Material Acceleration for Deep Learning Project in Order to Prevent Urban Flooding

Brice CONVERS[a], Jeanne LEMOINE[b]

[a]*Grenoble INP Phelma SICOM, brice.convers@grenoble-inp.org,*
[b]*Grenoble INP Phelma SICOM, jeanne.lemoine@grenoble-inp.org,*

Abstract

In order to detect the types of soils, we built a Multiclass Semantic Segmentation Model called U-Net. Then we trained it on the drone dataset: Aerial Drone Images, which includes many drones images in urban area. We performed binary and multiclass classification by parallelizing training on several GPUs. We also studied in greater depth the influence of hyperparameters on our results.

Keywords: Deep Learning, Image Segmentation, U-NET, Aerial Drone Images

1. Problem definition

In this project, we would like to map the soil types in urban areas for stormwater drainage. Knowing the extent of impermeable or permeable surfaces to absorb water could be very useful in preventing any type of flooding. We have set the goal of creating a model capable of detecting the types of soils that the drone flies over. To address this issue, we have implemented a U-Net model to perform image segmentation on a database called *Aerial Drone Images*. This model is the state of the art for image segmentation. To carry out this project successfully, we added classes objects, metrics, parallelism computation (5) and also data augmentation (4). We first focused on the binary case and then, we developed the multiclass classification with 6 classes. This document deals only with the image segmentation problem. We present our approach to achieving this goal, as well as the results and performance of our model.

2. Method implemented

The initial guideline with object-oriented programming from the U-Net: Training Image Segmentation Models in PyTorch (1) was interesting. We decided to extend it to our entire code by implementing classes and methods for the training, testing, data management and metrics.

2.1. Model Architecture

We implemented the U-Net architecture using Python 3 and the PyTorch library. The implementation of the model in the U-Net class can be divided into four parts. First, we defined the class named Block which represents our smaller component of our architecture composed of one 2x2 convolution layer followed by a ReLu activation layer and again a 2x2 convolutional layer. The second part is the encoder class used in the contraction path. This class consists of 4 Blocks and a 2×2 max pooling layer. The third part is the decoder class, which takes the feature map from the lower layer, upconverts it, crops and concatenates it with the encoder data of the same level. The last part is defining the U-Net model using the previous classes. The different parameters we used for our architecture are visible below in the table 1. After research, the Adam optimizer has been chosen as the best optimizer to start with. It is possible to find the U-Net architecture figure in the *README.md* file at (6).

In the code project (6) the parameters can be tuned by changing the *config.py* file. Next, the only thing to do is to run the *main.py* file which will call the U-Net class, all the other classes and methods we implemented depending on the parameters: the SegmentationDataset class to manage the dataset, the CustomDataset class which augment the data, the Metrics class to calculate the metrics, the TrainModel class that trains our model and the TestModel class which makes predictions.

Model architecture	
Encoding and Decoding channels	(3, 16, 32, 64) and (64, 32, 16)
Size of filters	2x2
Optimizer	Adam (Adaptive Moment Estimation)
Function loss	Cross-entropy loss and BCEWithLogitsLoss
Interpolation	torch.nn.functional.interpolate()

Table 1: Parameters of our architecture

2.2. Hyperparameters

In this section, we describe hyperparameters that we have chosen and which will be constant in the project.
We know larger kernel sizes are effective at capturing larger features in an image, while smaller kernel sizes focus on finer details. However, larger kernel sizes require more computations, leading to an increased computational cost. Since we work with many parameters in the model, and we have to capture details on images to segment accurately, we decided to use a small kernel size of (2,2) in order to concentrate on data augmentation, which takes longer. Since we do not have many images, we chose a split test of 15% in order to have as many trained images as possible. To avoid interpolation issues during prediction, we need to have an input image size for the model significantly larger than the size of our model's layer (in this case, 64). However, retaining the original size of our images is not feasible due to the excessive number of parameters it would introduce. Therefore, we have opted for image sizes of 512 by 512. We decided to have a constant learning rate during the training. In the future, it could be better to define a scheduler to adapt this learning rate. However, since we look for an effective initial learning rate for each type of classification, we consider it optimal. However, if the initial learning rate would be not well chosen, it can also lead to problems such as slow convergence or the risk of convergence to local minima.

