DELFT UNIVERSITY OF TECHNOLOGY

Artificial Neural Networks

PROJECT PROPOSAL UPDATE

Authors:
Michiel Bongaerts
Marjolein Nanninga
Tung Phan
Maniek Santokhi

May 14, 2015

Contents

1. Introduction

Mapping the world around us has always been a human endeavour to advance economical output. A better understanding of the places around us makes for more efficient travelling and exploitation of the land. However, it has always been a very slow and tedious process to produce these maps, something technology has not changed just yet.

A new opportunity has arisen with the arrival of satellite imagery and an ever increasing amount of computational power. An opportunity where this mapping can be done automatically so that this tedious and slow job can be processed even more quickly and perhaps more accurately. It is with this in mind we further analyse any possibilities.

This paper proposes an update. Now a more straightforward approach has been chosen in which the emphasis lies on the actual Neural Network rather than the conversion of interpreted images to vector graphic maps. The new approach deals with image patches rather than pixels. This document discusses the newly acquired concept with a list of features and the actual implementation details. Also an updated schedule will be presented.

2. Concept

Earlier attempts to conceptualise the idea to automate map making resulted in a proposal that too heavily focussed on the actual map creation rather than the classification. For a Neural Network course this was deemed not befitting enough. The plan was also quite far reaching to start with. Thoughts were put into downscaling this ambitious plan. We played around a bit and came up with a new concept which will be discussed in this chapter. Firstly, an impression is given how the end user interacts with our system. This will lay the groundwork for how the Neural Network will be constructed. Secondly through the principles of MoSCoW a flexible requirements list is established. Actual talk about the classification is done in the subsequent chapter.

2.1 Impression

Below in figure 2.1 an impression is given of what the end user will interact with and a possible result that might come about from said interaction.

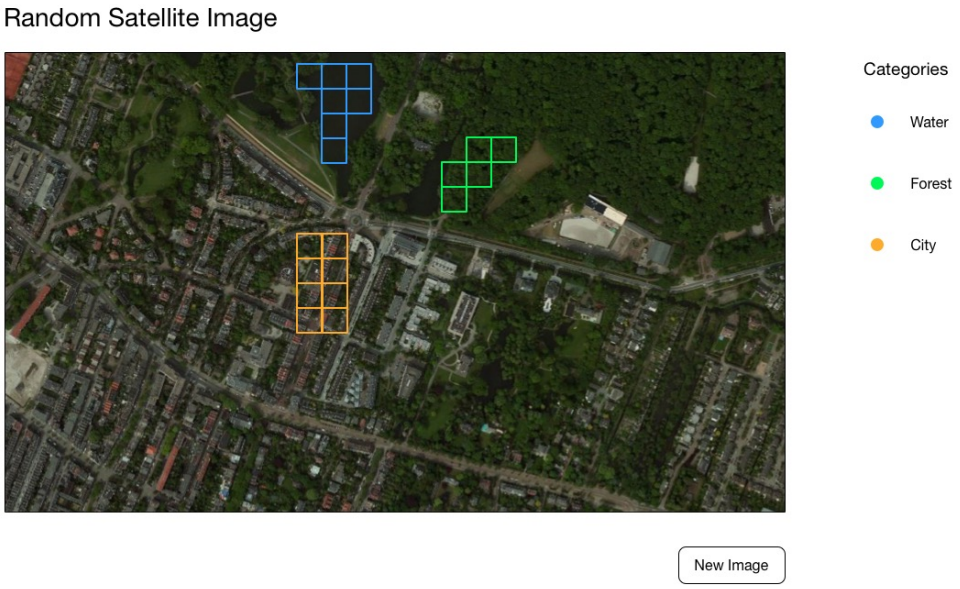

Figure 2.1: Impression of what the user interacts with and a possible outcome.

The most notable attention grabber is the satellite image. This image will be acquired through a random call from a homogenous satellite image database on every new instantiation of the system. Another possibility to acquire new data is by clicking on the 'New image' button.

Right of the satellite image one can see labels (Forest, City, Water) which correspond with the labels which are outputs of the classification.

Results obtained from our algorithm, given the current satellite image as input, are graphically feed back to the user. The impression above does that by showing a correspondence between image patches and their respective label via color coding laid over the satellite image. This rendition just shows a few islands of resulsts as an example. Normally the entire image will have such arching (which will be a lot more subtle).

Figure 2.2 shows how it works internally. Two grids are maintained. One with patches the size of 25 by 25 pixels. The other by the size of 50 by 50 pixels. The latter is actually fed to the Neural Network from which will be decided for that patch the percentage of type of label it contains. Four 50 by 50 pixels will be grouped together to create a square. In the middle the 25 by 25 pixels patch is placed. This patch will eventually be coloured on the satellite image to indicate the type of label. To decide that, a weighted majority vote of the four 50 by 50 pixels surrounding that smaller patch is computed. Like a convolution filter

this construction is shifted over the satellite image as to decide for every area on it. As one can imagine, 25 pixels at the borders are omitted.

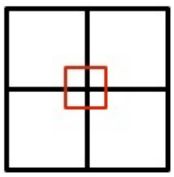

Figure 2.2: Grid structure which facilitates the algorithm.

2.2 MoSCoW

Since a limited amount of time is available and we thought of quite some experiments and features that could be added, we used the MoSCoW method to get our priorities straight and focus on the most important requirements. MoSCow stands for *Must have*, *Should have*, *Could have* and *Would have*. All the requirements are labeled in these four classes.

2.2.1 Must have

Must have requirements are critical to project success.

- Create code based on a *Convolutional Neural Network* (CNN) that enables automatic classification of patches from satellite images. Considering images acquired on provincial level and a substantial amount of pixels in one patch (at least 50 x 50 pixels).
- At least the following classes should be recognized: vegetation, city and water.
- Develop a way to visualize the automatic classifications clearly.

