Machine learning applied to Road Segmentation

Leonardo Aoun, Romain Artru, Thiercelin Marin
IC Faculty, EPFL, Switzerland

***Abstract*—The following report explains how we used various machine learning techniques to train a classifier to segment roads in a set of satellite images acquired from GoogleMaps.**

I. The linear approach

At first, we tried to follow the hint given in the statement of the project: extract features out of the images and then train a linear classifier to recognize the roads.

A. Extracting features

From the images, we had to find different ideas to extract features that could be relevant to train the model. We used some of the given method to extract two features concerning the color of a patch: one for the mean of grey and one for the variance.

Then, since we should never encounter a chunk of road alone, but as a part of a network, we added a feature that was equal to 1 if and only if one of the neighbor of the concerned patch had the same mean of grey than him more or less his variance.

B. Evaluating the quality of a model

We had to create a method to check the quality of a model to compare the different types of regression that we tried to use afterwards. To do so, once our classifier is trained, we take an image from the train data and its ground truth, extract the features (X) from the image and predict the categories with the classifier. For each patch, we then have a) the ground truth Y_{gt} (0 if it's not a road and 1 if it is) and a predicted value Y_p (0 or 1). We finally obtain the % of good guess by applying the following formula:

$$percentage = 1 - \frac{\sum |Y_p - Y_{gt}|}{length(Y)} \tag{1}$$

C. Selecting the model

To create the model, we had to choose a kind of regression. We used the library suggested called sklearn. This library propose a set of methods implementing the various regression seen during the lectures and also additional ones. The ones we tried are the logistic, the ridge and the Bayesian ridge regression. Of course, each of those models depend on some parameters that we had to optimize. To do so, we looped over different ranges of values for each parameter to find the best one. We found as best results :
- 0.577225% of correctness for the logistic regression
- 0.618110% of correctness for the Bayesian ridge
- 0.618114% of correctness for the ridge regression

As expected, we can see that the ridge and Bayesian ridge regression are very close, but only permit a very bad rate of correctness. So we tried another approach seen in the lectures to get better results: to use neural networks to solve the road segmentation problem.

II. Convolutional Neural Networks

After some research on Computer Vision, we found out that Deep Learning and Convolutional Neural Networks is the state of the art these days. Although more complex to implement, these techniques significantly improved our results.

A. Classification NN

We first continued with the idea of teaching a model to classify patches. We built a network that takes as input patches of width and height $16 * (2n + 1)$. So basically the network takes a patch, along with regions around it and decides if it represents a road or not. So we built the model represented by the graph in figure 1. It takes a patch and passes it through a few convolutional layers and then flattens it and uses 4 fully-connected layers to output 2 nodes: One being the probability of the pixel representing a road and the other of it representing a background. We then apply a categorical cross-entropy loss to this last layer and uses Adadelta by default as an optimizer, which in addition to Stochastic Gradient Descent dynamically adapts its parameters over time.

To predict an image we pad it we 16n pixels on each side and then predict each crop to decide the label of the center $16 * 16$ pixels. We did the padding to be able to predict the border patches, we chose the reflective padding so pixel $(-i, -j) = (i, j)$ and not just a black pixel. We justify this by assuming that a road that is going outside of the image will probably continue a little bit more.

We were getting pixelized images as expected, which is not a problem since the submissions will be done on patches of $16 * 16$. However, we were getting discontinued roads especially when trees or cars cover parts of them.

B. Semantic Segmentation

By researching about the problem we understood that this task is more of a semantic segmentation task. Roads were probably getting discontinued because the pixels in

