Submitted by
Group E8

Prepared at
Institute of Computer Graphics

Evaluator
Dr. Mohammed Abbass,
Dr. Oliver Bimber

January 2024

PROJECT REPORT

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
jku.at
DVR 0093696

Table of Contents

1. Motivation

This scientific project focused on the challenge of identifying individuals in thermal drone aerial images, particularly in densely forested areas. To address this issue, machine learning models were tested with the aim of learning to remove foliage and highlight individuals in leafy forests. This approach is intended to facilitate the localization of individuals for rescue operations. The following chapter will provide a more detailed explanation of the initial considerations made in the first phase of the research.

2. Research / Model Selection

The selection of models was based on a literature review, leading to the decision to proceed with U-Net, Vision Transformer, and DeepLab. These architectures were chosen due to their promising performance in prior scientific works addressing similar challenges. To facilitate an efficient interchangeability of these architectures, they were implemented using PyTorch. Due to time constraints and the early departure of David Fartacek from the group, we decided not to implement Vision Transformer. Instead, we focused on optimizing the parameterization of U-Net and DeepLab efficiently.

3. Data Pipeline

A data pipeline was developed to filter and preprocess samples in the provided dataset. This pipeline undergoes several steps to ensure efficient data preparation:
Initially, the data within the pipeline is filtered to remove incomplete or corrupted images. The remaining data is assigned a continuous, ascending integer index to allow efficient random access through the data loader. After the sequences have undergone this preparatory process, integral images are generated, and the input images discarded to conserve disk space.

Because of the time and resource-intensive nature of training, the decision was made to generate only 10 AOS integrals per sample, and to reuse some subset of those during later hyperparameter experimentation.

Finally, images were transformed using mirroring and rotation at inference time, increasing the effective size of the training dataset by a factor of 8 without consuming more storage space.

4. Models

4.1. U-Net

U-Net[1] is a convolutional neural network architecture designed for semantic segmentation, particularly in biomedical imaging. It features an encoder-decoder structure with skip connections between corresponding layers. The encoder captures contextual information and reduces spatial resolution, while the decoder upsamples features and retains fine details through skip connections. This architecture was modified to produce pixel outputs directly (rather than the categorical outputs normally used with this network type. Due to time and resource constraints, we set this approach aside to focus on DeepLab.

[1] https://arxiv.org/abs/1505.04597

4.2. DeepLab

DeepLab[2] is a family of deep learning models designed for semantic image segmentation. It uses a pretrained backbone network, such as ResNet, and incorporates features like atrous convolutions, spatial pyramid pooling, and ASPP to capture multi-scale contextual information. DeepLab predicts pixel-wise segmentation maps, assigning class labels to each pixel.

The provided code snippet modifies a DeepLabv3 model, adjusting the input channels for our image generation task with an input stack of 10 AOS integrals. Crucially, the output layer is modified from producing categorical outputs to producing pixel values directly.

Initially, the MobilenetV3 backbone was chosen for our model training. Further experimentation showed higher quality results and improved validation metrics using the larger Resnet101 backbone. Using this larger backbone slowed training by a factor of 10 and limited the team's capacity to run hyperparameter experiments.

