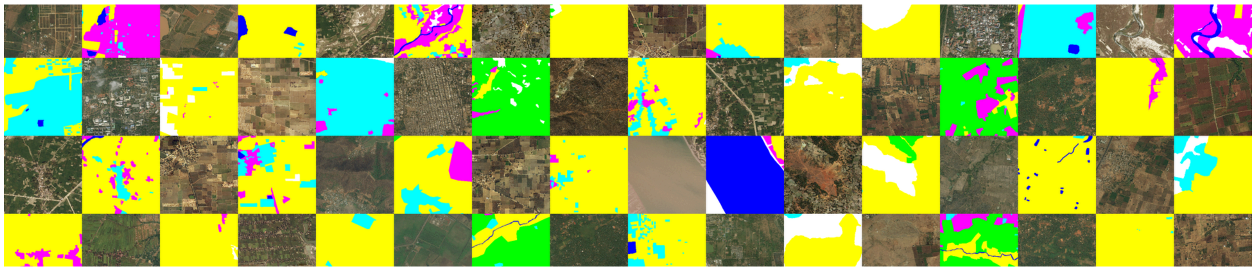
Deep Learning Assessment
Image Segmentation

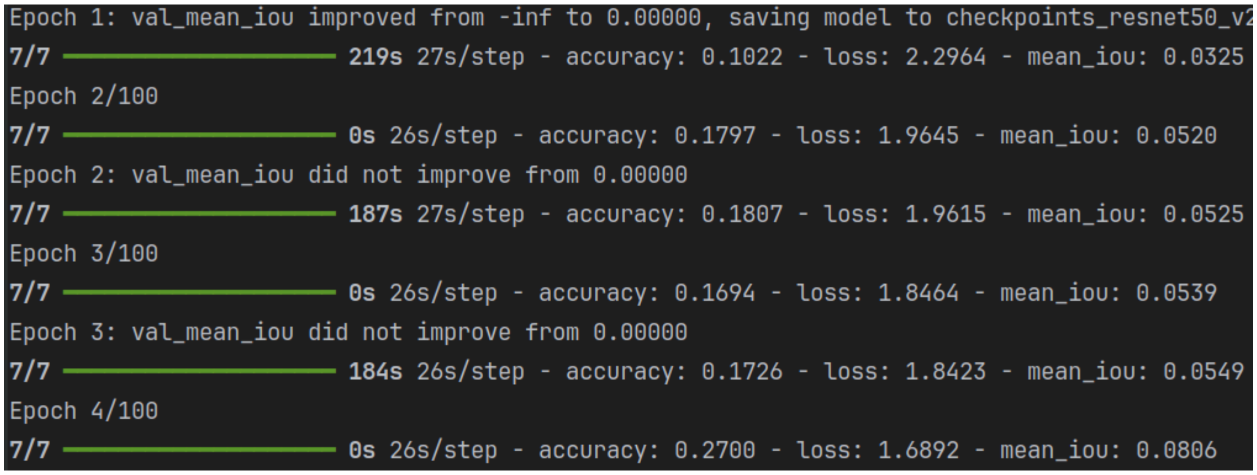


1. Task

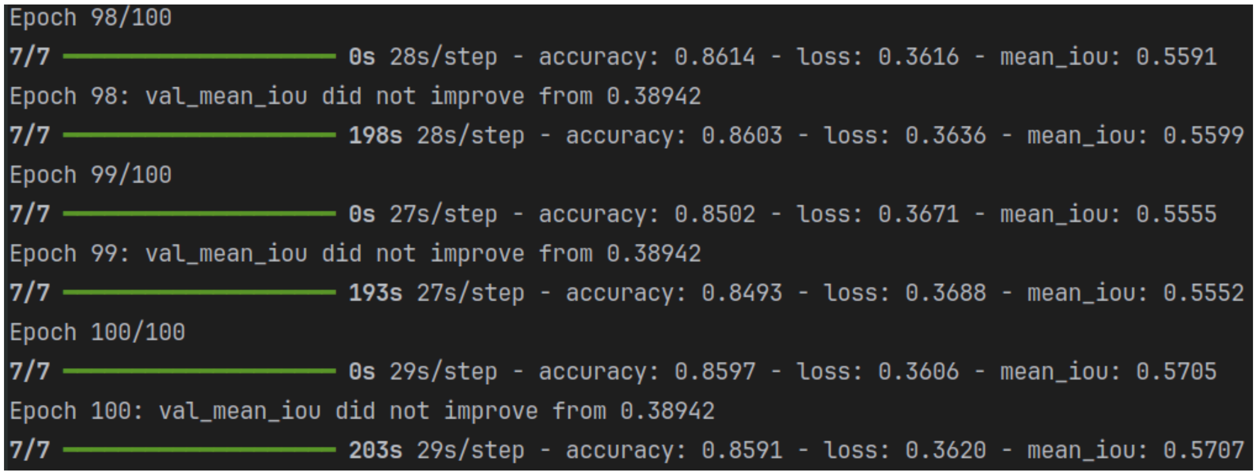
The task for this project is to pre-train a deep learning model on a segmentation dataset. Specifically, the segmentation task is semantic segmentation due to the specification of making use of unmarked pixels. In fact, image segmentation can be formulated as a classification problem of pixels with semantic labels (semantic segmentation) or partitioning of individual objects (instance segmentation). Semantic segmentation performs pixel-level class labeling with a set of object categories (for example, people, trees, sky, cars) for all image pixels (source).
In this report, we answer the question **"To what extent can our approach effectively predict and classify different land obstacles?"**. The outline of this assessment is as follows:
2/ Collect a remote sensing dataset for landvisors.
3/ Data preprocessing with histogram matching.
4/ Configure a pre-trained model.
5/ Create a custom loss function called partial cross-entropy.
6/ Train and validate the pre-trained model on the chosen dataset.
7/ Implement pseudo labeling on unlabeled data in the semi-supervised dataset.
8/ Implement ensemble learning for multi-model integration.
9/ Results, limitations and future work.

