Machine Learning - Road Segmentation

Cyril Vallez, Darius Nik Nejad, André Langmeier
Machine Learning (CS-433) : Project 2

***Abstract*—This report presents our strategy and implementation of a so-called convolutional U-net to perform road segmentation on satellite images. It is a variant of a network (used for biomedical image segmentation) presented by Ronneberger et al. [1], in which the output image has the same number of pixels as the input. The architecture of this network captures proximity as well as context of the input image. With the methodology explained in this report, we finally obtained a F1 score on the testing set of 0.891 on the submission website AIcrowd.**

I. Introduction

Given a set of colored satellite images, as well as a set of groundtruths where each pixel is labeled either as road or background, the objective is to train a model which predicts where are the roads on some different satellite images.

The training set for this task contains 100 colored images (RGB) of 400x400 pixels, with the associated groundtruths where road pixels are white, and background pixels are black. The testing set contains 50 colored images of 608x608 pixels. The goal is to assign a label to each pixel of the testing images, classifying them as either road or background.

II. Simple models

As a first step, we first used three very simple models to run the segmentation problem. The first one is the logistic regression as seen in class applied on 16x16 pixels patches, otherwise the linear layer would contain too many parameters. The second simple model is made of 2 convolution layers, each followed by a ReLU activation function, and 2 fully connected linear layers, still applied on 16x16 patches. The third model is just a 3D alternative to the convolutional network by stacking filtered images in the third dimension of the 16x16 patches. Because of this cropping, all models' predictions hence lack of context of the full image. It has trouble recognising objects, and is systematically fooled by local obstruction such as cars, trees, buildings, etc. Moreover, the groundtruth is also be computed as a mean on the patches, and hence we loose a lot of local information about the borders of the roads.

On the other hand, since the evaluation of the model on AIcrowd is made on 16x16 patches, it has the advantage of being trained the same way as it is going to be evaluated. The results were not great and a better architecture was thus needed.

III. U-Net

After some research, we chose to implement a fully convolutional neural network called U-Net. It was first presented by Ronneberger et al. [1] in their work on segmentation of biomedical images, as described by figure 1.

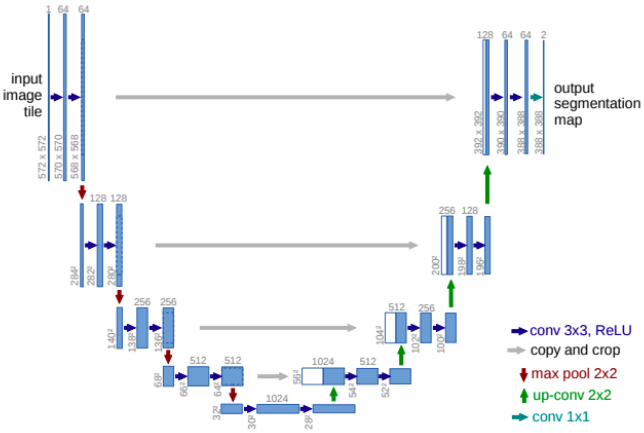

Figure 1: *Architecture of the original U-Net [1]*

The architecture consists of a contracting path to capture context (left of figure 1) and a symmetric expanding path that enables precise localization (right of figure). The contracted path consists of the repeated application of two unpadded convolutions with kernel size of 3x3, each followed by a ReLU activation function. Then a 2x2 max pooling with stride 2 is applied for downsampling, and the number of channels is doubled. Each step in the expanding path consists of an inverse convolution with kernel size 2x2 and stride 2, a concatenation with the corresponding feature map from the contracting path, and two unpadded convolution with kernel size 3x3, each followed by a ReLU. Finally, a 1x1 convolution is used to map each component of the feature vector to the desired number of classes, in our case 2 (for either road or background).