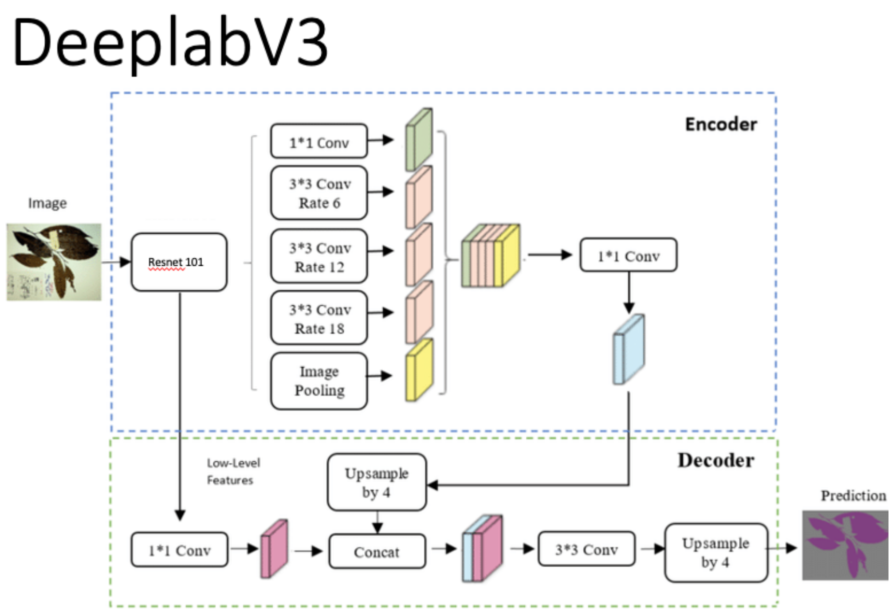
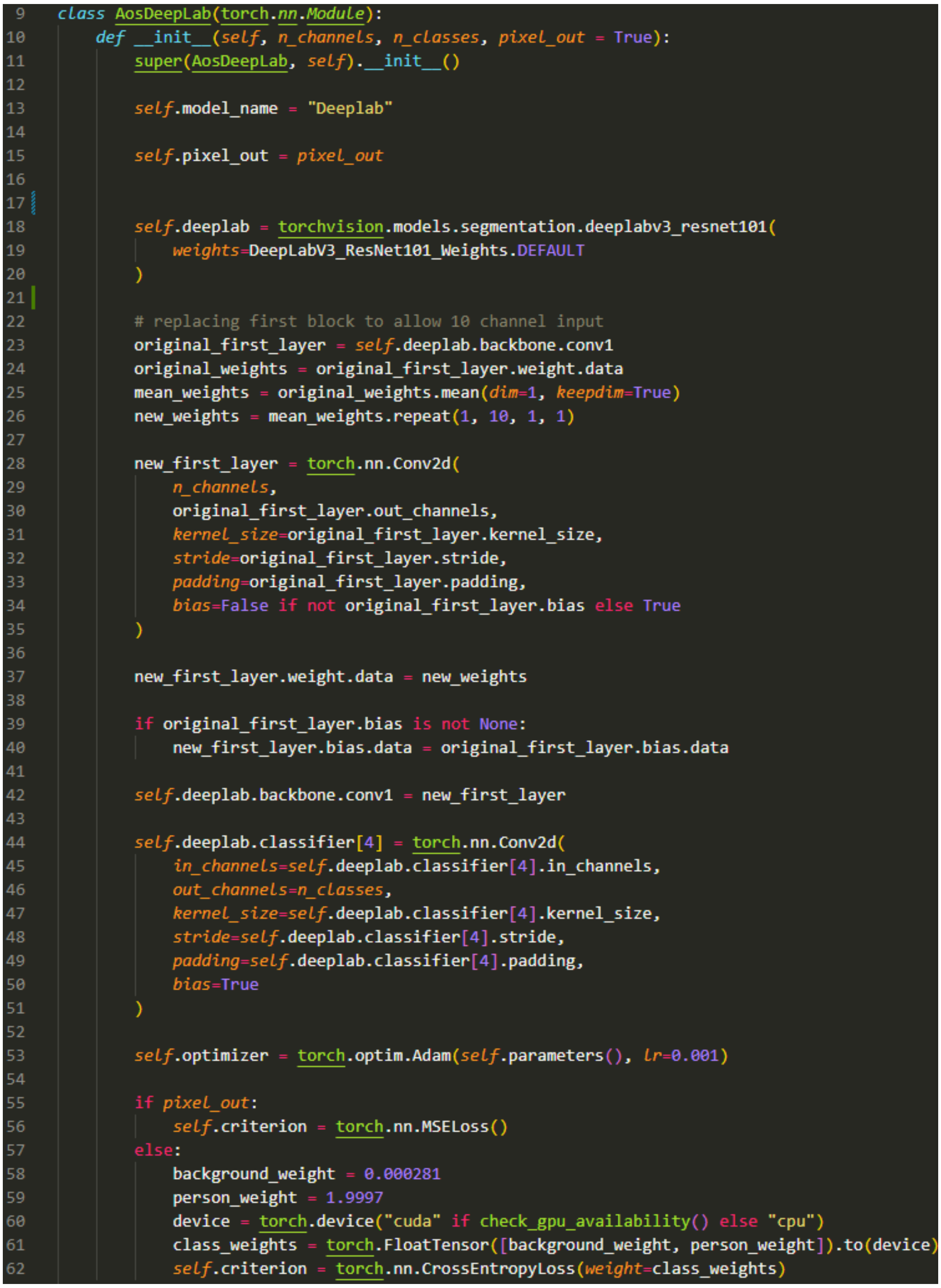


