such indices is NDWI (Normalised Difference Water Index) which works off the concept that different surfaces on earth reflect sunlight in different ways [8]. Water tends to reflect NIR (Near Infrared) and SWIR (Shortwave Infrared) which can be picked up on satellites. It produces as measured between +1 and -1, with the higher the number indicating a higher level of water in the measured area and is often used to measure spatial and temporal change in water.

As we as humans develop we need to find better ways to maintain our physical world and environment, especially in this time of rapid change. Earth Observation (EO) has become increasingly more advanced with the development of new satellite technology such as the Sentinel range operated by the European Space Agency (EAS) in their Copernicus project [3]. These aim to provide high resolution images of earth which provide us detailed understanding of the land we live in. Sentinel 2A provides 1 terabyte of valuable data a day, 365 terabytes a year [4]. With this much data provided it’s difficult to analyse it manually as

well as dangerous, as we as humans are relatively slow at processing data. When lives are at risk, the speed of flood detection becomes crucial, as delays could cost lives.. This is where AI comes into play.

Techniques such as semantic segmentation, which is a machine learning algorithm, more specifically a deep learning algorithm, are used to highlight and label elements in an image. It does this by classifying pixels in an image by using a convolution neural network (CNN) such as U-net, ResNet or Fully Convolutional Network (FCN). This can be seen in figure 1. Using this technique we are able to identify segments in images provided by Sentinel satellites. This has been shown to be effective in identifying both flooding [6] and wildfires [7]. This optimises disaster detection and reduces response time for emergency services, potentially saving lives..

This project is heavily inspired by the work of Zhang et al. 2022 [6] and will aim to reproduce similar work and then to use an alternative CNN backbone and flood dataset, therefore testing the adaptability of semantic segmentation method as well as its ability to consistently produce accurate results across different datasets. The project used U-net and WorldFloods dataset, in this proposed project will aim to use a ResNet and another sentinel 2 based dataset to produce a semantic segmentation program that can accurately identify flooding within the images.

This proposal is structured into Background research, Proposed project, Work plan, Proposed resources. The background section will contain closely related works and alternative systems which share similar goals to this project. The proposed project will involve an outline of how the project's aims and methodology for the semantic segmentation. The work plan will outline a timeline of workflow in order to maintain this project. Finally the proposed resources section will give a brief outline of the requirements for this project, this will break down the hardware that will be required for this project.

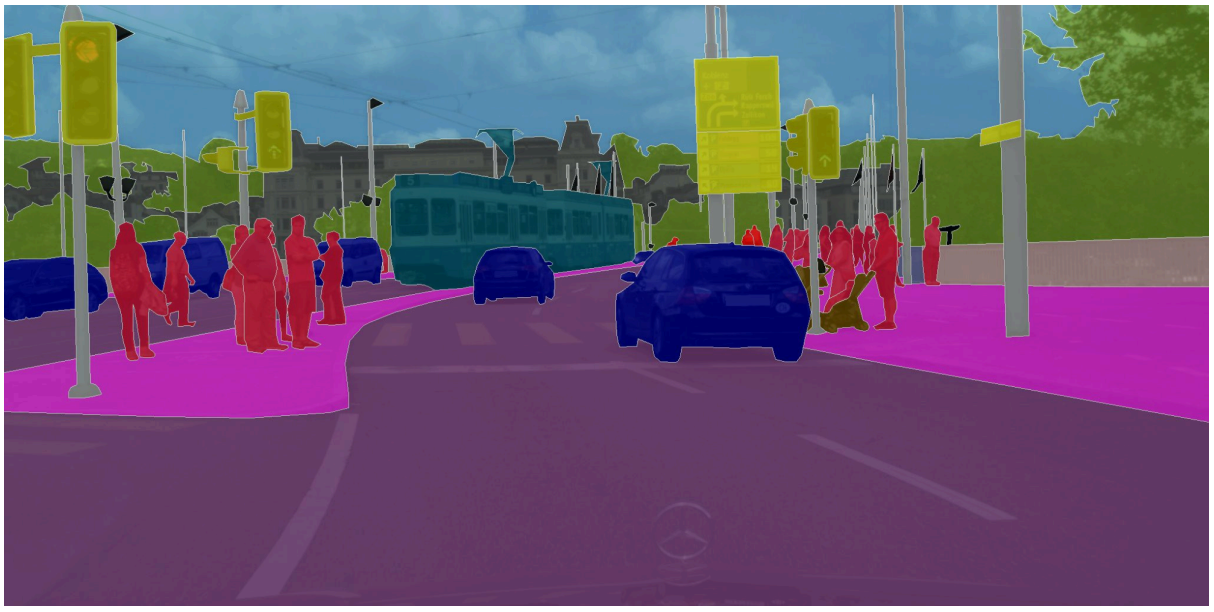


Figure 1: Segmentation of a road scene Image source [5]

