Identifying Flood Water Using Deep Learning Algorithms Trained on Satellite Data from Sentinel-1 and Sentinel-2 Satellites : CS 7643

Isaac Jensen, Jennifer Jordache, Dong Wuk Kim, Sungmin Park
Georgia Institute of Technology
ijensen6@gatech.edu, jjordache3@gatech.edu, dkim895@gatech.edu, spark805@gatech.edu

Abstract

Flooding has historically been one of the worst natural disasters, both in terms of monetary damage and lives claimed. While modern advances in infrastructure and disaster response have resulted in less deadly flood events, flood risk itself is expected to increase. Accurate flood mapping being critical to aiding in search and rescue efforts, there is a great demonstrable need for automated flood water identification techniques that can rely on quickly available satellite data. In this project, we implement various fully convolutional network architectures from scratch and train them to create segmentation maps of flooded areas using a curated flood water dataset [2]. We also replace the segmentation head of a vision transformer pre-trained on other satellite data and resume training on our flood water dataset. The following details our motivation for this project, our approach and experimental setup, and our results in comparing our networks against each other.

1. Introduction/Background/Motivation

Our main goal with this project was two-fold. The first of which was to gain an in-depth understanding of image segmentation by implementing fully convolutional networks using only functions from the Torch library. The second was to achieve the best flood water predictions that we could on the test dataset, as measured by an intersection over union (IOU) score. Image segmentation involves assigning a label to every individual pixel in an image.

Modern efforts in semantic segmentation using remote sensing data have largely revolved around fully convolutional networks (FCNs), particularly U-Net based architectures. Originally developed for biomedical image segmentation [3], a U-Net consists of an encoder portion, where high level semantic features are extracted using a series of convolutional blocks and max pooling layers, and a decoder portion, where these high level feature maps are upsampled and combined with the feature maps from the corresponding encoder layer. This method combines the feature localization information preserved in the higher layers of the network with high-level feature representation of layers lower in the network.

More recently, Vision Transformers have approached and in some cases, surpassed the performance of convolutional networks on computer vision tasks like classification and segmentation, particularly when pre-trained on large amounts of data. Through this model, an image is split into fixed-size patches, each of which are linearly embedded, with added position embeddings and then fed into a traditional transformer encoder. Performing classification or segmentation is done by adding an extra learnable layer.

A significant challenge that current state of the art models face when creating segmentation maps of flooded areas is generalization to unseen geographies and locations. While it is probable that models based on graph neural networks and neural operators can do a better job of generalizing across new case studies [8], these sorts of deep learning advancements are out of scope for this paper.

Success in our efforts will not only showcase that accurate flood maps can be produced with limited data, but can be produced quickly and in real time. Even if we are not able to produce better results than the current state of the art techniques, experimenting with different architectures that may not be commonly used for segmenting satellite data may open the door to further creative thinking for this particular task and segmentation of satellite data more generally.

For this project, we used the dataset provided alongside Cloud2Street's paper *Sen1Floods11: a georeferenced dataset to train and test deep learning*

flood algorithms for Sentinel-1 [1]. It is a collection of 512x512 images taken by the Sentinel-1, Sentinel-2, and Landsat satellites, as well as their corresponding ground truth labels, created for the purpose of aiding researchers in training flood segmentation models. It comprises: i) 446 hand-labeled chips of surface water from flood events; ii) 814 chips of publicly available permanent water data labels from Landsat (JRC surface water dataset); iii) 4,385 chips of surface water weakly-labeled from Sentinel-2 images from flood events and iv) 4,385 chips of surface water weakly-labeled from Sentinel-1 imagery from flood events. Weakly-labeled chips were labeled using OTSU thresholding with Sentinel-1 data and traditional Sentinel-2 Classification derived from Sentinel-2 data. More details on labeling methods and flood event locations used can be found in the original paper [1].

All flood event chips have corresponding 2 band Sentinel-1 images, and the 446 hand-labeled chips also have corresponding 13 band Sentinel-2 images. In this project, we tuned our models on Sentinel-1 data only, as it is easier to work with and would be more reliably available in real flood situations (Sentinel-2 data may be unavailable depending on weather conditions over the flood zone). It also appears that the Driven Data competition allowed use of only Sentinel-1 data [5].

Label chip pixels are labeled either 0 for no water, 1 for water, or -1 for no data, in the event a chip was from the edge of an imaged area, or the pixel could not be conclusively categorized. The hand-labeled set was further divided into training (252 chips), test (90 chips), and validation (89 chips) sets. An additional 15 chips, all from Bolivia, serve as a unique country test set. All weakly-labeled data was used for training only, never validation or testing. Hand-labeled chips from Bolivia were reserved for the test set only, in order to determine how well the networks performed on a flood event from a region on which they had not trained.