Figure 1: Model-Architecture

[2] https://arxiv.org/abs/1606.00915



```python
 9  class AosDeepLab(torch.nn.Module):
10      def __init__(self, n_channels, n_classes, pixel_out = True):
11          super(AosDeepLab, self).__init__()
12  
13          self.model_name = "Deeplab"
14  
15          self.pixel_out = pixel_out
16  
17  
18          self.deeplab = torchvision.models.segmentation.deeplabv3_resnet101(
19              weights=DeepLabV3_ResNet101_Weights.DEFAULT
20          )
21  
22          # replacing first block to allow 10 channel input
23          original_first_layer = self.deeplab.backbone.conv1
24          original_weights = original_first_layer.weight.data
25          mean_weights = original_weights.mean(dim=1, keepdim=True)
26          new_weights = mean_weights.repeat(1, 10, 1, 1)
27  
28          new_first_layer = torch.nn.Conv2d(
29              n_channels,
30              original_first_layer.out_channels,
31              kernel_size=original_first_layer.kernel_size,
32              stride=original_first_layer.stride,
33              padding=original_first_layer.padding,
34              bias=False if not original_first_layer.bias else True
35          )
36  
37          new_first_layer.weight.data = new_weights
38  
39          if original_first_layer.bias is not None:
40              new_first_layer.bias.data = original_first_layer.bias.data
41  
42          self.deeplab.backbone.conv1 = new_first_layer
43  
44          self.deeplab.classifier[4] = torch.nn.Conv2d(
45              in_channels=self.deeplab.classifier[4].in_channels,
46              out_channels=n_classes,
47              kernel_size=self.deeplab.classifier[4].kernel_size,
48              stride=self.deeplab.classifier[4].stride,
49              padding=self.deeplab.classifier[4].padding,
50              bias=True
51          )
52  
53          self.optimizer = torch.optim.Adam(self.parameters(), lr=0.001)
54  
55          if pixel_out:
56              self.criterion = torch.nn.MSELoss()
57          else:
58              background_weight = 0.000281
59              person_weight = 1.9997
60              device = torch.device("cuda" if check_gpu_availability() else "cpu")
61              class_weights = torch.FloatTensor([background_weight, person_weight]).to(device)
62              self.criterion = torch.nn.CrossEntropyLoss(weight=class_weights)
```

Figure 2: Code of DeepLab Model

Line 23-50:
In this code, modifications are made to the DeepLabv3 model with a ResNet-101 backbone. The first convolutional layer of the model's backbone is adjusted to have 10 input channels instead of

the original number of 3 channels (RGB input). This implies that the input provided to the model should have 10 channels.
Additionally, the number of output classes (num_classes) is modified. This modification is applied to the output layer of the model, ensuring that it has the correct number of output channels. For the initial pixel-classification model we set num_classes to 2, one class for the background and another one for the person classification. After adapting our model to output an image instead of doing classification, this parameter has been set to 1.
As an experiment to improve model convergence, we tried to calculate the mean of the original pretrained weights from the first Conv2d layer (3 channels) to all 10 newly generate input channels. This showed an improvement in validation loss in the first 5 epochs of training, but the effect is less impactful when training over longer periods.

4.3. Training and Evaluation of DeepLab

The following reconstruction quality metrics and loss functions were used during training and evaluation of the models.

4.3.1 MSE (Mean Squared Error)

MSE Loss measures the average squared difference between the pixel values of the predicted image and the target image. It was used as the primary minimization objective during training, used with the ADAM optimizer with learning rate 1e-4.

4.3.2 PSNR (Peak-Signal-to-Noise Ratio)

PSNR metric evaluates the quality of images by calculating the ratio between the maximum possible signal and the noise. Higher PSNR values suggest a lower perceived difference between the original image and the reconstructed version. This metric is commonly used in image processing.

4.3.3 SSIM (Structural Similarity Index)

SSIM measures the structural similarity between two images, considering both structural information and brightness. SSIM values range from -1 to 1, with 1 indicating perfect similarity. Higher SSIM values signify greater similarity between the images, making it a useful metric for assessing image quality.

4.4. Training metrics DeepLab

4.4.1 Trained models

In the process of this project's duration, we trained 9 different Deeplab models, trying different strategies, a short overview is given here (The experiment name in brackets is the same as in all tensorboard plots shown below):
Model 1 (deeplab_training_42_epochs): Initial model, pixel-classification, mobilenetv3 backbone.
Model 2 (deeplab_training_125_epochs): Copying mean of pretrained first layer weights over all 10 input channels. Faster convergence observed.

Model 3 (deeplab_training_200_epochs): Short experiment with higher learning rate. Result became worse than previously.
Model 4 (deeplab_training_111_epochs): Final pixel-classification model, learning rate scheduler implemented. This model has been presented in intermediate results presentation.
Model 5 (deeplab_training_100_epochs): First model with adapted architecture for image restoration. Results were very bad; model didn’t improve anymore after 4 epochs.
Model 6 (deeplab_training_50_epochs): Architecture change to resnet101 backbone. Model learning improved and results on test set looked promising.
Model 7 (deeplab_training_75_epochs): Test on the same architecture as before but with a sub-set of the training set of only 50% of the original images. This test has been done to see if we can train faster on a smaller training set with similar results. Conclusion: full dataset is preferable.
Model 8 (deeplab_training_20_epochs): Test with a lower learning rate of 0.0001. Model convergence was too slow.
Model 9 (deeplab_training_750_epochs): Final model that used the best training hyperparameters obtained from previous training runs. After initial training we fine-tuned the model with a lower learning rate (0.0001) on a sub-set of the training data that is most similar to the real integral images provided.

