Machine Learning Project
Road Segmentation

Maxime Schoemans
Pedro de Tavora Santos
Yannick Paul Klose
Department of Computer Science, EPFL , Switzerland

***Abstract*—The aim of this project is to perform road segmentation of satelite imagery by means of identifying in a picture what is road and what is background on images of urban areas. Several approaches are tested, all based on neural network architectures, such as fractal neural networks and U-nets but, in the end the U-net presented the best results. In order to improve our model we performed data exploration, image expansion and post image analysis to finally achieve an F1-score of 97% on the training set and of 89.8% on the test set.**

I. Introduction

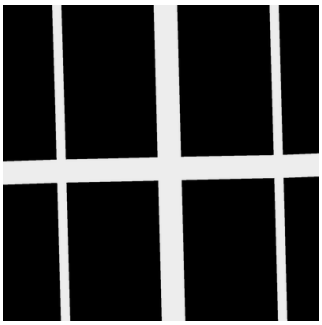
With the rise of computing power in the last few years and the ability of exploiting massively parallel computation with GPU's, image classification and segmentation has become a very hot topic among machine learning circles. This technique not only involves associating a type of image to a label (i.e images of cats to the label 'cat') but also more complex tasks such as identifying objects in images.
Nevertheless image segmentation continues to be a very complicated task, since the information is organized in a geometric way, so the algorithm needs to take into account their morphology, which leads to very complex algorithms. In this vein of thought there have been made new techniques such as *convolutional neural networks* which by means of sparse connections and weight sharing greatly reduce the complexity of the problem.
The aim of this project is to build a model that is able to perform road detection/segmentation on satellite images, more specifically we need to train a model that can classify roads from background. This report provides an overview of the models considered/used when trying to solve this problem. Section II presents our first exploration of the dataset and possible approaches to take, section III introduces the main criteria used to compare the performance of the elaborated models, section IV describes the aforementioned models and their detailed architecture, section V describes the implementation details of our chosen final model and finally section VI describes the end-results of this experiment.

II. Data exploration

The provided dataset consists of 100 satellite images of urban areas with a size of 400x400 pixels. In addition to every satellite image there is a ground truth counterpart in which the white pixels represent roads and the black pixels background (figure 1). Based on this dataset the task is to identify roads from background and afterwards test a model on a given test dataset consisting of 50 608x608 images.
At first glance we can observe that this task is not trivial. There are multiple objects blocking the roads such as trees, cars and bridges. In addition there are asphalt elements that are not considered as roads such as parking lots and sidewalks which may confuse the training model. For these reasons we can infer that the classifier should take into account some sort of context: it should look at neighbouring pixels and infer about the classification of the set of pixels being analyzed.

In order to do the classification based on the previously mentioned context we sliced the given 400x400 satellite image into smaller patches. We initially chose to do the classification on patches of 16x16 pixels, where the label of the patch will be assigned to background (pixel value = 1) if the average value of the ground-truth label is greater than 0.25 and to road (pixel value = 0) otherwise. The given threshold of 0.25 represents the average percentage of white pixels (road segments) in the ground-truth image patches. Later in this report we will refer to this patch size and evaluate whether it is better to use a pixel wise classification.

(a) Satellite image



(b) Ground truth image

Fig. 1: The given Dataset consists of a satellite image and an equivalent ground truth image

III. Benchmarking criteria

To compare the performance of our different models we will take two different criteria into account: accuracy and f1-score.

The accuracy represents the ratio of correctly predicted observations to the total number of observations. This measurement is a very simple and intuitive way for measuring the

performance but it only works well for symmetric datasets. This means that the ratio between false positive, where the model predicts true but the actual label is false and false negatives, where the model predicts false but the actual label is true, are almost the same. As we observed in Section II our dataset is not symmetric, since the average amount of pixels per image being identified as background is larger than the amount of pixels being identified as road. For this reason we additionally use the F1 score in order to measure the performance of our models [1].

IV. Models and Methods

In this section we will introduce our models and methods used for this segmentation task. As described in section III we will compare our models based on the two criteria: accuracy and f1-score.