2. Data Collection

The requirement for the dataset is 2D data focused on semantic segmentation to obtain the masks for each image. We considered the following sources for a dataset: Kaggle, Roboflow, TensorHub and Hugging Face. The first dataset for our experiment is called "Land Cover Classification" (source). Due to lack of samples, our model did not perform well due to underfitting (validation loss is lower than training loss), as shown below.

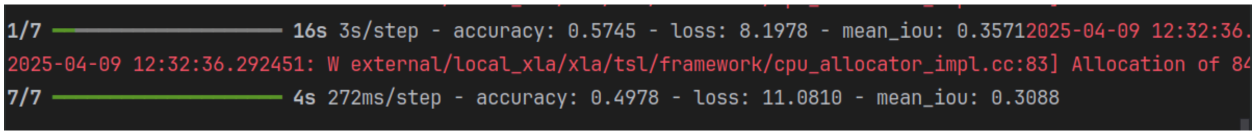


```
Epoch 1: val_mean_iou improved from -inf to 0.00000, saving model to checkpoints_resnet50_v2
7/7 ━━━━━━━━━━━━━━━━━━━━ 219s 27s/step - accuracy: 0.1022 - loss: 2.2964 - mean_iou: 0.0325
Epoch 2/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26s/step - accuracy: 0.1797 - loss: 1.9645 - mean_iou: 0.0520
Epoch 2: val_mean_iou did not improve from 0.00000
7/7 ━━━━━━━━━━━━━━━━━━━━ 187s 27s/step - accuracy: 0.1807 - loss: 1.9615 - mean_iou: 0.0525
Epoch 3/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26s/step - accuracy: 0.1694 - loss: 1.8464 - mean_iou: 0.0539
Epoch 3: val_mean_iou did not improve from 0.00000
7/7 ━━━━━━━━━━━━━━━━━━━━ 184s 26s/step - accuracy: 0.1726 - loss: 1.8423 - mean_iou: 0.0549
Epoch 4/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 26s/step - accuracy: 0.2700 - loss: 1.6892 - mean_iou: 0.0806
```



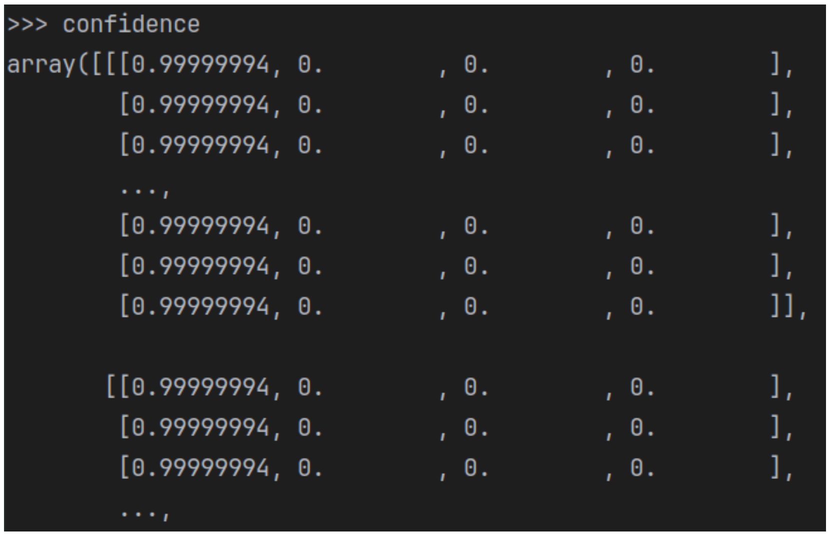
```
Epoch 98/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 28s/step - accuracy: 0.8614 - loss: 0.3616 - mean_iou: 0.5591
Epoch 98: val_mean_iou did not improve from 0.38942
7/7 ━━━━━━━━━━━━━━━━━━━━ 198s 28s/step - accuracy: 0.8603 - loss: 0.3636 - mean_iou: 0.5599
Epoch 99/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 27s/step - accuracy: 0.8502 - loss: 0.3671 - mean_iou: 0.5555
Epoch 99: val_mean_iou did not improve from 0.38942
7/7 ━━━━━━━━━━━━━━━━━━━━ 193s 27s/step - accuracy: 0.8493 - loss: 0.3688 - mean_iou: 0.5552
Epoch 100/100
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 29s/step - accuracy: 0.8597 - loss: 0.3606 - mean_iou: 0.5705
Epoch 100: val_mean_iou did not improve from 0.38942
7/7 ━━━━━━━━━━━━━━━━━━━━ 203s 29s/step - accuracy: 0.8591 - loss: 0.3620 - mean_iou: 0.5707
```