4.4.2 Loss

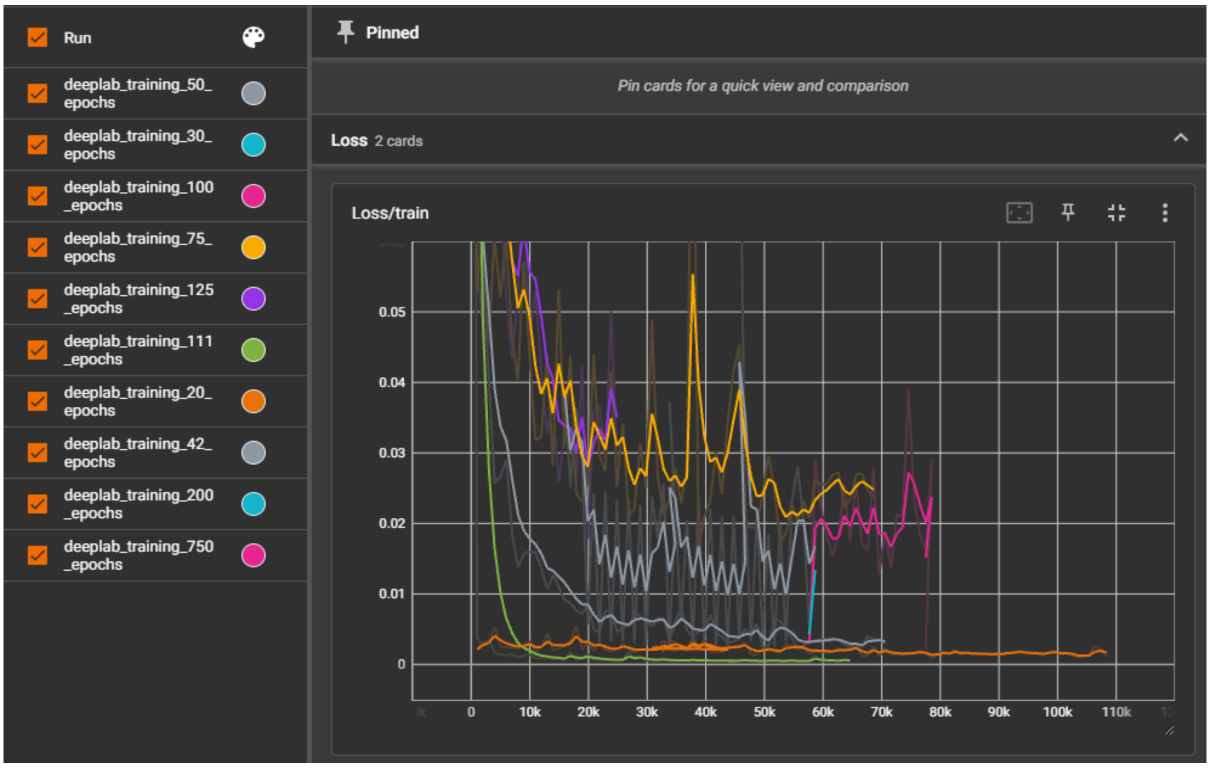


Figure 3: Loss/Train Plot

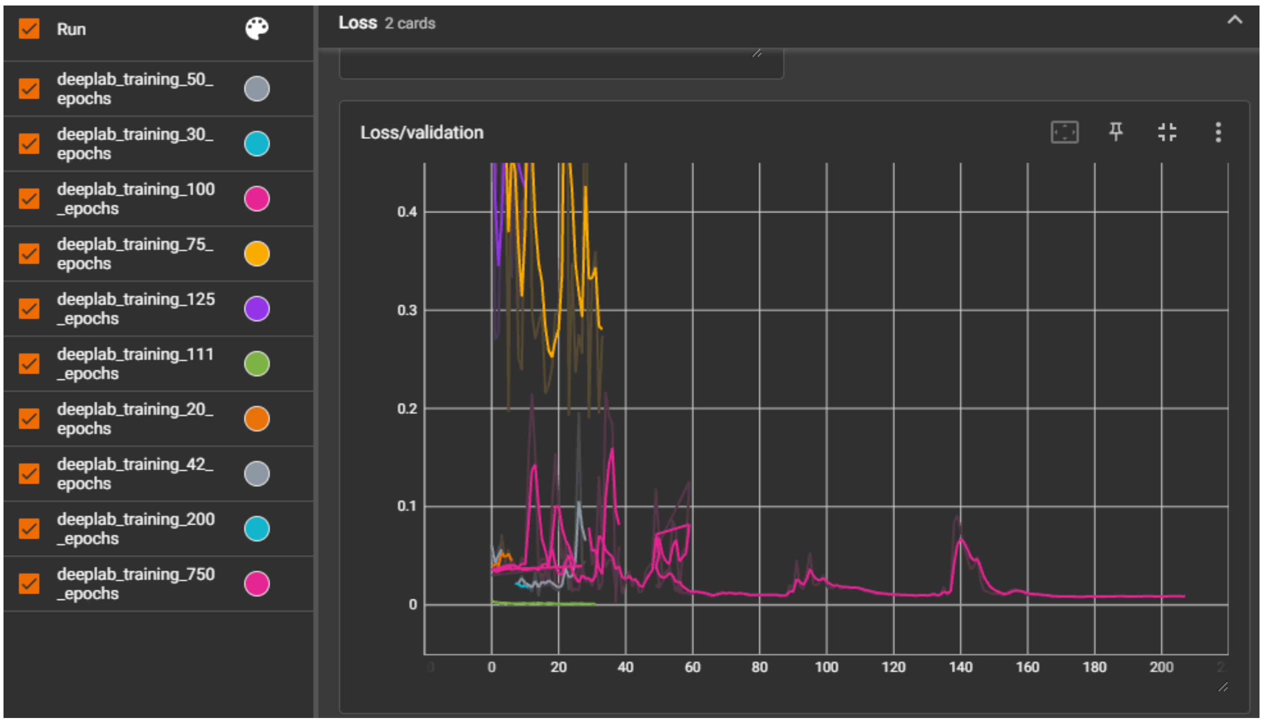
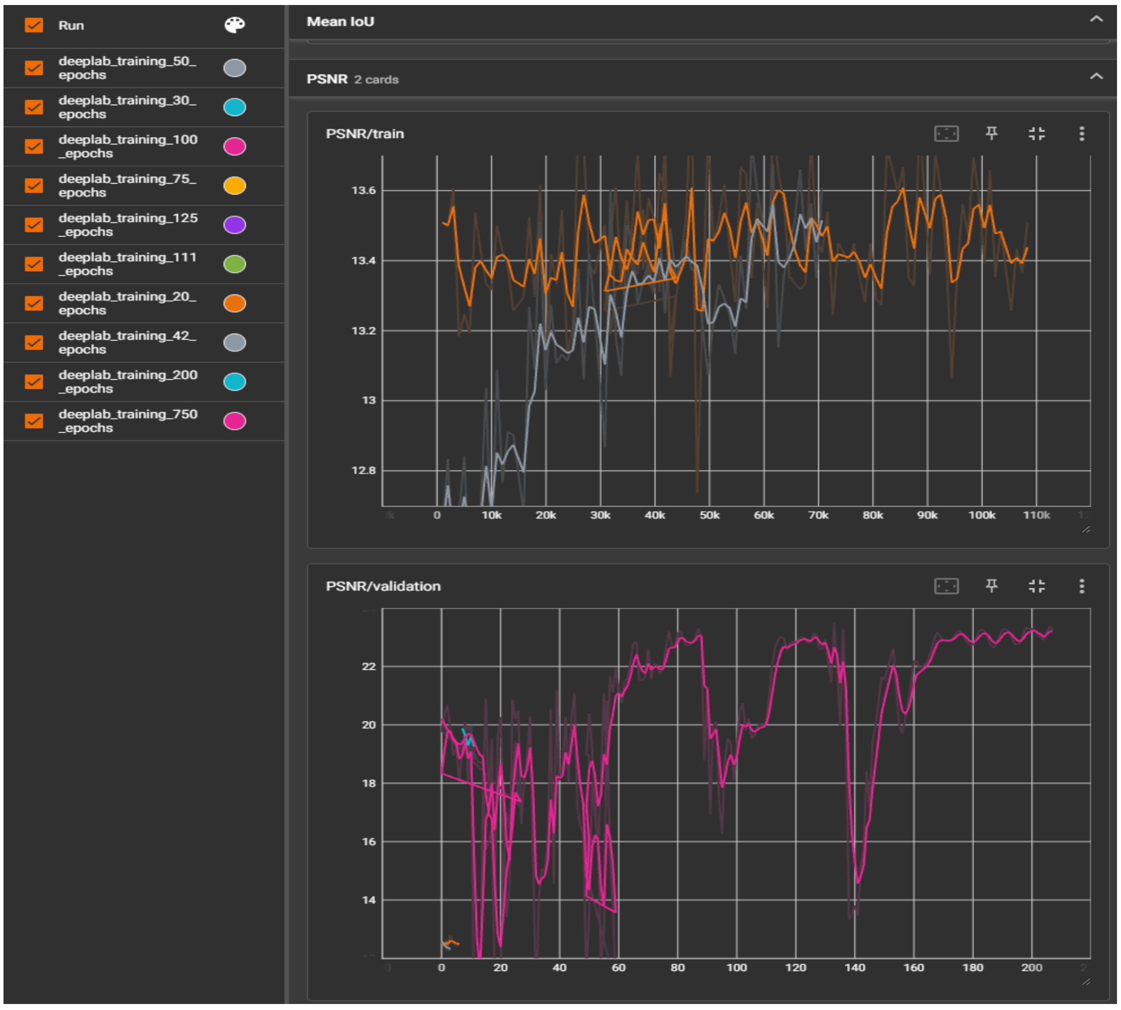


Figure 4: Loss/Validation Plot

4.4.2 PSNR

The graphs below show the Peak Signal to Noise Ratio metric. Only the last few trainings provide this information as this metric has been implemented at a later stage.



