ML Project 2 - Road Segmentation

Report

Eloïse Doyard
eloise.doyard@epfl.ch

Cyrille Pittet
cyrille.pittet@epfl.ch

Alessio Verardo
alessio.verardo@epfl.ch

December 23, 2021

Abstract

The Road Segmentation on Satellite images is a popular computer vision problem. This paper investigates different approaches and assesses the performances of several supervised machine learning methods typically used to tackle this problem. We obtained the best results with a modified version of the Fully Convolutional Network architecture.

1 Introduction

With the advent of neural networks in computer vision, and in particular convolutional neural networks, treatment of large amount of images became an easier task. Road Segmentation of Satellite image is such an example. This task classifies each pixel as being part of a road or not. This paper is an attempt at training different types of machine learning algorithms to determine the most efficient method for this specific assignment. Namely, we developed 2 linear classifiers, 2 non-linear classifiers, an auto-encoder and 2 neural networks based on the Fully Convolutional Network (FCN) architecture.

2 Method

2.1 Baseline

The training dataset consists of 100, 400x400 pixels, RGB images taken by satellites. These images are then cut into non-overlapping patches of 16x16 pixels each, which results into 62'500 such patches. We keep 80% of this dataset for a training set and the remaining 20% for a local validation set.

2.1.1 Features extraction

As a baseline, we use simple linear and non-linear binary classifiers (see 2.1.2 for more details). To use such models, we need to represent each input as a vector of features. A simple way to do so is to simply flatten the input image into a vector. As we consider patches of 16x16 pixels, extracted from the original image, this would yield vectors of 256 features. This is quite large for simple models, which will likely overfit and would result in very slow training. For those reasons, we extract aggregated features for each patch, summarizing it. Namely we consider the 9 following features for each patch :

- Mean of each channel (RGB) in the patch (3 features).
- The log-variance of each channel, because we noticed that the distribution of the variance features are very skewed toward the left.
- Maximum value of each channel.

As the features are not in the same range of values, we also standardize each of them using z-standardization.

The training dataset, once cut into patches, contains only 19.29% of positive (i.e. road) samples. If we directly train models on this dataset without any other processing, the model will tend to only learn how to detect negative (background) samples and will have a very poor recall. To counter this, we re-weighted the samples in the loss function to give more importance to the few positive samples and thus compensate for this unbalance. Specifically, we weighted each sample by $\frac{N}{2(\#\text{ samples of class})}$.

2.1.2 Models

As a first shot, we considered 2 linear and 2 non-linear classifiers. We used cross-validation with F1-score as metric to determine a good value for the L2 regularization strength (first 3 models). The running time for the GradientBoosting was too high and we did not cross-validate the hyperparameters (number of learners and maximum depth). To implement this model we used the sklearn package. In this framework, you need to specify the inverse of the regularization strength (coefficient C) and not the actual regularization coefficient.

- Logistic Regression, L2 regularization with inverse strength of 0.1;
- SVM, linear kernel (i.e. no kernel), L2 regularization with inverse strength of 1;
- SVM, Radial Basis Function (RBF), L2 regularization with inverse strength of 10^4;
- GradientBoosting, 1000 weak estimators, maximal depth of 5.

2.2 Neural Networks

2.2.1 Feature augmentation

In order to train our Neural Network which contains a lot of parameters, we thought that only 80 images (the training set explained earlier) is not enough. To overcome this issue,

we performed a data augmentation. For each image given as train image, we output 34 new images:

- 30, 200x200 pixels, random crops of the original image which are then resized to 400x400 pixels.
- 3 rotations (by 90, 180 and 270 degrees) of the original image.
- The original image.

This gives us 2720 training images in total and 20 testing images on which we did no transformation.

2.2.2 Auto-Encoder

The features for the patches we considered in the baseline models were handcrafted and somewhat arbitrary. Ideally, we would like to learn the features that are the most useful. This can be done via a convolutional autoencoder. It is composed of 2 main parts : the encoder which embeds the patch into $\mathbb{R}^{16}$ as a vector of learned features; and a decoder which takes as input such a vector of features and reconstruct the patch image. We jointly train both so that the encoder learns meaningful features with which the decoder is able to reconstruct the patch. In other words, it simply compresses the information of the patch as an image into a simple vector of summarizing features. As cost function, we use the MSE loss to train the model.

The architecture we used is shown in figure 1 and is inspired from [3] and [4]. It takes as input a small patch of 16x16 pixels, and downsamples it using convolutions. Then it flattens the results of the convolutions into a vector which is the input of a fully connected layer. The first layer of the decoder is also a fully connected layer whose result is reshaped into 512 features maps of size 4x4. These are then upsampled using Transpose Convolutions which finally reconstruct the 3 channels patches of 16x16 pixels.

