Satlas Shrugged - Segmenting Deforestation Drivers from Satellite Imagery

Brage Hamre Skjørestad[1], Ludvik Olai Slipersæter[1], and Martin Flo Øfstaas[1]

[1]University of Bergen

Abstract

This is an applied deep learning paper which details our contribution in Solafune's "Identifying Deforestation Drivers" competition [1]. Our task was to create a model which can as accurately as possible, segment and classify the deforestation drivers from Sentinel-2 satellite images. The possible drivers were plantation, grassland/shrubland, mining and logging, and the performance measure was pixel-wise F1-score. We had 176 $1024 \times 1024 \times 12$ images available, with provided annotations. We first implemented an initial solution, and then we tried to improve upon these results with different methods. In our original approach, we used a large-scale pretrained satellite model [2] which achieved an F1-score of 0.5921. Our best model, which improved upon this baseline, achieved an F1-score of 0.6091, around 5% below the competition winners. Repository available at https://github.com/bragehs/INF367A-DeforestationDrivers/tree/main.

1 Introduction

Deforestation is the process of removing large areas of forests in order to make room for some non-forest use, such as agriculture, logging or urban development. Excessive deforestation can cause biodiversity loss and soil erosion, but it also releases a lot of carbon into the atmosphere [1]. These processes can help accelerate climate change, which makes it is a major environmental issue worldwide. In Amazonas especially, we have witnessed relentless deforestation over many decades. This alarming trend has, in turn, motivated a great deal of research, particularly in the field of deep learning. By training neural networks on satellite imagery, researchers can automatically detect where deforestation is occurring, and identify what types of land cover are replacing the forest. These models are very useful, especially because of the fact that raw satellite data is difficult to understand completely. Our goal in this project was to develop a deep learning model capable of providing insightful information in this context.

2 Related Work

The easiest, and probably best, approach to solve this problem is to use semantic segmentation, meaning we classify every single pixel in the image. This is very standard for these types of problems. The paper [3] is a similar problem where they use semantic segmentation. The only difference is that this is a multi-class problem, not binary. However, the choice of model architecture is more varied in these types of problems. A few years ago, using a ResNet architecture would be the most common, but now we are seeing more attention based solution, including [3]. The paper [2] provides both ResNet and attention based solutions (Swin-Transformer). Due to current trends, and results in [3] we decide to use a Swin-Transformer architecture. We wanted to stick to one model architecture to reduce the amount of hyperparameter-tuning needed.

3 Methods

3.1 Original Approach

Before we started implementing new and exciting deep learning methods, we needed to have an original solution in place. We wanted to make it quite standard, and looked at solutions provided in the discussion of the competition as inspiration for an intial pipeline.

3.1.1 Data Preparation

In semantic segmentation, the goal is to classify every single pixel in the image. Therefore, we needed to convert the file of polygon annotations into completely labeled images. We convert the labels into a [B, H, W] shape, and include background as an explicit label. The images themselves were preprocessed with standard NaN-filling and normalization. However, in addition to normalization, we also adjust the distribution of the pixel values to get more sparsity. This is done by matching the pixel distribution of each channel (for whole dataset) to some normal distribution reference. We split the images into 256×256 patches for computational purposes, and we apply both geometric and pixel-altering augmentations. See A.1 for for further details.

3.1.2 Model

As we previously mentioned, we implemented one of the satellite models trained on the "SatlasPretrain" dataset. [2]. They provide many different models for different purposes, explained at their repository. We used the model named `"Sentinel2_SwinB_SI_RGB"`. They also have pretrained models available with 9 bands, and we initially used this to leverage more of our available data. However, after testing both, we found that only using the `RGB` channels gave us the best results. In the supplementary material of [2] they also showed that `RGB` produced the best performance. The `"Sentinel2_SwinB_SI_RGB"` model contains a pretrained Swin-Transformer backbone, feature pyramid network, and upsampling block. On top of this, we create a simple and randomly initialized head in accordance with the class `SimpleHead` from the satlaspretrain model architecture (with task type `"segment"`).