2. Approach

2.1. Starting Framework

Cloud2Street provided a starting framework in the form of a Jupyer notebook which we used in order to load training, validation, and test data. All data is normalized according to the mean and standard deviation calculated from the hand-labeled training set, as this is the only set with reliable labels that is not used in testing or validation. The notebook was initially configured to run training and validation with Torchvision's FCN_RESNET50 model, which Cloud2Street had used to set the benchmarks in their paper, and which we use to set our baselines. FCN ResNet-50 is a fully convolutional network with a ResNet-50 backbone. It lacks the skip connections between encoder and decoder stacks that characterize U-Nets, instead opting to upscale each layer of the encoder to the final output size, then combine them all together before the segmentation head. See the original paper for more details on its implementation [13].

The notebook also provided a function to convert batch BatchNorm2d layers into GroupNorm layers. Group normalization has been shown to work better than batch normalization when working with small batch sizes [12], which was necessary in our case due to the size of our weights and feature maps.

2.2. U-Net Architecture

The key component of the U-Net architecture is the skip connections between the encoder and decoder blocks. As seen in Figure 1, after the bottleneck, the output of each convolutional block is upsampled and concatenated with the output of the encoder block from the above layer. This concatenation combines the coarse but semantically meaningful information of the final feature maps produced by the encoder with the finer but less semantically meaningful feature maps of the higher layers. In other words, it produces the best of both worlds: good localization and good feature representation.

For this project, we implemented our own U-Net using only built-in PyTorch modules. The method for creating the convolutional blocks themselves using an OrderedDict was taken from a GitHub repository [14], and was modified to include dropout layers for regularization. Each convolutional block in the encoder stack consists of two 3x3 convolutions with same padding and ReLU activation layers, and is followed by a 2x2 max pooling layer with a stride of 2. The number of feature maps is doubled at each block of the encoder stack, and the resolution is halved by the max pooling layer. This is the same process a traditional convolutional network uses to extract feature information, but instead of the output of the final encoder block being fed into a linear classification head, it is upsampled using a learnable transpose convolution kernel that doubles its resolution and halves its number of feature maps. At this point, the output of the transpose convolution has the same dimensions as the output of the next-to-last encoder block output, allowing us to concatenate them and feed the concatenation into the first block of the decoder stack. This upsampling and concatenation process is repeated until the output of the top decoder block is fed through a 1x1 convolution in order to produce two 256x256 maps, the first corresponding to "not water" scores for each pixel, and the second corresponding to "water" scores. These scores are fed through a softmax layer to generate water and not water probabilities for each pixel. It should be noted that in the case of binary

segmentation, as exists here, the output of the segmentation head could be a single channel and be fed through a sigmoid activation in order to produce a final probability for water in each pixel. Functionally, it is almost the same, but we produced a two channel output because it tended to learn slightly more quickly at the beginning of training for negligible additional cost.

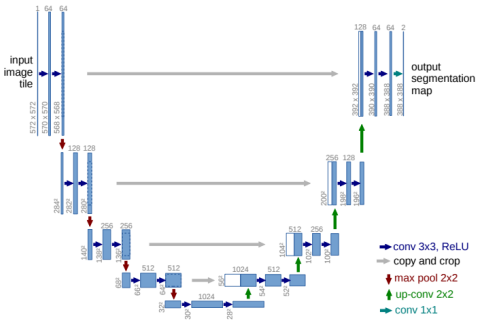

Figure 1. U-Net architecture [3]

2.3. Attention U-Net

We further investigated techniques to improve our network model and researched how attention can improve the learning. Attention is a widely implemented technique in neural networks. Attention focuses on features that are more meaningful, while deemphasizing the less meaningful features. We referenced the attention U-Net from the "Attention U-Net: Learning Where to Look for the Pancreas."[4]

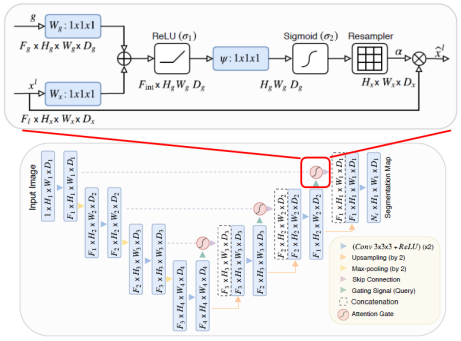

Figure 2. Attention U-Net structure [4]

In a standard U-Net, spatial information from the encoder path brings along the poor feature representation from the initial layers. A soft attention technique enhance the skip connections by actively suppressing activations at less relevant regions, while placing more weight to the features of interest. As shown in Figure 2, attention gates combine the skip connection from the encoder, which contains more spatial information, with the gating signal from the next lowest layer of the network, which contains better feature representation coming from the deeper layers on the network. Through the attention block, aligned weights from two inputs get larger and unaligned weights get relatively smaller, adding weights to pixels based on the relevance. Then a ReLU activation is applied, followed by 1x1 operation ψ applied to generate a weights matrix. A Sigmoid layer is then required to scale all output to between 0 and 1 and the result from the attention block is upsampled to the original size of the gating signal and is concatenated to follow the basic U-Net structure.