The network we used was made of 4 steps in the contracting path, and (obviously) equal number in the expanding path. It was not possible to use more, since the 2x2 max-pooling operations must be applied to a layer with an even x- and y-size, and we have images of 400x400 pixels. The images from the training and testing set have 3 channels (RGB images). Without any features enhancement (through filters, see the next sections), the input of the neural net is then of the form $(N, 3, W, H)$ and the output of the form $(N, 2, W, H)$, where N is the batch size, W and H are respectively the width and height number of pixels. In other words, we have two logits per pixels associated to each class (road or background).

As already mentioned, and contrary to the original U-Net, our variant of the U-Net has the same number of pixels as output as for input. This is useful since the training and testing images have different dimensions. Indeed, our way allows to

feed the pre-trained network with the testing images directly, without having to do any resizing.

IV. Data Processing and Augmentation

One main observation when having a look through the training and testing data sets is that, the training set contains roads that are mostly vertical and horizontal, whereas the testing set contains many diagonal roads as well as one big diagonal highway. To resolve this lack of training data, we applied different rotations of 15°, 45°, etc. to the training set. The *SciPy* library has a function that fills the corners of rotated images using mirroring of the image's edges. The obtained images highly resemble true images with diagonal roads (see figure 2). Flipping the training images vertically and/or horizontally was also used to augment the size of the training set. The total size of the training set N is then $N = N_0 \cdot r \cdot f$, where $N_0 = 100$ is the original set size and r (resp. f) is the number of applied rotations (resp. flips).

Figure 2: *Illustration of the rotation with edges mirroring : left : initial image, right : rotation at 45°.*

The other approach we had concerning the data processing is about filters. The *Python Imaging Library* (PIL) allows to apply filters to the images such as edge enhancing, contrasting, sharpening or blurring filters. Those filters allow to enhance the features that can enter into a model. For this project, three filters have been used : 'edge', 'edge+' and 'unsharp'. They are illustrated in figure 3. We hoped those filters would help the computer detecting the roads.

V. Models Training and Selection

We implemented three different variants of the U-Net described above. The first one is the simplest and corresponds to what has been explained in section III, we will call it U-Net2D. The second model takes as input the original image and the filtered images in the same channel as RGB, in other words it takes as input tensors of shape $(N, 3\cdot(1+F), W, H)$, where F is the number of filters added as features. This model follows the exact same steps as U-Net2D, with just more channels. We will call it filtered U-Net2D. The third model takes as input the original image and its filters again, however the filters are labeled in an added dimension : the input is of shape $(N, 3, F, W, H)$. This model applies the same convolutions but in 3D (instead of 2D) with respect to the 3 dimensions F, W, H. This last model is called U-Net3D.

Figure 3: *Original image (top left) with 3 filters: 'unsharp' (top right), 'edge' (bottom left) and 'edge+' (bottom right)*

A. Learning setup

We used the *PyTorch* library to define and train the networks. All models were trained on *Google Colab* in order to have access to decent GPUs, which are essential for the training not to be infinitely long. However, *Colab* restricts access to GPUs after some simulation time, so it was sometimes very tedious and annoying to be able to run long training. Moreover, *Colab* does not provide the best GPUs in the market, and computational time were still very long. It imposed us to be a bit more restrictive in the number of simulations to do.

B. Optimizer's choice

To choose appropriate optimizers, we performed several tests on a U-Net with the whole training set without any augmentation, i.e. 100 images. This set has been separated into 75 images for the training set and 25 images for validation. The evolutions of the accuracies for each optimizer on the validation set for 30 epochs and with the Cross-Entropy Loss are shown on figure (4).

From these results, one can deduce that the SGD must not be considered, and that Adam, AdamW and RMSprop are of equivalent efficiencies. Therefore, we decided to use RMSprop for the U-Net2D and in addition we added a momentum to this optimizer so it would present less fluctuations. For the models that asked more computational time like filtered U-Net2D and U-Net3D, we chose the AdamW optimizer as it would converge faster than RMSprop with momentum. For both optimizers, the learning rate has been set to 10^{-3} and the regularization term (weight decay) to 10^{-8}.

