Road Segmentation Project

Félix Klein, Valérian Mangé, Galann Pennec
Machine Learning CS-433, Department of Computer Science, EPFL, Switzerland

***Abstract*—This project address the road segmentation of satellite images collected from Google Maps using Machine Learning techniques. We will present how we tackled the problem of road segmentation using several methods such as K-nearest neighbours or RBF Kernel Logistic Regression. The approach with which we had the best performances in terms of F1-score is the convolutional neural network (CNN). We will specify also our approach to improve our results for the CNN architecture principally using data preprocessing and data augmentation.**

I. Introduction

In this report, we focus on patch-based classification techniques to perform the road segmentation task of satellite images. More precisely, in section II, we specify how patches are extracted from satellite images for classification. In section III, we make use of baseline Machine Learning techniques such as RBF Kernel Logistic Regression or KNN before designing in section IV more efficient models like Convolutional Neural Networks (CNN).

We provide in section IV a CNN architecture for the road segmentation task and present our methodology to improve its performance by preprocessing the dataset of image patches, fine-tuning the hyperparameters and augmenting the data.

The performance of our CNN model is evaluated using validation sets and has been tested over 50 Google Maps satellite images.

II. The Road Segmentation Task with Patch-Based Classification

In order to train our model, we divide each image into 16×16 patches, that are in most cases smaller than the size of the road. Therefore, we reduce the road segmentation of images to a binary classification task where each patch can be classified either as a road or as a background. The predictions for each patch are then reassembled to constitute the final prediction for the whole image segmentation [1].

In this project, we use 100 Google Maps images of size 400×400 for training and 50 images of size 608×608 for testing.

III. Baseline Machine Learning Models

We describe here some baseline models we implemented for the road segmentation task, that is to say, the K-Nearest Neighbours and the RBF Kernel Logistic Regression. In each one of these methods, we used the same approach. First, we reduced the number of features to 2, respectively the mean and the variance of each input patch. Then, we applied the Machine Learning models to this reduced number of features in order to generate the predictions.

A. K-Nearest Neighbours

A basic approach for road segmentation is to apply the KNN algorithm on the 2D vectors of mean and variance described in the introduction of this section.

Using a 5-fold cross-validation we can estimate a wise choice for K. According to Fig. 1, choosing $K = 74$ maximises the F1-score for the cross-validation (F1-score of 44.9%).

Even for this optimal parameter K, the true positive rate estimated on a single satellite image from the validation set was about 6% which is low and very unsatisfactory.

The poor performance of the KNN for this task is basically due to the imbalanced training dataset as explained in paragraph IV-A2. Indeed, in our case, background datapoints are a lot more present in the neighbourhood of any new datapoint which can mistakenly contribute to this new datapoint being affected to the background class. One way to deal with the imbalanced datasets for KNN is to weigh the two classes in the prediction.

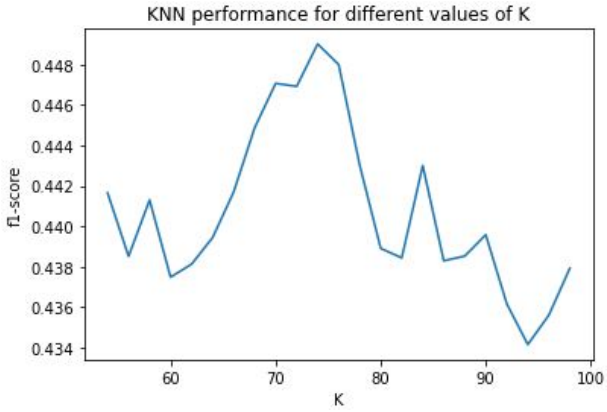

Fig. 1. hyperparameter tuning for KNN using 5-fold cross validation

B. RBF Kernel Logistic Regression

Since the decision boundary in logistic regression is linear, it only works well if the data (here the road and the background points) is linearly separable. Visually, we cannot use a linear separator, so we tried using a radial basis function (RBF) kernel. We also used PCA to visualize the data.

However, when looking at the data after applying Kernel-PCA, we still could not use a linear separator as road datapoints were still mixed with background datapoints (whether we used gray or colored images as input data). With a small sample, we were only able to achieve a true positive rate of around 18%.