Following the architecture, we had modified the Attention U-Net to upsample at the end of a layer, after the attention block, whereas the reference model [10] upsamples inside the attention block. The results of the two models were identical in terms of performance, thus we decided to pursue further experiments with the default Attention U-Net from the reference.

Adding attention blocks to U-Net showed immediate improvement to our validation IOU. The result of the experiments will be discussed in more detail in the results section.

2.4. LinkNet

In our research of U-Net architectures, we also came across an image segmentation approach called LinkNet. Similarly to U-Net models, the image is decomposed at the encoder and decoder levels, with the image being reconstructed before the final convolution layer. However, while in the U-Net model a concatenation is done at the decoder stage, the LinkNet model performs an addition instead, making it a very light neural network structure and quite suitable for real-time use. Given this advantage and our motivation for creating models that could have a use case during a real time flooding emergency, we decided to create a LinkNet model using PyTorch functions. The implementation was modeled off an example on the Kaggle website [7], with the major changes coming from different input channel values and different parameter values.

2.5. Transformer

Another architecture that we explored was vision transformers (ViTs). Given the heavy data requirement of transformers and the fact that we had limited data that was weakly-labeled, and even less data that was hand-labeled, we decided to look into pre-trained models that would remove the need for us to find a large and labeled remote sensing dataset.

One of the challenges with finding a pre-trained model for our task was that the majority of pre-trained models available have been trained on images with 3

input channels (R, G, B), whereas remote sensing data from Sentinel-1 and Sentinel-2 have 2 and 13 channels, respectively.

Luckily, we were able to find a pre-trained self-supervised ViT model that used both Sentinel-1 and Sentinel-2 unlabelled data to train its weights [9]. The team that produced the pre-trained ViT also provided methodology to add a task-specific head to the model that was able to be fine-tuned on much smaller, labeled datasets for different tasks.

More specifically, the model is defined by training a ViT on unlabelled data via self-supervised learning. Self-supervised learning (SSL) aims to learn data representations from unlabelled datasets, with this model in particular using SSL via contrastive learning. SSL contrastive learning trains neural networks to learn the relationships between different data points, with embeddings of similar points being structured near each other and further apart for different data points.

The ViT used in this model is known as a Swin Transformer, which addresses a downside of a standard ViT. Standard ViTs are known to have difficulty capturing fine details due to a fixed patch size, limiting their ability for dense pixel-level predictions. The Swin Transformer separates the image into patches and applies self-attention to groups of non-overlapping patches. Finally, a scheme that is known as a 'shifting window' is used in order to connect attention across groups, which overall creates a better holistic representation.

Our final approach consisted of using the pre-trained backbone as depicted in Figure 10, along with a segmentation head depicted in Figure 11, both located in the appendix section C. During our experiments, we tested both freezing the pre-trained model weights and fine-tuning the segmentation head, as well as fine-tuning the entire model.

3. Experiments and Results

It was difficult to determine what metric and baseline to use at the beginning of experimentation, as we did not yet know how challenging it would be to improve our models' performance. At first, we simply decided to see if we could beat the IOU score 0.4357 given as a baseline in the Driven Data competition for the Sen1Floods11 dataset [5]. After a little experimentation, this proved to be trivial, so we decided on an easy target of improving the FCN_RESNET50's validation IOU of 0.4779, and a difficult target of achieving a validation score of above 0.7.

Our initial experiments focused on our basic U-Net model. We added functionality to choose the number of layers in the model, although we discovered that 5 layers performed the best while also being the most that our testing hardware could accommodate.

3.1. Loss Functions

After getting the basic U-Net model working, we began experimenting with different loss functions. The obvious one to try was IOU, as that was what we were using for our evaluation metric, but it did not yield significantly better results than the weighted cross entropy loss that Cloud2Street used (CrossEntropy with a weight of 8 for water and 1 for no water).

