Machine Learning Project: Road Segmentation

Skander HAJRI, Guillaume MOLLARD, Adrien RUAULT
EPFL, Switzerland

I. Introduction

Machine learning regression techniques were implemented in order to classify pixels of satellite images acquired from GoogleMaps as road or background.

Our task was to train a classifier using a set of 100 pairs of satellite/ground-truth images of dimensions 400 x 400 pixels to segment roads in these images, i.e. assigns a label road=1, background=0 to each pixel. We then tested our model on the data provided on https://www.kaggle.com/c/epfml17-segmentation which consists of 50 satellite images of dimensions 608 x 608 pixels. The testing of the classifier was carried out on labeled patches of 16 x 16 pixels.

The classifier developed in the frame of this project is a Convolutional Neural Network (CNN) for pixel-wise prediction. The network is strongly inspired from the U-net configurations from [1] and [2]. The model was implemented and trained on TensorFlow.

II. Related Work

There exists many deep learning techniques applied to image segmentation, one article that summarizes and reviews many of them is [3]. Before the advent of deep learning techniques the standard for image segmentation was to use decision tree based techniques. Then came the CNN era which was first characterized in an attempt to carry out image segmentation by patch labeling. However the latter class of models was limited in a sense that it fails to capture the image features on a large context due to the restricted size of the patches. Nowadays the most efficient models are inspired from the Fully Convolutional Network (FCN) proposed by [4]. The model developed in this project can be considered as a variant of FCN.

One of our starting points was paper [5] about Aerial Image Segmentation provided by *OpenStreetMap* and *Google Maps*. Another deep learning model using satellite images segmentation is [2]. It uses different convolutional networks for each class of item they try to identify using a modified U-net architecture, we tried to do the same to have our model distinguishing between routes and background. As a basis for the architecture of our model, we also carefully studied the schema shown in [1] in order to implement and graphically represent our network in a clear and precise fashion.

III. Method

A. Preprocessing of the data

As the orientation of satellite images is not relevant for road segmentation the training set provided by kaggle has been enlarged by rotating the satellite/ground-truth images by 90 degrees. It multiplied the size of the dataset by four. This technique allows to generate more data and is a good solution against overfitting. However the generated images are redundant, they thus bring multicollinearity to the training set. Indeed they do not bring as much information as true new satellite images. Finally the pixel intensities were normalized in order to take values between 0 and 1 for every images.

B. Feeding of the neural network

In order to feed the neural network, satellite images of the training set had to get loaded into Tensorflow tensors, each of dimensions [batch size, 400, 400, 3]. The fourth dimension corresponding to the number of channels, which is 3 since our satellite images are in RGB format. The training groundtruths were also loaded into tensors of shape [batch size, 400, 400, 2], the fourth dimension being a 1-hot representation of the labels (background or road). Every batch is composed of a random combination of images. This shuffling allows the optimizer to take different directions while descending the gradient.

C. Network description

The purpose of this section is to detail the layers of the U-net used to produce the segmentation maps from the input satellite images. The whole network is summarized as a schema in Figure 1.

We begin with a 400 x 400 image with 3 channels and then convolve two times with a 3 x 3 filter of depth 64. This produce an activation map of the same size but with 64 channels. From there we pool the image, down-sampling it to half of its previous height and width and keeping the same depth. We then reproduce the same series of convolutions and down-sample the results five times, until we get an activation map of size 13 x 13 x 64. At this point, the receptive field reached is of size 345 x 345, which allows the neural network to learn from large patterns of the input image.

Thereafter, we start a succession of up-sampling using a 2 x 2 filter to convolve on the activation map and append a copy of the corresponding down-sampling result to this one. We therefore get a map with 128 channels that we reduce down to 96 and then to 64 again by convolution, the operations are repeated until we get back an image that has the same size as the input image. The final step consist in convolving the last result with a unit filter that will produce a 400 x 400 x 2 segmentation map. Finally a softmax function is applied to each entry of the map in order to obtain the pixel-wise probability prediction of each label.

