FSDL Report on Aerial Segmentation

Fabian Berressem, Jonathan Schaust

May 2021

1 Project Goal

The goal of the project was to build a neural network that is able to identify and mark 7 different classes of objects on aerial photographs and determine the real-world-height at each pixel. For this, we used a dataset that is available under the corresponding Weights & Biases task by DroneDeploy.[1] An example for an aerial photo and the corresponding segmentation can be found in Fig. (1).

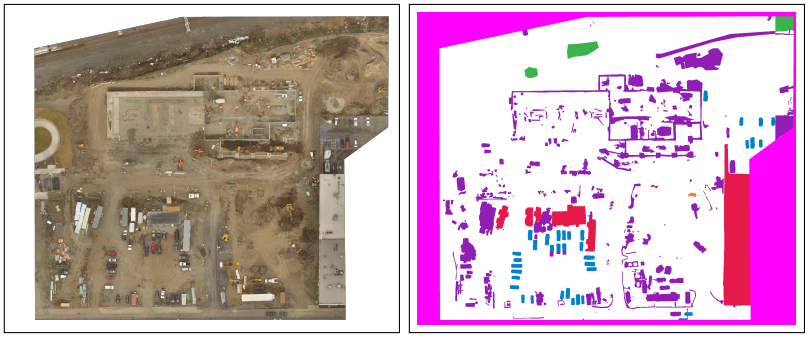

Figure 1: Example of input image (left) and segmentation mask (right). The magenta area in the segmentation mask is ignored.

2 Data Exploration

The dataset provided by DroneDeploy for the Weights & Biases Benchmark consists of 55 aerial orthomosaics[2] of different sizes and shapes and their corresponding segmentation into 7 different classes as well as their elevation images. The different classes comprise buildings, clutter, vegetation, water, ground, car and "ignore"[3] and their color representation can be found in Tab. (1). The orthomosaic photographs as well as the elevation images were given as TIF-images, with the former describing RGB-photographs and the latter describing the elevation with values between 0 and 255.

[1] https://wandb.ai/dronedeploy/dronedeploy-aerial-segmentation/benchmark

[2] An orthomosaic is an aerial image whose perspective and other geometric properties are corrected such that a perfectly straight-down view of all objects in the image is created.

[3] This class is not part of the classes which are to be predicted but rather denotes pixels where there is no information or label.

Object class	Color
Building	Red
Clutter	Purple
Vegetation	Green
Water	Orange
Ground	White
Car	Blue
Ignore	Magenta

Table 1: DroneDeploy dataset classes and corresponding colors.

As first step of the data exploration, we explored the relative frequency of the different classes, which revealed a strong class imbalance with two orders of magnitude between the rarest and the most common class, as can be seen from Fig. (2). This of course has ramifications for the task as the class imbalance has to be accounted for in the loss function so that rare classes are predicted equally well as common ones.

