Machine Learning Course
Project Road Segmentation

Mathilde Guillaumot - Priscille Guerrier de Dumast - Thibaut Chamard

École polytechnique fédérale de Lausanne, Switzerland

***Abstract*—The aim of this project is to build a classifier that identifies roads on given satellite images. We have implemented two distinct segmentation algorithms using first a logistic regression and second a convolutional neural network.**

I. Introduction

To address the problem of predicting the presence of road on satellite images, a first approach would be to train a logistic regression but the complexity of images needs a state of the art technique to learn and extract in the better way features that render the subtleties of an image. Convolutional Neurel Networks answer this need by providing a complex architecture which leads to great image recognition or classification.

II. Data Exploration

The dataset available is composed of 100 images and their ground truths. In this work we will not classify the pixels of the image but patches of 16x16 pixels extracted from the original images. Ground truths are binary images showing if a pixel belongs to the road or background. Input images are composed of three channels: Red, Green and Blue. Based on the ground truth image, a patch is considered as road if more than 25% of its pixels are assigned to road. In order to preprocess our training dataset, we have executed the following steps:

- **Standardization** For the Logistic Regression, we standardize the features while for the CNN, we standardize the patches on a channel basis, which means that we compute the mean and standard deviation of the patches along each color dimension.
- **Balancing the data:** When looking at our training images, we can see that the data is highly unbalanced: 25% of the patches are labeled as road and 75% as background. Leaving it as it is would bias the training of both our models as it will tend to predict the most likely class. To compensate for it, we sample our data to have equal contributions of both classes so that the models only rely on the features to classify the patches.

III. Models

A. Logistic Regression

1) Data preprocessing: The actual available informations on the patches are the Red, Green and Blue (RGB) components of each pixels. Working on those channels, we can extract more informations on the data.

Before extracting features from the images, we get more information on its composition.

- Image conversion: Having RGB images, we get a new version of it by converting it to gray scale level image. This new image is considered as a new channel of the image.
- Detecting boundaries: Based on gray scale images, we run a Sobel filter on each direction of the image, vertical and horizontal. This is done by convolving the new gray scale layer image with a kernel, accentuating differences along x or y axis. For instance, we have the kernel Gx to show horizontal changes:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

  The output of both Sobel filters are also considered as new channels of the image patches.

Features extraction: We chose to take as features to feed our logistic regression model, the mean and standard deviation of each channel of our patches, after getting all these new information. Having now 6 channels, this will result in 12 features.

Features engineering: To get more features, we combined all the extracted ones together. Each feature is multiplied by itself and to all other features. Assuming we have initial features [a, b, c], we will now get for features using [a]: [a, $a^2, a.b, a.c$].

2) Post-processing: Looking at the confusion matrix of the classification (cf. Table I), we see that the main misclassification lies in the false positives. Indeed, our model mostly tends to predict that a patch belongs to the road whereas it does not. This post-processing is done only on the patches of our testing dataset.

3) Method & Evaluation: We realize a logistic regression using regularization. The choice of the weight of the penalization is defined by implementing a 10-fold cross validation. Here we assess of the best value looking at the root mean square error. In the mean time, we evaluate the mean F-score, to assess the efficiency of the post processing.

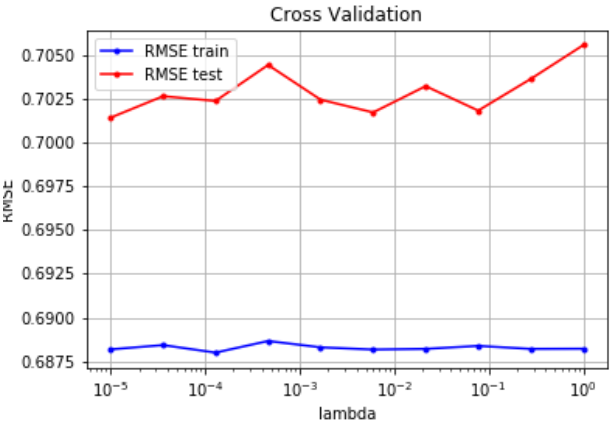

Fig. 1. Evolution of the root mean square error according to the penalization factor for the logistic regression

According to Figure 1, the best penalization weight is $\lambda = 10e{-}5$. Here, by looking at the confusion matrix (cf. Table I), we can better understand where our model fails.