We tried a larger dataset with 2610 images (excluding masks) and 25 labels (source). The challenging part was defining RGB coordinates of each mask. We decided to change the dataset because of the poor results (1100% training loss).



```
1/7 ━━━━━━━━━━━━━━━━━━━━ 16s 3s/step - accuracy: 0.5745 - loss: 8.1978 - mean_iou: 0.35712025-04-09 12:32:36.
2025-04-09 12:32:36.292451: W external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 84
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 272ms/step - accuracy: 0.4978 - loss: 11.0810 - mean_iou: 0.3088
```

We also experimented with "Grass Dataset" (source), contains 2,480 samples and 3 labels. We faced the challenge of defining RGB coordinates due to the similarity in colors (all in black). It showed good loss results and low but increasing accuracy (from 0.75), however, the model was not able to distinguish between classes, suggesting class imbalance and we concluded that it was because of the mask color similarity. The figure below motivates our decision to change the dataset, given the fact that our model struggled with classification.



```
>>> confidence
array([[[0.99999994, 0.        , 0.        , 0.        ],
        [0.99999994, 0.        , 0.        , 0.        ],
        [0.99999994, 0.        , 0.        , 0.        ],
        ...,
        [0.99999994, 0.        , 0.        , 0.        ],
        [0.99999994, 0.        , 0.        , 0.        ],
        [0.99999994, 0.        , 0.        , 0.        ]],

       [[0.99999994, 0.        , 0.        , 0.        ],
        [0.99999994, 0.        , 0.        , 0.        ],
        [0.99999994, 0.        , 0.        , 0.        ],
        ...,
```

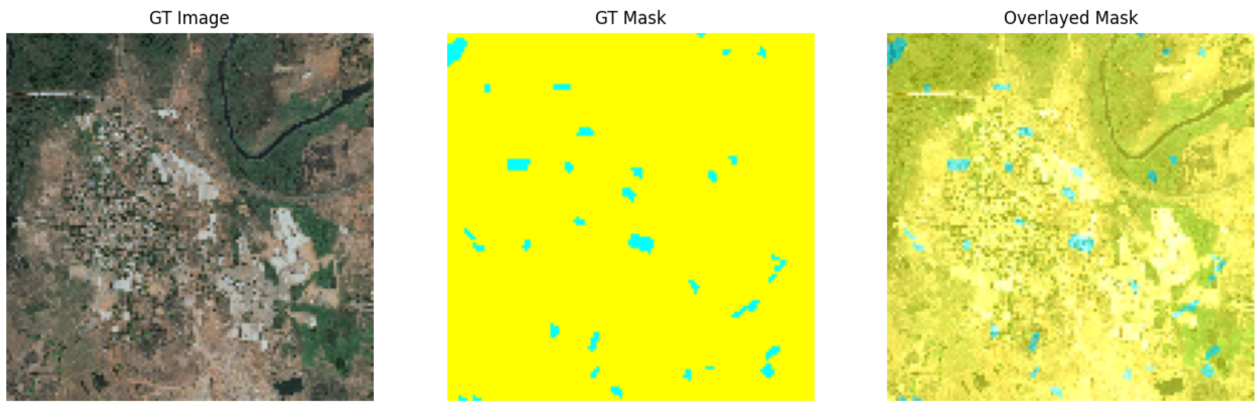
Current dataset

The current dataset, labeled “DeepGlobe Land Cover Classification Dataset” (source), This dataset contains 1606 samples with masks and 172 unlabeled samples.
Each satellite image is paired with a mask image for land cover annotation. The mask is a RGB image as follows:

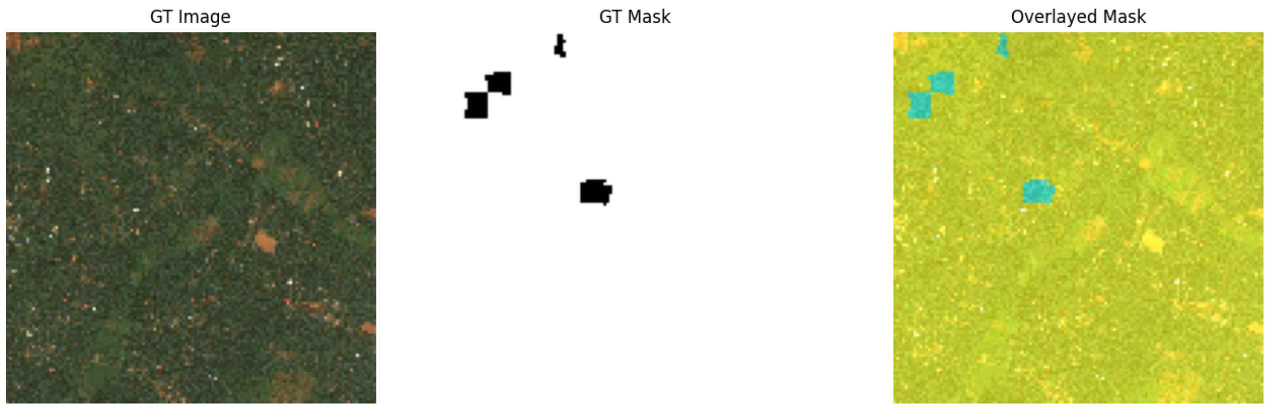
- Urban land: 0,255,255
- Agriculture land: 255,255,0
- Rangeland: 255,0,255
- Forest land: 0,255,0
- Water: 0,0,255
- Barren land: 255,255,255
- Unknown: 0,0,0

3. Data Preprocessing and Histogram Matching

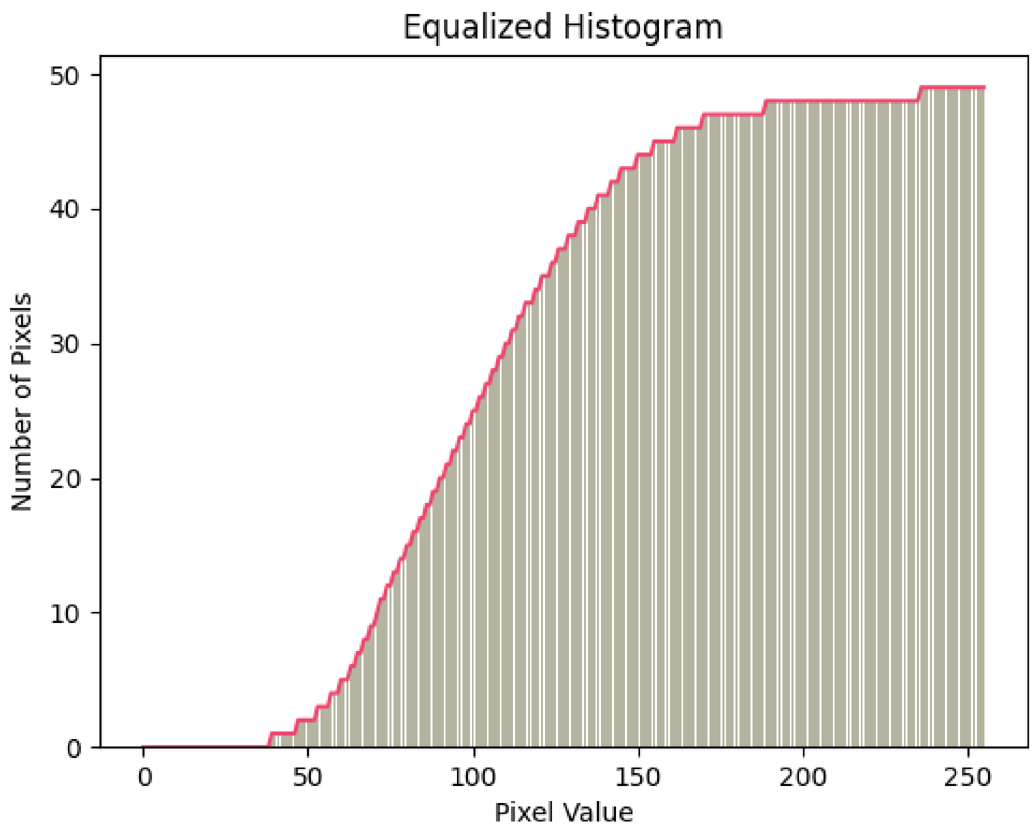
First, we start with system configuration (seeds and GPU acceleration), dataset configuration (batch size, image size and number of classes) and training configuration (model, epochs and learning rate).
Second, we split our dataset into training and validation set with a 7:3 ratio. Third, we load and unpack the images and masks. Fourth, we visually compare the original image and the image with the overlaying mask, as shown below.