3.1.3 Training

After examining how many pixels each class had in the entire dataset, we found a large imbalance. Therefore, we decided a standard cross entropy loss would not be ideal for this task. We implemented a custom loss function, which separates the loss for all classes and for background/foreground. For all classes, we implemented the loss function provided in [4]. For background/foreground we implemented a binary focal loss. This was done to prohibit over-segmentation of classes, as the F1-score is very punishing in this regard. The background/foreground loss was given weight 2.0, as we found the robust loss dominated otherwise. We did standard hyperparameter-tuning with 20% of the dataset, for computational purposes. See appendix for more detail.

3.1.4 Inference

For the test set we naturally implemented the same preprocessing steps as for training. We run predictions with 256 patch size and 64 patch stride, and try both with and without test time augmentation, with the same augmentations as in training. Test time augmentation did not improve our results in this instance. After running inference, some post-processing is required in order to submit a solution. First we need to stich together and average all predictions for an image, in order obtain image predictions in shape $1024 \times 1024 \times 5$. After this, we tried a variety of post-processing steps, for example Gaussian smoothing and removing small objects, but our best results came with only using the `argmax` function. Then we use the function `findContours` from the library `OpenCV` to find the the borders of the connected regions (polygons) in the labeled image. At last, we we write these polygons to a `json` file, where we disregard the background polygons.

3.2 PseudoSeg

PseudoSeg is a semi-supervised method, which tries to leverage unsupervised / weakly supervised data in addition to standard supervised data [5]. It uses a refined version of Grad-CAM, gradient class activation maps, and generates pseudo labels by combining these maps with the standard decoder output. The refinement process is trained on supervised data, and the pseudo labels are used to train with unsupervised / weakly supervised data. We chose to implement this method because of the size of our supervised dataset, and because we found a dataset from an old Kaggle competition which seemed promising. [6]. This was an image classification task, which means there are image labels available which can be utilized in the semi-supervised method. There were around 40000 $256 \times 256 \times 4$ images available, but with a lot more labels than we were interested in. After filtering, we ended with 1846 of those images to incorporate into our method. The data is preprocessed identically to the Solafune dataset. A.3 contains further details.

3.2.1 Implementation

The main obstacle in implementing PseudoSeg in this competition, is integrating it into the `"Sentinel2_SwinB_SI_RGB"` model. To do this, we modify the head we created in the original approach. Instead of just outputting the logits for each pixel, it also outputs classification logits and the "self-attention Grad-CAM" needed for the pseudo-labels [5]. To construct these maps we need the access to the backbone output. The backbone of our chosen model returns a list of outputs with varying sizes of spatial dimensions and feature channels. To construct the needed hyper-column [7], we use two of them. We concatenate the output in shape $256 \times 32 \times 32$ with the output in shape $512 \times 16 \times 16$. The 16×16 spatial dimensions is upsampled to 32×32 using the method `torch.nn.functional.interpolate`. After extracting the hyper-column from our chosen model, we can construct our self-attention Grad-CAM (SGC) maps and pseudo labels. We start by producing the Grad-CAM output [8] for each class, and then setting for non-present class maps to zero. These maps then gets inputted into the self-attention refinement module which is computes as follows

$$\hat{m}_i = \left(m_i + \sum_{j=0}^{L-1} \frac{e^{\kappa\left(W_k h_i, W_q h_j\right)}}{\sum_{k=0}^{L-1} e^{\kappa\left(W_k h_i, W_q h_k\right)}} m_j \right) W_c. \quad (1)$$

$\hat{m}_i$ is the Grad-CAM for a region i [5]. In my implementation, L is $32 \times 32 = 1024$, where L of course is the number of regions. We use the scaled dot product, denoted by κ, to find the similarity between a the projected regions h_i and h_j, where h_i

is the hyper-column data for region i. In our case, this will be 768 values per region i. Then a softmax is applied to these similarities to arrive at attention weights, and a skip connection to preserve the initial values. At last, another projection is applied to adjust for class differences. The "key" and "query" projections, $W_k \in R^{H\times H}$ and $W_q \in R^{H\times H}$ are implemented as 1x1 2D convolutions, and in our implementation they reduce the amount of feature channels from 768 to 128. $W_c \in R^{C\times C}$ is also a 1x1 2D convolution. The refinement module also applies batch normalization after W_c. The SGC maps for the background class is $2 - max(foreground)$, because of the skip connection. The pseudo labels is constructed by fusing these SGC maps with decoder outputs on weakly augmented images, and then applying a standard sharpening with a temperature T. The fusing process includes normalizing the decoder output and SGC output so they are on the same scale, applying a softmax function, and then weighting them with a hyperparameter γ (and $1-\gamma$).