Dice Loss, based on the Sørensen–Dice coefficient, was our next choice for loss function. It is similar to IOU, which can be computed in the binary case as $\frac{TP}{TP+FP+FN}$, but instead multiplies both true positive counts by 2. This causes proper classification of true positives to contribute more overall to the score and can be useful for segmentation in imbalanced datasets where the background greatly outnumbers the target class [6], as is the case for our project. As we hoped, Dice Loss produced a sizable increase in validation IOU (0.05).

Finally, we experimented with Focal Tversky Loss, a variation on dice loss with α and β balancing coefficients for *FP* and *FN*, as well as a γ focal loss exponent. Initially, Dice Loss outperformed Focal Tversky Loss, although ultimately Focal Dice Loss would provide the best validation results. Code for loss functions was taken from a Kaggle example [15] and modified to accept our models' 2 feature outputs and masked label maps.

Table 1 shows our initial validation results for our candidate U-Nets run for 100 epochs using Dice Loss. Learning curves can be found in appendix section A.

Network	U-Net	Link Net	Attention U-Net	ResNet50 Baseline
Validation IOU	0.5871	0.5083	0.5977	0.5270

Table 1. Initial validation IOUs using AdamW, cosine annealing with warm restarts, and a learning rate of 5e-6 for 100 epochs (ResNet used learning rate of 5e-4)

3.2. Data Augmentation and Regularization Layers

The notebook provided alongside the data contained functions to augment the training data by randomly cropping 512x512 images down to 256x256 samples then horizontally and vertically flipping them. In order to improve regularization, we added random rotations as well as random gaussian blur. The addition of

gaussian blur in particular resulted in eliminating overtraining on hand-labeled S1 data in Attention U-Net even beyond 1000 epochs.

While we initially experimented with adding dropout layers to the convolutional blocks of the U-Nets, we discovered that we did not see much improvement from them. We theorize that this is because, while dropout layers in a CNN do zero portions of the inputs to convolutional layers, they do not prevent any weights from contributing to the output. As the convolutional layer strides over the input feature map the zeroed inputs act more like random noise inserted into the image. This may be beneficial, but it may not help build independence between weights. We therefore switched the dropout layers to dropout2d layers, which randomly zero entire channels of the input feature map, making them more akin to how dropout layers act for a fully-connected layer. This change led to an immediate improvement, taking our best validation IOU on Attention U-Net from around 0.6 to 0.62.

3.3. Optimizer and Scheduler

With Attention U-Net being consistently our best performer, we conducted further experiments to improve the model. Optimizers are methods used to change attributes such as weights and the learning rate in order to reduce loss more efficiently. In particular, different optimization algorithms can work differently for a given model. Adam, AdamW and Stochastic Gradient Descent (SGD) optimizers were selected and the experiment results are shown in Table 2 with control parameters (LR: 5e-4, Epochs: 30, Momentum: 0.9).

Optimizer	Max IOU
SGD	0.4973
Adam	0.4563
AdamW	0.5087

Table 2. Optimization experiment results

As a result, we concluded that AdamW and SGD are the two best performing optimizers with the Attention U-Net.

We then explored a number of schedulers, as depicted in Table 3, to alter the adaptive learning rates in combination with the optimizer functions. We first ran various schedulers to compare the performance with default parameters to sort out the better performing schedulers. CosineAnnealing with AdamW optimizer showed a solid performance over others and CyclicLR with SGD and momentum showed positive results. Cyclic LR is a technique used to apply cyclically changing learning rate for each batch, which may help the loss escape from saddle points or local minima by applying higher learning rates within the loop. A maximum learning rate of 1e-2 was used, while previous control parameters remained the same.

IOU Type	Cosine Annealing	MultiStep LR	Exponential LR	Cyclic LR
Training IOU	0.4037	0.2773	0.2592	0.5244
Validation IOU	0.4636	0.2373	0.1962	0.5958
Max IOU	0.5121	0.3415	0.3636	0.5958

Table 3. Scheduler experiment results

Consequently, further investigation and experiments were conducted for the two best performing optimizers and schedulers to achieve the final best max IOU. More detailed tuning can be found in Tables 6 and 7, located in appendix section D.

3.4. Focal Dice Loss and Lowering the Learning Rate

As Attention U-Net continued to be our best performer, we decided to conduct longer training runs with it for the S1 hand-labeled data. We noticed that, while it usually achieved good results, it had a very unstable learning curve (even more so than the other networks). This was likely in part due to the small batch sizes, which were necessary even on a V100 GPU due to the size and number of the feature maps and kernels in the 5 layer Attention U-Net. Our final improvements came from decreasing the learning rate to 1e-6 (down from 5e-4 originally, and 5e-6 for our network comparison runs). This learning rate and the use of a γ focal term of 4/3 applied to Dice Loss, combined with a Cosine Annealing scheduler and AdamW optimiser, achieved a best validation score of 0.6475. The loss curve was still quite unstable, even with such a low learning rate, as can be seen in Figure 9, located in section B of the appendix.