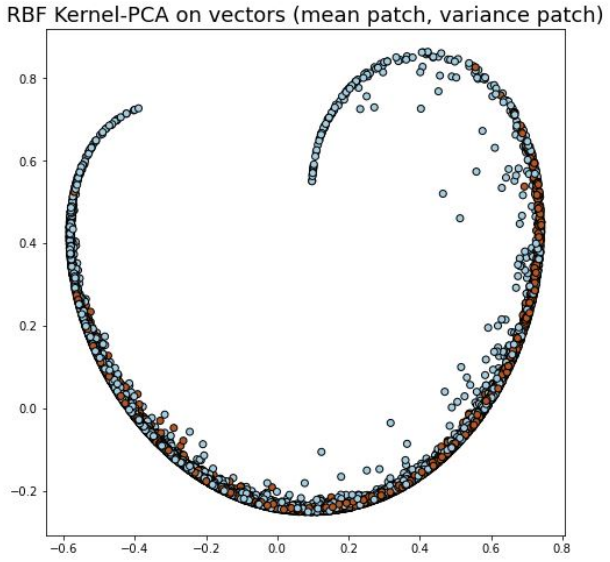

Fig. 2. Kernel-PCA applied to 2d vectors of mean and variance of the patches

IV. Patch-based Classification with CNN

The baseline models we used put in evidence the need to build a robust solution to imbalanced datasets and to design a complex decision boundary to classify the patches. More generally we need to consider preprocessing our data and developing more sophisticated models like Convolutional Neural Networks (CNN).

A. Data Preprocessing

1) Normalizing and Rescaling the Input Patches: From one satellite image to another, the sunshine and, more generally, the weather may change. The satellite images could even be taken in different cities and regions of the world (many landscapes). Our implementation should not be sensitive and generalize well to such variability of the input data. Normalizing and rescaling the patches in order to lessen the impact of such perturbations appeared essential for our task.

The mean and the standard deviation we used for rescaling our patches were estimated for each channel on the training set and then applied for the prediction, on both the test and the validation sets. This is supposed to bring consistency in the data to accelerate the convergence of our CNN model.

2) Oversampling and Undersampling: The proportion of roads is much smaller on usual satellite images than the proportion of background. Namely, only 26% of the patches correspond to roads in the training dataset. This resulted in the latter being imbalanced and our CNN under-performing in recognizing roads. At that time, the CNN was always predicting a background no matter what it was given (F1-score of 0%). To fix that issue, our approach has been to oversample the road patches from our training set and to allow some balance in the way we sample the training patches.

3) Increasing the Size of Input Patches: As explained in II, the road segmentation task is carried out by classifying patches of fixed size (16×16). However, feeding our CNN model with such small patches is inconsiderate. Even a human could not guess if such uninformative patches are a road or not. An

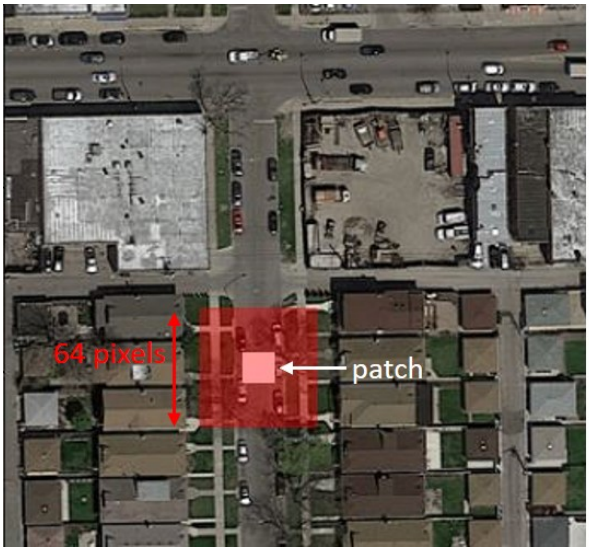

Fig. 3. A frame of size 64×64 surrounding the patch of size 16×16

option is instead to consider keeping a certain area of pixels surrounding the patch (Fig. 3). The generated image is now bigger (64×64 in our work) and is much more meaningful for our CNN to make a prediction on. In particular, this led our model to perform considerably better and to reach a F1-score of 90% on the validation set (see IV-C).

