CS-433 Project 2: Road Segmentation

Chitrangna BHATT, Pierre LIORIT and Héloïse DUPONT DE DINECHIN
Department of Computer Science, EPFL, Switzerland

Abstract—**Road segmentation is a very common problem in Machine Learning and it has many applications: recognizing particular patterns from satellite photographs is often a hot topic. Here, we work on a set of satellite images acquired from GoogleMaps: our goal is to perform road segmentation, that is training a classifier to segment roads. Therefore, we output a label for each 16 by 16 patch from each given image.**

Here, we work on sets of methods: firstly, we try polynomial and ridge regression, which give some results but are quickly limited. Than, we use a neural network. We focus on selecting the good features, enhancing our dataset and post processing our output. In particular, we apply Image Processing techniques.

We obtain best results with neural network, even if the time of computation limits our tests. We end with a quite good model, which could be easily enhanced in the future, with more computational time and memory.

I. Introduction

Our task is to train a classifier to segment roads in the given images: it will assign a label 1 for roads, and 0 for background to each pixel. To set of images are given: the training set, which also contains mask images corresponding to labels, and the test set. Working with images has the advantage that results can be easily visualized. For example, see fig 1 what we can obtain for an image.

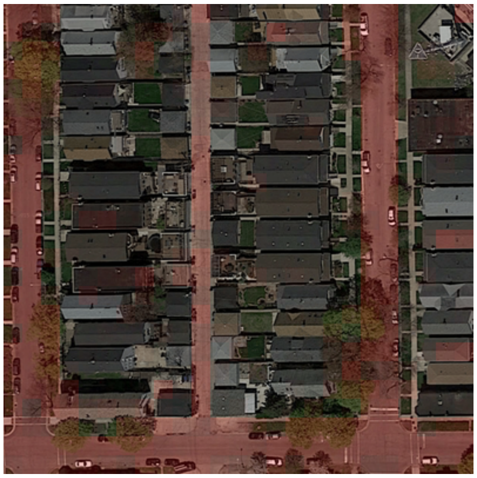


Fig. 1: One of our result obtained

II. Methods

A. Data analysis and preparation

1) Problem input and output: We have at our disposal a set of 100 images (400x400 pixels each) with corresponding ”ground truth”, that is a black and white handmade image, **white pixels standing for a road pixels** and black pixels for non-road pixels.

The output should be a similar black and white image, for each of the 50 images of a non-annotated test set. An important detail is that the final submission is rougher, since the output is separated into **patches of size 16x16 pixels** (all the pixels of a given patch have same annotation.

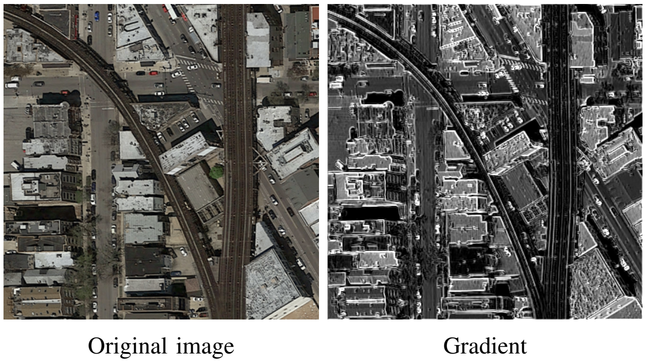
We can have two different approaches here : either making a **pixel-per-pixel prediction** and then patching (by considering the proportion of white pixels in the patch), or transforming the input image into a ”vector of patches” and **only manipulating patches all along the algorithm**. We preferred the first approach in the polynomial regression, and the second one for the neural network.

2) Choice of the features: A good model only works if we feed it with correct data. Especially when we use polynomial regression, the choice of the input data turned out to be crucial. Thanks to manually annotated data, we are able to compare different correlations between input features and expected labels.

When the input is an image, the first ideas coming to the mind are **basic statistics of RGB code**, such as average and variance, per patch.

An easy feature augmentation is also to keep the concatenated vector of statistics concerning the 4x4 sub-patches.

The goal of this project is to detect roads. Intuitively, the colors on the road should be less contrasted, and more grey, than landscape colors; and the edge detection could help the detection. This why we considered **adding gradient and second derivative images** (see fig 5), first in shades of gray, and then in RGB dimensions. This approach is particularly crucial in the case of polynomial regression, because these are features that cannot be approached by any polynomial approximation.



