Machine Learning Project 2 Report: Task B, Road Segmentation

Carl Bou Akl (carl.bouakl@epfl.ch, 325635) and Ralph El Haddad (ralph.elhaddad@epfl.ch, 302360)
December 2021
EPFL, Switzerland

***Abstract*—The aim of this project is to perform road segmentation on satellite images using Machine Learning. Three models were used: logistic regression (LR), fully convolutional networks (FCN), and U-nets.**

I. Introduction

Road segmentation consists of using Machine Learning to discern roads in images, by labeling pixels as 1 if they are part of a road, and 0 if they are not. It has been a popular field in the past decade as we have witnessed the rise of real-time satellite images (Google Maps), driver-less cars, and even delivery drones. The growing popularity and demand, as well as the need for real-time predictions, have led to more and more innovation, which means faster and more accurate Machine Learning models. This paper explores and compares several of these models, namely Logistic Regression, Fully Convolutional Networks (FCN), and U-Nets. The different steps have been documented as follows: (i) Exploratory Data Analysis, (ii) Data Pre-Processing, (iii) Model Architecture, (iv) Model Tuning and Evaluation, (v) Results.

II. Exploratory Data Analysis

The data set used was provided by Prof. Jaggi and downloaded from the Machine Learning Project 2 page on AIcrowd. It consists of a training set, and a test set. The training set consists of one folder of 100 real satellite images of dimension 400x400, and another folder of the corresponding ground truth images (or mask images, where roads are labeled as 1 and background as 0). The test set consists of 50 images of dimension 608x608. The difference in the dimensions of the training and testing images indicates that the code should be flexible to the image size.

The first step in taking on the project is to take a closer look at the data set at hand. We noticed that most roads in the provided images are horizontal or vertical, which might lead to weaknesses in the models when predicting roads that are diagonal or round. We also saw that many roads were covered by trees, which basically accounts to noise on the data we aim to learn. Finally, we flagged images that had odd groundtruth complements: a few images had missing pixels on roads, or weird shapes.

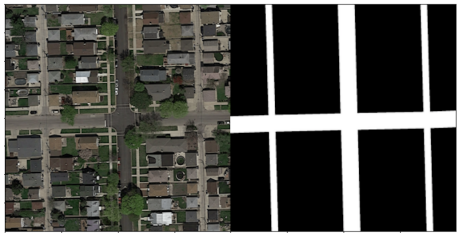

Figure 1. Example of a satellite image from the training set with its corresponding groundtruth image

III. Data Pre-Processing

Preliminary Data Pre-Processing starts with the dropping of images with odd predictions [images 65 and 76], as well as the fixing of images that had missing pixels on roads [images 31, 33, 35, 41, 72, 77, 96, 99] using Photoshop. The first touch of pre-processing is indeed manual. It may not have a huge impact on the end results.

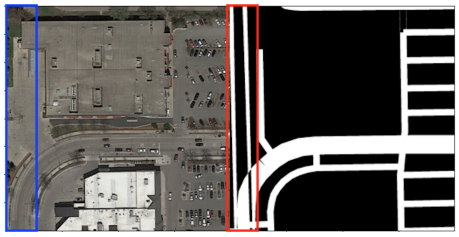

Figure 2. Image 33, an example of training images that require fixing

Since it is not reasonable to train the Logistic Regression model pixelwise (an image of 400x400 would yield 160 000 features!), each training image is cropped up into patches. The patches attributes are then used as features: either 2-Dimensional features consisting of the average gray color of each patch, as well as the variance, or 6-Dimensional features consisting of the average RGB colors, and the variances. In order to generate even more features, a polynomial feature expansion was implemented using the Sci-Kit Learn function "PolynomialFeature". This feature processing was only used for the baseline Logistic Regression model.
For the FCN and the U-Net the data set is augmented after the preliminary pre-processing. We implemented different data augmentation techniques: Gaussian blur, median blur, rotation (random angles between 0 and 360), flip, contrast

change, sharpness change, and add random brightness. Augmenting the data set helps the model learn better, as it exposes it to many more scenarios. Rotating the data set helps the model recognize roads in different angles, since most roads in the training set are horizontal and vertical, as noted in section II.

As we still wanted to improve the accuracy using Data Augmentation, we looked for different approaches. One interesting alternative was AUGMIX. AUGMIX utilizes stochasticity and diverse augmentations,a Jensen-Shannon Divergence consistency loss, and a formulation to mix multiple augmented images to achieve state-of-the-art performance. To put it simply, AUGMIX consists of mixing the results from augmentation chains or compositions of augmentation operations: first, it randomly samples k augmentation chains, where each augmentation chain is constructed by composing from one to three randomly selected augmentation operations. Then, the resulting images from these augmentation chains are combined by mixing using elementwise convex combinations. The final image incorporates several sources of randomness from the choice of operations, the severity of these operations, the lengths of the augmentation chains, and the mixing weights.