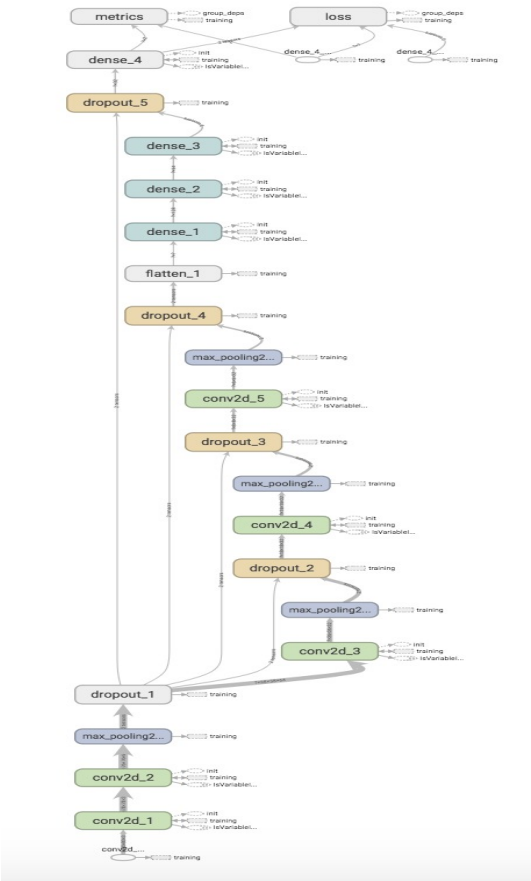

Figure 1: Graph of the CNN model

the intermediate tensors don't have a big enough perceptive field.

Networks like Unet or SegNet offer a solution for this by using encoding layers that reduce the dimension of the input image by using a technique called MaxPooling. This technique takes neighbouring pixels of the feature map and selects the largest value to pass to the next layer. This will reduce the size of the image, and allow the kernels of the deeper layers to have more knowledge about what's happening in a larger entourage around the pixel.

1) Unet: Unet is a model developed for Biomedical studies at the University of Freiburg for detection of cells in tissue images like in figure 2. By examining the results in [1] it seemed to us that our task is very similar so we implemented a similar model but with less filters per layer, a couple of layers less in order to fit the model in our memory.

The interesting part about the Unet architecture 3 are the concatenation steps represented by the gray arrows, which combine the results of the decoding with the non-encoded tensor to attenuate the possible loss of information that might have occurred in the process. However this had to augment the size of the model a lot compared to our memory

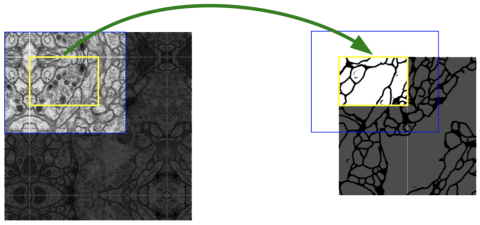

Figure 2: Detection of scar tissues using Unet

limitations and reducing the size further was not leading to results as good as those of the next model.

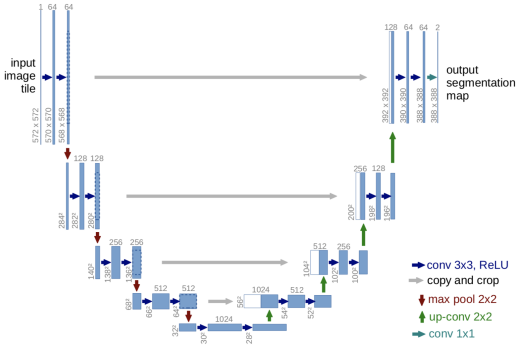

Figure 3: Unet Architecture

2) Segnet: Segnet [2] is another CNN model we experimented and gave us better results than Unet. The demo given by the University of Cambridge researchers is showcasing the results for road predictions so this pushed us to experiment with it. We didn't need 12 labels as in their case, 2 were enough so our last layer had to be of depth 2. At first, we did not want to have a network as deep as theirs and only constructed 3 encoding/decoding blocks instead of 5.

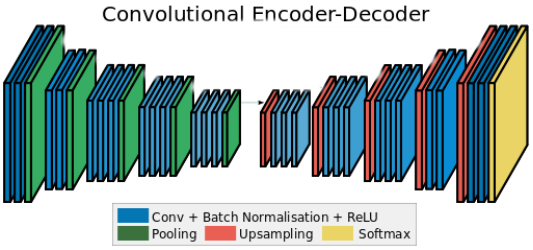

Figure 4: Convolution Encoder-Decoder

We reused the architecture presented in the Segnet Paper [2] but we used a regular upsampling function instead of the one used in the original segnet, that reused the indices of the maxpooling downsampling layers to perform "symmetric" upsampling. Even though this upsampling technique could have increased the learning pace, the actual implementation would have occupied to much of our time and we chose

to use one of the upsampling layers provided by keras, and it didn't seem like that much upsampling precision was needed for our task, especially since the prediction was being evaluated on patches of size $16 * 16$. Although this was already an improvement compared to our previous results, we still had some discontinuity in the roads, similar to the case with the classifier. Most probably this is due to the usage of just 3 max pooling layers, since each kernel is of width 3, doing 3 pools of size (2, 2) would give a perceptive field of width $3 * 2^3 = 24$ per pixel of the feature maps in the center of the network. Which is less than the crop size we were giving to the patches.