	Predicted background	Predicted road
Actual road	2679	2094
Actual background	833	644

TABLE I
CONFUSION MATRIX ISSUED OF THE LOGISTIC REGRESSION

To partly solve this problem here, we consider that a patch cannot be classified as road if most of its neighbors are labeled as background. To implement this, we consider the matrix of the predictions as a binary image. We convolve it with the following kernel K, which will produce a new image. At each point, we will get the number of neighbors classified as road. If a pixel get a value strictly inferior to 2, we update its prediction as background. Thus, we cannot have isolated road patches.

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In average, this post-processing makes us gain up to 0.05 on the mean F-score. As shown on Figure 2, we see that real isolated road patches are adjusted, even though we still have misclassification. This is especially visible on roof of big buildings, which could be very similar to roads. However, it is not a robust post-processing technique taht we can use for all models. It is implemented as it is, to cope with the high number of false positives.

B. Convolutional Neural Network

1) Introduction: Convolutional Neural Networks (CNNs) have proven to be very effective in areas such as image classification or recognition. It involves 4 main operations which are the following:

- Convolution
- Non Linearity (ReLU)

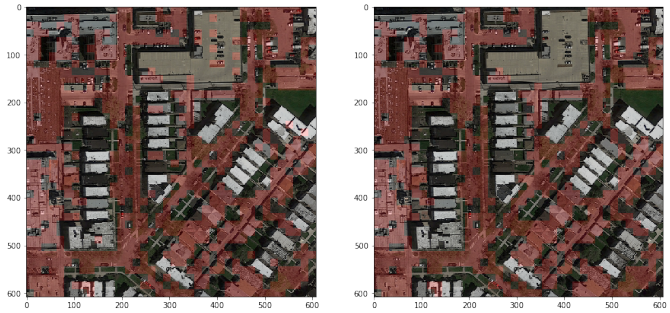

Fig. 2. Prediction on an image before (left) and after (right) running the postprocessing

- Pooling or subsampling
- Classification (Fully Connected Layer)

The **convolution** operation can be considered as the feature detector. We define a certain number of NxN filters which we slide over the image and by computing the dot product we create our 'Feature Map'. Different values for such filters will produce different feature maps. The CNN will learn the values of the filters used in the convolutional layers on its own but we will need to give it specific parameters (filter size, number of filters, etc). Intuitively, we understand that a big number of filters will enable us to extract more image features and will improve the network ability to predict the patterns of unseen images. Thus, we must decide on three important parameters before performing the convolution step: depth, stride and zero-padding.

The **non linearity (ReLU)** operation replaces all negative pixels with zero value. This step introduces non linearity to respect the non linearity of real world datasets. This operation rectifies the feature maps previously computed with only linear operations.

Posterior to this step comes the **pooling** operation which is needed to reduce dimensionality of our feature maps nevertheless keeping the most important information. Pooling helps reducing the number of parameters of the network and plays a key role in controlling overfitting. It also makes the network invariant to small transformations such as translations as pooling takes the max of a local neighborhood. Pooling is applied to each feature map separately.

The three above operations constitute the main block of the CNN and play together the role of features extractor for our input images. The last structure element of our CNN are the **fully connected layers** which perform the final classification step. It uses classifiers such as the softmax activation in the output layer. It will use the high-level computed features extracted in the previous steps to classify our input image into the classes specified by the training set. The fully connected layer is also efficient in learning combination of our features which will improve image classification.

Now that we have clarified the main purposes of each element of the CNN, we need to explain the process used when training the model:

- Step 1: Random initialization of filters, parameters and weights
- Step 2: Forward propagation step (Convolution, ReLU, Pooling, Fully Connected Layer) to output probabilities for each class (road or background)
- Step 3: Compute loss (includes regulrization term)
- Step 4: Backpropagation to calculate gradients of the loss and use optimizer such as Gradient Descent or Adam Optimizer to update values of filters, parameters and weights.