A. Basic Convolutional Model

To have a first reference we run the test model given to us as a basis to start our project, which uses a two layer convolutional neural network with softmax loss and 32 filters on the first layer, which got an accuracy of 60% and an f1-score of 55% on the training set. Since this result was not satisfying, the model was build upon in order to achieve the desired performance. Like in the next studied model, this one uses the patch based method in order to perform classification.

B. Fractal Model

Our first attempt to improve this model was by using more convolutional layers embedded into a fractal structure (figure 2). The fractal model can be subdivided into different blocks, that contain convolution and joining layers. Between each block there is a pooling operation that reduces the input image based on the pooling size. The characteristic of the fractal model is the arrangement of the convolution and joining layers in the blocks, which can also be seen in figure 2.

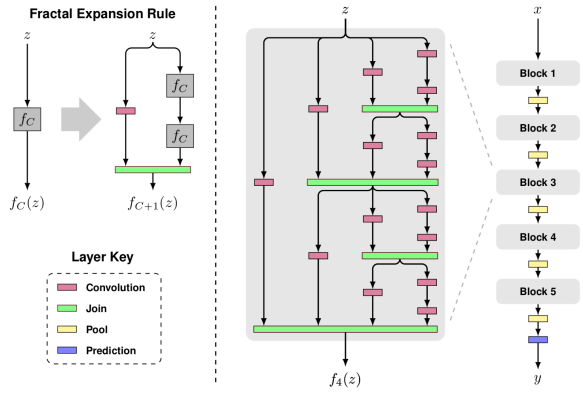


Fig. 2: Fractal architecture [2]

The model we used consisted of four blocks each with a three convolutional layers sequenced as described in figure 2 with 16 filters on the first layer. This model leads us to an average accuracy of 78% and a f1 score of 73% on the training set.

After analyzing the results in more detail, we found out that there were a few isolated patches that that were surrounded by their opposite segment class and are clearly a false prediction. Therefore we adapted our prediction method to perform a closing method after the actual prediction. The closing method detects the isolated patches (for example identified as roads surrounded by background) and replaces them by the value of the surrounded segment class. This improved our f1-score by 2% and brought us to a total f1-score of 75% and an accuracy of 80% on the training set. The result of this advanced prediction can be seen in figure 3.

(a) normal prediction (b) advanced prediction

Fig. 3: Isolated nodes on overlay prediction based on the fractal model. Red overlay indicates background

Despite the results improving from our last model, it was still not good enough to perform a good segmentation of images so we chose to use another neural network architecture. By performing data exploration once again we can observe that within the 16x16 patches of the ground truth images there are pixels that are identified as road and background (in a single patch there is not always all road or all background pixels). Therefore if we classify the whole patch uniformly these pixels will be misclassified, resulting in worse performance. Based on this we decided to opt for a pixel wise approach, explained in the next section.

C. U-Net Model

Based on research one of the most popular and successful models for image segmentation is the U-Net architecture [3]. This neural network architecture consists of two paths: the contracting path and the expansive path (figure 4).

In our model the contracting path consists of one convolutional block with filters of size 5x5 and four convolutional blocks with filter size of 3x3. Each block consists of 2 layers comprising a convolution, a batch normalization and an activation function. Then each block is followed by a dropout and a 2x2 pooling operation. The first layer blocks have 64 filters each, and this number is doubled each layer, to finally get to 1024 filters at the deepest layer of the network. The objective of the contracting path is to capture the context of the input image.

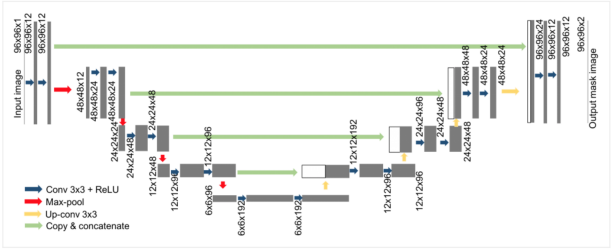


Fig. 4: Schematic U-Net architecture, left side contracting path, right side expansive path [3]

