Image processing for Earth observation (ENV-540)

FloodNet: Segmentation of images during a flooding event

Lea Marxen, Nadège Sauvin and Julie Weverbergh

January 15, 2023

1 Introduction

We are currently living a climate crisis. Global change is occurring. With it, extreme events are more and more frequent. Regarding precipitation, while annual mean values are expected to be fairly constant, in Italy for example but not only, "there is a tendency both toward an increase of heavy precipitation events and long dry spells".[1] It is of great importance and interest to be able to detect quickly, thus automatically, if an area has been flooded or not. Indeed, first-aid emergency services and therefore humans in general would benefit from reliable quick response data analysis.

The following project aims at segmenting images in order to distinguish flooded from non-flooded areas. The work that has been done is strongly based on a paper published in 2020 *FloodNet: A High Resolution Aerial Imagery Dataset for Post Flood Scene Understanding*. The images we are working with come from the same dataset they used. "The data [was] collected with small UAV platform, DJI Mavic Pro quadcopters, after Hurricane Harvey. [...] Imagery [was] taken from several flights conducted [in] 2017, at Ford Bend County in Texas and other directly impacted areas."[2]

In the paper, the authors implemented image classification as well as semantic segmentation and Visual Question Answering. It was proved that larger “crop size” and “batch size” improve the performance of the models. In general, for the classification process, accuracies were above 90% for all three models they compared. As for the semantic segmentation, performance analysis can be seen in Figure 1. mIoU was the metric used to compare ENet, DeepLabv3+ and PSPNet.

Method	Building Flooded	Building Non Flooded	Road Flooded	Road Non Flooded	Water	Tree	Vehicle	Pool	Grass	mIoU
ENet [62]	21.82	41.41	14.76	52.53	47.14	62.56	26.21	16.57	75.57	39.84
DeepLabv3+ [59]	28.10	78.10	32.00	81.10	73.00	74.50	33.60	40.00	87.10	58.61
PSPNet [48]	**65.61**	**90.92**	**78.69**	**90.90**	**91.25**	**89.17**	**54.83**	**66.37**	**95.45**	**80.35**

Figure 1. Per-class intersection over union (in %) and their mean value (mIoU) on FloodNet testing set (results from Rahnemoonfar et. al [2]).

They also mention difficulties identifying small objects and detecting flooded building and roads. "Since UAV images only include top view of a building, it is very difficult to estimate how much damages are done on that building. Segmentation models do not perform well in detecting flooded buildings. Similarly flooded roads pose challenge in distinguishing them from non-flooded roads and results from segmentation models prove that."[2]

2 Processing routine

We decided to base our work on the semantic segmentation part of the paper discussed in the introduction[2]. We decided to implement the PSPNet model as it had by far the best performance out of the three compared. And just like them, we used the ten following classes with the same color scale.

- 'background': 0
- 'building-flooded': 1
- 'building-non-flooded': 2
- 'road-flooded': 3
- 'road-non-flooded': 4
- 'water': 5
- 'tree': 6
- 'vehicle': 7
- 'pool': 8
- 'grass': 9

Figure 2. Classification color scale

In the paper, the authors described their implementation of the PSPNet as follows: "For implementing PSPNet, ResNet101 was used as backbone.[...] For image augmentation we used random shuffling, scaling, flipping, and random rotation which helped the models avoid overfitting. [...] During training, we resized the images to 713 $\times$ 713 since large crop size is useful for the high resolution images. For semantic segmentation evaluation metric, we used mean intersection over union (mIoU)."[2]

Taking this into account and based on our knowledge, we implemented a processing routine which is outlined hereafter.

Concerning the data, we split it into train, validation and test sets according to the *FloodNet_split_train_valid_test.csv* we received and which already specified a splitting. Here, about 70% of the images are used for training (278 images), 15% for validation (60 images) and 15% for testing (60 images). To create the DataLoaders, we used a training batch size of 8 or 12 images (depending on the trial) to have the highest number of images in one batch possible. Indeed, we could not go until a batch size of 16, as in this case we reached the total allocated memory size of the available GPUs. For the validation and test sets, we used batches of 4. See Figure 3 for a visualization of the process.

The image augmentations we used for the training are the following : random cropping, horizontal and vertical flipping, changing contrast, rotating and normalizing. For the validation and test steps, we only randomly cropped and normalized the images.