To begin, we have trained our data on a simple LeNet architecure made of 2 convolutional layers each followed by one ReLU and one pooling layer and one fully connected layer to classify our training data into road or background. To compute the loss in output of the CNN training steps, we have chosen the cross entropy to which we add a regularization term computed with the L2 norm of all the weights in the architecture. For optimizing the parameters we have chosen the Adam Optimizer. To reduce computational costs, we submit to the CNN batches of 100 inputs picked successively in the training dataset after each iteration. We have been inspired by the pipeline proposed in this course "Introduction to convolutional neural networks using tensorflow" by Jesús Fernándes Bes [1]

2) Pre-processing: In order to improve the efficiency of our CNN, we have proceeded to five pre-processing steps (some explained previously): image rotations, standardization of patches, labeling ground truth images by threshold, balancing the data and splitting the data into train and validation sets.

Rotation of training images: We have observed in our train and test sets the presence of diagonal roads on some images. With mostly horizontal or vertical roads submitted to our CNN, the learning process will be less prone to assess for diagonal roads and it will therefore impact the good classification of our patches. In order to compensate for it, we are applying to each original image a random rotation thus doubling our amount of training data (see Figure 3).

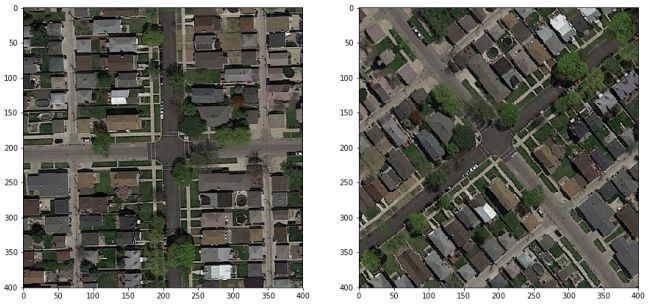

Fig. 3. First training image and its rotated associated image

3) Fine tuning of the CNN's parameters: In the above described configurations of parameters of the CNN, we achieve a score of approximatively 0.81% on Kaggle with 10000 iterations and a dropout rate of 0.5. Let's consider which of the parameters of the CNN could be fine tuned to improve our classification score. It is important to note that in regard of the computational costs of training CNNs, we are limited in our choice of tuning combinations.

- **CNN architecture:** To compare different CNN architectures, we have also implemented a CNN with the AlexNet architecture based on a first convolutional layer with filter size 11x11, stride 4 and depth 96, followed by a first pooling layer. Then we go back to the filter size of 5x5 and stride 1 with depth 256 in a second convolutional layer again followed by a max pooling step. And then, we add 3 other convolutional layers in a row with filter size 3x3, depth = [384,384,256]. We end up doing a max pooling step and add two fully connected layer of depth 4096 before applying the softmax activation classifier. We did not encounter significant improvements in the predictions of test images classes but we did have to face a long time of computation. For the rest of our fine tuning experiments, we will stick to the basic CNN LeNet architecture introduced in the beginning as it provides good classification without being too computationally costly.

- **Filter size:** As mentioned above, the convolutional layer is characterized by three essential parameters: its depth, stride and padding. But the filter size also matters. We have chosen an initial filter size of 5x5x3 on conv1 because the amount of weight and bias parameters depends on this architecture as it results in 5*5*3 = 75 weights and bias variables. Growing the size of the filter would jeopardize both our computational efficiency and the control of overfitting has it would increase the number of parameters. A solution to that could be to increase both filter size and stride, for example: (11x11) filter size with stride 4 as in the AlexNet architecture. In the case of our simple architecture, we will stick to a stride of 1 but play with the filter size and observe the consequences on the accuracy for training and validation sets. We lower the number of iterations to 5000 to reduce computation time. From the above table, we can infer

Filter size	Validation set accuracy
3x3	0.7698
5x5	0.7893
8x8	0.7700
12x12	0.7490

TABLE II
VALIDATION SET ACCURACY IN REGARD OF FILTER SIZE

that a filter size of 5x5 seems to be the most efficient. These results can be explained by the fact that growing filter size does not necessarily improve accuracy as it may bring along overfitting by growing the number of parameters in the CNN. From now on, we will stick to

a filter size of 5x5.