## 2.Background research

This proposed project is heavily linked to Zhang on interpretable earth observation using semantic segmentation [6]. This paper proposed and tested a prototype-based semantic segmentation method for detecting flooding using the sentinel 2 satellite imaging. Their use of the massive 300Gb WorldFlood dataset provides a broad array of images to train and to test on. This was driven by a CNN called U-net which is structured into 2 sections, an encoder and decoder. The encoder is responsible for breaking down the initial input image into features, while the decoder is used to upscale these features, eventually combining into a map of all the features on the original image. Both of these are symmetrical and are connected via paths which can help speed up the process.

The prototype-based segmentation technique allowed for an increased interpretability of the results. Classes of features could be broken down into further sub classes (for example different types of water bodies) due to the clustering methods used such as k-means. Each cluster would represent a prototype of a certain sub class. These prototypes were used to increase human interpretability as they could be used to form linguistic rules. Explainable AI (XAI) is a massive part of deciphering the decision making and results created by AI.

Importance of explainability is further emphasised in related research, particularly in the context of AI legislation such as the Artificial Intelligence Act (AIA) [9]. The second paper delves into the concept of explainability and its broader implications for metrics and legislation. The AIA aims to ensure transparency, lawfulness, and fairness in AI while establishing standards for quality and accountability. This is particularly relevant to earth observation and similar applications where explainable AI can empower users and ensure effective interpretation of outputs so the correct course of action can be taken.

Zhang's project proved their method could accurately segment satellite imaging to highlight areas prone to flooding. This has incredible applications for flood response and disaster management. It can provide a holistic view of flood impact in both urban and rural environments and provide a comprehensive overview of the damage for responders. This drastically reduces wasted resources, time to respond, creates easy identification of high risk areas and allows faster response times from emergency services.

## 3.Proposed project

### 3.1Objectives

The aim for this project is to produce a similar program to the work of Zhang more specifically his work on interpretable deep semantic segmentation for earth observation [6]. Furthermore, this project will then be altered by altering the CNN backbone and the satellite image dataset. The following objectives include:

- Implementing a deep semantic segmentation for earth observation - this involves implementing a  program that segments sentinel-2 images to identify whether there was flooding in the region.

- Adapting the program to use a different CNN - this will involve replacing the U-net backbone in the program. The replacement CNN will be ResNet due to its capabilities in handling high definition earth observed images [10].
- Finding a new flood data set - this will involve finding a new sentinel-2 flood dataset for the model to be trained and tested on.

### 3.2Methodology

This project will utilise the waterfall method to drive its implementation due to the project's linear direction. The project will be split up into logical sections. This will be described in the Work Plan (section 4).

This project will be divided into two phases: replicating Zhang's work on semantic segmentation for earth observation and ensuring consistent results with his work. Second, the adaptation of his work, this project will first test Zhang's original work with a new data set to see if it can produce accurate results using different satellite images (sentinel-2 based). Finally this project shall adapt the CNN backbone of Zhang's project, replacing its U-net CNN with ResNet.

The key tasks in the implementation stage are as follows:

- **Data preprocessing**: The satellite images from the WorldFlood dataset will be preprocessed to match the format Zhang used. This includes normalising the images, resizing them as needed.
- **Model construction**: A U-Net architecture will be implemented. This will involve defining both the encoder and decoder components, as well as the symmetrical skip connections that allow for the model's efficient feature upscaling and integration.
- **Training and validation**: The model will be trained on the WorldFlood dataset. The key metric for evaluation will be Intersection over Union (IoU), which measures the overlap between the predicted flood regions and the actual flood regions. Additional metrics such as recall and precision will be used to assess the accuracy and reliability of the segmentation.
- **Testing and fine-tuning**: Once trained, the model's performance will be critically analysed, including visual inspection of the segmentation outputs to ensure accurate flood detection. If necessary, parameters such as learning rate and batch size will be adjusted to optimise the model's performance.