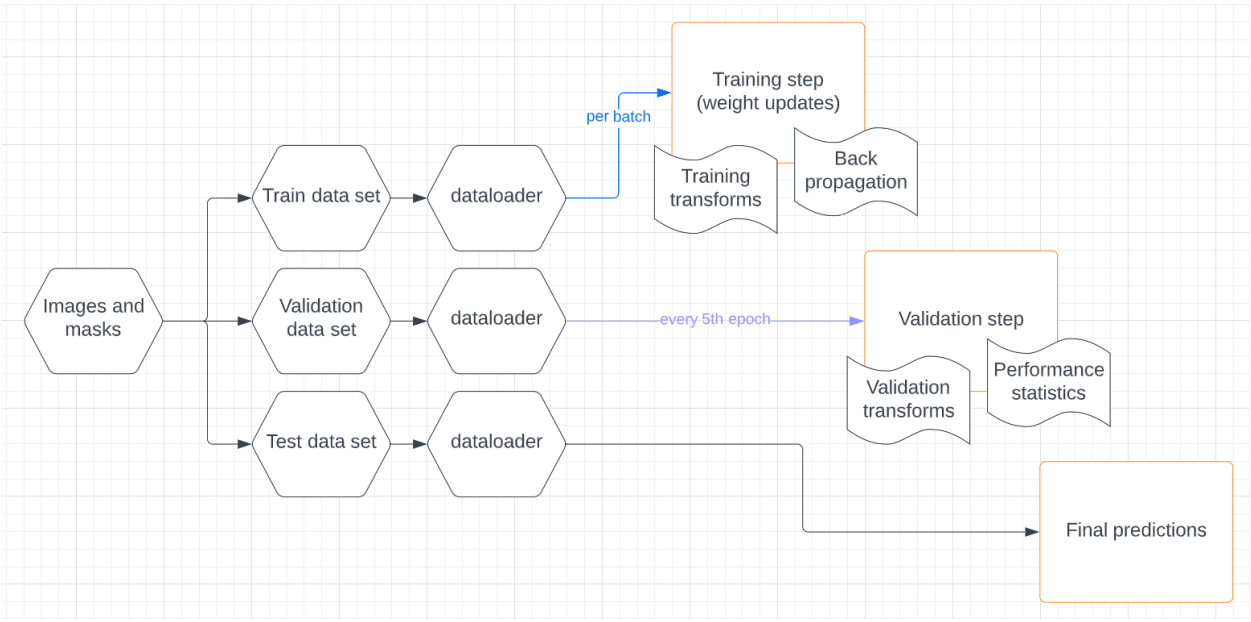

Figure 3. Processing routine

Note that images are not all of the same size. Some are 3000x4000 and some others are 3072x4592. We cropped them to be all of the same size.

Now that we had our data, we could train the model on them. As explained previously, we chose the PSPNet, schematically shown in Figure 4.

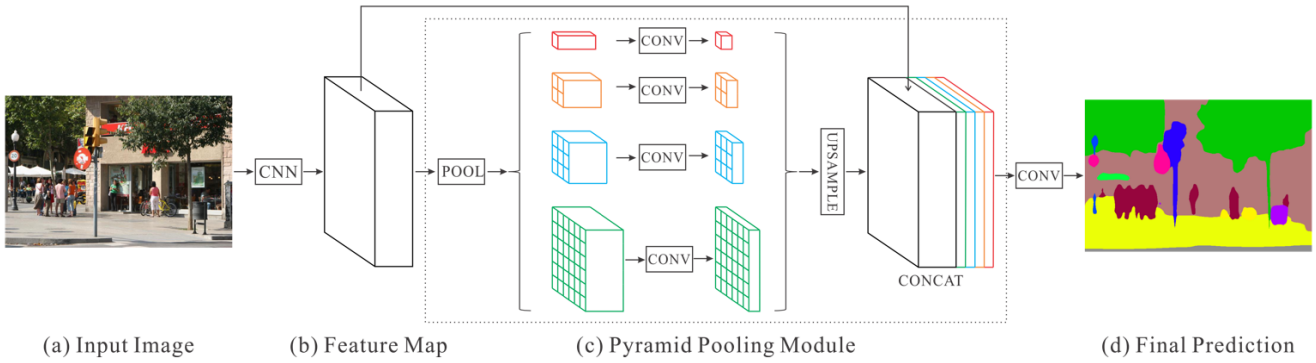

Figure 4. PSPNet scheme (from [3])

More precisely, the input image is the transformed image. Then, it is given as input to a CCN which outputs a feature map. This CNN consists of several layers from a pre-trained ResNet model, which are a succession of convolutions and batch normalizations, to finally obtain a feature dimension of 2048. In a second step, the feature map is given to the pyramid pooling module (PPM). This PPM allows to gather contextual information. The idea is to fuse features under four different pyramid scales [3], by using an adaptative average pooling with a different parameter each time. Through each scale, we then compute a convolution, a batch normalization and a ReLU. Afterwards, we upsample to recover the size of the original feature map by using a bilinear interpolation algorithm, and we concatenate the result from each pyramid scale. We finally use a last convolutional layer to

compute the predictions on our transformed image.

The PPM allows to keep information about the context because at each level of the pyramid, we use a convolutional layer with a kernel size of 1, and this layer reduces the size of the input by 4 only.

To train the model, we made several (30 or 60 depending on the trial) training steps where we optimized the weights at each step, by looping over all batches (for the first simulations, we only considered the 20 first batches, as we had memory errors) and computing the loss. During training, the loss was composed of a cross entropy loss of the overall output and an auxiliary cross entropy loss computed for an intermediate result (shortly before the PPM in the model), in order to enforce the training of the earlier layers of the model. The losses are weighted 60/40 respectively. We used two different optimizers : the Adam optimizer and the Stochastic Gradient Descent (SGD) optimizer. We tried with and without a weight decay and a momentum, respectively of values 0.001 and 0.9 (taken from [2]). We changed the learning rate from 0.0001 to 0.001 and 0.01, with and without a poly-learning rate. The poly-learning rate allows to decrease the rate at each iteration, as described in [2]. To take into account the different number of pixels from each class (indeed, we had a lot of pixels corresponding to trees and grass for example, compared to the number of pixels for the (non-)flooded roads/buildings), we tried different weight configurations for each class in the loss. While other authors like Kampffmeyer et. al[4] used a median frequency balancing to account for unequal class frequencies, we wanted to use different weights to put emphasis on flooded and non-flooded houses and roads, as they are more important than the other classes in this project. We tried three different sets of weights (ordering of classes like in 2):

- Set A : [0.05, 0.2, 0.2, 0.15, 0.15, 0.05, 0.05, 0.05, 0.05, 0.05]
- Set B : [0.00001, 0.3, 0.3, 0.14, 0.14, 0.05, 0.00001, 0.03, 0.03997, 0.00001]
- Set C : [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

Set A allows to give more importance to errors for the (non-)flooded roads/buildings classes, set B increases this effect and set C gives all classes the same weight (default). This difference in weights also allows to emphasize the fact that we want to have less errors for the detection of flooded and non-flooded zones, whereas detecting trees, vehicles and other pools is not of great interest to us.

