Machine Learning Project 2 (CS-433)
Road segmentation challenge

Nicola Ischia
Computational Science and Engineering
nicola.ischia@epfl.ch

Marion Perier
Life Sciences Engineering
marion.perier@epfl.ch

Leonardo Pollina
Life Sciences Engineering
leonardo.pollina@epfl.ch

***Abstract**: The aim of this Machine Learning project is to perform image segmentation in order to distinguish the streets from the background in aerial pictures taken from a satellite. Therefore, a model performing classification was built to segment these images into street areas and background areas. CNNs architectures were shown to solve this task with satisfactory results. Parameters of CNN architecture were tested to reach the optimal architecture and to avoid overfitting. Data augmentation, image processing and postprocessing were key implementations to improve the accuracy of the model. Our final model gave a F1-score of 0.86 on CrowdAI.*

I. Introduction

Image segmentation is applied in this project to distinguish the streets from the background. Background areas mainly include grass, houses, fields, parking lots and train lines. Classification was performed by building a model to segment these images into street areas and background areas.
Performance of the models was evaluated by scoring the predictions with F1-score. This metric is based on precision and recall, thus it mainly takes into account the positive labels (composed by the number of false and true predicted positives). Thereby, true negatives are not considered with this metric. It makes sense in the road segmentation task, since we are more interested in predicting the location of streets (positive labels) than background (negative labels).
This report presents our solution approach for the road segmentation task. Materials on which the project was based on are presented in Section II. Section III-A provides explanation of the data augmentation we had to implement, whereas Section III-B details how images were processed. The models we tested are presented in Section III-C. In Section IV the main results obtained will be shown and they will be discussed in Section V.

II. Materials

Our dataset consists of 100 training images and 50 test images. The training set is composed of 400×400 RGB images, while test images are bigger: 608×608. Since the ground truth images for the training set were provided, it is a Supervised Machine Learning task. Ground truth images are composed of white and black pixels, where the white ones represent the street and the black ones represent the background. By simply looking at the training images, see Figure 1, it is possible to notice that the streets are in general straight (no curves are present) and that a lot of them are perpendicular. In addition, it is important to take into consideration that some trees might cover some parts of the streets, which might be an issue for our classification.

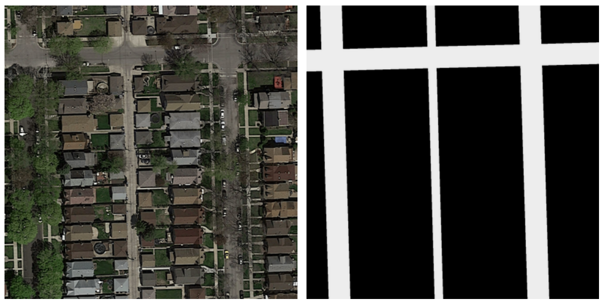
Figure 1. A train image (left) and its corresponding ground truth image (right).

In addition, our project was performed using *Python v. 3.6.5*, *Numpy v. 1.14.3*, *Scipy v. 1.1.0*, *Scikit-learn v. 0.19.1*, *Keras v. 2.2.4*, *TensorFlow v. 1.12.0.*

III. Methods

A. Data Augmentation

One of the first problem encountered in the frame of this project was the small size of the training set (only 100 images). Moreover, we noticed that there were only few diagonal streets in the training set, which could have been an issue for the prediction of this kind of streets. Thus, data augmentation was performed using the following approaches:

*1) **Flipping**:* Up-down and left-right flips were applied to samples in order to obtain a bigger training set. It is important to mention that the two flips were never applied together, since the overall transformation would have been the same as applying a 180° rotation.

*2) **Rotation**:* At the beginning of the project and again with the purpose of increasing the number of data available, simple rotations of 90° each were applied. Indeed, this kind of transformation is easier to implement since the size of the image does not change. However, the use of rotation with different degrees than 90° could improve the results by creating images with diagonal streets and improve the robustness of the model. Thus, the implementation of such transformations was performed. In this case, the samples were previously padded, then rotated and finally cropped to get the final desired size.

*3) **Contrast & Brightness**:* The last approach used to augment the dataset was to change the brightness and contrast of the samples. Although the images (the training ones as well as the test ones) come all from the same satellite and have barely the same contrast and brightness, a change of these properties might help to distinguish the features that do not depend on brightness and contrast but still play an important role in image segmentation (e.g. edges). In addition, this approach could be useful for generalization of the model to test images from a different satellite. The contrast was stretched up to a factor of 0.4, while the brightness was augmented/diminished up to a value of 0.5.

