GI_Forum 2017, Issue 2
Page: 173 - 188
Full Paper
Corresponding Author:
geometalab@hsr.ch
DOI: 10.1553/giscience2017_02_s173

OSMDeepOD - Object Detection on Orthophotos with and for VGI

Samuel Kurath, Raphael Das Gupta and Stefan Keller
Geometa Lab at Institute for Software, HSR, Switzerland

Abstract

A great deal of the interesting information captured by aerial imagery is as yet unused, even though it could help to enrich maps and improve navigation. For this information to be made available, objects such as buildings or roads need to be recognized on images. This is laborious to do entirely manually, but non-trivial to perform computationally. In this paper, we present an automated method for detecting objects of a chosen class (pedestrian crosswalks) on orthophotos, a method which can be adapted for various classes of objects. The method uses a supervised machine-learning approach with a deep convolutional neural network. We re-trained the final layer of a pre-trained neural network using specific imagery and crowdsourced geographic information from the OpenStreetMap (OSM) project. The result is an easily enhanceable and scalable application which is able to search for objects in aerial imagery. We achieved an accuracy of well over 95% for crosswalks and promising preliminary results for roundabouts.

Keywords:

visual recognition, deep convolutional neural networks, aerial imagery, VGI, parallelism

1 Introduction

As in many other fields, machine learning has become a hot topic in geography and geoinformation technology, including VGI. It can be used to derive conclusions from geoinformation or to derive geoinformation from geodata, e.g. feature positions from raster data using image recognition techniques. In this work we focus on the latter.

As a proof of concept, we chose the task of identifying registered Swiss pedestrian crosswalks in OpenStreetMap using satellite and aerial imagery (orthophotos). To do this, we had to devise a method to detect crosswalks as well as a way to feed results back into OpenStreetMap while adhering to the OpenStreetMap community's quality standard for edits made or reviewed by humans.

The objective was to come up with a process which accomplished these aims and which could be adapted easily for other feature classes. Furthermore, in order to keep

OpenStreetMap up-to-date, the process needed to be replicable using new orthophotos when they become available.

This article is in 5 main sections. Section 1 is introductory. In section 2 we describe our method, and in 3 we present the results of applying the approach to pedestrian crosswalks in Switzerland. We discuss both method and results critically in section 4. Section 5 summarizes what this work achieves and outlines possible future research.

Motivation: to increase crosswalk coverage in OpenStreetMap

OpenStreetMap®[1] is a worldwide community of mapping enthusiasts, and in the words of Neis, Zielstra, & Zipf (2012) 'a well-known project in the field of Volunteered Geographic Information (VGI)', a term coined by Goodchild (2007), also known as 'crowd sourced geodata'. It makes available to the public not just a web map, but also the underlying database of geographic information, which anyone can edit.

Although the OpenStreetMap project initially set out to create a free street map (hence its name), today it collects all geographical information (using vector geometry with attributes/tags) deemed interesting by the community members, as long as that information is verifiable 'on the ground'. Thus, many features not displayed on a typical street map can be found in the OpenStreetMap dataset, including the positions of pedestrian crosswalks on a road axis.

While pedestrian crosswalk locations are vital for assisting pedestrian navigation and might also become important for automated driving, a crosswalk existing in reality but missing from the dataset will not stand out as much to most mappers as a missing road would. This makes it difficult to improve the completeness of the crosswalk location information in the dataset, even for mappers interested in this particular topic.

Satellite and aerial imagery are used to assist data collection when surveying in the field and for directly tracing recognizable objects remotely. However, manually searching the orthophotos of even just a small country like Switzerland for all instances of one feature such as pedestrian crosswalks would be a very time-consuming and tedious task. We decided to help improve the dataset's completeness by providing more automation, thereby facilitating targeted editing.

Challenge: Automated visual object detection

Although humans are very good at visual pattern recognition, accurately detecting and identifying objects on a picture can be difficult or (depending on light conditions, shadows, and other objects that obscure the interesting aspects of the images) impossible, even for them. Consider for example the orthophotos in Figure 1 showing pedestrian crosswalks, and compare them to the orthophotos in Figure 2, which do not show crosswalks.