During training, we tested the model parameters by predicting the classes on the transformed validation dataset, evaluating performance metrics such as accuracy, balanced accuracy, Intersection over Union (IoU) and its mean (mIoU). The balanced accuracy allows to treat each class as if they all consisted of the same number of test pixels, inter alia giving more importance to the classes of interest, i.e. flooded and non flooded roads/buildings.

Finally, we predicted the classes on the test dataset, using the same transforms as in the validation step, and computed statistics.

3 Results

One can find the code for the work we conducted on the following GitHub: `https://github.com/leamarxen/ENV-540-Floodnet.git`

Results in terms of accuracies are depicted in Figure 5, the corresponding IoUs and mIoUs can be found in Figure 6.

	optimizer	#epoch	weight decay and momentum	batch size	#batches per epoch	learning rate	poly-learning rate	weights	train accuracy	train balanced accuracy	validation accuracy	validation balanced accuracy
1	SGD	30	0	2	20	0.01	no	A	72.87%	67.17%	67.91%	44.45%
2	SGD	30	0	2	20	0.001	no	A	69.13%	54.09%	62.63%	30.05%
3	Adam	30	0	2	20	0.01	no	A	50.98%	33.58%	47.04%	17.74%
4	Adam	30	0	2	20	0.001	no	A	59.65%	42.45%	54.48%	15.77%
5	SGD	60	0	3	20	0.01	no	A	70.90%	59.16%	86.01%	52.66%
6	SGD	30	yes	3	20	0.01	no	A	63.77%	50.70%	86.25%	61.13%
7	SGD	30	yes	8	20	0.01	no	A	75.99%	57.04%	78.25%	53.77%
8	SGD	30	yes	8	20	0.01	no	B	11.47%	22.54%	14.47%	23.30%
9	SGD	30	yes	8	20	0.01	no	C	58.33%	14.61%	55.42%	10.00%
10	SGD	30	yes	8	20	0.01	yes	C	79.06%	55.10%	78.73%	55.54%
11	SGD	30	yes	8	all	0.01	yes	C	84.18%	62.95%	80.96%	53.16%
12	SGD	30	yes	8	all	0.01	yes	A	82.14%	60.30%	80.31%	52.13%
13	SGD	30	yes	8	all	0.001	yes	C	80.06%	55.46%	81.65%	54.02%
14	SGD	30	yes	8	all	0.001	yes	A	80.53%	58.09%	83.12%	59.21%
15	SGD	30	yes	8	all	0.0001	yes	C			78.67%	39.71%
16	SGD	30	yes	8	all	0.0001	yes	A	74.86%	49.33%	78.16%	45.17%
17	SGD	30	yes	12	all	0.001	yes	A	81.47%	59.47%	83.19%	55.54%
18	SGD	30	yes	12	all	0.001	yes	A	81.36%	58.58%	81.05%	56.83%

Figure 5. Accuracy metrics for different parameter configurations (1-18).

	Building flooded	Building non-flooded	Road flooded	Road non-flooded	Water	Tree	Vehicule	Pool	Grass	mIoU
1	18.83%	36.21%	14.77%	0.00%	53.04%	41.89%	0.00%	0.00%	63.75%	25.39%
2	3.73%	51.09%	0.49%	0.00%	37.20%	41.73%	0.00%	0.00%	56.43%	21.19%
3	0.00%	0.00%	0.00%	6.58%	18.14%	0.00%	0.00%	0.00%	50.87%	8.40%
4	0.00%	6.60%	0.00%	0.70%	0.00%	22.15%	0.00%	0.00%	56.37%	9.54%
5										
6										
7	25.85%	42.05%	24.44%	55.38%	60.52%	60.75%	0.00%	0.00%	78.81%	38.65%
8	0.00%	6.37%	13.55%	0.00%	31.47%	11.41%	0.00%	0.00%	0.00%	6.98%
9	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	55.42%	6.16%
10	32.56%	47.74%	29.00%	37.44%	60.28%	63.30%	18.20%	18.00%	79.23%	42.86%
11	26.66%	71.27%	26.59%	63.25%	61.99%	69.20%	1.91%	20.59%	78.77%	46.69%
12	31.56%	60.98%	24.16%	39.47%	65.58%	57.24%	0.00%	0.00%	80.68%	39.96%
13	24.16%	68.43%	20.38%	69.67%	45.50%	64.85%	18.90%	12.26%	80.47%	44.96%
14	38.29%	75.06%	39.21%	65.61%	75.96%	59.87%	0.00%	0.00%	80.56%	48.29%
15	14.34%	33.54%	6.50%	52.20%	65.68%	57.59%	0.00%	0.00%	77.87%	34.19%
16	21.05%	40.83%	6.52%	60.34%	74.92%	48.88%	0.00%	0.00%	79.24%	36.87%
17	28.05%	60.65%	29.39%	66.86%	78.57%	64.72%	0.00%	0.00%	82.25%	45.61%
18	34.88%	67.00%	41.53%	59.06%	70.80%	57.00%	0.00%	0.00%	78.41%	45.41%

Figure 6. Intersection over union (IoU) and mean IoU for the same parameter configurations as in Figure 5

For the last configuration (last line in Figure 5), which is also the configuration used for testing, we added some more transforms during the data augmentation step. All the other parameters were unchanged from the previous simulation (line 17 in Figure 5).