B. Image processing

This section explains how our dataset was processed and manipulated in order to improve our performance. The procedure used was the following:

*1) **Patches**:* The most naive way of dealing with the images would be to pass as input every single pixel to the model. However, it was quite intuitive to work with patches, that are small squares containing many pixels, and not with the whole image to avoid GPU memory issues. Patches were chosen to be 16 pixels-size for both satellite and their corresponding ground truth images. To assign a label to the patches in the ground truth images, the average of pixel values within each patch was computed. When the average was above 0.25 (which corresponds to a patch with more than one fourth of white pixels) the patch was labelled as white (street). Note that the centers of the patches were randomly chosen, allowing partial overlapping of different patches. This augments the robustness of the model with respect to shifting.

*2) **Windows**:* Although the patches method already helped to take into consideration how neighbouring pixels ”behave”, surroundings of these patches also bring information about common patterns. This idea was implemented using windows enclosing patches. The procedure applied was first to select a patch at a random position in the figure, and then a window of a fixed width was cut around the patch. The model was fed with these windows, and trained to be able to well predict the label of the central patch only. Window sizes (called $input\_size$ in the code) of 48, 64, and 80 pixels were tried and compared.

*3) **Padding**:* To be able to crop windows around patches placed on the borders of the image, the padding of the original image was necessary. Since our specific task benefits of good symmetry qualities due to the fact that the street are always straights and often perpendicular, mirror padding was preferred to zero padding.

C. Models

In the frame of this project we decided to focus on *Convolutional Neural Networks* (CNN) models, which are the state-of-the-art models for image recognition and segmentation [1]. However, in order to get a baseline model's performance, the simpler *Regularized Logistic Regression* model was implemented.

*1) **Regularized Logistic Regression**:* is a binary model which uses the *sigmoid* function to map a regressed function to the space between 0 and 1 values. Thus, it actually gives a value which can be interpreted as the probability to belong to a specific class. A regularization term was used to avoid overfitting. Hyperparameters, λ for the regularization and the optimal degree d for feature expansion, were optimized using a grid search algorithm with a 4-fold Cross Validation.

*2) **CNNs**:* represent the best type of Neural Networks to perform a task of image recognition. As specified by the name, the peculiarity of this model is the presence of convolutional layers. Their purpose is to extract complex features from the original image by applying several convolutional filters (of different sizes) and hence creating the so-termed feature maps.
CNN models are characterized by its architectures. Instead of implementing an architecture from scratch, we decided to look for famous architectures and to try to adapt them to our road segmentation task. The VGGNet architecture [2], which was implemented for the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) and represents a good feature extractor, was used as starting point for our project. Naturally, for reasons of computational cost and complexity, we reduced the number of blocks and changed some hyperparameters of the original architecture.

Feeding the network: The techniques described in Sections III-A and III-B led to a huge amount of data, we hence decided to implement a dynamic version of them using a *Mini-batch generator*. This generator allowed the random selection of patches' center, randomly applying rotations, flips and changes in contrast and brightness only once the patch was chosen. This process slows down the training procedure but has some interesting insights. First of all, it saves memory space by applying these techniques only on selected patches instead on the whole dataset of images. Secondly and most importantly, it allows random selections improving substantially the training set dimensions. This last feature also helps reducing overfitting.

Epochs/Steps/Batch size: The number of epochs, the number of steps per epoch and the batch size were fixed. One epoch is generally defined as a full training cycle, that is a cycle in which the entire dataset is passed forward then backward in the Neural Network (NN). However, both the size of our training set (considering all the possible windows cut in each image), along with the fact that the *Mini-batch generator* is selecting data augmentation parameters in a random manner, resulted in an entire dataset too huge to feed the NN. Hence, we arbitrarily selected the batch size and the steps per epoch, trying to have a high number of samples at each epoch.

Filters: Several sizes and numbers of filters in the convolutional layers were tried in order to find an optimal architecture. Generally speaking, the number of filters were always a power of 2 (32, 64, 128, 256) and their sizes was chosen among low odd numbers (such as 3x3, 5x5 and 7x7).

Optimizer: Instead of using the SGD optimizer as in the aforementioned VGGNet, the Adam optimizer was employed [3].