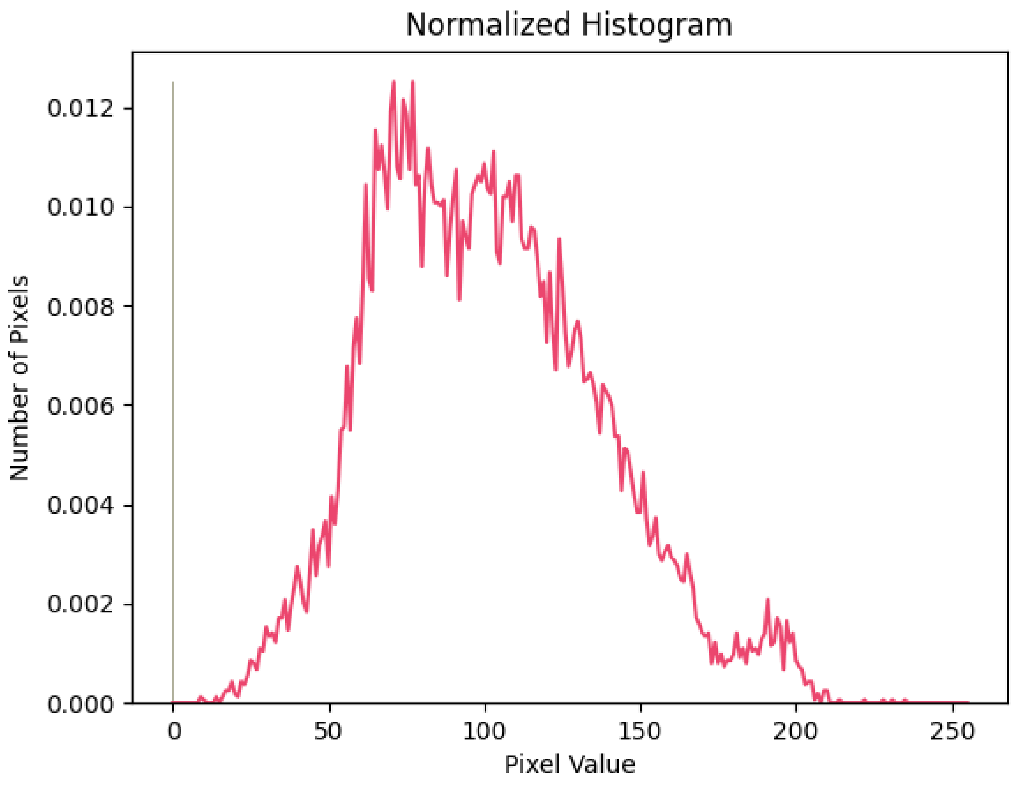


We apply data augmentation by implementing random flipping and random cropping in order to increase the robustness of the model.



We also calculated the equalized and the normalized histogram as shown below. The normalized histogram represents the probability distribution of pixel intensities and the equalized histogram enhances contrast by stretching the intensity distribution (source).


Equalized Histogram
Number of Pixels
Pixel Value
0
10
20
30
40
50
0
50
100
150
200
250


Normalized Histogram
Number of Pixels
Pixel Value
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0
50
100
150
200
250

4. Configure the Pre-trained Model

In the first iteration, we considered and implemented the Mask_R-CNN model for its large size. However, this pre-trained model is exclusively for object detection and instance segmentation, thus, we looked for a more appropriate model.

Current Pre-trained Model

The current pre-trained model is **DeepLab-v3** with **ResNet50** as the backbone. A backbone model is a base network architecture that handles the initial feature extraction (source). ResNet50 is a 50-layer deep convolutional neural network designed for image classification. It's typically pre-trained on large datasets like ImageNet, where it learns general visual features like edges, textures, and patterns (source). On the other hand, DeepLab-v3 is a semantic segmentation architecture that takes these extracted features and processes them further to produce pixel-wise classifications (source).

5. Partial Cross-Entropy

In this project, we use partial cross entropy loss. **Partial Cross Entropy (pCE)** differs from standard CE by applying a threshold to determine which images are considered for the loss computation. There are not many pCE implementations, although we noticed that the loss formula includes **focal loss**, a loss that modifies the standard cross-entropy loss by introducing a modulating factor that down-weights the loss assigned to well-classified examples (AI-generated response due to lack of sources on pCE).
We implemented our custom loss function by calculating the focal loss (focal weight * cross entropy) then pCE by normalizing the focal loss with respect to the mask, as shown in the pCE loss formula.

