Road Image Segmentation using CNN
EPFL CS-433, Project 2

Louis-Alexandre Ongaro, Garance Haefliger, Julia Wälti

***Abstract*—In this project, we used the U-Net neural network architecture to train an image segmentation model to detect roads. The original data set contained 100 training images and 50 testing images for submissions. We used different techniques and tuned hyperparameters to enhance the performance of our model. Indeed, we tuned the learning rate using grid-search and then tested a learning rate scheduler, weight decay, dropout, and image augmentation techniques. Our best result (test F1 = 0.872) was obtained for an augmented dataset, with constant learning rate $5.5e-4$, weight decay $1e-7$ and dropout probability 0.25.**

I. Introduction

Image segmentation is a subtype of classification problem, where each pixel of an image must be labeled according to the different classes of labels available. Using convolutional neural networks (CNN), we will try to come up with a model that we will train on the provided data, to achieve the best possible predictions of the labels on a test set. Thus, we will be able to compare the predicted class to the real labels of the test images. We chose the PyTorch library to create the model, loaders, and training. The goal of the model is to do image segmentation based on the classification of the ground as background or road using the groundtruth images provided. These groundtruths serve as masks of the satellite images, where every pixel is set either black (0 for the background) or white (1 for the roads).

II. Base case

In order to get a first result and to have a basis to carry our next steps of the project, we came up with a simple baseline model. We thus used a multilayer perceptron (or MLP), a simple fully connected neural network for the baseline. In a similar fashion to the provided convolutional neural network, this model works on the mean and variance of 16x16 patches of the images instead of working on individual pixels. Moreover, a balancing of the data was also performed to help improve the accuracy of the baseline prediction.

The multilayer perceptron model used consisted of 2 hidden layers of 10 neurons each. Separating the train data into a training set and a validation set, we then used the training set to fit the model, followed by a prediction of the model on the validation set. The F1 score on the validation set was 0.662. The trained model was used on the test set to generate a submission. We thus obtained a test F1 score of 0.466. The F1 score on the test set is a lot lower than on the validation set; it is maybe due to the fact that training on only the mean and variance of patches is not as accurate as pixel-wise training on small data sets and that the model can overfit the validation set.

III. U-Net Model

To best fit the problem we have at hand, we used the U-Net model. This model is designed specifically for image segmentation and was first proposed in the paper [1] *U-Net: Convolutional Networks for Biomedical Image Segmentation*.

The name of this fully convolutional network comes from its particular U-shape, with a structure divided into an encoder (the downward segment), a bottleneck (bottom segment), and a decoder (upward segment). This structure is shown below on Figure 1.

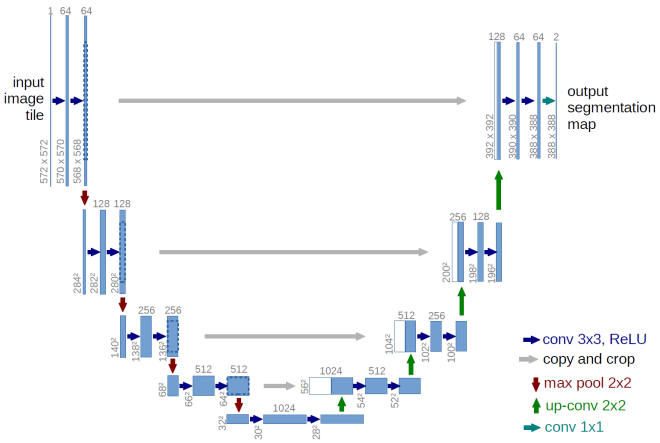


Fig. 1: UNet Architecture

The encoder (contraction path) is composed of four contraction steps with two 3x3 convolutional layers with an activation layer, followed by a 2x2 max pooling layer. The ReLU function is used for the activation layer (except the last one). This path allows for an extraction of the features of the image through the halving of the image size and the doubling of the image features at every step.

The next step is the bottleneck, consisting of two 3x3 convolutional layers and an activating layer as before. This is the step of the U-Net where we get the smallest size of the image, but on the other hand, we have the highest number of features.