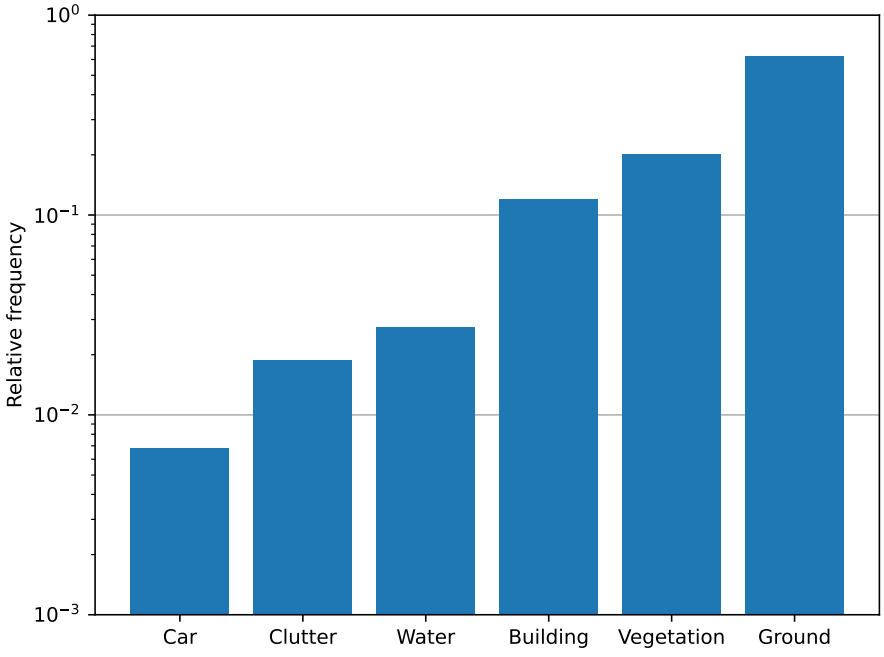

Figure 2: Relative frequency of the different classes of the whole dataset. The class "Ignore" was omitted as it is not part of the relevant classes.

Furthermore, we analyzed distribution of heights for each class, with the result depicted in Fig. (3). As can be seen there, the distributions are very similar with all classes except for water covering the whole range of elevations, which is surprising given the fact that it implies that *e.g.* some cars are larger than some

buildings. This observation hints at mislabeled data, which will be discussed later in this report. Furthermore, the similar distributions of elevations limit the information conveyed by the elevation map if used both as additional input as well as additional target in joint learning.

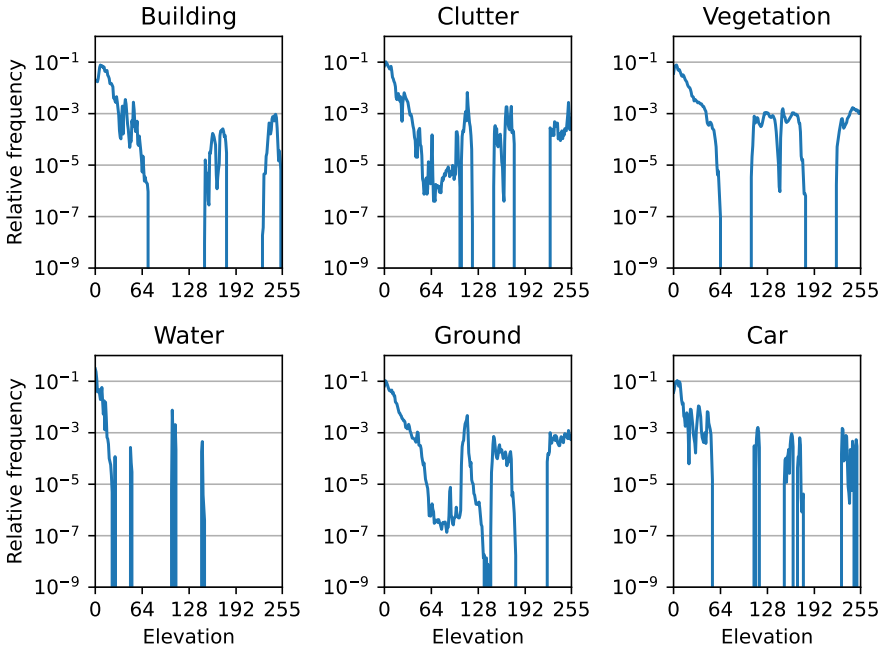

Figure 3: Distribution of elevations for the different classes for the whole dataset. The class "Ignore" was omitted as it is not part of the relevant classes.

3 Approaches

After exploring the provided dataset we decided to build a training framework which allows us to quickly iterate through different experiments. The framework is based on the PyTorch Lightning[1] framework from the Full Stack Deep Learning course[4]. For the preprocessing of the raw dataset we used modified versions of the functions of the Weights & Biases Benchmark repository. Besides that, we used Weights & Biases[2] for tracking and managing of our experiments.