In this method, training requires 4 different loss components which are summed together without weights. A classification loss for the Grad-CAM output, an SGC loss, a standard decoder loss, and a consistency loss. Every loss is but with different variations. The classification loss is binary cross entropy, because we need multi-label classification, and the consistency loss is a soft cross entropy loss (between `log_softmax` and `softmax` of inputs). The inputs into the consistency loss is the predictions of the decoder on a strongly augmented image and the pseudo labels (Kaggle dataset). In addition, the decoder loss is implemented as standard cross entropy with inverse median class weights, and the SGC loss is implemented with Lovász loss [9] (Solafune dataset). The SGC loss is only meant to update the refinement module, which is achieved by a stopping gradient. We conduct hyperparameter tuning with 20% of the supervised dataset, and 50% of the weakly supervised dataset. See A.3 for further details.

3.2.2 Inference

To arrive at final predictions, we use exactly the same methods as in the original approach. We just need to initialize our model in "only-decoder mode", where it only outputs the decoder logits. Our best results came with test-time augmentation and `argmax` function.

3.3 Segment-Then-Classify

Segment-Then-Classify (STC) is a strategy for doing instance segmentation on remote sensing data [10]. The main idea is to first use a model which is trained to segment, then another to classify the segment. In the STC paper they use SAM [11] which at the time was considered state of the art. Additionally, they use a pre-trained Vision Transformer(ViT) from researchers at Google[12], which is tuned to classify objects in the NWPU VHR-10 dataset. The reason for using this strategy is that creating a model that can correctly perform instance segmentation when only spare data are available is very difficult. The Solafune competition [1] also had a rather sparse remote sensing dataset, and the results achieved in the STC paper seemed promising.

3.3.1 Implementation

This STC pipeline was not compatible with our baseline solution, due to our baseline model predicting single pixel values instead of whole segments. In order to implement a pipeline to similar to the one in the STC paper [10], we needed to create a new training dataset from the training images. The new dataset consists of images using only RGB bands, due to limitations of SAM [11]. They were cropped by the bounding boxes around every annotation from the training images in the Solafune dataset. Then resized to the 224×224 size that the ViT was originally trained on, using a transforming function4. However, the labels were simple integer representing the type of deforestation or background.
When training the ViT on the bounding boxes we started with a pre-trained version [13]. The model was then tuned on the images specified above. However, every image going into the Vit later was resized to 224×244. The model performed rather poorly compared to the other models. This was indicated early when tuning the ViT to the different deforestation classes. The ViT achieved an accuracy of ~30% after training for 20 epochs with 70% percent of the dataset. To try to improve the model it was trained with all of the available data, but it did not improve. The SAM model was not tuned, but used as specified in the STC paper. There was however not specified the parametric values for non-maximum suppression, hence the default values were used. See A.2 for details.

3.3.2 Inference

To make predictions with the STC strategy. The SAM model generates masks from the image with the mask generator, from then every bounding box is fed into the ViT which then predicts a classification. The segmentation is then marked as a the predicted class and returned as a polygon.

3.3.3 Challenges and Improvements

The STC pipeline did perform significantly worse than the baseline and the other methods. One challenge was the resizing of bounding boxes to fit to the ViT, which could have been problematic for very

small or large segments. As the skew in size could lead to "low" resolution when up-scaling and loss of pixels when downsizing. Another possible problem was that every bounding box potentially included much more than the current segment. With some of the segments in the training images being thin roads, this was likely a problem. There could in worst case be several segments in one bounding box, making it difficult for the model to properly learn the correct class. A way to mitigate this could be to remove all other pixels than the ones in the segment, both in training and predictions. This was not done since it was not done in the STC paper [10]. Further work on this section would mainly be to improve the classifier.