Finally, the decoder (expanding path) has also four expanding steps for the corresponding encoder steps. Each step is composed of an up-sampling 2x2 convolution, followed by a concatenation of the corresponding contraction of the encoder step. Then, the image will go through the convolution and

activation twice (similarly to what has been done before). This allows doubling the image size and halving the image features, to recover step by step the original size and features.

The final step of the U-Net is a 1x1 convolutional layer that the image has to go through to map each 64-component feature vector to the desired number of classes (projection layer for dimensional reduction).

It is also important to note that batch normalization was also performed during each double convolutional layer. This helps our simulation in terms of time performance and allows for regularization. Moreover, we used padding equal to 1 to obtain the same image size for the input and the output, which is not the case in Figure 1.

IV. Dataset

The original dataset is composed of 100 training images, each associated with a groundtruth mask. To use the data, dataloaders from PyTorch have been used. Thus, we created our own custom dataset class that inherits from PyTorch.Dataset module and overrides the init, len, and getitem methods. The init method reads the folder and according to the mode chosen (training or validation) gives the image ids to be used in getitem. The split ratio chosen for training/validation is 80%. Then, the getitem method allows to iterate through the data samples and read the images without storing them in the memory. At last, since the groundtruth is composed of a black and white image but with pixels values between 0 and 1, we transform it into binary values 0 or 1 before returning it for training. For this, a simple threshold of 0.5 has been used.

The presence of the validation set prevents overfitting as the model chosen will be the one that performed best on the validation set and not the training set used for learning.

V. Training

We train the model on 80 percent of the training data that were provided and we keep the resting 20 percent to have a validation set.
To train our data with the U-Net CNN, we have to specify different parameters: the number of epochs, the batch size, the loss function, the optimizer, the calculation of the error, and the learning rate.

To train the model, 100 epochs have been used because it is sufficiently large to see good results and the training can be computed in a reasonable amount of time.

The train and validation error are calculated with the F1 score (equation 1) and the accuracy (equation 3). However, we based our decision criterion when tuning parameters specifically on the F1 score, because we have unbalanced data; there is not the same amount of road as background in the images. Indeed, the F1 score takes into account both false positive and false negative and so, it is better than accuracy for evaluating unbalanced data.

$$F1\ score = 2 * \frac{precision * recall}{precision + recall} \tag{1}$$

$$= \frac{true\ pos}{true\ pos + 2 * (false\ neg + false\ pos)} \tag{2}$$

$$Accuracy = \frac{Nb\ of\ correct\ pred}{Total\ nb\ of\ pred} \tag{3}$$

VI. Hyperparameters

To tune the hyperparameters, we do not have sufficient computing capacity to cross-validate them. So, we decided to tune them one by one using grid search. Firstly, we have to tune the batch size and learning rate and then, we are going to see if adding a scheduler, a weight decay coefficient, a dropout coefficient, or using data augmentation can improve our results.

We could also have tuned the optimizer and the loss function. However, we decided to set these two parameters according to the literature due to time limitations.
For the loss, first, a binary cross-entropy (BCE) loss with logits was chosen as in paper [1]. However, after more research, we found that an IuO loss (equation 4) would be more adapted for a segmentation problem [5].

$$IoU\ loss = 1 - IoU = 1 - \frac{intersection + 1}{union + 1} \tag{4}$$

with

- intersection equivalent to true positive count, $intersection = \sum_i input[i] * target[i]$
- union is the mutually inclusive area of all labels & predictions $union = \sum_i input[i] + target[i] - intersection$

We decided to use the Adam optimizer as it is often used in image segmentation [3][4]. We are going to tune 2 parameters of this optimizer: learning rate and weight decay. The betas and eps parameters are default values : (0.9, 0.999) and 1e-08 respectively.

VII. Optimization

The strategy adopted for this project was to start with a simple Unet model and then add more complex parameters that should have a positive effect on the F1 score. So, for each hyperparameter, the best results are saved and added to the training as we go along the optimization steps. All F1 scores and accuracies in the next tables are computed on the validation set.

A. Batch size

As we were limited with the free GPU memory provided by "Google colab" (Nvidia Tesla V100 GPU), we have to choose a small batch size. Batch size 8 and 4 were still too big for the "Google colab" GPU, so we decided to take batch size 2. We had to keep in mind that the batch size influences the choice of the learning rate; indeed, a bigger batch size leads to a bigger learning rate. So, we have to pay attention to having a consistent learning rate with our batch size in order to not have bad consequences.