3. Dataset Protocol

The initial dataset consists of 400 images of 4000x6000 pixels, which we resize to fit the problem. Over the course of the project, to avoid memory saturation, we manipulate the image paths rather than the images themselves. We created the training and test data loader by splitting the dataset using the *train_test_split()* function in the *sklearn* library by setting a percentage in the *config.py* file. It's important to bear in mind that when training, the small amount of retained images known as the test dataset will be used to test the model later. Unlike a cross-validation set, these images are never used during the training process. The aim is to obtain a perfect, unbiased validation model.

3.1. Cross validation and early stopping

In order to keep an idea of how the training is going, we have set up a cross-validation loop which allows us to test a batch not involved in the training at each epoch. For this specific batch, the model makes predictions, and calculates the validation loss and the DICE score. It is also at this point in the code that early stopping is implemented. This is a technique for avoiding overfitting and improving the generalization of our model. The idea is to monitor the performance of our model on the validation set during training and to stop the training process when it starts to deteriorate, even if the performance on the training set continues to improve.

3.2. Data augmentation

We noticed that the performance of our model was very limited. Thus, we decided to do some data augmentation, that is to use our 400 images to make 1600. It has been done by cropping randomly and flipping horizontally and vertically the initial dataset. We obtain 1600 images, which gives us 1360 for the training and 240 for the test (15%). The data has to be generated once on the machine. Then it can be used by turning on the dataset augmented using the *config.py* file. In the results section, we will study the effect of data augmentation.

4. Parallelization

We implemented the data parallelization illustrated in the figure 1, allowing us to divide the dataset among different GPUs, and each GPU processes a subset of the data simultaneously with the same model architecture. The data parallelism leads to faster training times compared to a single GPU and it is suitable for large amounts of datasets as we augmented the size of the dataset. It also reduces the memory requirements since each GPU processes a fraction of the data independently. Yet, we observed some cons when using data parallelism. The performance is lower regarding the accuracy, implying a bad communication between GPUs to synchronize model parameters. Also, the speedup achieved with data parallelism is not really linear when increasing the number of GPUs. It can even slow down as one GPU can be limited by the size of the model. In our tests, we make sure that the batch size is a multiple of the GPU size to have faster computation.

The proper procedure that we follow was to train the model on several GPUs to define the best parameters in the shortest possible time, then train it on a single GPU to improve the results.

5. Binary Classification Results

Let's focus first on a simple way to test our model: unlabeled binary classification.

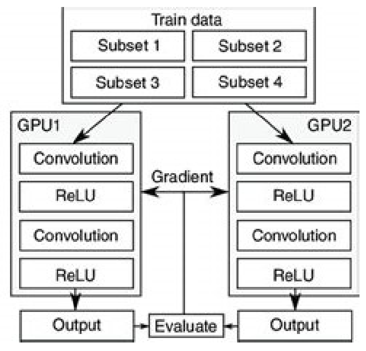

Figure 1: Data parallelism

5.1. Unlabeled Binary Classification

5.1.1. Description

To begin with a very simple manner to test our network, we created new ground truth data with a mean threshold. In fact, as we didn't have semantic meaning, a mean threshold is the simplest and so the best choice possible as we follow the say: *"keep it simple"*. First, we standardize our ground truth mask images to have the right distribution of pixels. Then, we binarize our mask images to have a ground truth unlabeled dataset. The figure 2 illustrates one of the best results that our model produces. We used data augmentation and hyperparameters explained in 2.2 with only one GPU.

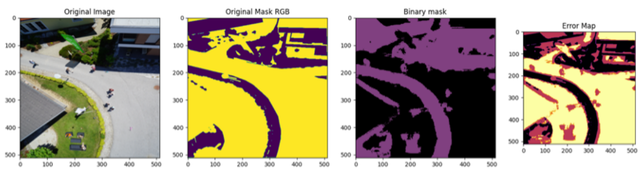

Figure 2: Unlabeled Binary Classification
From left to right: the original image, the unlabeled ground truth, the unlabeled binary classification prediction, and the error map (True prediction: yellow for one class and black for the other, False Positive and True Negative: red).