Here are the key tasks for the adaption stage:

- **Dataset adaptation**: In this step, I will replace the WorldFlood dataset with an alternative satellite image dataset, that uses the Sentinel-2 satellite. Testing across datasets will help assess the model's generalizability and robustness. The dataset will be preprocessed similarly to WorldFlood to maintain consistency in input data formats.
- **CNN backbone modification**: After validating the baseline model, the next step involves replacing the U-Net architecture with a different CNN backbone, such as ResNet. This will help explore whether a more advanced or alternative CNN can improve the model's accuracy and speed. These architectures may offer better feature extraction capabilities due to their deeper networks or more efficient designs.

The backbone will be integrated into the segmentation pipeline, trained, and evaluated using the same metrics as in the replication phase, allowing for simple comparisons.

The data shall be analysed in a similar fashion to Zhang in order to stay consistent with his work and allow for easy comparison between all objectives and the original source material.

The project will first be trained and criticised using human eye and images that are already confirmed to /to not contain flooding in order to confirm that the training is producing less loss and incorrect results. This will be performed in both the replication stage and the adaption stage concerning the CNN interchange. Ensuring that progress is being made in a positive direction.

## 4.Work plan

This project will begin in October 2024 and conclude in March 2025. It will be divided into the following sections:
Replication stage:

- Design and Analysis: This will involve analysing Zhang's work and then creating a design plan for the implementation of the  semantic segmentation project. This should take around 1 week
- Implementation: this segment will involve developing the replicated application based on the designs from the previous stage. This should take around 2-3 weeks.
- Training and testing: this will involve using the worldFlood dataset to train the application. Following the training, testing shall be performed on the application to produce the results that are consistent with similar studies. This should take 1-2 weeks.

Adaptation stage:

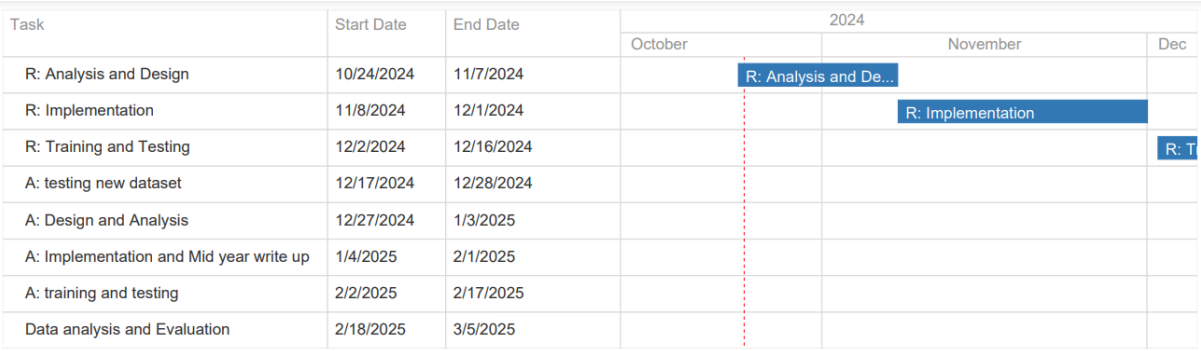
- Testing a new dataset: this will involve testing a new dataset on the project to test the robustness of the program. This should take around 1 week.
- Design and analysis of the new backbone: this will involve analysing the original backbone and designing a new one so they can be interchanged. This should take around 1 week.
- Implementation of the new backbone: this will involve stripping the current backbone out and implementing the new backbone into the program. This should take around 2 weeks.
- Training and testing: this should involve training the new backbone to work with the new dataset. Following this will be testing the adapted project. This should take 1-2 weeks

Data analysis and evaluation

- Once both the replicated project and adapted project have been completed and tested, empirical tests (such as IoU) shall be performed on both of them to highlight similarities and differences between these and the original material. This will bring to light whether the adaptions are better for earth observation regarding flood detection or are in fact detrimental. This will take around 1-2 weeks.

The overall schedule for this project is displayed in the Gantt chart (Figure 2) below:

| Task | Start Date | End Date |
|---|---|---|
| R: Analysis and Design | 10/24/2024 | 11/7/2024 |
| R: Implementation | 11/8/2024 | 12/1/2024 |
| R: Training and Testing | 12/2/2024 | 12/16/2024 |
| A: testing new dataset | 12/17/2024 | 12/28/2024 |
| A: Design and Analysis | 12/27/2024 | 1/3/2025 |
| A: Implementation and Mid year write up | 1/4/2025 | 2/1/2025 |
| A: training and testing | 2/2/2025 | 2/17/2025 |
| Data analysis and Evaluation | 2/18/2025 | 3/5/2025 |