Run
deeplab_training_50_epochs
deeplab_training_30_epochs
deeplab_training_100_epochs
deeplab_training_75_epochs
deeplab_training_125_epochs
deeplab_training_111_epochs
deeplab_training_20_epochs
deeplab_training_42_epochs
deeplab_training_200_epochs
deeplab_training_750_epochs
Mean IoU
PSNR 2 cards
PSNR/train
13.6
13.4
13.2
13
12.8
0
10k
20k
30k
40k
50k
60k
70k
80k
90k
100k
110k
PSNR/validation
22
20
18
16
14
0
20
40
60
80
100
120
140
160
180
200

Figure 5: PSNR Plot

4.4.2 SSIM

The graphs below show the Structural Similarity Index metric. Only the last training provides this information as the metric has been implemented last in our training pipeline.

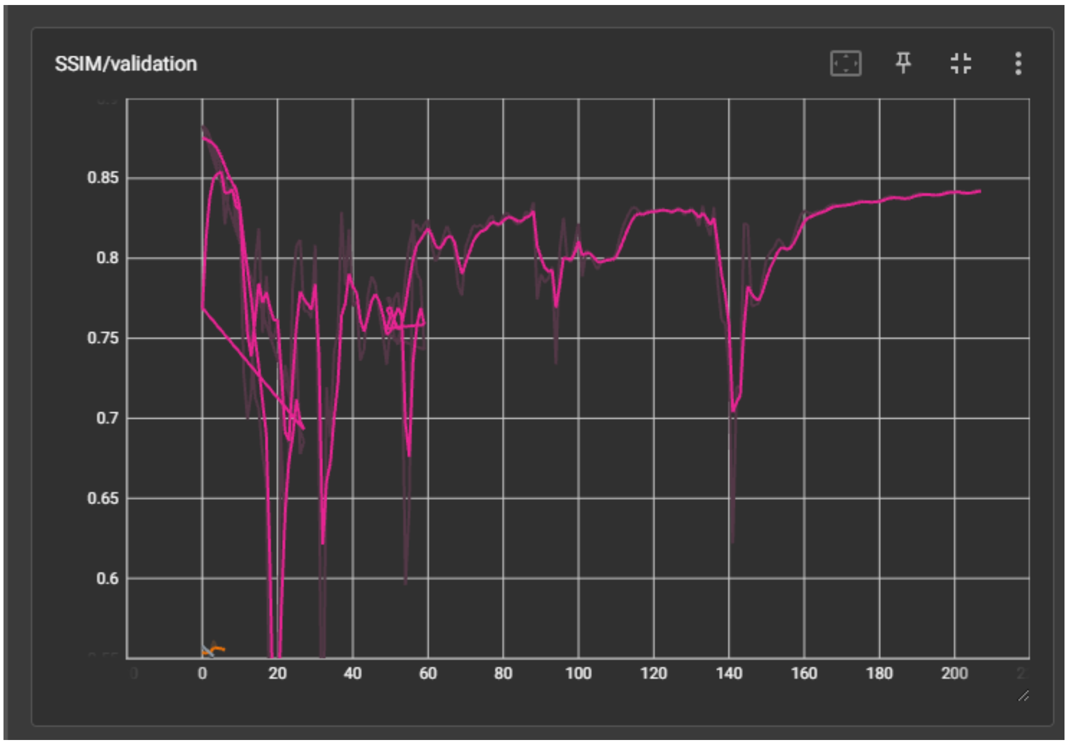


Figure 6:SSIM Validation Plot

4.5. Outputs on Test-Set

The images below are some example outputs that we achieved on a separate test set, that has been unseen by the trained model.
As can be seen, the model can remove occlusions on some occasions while not losing the person. However, in general the performance of deeplab when using it as an image restoration model is much lower than when using it as a pixel-classification model.