To obtain the prediction, we transform the output of our model (a tensor with a value between 0 and 1) into a mask using a median (or mean) threshold as we want an equal distribution of pixels.

In our analysis, we thought about looking more into this part to find a better threshold using advanced techniques such as optimization. However, because we do not have semantic training, expecting meaningful predictions does not make sense. Thus, it was not really necessary for us to spend more time on this type of classification. The main purpose of these predictions is just to extract some contours.

5.1.2. Results

Since we threshold our class masks with a binary threshold, we differentiate our images into two classes based on pixel intensity. This might have the effect of extracting several contours from images. Indeed, a contour is defined by an abrupt variation in pixel values and with these techniques we differentiate image area with that. Finally, with the figure 2 we can see that we succeeded in extracting some contours as expected. It is very approximate, but given our approximate training data, it is visually satisfactory. We can recognize borders between shadows for example and this is an element that will not help for our project. We want to distinguish soil types to prevent flooding, and shadows damage the results. Now we focus on metrics to go more in depth.

5.1.3. Metrics

The error map extracts wrong detection areas on the predicted image. We can see in figure 2 that most of the cases with fewer error predictions (in red) are at the boundaries between classes. This is good, as it is more a question of model inaccuracy than actual error. However, there are isolated areas of error that correspond to other contours that we can explain by the wrong defined contours during the creation of our training dataset. Then, we use metric scores such as F1 and dice which measure the similarity between two images. We found F1 equal to 7% and dice equal to 13%. We explain that these scores are very low by the fact that our predictions are very random.

5.1.4. Conclusion

This section shows that with a very simple problem without any labels we can separate our images according to contours. Metrics show that this technique was very limited and results can be very random. It is because our initial problem is not properly posed. The contours we have in this part are too sensitive to disturbances such as shadows, and as we have no semantics, we cannot determine whether a soil is waterproof or not. This first stage was the entry point for validating the project's architecture. Let's add a semantic on the binary classification.

5.2. Semantic Binary Classification

5.2.1. Description

In this section, we have tried binary segmentation with labelled classes: we are trying to detect a single known class in the image. We chose to detect paved ground because it is the most represented class in the dataset, and it is an important point for accurate results. The aim is to highlight which pixels are relative to this zone and which are not in order to determine which soils cannot absorb water. As opposed the the previous part we have now a meaning to give to the predicted image. To form the ground truth images, we created a tensor with a dimension where the pixels are 1 for the paved areas and 0 for all other classes. We tried first to adapt hyperparameters to have good results with multiple GPUs, and then we studied the effect of the variation for some hyperparameters with only one GPU. We found that we always obtain consistent convergence of our model for a learning rate of 0.0001, a batch size of 8 and a kernel size of (2,2). In the next part we study the variation effect for augmented data, the complexity of the model, and the number of epochs.

5.2.2. Results

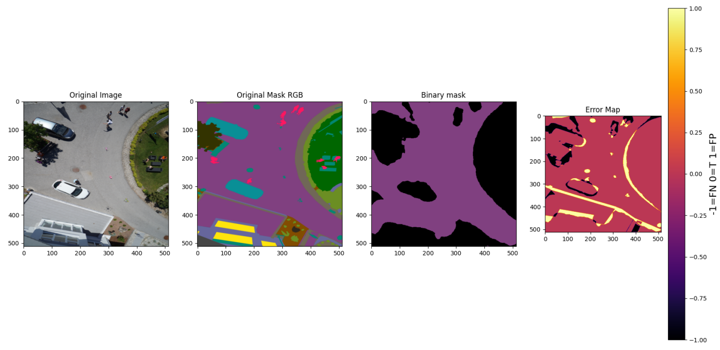

Figure 3: Test Example of Semantic Binary Classification Results for ID Session Equal to binaire-1-4

We can see that the results in the figure 3 are visually very consistent. We have managed to detect the paved areas of this image uniformly, i.e. without overfitting. We can see from the error map that most of the errors (yellow and black) are present at class boundaries. We manage to correctly exclude small and large elements such as cars and people. This means that in the binary-1-4 test, we have succeeded in finding the right number of model depths for this problem. This image is a good representation of the level of performance we were able to achieve with our model and fine-tuned hyperparameters. For example, for this binary-1-4 test, we obtained an F1 score of 75% and the mean of normalized confusion matrix through all tested images is about 83% which means we have good detections for paved areas. More data about this test can be seen in the appendices 8.