3.1 Testing

The predictions for the test set were computed with *optimizer=SGD, epochs=30, weight decay=0.0001, momentum=0.9, batch size=12, batches per epoch=max, poly-learning rate with base learning rate 0.001, class weights=[0.05, 0.2, 0.2, 0.15, 0.15, 0.05, 0.05, 0.05, 0.05, 0.05], set A.*

The statistical results we obtained with this model for the training and validation set are depicted in Figure 7, Figure 8 and Figure 9. Note that we only computed the validation metrics every 5th epoch, hence the particular curves in Figure 7.
Figure 11 to Figure 18 are a few representative label maps of what we obtained from the test set. The ground truth can be seen on the left, coupled with the prediction of the model on the right.

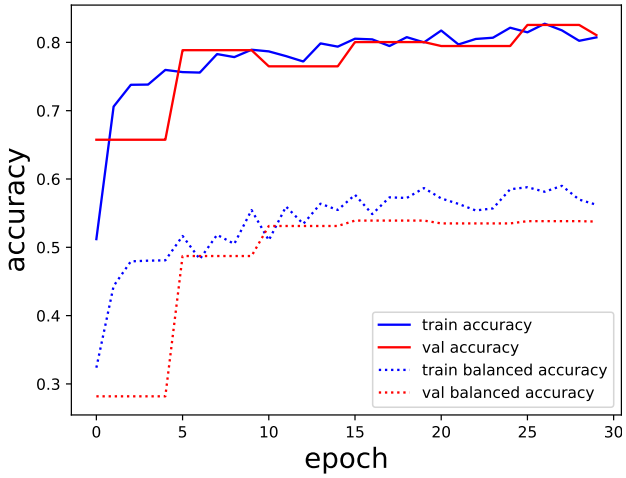

Figure 7. Development of the training and validation accuracies

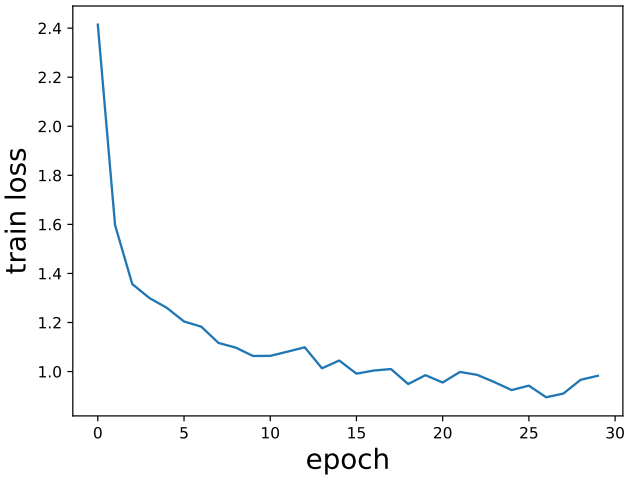

Figure 8. Development of the loss during training

For the statistics over the test set, we got the following results:

- Accuracy : 76.98%
- Balanced accuracy : 54.79%
- Loss : 0.62
- Intersection over Union [%] in Figure 19

and the confusion matrix is represented on Figure 10.

4 Discussion

Accuracies are of course valid assessment metrics, but we believe that for this specific project, IoU is even more relevant. Indeed, as we explained already setting up the class weights, some classes are more important than others. We want to be sure to correctly detect (non-)flooded areas more than any other class.

Unfortunately, the IoU values we got as well as the mIoUs are not satisfying. Compared to the mIoU of 80.35% from [2], we did not even reach 50% on our side (mIoU for test set: 42.15%). Our results are rather similar to those of the ENet, which performed the worst out of the three models tried out in the paper. Apart from the vehicle class, especially the Iou for flooded buildings and roads are low, which is contrary to our goals for this project. However, those classes did also not perform well in the reference paper [2] (although on a higher level), so this is not an unknown problem. In

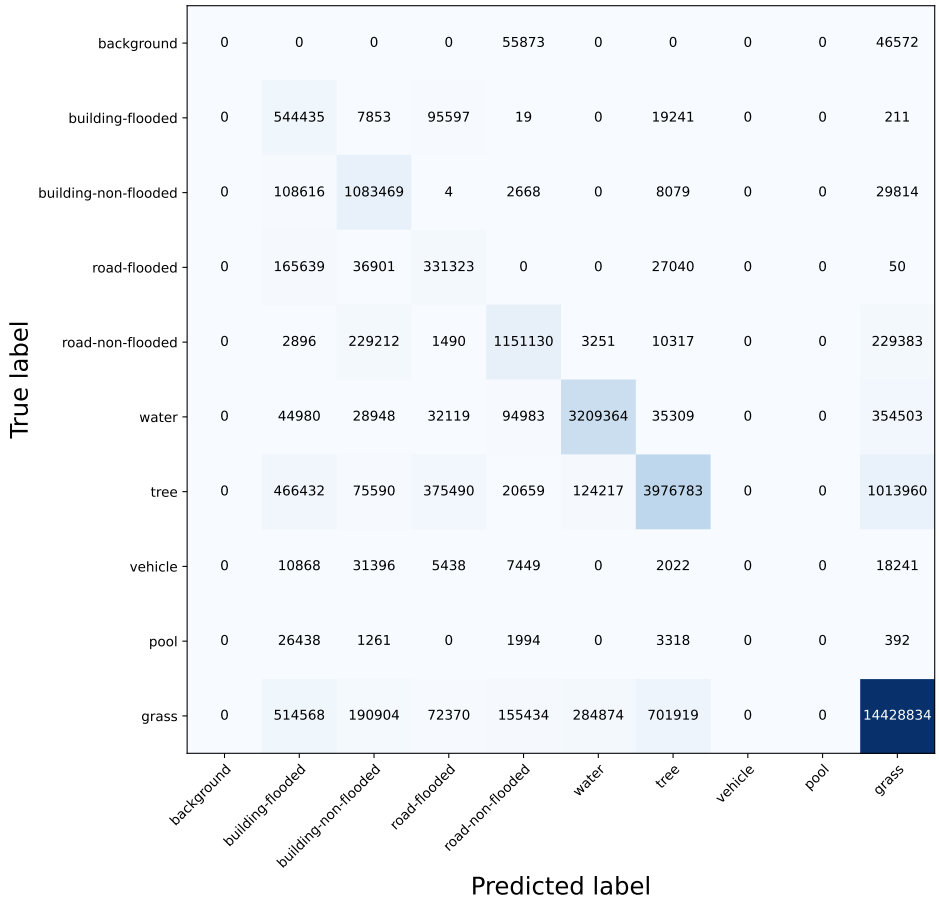

Figure 9. Confusion matrix for validation set

the following, we will analyze our results in more detail and work out the strengths and limitations of the model.