Original image Gradient

Fig. 2: Transforming the image

3) Data augmentation:

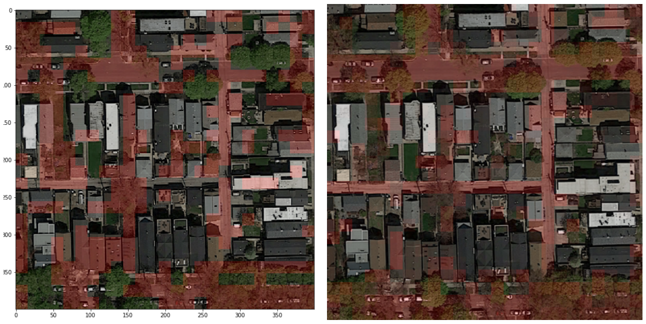
"There is no data like more data."

Based on this principle, we try to increase the amount of available data, by using usual transformations : **rotations and symmetries**. Because of weak computational power, we limit this increase to a factor of four.
The file $feature_{a}ug.py$ first rotates each image with an angle of $\frac{-\pi}{2}$, which gives us a second image. Second, it chooses a random symmetry (central, horizontal axis or vertical axis) an applies it to the original image, giving a third image. Third, a random symmetry (possibly different) is applied to the rotated image, that is a fourth image. With a more powerful hardware, we would have used **all** possible transformations, but our program with neural networks already required 5 hours to train without using rotated and mirrored images. The maximum number of images used for training was 300.
One notable advantage of using a $\frac{-\pi}{2}$ rotation is to have a perfect symmetry between vertical and horizontal axis : North-South roads have as much weight as West-East ones. We reduce thereby a possible bias of the training set.

4) Data enhancement: For machine learning on images, it is often useful to process the images upstream. Here, we chose to try a simple Gaussian filter, to decrease the possible noise. It would have also be possible to apply edges enhancement filters.

B. Choice of the model

1) Polynomial and ridge regression: As suggested by the organization of the course and the provided template, our first approach relied on **polynomial approximation**. We varied a lot the nature and amount of features describing a patch, but in any case each input vector was *only* depending on the inside of the patch, with no interaction with neighbor patches (except for gradient calculation).



With ridge regression With neural network

Fig. 3: Comparison of results
s

2) Neural network: Once again, we start our work using the file *tf_aerial_images.py*, and try two image-to-input-vector projections.

C. Optimizations

1) Parameter optimization: The tons of parameters of a neural network are very well optimized, but some extra-parameters need to be optimized, too. It is the case of the threshold.
As we are putting some labels on patches of size 16x16, we have to decide the percentage of "road pixels" required for a patch to be a "road patch". This proportion, called threshold, is used twice :

1) When we get the training set, and turn the pixel image into a list of labels. This case doesn't happen if we work with pixels and just convert the output prediction into a patch prediction at the very end.
2) When we convert a pixel result to a patch result.

In both cases, this is important to find the right threshold. By cross-validation, we tried different threshold values and optimized this parameter in the different cases. Concerning the threshold used to get the training set, we had the idea not to have 0-1 inputs, but float labels instead, representing the proportion of road pixels in the patch. The graph corresponding to the optimization of this threshold (through cross-validation), is shown below. We can clearly see the balance between precision and recall.

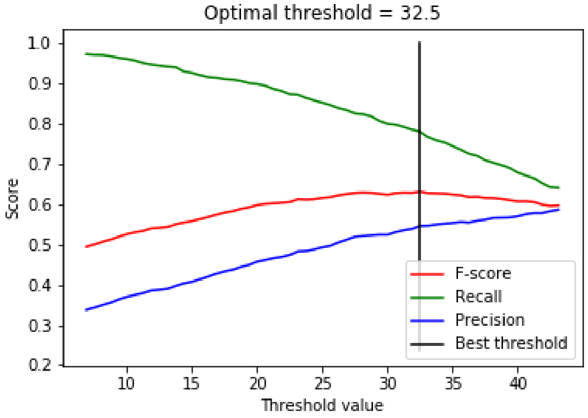


Fig. 4: Threshold optimization for prediction

D. Output

1) Submission: The wanted submission is a .csv file containing all the patches from all the images with their corresponding labels. It is not directly obtained from the machine learning algorithms: they both output images masks, were white correspond to road and black to non-road. For the different models used, we chose to output also this masks in grey scale: they can be post processed for better accuracy.

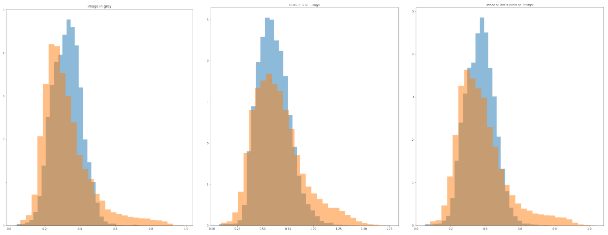
2) Post processing: Then, the aim is to go from a grey-scale image to a black and white image. The simplest way is to chose a certain threshold. By cross-validation, this threshold is optimized.