The expansive path, also known as up-convolution, consists of a similar structure, but instead of the max pooling a transpose convolution is applied in order to upsample the feature map and a concatenate operation to recover some more precise information that was lost during the downsampling. With this method we are able to combine the high precision of the expansive path with the contextual neighbourhood information of the contracting path. The final CNN architecture with all the layers can be seen in table I

Layer type	Parameters
Input	400 x 400 x 3 RGB image
Convolutional Block	64 (5 x 5) filters
Max Pooling	2 x 2
Dropout	p = 0.25
Convolutional Block	128 (3 x 3) filters
Max Pooling	2 x 2
Dropout	p = 0.5
Convolutional Block	256 (3 x 3) filters
Max Pooling	2 x 2
Dropout	p = 0.5
Convolutional Block	512 (3 x 3) filters
Max Pooling	2 x 2
Dropout	p = 0.5
Convolutional Block	1024 (3 x 3) filters
Transpose Convolution	512 (3 x 3) filters
Concatenate	
Convolutional Block	512 (5 x 5) filters
Transpose Convolution	256 (3 x 3) filters
Concatenate	
Convolutional Block	256 (5 x 5) filters
Transpose Convolution	128 (3 x 3) filters
Concatenate	
Convolutional Block	128 (5 x 5) filters
Transpose Convolution	64 (3 x 3) filters
Concatenate	
Convolutional Block	64 (5 x 5) filters
Convolutional layer	1 (1 x 1) filter
Output	400 x 400 x 1 Grayscale image

TABLE I: Final CNN architecture with all layers

In order to improve our model we considered the following parameters:

1) Activation functions: The purpose of an activation function is to decide whether the neuron should be activated or not, based on the weighted sum of its inputs with a bias added. Given the neuron $Y = \sum(weight \cdot input) + bias$ the activation function operates as a classifier. As seen in class there are many ways of implementing a classifier, such as sigmoid function, Rectified Linear Unit (ReLu) and the Leaky ReLu. One of the biggest issues of most of the activation functions are "dead" filters. This effect can occur when the chosen classifier converges to or assumes a certain value. At this point the gradient will be very small or even vanish, which implements very slow or even no learning progress. For that reason we chose the Leaky ReLu activation function with $\alpha = 0.1$.

2) Image augmentation: There are two main reasons for which we decided to perform image augmentation: the first reason is that our dataset is simply too small to train a neural net with high accuracy values. The second reason is because in the previous models we saw that the diagonal roads had problems with being detected correctly so it makes sense that in the process of expansion we rotate the image, so as to train the neural net in unusual road orientations. In terms of the images there are many image processing methods to modify the image such as flipping, rotating, resizing etc. In our case we randomly rotate each image/ground truth pair in a range from 0 to 90 degrees and randomly flip them horizontally and vertically. We used a generator that feeds the model with these images so our dataset has as many images as we need for the fitting process.

3) Regularization: A common way of improving the model is the use of regulariser. For this reason we added dropouts of 0.5 after each max-pooling operation (except the first one, where it is 0.25). Another important regulariser is the batch-normalisation that normalises the input layer by adjusting and scaling the activations. As a result the network trains and converges more quickly.

4) Loss Function: As we observed in Section II the given dataset is not symmetric, meaning the ratio of segment classes is not equal. Therefore we implemented the Jaccard coefficient loss, which penalises the model in a linear matter for misclassifications in both ways and therefore performs the best for asymmetric segment classes [4].

As a result with the implemented functions this model leads us to an average f1-score of 97% on the training dataset.

V. Implementation Details

To train the models we used the open source library *keras* which runs *tensorFlow* as backend. During the training process we realised that the computing power provided by our computer (Intel Iris Graphics 6100 1536 MB) was not sufficient to train the model in a reasonable time. Therefore we ran the fractal model on the *Google Colaboratory* notebook that uses an external GPU to train the model. For our final model, the U-Net model, we used the more advanced *Google Cloud* Services in order to train our model.