Our results on the test sets, shown in Table 4, were largely in line with what we expected. On non-Bolivia test data, the S2 weakly-labeled data outperformed the S1 hand-labeled data, which was consistent with the results published in the original paper [1]. The S1 weakly-labeled data shows a quite large (39%) improvement on the Bolivia test set over non-Bolivia, but this is also consistent with the original paper, where it showed a 36% improvement. The Bolivia results

could be interpreted positively, such as the network properly learning generalized flooding representations independent of the training data's country of origin, but it should be noted that the Bolivia test set contains only 15 images. IOU scores above 0.7 were not all that uncommon in individual training batches. We believe we would need more high-quality training data from a new location in order to truly determine how well our model is generalizing.

Dataset	Non-Bolivia	Bolivia
ResNet50 Baseline	0.4810	0.5046
S1 Hand-Labeled	0.5464	0.5587
S1 Weak Otsu	0.4860	0.6756
S2 Weak	0.5723	0.6275

Table 4. Final test IOU scores for Attention U-Net compared to the ResNet50 baseline. Training times of 15 epochs for S1 Weak Otsu, 62 epochs for S2 Weak, and 1613 epochs for S1 Hand-Labeled

3.5. Transformer Results

We tested the transformer on both the hand-labeled and the weakly-labeled datasets, with ultimately the most success coming from the hand-labeled datasets (particularly images from Sentinel-1). This was largely expected given that the model was designed to be able to add a segmentation head for the supervised training of a small, labeled dataset. The highest IOU of 0.6052 was obtained after 800 epochs, using a learning rate of 1e-3, the AdamW optimizer and Dice Loss as the loss function. In particular, this IOU was achieved by freezing the pre-trained weights and only training the segmentation head, whereas fine tuning the entire model resulted in poor performance. The reasoning behind this outcome could be described by a number of factors, such as not having enough compute time to train the entire model or not finding the right combination of parameters that would lead to model weights as useful as the pre-trained version. Overall, tuning of parameters was done extensively for both the frozen pre-trained weights version of the model and the fine-tuned weights version for both the Sentinel-1 and Sentinel-2 datasets, however we could not reach the success displayed by the Attention U-Net. The results of the tuning on the hand-labeled Sentinel-1 dataset can be viewed in Table 8 and Table 9, in section E of the appendix.

4. Conclusion

Considering that we did not meet our initial goal of 0.7 validation, it is difficult to call our project a total success. Looking back, a few areas for further experimentation and improvement stick out.

Our data came with pre-made training, validation, and test splits, as well as a training-validation loop function which we relied upon, but in actuality it would have probably been wise to make our own data loader with support for cross validation. The fact that the weakly-labeled S2 training data lead to better performance on the test set than the hand-labeled S1 data (which had better performance in validation) indicates that the model was not generalizing as well as it should have been. Given the small size of our high-quality hand-labeled dataset, however, this may not have yielded the results that we were after.

Had we begun planning of this project after assignment 4's release, vision transformers may have been a much larger focus. They were not part of our original proposal, and we did not have time to test them as much as we would have liked, but our initial experiments with transfer learning using the pre-trained Swin Transformer were promising. Further research would likely focus on finding large datasets that we could use for more self-supervised training.

That being said, it is hard to be too disappointed looking at the final results of our work. While far from a perfect reproduction of the label image, our Attention U-Net trained on S2 weakly-labeled data does a much better job capturing fine details and does not overestimate water coverage nearly as much as the results produced by the baseline ResNet50 model.

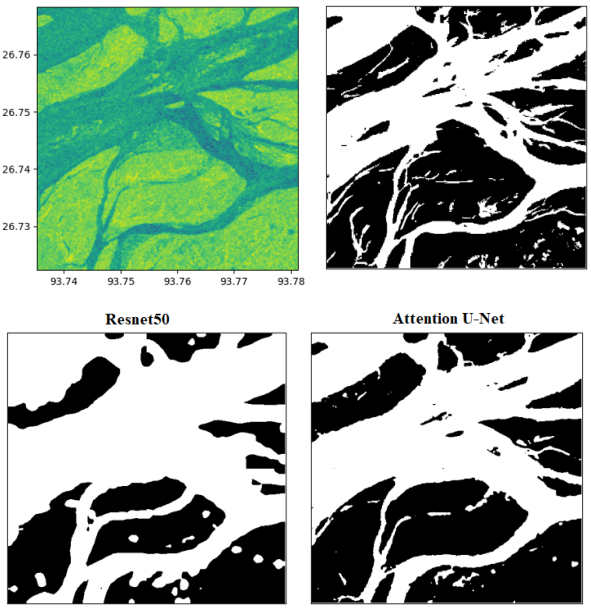

Figure 3. Baseline ResNet50 Prediction vs. S2 Weak Trained Attention U-Net Prediction for image patch India900498l

5. Work Division

Summary of contributions are provided in Table 5.