6. Training and Validation

Before starting the training, we define an accuracy metric called mean IoU. Intersection over Union (IoU) is a metric often used in segmentation problems to assess the model's accuracy. It provides a more intuitive basis for accuracy that is not biased by the imbalanced percentage of pixels from any particular class (source).
We also define a second loss function commonly used for segmentation problems, called Dice. The Dice loss originates from the Dice Coefficient, which is a measure of similarity between two sets of data. It is commonly used to evaluate the overlap between a predicted segmentation mask and the target segmentation mask (source).

Moreover, we start TensorBoard to visualize the model lifecycle as well as saving the weights. The training set contains 699 samples, 299 validation samples and 608 testing samples. We start the training and validation by compiling the model and fitting our data.
We evaluate our model according to the mean IoU per class in order to check the performance of the model according to each class and discover potential signs of data imbalance.

7. Pseudo Labeling (semi-supervised learning)

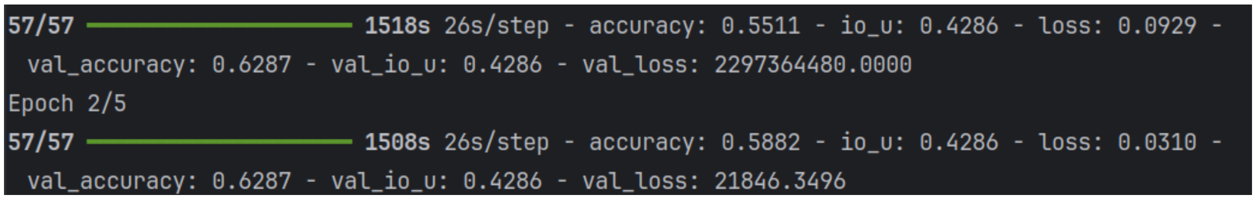
We perform a semi-supervised technique for labeling the unlabeled portion of the dataset, called pseudo labeling. In short, pseudo labels are target classes for unlabeled data as if they were true labels (source). This technique serves for label prediction when generating unlabeled synthetic samples as part of data augmentation.
After evaluating the model, we use the validation set to predict pseudo labels with a confidence threshold.

8. Ensemble Learning

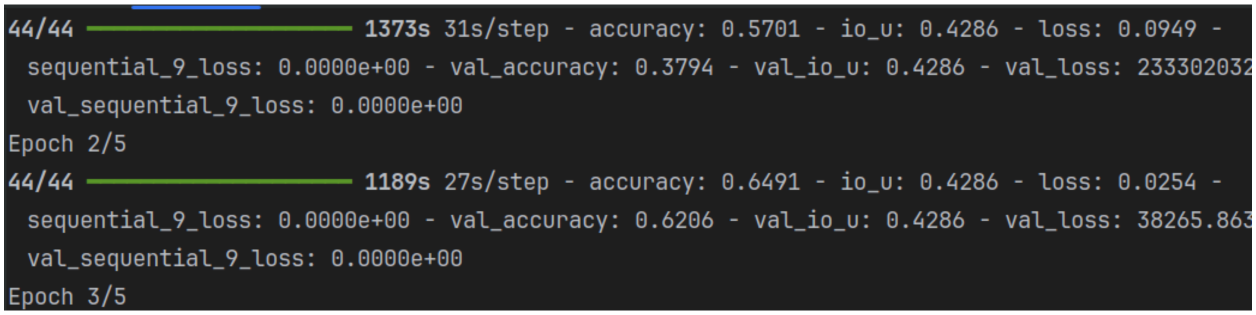
We use the testing set to predict classes, while amplifying this process with ensemble learning to combine predictions of multiple models.
For now, we only included the DeepLabv3 pre-trained model, however, another model can be added in the list "models".

9. Results Interpretation and Limitations

The figure below shows that our model initially struggled with classifying classes with a high validation loss, suggesting overfitting.

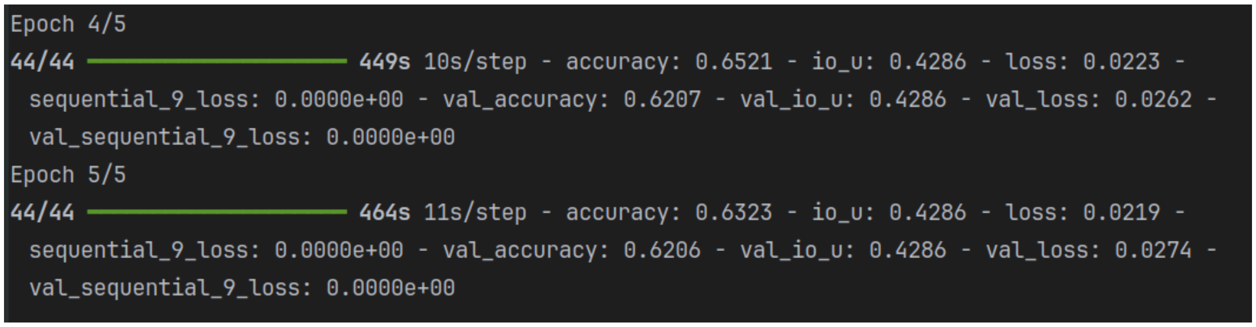


```
57/57 ━━━━━━━━━━━━━━━━━━━━ 1518s 26s/step - accuracy: 0.5511 - io_u: 0.4286 - loss: 0.0929 -
 val_accuracy: 0.6287 - val_io_u: 0.4286 - val_loss: 2297364480.0000
Epoch 2/5
57/57 ━━━━━━━━━━━━━━━━━━━━ 1508s 26s/step - accuracy: 0.5882 - io_u: 0.4286 - loss: 0.0310 -
 val_accuracy: 0.6287 - val_io_u: 0.4286 - val_loss: 21846.3496
```



