Road Segmentation of Satellite Images Using Convolutional Neural Networks

Jean Gillain
sciper no. 331411

David Desboeufs
sciper no. 287441

Mathieu Caboche
sciper no. 282182

***Abstract*—Outlining roads on satellite images can be useful for creating accurate maps of all four corners of the earth. Machine Learning enables us to automate this task. In fact, convolutional neural networks (CNNs) are a suitable tool for image recognition and thus road segmentation as well. In this project we design a CNN and evaluate the resulting model. We compare our various CNN designs to a baseline model obtained from a basic CNN.**

I. Introduction

II. The Google Maps Satellite Images Data Set

The data set we are given to train a model on consists of 100 satellite images from Google Maps. The pictures look like they were all taken from a very similar locations, probably one single city. As such, the given data set is not very diverse. Furthermore, it is quite limited in size. This explains why we need to enlarge our data set. Note that our training set is a fraction of the same Google Maps satellite images. Thus, we expect that our road classifier can be trained to perform well for segmenting roads of that particular city or similar cities. Its performance when applied to radically different roads and areas can however not be certified.

In section III-A we describe how the data set was enlarged and how we managed to train our CNN on a substantial amount of images, so as to yield proper results.

III. Models and Methods

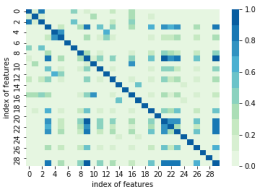

Figure 1: Correlation matrix of all features

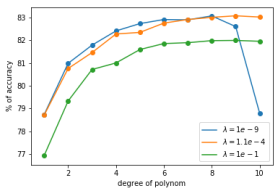

Figure 2: Accuracies obtained for different polynomial extension degrees and λ values

A. Creating a Large Data Set

We used a variety of methods and combined them in order to have a lot of data available to train our CNN.

Firstly, although we only have 100 images at our disposal for training, we actually train on smaller images that are subsets of the given images. For example, we can choose to train on images of size 20×20 pixels. This means that we get 400 training input sets for one single image (the given images have size 400×400). We are effectively partitioning large images into many small images, which has the added benefit of reducing the parameter count of our CNN.

Another common way to enlarge ou data set we used is to apply various transformations on the original images. By rotating, cropping and flipping the large scale images, we can obtain many variations of that image which prove very useful for training. Note that in this process we might lose image resolution, for example if we rotate and then crop. However that loss of resolution is effectively a zoom, another transformation. Any additional transformed images can only increase the sturdiness of our trained model.

We explored further ways of increasing the amount of input data we can feed to our CNN. One way to attain this is to give as input the edges found in an image along with the pixels of that image. We simply process the image with a Canny edge detector. This yields a gray-scale image indicating where edges are to be found. We do not extract vectors from the resulting gray-scale picture, but only give the latter's pixels as an additional input to our CNN.

B. Evolution of our CNN

$$f(x) = \log \frac{1}{1+x} \tag{1}$$

C. Model Validation

IV. Results

Regression Technique	Accuracy (Avg.) - STD
Least Squares GD (and SGD)	68.5% - 0.20%
Least Squares	74.4% - 0.25%
Ridge Regression	74.4% - 0.24%
Logistic Regression	65.7% - 0.29%
Newton Logistic Regression	65.7% - 0.29%
Our Best Regression	81.3% - 0.82%

Table I: Accuracy of Various Models Obtained by Our Regressions

V. Conclusion