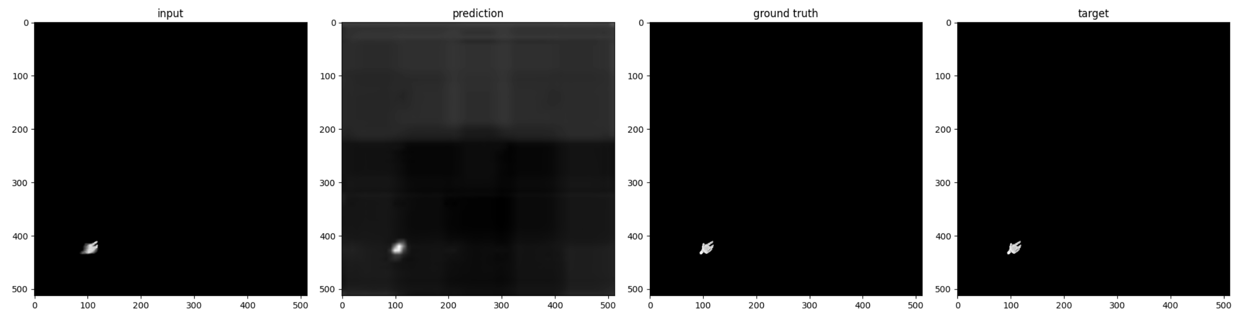


Figure 10:Test-Set Plot 1

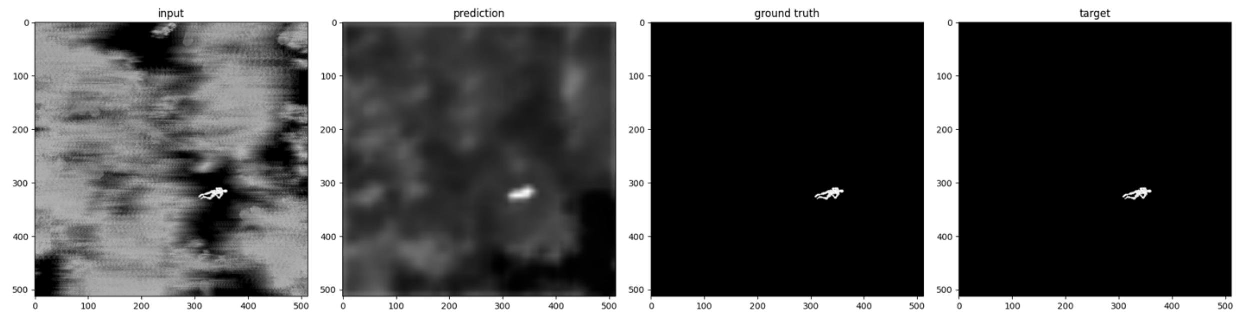


Figure 9: Test-Set Plot 2

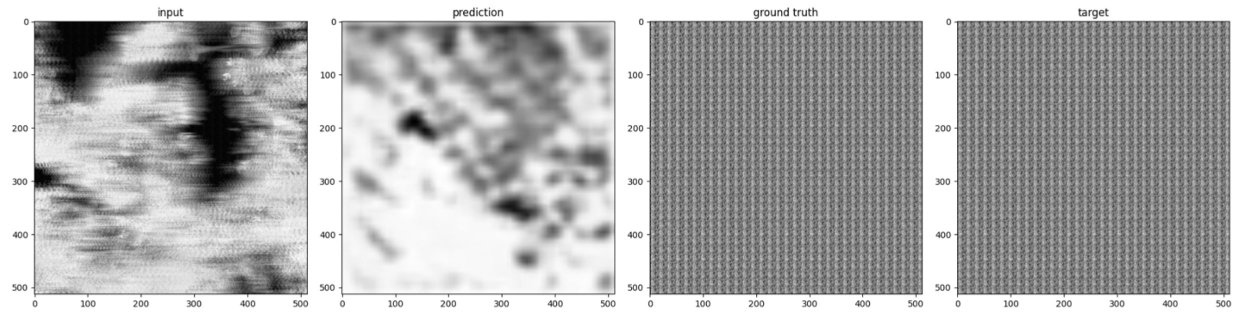


Figure 8: Test-Set Plot 3

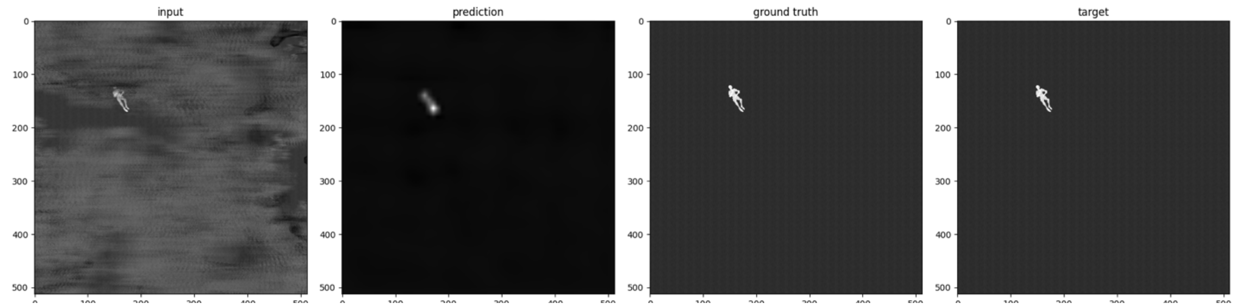


Figure 7: Test-Set Plot 4

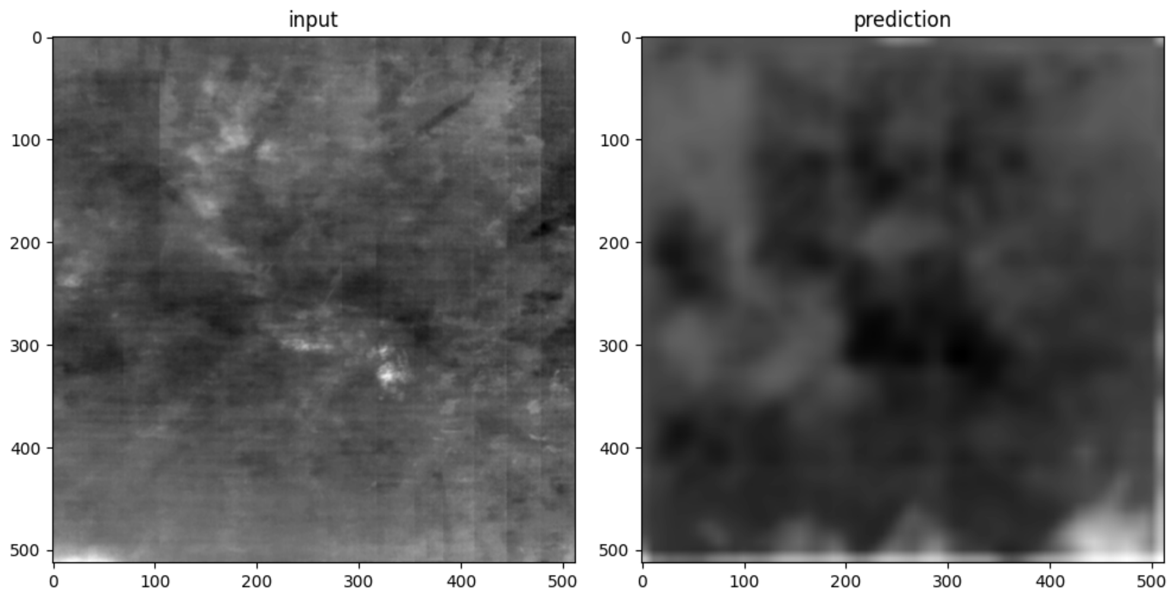


Figure 11: real integral output

4.6. Output on real integral Image

As can be seen in the image below, the output on the real integral image was very lackluster. The model seems to not have generalized well enough to be able to restore any meaningful information from the input image.

5. Docker

The project was Dockerized to achieve seamless functionality across diverse devices and address the notorious "it works on my machine" challenge. The primary motivation was to establish a consistent operating environment that would enable the training and testing of our neural network under uniform initial conditions. Detailed instructions on how to build and run the image are provided in Readme.md

6. Challenges

The Project brought forth unexpected challenges, the magnitude of which may have been underestimated at the outset. A central concern was the effective organization of the team, given that members had no prior knowledge of their colleagues' skills before the project commenced, and teammate David stopped communicating with the rest of the team early into the project.

This complexity hindered task distribution and the assessment of individual contributions. Communication also proved to be demanding, particularly in the synchronization of code fragments and interfaces. Uncertainty regarding expected parameters during implementation, coupled with limited experience with such models, impeded seamless collaboration.

The substantial volume of data presented an additional hurdle, as the storage space available on team members' personal laptops was often insufficient for storing the entire image dataset, requiring substantial efforts spent on data management.

As each training run could take days to complete, minor changes or patches to the visualization and tracking of metrics were not easily implemented. In hindsight, visualizations and quality metrics should have been a priority from the start and should have been stable before hyperparameter experimentation began.

Another challenge was moving from a pixel-wise classification model towards an image restoration output model in the last 3 weeks of the project due to a misunderstanding of the project's requirements. The experiment to adapt the DeepLab architecture to this new requirement was unsuccessful but meaningful insights have been generated.
These challenges underscore the necessity for a structured approach to team organization, communication, data management, and visualization to ensure the smooth progression of projects of this complexity.

7. Summary

In retrospective analysis, we find satisfaction with the project’s results despite numerous challenges, even though the original image restoration goal wasn’t fully achieved. This discrepancy is primarily attributed to limited time resources and computational capacities, which constrained us in terms of hyperparameter optimization, as well as the departure of David F. from the team early on. It is worth noting that DeepLab was originally designed for classification tasks, and this proved to be limiting when adapting the input layer for segmentation tasks.