1 http://openstreetmap.org

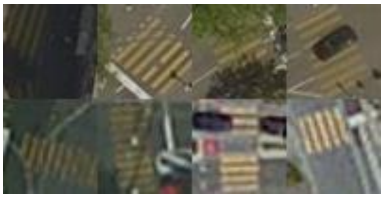
Figure 1: Crosswalk examples

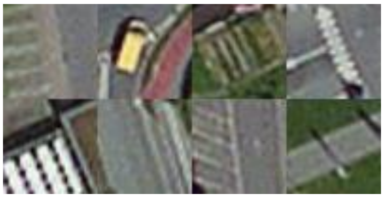
Figure 2: Non-crosswalk examples

Devising a computer program to do the image recognition is particularly difficult, as traditional rule-based programming usually falls flat in this area.[2] However, recent developments in artificial intelligence (AI) research have made computational object classification feasible. Artificial neural networks, a computational approach that loosely mimics the way a biological brain works on the neuron level, have been a subject in AI research for a long time (e.g. Rosenblatt, 1957; for a brief history see Russell, 1996). In our case, the input is an image (a small part of an orthophoto) and the desired output is a classification of the object(s) seen there, albeit a simple one: a crosswalk or not a crosswalk.

In recent years, the best results in image recognition have been achieved by deep convolutional neural networks (Krizhevsky, Sutskever, & Hinton, 2012). A deep convolutional neural network (dCNN) is a kind of further-developed neural network, designed to mimic the visual cortex. The networks get their name from stacking many convolutional layers. A convolutional layer has 'neurons', and each layer has a convolution matrix (called a 'kernel' or 'filter') as its learned parameter. Each of these layers acts as a feature-detector working on a fairly local feature set. For the first layer, the input feature map comprises simply the pixel colour values. Early layers will recognize simple, locally-confined features like lines, edges and patches of colour. Each successive layer will recognize features that are visually more and more complex as arrangements of simpler features from the previous layer(s), until the output layer produces the classification of the whole image.

The chosen approach was not without challenges. We had to: (1) get enough training data for the neural network; (2) reduce the neural network's training time sufficiently to allow for

2 For the specific case of pedestrian crosswalks, one might expect a spectral-analysis-based approach to work. The frequency-based methods we evaluated in Keller, Bühler, & Kurath (2016), however, failed to perform better than the deep learning-based one, with only fast Fourier transform coming close.

training the network with different parameters, so that we were able to improve and fine-tune the result; (3) deal with a huge amount of satellite imagery, and (4) make the results available in OpenStreetMap without breaking the community's rules against automated imports.

Related work

Mnih & Hinton have applied machine learning to orthophotos for per-pixel labelling for some time now. They have improved road detection with the help of a deep network (2010). They have also studied the effects of noisy data and the lack of learning data, and devised methods that allow the training process to better cope with this (2012). This is particularly relevant when learning from VGI, where '(asymmetric) omission noise' (object visible on orthophoto, but missing from the VGI) and 'registration noise' (different locations of an object on the orthophoto vs. the VGI) are prevalent.

Chen & Zipf (2017) have applied deep learning to detect buildings by using positive learning samples from buildings already mapped in OpenStreetMap, and negative samples from data generated by the MapSwipe mobile app. In this app, users indicated that an area did *not* contain buildings. The authors turned this supervised learning setup into an active learning one by manually labelling the tiles where their network disagreed with labels provided by MapSwipe, using these samples to re-train the network and thereby significantly improve its performance,[3] though not quite reaching the performance level of MapSwipe volunteers.

In contrast to these studies, we did not devise our own neural network design. Instead, we chose an existing network which had already proved successful for (non-orthophoto) image classification, and focused on how to get training data, how to avoid having to train a full dCNN, and how to automate and computationally optimize the procedure.

2 Method

To detect objects on orthophotos we applied the approach visualized in Figure 3. In the following subsections, we will describe the individual steps in depth.