2.2.2 Should have

Requirements labeled as should have are important to book success, but not necessary for delivery.

- Create a clear interface in which the unlabeled images can be uploaded, and the output consists of labeled images.
- Calculate the uncertainty in the classification and ask user input for very uncertain patches.

2.2.3 Could have

It would be very nice if we would be able to reach the Could have feautures, but they are not critical.

- Experiment with pre-processed images (noise reduction, gradient calculations)
- Analyze images on city level, so with more details present. For this purpose new classes have to be added, such as roadways, cycle paths, buildings, distinct vegetations etc.
- Experiment with other models than the state-of-the art LeNet-1 CNN. For examples, a CNN in which Genetic Algorithms are incorporated, or implementing an Extreme Learning Machine for the training of the weights.

2.2.4 Would have

These requirements are implemented only in the most ideal situation. They are considered as the dream project, sometimes serving as a suggestion for further projects.

- Develop a method for high-detailed automatic vector graphics, in which segmentation of the distinct labeled classes is incorporated.
- Use the input of the users to improve the automatic classification.
- Sell the software package to Google.

3. Implementation

3.1 Dataset

HALLO KAN IEMAND HIER IETS TYPEN? :D

3.2 Convolutional Neural Network

When we sat down and thought about the network architecture, we had to keep in mind the complexity of the network. Dealing with satellite imagery meant dealing with a large amount of data per image. In turn, this meant a large amount of connections between the input from the image data to the hidden layer, which leads to a large amount of parameters to tweak. Without preprocessing the image to extract features, the number of training examples might be small considering the pattern dimension, resulting in overfitting easily. Manually extracting feautures, however, is often empirical and therefore suboptimal.

During our research phase, we encountered a variety of papers that touched on this subject. Most of these papers have one thing in common, which is their use of convolutional neural networks (CNN). Using convolution operations on small regions of the image, the previous problem could be avoided. Another advantage of using CNN is the ability to take into account correlations of neighboring input data, which are plentiful in satellite images. CNN also makes use of the principle of weight sharing in convolutional layers, which further reduces the amount of free parameters.

3.2.1 LeNet-1

The model we implement is LeCun's LeNet-1. This model uses an alternating sequence of convolution and sub-sampling layers. The architecture of this network is shown in Figure 3.1. The convolution layers act as feauture maps, they consist of a window of a certain size, whose pixels are trainable weights. The sub-sampling layers reduce the dimensionality of the outputs.

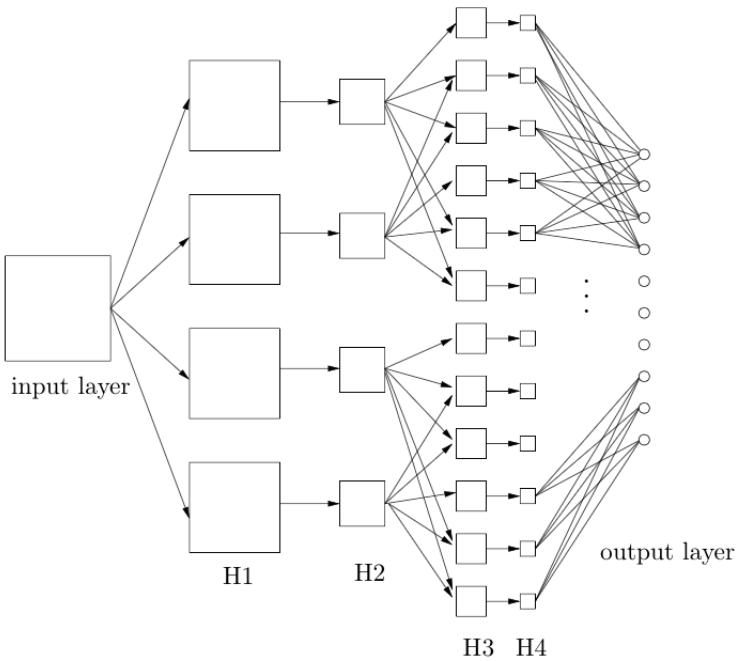

Figure 3.1: The architecture of the CNN proposed by LeNet. H1 and H3 are the convolution layers, H2 and H4 the sub-sampling layers to reduce the dimension.

One of the biggest advantages of using CNN, and especially LeCun's LeNet-1 implementation, is the incorporation of backpropagation learning. Meaning that all the weights of the layers are adjusted iteratively, eliminating the need to manually create the convolution masks.

Currently we are experimenting with input patches of 50x50 pixels. The layer H1 consists of 4 neurons with a window size of 5x5 pixels. H2 reduces the dimensionality by 2. H3 consists of two neurons per output, both of convolution kernels of 4x4 pixels. Finally the neurons in H4 reduce the dimensionality by 5, resulting in an output of 4x4 pixels. Using this architecture with set convolution weights and dimensionality reductions, so only training the output weights, the model can be trained to classify forest correctly.

4. Time schedule

Week	Description of work	Deadline	Labour (hours)
20	• Implement backpropagation • Start examining how to take into account RGB values • Train on more pictures and examine performance on other classes like city and water	Finish updated proposal	40
21	• Start developing frontend/ backend architecture • lala check check lala	Finish backpropagation	40
22	• In-between overall evaluation • Make a new planning which adjustments to implement • Experiment with preprocessed images (noise reduction)	Finish model with complete RGB values and the frontend/backend architecture must be completely thought over	40
23	Start report writing and implement last adjustments	-	40
24	Preparation for demo	Demo	30
25	Project finalization	-	30
26	Project finalization	Project report + individual document	40
			Total: 468