Each convolution applied in the present network is carried out in several steps. Firstly a batch normalization is applied to

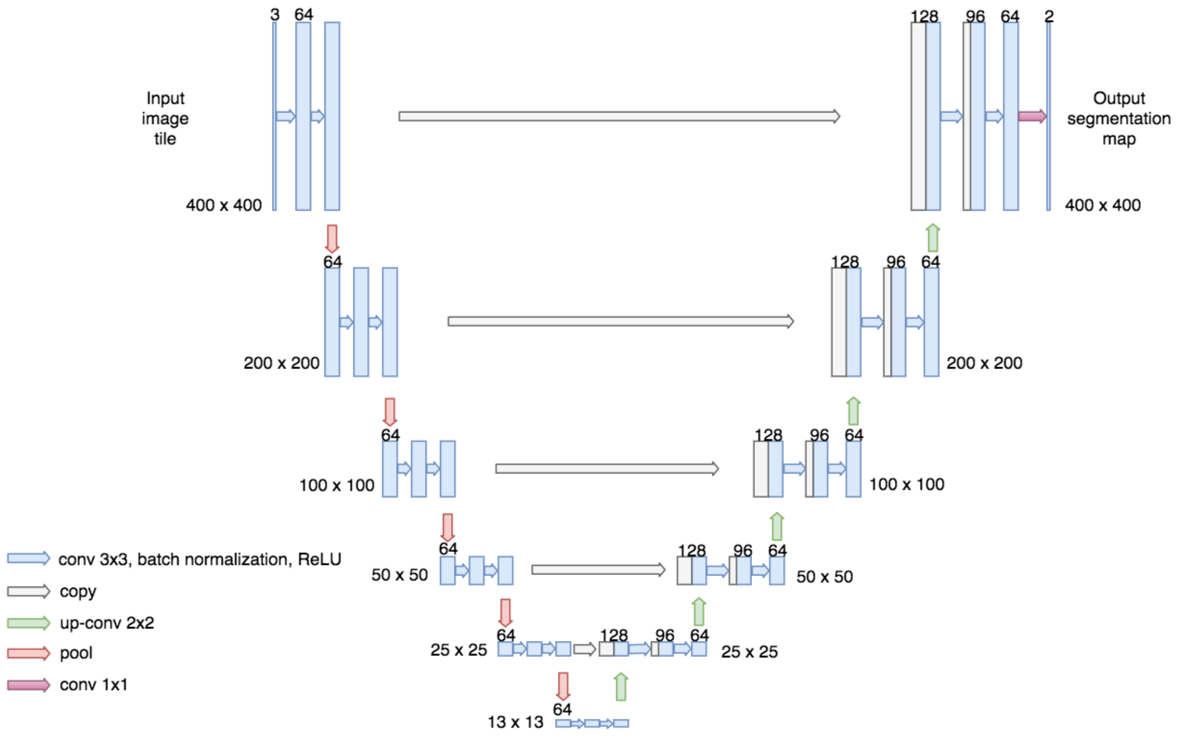

Fig. 1: U-Net Convolutional Network

the input of the convolution. It allows the input to have zero mean and unit variance. Batch normalization helps dealing with badly initialized weights. It is quite helpful in our case since the model is not pre-trained. The convolutional filter is then applied to the output of the batch normalization. Finally in order to deal with overfitting a dropout has been added to the output of the convolutional filter. This dropout operation is actually never really used in our case for the reasons explained in section III-E. Note that the batch normalization and the dropout operations are not applied when using the model for prediction. Those two operations are only used for the purpose of training.

D. *Loss function*

The loss function used to optimize our classifier was a cross entropy loss function computed pixel-wise and then averaged over all the pixels of a batch. The cross entropy loss function is a standard choice for learning a neural network because it addresses the learning slow-down that occurs when the parameters are badly initialized.

E. *Addressing overfitting*

One of the most important pitfall of machine learning and especially of neural networks is their tendency to overfit the training data. Indeed if care is not taken the model can perfectly learn the training data without being able to generalize.

In order to avoid overfitting as much as possible some techniques have been implemented in the TensorFlow code. Sadly the loss-regularization and the dropout operations are barely used due to a lack of parameter tuning. Indeed because of our limited computation resources it turned out to be very hard to carry out a true validation. The only parameter that was investigated under validation is the number of epochs on a limited part of the training set (see section IV-D). Below is recap of all the measures taken against overfitting that were implemented in this project.

- A regularization term has been added to the cross-entropy function that is based on the L-2 norm of the weights and biases of the network.
- A tuning has been carried out on the number of epochs as described in section IV-D
- A dropout operation have been implemented after every convolution. The dropouts are actually never really used in our submissions since their keep rate is set to 0.
- The batch normalization of the input of every convolution adds regularization to the data.
- The flipping of the training images in order to generate new data helps against overfitting

Those techniques would show most of their interest with greater computation resources and if the model was improved as recommended in section V.

F. *Optimization*

1) Optimizer: The optimizer we have chosen for minimizing the loss function is the Adam optimizer [6]. This solver was chosen firstly because it requires few parameter tuning. Indeed it computes adaptive learning rates for every parameter of the model. This was quite important in the frame of this project given our limited computation possibilities. Moreover the Adam optimizer is known to be quite aggressive in descending towards a minimum.