We then use the encoder only to compute the features for our training and test sets, which gives us a new matrix X where a row is the feature vector for a patch. The idea is then to simply train a SVM model on this learned dataset. We use the same kernelized SVM model as used in the baseline, namely a RBF kernel with L2 regularization and an inverse regularization strength of 10^4.

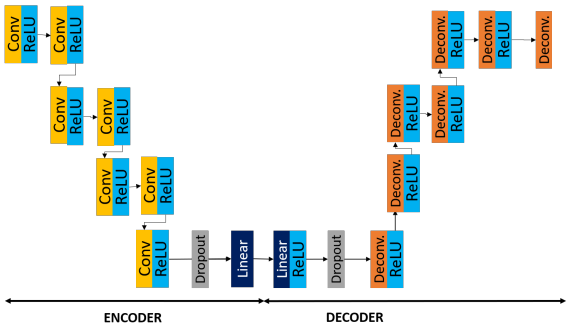

Figure 1: Auto-Encoder architecture

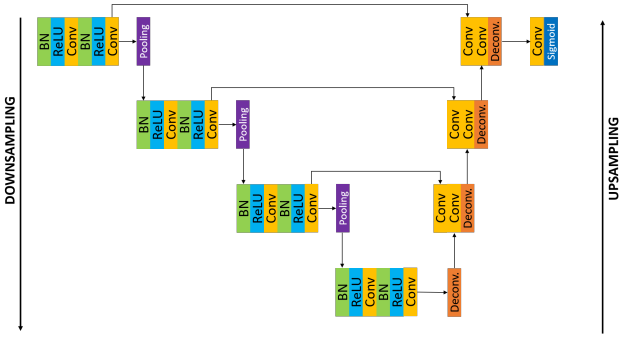

Figure 2: U-Net architecture

2.2.3 U-Net

The U-Net is a variant of a Fully Convolutional Network [7]. We constructed a 4-layers encoder part and 4-layers decoder part. The architecture we used is inspired by both [5] and [2] papers. We illustrated the architecture used in figure 2. The neural net is composed of a succession of Down Sampling blocks for the encoder part and Up Sampling blocks for the decoder part. A Down Sampling block is composed of twice the sequence of the following layers :

- Batch Norm layer that re-centers and re-scales inputs.
- ReLU layer for the activation function.
- Convolutional layer to extract features from the patch.

We finish a Down Sampling layer with a max pooling layer to reduce the size of the extracted features by 2. An Up Sampling block is composed of two convolutions layers and a Transpose Convolution one. The final layer is a 2D convolution with only one output channel and then a sigmoid activation function. We choose this architecture for the losses that expected only one input value per pixel. Thus, we output for each pixel the probability that it belongs to the class 1.

We chose 3 as input number of channels corresponding to the number of channels the input patches have : Red, Green and Blue. Regarding the argument of the number of output channels of the first layer we chose 32 for computational reasons.

2.2.4 Fully Convolutional Network

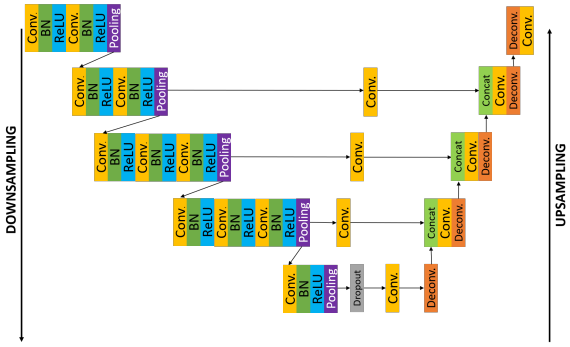

Figure 3: Fully Connected Network architecture

Mainly inspired from [5], this network can be seen as a generalization of the U-Net shown in subsection 2.2.3. Due

to computational limitations, we weren't able to reproduce exactly the network described in [5]. Figure 3 represents the architecture obtaining our overall best scores on AIcrowd and on our local test. We did not use the same architecture in all our tests, but we chose to present the most complex architecture and our most successful one at first. We can see that, as in the dowsampling part of the U-Net, we have a number of blocks whose purpose is to extract features from different parts of the image. The main differences with the previous U-net are

- We do not necessarily use batch-normalization in all our tries.
- We use dropout layers hoping to avoid overfitting.

Furthermore, we try three different values, 128, 64 and 32, for the number of output channels of the first convolutional layer. As one might expect, the higher we set that the number, the longer it took to train the network. The performance of those different architectures are reported in 3. As for the U-Net, the number of input channels is set to 3 to match the RGB channels of the original images. One could highlight the use of so-called **skip layers** in our architecture in figure 3. The objective of such layer is to preserve "high-definition" details of the image (at different stages of the encoder part) and to combine them with the up-sampled output of the encoder. Finally in the upsampling part, we implemented *fuse* layers described in [5] by concatenating the output of the two different layers and then performing a convolution on the concatenated vector of pixels.