First, we can see that our project serves as a perfect example why it is not always useful to only look at accuracy values. For example, the two simulations with the Adam optimizer (Figure 5 and Figure 6, lines 3 and 4) resulted in relatively high validation accuracy scores, while the mIoU column shows extremely low results. This is probably due to the very unbalanced class distribution in the image dataset. A look at the confusion matrix (Figure 9) confirms this observation, showing a heavy proportion of grass pixels. Indeed, many images contained only or for the most part grass (see e.g. Figure 12), which is not at all the interest for this project.

Moreover, one can notice that small objects, as in the paper, do not perform well at all. This can be seen in the prediction maps (Figure 11, Figure 13, Figure 14, Figure 15 for example) as well as in the IoUs table (columns "Vehicule" and "Pool", Figure 6). This problem might also be due to the class imbalances. As small objects by nature do not have many pixels, their class does not appear as often during training and thus cannot be sufficiently learned by the model.

One strategy to counteract this problem of class inequalities is to use different class weights dur-

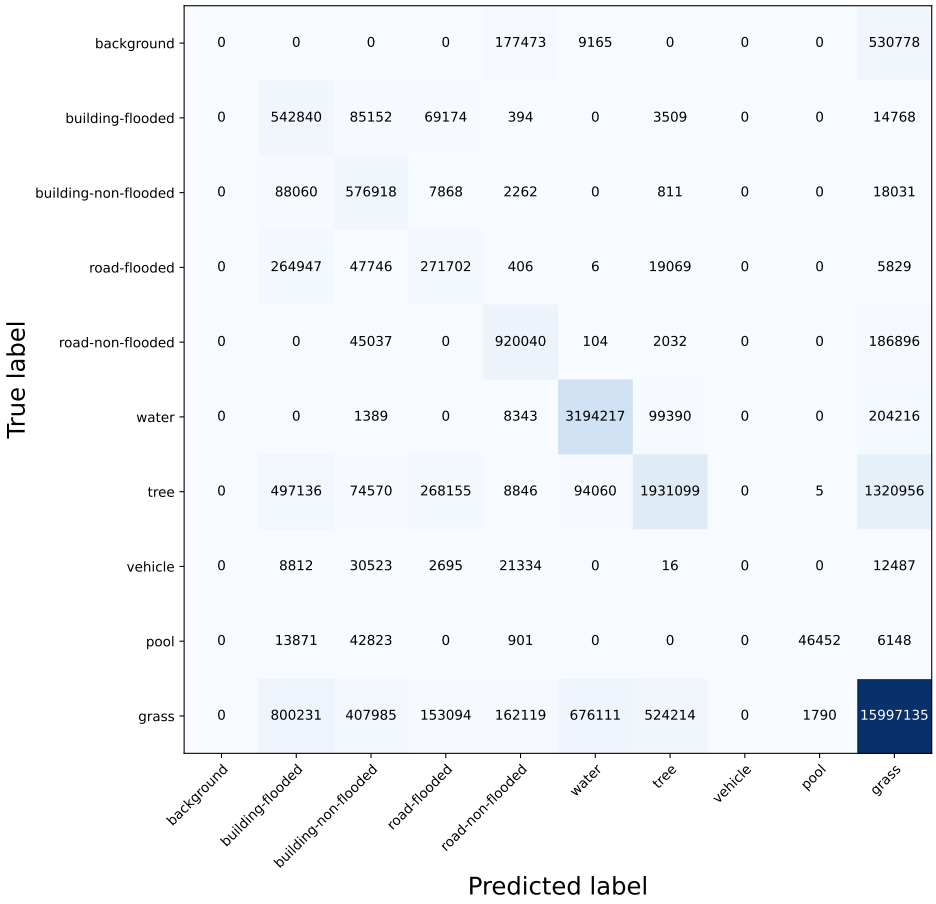

Figure 10. Confusion matrix for test set

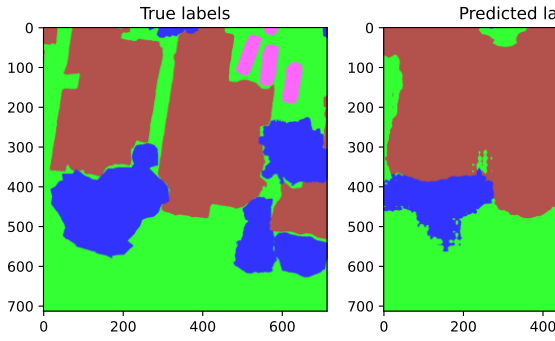

Figure 11

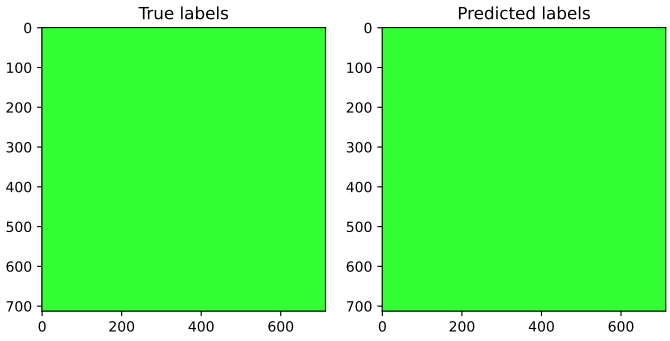

Figure 12