The Adadelta optimizer [7] was also tried. However it turned out that it was much slower that Adam optimizer to minimize our loss function. We thus decided to keep the Adam optimizer for its efficiency.

2) Batch size: A batch size of one image was attempted in the very beginning of our investigations. It turned out that it was not a good choice because the loss struggled to decrease significantly during the optimization. We assume that it was due to a limited number of gradient directions offered by the dataset. We thus used a batch size of 2 images to offer more possibilities of descent directions to the solver. It significantly improved the efficiency of the optimization. No other batch sizes have been tried due to the limited memory of our computation hardware.

IV. Results

All the computations described in this section were carried out with the CPU version of Tensorflow running on an Intel Core i5 7th generation. Training the model for 1 epoch with 400 images takes about 1h30.

A. Converting test images to correct format

In order to compute our predictions on the *Kaggle* test set images some pre-processing had to be carried out. Indeed the model was trained on 400 x 400 images whereas the test set images were of size 608 x 608. To deal with that issue the test set images were divided in four 400 x 400 images on which the prediction was carried out. The probability maps have then been merged by averaging the probability predictions in overlapping area.

The prediction images showed some boundary effects after merging. It was due to the fact that the 400 x 400 images did not capture the whole context of the 608 x 608 images. The four images have then been cropped on the side they share with the others images in order to reduce this effect.

B. Descriptive analysis

Two predictions made on the train set are displayed in Figure 2 and 3. In Figure 2 we can observe that the classifier does very well in predicting straight road networks. We expect the classifier to be able to detect cross roads and to interpolate roads from them.

In Figure 3 we can still remark the ability of the classifier to detect straight road networks. However it struggles resolving highways. We believe that it is because highways shows less structural characteristics at this level of zoom. Then the classifier has less clues available to clearly classify it as a road.

C. Final submission

The characteristics of the final submission chosen on kaggle are described in table I. This submission yielded a F1-score of 0.92772 on the 50 % unveiled test set of Kaggle.

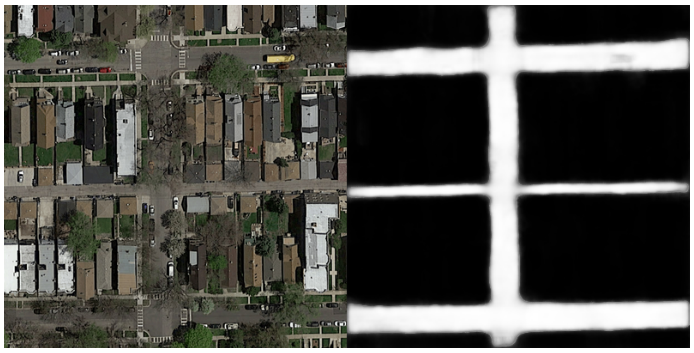
Fig. 2: A prediction after 7 epochs on the train set of 400 images along with its groundtruth.

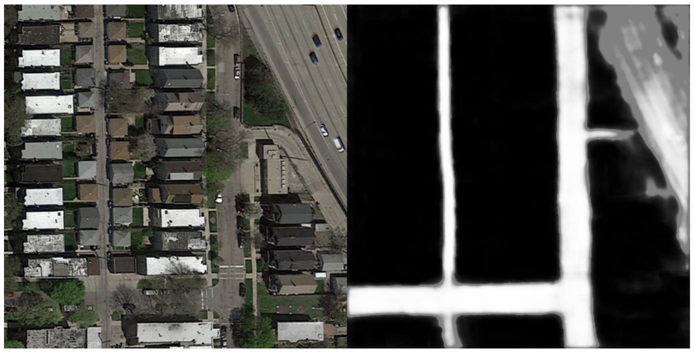
Fig. 3: A prediction after 7 epochs on the train set of 400 images along with its groundtruth.

Training size	400 images
Batch size	2 images
Number of epochs	9
Regularization parameter	1e-15
Dropout	0 %

TABLE I: Characteristics of final submission

D. Validation

Very few parameters were tuned in our model because of the limited computation resources available. The only parameter that was tuned was the number of epochs. To do so we dedicated 80 % of the initial 100 images provided by Kaggle to the purpose of training and the 20 % remaining for the purpose of validation. Despite the validation was not run on the 400 images that constituted our training set, it helped us estimating the number of epochs to be run before the model overfits the data.