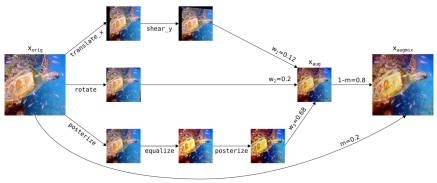

Figure 3. Mosaic of an image augmented with AUGMIX

IV. Model Architecture

Three models were implemented in this study: Logistic Regression, Fully Convolutional Networks (FCN8), and U-Nets.

Logistic Regression and FCN8 serve as baseline models, as we will compare their performance to that of the U-Net models, which are expected to yield stronger F1 scores and accuracy. This section documents how each model was implemented in detail.

Logistic Regression: The Logistic Regression model can be viewed as a special case of Neural Networks, as a single layer model, without any hidden layers (see figure 4). The activation function used is the sigmoid function.

This model serves as a baseline to the Convolutional Neural Networks (CNN).

The Logistic Regression model used in this paper was implemented using the Sci-Kit Learn (sklearn) library (”linear_model”).

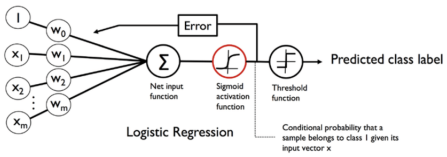

Figure 4. Architecture of the Logistic Regression Model

Fully Convolutional Networks: When dealing with image segmentation, there is a need for a CNN with variable input dimensions. Which is why Fully Convolutional Networks have been a popular choice in computer vision. FCN is a network that does not contain any ”Dense” layers as in traditional CNNs. Instead it contains 1x1 convolutions that compute the task of fully connected layers (Dense layers).

As shown in figure 5, the FCN implemented in this project is the FCN8, which adds 1x1 convolutions to an VGG16 convolutional network. The modifications in question are: taking the output of conv4, adding a convolution layer, and taking the output of the VGG16, upsampling it, and adding both. Then we upsample that output, we take the output of conv3, convolute it and we add both. Finally, we upsample the output and use softmax.

The FCN8 used in this study was coded using the Keras library (”tensorflow.keras”).

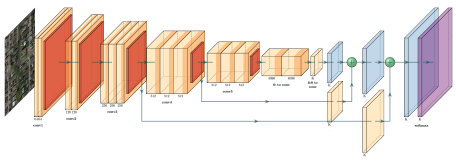

Figure 5. Architecture of the FCN8 Model

U-Nets: U-Nets build upon the FCN model and extend its architecture such that they work with very few training images and yield more precise segmentations. The most important modification is that in the upsampling part U-Nets also have a large number of feature channels, which allows the network to propagate context information to higher resolution layers. As a consequence, the expansive path (left side) is more or less symmetric to the contracting path (right side), and yields a u-shaped architecture (see figure 6).

The contracting path follows the typical architecture of a CNN. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed

by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers (see figure 6.
The FCN8 used in this study was coded using the Keras library (”tensorflow.keras”).

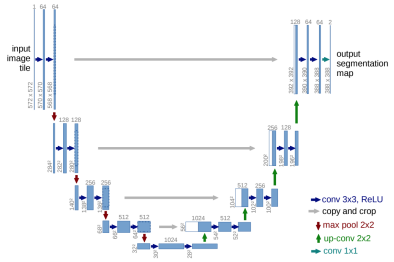

Figure 6. Architecture of the U-Net Model

V. Model tuning and Evaluation

After defining our models, we had to tune and evaluate them in order to optimize their performance.
The metric used for evaluation is the F1 score, as well as the accuracy as a secondary metric. Each model is evaluated through 4 or 5-fold cross-validation, always using the same seed for reproducibility.

Logistic Regression: Logistic Regression is the first baseline model of this study, as it is a relatively simple, linear model.
Cross-validation was performed on the Logistic Regression model using the Sci-Kit Learn library (”GridSearchCV”). Folds were created using ”KFold”, and the scores were computed using ”cross_validate”.
The baseline model was first tuned by regularization. The optimal hyperparameter C is found by cross-validating over a range of values of C, and retaining the value that yields the best results. Regularization reduces overfitting and improves the model’s performance.
Further tuning is done by polynomial feature expansion of order 4, using the sklearn function ”PolynomialFeature”. Polynomial feature expansion helps the model explore further dimensions in the training set, in the hope of finding deeper connections to the output in higher dimensions.
Improvements can clearly be seen after the tuning (table III), as the F1 score rises to almost 50%, while its variance drops, which means better chances of reproducibility.

The weak F1 score shows that the Logistic Regression is not the optimal model for road segmentation. Even though the model can guess a fair amount of roads, it seems that the False Negative rate is too high (see figure 7). That is the problem of patch-wise labeling: it is not precise enough. It is not optimal either in case of diagonal roads, or roads of width smaller than the patch width. With more time at hand

Table I
Table of validation scores of the Logistic Regression model

Logistic regression	F1-score	Accuracy
2D FT	$41.49\% \pm 5.10\%$	$50.50\% \pm 11.74\%$
6D FT	$32.75\% \pm 18.18\%$	$64.70\% \pm 4.39\%$
2D FT + Pre	$10.41\% \pm 18.02\%$	$61.90\% \pm 20.25\%$
6D FT + Pre	$44.98\% \pm 2.32\%$	$58.24\% \pm 5.63\%$
6D FT + Pre + PE(4)	$49.07\% \pm 1.68\%$	$56.54\% \pm 6.08\%$

FT: features.
Pre: preliminary data pre-processing.
PE(order): polynomial expansion of order 4.

we would have tried reducing the patch size by half (training was done with 16x16 patches), as training with 8x8 patch size takes around 4 hours.