B. Learning rate

One of the most important hyperparameters to tune in a CNN is the learning rate. This is why it was the first one we focused on. The learning rate is generally less than 1 and greater than 10^{-6}. As we have a small batch size we can try different learning rates between 10^{-2} and 10^{-6}. Then, a value between the two best learning rates was tested to get a more precise result. The evaluation of the model with a batch size of 2 and different learning rates can be found in table I and figure 2.

Learning rate	1e-2	1e-3	1e-4	1e-5	1e-6	5.5e-4
F1 score	0.7674	0.8206	0.8278	0.7546	0.7099	0.8317
Accuracy	0.9033	0.9197	0.9281	0.8811	0.8676	0.9277

TABLE I: F1 score and accuracy on the validation set for different learning rates

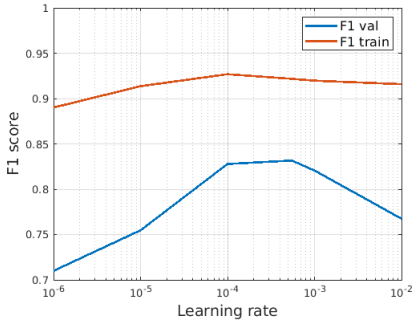


Fig. 2: F1 score in terms of learning rate

Given the results, a learning rate of $5.5e-4$ has been chosen for the rest of the optimization.

C. Scheduler

To further improve the learning rate part of the model, a learning rate scheduler can be added. This feature adjusts the learning rate based on a given number of epochs. Generally, the decrease of the learning rate as we get closer to the best result is beneficial. Here, we used ReduceLROnPlateau (from PyTorch). It allows dynamic learning rate reduction based on some validation measurements: the learning rate is reduced when a metric has stopped improving during several epochs specified by the patience argument. Models indeed often benefit from reducing the learning rate by a factor of between 2 and 10 once results stagnate.
The evaluation of the model with a batch size of 2, a learning rate of $5.5e-4$, and different parameters (the patience and the tracked metric) of ReduceLROnPlateau scheduler can be found in table II.
Given the results, we chose not to use a scheduler and to keep the learning constant at $5.5e-4$ for the rest of the optimization.

D. Weight decay

Until now, weight decay was set to zero. We now want to determine if adding this parameter helps our model performance.

Scheduler	none	p=5 on train loss	p=5 on val F1	p=8 on val F1
F1 score	0.8317	0.8283	0.8272	0.8273
Accuracy	0.9277	0.9216	0.9240	0.9176

TABLE II: F1 score and accuracy on the validation set for scheduler with different patiences (p) and applied on different metrics

Weight decay is a regularization technique that consists of adding a small penalty, usually the L2 norm of the weights (all the weights of the model), to the loss function. PyTorch applies weight decay to both weights and bias. This helps prevent overfitting, it keeps the weights small and avoids exploding gradient. The weight decay value is passed to the Adam optimizer. A little tuning is needed, thus we train our model for different values. Given the results presented in table III, we added a weight decay coefficient of $1e-7$ to our model for the rest of the optimization.

Weight decay	0	1e-6	1e-7	1e-8
F1 score	0.8317	0.8328	0.8347	0.8276
Accuracy	0.9277	0.9279	0.9271	0.9217

TABLE III: F1 score and accuracy on the validation set for different weight decays

E. Dropout

The next optimization technique that has been tested is dropout. It consists of randomly dropping out neurons in a layer with a probability that corresponds to the dropout coefficient. Here, we decided to add dropout after each double convolution of the Unet. This technique is mostly used to prevent overfitting on the train set. The results are shown in table IV. We decided to set a dropout probability of 0.25 to our model for the rest of the optimization.

Dropout	0	0.15	0.25	0.35
F1 score	0.8347	0.8316	0.8357	0.8230
Accuracy	0.9271	0.9252	0.9290	0.9213

TABLE IV: F1 score and accuracy on the validation set for different dropout probabilities

F. Data Augmentation

After choosing the right model, the next step is to see if data augmentation helps increase the performance. Only 100 samples were available in the provided dataset so data augmentation is a good way to get a more general result. To increase the dataset, new png images have been created using the original ones. However, because of our limitation with the free GPU and computation time, we were not able to add a lot of images.