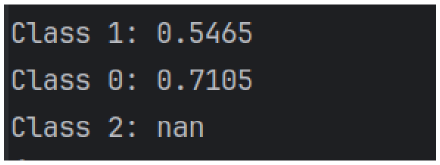
```
44/44 ━━━━━━━━━━━━━━━━━━━━ 1373s 31s/step - accuracy: 0.5701 - io_u: 0.4286 - loss: 0.0949 -
 sequential_9_loss: 0.0000e+00 - val_accuracy: 0.3794 - val_io_u: 0.4286 - val_loss: 233302032
 val_sequential_9_loss: 0.0000e+00
Epoch 2/5
44/44 ━━━━━━━━━━━━━━━━━━━━ 1189s 27s/step - accuracy: 0.6491 - io_u: 0.4286 - loss: 0.0254 -
 sequential_9_loss: 0.0000e+00 - val_accuracy: 0.6206 - val_io_u: 0.4286 - val_loss: 38265.863
 val_sequential_9_loss: 0.0000e+00
Epoch 3/5
```

We successfully addressed overfitting when we froze ResNet50 layers, as well as increasing validation samples from 10% of the training data to 30% (as described). As shown below, the validation loss is approximately equal to the training loss. However, our mean IoU is still low.

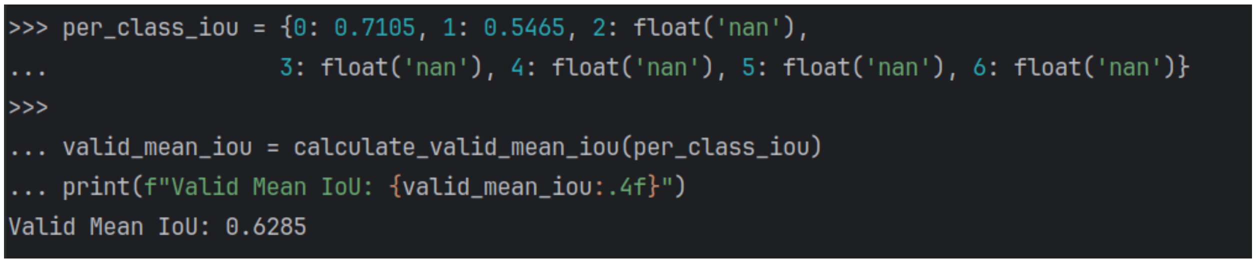


```
Epoch 4/5
44/44 ━━━━━━━━━━━━━━━━━━━━ 449s 10s/step - accuracy: 0.6521 - io_u: 0.4286 - loss: 0.0223 -
 sequential_9_loss: 0.0000e+00 - val_accuracy: 0.6207 - val_io_u: 0.4286 - val_loss: 0.0262 -
 val_sequential_9_loss: 0.0000e+00
Epoch 5/5
44/44 ━━━━━━━━━━━━━━━━━━━━ 464s 11s/step - accuracy: 0.6323 - io_u: 0.4286 - loss: 0.0219 -
 sequential_9_loss: 0.0000e+00 - val_accuracy: 0.6206 - val_io_u: 0.4286 - val_loss: 0.0274 -
 val_sequential_9_loss: 0.0000e+00
```

We discovered that it was due to lack of samples from class 2 to 6 (starting at 0). Thus we calculated the true validation mean IoU if it would ignore the nan values.