ing the computation of the loss [4]. As we were mainly interested in the flooded and non-flooded houses and roads, we chose to neglect the other underrepresented classes (like vehicle and pool) and concentrated on a few classes instead. For Set A, this resulted in the weights 0.2 and 0.15 for (non-)flooded houses and roads respectively, while the rest was set to 0.05. A more extreme approach (Set B), which highlighted the house pixels even more (weight of 0.3), set the weights for roads to 0.3 and the rest to different low numbers (e.g. 0.00001 for grass), was abandoned after one try, as it performed very badly. The last configuration, Set C, passes the same weights to each class. Although we expected the model to perform worse on classes like cars and pools (which also

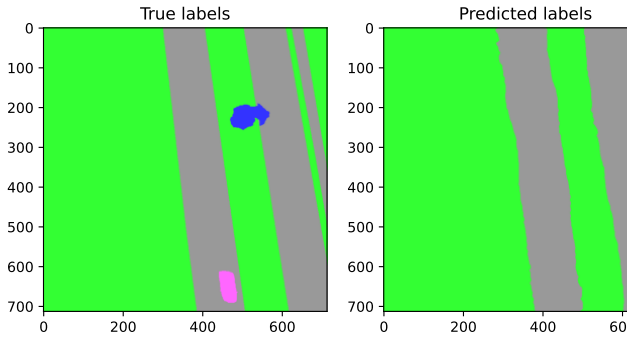

Figure 13

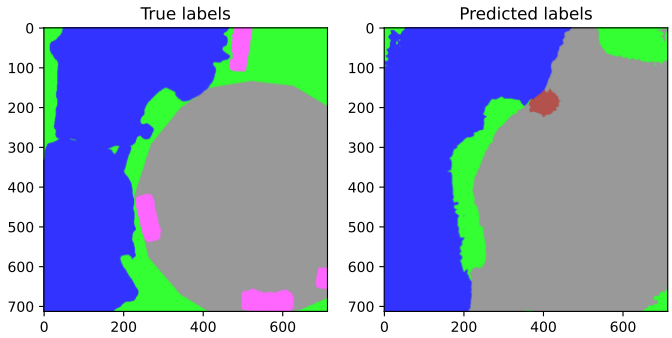

Figure 14

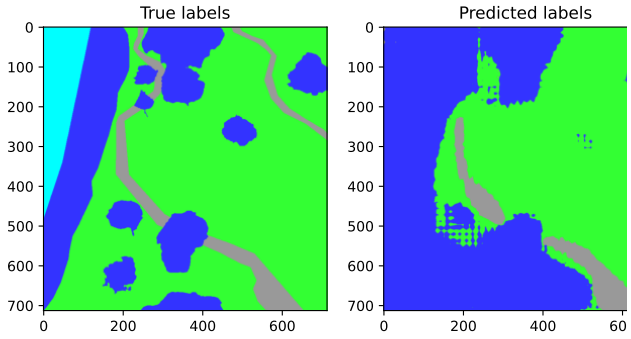

Figure 15

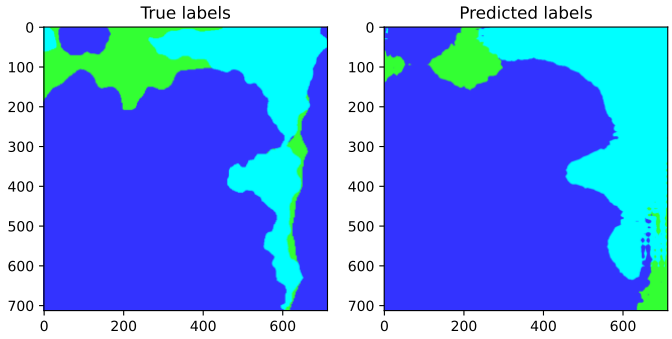

Figure 16

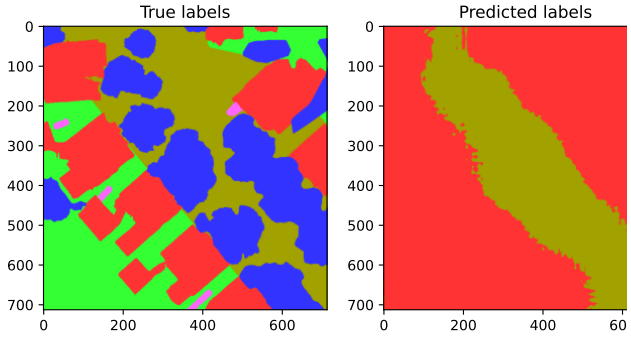

Figure 17

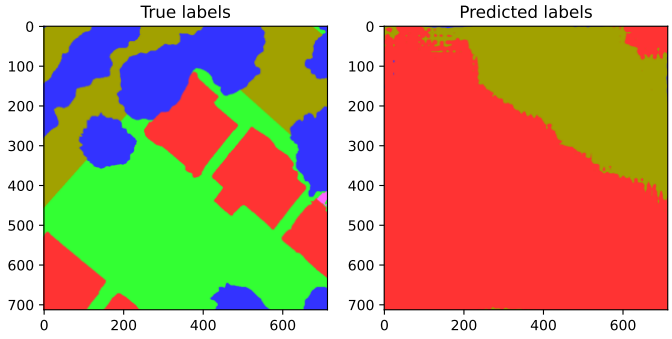

Figure 18

	Building flooded	Building non-flooded	Road flooded	Road non-flooded	Water	Tree	Vehicule	Pool	Grass	mIoU
test	22.72%	40.37%	24.46%	59.89%	74.51%	39.87%	0.00%	41.48%	76.09%	42.15%

Figure 19. Intersection over Union for the test set

lowers the mIoC), we were hoping for better accuracies for the classes with higher weights.