Our hypothesis to why it was still giving us better results is that the kernel weights are being adjusted during the back-propagation in a way to minimize the loss for the whole image and not just a crop (or a batch of images and not a batch of crops) so keeping the image together was beneficial to us.

In order to actually try having 5 encoding/decoding blocks, we had to minimize a lot the number of filters per layer, and at the last encoding layer and first decoding layer, we experimented with a kernel width of size 5, giving us a perceptive field of width $5 * 2^5 = 160$ pixels for each pixel in the central feature maps. Having less feature maps per convolution was allowing our training to go faster even though we had a deeper architecture so that was also a plus.

C. Combination

The successful results we got using the last model inspired us to try combining this with our previous classifier by training a model that takes as input the results of the classification net and corrects it using the fully convolutional one. The results were not as good as just our last implementation so we didn't pursue it further.

We think it's because the classifier is already losing a lot of data that our fully convolutional network would have preferred to have.

III. Avoiding Over-fitting

A. Data Augmentation

After trying out several learning methods, we noticed our models had trouble finding roads that were diagonal through the image. This may have been caused by the fact that the majority of roads in the training set were either horizontal or vertical straight lines. To correct that we decided to artificially augment the training set. We created copies of the training images and then we randomly transformed them using rotations of angle $\in [\frac{-\pi}{4}, \frac{\pi}{4}]$, image flip (horizontal or vertical), and zoom-in that keep at least 75% of the original image.

The data augmentation made our models more resistant to rotation. As we can see in the images 5 the model was able to better predict roads in diagonal with this data augmentation. Although a 3-fold data increase was shown useful for the model, more increasing didn't improve the results enough too justify the additional memory usage. We believe this is because our transformed images are not too different from each other and at some point the data becomes redundant.

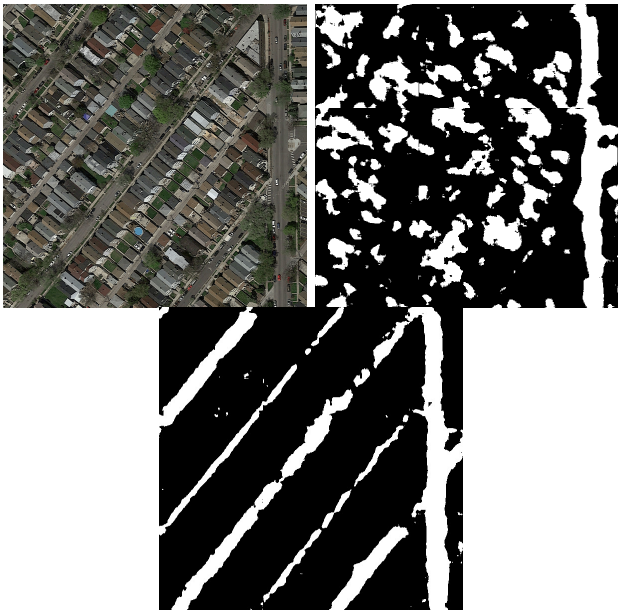

Figure 5: Prediction before and after data augmentation

B. Dropout Layers

Even after the data augmentation, the pixel by pixel accuracy was way higher for our training dataset than for our validation set. Therefore we decided to add dropout layers to make the model more robust. We set a dropout layer of probability 0.1 after each pooling layer and before each upsampling. The per pixel accuracy got a bit higher but the larger improvement was seen on the patch accuracies.

IV. Training

We trained the model on an Nvidia GeForce Titan Z with a memory of 6GB for a total of 2 hours and 20 minutes.

A. Learning Parameters

We used the Keras implementation of Adadelta to optimize the training. We kept the default values for the decay, rho and epsilon parameters but wanted to have control over the learning rate. We started the training with a learning rate of 1, and on numerous runs we were getting blocked at some loss value, so we were advised to Keep Calm and Lower Our Learning Rate.