B. CNN Architecture

We developed a CNN architecture [4] with 3 convolutional layers, respectively of 64 filters (5×5), 128 filters (3×3) and 256 filters (3×3). We also applied batch normalization and a max pooling filter of size 2×2 to each convolutional layers of the network [1].

These layers are followed by 3 fully connected layers respectively of size 256, 64 and 2 in charge of reducing the dimension. A softmax classifier predicts in the end a probability for each class (road or background).

In addition, we used a Leaky Relu activation function at each node with a negative slope of 0.01. For every layer we also noticed that adding a dropout layer of probability 20% slightly improved of 1% or 2% the F1-score we had on AICrowd's test set by limiting overfitting issues.

We have tried with 2 and 4 layers of each, however our results were better with 3 layers.

C. Performance on Validation Set

For training our CNN model we used ADAM optimizer with a learning rate of $\lambda = 10^{-3}$ and the Cross Entropy loss. We also defined a scheduler (`ReduceLROnPLateau` from `torch.optim` package) in order to gradually make the learning rate lower when the validation loss does not improve anymore with the actual learning rate.

We evaluated the performance by splitting the original dataset into 80% for the training (50000 image patches)

and 20% for the validation (12500 image patches) and using a batch size of 200.

We could reach an accuracy of 92% and a F1-score of 86% on the validation set after 80 epochs and without data augmentation. After 160 epochs, we had on the validation set an accuracy of 94% and a F1-score of 90%. Regarding the convergence of the CNN, we notice that the loss is stabilizing starting from the epoch 140 (see Fig. 4).

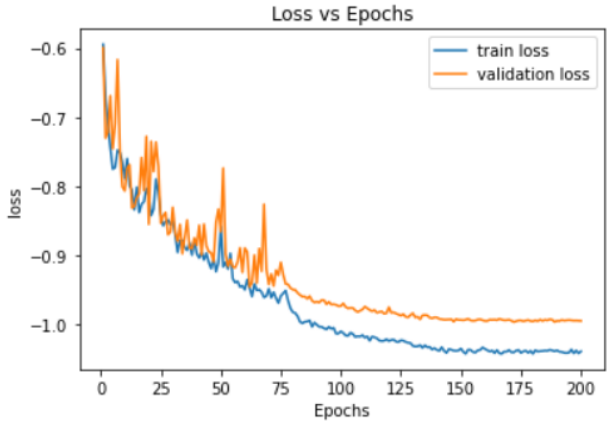

Fig. 4. Training and validation loss curves (200 epochs, no augmentation, log scale)

D. Hyperparameter Tuning

We discuss here the hyperparameters we chose for the training, especially the learning rate in ADAM optimizer and the probability of dropout. The following estimation of the F1-score were measured using only 22 images (13750 patches) for the training and 8 images (5000 patches) for the testing.

In a grid search manner we tested different values for the learning rate: 10^{-2}, 10^{-3} and 10^{-4}. We carried out this experiment over several independent training attempts of the CNN, each time by randomly picking new training and validation sets.

For big values of the learning rate like 10^{-2} on Fig. 5, we remarked that the validation loss starts oscillating between lower and higher values but globally stabilize on a high value where it does not improve anymore. On the opposite, for small values of the learning rate, like 10^{-3} of 10^{-4}, the convergence takes much more time but is more fine-grained which allows in the end smaller loss values and better performance on the validation set. The best value we found for the learning rate was 10^{-3} because it has the advantage of being fine-grained enough for our task and with a fast convergence.

Additionally, as mentioned in IV-C, we make use of a learning rate scheduler (`ReduceLROnPLateau`) in order to continue improving the loss after many epochs.