VI. Final Results

The final results can be seen in table II and table III. The first significant improvement was made by adding more convolutional layers and adding the fractal structure. This increased the f1-score by 32% compared to the basic convolutional layer and with the advanced prediction by 36%. With adding a more complex U-Net model that does the classification pixelwise

and not patch wise, we were able to increase the average f1-score by another 30% to a score of 0.97 on the training set. This model was then used to run on the test dataset (table III).

Model	F1 - Score
Basic Convolutional Model	0.55
Fractal Model	0.73
Fractal Model (advanced prediction)	0.75
U-Net Model	**0.97**

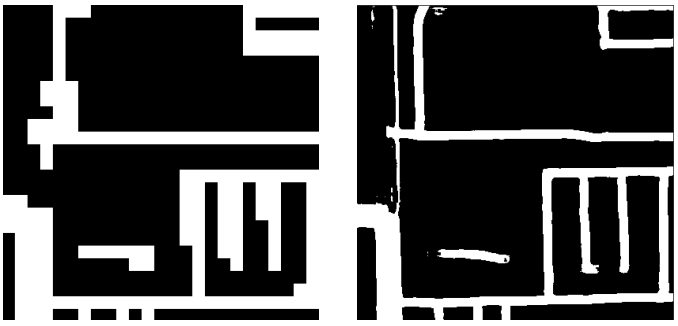
TABLE II: F1-Scores for different models on training set

Model	F1 - Score
U-Net Model	**0.898**

TABLE III: F1-Score for U-Net model on test set

Since the images of the test dataset did not have the same size as the testing images we trained our model on, we predicted each test image in with a sliding window of 400x400 pixels. With every step we made, we took into account only the non overlapping part.

As an additional remark the submission was based on 16x16 patches, however the model was built on a pixel wise structure. This caused a reduction of the overall f1-score because, as explained above, inside a single 16x16 patch there may be road and background sections simultaneously. The comparison between the different predictions can be seen in figure 5.



(a) patch wise prediction (b) pixel wise prediction

Fig. 5: Comparison between patch and pixel wise prediction

VII. Conclusion

As a conclusion we were able to train a convolutional neural network with a total f1-score of around 90% on the test set. In order to achieve this performance we expanded the dataset, the amount of layers and filters and optimised activation, regularisation and loss functions in the U-Net structure. With regard to our concerns about the misclassification of roads that were covered by trees or other objects, the model was then capable to achieve a very good prediction, thanks to the neighbourhood context detection on a pixel wise level. The overlay prediction of a sample image on the test set can be seen below (figure 6).

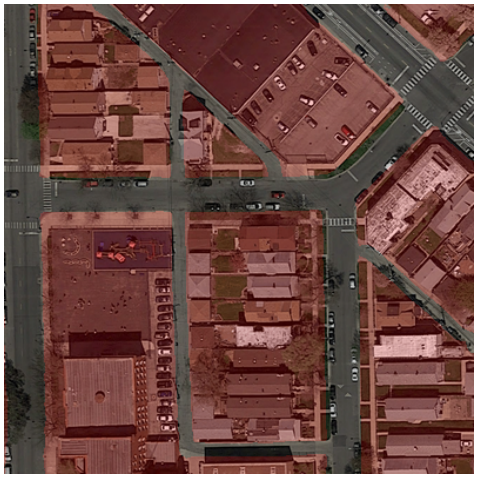


Fig. 6: Final results overlayed with the original image for the pixelwise prediction

References

[1] K. P. Shung, "Accuracy, precision, recall or f1?" 2018, https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

[2] G. Larsson, M. Maire, and G. Shakhnarovich, "Ultra-deep neural networks without residuals," 2017, https://arxiv.org/pdf/1605.07648.pdf.

[3] T. Sterbak, "U-net for segmenting seismic images with keras," 2018, https://www.depends-on-the-definition.com/unet-keras-segmenting-images/.

[4] wassname, "Jaccard-distance-loss," 2017, https://gist.github.com/wassname/f1452b748efcbeb4cb9b1d059dce6f96.

[5] F. F. Li, J. Johnson, and S. Yeung, "Cs231n: Convolutional neural networks for visual recognition," 2018, http://cs231n.github.io/convolutional-networks/.