2.3 Training

During training, we tried a number of different combinations of losses, optimizers and hyper-parameters values. In addition to the binary cross entropy loss (BCE), we tried to use the Dice Loss [8] and the Tvserky Loss [1].

3 Results

As explained in 2.3, we used in this project different losses as well as different optimizers. For the Auto-Encoder we used only the combination of MSE loss with Adam optimizer. For the U-Net and the Fully Connected Network, we train on multiple combinations of Tversky, MSE, BCE and BCELogit losses with Adam and Stochastic Gradient Descent optimizers. We report the results of those training in this section.

3.1 Baseline models

Performances of the different baseline models are shown in table 1. One can observe the gap between the linear models (logistic regression and linear SVM) and the non linear ones (SVM with RBF kernel and GradientBoosting) : the SVM with RBF having a higher recall (i.e. it classifies correctly many road patches) but a poor precision (i.e. it tends to over-classify as road). In contrast, GradientBoosting tends to miss some road patches (low recall) but is more confident when detecting a road patch (higher precision).

Method	**Local validation set**			
	F1	Acc.	Recall	Precis.
Logistic Regr.	0.39	0.64	0.59	0.29
SVM (linear)	0.39	0.65	0.59	0.29
SVM (RBF)	0.58	0.76	0.85	0.44
GradientBoost.	0.49	0.84	0.40	0.64

Table 1: Accuracy, F1-score, Recall and Precision scores for each baseline model on our local test set.

Model	**Local validation set**			
	F1	Acc.	Recall	Precis.
Autoenc.+SVM	0.49	0.69	0.77	0.36
U-Net	0.73	0.88	0.69	0.77
FCN	0.82	0.92	0.81	0.83

Table 2: Accuracy, F1-score, Recall and Precision scores for our different neural networks, using Adam optimizer and respectively the MSE, BCE and BCElogit losses.

3.2 Neural networks

The performances of our different neural networks are summarized in table 2. Overall, we can see that the Fully Convolutional Network yields the best results. This is not surprising since it is the network that we tuned the most being the fastest to train. Results for FCN are further discussed in 3.2.3.
The Auto-Encoder performs slightly worse than the baseline SVM model. Although we have no proper explanation for this result, one can suspect that the convolutional used might overfit the training data, i.e. the 16x16 patches.
Regarding the U-Net, its performances are closer to the one of our best model. Comparing the U-net with the FCN, this latter performs better with nearly the same parameters (Trial n°3 in table 3). Furthermore, FCN was almost 2 times faster to train than the U-Net. For this reason, we will explain the methodology we used on this FCN model to improve our results.

3.2.1 Auto-Encoder

The performances of the Auto-Encoder model are summarized in table 4. Compared to the corresponding baseline SVM, it performs slightly worse. A potential problem is that the Auto-Encoder overfits on the training data and do not generalize well to the test set. To mitigate this, we introduced a L2 regularization via the weight decay. We tried 2 values for the coefficient: 10^{-4} and 10^{-6}. We can see that indeed, adding more constraints on the weights improves the results. However, it is still not as good as the baseline. As the training patches contain mostly background samples (not road), we suspect that the auto-encoder prioritizes a good fit on these samples which implies less meaningful features for the road patches. A possible way to circumvent this issue would be to use side information such as the true label of the patch. This would allow us to put more weight on these samples or to incorporate this information in the output of the auto-encoder.

Trial	Loss	Comments	Local validation set			
			F1	Acc.	Recall	Precis.
1	BCE	300 epochs, $\eta = 10^{-3}$, $B = 50$, $S = (50, 0.1)$, $C = 32$, w/o first dropout, w/o batch normalization layers	0.76	0.90	0.70	0.82
2	BCE	100 epochs, $\eta = 10^{-4}$, batch size of 10, scheduler $(30, 0.1)$, $C = 64$, w/o first dropout, w/o batch normalization layers	0.75	0.90	0.69	0.83
3	BCELogit	150 epochs, $\eta = 10^{-3}$, $B = 40$, $S = (50, 0.1)$, $C = 32$	0.80	0.91	0.77	0.83
4	BCELogit	250 epochs, $\eta = 10^{-3}$, $B = 40$, $S = (60, 0.1)$, $C = 32$, one more dropout layer after the 3rd max-poling	0.78	0.91	0.73	0.83
5	BCELogit	250 epochs, $\eta = 10^{-3}$, $B = 40$, $S = (60, 0.1)$, $C = 32$, class reweighting	0.80	0.91	0.81	0.80
6	BCELogit	300 epochs, $\eta = 10^{-3}$, $B = 20$, $S = (150, 0.1)$, $C = 64$, class reweighting, regularization ($\lambda = 10^{-5}$) and new training set.	0.82	0.92	0.81	0.83