After testing our code by training a simple MLP architecture we decided to focus our search for models to U-Nets[3] because our task is a segmentation task where U-Nets are a decent architecture as has been shown in previous works. Furthermore, U-Nets have the advantage of being applicable to different input-sizes if no dense output network is used and the input-images are padded appropriately. For the generation of models we used an open-source package called Segmentation Models PyTorch[4], which allows for creating models applicable to segmentation tasks. Besides that, Segmentation Models PyTorch

[4]https://github.com/full-stack-deep-learning/fsdl-text-recognizer-2021-labs

offers different encoders which are already pretrained on different tasks such as ImageNet. In our experiments we restricted our search to U-Nets using either "ResNet18", "ResNet50", "ResNet101"[5] or "MobileNetV2"[6] as encoders.

Compared to the baseline provided by Weights & Biases in addition to the hyperparameter search, we implemented some improvements, which can be found below:

- In addition to the cross-entropy we implemented loss functions such as F_1-score or the Focal Tversky Loss[7]. We decided to use these loss functions in our hyperparameter search in order to deal with the class imbalance mentioned in Sec. (2).

- We implemented joint learning by making the model not only predict the segmentation mask but also the elevation map. Even though the distribution of elevations as shown in Fig.(3) hints at some issues with the labeling and the elevation data, we hoped that joint learning might lead to better generalization of the model. The joint learning was realized as additional MSE loss applied to the predicted elevation map that was weighted by a factor α_{elev} to balance the segmentation and the elevation loss.

- We implemented data augmentation techniques such as flipping and rotation in order to allow for a better generalization. Typically the learning procedure converged after a few epochs already so that we deemed it unnecessary to implement additional augmentation techniques such as shearing or occlusion.

- In order to improve generalization and reduce the sensitivity to noisy data, we implemented a flag for optionally learning on adversarial examples using the Fast Gradient Signed Method (FGSM).

Besides these modifications of the training procedure, we implemented some additional improvements of the inference procedure, which are discussed in Sec.(5).

4 Encountered Problems

As mentioned in Sec.(2), during the exploration of the data we found some problems of the data, which will be discussed in more detail in the following together with an evaluation of how this effects the performance of our approach.

- There are examples, where moving objects are not fully captured in the photograph but appear "sliced". This mostly affected cars and should in principle not affect the overall performance significantly, as this occurred sufficiently often for the network to have training examples. Furthermore, this error is inevitable and will occur also in real-world-applications (see for example other applications like Google Street View), as it is not possible to take a orthomosaic picture of large size in one shot. However, as cars are a relatively rare class, miscategorization of these examples can have an influence on the measured performance if the performance measure is accounting for class imbalances. An example of sliced cars can be found in Fig. (4)

- There are multiple examples of mislabeled data, which of course has a significant impact both on the training of the network as well as on the measured performance. Further we assume, that this causes the similar distributions of elevations per class as shown in Fig. (3), rendering the information from the elevation maps less useful. This error could be reduced by relabeling the images, however, unfortunately this was not possible in the scope of this project due to a lack of time and manpower. An example for mislabeled data can be found in Fig. (7), where multiple objects are mislabeled:

  1. Most of the buildings (except for one) are classified as clutter, even though they can clearly be identified as buildings.
  2. In the labeled picture there is vegetation indicated between the two buildings in the top part of the picture even though there is no vegetation this part of the picture at all. In Fig. (7) this is indicated via a red circle.
  3. There are multiple trucks in the picture, that are labeled as clutter. In Fig. (7) this is indicated via blue ellipses.
  4. There is a pick-up truck that is partly identified as car and partly identified as clutter. In Fig. (7) this is indicated via a green ellipsis.

  These are just a few examples of mislabeled objects and there are significantly more in the dataset and even in this picture alone.

- The labeling criterions seem inconsistent, as for example in this picture, small bushes next to the water at the bottom of the picture are labeled as vegetation whereas in other pictures, only trees are considered as vegetation while bushes are not. Again, as is the case for mislabeled data, this has an impact both on the training procedure and the measured performance and could be improved by relabeling the images.