Figure 7. Prediction of the Logistic Regression on test image 50

FCN8: Enter Convolutional Networks. The first CNN implemented in this study is the FCN8, as it will serve as a baseline to the U-Net model. We train the CNN using a GPU RTX3090.
FCN8 training was done by randomly splitting the training set into a training set, and a validation (test) set with a ratio of 90%/10%. We started with a batch size of 32 and a dropout of 0.1 after each convolution. In order to speed up the training and to avoid decreasing F1 score, we used an early stopper with patience 10.
The tuning of the FCN8 model started with the parameters. We decreased the batch size from 32 to 8 because of the very high number of parameters of the model. We also increased the patience of the early stopper from 10 to 30. We added a learning rate decay of
Tuning is also done by adapting the data augmentation to the model. Following several trials, we found that the results were best when augmenting the data with horizontal and vertical Flip as well as random Rotate.
Furthermore, we added batch normalization after every ReLU.Batch normalization yields better learning than normalizing the inputs at once, since we normalize every batch

according to its mean and standard deviation, at every convolution.

U-Nets: Training the U-Nets was similar to that of the FCN8.

We start by splitting the training images: 85% training and 15% validation. The initial batch size is 32, the dropout is 0.5 after every convolution and the early stopper patience is 30.

In order to optimize the weights, we tried different losses and combinations of loss: dice loss, binary cross-entropy and F1-loss. We found that F1-loss was the best optimizer for our U-Nets model.

After trials we notice that parameters can be tuned to improve performance: batch size is changed to 25, dropout to 0.1, and patience to 10.

Data augmentation definitely helped improve the accuracy of the model. But after thought, we tried to drop a few transforms as too many combinations of data augmentation can lead to under-fitting. Different combinations of data augmentation were tried in order to find the most appropriate one for the main model (U-Nets). The combination found to be the most efficient was to rotate the images at random degrees, and then flipping them (left to right and top to bottom). The data was also augmented flip (left-right, top-bottom) alone, as well as random rotate alone.

Table II
TABLE OF VALIDATION SCORES OF THE CONVOLUTIONAL NEURAL NETWORKS

CNN	Avg F1-score	Avg Accuracy
FCN8 + PDPP + Aug	0.791	0.875
FCN8 + PDPP + Aug + Augmix	0.796	0.885
FCN8 + PDPP + Aug + Augmix + BN	0.822	0.889
UNet + PDPP	0.8729	0.8994
UNet + PDPP + Aug + Augmix + BN	0.9215	0.9382

FT: features.
PDPP: preliminary data pre-processing.
BM: Batch Normalization

VI. RESULTS

The results clearly show the superiority of the U-Nets model over the FCN8 and the Logistic Regression. U-Nets yield higher F1 score, higher accuracy, and better predictions, all in less time and with less computational operations: indeed, the number of parameters used by the FCN8 is 13 millions, while the U-Nets model uses 1 million parameters.

Both models out-perform the logistic regression, which was of course expected since linear models struggle with computer vision task: the use of kernels when convoluting in CNN layers help the classification take pixels in context, which facilitates learning and increases accuracy.

Table III
FINAL PERFORMANCE COMPARISON

Model	F1-score	Accuracy
Log. Regression	49.07% ±1.68%	56.54% ±6.08%
FCN8	82.23 % ±2.43%	88.91 % ±0.54%
UNet	91.92% ±0.63%	93.22% ±1.34%

VII. DISCUSSION

Our report shows results are optimal with the U-Net model. These results were expected since U-Nets normally out-perform FCN models, as they have been designed as an upgrade to the fully convolutional networks. Both CNNs largely out-perform the logistic regression baseline model, which is also expected since CNNs are optimal with computer vision problems, which is not the case with linear models.

Ideally, we would have tried to better tune our U-Net by optimizing the data augmentation, or by feeding it a bigger data set that is similar to the 100 images provided by the competition.

We would have also considered post-processing our predictions if we would have had more time and ressources at hand.

All in all, we can confirm that U-Nets are the new state-of-the-art in segmentation problems, as this new technology is faster, less computationally expensive, and more accurate than its predecessors.

Figure 8. Prediction of the U-Net on test image 1

REFERENCES

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[2] M. Saifi, J. Singla, and Nikita, “Deep learning based framework for semantic segmentation of satellite images,” 03 2020, pp. 369–374.

[3] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, “Augmix: A simple data processing method to improve robustness and uncertainty,” 2020.

[4] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015.