Since it appeared on the graph 4 that for a training set of 80 images, we reached a plateau of low loss after approximately 20 epochs, we made the rough estimation that we could feed the neural network with 80 * 20 = 1600 images without taking a risk of overfitting. We thus trained the model on 4 epochs with the full training set of 400 images being pretty confident about the fact that our neural network was not overfitting the training data. We then ran the epochs one by one, trying to find the lowest loss, but it appeared that we were as well in

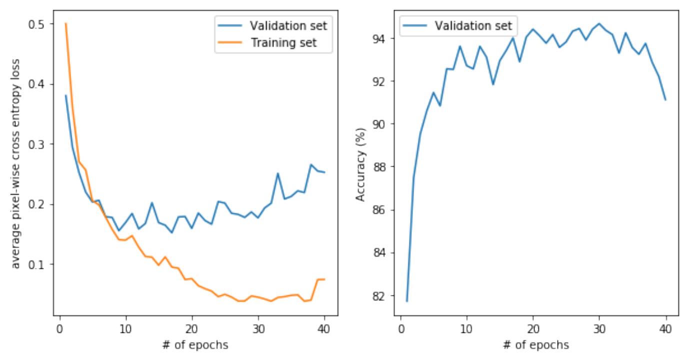

Fig. 4: Accuracy and loss obtained running 40 epochs with a training set of 80 and a validation set of 20 images.

the "plateau" showed on the figure 4, and our submissions on Kaggle for the following epochs gave approximately the same results.

V. Potential improvements

One of the main weakness of the actual model is that it has not been subject to parameter tuning due to non sufficient computing power. Indeed, regularization and dropout parameters were not tuned in order to limit the overfitting behavior, while this seems to be a problem we are actually facing. Also, the depth of the layers have been set to 64 for the entire network, restricting the complexity of the patterns the network can learn. With more computation power, it would have been possible to increase the depth of these layers, allowing the neural network to learn more complicated patterns, and to avoid overfitting choosing adequate values for the dropout and the regularization parameters.

With more computation power, it would also have been possible to increase filter sizes, to allow the neural network to learn more complicated correlations between pixels. Our whole neural network actually has filter values of 3 x 3 or 2 x 2, but it could be increased to 5 x 5.

Since our training set is limited in size, one other way to boost the performance of the model would have been to perform a pre-training of the model with a standardized dataset like the Visual Object Classes Pascal dataset [8], as recommended in [5] and in [3]. The pre-training would allow the neural network to recognize basic image features before starting the training on the real training set, and thus help it to be more efficient during the training and more robust regarding overfitting.

An other improvable point in our pipeline is the merging of the images 400 x 400 into one of 608 x 608. In several groundtruth predictions, we observe a "loss of context" in the drawing of the road lines, which makes the road disappear at the limit between two concatenated images. Maybe a simple down-sampling of the test images from 608 x 608 to 400 x 400 would provide better results.

VI. Conclusion

A classifier for pixel-wise road segmentation has been developed in the frame of this project. The predicter is a CNN adopting the U-net configuration. The model was trained on 400 RGB images of size 400 x 400 pixels. It showed quite good results on the kaggle test set with a F1-score of 0.92772 after a training of 9 epochs on 16 x 16 pixels patches.

The model did show a little bit of overfitting with the number of epochs. It could therefore be improved by tuning the parameters of the dropout operation and the regularization. However our limited computation resources did not allow us to properly tune those parameters.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation." [Online]. Available: http://lmb.informatik.uni-freiburg.de/http://lmb.informatik.uni-freiburg.de/people/ronneber/u-net.

[2] Arkadiusz Nowaczyński, Michał Romaniuk, Adam Jakubowski, Michał Tadeusiak, Konrad Czechowski, Maksymilian Sokołowski, Kamil Kaczmarek, and Piotr Migdał, "Deep learning for satellite imagery via image segmentation." [Online]. Available: https://blog.deepsense.ai/deep-learning-for-satellite-imagery-via-image-segmentation/

[3] A. Garcia-Garcia, S. Orts-Escolano, S. O. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A Review on Deep Learning Techniques Applied to Semantic Segmentation."

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[5] P. Kaiser, J. D. Wegner, A. Lucchi, M. Jaggi, T. Hofmann, and K. Schindler, "IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING Learning Aerial Image Segmentation from Online Maps."

[6] D. P. Kingma and J. L. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION."

[7] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," 2012.

[8] Mark Everingham (University of Leeds), Z. Luc van Gool (ETHZ, Chris Williams (University of Edinburgh), John Winn (Microsoft Research Cambridge), and Andrew Zisserman (University of Oxford), "The PASCAL Visual Object Classes Homepage." [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/