- As discussed in Sec. (2) the dataset suffers from severe class imbalance. This, in combination with the aforementioned mislabeling and inconsistent labeling criterions of classes with fewer examples, leads to metrics correcting for class imbalance like F_1 or Tversky-loss to be partly ineffective and possibly even harmful to the overall performance, as, in order to optimize this measure, the network has to overfit these inconsistencies and mislabeled examples.

- In some pictures there are parts missing from the picture not only at the borders but also in parts of the picture itself. This might cause problems as these blanks have to be replaced by some value so that the network can still be applied the corresponding parts of the picture. However, these sharp transitions might confuse the network either by indicating an edge or by indicating that edges alone are not sufficient to distinguish different objects, even though they typically are. An example of this can be found in Fig. (5), where there are multiple blank spots. In some pictures, said spots were significantly larger, masking even smaller buildings or lakes.

- In our approach we kept the training similar to the one from the Weights & Biases benchmark repository, where the training and validation images

were clipped to subimages of size 300×300. During this clipping, the subimages were checked for missing parts and ignored if any were found. This fact together with a possible difference in complexity of the corresponding data sets and tasks, however, led to the performance on the validation and test set to differ significantly. We attribute this especially to the padding mentioned in the point before as well as pictures in the test set that were significantly harder to predict on due to the effect mentioned in the following point. In order to quantify this discrepancy we trained multiple networks and compared their performance on the validation set to the one on the test set, yielding a correlation coefficient of just 0.8 as calculated by the Weights & Biases parameter performance feature. We stress that we did not use the evaluation of performances on the test set for our choice of the model, so as to avoid leakage.

- Finally, the large size of the images pose a big problem for our approach: As the images are too large to be processed in one step, they had to be divided into subimages, which are predicted on successively and then combined to give the prediction for the whole image. As our chosen approach is purely convolutional, it can deal with different shapes for the input, so that the subimages can be larger than the chips the network was trained on. However, due to hardware constraints and the limited receptive field of the network, it is hard to predict on subimages containing objects to large to be processed in a single step. For example, in some images there were buildings so large, that, if only a subimage of the size of the receptive field of large networks was considered, its roof was almost indistinguishable from normal ground. A part of the corresponding example can be found in Fig. (6) where we kept some cars next to the building as a reference. Besides the problems of propagation of context, predicting on subimages and putting these predictions together for the final predictions also causes artifacts at the borders of each subimage, which is due to the missing context and the padding.

Due to all of these issues with the dataset, training and evaluation are significantly more difficult and less reliable. In Sec. (5) we propose some strategies to potentially overcome some of these problems.
However, the mislabeling alone poses a huge obstacle and makes any attempt of comparing performances sure to fail. Due to the large amount of mislabeled data in all datasets, there are three factors that come into play:

- Training of models is suboptimal due to mislabeled training data
- Choice of models is suboptimal due to mislabeled validation data and hence unreliable validation estimates
- Evaluation of model performance is unreliable due to mislabeled test data

In fact, a model would actually have to do a rather bad job of categorizing objects in order to maximize its performance on the final benchmark, which of course would lead to failure in real-world applications. Therefore, we deem it necessary to improve dataset quality before any real improvement can be made on this specific task, regardless of other problems and improvements.

Figure 4: Multiple example of “sliced” cars in a single picture from the dataset.

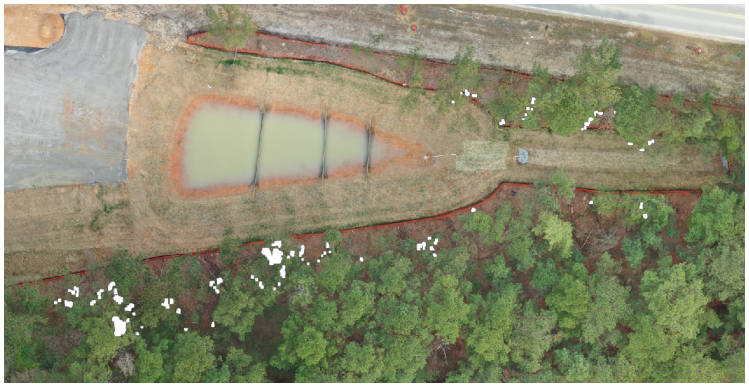

Figure 5: Examples of blank spots in the image.