3.4 Segment anything 2 (SAM2)

Recent advancements in foundation models, particularly for image and video segmentation [14], motivate this section. In late July 2024, Meta released Segment Anything 2 (SAM2) [15] as an open source successor to the original SAM [11]. A key distinction is SAM2's extension to video segmentation, whereas SAM was confined to images. However, SAM2 also offers notable improvements in image segmentation performance. These gains are primarily driven by architectural refinements, specifically its improved image encoder and multiscale mask decoder, facilitating enhanced feature representation and more precise mask generation. This results in SAM2's strong zero-shot capabilities [15]. A key characteristic of SAM2 is the model's inherently promptable design; SAM2 generates segmentations interactively based on input prompts, such as points, bounding boxes, or approximate masks.

Building upon the initial segmentations generated by the SatlasPretrain model [2], we investigated using SAM2 [15] to improve these preliminary results. Our central hypothesis was that SAM2, when applied to the test set (`X_test`) and guided by prompts derived from the Satlas output (such as the masks that Satlas pretrain predicts), could produce more accurate segmentation boundaries for the identified deforestation drivers. The specific aim was to leverage SAM2's zero shot segmentation capabilities to refine the initial Satlas masks.

3.4.1 Implementation

To integrate SAM2 into our project, we utilized the official SAM2 codebase and pre-trained weights[1]. For weights, we loaded in the `SAM2.1 Hiera Large` checkpoint (`sam2.1_hiera_large.pt`) using the provided configuration files. We then instantiated the `SAM2ImagePredictor` class, which facilitates efficient inference by pre-computing image-embeddings [15].

[1]The SAM2 project repository needed to be cloned locally to access necessary model definitions and helper functions.

All refinement processing was performed locally on a MacBook Pro (M1 CPU). The inputs to the refinement stage were the evaluation dataset provided by Solafune [1]. Then, instead of using Satlaspretrains predictions as prompts to SAM2, we ended up using the predictions form the PsuedoSeg implementation, as this yielded better score 1. A helper function (`_prepare_image_for_sam`) ensured the input images matched the predictors expected HWC uint8 format.

Our *selective refinement strategy* involved iterating through each initial segmentation map (generated via PseudoSeg) and identifying components belonging to the target classes designated for refinement: 'plantation' (label 1), 'grassland/shrubland' (label 2), and 'mining' (label 3). We deliberately excluded 'logging' (label 4) from this process to preserve its fine structures identified by the preceding model (see Section 3.4.2). For each target class within an image, we first created a binary mask isolating pixels belonging only to that class. To identify individual connected component instances within this binary mask, we employed OpenCV's `cv2.connectedComponentsWithStats`. Finally, for each distinct instance found, a bounding box was calculated and stored alongside its corresponding class label. These bounding boxes served as the prompts for SAM2.

In the prediction loop, each image was processed individually. First, the evaluation images were loaded into the predictor the helper function `predictor.set_image()`. Subsequently, `predictor.predict()` was invoked with all the generated bounding boxes prompts for that image, using the setting `multimask_output=False` to obtain the single highest scoring mask predicted per prompt. This returned the refined masks and their associated predicted IoU scores. We filtered these predictions based on the predicted IoU score thresholds stored in `CLASS_SCORE_THRESHOLDS`8. The threshold for 'grassland/shrubland' was set significantly lower to accommodate the model's lower preference for 'grassland/shrubland'. Predictions with scores lower than their class thresholds were discarded.

The final segmentation map was reconstructed onto an initialized 1024×1024 zero array. For each non-logging prediction passing its class-specific IoU threshold, the corresponding SAM2 output mask was binarized (using `predictor.mask_threshold`) and used to assign the associated class label to the output map pixels. Crucially, after processing all filtered refinements, pixels identified as 'logging' (label 4) in the original PseudoSeg segmentation map were explicitly stamped onto the final map, ensuring preservation by overwriting any prior assignments in those locations. After processing each image, the predictor state was reset using `predictor.reset_predictor()`, and the resulting refined map was added to a collection. This complete

collection of maps was subsequently saved to disk for evaluation and analysis. Executing this entire refinement pipeline for all evaluation images took approximately 5 minutes on the specified hardware.