Moreover, we used the Cross Entropy loss, with weights for some simulations in order to take into account that the training

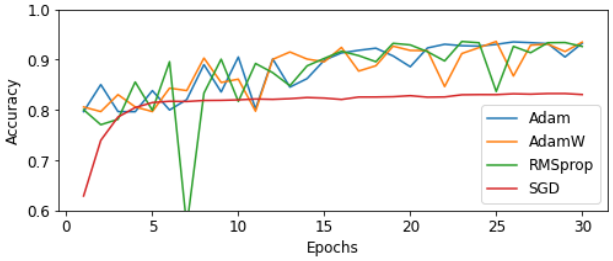

Figure 4: *Accuracies on the validation set for different optimizers.*

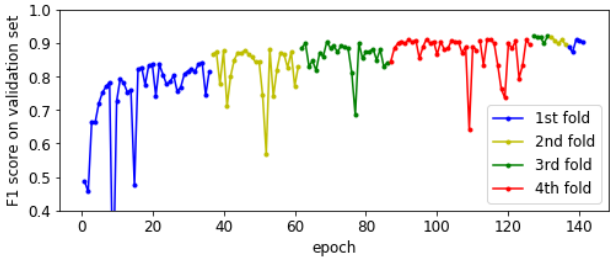

Figure 5: *Evolution of the computed F1-score for each fold.*

set is unbalanced (there is way more background than roads, approximately 20% roads and 80% background). However, the difference in accuracy and F1 score was almost nonexistent when using weights or not.

C. K-fold cross training

In order to avoid overfitting, we always kept a part of the full training set as validation set. However keeping a part of the data out of training makes it less efficient as well. To solve this issue we decided to train all our models with k-fold cross training. k-fold cross training consists of separating the training set into k categories and then training the model leaving one of the categories as validation set. At this point, there are two options : either training a model from scratch for each fold (so training k models from beginning), or using the same model for all folds, without reinitializing it each time. In both cases, the model is saved after a certain number of epochs, and then trained (either from scratch in the first approach, or from the previous last epoch in the second) with another category of the total training set (and saved somewhere else). So on, until all categories have been validated. One also comes back to an already validated fold as validation set, if its corresponding model needs to be improved (fig.5). In the end, one ends with k models (k independent models in the first case, and k models corresponding to different instances and number of epochs of the same model in the second case).

We guessed that training independent models each time would be better, but decided to test both approach either way. Moreover, training from beginning each time takes a very long time. So in the end, we believe the second approach is still a good compromise, because when switching from a fold to another as validation, the weights might drive away from a "bad" local minima they were stuck in.

To establish a prediction given an input image, the usual method consists of averaging the predictions given by the application on this input with each of the k models. However, we took a slightly different approach. We defined a parameter we called sensitivity, which is an integer between 1 and k. Per pixel, it corresponds to the minimal number of roads the k models need to predict for this pixel to be considered as road. It can go from 1 (only one model finding a road at a pixel is enough for it to be classified as a road) to k (all the models need to predict a road to classify this pixel as a road). To choose the appropriate sensitivity, one can choose the one that maximizes the F1 score (or the accuracy) on the validation part of the training set.

VI. Results and predictions

Table (I) shows the results and scores we obtained from the submission platform for different versions and models.

As already mentioned, the U-Net we implemented applies a pixel-wise classification of the images. Some examples of the final predictions on the testing set are shown on figure (6).