Figure 6: Example of part of the roof of a large building with cars as size reference. If the edges of the building are not visible, the roof looks very much like normal ground.

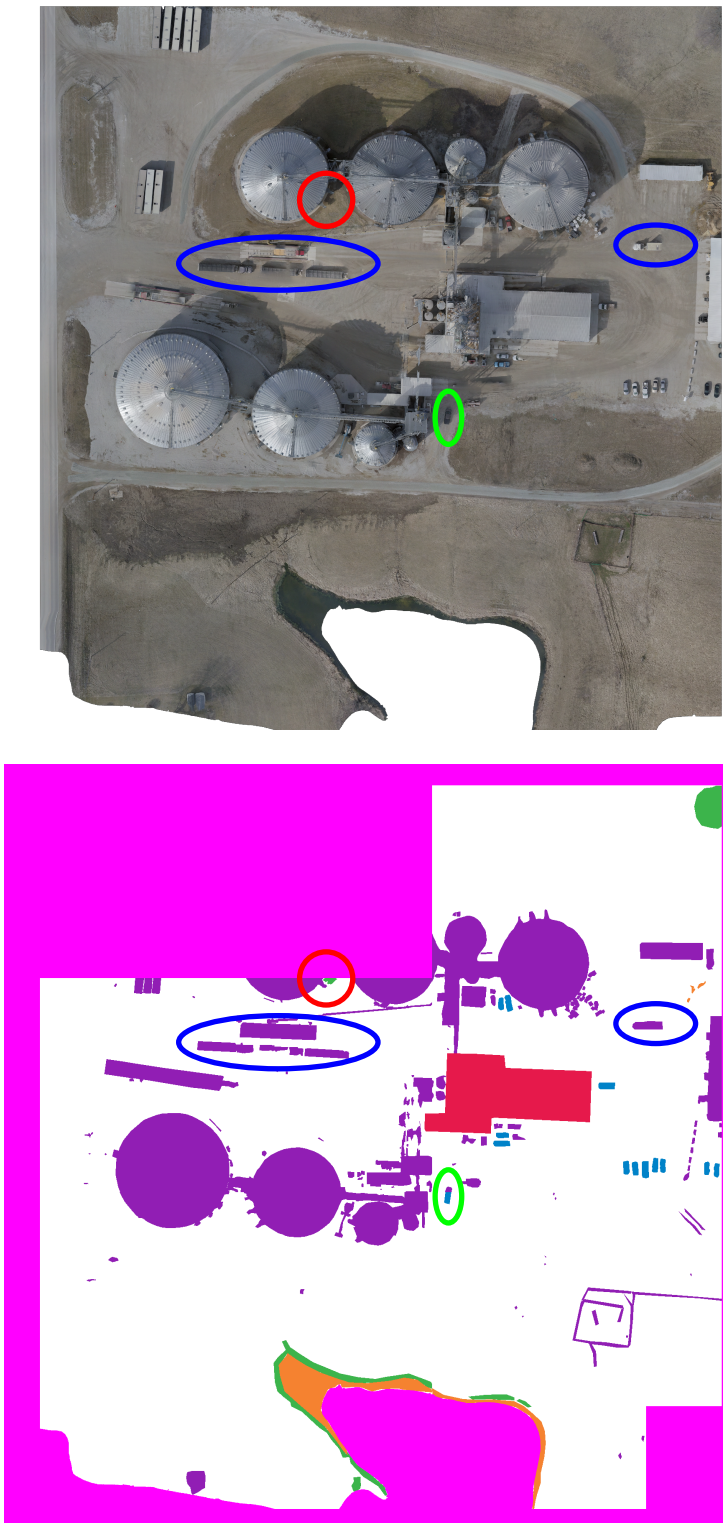

Figure 7: Example for multiple wrong labels in a single picture as described in the list in Sec. (4)