Activation function: These types of functions are used to introduce non-linearity in the network. The *ReLU* activation function $f(x) = max(0,x)$ was employed. *ReLU* was preferred to *sigmoid* because it could lead to a faster convergence of the back-propagation algorithm. For the sake of comparison, *Leaky-ReLU* was also tried with $\alpha = 0.1$.

Pooling: Pooling layers were applied after some activation layers to downsample the local image and to reduce the spatial dimension. With a 2x2 filter size and a stride of the same length that convolves around the image, both the dimension in length and in width were decreased by a factor of two. This helps reducing the computational cost and controls overfitting. To follow the idea of the VGGNet architecture, pooling layers were chosen as maxpooling layers.

Regularization/Dropout: The last technique to avoid overfitting was to add regularizers and dropout layers. The $L2-norm$ regularizer was tried. Dropout layers were used in several architectures and different dropout probabilities were tried ($p = 0.25\&0.5$). The network containing no dropout at all ($p = 0$) was tested as well.

Output layer: For what concerns the output layer, two different configurations were tested: one neuron and two neurons. In the first case *sigmoid* activation function was used, while in the latter *softmax* was applied.

Residual Neural Network (RNN): A RNN was also implemented. This model allows skipping some layers, hence creating sub-architectures inside the same one.

Data partition: The implementation of a k-fold cross validation was not convenient to implement because of the high computational cost of the training. Thus, a simple splitting into train(80%) and validation(20%) sets was performed. It was then possible to assess locally the performance of a model and the convergence of the training (stopping and restarting the procedure after some epochs), without resorting to the CrowdAI set.

D. *Post-processing*

A post-processing was applied to further improve the F1-score achieved with the prediction. The idea was to change wrong labels taking into account their surroundings. For instance, when a background patch was completely surrounded by street patches, it was switched to street. The same principle was used for street patches alone in a background area. In addition, we also tried to connect patches of the same street. This was possible thanks to the specific symmetry of our task. Indeed, most of the streets are just vertical and horizontal, hence allowing an easy and naive implementation. However, it is important to notice that the post-processing we applied could also lead to some new mistakes (where in the original prediction a patch had been well predicted). Thus, several threshold values were tested and only the best trade-off between the number of errors added and the corrected patches was kept.

IV. RESULTS

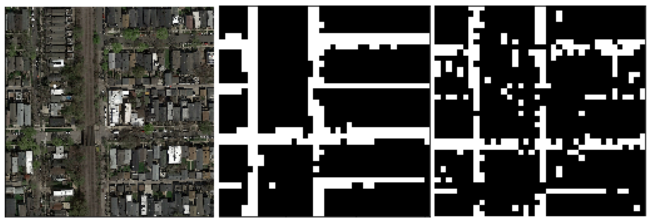

Figure 2. Satellite image and its corresponding predictions for CNN models with and without windows strategy

Figure 2 shows that predictions without using the windows strategy around the patches are quite sparse, being very different from the expected ground truth image. The prediction image shown in the center gives more suitable results as it takes into consideration the surroundings of patches. It confirms the choice of using the windows strategy for all the followings CNN architectures.

Model	F1-score
Regularized Logistic Regression	0.601
CNN_1 + W	0.821
CNN_1 + W + F&R	0.833
CNN_1 + W + F&R + D ($p = 0.25$)	0.829
CNN_1 + W + F&R + D ($p = 0.5$)	0.837
CNN_1 + W + F&R + D + C(= 0.25)&B(= 0.3)	0.846
CNN_1 + W + F&R + D + C(= 0.4)&B(= 0.5)	0.831
Comparison of W_size (= 48)	0.801
Comparison of W_size (= 64)	0.846
Comparison of W_size (= 80)	0.837
Comparison of activation function (ReLU)	0.846
Comparison of activation function (Leaky-ReLU, $\alpha = 0.1$)	0.813
CNN_1 + Above best combination + Residual connections (RNN)	0.806
CNN_1 + Above best combination ($\rightarrow$ sigmoid)	0.846
CNN_2 + Above best combination ($\rightarrow$ softmax)	0.853
CNN_2(with ReLU) + W(= 64) + F&R + C(= 0.25)&B(= 0.3) + postprocessing	**0.86**

Table I
Models and their corresponding performances.
Legend: CNN_1 = CNN using a neuron in the output layer, CNN_2 = CNN using two neurons in the output layer, W = windows, F&R = Flips&Rotations, D = Dropout, C&B = Contrast&Brigthness