To our surprise, the configurations of set A and set C did not perform significantly differently. Even the classes which were given a high weight were not classified much better by the model. This can also be observed when looking at exemplary label maps : the images from Figure 20 to Figure 25 show predictions where only the class weights are changed, leaving the rest of the parameters as follows: *SGD optimizer, weight decay and momentum, batch size of 8, all batches, poly-learning rate*

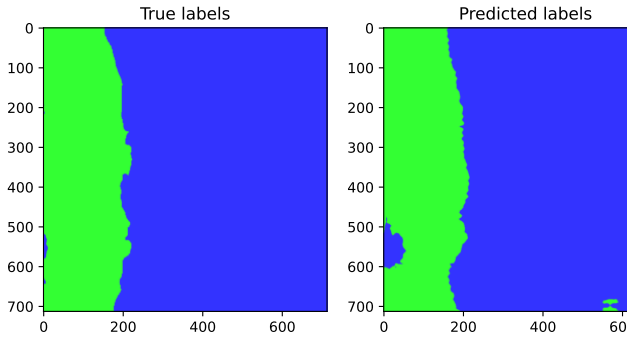

Figure 20. Equal weights : 0.1 all (set C)

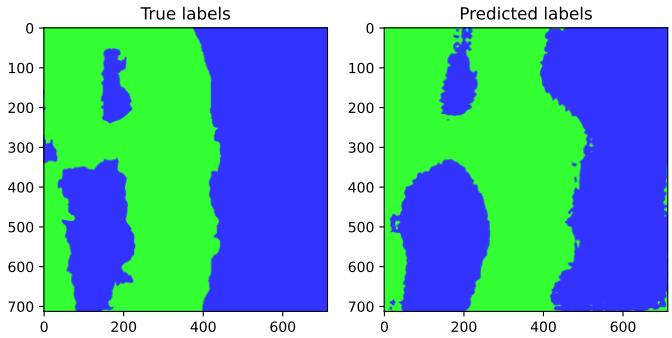

Figure 21. Non equal weights (set A)

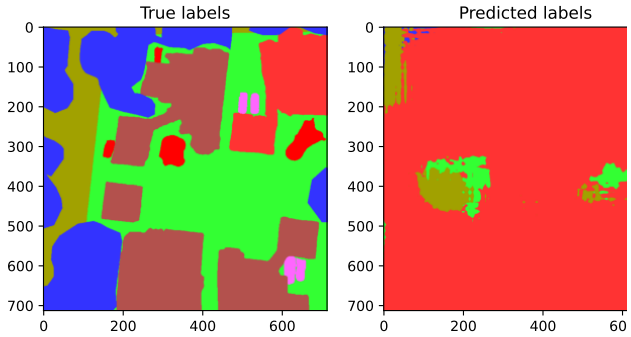

Figure 22. Equal weights : 0.1 all (set C)

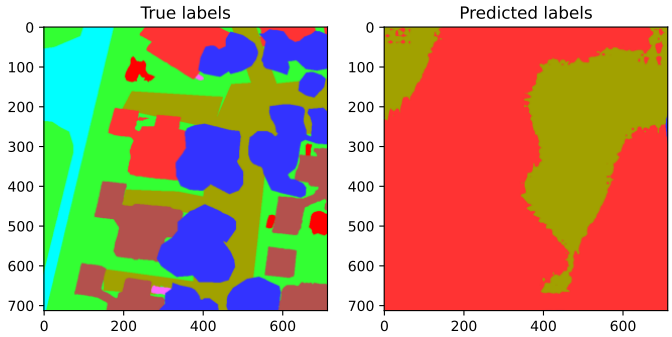

Figure 23. Non equal weights (set A)

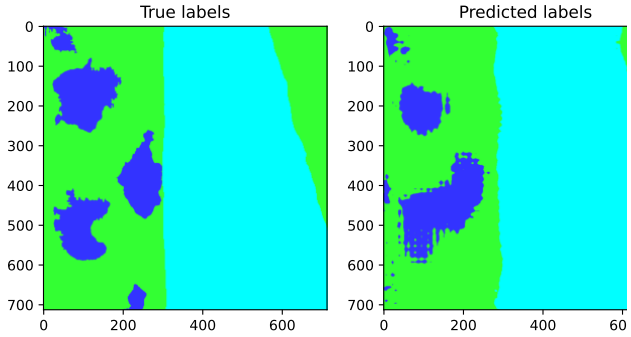

Figure 24. Equal weights : 0.1 all (set C)

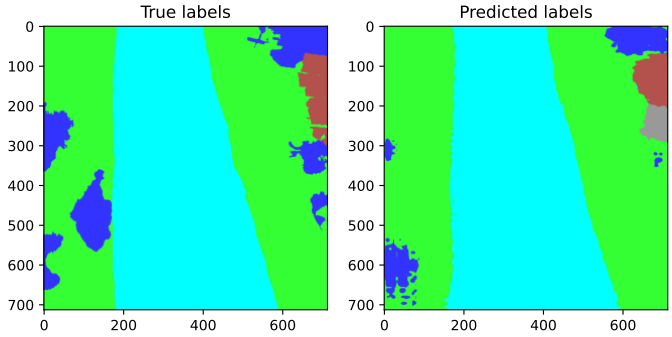

Figure 25. Non equal weights (set A)

with base rate of 0.001. Due to the cropping of the images during the data loading, the predictions were not executed on the exact same cutout of the image. Thus the label maps look a bit different. Knowing this, there is no clear visible advantage of the set A weights over the equal weights. It could be possible that in order to correctly classify flooded houses and roads, the model needs the other classes as context for a correct labeling of flooded areas. This circumstance would be interesting to investigate further, but is beyond the scope of this project.