Table 3: Accuracy, F1-score, Recall and Precision scores for the Fully Convolutional Network on our local test set. *Parameters : $S = (\alpha, \beta)$ the scheduler - we multiply the learning rate by a factor β every α epochs- ; C the number of output channels after the first convolution in the network ; B the batch size and η the learning rate.*

Method	Local validation set			
	F1	Acc.	Recall	Precis.
No WD	0.47	0.69	0.72	0.35
WD of 10^{-6}	0.46	0.68	0.71	0.34
WD of 10^{-4}	0.49	0.69	0.77	0.36

Table 4: Accuracy, F1-score, Recall and Precision scores for the SVM with RBF kernel, inverse regularization coefficient 10^4 (as for the baseline) and features obtained from the Auto-Encoder with different values for the weight decay (WD) coefficient on our local test set.

3.2.2 U-Net

The performance of the U-Net network are reported in the table 2. As one can observe the performances of the models were not as high as expected using the BCE loss and the Adam Optimizer. Moreover, with this network , we tried to implement the Tversky loss, modifing our original architecture and using the parameters from [1], but the results were not conclusive.

3.2.3 Fully Connected Network

We will now give a higher focus to the different results obtained with this network. As said earlier, we try a lot of different versions of this network but we only described one architecture in 2.2.4. Table 3 shows our different results, the modifications from the architecture shown in 2.2.4 and parameters used for training.

One thing to notice is our exclusive use of Adam optimizer in our training. Indeed, our scores obtained with SGD were lower than the ones with Adam.

Another remark is that we present here only the results obtained with two classification losses : BCE and BCELogit. The differences between the two are the following : BCE needs as input values values in the range from 0 to 1 whereas BCELogit will apply the sigmoid step internally; and the BCELogit loss allows for a reweighting of the samples with respect to their class for imbalanced dataset.

One could notice how dropout layers and batch normalization layers improve the performance of our neural network most likely by reducing the overfitting (Trial 3 compared to trials 1 and 2). But, on the other hand, adding too many dropout layers prevents the model from generalizing well (Trial 4 vs 3).

Reweighting the classes increases slightly the results using our standard architecture (Trial 5). That is why we only reweight the positive sample (the road pixels) by $\frac{\text{Nb negative samples}}{\text{Nb positive samples}}$. Finally, one can observe that with this model, it seems that we reached the limit of optimization as we never managed to surpass those 0.80 of F1-score .

One hypothesis could be that the training set contain many horizontal and vertical roads and not enough diagonal ones. To overcome this issued, we tried to reduce the number of random crop to 10 and to generate multiple rotations with different not 90-multiples angles of each image, crop it at the center and rescale it to its original size. By setting $C = 64$ and using weight decay, we managed to improve the F1 by almost 2% and the accuracy by 1% w.r.t. to our previous best score (Trial 6).

4 Conclusion

We explored a variety of different models and assessed their performances on the road segmentation task. We started from simple linear classifier models based on hand-crafted features and gradually increased the complexity of our models. Our best results were a obtained using a CNN with skip layers. To improve our performances, one could try to apply post-processing to smooth the detected roads [6] or to use patches larger than 16x16 pixels but smaller than the images (for instance, 72x72 pixels) with the objective to reduce the number of pooling layer we use.

References

[1] 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)*, pages 683–687. IEEE, 2019.

[2] Tamara Alshaikhli, Wen Liu, and Yoshihisa Maruyama. Automated method of road extraction from aerial images using a deep convolutional neural network. *Applied Sciences*, 9(22):4825, 2019.

[3] Golnooshsadat Elhami and Romann M. Weber. Audio feature extraction with convolutional neural autoencoders with application to voice conversion. 2019.

[4] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. pages 373–382, 10 2017.

[5] Pascal Kaiser, Jan Dirk Wegner, Aurélien Lucchi, Martin Jaggi, Thomas Hofmann, and Konrad Schindler. Learning aerial image segmentation from online maps. *IEEE Transactions on Geoscience and Remote Sensing*, 55(11):6054–6068, 2017.

[6] Mullany Lila. Smoothing semantic segmentation edges. `https://towardsdatascience.com/smoothing-semantic-segmentation-edges-8b9240052904`. Accessed: 2020-12-15.

[7] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[8] Jerin Paul. Segmentation of roads in aerial images. `https://towardsdatascience.com/road-segmentation-727fb41c51af`. Accessed: 2020-12-13.