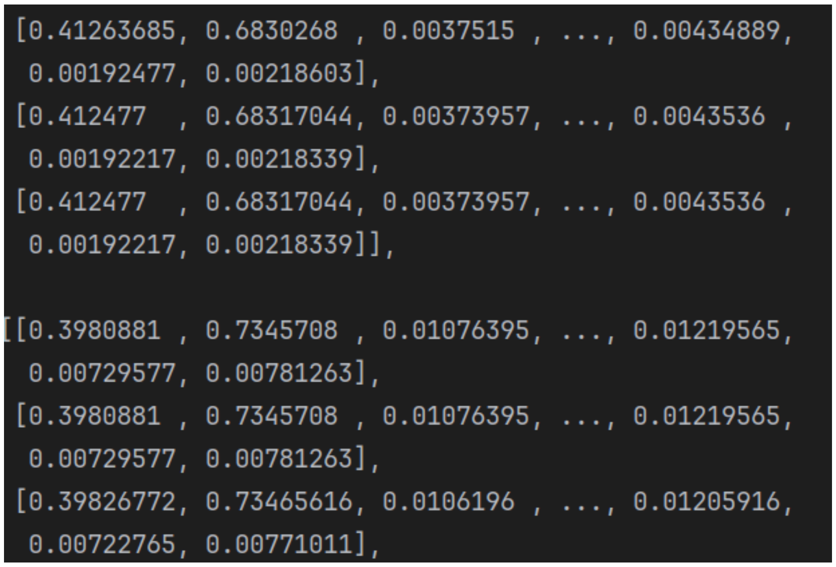


```
Class 1: 0.5465
Class 0: 0.7105
Class 2: nan
```



```
>>> per_class_iou = {0: 0.7105, 1: 0.5465, 2: float('nan'),
...                  3: float('nan'), 4: float('nan'), 5: float('nan'), 6: float('nan')}
>>>
... valid_mean_iou = calculate_valid_mean_iou(per_class_iou)
... print(f"Valid Mean IoU: {valid_mean_iou:.4f}")
Valid Mean IoU: 0.6285
```

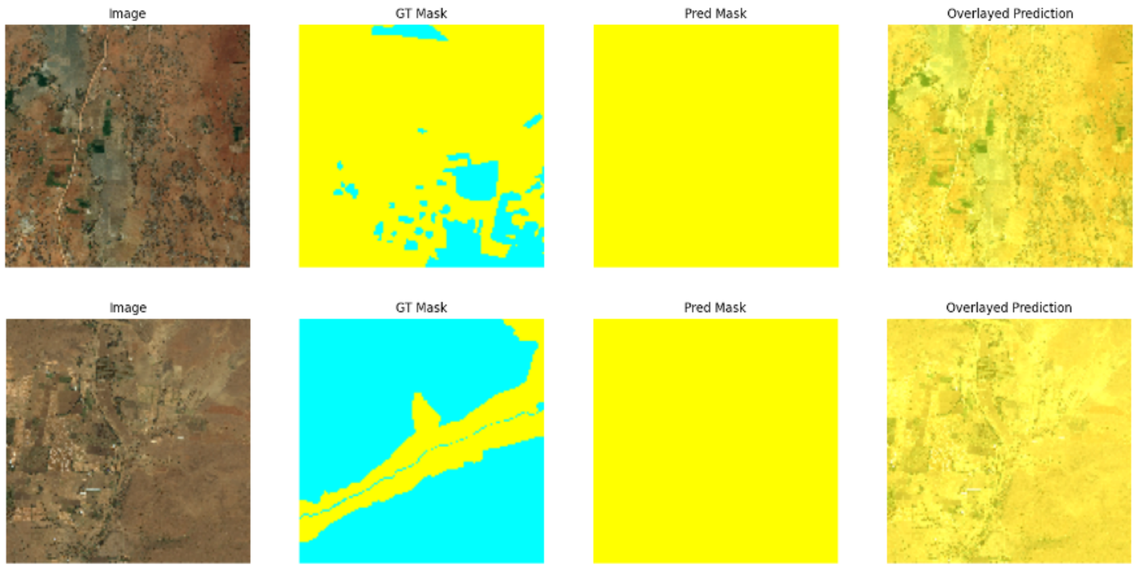
The confidence in predicting labels ranges from 68% to 75%, therefore, it needs more improvement and diversity in classes.



```
[0.41263685, 0.6830268 , 0.0037515 , ..., 0.00434889,
 0.00192477, 0.00218603],
[0.412477  , 0.68317044, 0.00373957, ..., 0.0043536 ,
 0.00192217, 0.00218339],
[0.412477  , 0.68317044, 0.00373957, ..., 0.0043536 ,
 0.00192217, 0.00218339]],

[[0.3980881 , 0.7345708 , 0.01076395, ..., 0.01219565,
 0.00729577, 0.00781263],
[0.3980881 , 0.7345708 , 0.01076395, ..., 0.01219565,
 0.00729577, 0.00781263],
[0.39826772, 0.73465616, 0.0106196 , ..., 0.01205916,
 0.00722765, 0.00771011],
```

The figures below represent the results of our model prediction on the test set. As shown, our model successfully predicted the yellow areas representing agricultural areas, however, it was not as successful with the light blue areas representing urban areas.



Conclusion:

Our approach shows promise in semantic segmentation; however, minority classes were not successfully predicted due to data imbalance and other factors. For future work, we will generate synthetic samples using diffusion models, such as Stable Diffusion.