Then, we studied different choices for the dropout: 0%, 20% or 50% at each layer. We also tried adding a dropout of 80% for the first layer and a dropout of 50% for all the other ones. We compared the performance over several independent

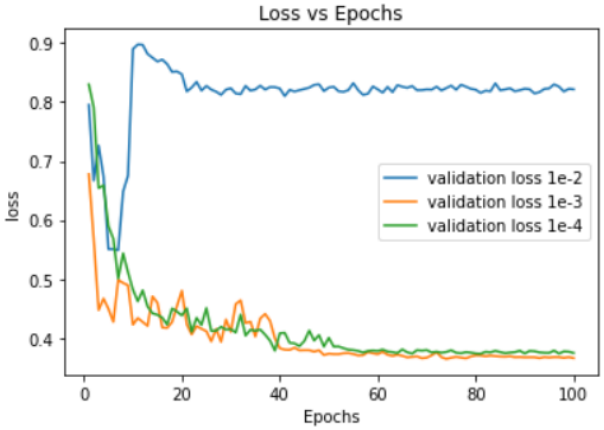

Fig. 5. Validation loss curves for different learning rates (100 epochs)

training attempts, each time by randomly picking new training and validation sets.

Looking at the validation loss curves on Fig. 6, the best performance we had on the validation were without dropout. However, to make our CNN generalize well to new test images, for example the ones used for AICrowd challenge, we decided to maintain anyway a small non-zero value for the dropout.

That is why, we opted for a value of dropout of 20% at each layer of the CNN. Indeed, our task is well-defined and does not need much more regularization since we already use data augmentation (discussed in the next subsection).

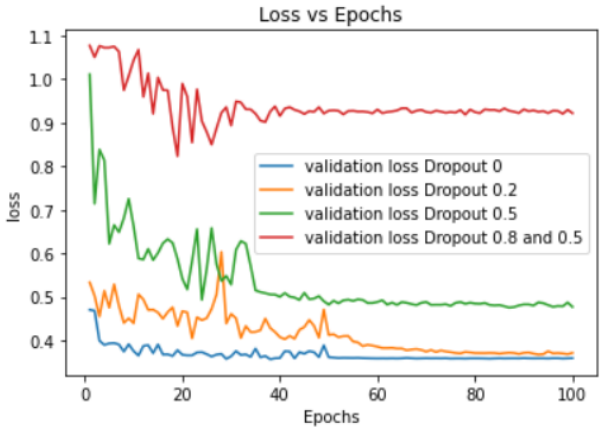

Fig. 6. Validation loss curves for different dropout choices (100 epochs)

E. Data Augmentation

Evaluating our CNN model on the validation set of image patches is not always the same as testing it on real test images. Indeed, the images used for validation may not be representative and may vary a lot from real life satellite images. In particular, the size of the roads, the weather, the orientation of the roads may vary a lot.

For that reason, and also to prevent from overfitting the training data, we made use of data augmentation which is well-suited for our road segmentation task.

We made use of the following augmentations [3, 2]:

- random vertical and horizontal flips with probability $p = 0.5$
- random rotation of images of up to ± 90 degrees
- up to $\pm 10\%$ change in brightness, contrast and saturation
- randomly adjusting the sharpness of the image with a probability $p = 0.5$

We notice on Fig. 7 that with such image augmentation, the validation loss decreases below the training loss. This is because data augmentation acts as a regularizer.

Even if data augmentation seems to worsen the F1-score estimated on the validation set it has allowed us to generalize better to new test images and to achieve our best F1-score (83.2%) on the AICrowd test set.

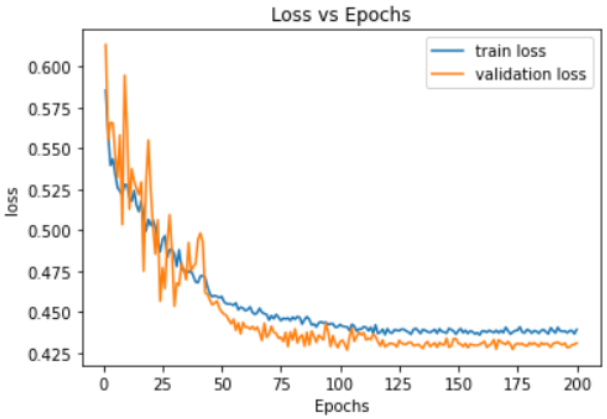

Fig. 7. Training and validation loss curves with data augmentation (200 epochs)

F. Performance on Test Set and Submissions

For the submissions, we trained the neural network using all 100 training images. Then, we made our predictions on AICrowd's test set consisting of 50 Google Maps satellite images of size 608×608. Without data augmentation, we had an accuracy of 89.5% and an F1-score of 81.2%. Data augmentation, allowed us to reach an accuracy of 90.5% with an F1-score of 83.2%. The figure 8 displays the predictions we had for the road segmentation of two images from AICrowd's test set using our CNN with data augmentation.