It is possible to go further: some filter can be applied to increase accuracy: indeed, we know for example that a road is not likely to be a single pixel surrounded by non-ground pixels. We chose to apply some convolution filters to take into account the neighbours of every pixel to output its result. To go even further, we also train a neural network with grey-scale masks, so that it would be able to to this ”post-processing” in a cleaver way.

III. Results

A. *Features Selection*

As said in II, we chose to add two features: the images' gradients and the second derivative. We plot histograms to select interesting outputs. It seems that the RGB values, grey value and second derivative are the most discriminating features. They are the one we chose to use.



Grey Color Gradient Second Derivative

Fig. 5: Histograms for different features (Color=labels)

For the polynomial model, the score results are indeed increased when gradient and second derivative are added (F1 score from 0.475 to 0.506). It is still less accurate than the RGB values: when trying our neural network on second derivative images only, a lower score is obtained (0.636 vs 0.655).

The Gaussian filtered images give pretty good results: when training our neural network with Gaussian-filtered images, the F1 score is slightly better. It could have been interesting to find an optimum Gaussian filter, but we only tried on σ value.

1) Polynomial model: The quality of the results naturally increase with the degree, the amount of the data being too big for over-fitting it with just a dozen of coefficients. The F-score jump from 0.463 for linear regression to 0.507 for degree 10 (with the same extraction function). Increasing the number of features induce also significant progress : with both second derivative and RGB code, we get out greatest score of polynomial regression, that is 0.537.

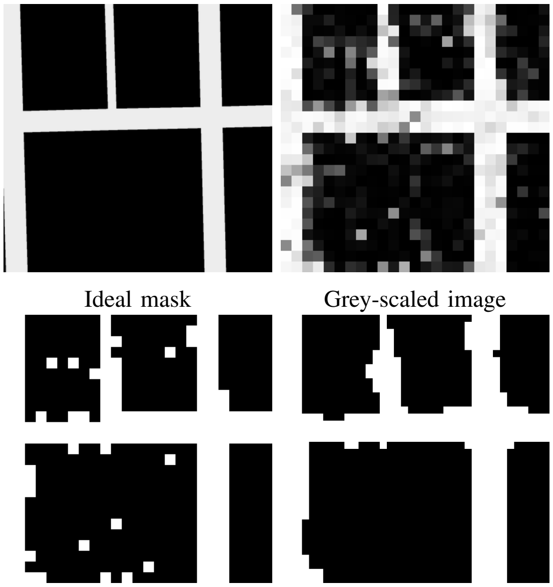
2) Neural network model: Even without adding any features, we obtain better scores as long as the number of epochs is large enough. Without any data augmentation nor filtering, the score is 0.663. With post-processing, this score increased by 0.712.
With an augmented data set, the unfiltered prediction doesn't increase significantly (F-score is 0.667) but the filtered result is better (F-score is 0.731).

3) Merging both results: We simply average the results, with a weight coefficient reflecting the reliability of each model. The average is then filtered by post-processing. The result was the best we obtained, reaching a 7.45 F-score.

B. *Output*

Firstly, the transformation between grey scale masks to black and white mask is done by setting a threshold. As mentioned above, we determine optimal values for the threshold adapting cross-validation.
Secondly, we perform a filtering of the grey-scaled outputs to increase our results' quality. As you can see fig 7, it does delete many false positive, in particular the pixels that were alone. The filter is very simple: it is a matrix of size 5 by 5, giving weights to the neighbours.

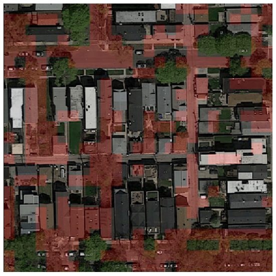
The result is still not perfect: this is why we try to find the function transforming from grey-scaled to black and white by machine learning. Sadly, it doesn't perform well on our images (it only erase some isolated points). Therefore, we chose to keep the filtering method.



Ideal mask Grey-scaled image

Threshold applied Gradient

Fig. 6: Filter applied



Trained filter

Fig. 7: Filter applied

IV. Discussion

A.

The regressions quickly reach their limits:: they aren't best designed for dealing with images.

The neural network gives better results. However, it takes a lot of time to compute, which makes it difficult to work on. In fact, it has been one of our main difficulties.

Our models can be improved for sure: for example by playing on more parameters of the neural network.

This project has been a good occasion to practice several machine learning techniques. Having a project on images enabled us to play also with image processing, which is often used in Machine Leaning.

Acknowledgements

We are very grateful to the staff of the machine learning course, especially the teaching assistants who helped us, not only for this project but also for their help in general about machine learning concepts.