5 Solution Strategies

5.1 Strided Inference

In order to avoid the artifacts at the borders of subimages as described in Sec. (4), we implemented a strided inference, *i.e.* the subimages the network predicts on are overlapping, such that aforementioned artifacts can be corrected for. Here, we define the stride s to mean that, given a subimage-dimension of n the subimages will be moved n/s pixels in this dimension for each following subimage, such that the overlap is given by $o = n \cdot (1 - 1/s)$ pixels in this dimension. This of course comes at the price of increased runtime, as there are more predictions on subimages needed to cover the whole image. However, the increase is only quadratic in the stride and scales the same with the size of the image as unstrided inference. In order to improve the mitigation of boundary artifacts even further, we also added an optional gaussian weighting of the image such that the center of the subimage predictions is weighted more strongly compared to the edging parts when putting the subimage predictions together. This potentially allows for less striding to be sufficient to eliminate boundary effects as the regions suffering from these artifacts are weighted less strongly.

5.2 Cascading Inference

As mentioned before, one of the problems we encountered was the fact, that the subimages provided to the neural network sometimes are too small and lack the context to be analyzed and categorized appropriately. Hence it is desirable to predict on subimages that are as large as possible so as to make full use of the receptive field of the neural network. In principle it is possible to provide larger images to the network compared to those seen during training due to the purely convolutional nature of our neural network, which allows for input of varying size. However, both the memory of the system on which the network is running as well as the receptive field of said network are limited, which in most cases makes it unfeasible to predict the segmentation of the whole image at once. To overcome this issue, we came up with the idea to use what we call cascading inference (CI). In CI we predict the segmentation mask not only once but multiple times, starting from an image that is a downsampled version of the original image, *i.e.* the network predicts first on the whole image that was downsampled such that it can be predicted on at once. Afterwards, the prediction is upsampled to the original shape using linear interpolation or nearest neighbor methods. At each of the following stages, the resolution of the image is increased (typically doubled), cut into subimages and predicted on successively, as is the case in the conventional prediction. After this more fine-grained prediction, the segmentation is upsampled again and merged with the previous, more coarse-grained prediction, either via weighted addition of the probabilities or via weighted addition of the discretization of the coarse-grained prediction.[5] With this method we hope to successively propagate the context from coarser predictions to finer predictions and effectively increase the receptive field of the network while still fulfilling the memory constraints. Of course, CI

[5] With discretization we mean applying argmax along the class-axis and one-hot-encoding the corresponding output.

is more expensive than conventional predictions as there are more predictions needed per image, however as it turns out and is proven below, the runtime only increases by a factor of $S < \frac{4}{3}$ if the runtime of up- and down-sampling images is negligible and if the image resolution is doubled at each stage.
For the proof of above statement, assume w.l.o.g. the image to have the shape $n \times m$ with $n \geq m$, and the input to the network to have the shape $k \times k$ with $k < n, m$. Further, assume that at each step, the resolution is increased by a factor of e. The number of predictions necessary for the whole image using the conventional method is given by

$$N = \frac{n \cdot m}{k^2} \tag{1}$$

If the image is downsampled by a factor of $\epsilon > 1$, the number of predictions becomes

$$N(\epsilon) = \frac{n/\epsilon \cdot m/\epsilon}{k^2} = \frac{n \cdot m}{\epsilon^2 k^2} = \frac{1}{e^2} N \tag{2}$$