3.4.2 Discussion & Further work

A specific consideration during the SAM2 refinement process was the handling of the 'logging' class (label 4). Initial tests revealed that using bounding box prompts derived from the PseudoSeg output for these sparse, network-like features often resulted in significant over-segmentation by SAM2. While we also explored using the segment masks themselves as more precise prompts, this alternative did not produce confident predictions from the model for this particular class.

Considering these difficulties in achieving satisfactory refinement for 'logging' with SAM2, and observing that the initial PseudoSeg predictions for these features were already qualitatively quite detailed and useful (e.g., Figure A.1), we made the strategic decision to preserve them. Therefore, our final implementation focused SAM2 refinement—using box prompts filtered by class-specific IoU thresholds—exclusively on the 'plantation', 'grassland/shrubland', and 'mining' classes. The original 'logging' pixels from the PseudoSeg map were then explicitly merged into the final output, ensuring their preservation.

Future work could focus on several areas for potential improvement. Exploring more advanced prompting strategies beyond simple bounding boxes or single-instance masks, perhaps techniques sensitive to linear features, might enhance results. Further optimization of the class-specific score thresholds could also refine the balance between applying beneficial refinements and discarding poor ones. Lastly, domain-specific finetuning of SAM2 components, while resource-intensive, remains a promising avenue for boosting performance on this type of satellite imagery, though it was beyond the scope of this project.

4 Results

`"Sentinel_2_SwinB_SI_MS"` is the previously mentioned 9-band satlaspretrain model. After achieving a better performance strictly with RGB-channels, we decide to make `"Sentinel_2_SwinB_SI_RGB"` are only baseline model. Which is also ideal for the Kaggle dataset which only contains 4 bands. The PseudoSeg method improves our baseline method by 1.7%. This performance is around 5% below the competition winners [1]. Figure A.1 shows a comparison between these two models. It displays their predictions on an image from the test set, which means we do not actually have the true labels available. So, in theory, both of these models could have 0% performance in this example. However, we included this comparison to showcase a possible reason for the performance boost by PseudoSeg. Mining and logging were the most minor classes in the Solafune dataset, so increasing the exposure of these classes with the Kaggle dataset may have improved their performance. The STC pipeline did not improve over our baseline model. The low accuracy can possibly be attributed to the little amount of data and the lack of hyperparameter-tuning. The STC pipeline achieved an Iou-score of 31.63%, which was significantly worse than our baseline model. We decided to excluded it from the final as it would most likely worsen the results.

Table 1. Performance of our Models

Model	Results
`Sentinel_2_SwinB_SI_MS`	0.5634
`Sentinel_2_SwinB_SI_RGB` (baseline)	0.5921
`Sentinel_2_SwinB_SI_RGB` (+PseudoSeg)	0.6091
`Segment-Then-Classify_RGB`	0.3163
`Segment anything 2`	0.5536

5 Discussion

In this project we have managed to produce a model that can provide meaningful information in identifying deforestation drivers from a satellite image. Our original approach was very strong, producing a model very close to our final and best model. We found that `RGB` channels alone produced the best results, which was in accordance with the results in [2], and with the solution of the competition winners. The winners used an only `RGB` approach in order to fully take advantage of pre-trained foundation models. They also leverage more images, but uses synthetic data instead of trying to leverage a real dataset. Our real dataset did not drastically improve performance as much as the results in [5]. This may be because of the fact that the Kaggle dataset was not Sentinel-2 images, and therefore had a pixel size of around 3 meters [6], instead of the 10 meters in the Solafune dataset. We hypothesize that the varying pixel sizes is also one of the main reasons using all the bands together is ineffective.

References

[1] *Identifying Deforestation Drivers*. https://solafune.com/competitions/68ad4759-4686-4bb3-94b8-7063f755b43d?menu=about&tab=overview. Solafune Competition. 2025.

[2] F. Bastani, P. Wolters, R. Gupta, J. Ferdinando, and A. Kembhavi. *SatlasPretrain: A Large-Scale Dataset for Remote Sensing Image Understanding*. 2023. arXiv: 2211.15660 [cs.CV]. URL: https://arxiv.org/abs/2211.15660.