First, we wanted to see if doubling the dataset size with rotated images helps increase the F1-score. The rotations are random for each image for an angle between 40 and 320 degrees. The reflect mode is used to keep the size of the image and the integrity of the road shape.

The performance of our model on the original dataset compared to the augmented one is presented in table V. With 200 images now, the script stopped at epoch 74/100 due to GPU limitations. The best result comes from epoch 69.

Then, the model was trained on a new dataset with 100 added noisy images. The noise is Gaussian with a mean of 0 and a variance of 5. With this, we can check if our model is robust to noise or low-quality images. For this training, the script stopped at epoch 79.

In table V, we can clearly see that the validation F1-score is the best for the noisy dataset. However, this is because the dataset is now completely modified and different from the test set so this model overfits the data. Hence, the test score is lower. The rotations seem to help create a more general model with less overfitting on the validation set and better results on the test set.

To choose the final dataset, we based our decisions on the test results. Indeed, the dataset on which the metrics calculation are made is different and data augmentation prevents overfitting on the validation set so the best indicator is now the test set. Given the previous results, we noted that increasing the dataset size with geometric operations seemed to be the best option to increase our model performance. We also decided to keep some noisy images to make the model more robust. Hence, our last trained dataset is composed of 100 original images, 100 rotated, 100 flipped and 50 noisy ones. This training stopped during epoch 50 due to GPU limitations with the best result at epoch 49 presented in table V. As expected, data augmentation helped get better test performance. A model trained on more images is more general and less prone to overfitting.

Dataset	original	rotation	noise	rot/flip/noise
F1 score	0.836	0.793	0.885	0.792
Accuracy	0.929	0.905	0.945	0.914
Test F1 score	0.852	0.868	0.851	0.872

TABLE V: Metrics performance of the final data augmented models

VIII. Results

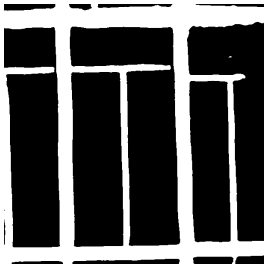
At each step of the optimization, we do a submission with the parameters that give the highest validation F1 to obtain a test metric (F1 and accuracy). Table VI summarizes the results obtained at each step of the optimization. The final model, the one that gives the best submission on AiCrowd, thus has the parameters presented in table VI. The best results are on the last line.

Figure 3a shows an overlay of the resulting mask on a test image. Figure 3b shows the actual resulting mask for a test image. These images show that our model seems to really work adequately to identify roads. Nevertheless, the resulting lines on the mask are not straight and some small patches seem to be misclassified. This could be the object of further optimization.

	Best param.	Val F1	Test F1	Test acc.
Learning rate	$5.5e-4$	0.832	0.854	0.921
Scheduler	none	0.832	0.854	0.921
Weight decay	1e-7	0.835	0.850	0.920
Dropout	0.25	0.836	0.852	0.919
Data augmentation	rot/flip/noise	0.792	0.872	0.927

TABLE VI: Results

(a) Mask overlay from test image 1 with our best model



(b) Mask result from test image 1 with our best model

IX. Conclusion

We got our best result on this road image segmentation problem with a final tuned Unet (table VI); 0.872 for the test F1 score and 0.927 for the test accuracy.

If we were not limited to the free GPU provided by "Google Colab", we could have augmented the data set with more than 350 images and we could have gone higher in the epochs to get better results. We could also have done cross-validation instead of grid search to tune the hyperparameters.

References

[1] O. Ronneberger, P. Fischer, en T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", in Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 2015, bll 234–241.

[2] C. Shorten en T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning", Journal of Big Data, vol 6, no 1, bl 60, 2019.

[3] O. Öztürk, B. Sariturk, en D. Seker, "Comparison of Fully Convolutional Networks (FCN) and U-Net for Road Segmentation from High Resolution Imageries", International Journal of Environment and Geoinformatics, vol 7, bll 272–279, 09 2020.

[4] A. Persson, "PyTorch Image Segmentation Tutorial with U-NET: everything from scratch baby". 2021.

[5] F. van Beers, A. Lindström, E. Okafor and M. Wiering, "Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation", Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods, 2019. Available: 10.5220/0007347504380445.