Student	Contributed Aspects	Details
Isaac Jensen	All modifications to starter notebook, U-Net architecture implementation, loss experiments, augmentations, prediction visualizer, regularization experiments, Google Compute Engine environment setup.	Added all loss functions, additional image augmentations, and other changes to the starter notebook. Implemented the basic U-Net with an adjustable number of layers. Ran U-Net experiments save for optimizer/scheduler experiments.
Jennifer Jordache	LinkNet and Transformer model development, analysis and experimenting	Adapted LinkNet and pretrained ViT models to fit our data and model training architecture.
Dong wuk Kim	Attention U-Net model development Schedulers experiments .tif to image converter script	Investigated Attention U-Net implementation and adapted a compatible Attention U-Net. Ran experiments with schedulers, generated a script to convert .tif data to images.
Sungmin Park	Attention U-Net model Development Experiments on optimizers and schedulers	Tested on different combinations of optimizers, loss function and schedulers.
All Group Members	Report writing Bi-weekly meetings	Contributed to writing, editing and finalizing of final report. All members participated in live coding sessions and discussion towards project progression.

Table 5. Contributions of team members

References

[1] Bonafilia, D., Tellman, B., Anderson, T., Issenberg, E. (2020), Sen1Floods11: a georeferenced dataset to train and test deep learning flood algorithms for Sentinel-1. *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshop*s, pp. 835-845, doi: 10.1109/CVPRW50498.2020.00113. 2, 5

[2] Bonafilia, D., Tellman, B., Anderson, T., Issenberg, E. (2020), Sen1Floods11: a georeferenced dataset to train and test deep learning flood algorithms for Sentinel-1. *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshop*, 2020, pp. 210-211. 1

[3] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Lecture Notes in Computer Science(), vol 9351. Springer, https://doi.org/10.1007/978-3-319-24574-4_28 1, 3

[4] Oktay O., Schlemper J., Folgoc L., Lee M., (2018) Attention U-Net: Learning Where to Look for the Pancreas, Biomedical Image Analysis Group, Imperial College London, London, UK 3

[5] Stack Overflow: Map Floodwater from Radar Imagery, *DrivenData*, https://www.drivendata.org/competitions/81/detect-flood-water/. 2, 4

[6] Jadon, S., (2020), A survey of loss functions for semantic segmentation. I*n Proceedings of the 2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Via del Mar, Chile, 27–29 October 2020; pp. 1–7. 28. Yi-de, M.; Qing, L.; Zhi-Bai, Q. Automated image segmentation using improved PCNN model based on cross-ent 4

[7] Sokolenko, M. (2021), Lips segmentation LinkNet Pytorch. *Kaggle*. Retrieved December 13, 2022, https://www.kaggle.com/code/mikhailsokolenko/lips-segmentation-linknet-pytorch/notebook 3

[8] Bentivoglio, R., Isufi, E., Jonkman, S. N., and Taormina, R., (2022), Deep learning methods for flood mapping: a review of existing applications and future research directions, *Hydrol. Earth Syst. Sci., 26,* 4345–4378, https://doi.org/10.5194/hess-26-4345-2022,. 1

[9] Scheibenreif, H., Mommert, B., (2022), *S*elf-supervised Vision Transformers for Land-cover Segmentation and Classification, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1422-1431 4

[10] Malav B., (2018), Unet Segmentation Pytorch Nest of Unets,https://github.com/bigmb/Unet-Segmentation-Pytorch-Nest-of-Unets 3

[11] John D., Zhang C., (2022), An Attention-based U-Net for Detecting Deforestation within Satellite Sensor Imagery, *International Journal of Applied Earth Observation and Geoinformation*

[12] Yuxin. X., Kaiming H., (2018), Group Normalization, *Facebook AI Research.* 2

[13] Jonathan, L., Evan, S., Trevor, D., (2015) Fully Convolutional Networks for Semantic Segmentation, *UC Berkeley,* 2

[14] Mateusz B.,, (2021), brain segmentation pytorch, https://github.com/mateuszbuda/brain-segmentation-pytorch/blob/master/unet.py 2

[15] RNA, (2021), Loss Function Library - Keras & Pytorch, https://www.kaggle.com/code/bigironsphere/loss-function-library-keras-pytorch/notebook 4