[3] D. John and C. Zhang. "An attention-based U-Net for detecting deforestation within satellite sensor imagery". In: *International Journal of Applied Earth Observation and Geoinformation* 107 (2022), p. 102685. ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2022.102685. URL: https://www.sciencedirect.com/science/article/pii/S0303243422000113.

[4] S. Bhat, M. Amit, M. Soni, and Y. Yasui. "Robust loss function for class imbalanced semantic segmentation and image classification". In: *IFAC-PapersOnLine* 56 (Jan. 2023), pp. 7934–7939. DOI: 10.1016/j.ifacol.2023.10.320.

[5] Y. Zou, Z. Zhang, H. Zhang, C.-L. Li, X. Bian, J.-B. Huang, and T. Pfister. "Pseudoseg: Designing pseudo labels for semantic segmentation". In: *arXiv preprint arXiv:2010.09713* (2020).

[6] B. Goldenberg, B. Uzkent, C. Clough, D. Funke, D. Desai, grischa, JesusMartinezManso, K. Scott, M. Risdal, M. Ryan, Pete, R. Holm, R. Nair, S. Herron, T. Stafford, and W. Kan. *Planet: Understanding the Amazon from Space*. https://kaggle.com/competitions/planet-understanding-the-amazon-from-space. Kaggle. 2017.

[7] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. "Hypercolumns for object segmentation and fine-grained localization". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 447–456. DOI: 10.1109/CVPR.2015.7298642.

[8] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 618–626. DOI: 10.1109/ICCV.2017.74.

[9] M. Berman and M. B. Blaschko. "Optimization of the Jaccard index for image segmentation with the Lovász hinge". In: *CoRR* abs/1705.08790 (2017). arXiv: 1705.08790. URL: http://arxiv.org/abs/1705.08790.

[10] Y. Hu, K. Caylor, and A. S. Boser. "Segment-then-Classify: Few-shot instance segmentation for environmental remote sensing". In: *NeurIPS 2023 Workshop on Tackling Climate Change with Machine Learning*. 2023. URL: https://www.climatechange.ai/papers/neurips2023/53.

[11] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. "Segment Anything". In: *arXiv:2304.02643* (2023).

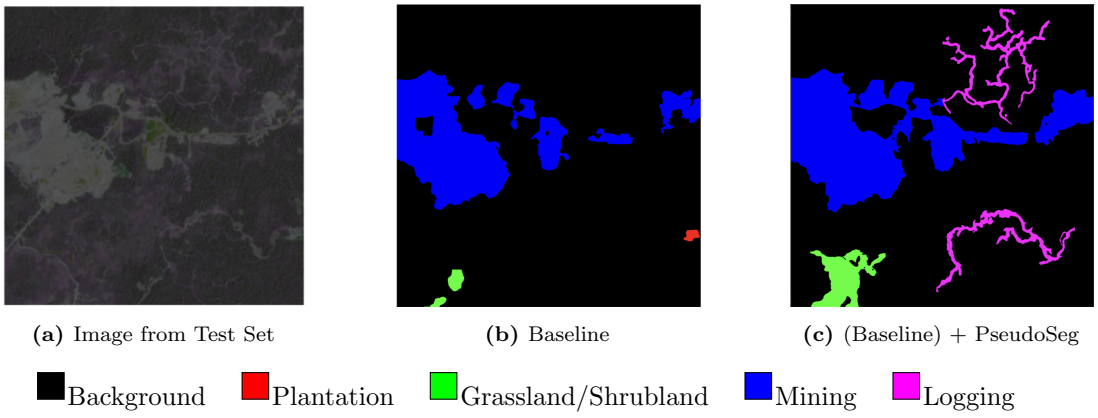
[12] X. Chen, C.-J. Hsieh, and B. Gong. "When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations". In: *arXiv preprint arXiv:2106.01548* (2021).

[13] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda. *Visual Transformers: Token-based Image Representation and Processing for Computer Vision*. 2020. arXiv: 2006.03677 [cs.CV].