This of course implies by simple substitution, that if this downsampling is done d times, *i.e.* the image is downsampled by a factor of ϵ^d, the number of predictions needed at this stage becomes

$$N(\epsilon^d) = \frac{1}{\epsilon^{2d}} N \tag{3}$$

Now the total number of predictions can be calculated as the sum of all the predictions done per stage, where the lower limit is given as $d = 0$ (no downsampling, *i.e.* original size of the image) and the upper limit is given as $d = \lceil \log_\epsilon(n/k) \rceil$ (maximum downsampling, *i.e.* the whole image is downsampled to $k \times k$):

$$N^{CI}(\epsilon) = \sum_{d=0}^{\lceil \log_\epsilon(n/k) \rceil} N(\epsilon^d) \tag{4}$$

$$= \sum_{d=0}^{\lceil \log_\epsilon(n/k) \rceil} \frac{1}{\epsilon^{2d}} N \tag{5}$$

$$= N \sum_{d=0}^{\lceil \log_\epsilon(n/k) \rceil} \frac{1}{\epsilon^{2d}} \tag{6}$$

$$\leq N \frac{\epsilon^2}{\epsilon^2 - 1} \tag{7}$$

In the last step we used that the sum converges (even for $n \to \infty$). By choosing $\epsilon = 2$ we find $N^{CI}(2) = \frac{4}{3} N$, which means that the increase in runtime for this choice of epsilon is only about $\approx 33.3\%$. Moreover, the above proof shows that the cascading inference still scales like the conventional one, *i.e.* both methods scale with $\frac{n \cdot m}{k^2}$ and the cascading inference only yields a constant prefactor to the runtime. Besides that, it is of course possible to start and end at any resolution stage, resulting in a trade-off of runtime, receptive field and resolution and to combine this approach with the strided inference.

6 Evaluation

For our final model, we used the F_1-score on the validation dataset for the model evaluation, but as it turned out, the performances did not vary significantly for different sets of hyperparameters. This implies that the models solve the task almost equally well, which in turn means that the performance is limited more by the task itself rather than model capabilities, especially due to the quality of the dataset, as alluded to earlier.
The model evaluation led to us choosing a MobileNetV2 pretrained on ImageNet as encoder with a cross-entropy loss and $\alpha_{elev} = 10$ to use for joint learning. The model was trained on subimages of size 300×300 while inference was done with subimages of size 4000×4000. For the inference we further used a stride of $s = 7$ with smoothing and a cascading inference with exponent $\epsilon = 2$ covering the whole resolution space of the given images. Using this setup, we trained the model with an early-stopping-callback and evaluated its performance on the test dataset, which yielded an F_1-score of $\langle F_1 \rangle_{test} = 0.8496$, which is up 1.6% from the baseline-value of $\langle F_1^{base} \rangle_{test} = 0.8361$. The corresponding experiment can be found at our project page or at our Weights & Biases submission which was also submitted to the Weights & Biases leaderboard.

7 Conclusion

To summarize our project, we were able to slightly improve on the task of aerial segmentation compared to the baseline provided by Weights & Biases by implementing some conceptual modifications of the training and inference procedure. These modifications included additional loss functions, joint learning, strided inference and cascading inference, which were explored in a hyperparameter search covering additional encoder-models for the applied U-Net architecture as well. However, as we showed and discussed in Sec.(4) there are some severe problems with the dataset, which are especially caused by mislabeling. This was also confirmed by the fact that different models with strongly varying hyperparameters all performed almost equally well indicating an inherent limitation of the task rather than a lack of complexity of the models. So in order to allow for significantly better results, the most reasonable thing to do is to repeat the labeling process in order to get higher quality data for training and evaluation of models. Unfortunately, this was not possible in the scope of this project due to time constraints.

References

[1] William Falcon et al. *PyTorch Lightning*. Software available from github.com. 2019. URL: `https://github.com/PyTorchLightning/pytorch-lightning`.

[2] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: `https://www.wandb.com/`.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[4] Pavel Yakubovskiy. *Segmentation Models Pytorch*. Software available from github.com. 2020. URL: https://github.com/qubvel/segmentation_models.pytorch.

[5] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[6] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[7] Nabila Abraham and Naimul Mefraz Khan. "A Novel Focal Tversky Loss Function With Improved Attention U-Net for Lesion Segmentation". In: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. 2019, pp. 683–687. DOI: 10.1109/ISBI.2019.8759329.