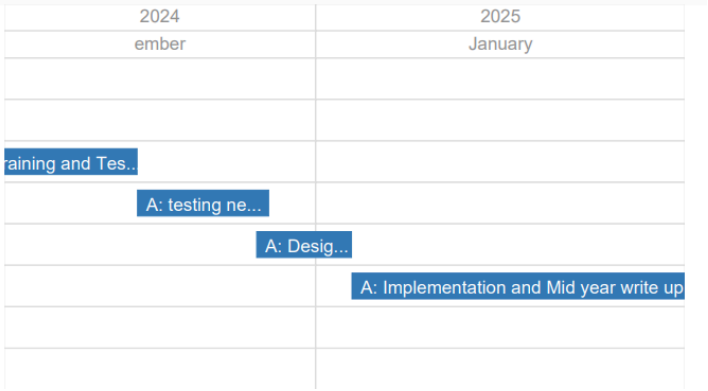




Figure 2: project schedule 2024-2025. R = Replication stage, A = Adaptation stage

Dates are subject to change.

## 5.Proposed resources

Access to a powerful GPU, CPU and a large amount of RAM will be vital for this project due to its massive processing requirement. This device will be provided by myself or Lancaster University if my own machine does not have the specifications required in order to run and build this project.

## 6.References


1. Number of recorded natural disaster events (Our World Data): https://ourworldindata.org/grapher/number-of-natural-disaster-events?country=All+disasters~Flood
   (accessed 22/10/2024)
2. Past and Future sea level rise (Met Office): https://www.metoffice.gov.uk/weather/climate-change/organisations-and-reports/past-and-future-sea-level-rise#:~:text=Past%20sea%20level%20rise&text=It%20is%20also%20clear%20that,over%20the%20period%202006-2018.
   (accessed 22/10/2024)
3. Onoda, M. & Young, O. R. Satellite Earth Observations and Their Impact on Society and Policy (Springer Nature, 2017):

https://onesearch.lancaster-university.uk/discovery/fulldisplay?docid=cdi_askewsholts_vlebooks_9789811037139&context=PC&vid=44LAN_INST:LUL_VU1&lang=en&search_scope=MyInst_and_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,M.%20Onoda%20and%20O.%20R.%20Young,%20Satellite%20earth%20observations%20and%20their%20impact%20on%20society%20and%20policy.%20Springer%20Nature,%202017&offset=0

4. What you should know about Sentinel-2 climate satellites (Airbus 2024): https://www.airbus.com/en/newsroom/stories/2024-08-what-you-should-know-about-sentinel-2-climate-satellites#:~:text=The%20Sentinel-2%20satellites%20capture,swath%20width)%20of%20290%20km.
(accessed 22/10/2024)
5. Gupta, Divam. A Beginner's Guide to Deep Learning based Semantic Segmentation using Keras (2019): https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html
(accessed 22/10/2024)
6. An Interpretable Deep Semantic Segmentation Method for Earth Observation (IEEE explore 2023): https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10019621
(accessed 22/10/2024)
7. Uni-temporal Sentinel-2 imagery for wildfire detection using deep learning semantic segmentation models (Taylor & Francis, 2023): https://www.tandfonline.com/doi/epdf/10.1080/19475705.2023.2196370?needAccess=true
(accessed 22/10/2024)
8. How Normalized Difference Water Index is Used to Map Flooding (Geography Realm 2020): https://www.geographyrealm.com/normalized-difference-water-index-flooding/
(accessed 22/10/2024)
9. Metrics, Explainability and the European AI Act Proposal (MDPI AG 2023): https://onesearch.lancaster-university.uk/discovery/fulldisplay?docid=cdi_doaj_primary_oai_doaj_org_article_a8e52d03c3cb43dcb9dc4646ef63ca2b&context=PC&vid=44LAN_INST:LUL_VU1&lang=en&search_scope=MyInst_and_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,metrics,%20Explainability%20and%20the%20European%20AI%20Act%20Proposal
(accessed 23/10/2024)
10. MP-ResNet: Multipath Residual Network for the Semantic Segmentation of High-Resolution PolSAR Images (IEEE Geoscience and Remote Sensing Letters, 2022, Vol.19, pp.1-5): https://onesearch.lancaster-university.uk/discovery/fulldisplay?docid=cdi_proquest_journals_2615165285&context=PC&vid=44LAN_INST:LUL_VU1&lang=en&search_scope=MyInst_and_CI&adaptor=Primo%20Central&tab=Everything&query=any,contains,resnet%20semantic%20segmentation&offset=0
(accessed 23/10/2024)