IDSESSION	binaire-1	binaire-1-2	binaire-1-3	binaire-1-4	binaire-1-5
Hyperparameters					
Augmented Data	True	False	False	True	True
Model Layers	(3, 16, 32, 64)	(3, 16, 32, 64)	(3, 16, 32, 64)	(3, 16, 32, 64, 128)	(3, 16, 32, 64, 256)
Epoch Number	25	15	45	25	25
Last Test Loss Score	0.58	0.42	0.38	0.34	0.34
Metrics					
F1	0.59	0.51	0.49	0.75	0.63
dice	0.59	0.73	0.73	0.79	0.75
Mean of Confusion Matrix Diagonal	0.82	0.83	0.75	0.83	0.81
Training Time (s)	2806	1683	5051	6294	28374

Table 2: Semantic Binary Segmentation Metrics

5.2.3. Metrics

In this section, we go deeper into the hyperparameter effects to understand how to improve the model. The table 2 sums up our tests in functions of hyperparameters. Firstly, we can see that the use of data augmentation has improved our results. For example, between the binary-1-3 and binary-1 tests, there was an 18% improvement in the F1 score thanks to the data augmentation. This can be explained by the fact that, with data augmentation, we have more training data and we can generalize our model more easily. The loss curve during training showed us that this would not have been possible without data augmentation. However, it comes with a 60% increase in calculation time.

Next, we studied the influence of the complexity of our model. We can see that by adding depth to our U-Net model, we were able to increase performance on the F1 score by 27%. We also observe much faster convergence. Model complexity increases with computation time, but as we increase convergence time, this was not too significant between the binary-1-3 and binary-1-4 tests. We explain these observations by the fact that when we add depth to the model, it allows us to capture finer details in the image, but it also significantly increases the number of parameters. In that example, finer details lead to better segmentation. Subsequently, with the binary-1-5 test, we observe that adding another layer of size 256 was not useful in this problem. The result was a huge increase in computing time and zero accuracy improvement.

The number of epochs is important, and we observe that between the binary-1-2 and binary-1-3 tests, by increasing the number of epochs we have reduced the model's performance. For example, by converging more we lowered the loss function but we lost in F1 score (minus 2.5%), in the mean of the diagonal of the confusion matrix (minus 10%) and we increased computation time (plus 60%). We explain this by the fact that we overfit the model. In this case, we had not activated data augmentation, we have too little data, so we might start to get too specific.

5.2.4. Conclusion

To conclude, in this part we analyzed our results and were able to observe the influence of various parameters on binary semantic segmentation. We obtain for our best results an F1 score of 75%, a dice score of 75%, an average detection for the diagonal of the confusion matrix of 83%. It is possible to find much more metrics int the *output* directory in (6). By taking these approximations into account, we can easily exploit this model in the calculation of impervious surfaces such as paved floors. To go further, in the following part we seek to carry out multiclass segmentation in order to integrate new classes of water permeable zones.

6. Multiclass Classification Results

During our study, we conducted many tests and we present in this document only relevant outcomes for the study of U-Net.

6.1. Description

The goal is to classify all pixels of an image in various labeled classes. During this part we choose to classify 6 classes as follows in figure 3. We aimed to prevent flooding in urban areas and to do so we sought to segment images in function of various zones with different rainwater absorption coefficients. In the table 3, the rainwater absorption coefficient for Unlabeled is unknown, but for the others we sort them in increasing order. we assume that dirt absorbs rainwater less well than grass because it will be drier. In addition, we also assume that the roofs are equipped with pipes, which improves absorption (it might be called into question in the event of network saturation).