G. Perspective of improvements

Naturally, there is much room for improvement in this project. Here are some areas we can investigate.

1) Further augmentation of data: Our CNN model should be able to work on satellite images of different scales. Using data augmentation we could generate augmented images zoomed in more or less on the center of the patch. The difficulty we faced in implementing this idea is that by zooming in more or less we ended up with augmented images that are not consistent in terms of size.

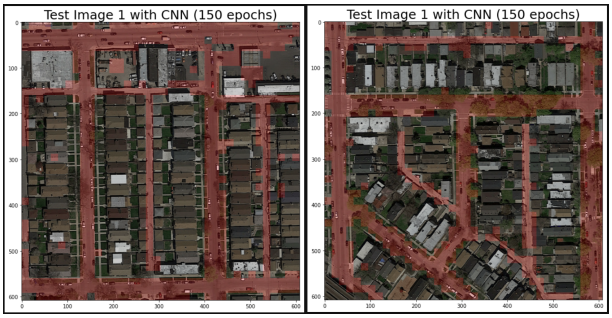

Fig. 8. Predictions for the road segmentation of two test images

2) Cross Validation for hyperparameter tuning: We are mostly satisfied with our choice of hyperparameters: the dropout (20%), the learning rate (10^{-3}), the size of the frame surrounding the patch (64×64), the slope for Leaky Relu (0.01), etc. They were chosen on the basis of their performance on the validation set and over several independent training experiments, each time picking at random a training and a validation set for evaluation (see IV-D). Our approach could be improved by optimizing them jointly. We could also make use of a cross-validated grid-search methodology to reduce the variability of our estimations of the performances.

3) U-Net: U-Net architecture is well known for its performance for the segmentation task and in particular the road segmentation task. We may want to explore this kind of model to compare it with our approach using CNN.

4) Investigating other metrics: We could try evaluating our performance with other metrics, for example, Jaccard similarity, and see if our performance remains the same. This way we could directly evaluate the similarity between our ground-truth images and our predictions.

5) More Data: Even with having augmented our data in almost every way we could think of, it is still fairly limited since we only have 100 images to train our neural network. What we could do is to gather more images from other sources to significantly increase the performance of our neural network.

V. Conclusion

Our baseline approach in III was essential to deeply understand the training dataset and was at the root of the way we decided to use data preprocessing in our CNN model (see section IV-A). These steps have been essential to building a CNN model that was robust and efficient on new test images, in particular, by normalizing over the patches, oversampling the roads and considering feeding our model with larger frames surrounding each patch. By evaluating our performance locally (sections IV-C and IV-D) on validation sets, and over several independent trainings we could understand better the role of each parameter in the performance of our model and optimize them. Making use of all these techniques, we could predict the road segmentation of new Google Maps satellite images with an Accuracy of 90.5% and an F1-score of 83.2% (see IV-F).

References

[1] Eliott Brion. *Retina Nerve Segmentation: Preventing Blindness with Deep Learning*. 2017. URL: https://github.com/eliottbrion/retina_nerve_segmentation.

[2] *Illustration of transforms — Torchvision 0.11.0 documentation*. URL: https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py (visited on 12/22/2021).

[3] Sergei Issaev. *Beginner's Guide to Loading Image Data with PyTorch*. en. Dec. 2020. URL: https://towardsdatascience.com/beginners-guide-to-loading-image-data-with-pytorch-289c60b7afec (visited on 12/22/2021).

[4] Jake Krajewski. *Towards Data Science*. en. Jan. 2020. URL: https://towardsdatascience.com/pytorch-layer-dimensions-what-sizes-should-they-be-and-why-4265a41e01fd (visited on 01/11/2020).