[14] T. Zhou, W. Xia, F. Zhang, B. Chang, W. Wang, Y. Yuan, E. Konukoglu, and D. Cremers. *Image Segmentation in Foundation Model Era: A Survey*. 2024. arXiv: 2408.12957 [cs.CV]. URL: https://arxiv.org/abs/2408.12957.

[15] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer. "SAM 2: Segment Anything in Images and Videos". In: *arXiv preprint arXiv:2408.00714* (2024). URL: https://arxiv.org/abs/2408.00714.

A Appendix



(a) Image from Test Set **(b)** Baseline **(c)** (Baseline) + PseudoSeg

Figure A.1. Prediction of Two Different Models on "evaluation_50"

A.1 Original Approach

A.1.1 Hyperparameters and Augmentations

Below are the augmentations we apply during training. `A` is the library `albumentations`.

```
self.transform = A.Compose([
    A.OneOf([
        A.RandomRotate90(p=0.5), A.HorizontalFlip(p=0.5), A.VerticalFlip(p=0.5),
    ], p=1),
    A.OneOf([
        A.ColorJitter(  brightness=0.5, contrast=0.5,
                        saturation=0.5, hue=0.25, p=0.5
        ),
        A.ShiftScaleRotate( shift_limit=0.1, scale_limit=0.1,
                            rotate_limit=10, p=0.5
        ),
        A.RandomGamma(gamma_limit=(80,120), p=0.5
        ),
    ], p=1),
])
```

Listing 1. Augmentations

```
mean = 0.2 #mean of normal distribution for brightness reference
std = 0.05 #std of normal distribution for brightness reference
batch_size = 8
patch_size = 256
patch_stride = 128
num_epochs = 35
delta = 1.0 #penalty factor for robust loss
learning_rate = 0.0001
weight_decay = 0.01
optimizer = AdamW
lr_scheduler = ReduceLROnPlateau(
mode='min',factor=0.1,patience=5
)
```

Listing 2. Best Hyperparameters

A.2 STC

A.2.1 SAM hyperparameters

```
sam_path= "/content/drive/MyDrive/Colab_Notebooks/INF367A/sam_vit_h_4b8939.pth"
sam = sam_model_registry["default"](checkpoint=sam_path)
mask_generator = SamAutomaticMaskGenerator(sam,points_per_side=32,  # 32x32 grid
    = 1024 prompts
    min_mask_region_area=int(threshold_pixels),  # following threshold
    pred_iou_thresh=0.88,
    box_nms_thresh=0.7,
    stability_score_thresh=0.85,
    crop_n_layers=0,
    output_mode='binary_mask')
}
```

Listing 3. SAM implementation

```
image_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
])
```

Listing 4. ViT Resize

A.3 PseudoSeg

A.3.1 Kaggle Dataset

```
label_mapping = {
        'conventional_mine': 'mining',
        'artisinal_mine': 'mining',
        'agriculture': 'plantation',
        'cultivation': 'plantation',
        'selective_logging': 'logging',
        'slash_burn': 'grassland_shrubland'
    }
```

Listing 5. Label Mapping

Images without label `"clear"` are filtered out (no clouds). We map our relevant labels, and filter out images not containing one of these labels. But, to reduce class imbalance, we only keep 1000 images where `"plantation"` is the only label present.
Below is the weak on strong and weak augmentations used for consistency loss.

```
self.weak_transform = A.Compose([
    A.HorizontalFlip(p=1.0),
        ])

self.strong_transform = A.Compose([
    A.HorizontalFlip(p=1.0),
    A.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.25, p=1.0),

])
```

Listing 6. Strong and Weak Augmentations

A.3.2 Hyperparameters

Hyperparameters not defined here have the same value as in the original solution.

```
num_epochs = 50
T = 0.7 #temperature for sharpening
gamma = 0.5 #equal weight between SGC and decoder output
lr_scheduler = PolynomialLR(total_iters = 50, power=0.9)
```

Listing 7. Best Hyperparameters

A.4 SAM2

```
CLASS_SCORE_THRESHOLDS = {
    1: 0.80,  # Plantation
    2: 0.10,  # Grassland/Shrubland
    3: 0.80,  # Mining
}
```

Listing 8. Class score