Names	Unlabeled	Paved-Area	Dirt	Grass	Roof	Water
Colors	⬛	🟪	🟫	🟩	◼	🟦

Table 3: Table of 6 labeled classes

To make our multiclass classification problem as simple as possible, we have chosen classes also that are as representative as possible. To do this, we plotted average histograms of our ground-truth images. We have also chosen classes that represent large, more or less uniform areas. In this section, we won't deal with more difficult and different objects, such as pedestrians, bicycles, etc.

6.2. Hyperparameters

For the 6-class multiclass segmentation, we began by independently testing several hyper-parameters in order to achieve good convergence toward consistent results. Then we tried to improve our results by varying several parameters such as: complexity of our model, number of classes and data augmentation. We found consistent convergence with a batch size of 16. We established a learning rate of 0.001. It is higher than the binary classification. In fact, with 6 classes our problem is much more complicated and in order to converge to a global minimum we need to be more stable and robust. So this larger learning rate might be useful for escaping local minima and quickly finding global minimum areas. Furthermore, We have a considerably greater number of parameters, so we want to converge faster with a larger learning rate.

6.3. Results

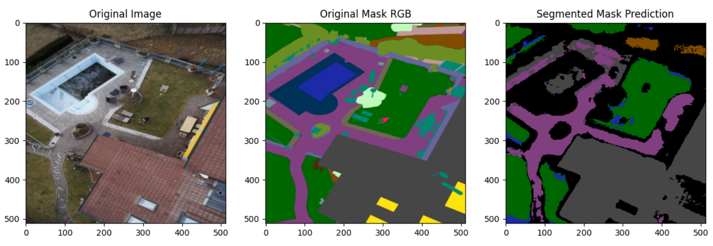

Figure 4: Multiclass Classification Results for 6 Classes with Test ID equal to multiclass-3

In the figure 4 we can observe a consistent segmentation which illustrates the performance of our fine-tuned model. There is some noise in the segmentation, indicating a slight overfitting during training. So some regularization techniques could improve this result. We notice with the figure 5 that it is difficult for the model to detect water and dirt areas. This is completely the case in our example in the figure 4. We can explain this by the fact that even though we have used data augmentation, there is too little representation of water and land in the classes. These areas are often uniform and easily distinguishable, that is why it is a matter of representation. A solution to go further could be to adapt weight for each class in the loss function in order to penalize a bit some classes that are too frequently represented. In the next session, we will study the hyperparameters effects.

6.4. Metrics

IDSESSION	multi class-3	multi class-3-2	multi class-3-3	multi class-2
Hyperparameters				
Augmented Data	True	True	False	False
Model Layers	(3, 16, 32, 64)	(3, 16, 32, 64, 128)	(3, 16, 32, 64)	(3, 16, 32, 64)
Epoch Number	45	45	45	30
Class Number	6	6	6	5
Last Test Loss Score	0.5	0.5	0.85	0.5
Metrics				
F1	0.47	0.45	0.34	0.49
dice	0.47	0.52	0.45	0.49
Mean of Confusion Matrix Diagonal	0.67	0.66	0.43	0.68
Training Time (s)	31200	29958	11968	16200

Table 4: Semantic Binary Segmentation Metrics

First of all, we observe that the augmented data leads again to better results. Compared to the data augmentation, we have a 27% loss on the F1 score if we do not activate it. It is one of our big

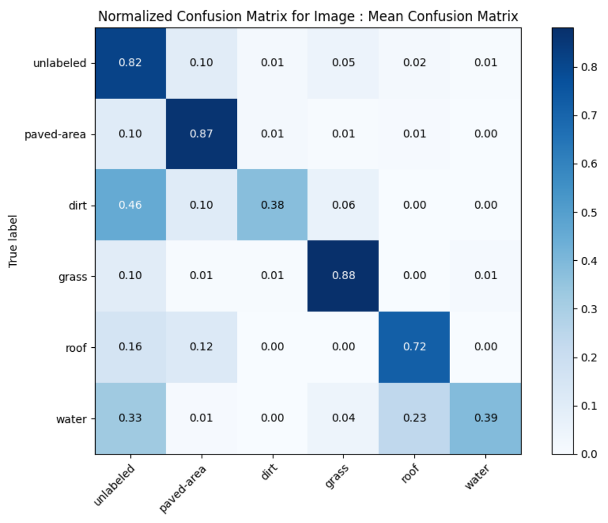

Figure 5: Normalized Confusion Matrix for 6 classes and for ID test equal to multiclass-3