The performances of the models we implemented are reported in Table I. It shows the improvements we applied at each step of this report and that brought us to our final best model. Note that validation performances are not reported

because they were corresponding to the improvements obtained with the F1-score on CrowdAI. Thus, only CrowdAI results from models trained on the whole dataset are reported in this table.
Regularized Logistic Regression method was confirmed to be the baseline model as it gave the lowest F1-score (0.603). The F1-score with CNN method was significantly higher than the Logistic one, which explains our choice to optimize this type of model.
The final best model is set with (hyper)parameters listed in Table II. The network architecture is a CNN with the layers described in Table III. Data augmentation was applied with random degrees of rotation, flips, contrast stretch up to 0.25 and brightness augmentation/diminuition of magnitude up to 0.3. Post-processing was also performed on the prediction obtained from the CNN model. Figure 3 shows predictions before and after post-processing. A street patch completely surrounded by background patches was corrected and street were linked in the case of background patches mistakenly predicted. Finally, the F1-score obtained on CrowdAI was 0.86.

Parameter	Value
Batch size	128
Steps per epoch	250
Epochs	40
Learning rate	0.001
Threshold patch	0.25
Window size	64
Activation function	ReLU
Dropout probability	0.5
Optimizer	Adam
Maxpooling	(2,2)
Prediction function	Softmax

Table II
Table showing the major (hyper)parameters of the final model.

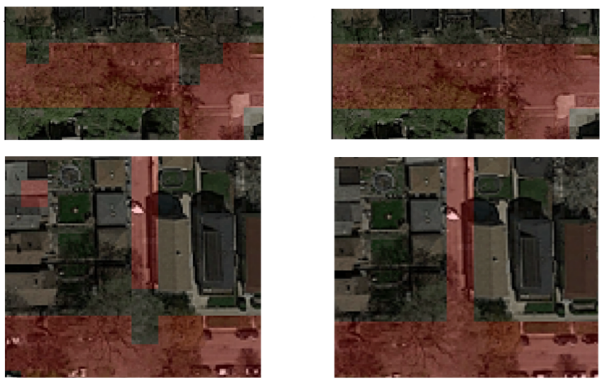
Figure 3. Segmentation of two images from the test set, zoomed, before and after postprocessing

V. Discussion

As it has been shown in section IV, the various steps we implemented allowed us to improve the final performance of

Layers	**Parameters values**
Input	64×64×3
Convolution + ReLU	32 3×3 filters
Convolution + ReLU	32 3×3 filters
Maxpooling	2×2
Convolution + ReLU	64 3×3 filters
Convolution + ReLU	64 3×3 filters
Maxpooling	2×2
Convolution + ReLU	128 3×3 filters
Convolution + ReLU	128 3×3 filters
Convolution + ReLU	128 3×3 filters
Maxpooling	2×2
Fully connected + ReLU	1024 neurons
Dropout	$p = 0.5$
Fully connected + ReLU	512 neurons
Dropout	$p = 0.5$
Output with Softmax	

Table III
Architecture of our final model.

our model. In particular, it is interesting to notice that both the data augmentation and the addition of dropout layers helped to achieve a better result. This is what was expected, since these strategies are known to reduce overfitting: the first providing more data and the second increasing the capacity of the network to generalize. The fact that patches were taken along with their surroundings showed to be essential. However, from the comparison between the sizes of the window it seems that there is a trade-off in the distance that has to be taken into consideration for the prediction of the central patch. This was shown by the fact that the best performance was achieved with a window size of 64, while both sizes 48 (too close) and 60 (to far) gave a lower score. The final F1-score was satisfactory, especially because predictions were correct even for limit cases. For instance, patches with trees covering the streets were well predicted as streets most of the times. In addition, train lines showing brightness and contrast similar to streets were well predicted as foreground. However, the improvement of the results was limited by the computational cost which was quite high and which was a main issue for testing several models. An increasing in the number of epochs and steps is not thought to be useful to boost our results, since the convergence was already reached. Indeed, by looking at the F1-score after each epoch, it was possible to see that the performance reached a plateau already after 40 epochs. This means that it is the architecture itself that should be changed in order to achieve a better performance. A larger and deeper architecture such as U-Net [4] could be employed. Finally, it is important to notice that our segmentation task was quite specific, since streets were almost always straight (vertical, horizontal and diagonal). This means that the model could not be adapted to more sophisticated road segmentation tasks, in particular in the case where the streets are curved and tangled.

REFERENCES

[1] M. A. Nielsen, "Neural networks and deep learning," 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[3] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[4] P. F. Olaf Ronneberger and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *MICCAI*, 2015.