On a different note, one can see that the model struggles a lot when predicting labels in neighbourhoods. As long as the image is composed of water, grass or trees, the predictions seem accurate enough. But as soon as many small and different elements appear, especially houses, we note inaccuracies. This effect is even stronger when flooded buildings are involved (see Figure 22 and

Figure 23). The model tends to exaggerate the flooded areas, even irrespective of the choice of weights for the respective classes. Although this is a flaw we must acknowledge, this situation is preferable to an underestimation of the flooded housing area - better be safe than sorry.

Additionally, we studied other parameter configurations. Among them, we trained 60 epochs instead of 30, but the accuracy plot showed a saturation around 30 epochs. As each epoch takes around 3 minutes to compute, we returned to the lower number. The Adam optimizer equally did not perform as well and was thus abandoned in favor of the SGD optimizer. We also changed other parameters like the learning rate, weight decay or momentum, but did not perform a thorough grid search, which would have been preferable. This shortcoming was mainly due to the limited computing power we worked with, which brings us to the limitations of the project.

4.1 Limitations

In the paper, it was advised to use a large crop and batch size in order to improve the learning capacities of the model during training. We used the same crop size as was presented in the paper, though it seemed rather small to us. However, considering the memory challenges we faced when training the model, we decided not to change it. The same held for the batch size the authors of our reference paper did not specify the term "large batch size", so we tried different sizes. However, we soon found limits: As our model was already relatively large, one GPU could not compute more than 6 images per batch. We were able to implement multiprocessing in Pytorch, so we could double the batch size by using two GPUs during training. Theoretically, we could have increased the batch size by requesting more GPUs, but seeing that there were 8 GPUs in total reserved for the whole course, we contented ourselves with two.

At the beginning, when we were training the model, the SCITAS kernel died without error message at some point. We gradually found out that it must be caused by memory issues on the cluster. At first, our best solution was to only use a (random) selection of all images for each epoch. After a while, we figured out that we could request more CPU memory through the additional comment "-m 64G" when requesting computation and memory capacities in SCITAS, which completely solved our problem. So in the end, we were able to train the model with all training images in each epoch.

Last but not least, another limiting factor was simply time. We had many ideas how to change and improve our model and the parameters, but the scope of the project only allowed for the implementation of some of them.

4.2 Further Ideas

When realizing a deep learning project, there are always yet untried ways to improve a model. Our project is no different. First, we could have explored more and different parameter configurations: we could have implemented a step learning rate instead of a poly-learning rate, tried a bigger batch size through the use of more GPUs or a larger image size than 713x713. The effect of changes of parameters like weight decay, momentum and the weight of the auxiliary loss could likewise be evaluated.

Another exploratory route would be more image augmentation. Although we tried one other transformation configuration in the end, there is a lot of potential in this direction. Especially seeing that the grass class was largely over-represented during training, it might be beneficial to select those images which show flooded and non-flooded houses and streets and to multiply them with the help of data augmentation.

Moreover, although PSPNet performed better than two other models in our reference paper, it would have been interesting to implement a second model ourselves. There is a wide range of deep learning models and configurations used for semantic segmentation [5], e.g. the UNet [6] could be implemented, or in general a CNN which also learns the upsampling instead of using a pre-defined interpolation algorithm.

Considering the use case of first-aid during a flooding event where a model should predict the flooded areas without a large error, it would be important to know how well the model generalizes. For this, data from other flooded regions could be collected and annotated. The model could be fine-tuned with the help of images from some regions and then tested on images of other regions. Like this, the generalization capacity of the model could be both improved and tested.

5 Conclusion

In this project, we implemented a CNN to apply semantic segmentation on the FloodNet image set, with the goal to reproduce results achieved by Rahnemoonfar et al. [2]. Although we were not able to reach the same IoUs and mIoU, we could successfully implement the PSPNet model and contributed own ideas like the different weighting of classes during the loss computation. All in all, the topic was very interesting to work on and has a lot of potential for future research.

References

[1] M. Brunetti et al. *Temperature, precipitation and extreme events during the last century in Italy*. 2004. URL: https://doi.org/10.1016/S0921-8181(03)00104-8.

[2] M. Rahnemoonfar et al. *FloodNet: A High Resolution Aerial Imagery Dataset for Post Flood Scene Understanding*. 2021. URL: https://doi.org/10.48550/arXiv.2012.02951.

[3] *PSPNet*. 2017. URL: https://doi.org/10.48550/arXiv.1612.01105.

[4] Michael Kampffmeyer, Arnt-Børre Salberg, and Robert Jenssen. "Semantic Segmentation of Small Objects and Modeling of Uncertainty in Urban Remote Sensing Images Using Deep Convolutional Neural Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016, pp. 680–688. DOI: 10.1109/CVPRW.2016.90.

[5] Devis Tuia et al. "Deep Learning-based Semantic Segmentation in Remote Sensing". In: *Deep Learning for the Earth Sciences*. John Wiley & Sons, Ltd, 2021. Chap. 5, pp. 46–66. ISBN: 9781119646181. DOI: https://doi.org/10.1002/9781119646181.ch5.

[6] 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. DOI: 10.1007/978-3-319-24574-4_28.