- **Dropout rate:** Another parameter needed to be considered is the dropout rate. Dropout is useful to build a robust neural network. Indeed, by dropping a certain amount of nodes during training, the CNN prevents itself from being influenced by inter-dependencies in the data. We will evaluate the importance of drop out by training the CNN with dropout rates between the interval p = [0.4, 0,9], p being the probability of a node to be left out. Testing rates below 0.4 is unnecessary as we know from litterature that minimum dropout rate of 0.5 is commonly used. From our experiments, very little improvements are seen when growing the probability of dropout so we will chose to stick to the p = 0.5 value.

Ideally, we would have liked to try out more architectures and combinations of parameters (drop out probabilities, filter size depth, stride or padding, weights and bias initialization, etc) but we are limited by our resources and we have made the choice of documenting ourselves to know more about the right choices of parameters values to theoretically fine tune our model.

4) Context of patches: One thing being observed after several trainings is that from our actual implementation patches predictions is done assuming that they do not have any dependencies among each other. However, we know for a fact that a road is a succession of road patches and therefore knowing that patches depend on each other can improve predictions especially in the cases were the road is occluded or hidden by a third-party element such as cars or trees. Likewise, parts of buildings may be interpreted as road patches but by knowing their context and thus their neighbor patches our CNN could make better predictions. For now, we submit to our CNN 16x16 images to which is associated a binary value expressing whether it is road or background. We will now consider associating to our original patch image a context made of its neighbor pixels. To do so, we use Numpy padding of arrays and its reflection property which allows to deal correctly with border patches (see Figure 4). In this way, when training the data the CNN will be able to predict road labels in regard of patches context. This method greatly improves our CNN accuracy and f1 score on the test set despite needing a greater time of computation as the inputs are bigger when adding context to patches. We add a 16 pixels border but would have liked to use a bigger one such as a 32 pixels border which tells even more about the patches context.

IV. Results

The metric used to determine which model perfoms the better is the mean F-score, corresponding to the harmonic mean of precision and recall. As a final model, we chose the one that maximizes this score. Results for each tested models are shown in Table III. On Figure 5, we can asses visually of the correctness of the predictions.

Fig. 4. Reflection of image around its border to extract patches with context

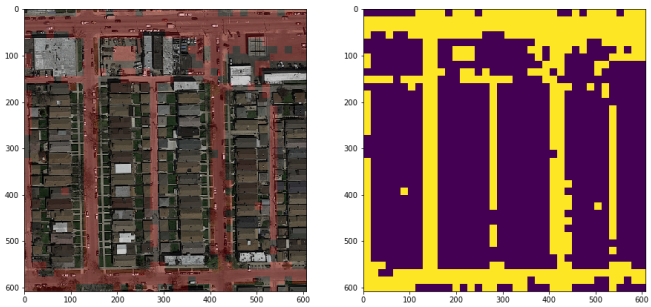

Fig. 5. Overlay of test image and road segmentation predictions (left) and binary output image predicted by the CNN (right)

Model	mean F-score
Logistic regression	0.63445
Logistic regression (post-processing)	0.65022
Convolutional neural network	0.84628

TABLE III
FINAL MEAN F-SCORE FOR THE MODELS TESTED

V. Discussion

We had several leads in mind to proceed to the post-processing of our test images predicted by our CNN but their implementations did not fully worked out. Thus we preferred to stick to a less efficient but totally achieved pipeline. We aimed to predict continuous lines for roads, to do so we thought of the Hough Transform which first purpose was to detect lines in images. We also implemented a function that drew lines that went over the images up and down as well as diagonally. When a certain percentage of the pixels on these lines was above a defined threshold, all the pixels in these lines were assigned as road, the rest of the pixels were assigned as background. This latter method had the merit of drawing clear lines and getting rid of lonely road labeled patches but was a bit hard on selecting road lines and was struggling with some of the diagonal roads, thus it needs to be deepened.

References

[1] J. F. Bes, “Introduction to convolutional neural networks using tensor flow.” [Online]. Available: http://jesusfbes.es/wp-content/uploads/2016/04/MLG_tensor.pdf