improvements in the problem. We also remarked that the training time is longer for augmented data as intended because we have more data to process. During the project, we also try to increase the complexity of the model. We have added depth to our model with a convolution size of 256. This resulted in a huge increase in calculation time, and the results were no more accurate. One of our interpretations is that the advantage of increasing the depth of our model is to detect finer details in the image, but surely in this complicated problem, there are no interests as we are already generalizing well enough. We also observed if we try with fewer classes, the problem is simpler and detection is more accurate. With 5 classes, we have an 8% increase in the F1 score compared with 6 classes but we lost the information where is the water area.

In the figure 5, we have already spoken about the confusion matrix, which makes it easy to see false and true detections. We observe that we have good detection for 4 classes with a score upper more than 0.72.

6.5. Conclusion

To conclude this section, we have detailed the results we obtained in the multiclass segmentation section. On average, our results are much poorer than those obtained with binary segmentation, but this is due to the fact that we have a much more complicated problem for exactly the same amount of data. We did, however, obtain results that would be useful in calculating water absorption by soils in our initial problem. We have noticed a predominance of certain classes, which lower the performance of other classes. Possible solutions would then be to adjust the weights for each class appropriately, in order to obtain better accuracy.

7. Conclusion

We showed how we trained our U-Net model to recognize different areas in a drone image to calculate rainwater absorption. We carried out this project by starting with the simplest way of dealing with classification, then making it more complex to obtain more interesting results. From the initial model, we implemented techniques such as data augmentation and fine-tuning of our parameters to improve the results of our model. We managed to build a Semantic Segmentation Model that works. We can train it using a single GPU and perform parallel computation over multiple GPUs. We also had a look at a set of metric evaluations. We are satisfied with our results for both binary and multiclass classification that answer the project issues with a clear implementation, even though we know there is room for improvement regarding the results. We could go further by studying other aspects of the project: the influence of kernel size in convolutions, adding a scheduler to vary our learning rate or improving data augmentation. You can find in the Appendices section more details about metrics used to evaluate within our best training model or for 24-class training.
Although both members of the group participated in the project, we have mainly retained Brice's code and its results in this report. For this reason, we have allocated a share of the work as follows: 70% to Brice Convers and 30% to Jeanne Lemoine.

References

[1] Aerial Semantic Segmentation Drone Dataset Bulent Siyah,
[2] U-Net: Training Image Segmentation Models in PyTorch, Shivam Chandhok
[3] Lecture's Dawood Al Chanti, Dawood Al Chanti
[4] Multiclass Semantic Segmentation of Aerial Drone Images Using Deep Learning, ayushdabra
[5] Distributed Data Parallel in PyTorch Tutorial Series, Suraj Subramanian
[6] Semantic_Segmentation_U-NET Repository, Brice Convers, Jeanne Lemoine

8. Appendices

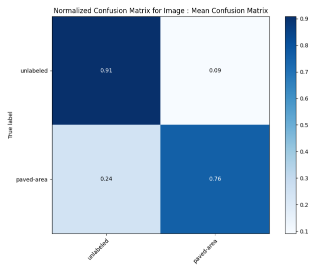

Figure 6: Normalized Confusion Matrix for Semantic Binary Segmentation which illustrate the average through 15 images and for ID training equal to binaire-1-4

Figure 7: Precision Recall Curve for Semantic Multiclass Segmentation, which illustrate the average through 15 images and for ID training equal to multiclass-3

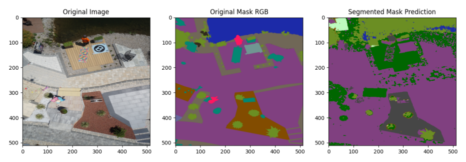

Figure 8: Example of Prediction for Semantic Multiclass Segmentation for 24 classes and for ID training equal to train_12_12_23
F1 score: 0.09