Some remarks are in order here. First, images (6a) and (6b) demonstrate that the network performs extremely well on images that look like the images from the training set. Moreover one is able to see that the network did learn how to recognise diagonal roads, as was intended when adding rotated images to the training set. Figure (6c) shows how the model is destabilized when images start to differ from the "classic" images from the training set. Indeed, Image 12 depicts what seems to be an industrial zone (or maybe even some kind of waste ground on the bottom left), where everything is grey, and contrast is very weak. Even for a human, it can be unclear what is a road, what is a parking or what is just dirt. For those reasons, it is no surprise that the model struggles way more to predict this image correctly (especially considering that it did not train on images of this kind). Figure (6d) illustrates the fact that the network is not foolproof, and that it is hard for it to predict complex streets structure.

However, even if we predict pixel-wise the classification of roads, the submission system (AIcrowd) only accepts patch-wise prediction, i.e it only accepts 1 label per 16x16 patch. Figure (7) shows the patch-wise prediction from the pixel-wise prediction of figure (6). The patch-wise prediction is computed using a threshold of 0.25 : if the mean of the

Model	R	F	CV	Fil	F1 score
2D U-Net	×	×	×	×	0.856
2D U-Net	✓	✓	×	×	0.866
2D U-Net	✓	✓	✓	×	0.881
2D U-Net	✓	✓	✓	✓	0.889
3D U-Net	✓	×	×	✓	0.865
3D U-Net	✓	×	✓	✓	0.873
Mix of models	✓	✓	✓	✓	**0.891**

Table I: Results with different models: **R** adding rotations to the training set; **F** adding flips to the training set; **CV** 4-fold cross validation; **Fil** adding filters

(a) Image 37 (b) Image 48

(c) Image 12 (d) Image 3

Figure 6: *Selection of pixel-wise prediction from the network (prediction as red overlay)*

(a) Image 37 (b) Image 48

(c) Image 12 (d) Image 3

Figure 7: *Selection of patch-wise prediction from the network (prediction as red overlay)*

pixels is above this threshold, we assign a label 1 (road) to the patch, otherwise we assign 0 (background). Obviously, the result seems way less smooth. Edges of the roads are not well captured this way, and diagonal roads cannot be properly taken into account by square patches.

So we believe our pixel-wise approach is cleaner than the patch-wise format required by AIcrowd, but we had to stick with it.

The final model that has been used for our final submission on AIcrowd has been a mix of models. This process is performed by applying the available already trained models to a validation set and add the predictions for all combinations of models. The obtained tensor thus contains values ranging from 0 to the number of models considered. Then, a threshold sensitivity is set. Above this threshold, a pixel is predicted as road, otherwise it is predicted as background. This parameter is also varied for all its possible values and for all combinations of models. At each pair "models-threshold", the F1-score is computed, which gives information about what combinations perform the best at predicting roads.

Based on this process, our best combination appeared to be the mix of the two 4-folded U-Nets in two dimensions, i.e. the one described in section III and the one with filtered images in the channels. It achieved our best F1-score with a sensitivity of 4 on a validation set of 500 images. The validation set has been created from the training data and rotations were applied so that the models used had not been trained on the exact images of the validation set. This mix of models finally achieved an F1-score on the test set of **0.891**.

VII. Possible improvements

Here are a few points that could have improved our predictions.

First, as always in machine learning, improving the number of original images in the training set could have improved our models a lot by giving them more material to learn.

As the computing time was a limiting factor when training models, one could also add that an access to a more powerful GPU without restrictions could have allowed to train models for bigger and more complete training sets.

The use of more advanced models could also predict roads in a more efficient way. However this type of models is usually even heavier (computationally speaking) than our U-Net, and therefore they are unusable in our project.

VIII. Conclusion

We used a very powerful method for image segmentation. In the end, even if the model is not foolproof, it has a very great idea of where the roads are on the satellite images from the testing set, achieving a F1-score of almost 90%.

However, the difficulty of having easy access to GPUs greatly restrained us in all our process of tests and parameters tweaking once we had the general model defined.

References

[1] O. Ronneberger, P. Fischer, T. Brox, 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Available on : https://arxiv.org/pdf/1505.04597.pdf