We saw that Keras has a callback function called ReduceLROnPlateau which divides the learning rate by a certain factor when a specified metric stops going below a certain value. For the first 168 epochs the learning rate didn't change, then we ran the model again for another set of 168 epochs, manually changing the learning rate to 0.5

and after 78 of them, the callback got called three times as shown in Figure6.

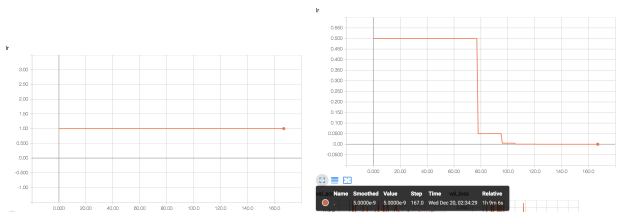

Figure 6: Reduction of the Learning Rate across the training

B. *Results*

These changes to the learning rate helped our model go a long way while still improving. Once it got to $5*10^{-4}$ it was clear it was not going to learn much more if we left it. As we can see in Figure 7 the change in learning rate helped the model get back to speed when the training stagnated. Moreover, the graphs in Figure 8 show that whenever the accuracy was stagnant and the learning rate got divided, it improved in the following epochs. Please note that the curves are smoothed by a factor of 0.6. The loss was actually going lower than 0.01636 before adding the dropout layers, but that was a result of over-fitting. The accuracy we are plotting on Tensorboard is just the per pixel accuracy we were calculating, different than the one per patch accuracy on the Kaggle Submission.

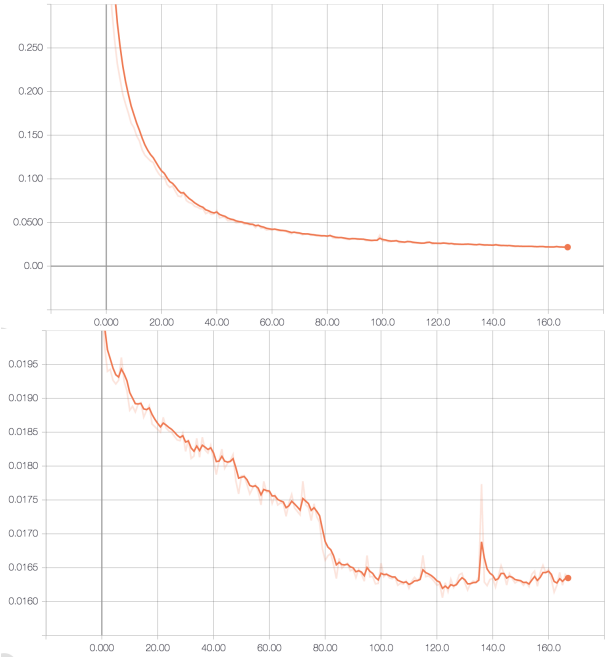

Figure 7: Loss over Epochs

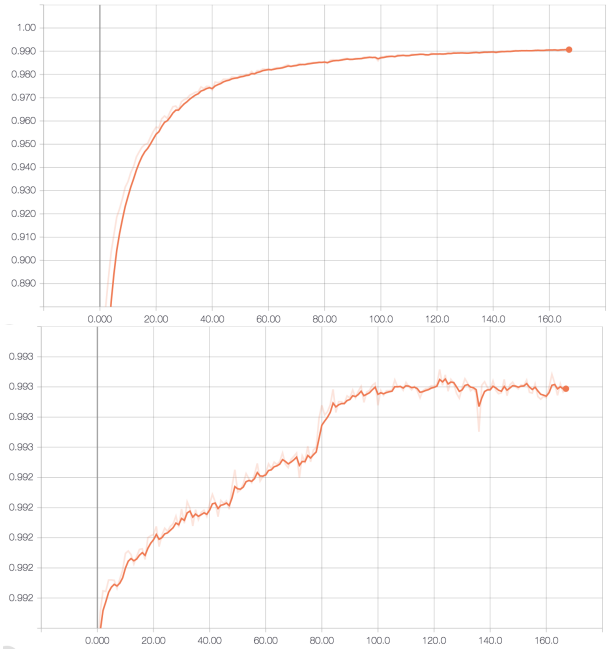

Figure 8: Accuracy over Epochs

REFERENCES

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015. [Online]. Available: http://arxiv.org/abs/1511.00561