A. Model Results

A.1. U-Net

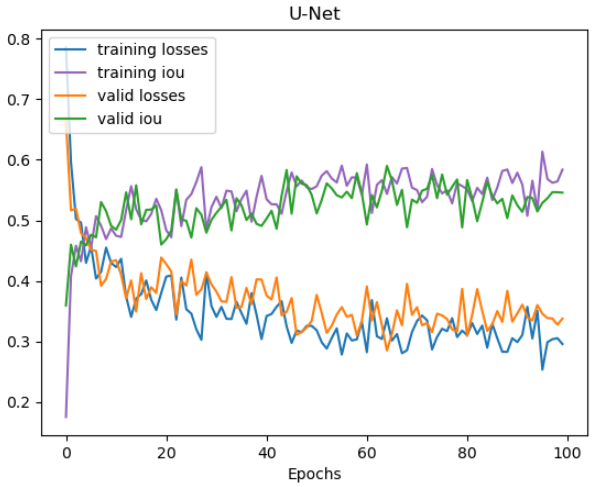

Figure 4. U-Net run for 100 epochs with Dice Loss, AdamW, cosine annealing with warm restarts, and a learning rate of 5e-6

A.2. ResNet 50

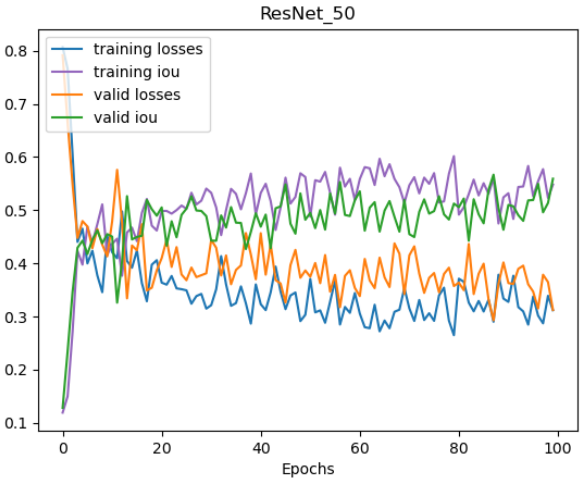

Figure 5. ResNet 50 run for 100 epochs with Dice Loss, AdamW, cosine annealing with warm restarts, and a learning rate of 5e-6

A.3. Attention U-Net

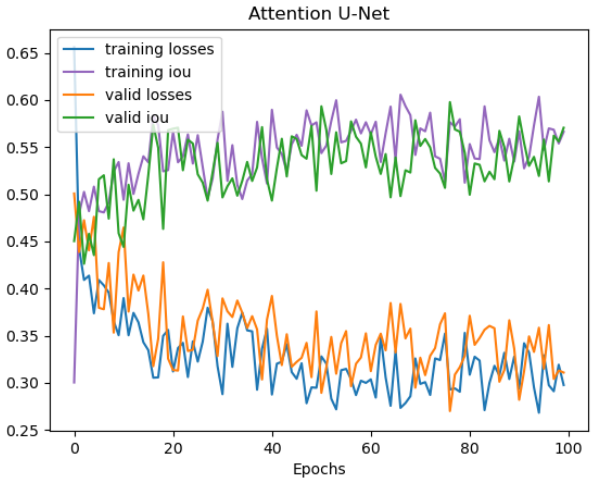

Figure 6. Attention U-Net run for 100 epochs with Dice Loss, AdamW, cosine annealing with warm restarts, and a learning rate of 5e-6

A.4. LinkNet

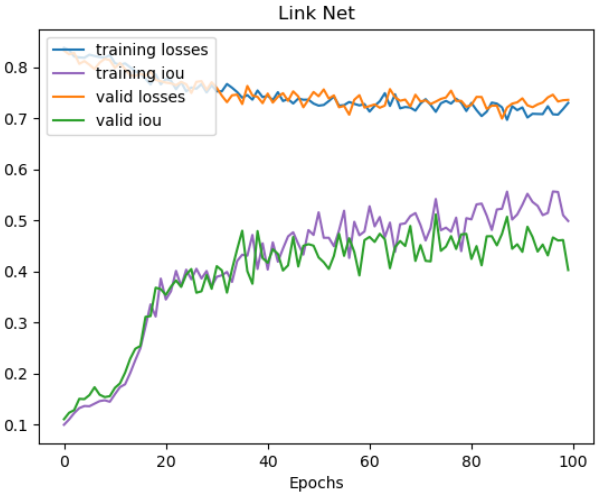

Figure 7. LinkNet run for 100 epochs with Dice Loss, AdamW, cosine annealing with warm restarts, and a learning rate of 5e-6

A.5. Pre-trained Transformer

Figure 8. Pre-trained transformer run for 100 epochs with Dice Loss, AdamW, cosine annealing with warm restarts, and a learning rate of 5e-6

B. Attention U-Net over 3000 Epochs

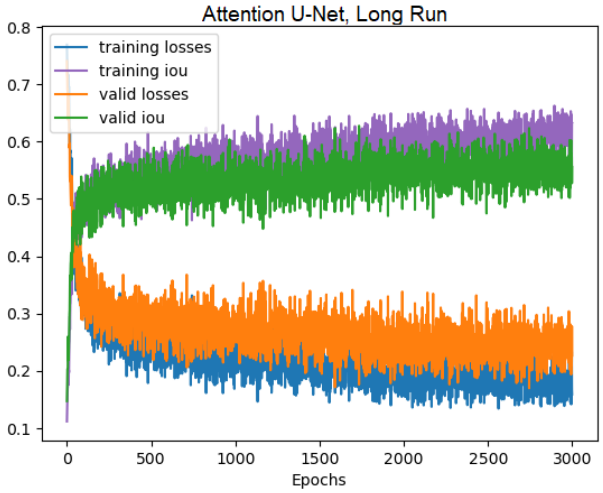

Figure 9. 3000 epoch training run of Attention U-Net with Gaussian blur and 1e-6 learning rate.

C. Transformer Architectures

C.1. Task-agnostic Backbone

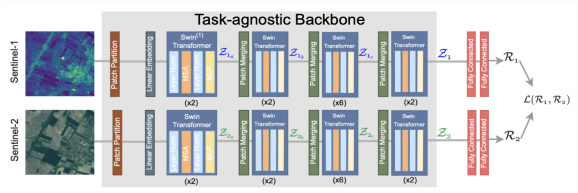

Figure 10. Architecture of the task-agnostic backbone, pre-trained on Sentinel-1 and Sentinel-2 input pairs

C.2. Segmentation Head

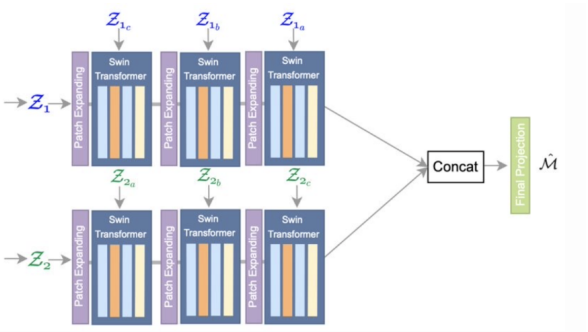

Figure 11. Architecture of the segmentation head that can be used in

D. Optimizer and Scheduler Tuning Values

	Learning Rate			
IOU Type	2.5e-5	3e-5	3.5e-5	5e-5
Training IOU	0.4495	0.5041	0.5199	0.5091
Validation IOU	0.4857	0.5189	0.5683	0.5551
Max IOU	0.5533	0.5518	0.5844	0.5626

Table 6. AdamW with CosineAnnealing for varying learning rates

	Learning Rate			
IOU Type	2.5e-5	5e-5	1e-4	2e-4
Training IOU	0.5158	0.4602	0.5417	0.4934
Validation IOU	0.5622	0.5239	0.585	0.5235
Max IOU	0.5992	0.5917	0.585	0.5729

Table 7. SGD with CyclicLR for varying learning rates

E. Transformer Fine Tuning

E.1. Frozen Pre-trained Layers Tuning

The results in Table 8 pertain to the transformer model with a frozen backbone (weights unchanged) on the Sentinel-1 hand-labeled dataset using Dice Loss.

	Learning Rate				
Optimizers	1e-2	1e-3	1e-4	1e-5	1e-6
Adam	0.4218	0.5331	0.5258	0.4888	0.4695
AdamW	0.4334	0.5367	0.5299	0.4917	0.4714
SGD	0.3662	0.4825	0.4637	0.4557	0.4226

Table 8. Frozen validation IOU results for different learning rates and optimizers over 100 epochs.

E.2. Full Model Tuning

The results in Table 9 pertain to the transformer model that we fine-tuned the pre-trained weights for on the Sentinel-1 hand-labeled dataset using Dice Loss.

	Learning Rate				
Optimizers	1e-2	1e-3	1e-4	1e-5	1e-6
Adam	0.189	0.1957	0.1567	0.1025	0.0841
AdamW	0.1922	0.1983	0.1462	0.0746	0.0712
SGD	0.0824	0.238	0.1369	0.1295	0.0831

Table 9. Fine-tuned validation IOU results for different learning rates and optimizers over 100 epochs.