3 Here we are referring to precision, recall, F1 score and accuracy, not the computational performance.

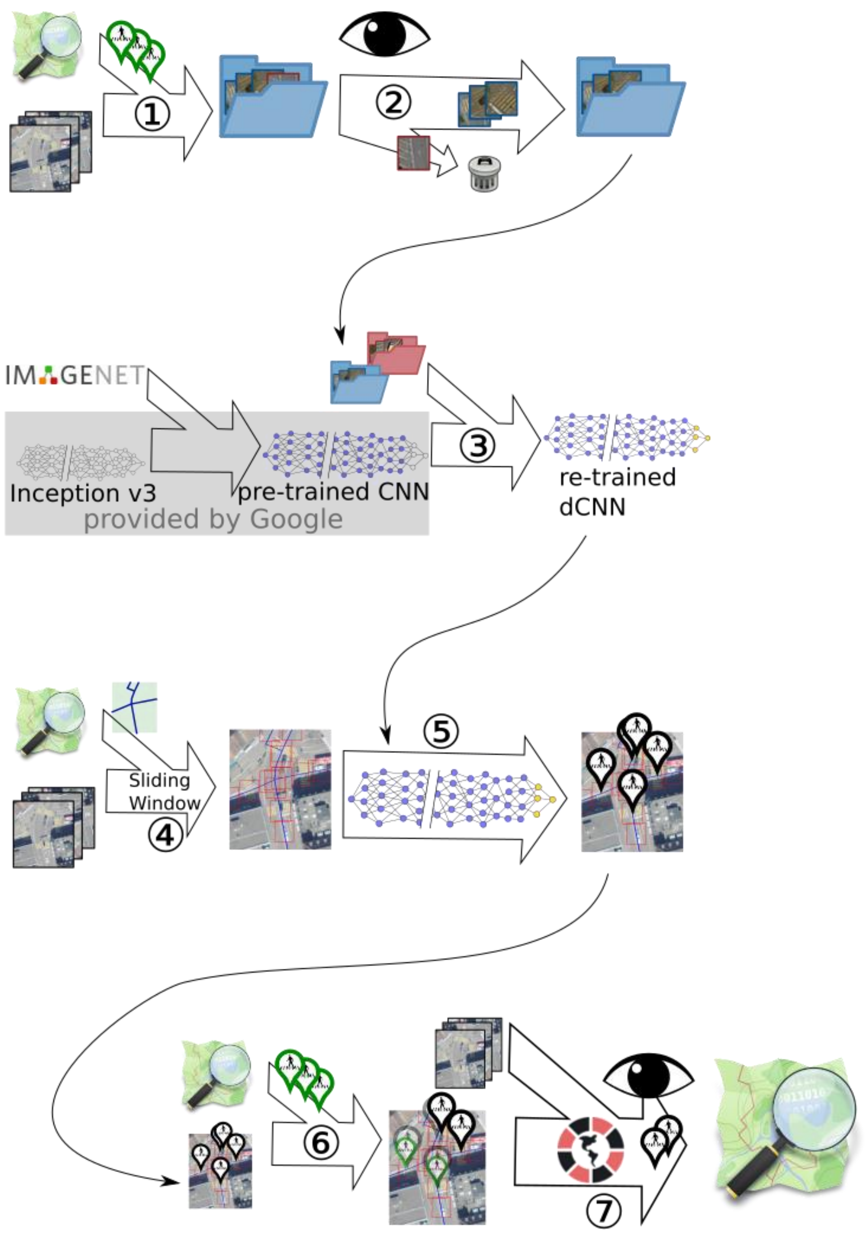

Figure 3: Method overview. Positive samples comprising (1) orthophotos at locations of known crosswalks (from OSM), from among which (2) false positives have been discarded manually, and semi-manually collected negative samples (red folder icon) are (3) used to re-train a pre-trained dCNN. Next (4), orthophotos along OSM streets are then (5) classified. Crosswalks already in OSM are (6) pruned from the results and those with identical or nearby coordinates are merged. The remaining results are then (7) fed through MapRoulette for crowd-sourced review and for manual integration into OSM

At step (1) in Figure 3, we select images from a satellite imagery source with the help of OpenStreetMap. These images serve as the dataset for the training mechanism of the neural network. The dataset consists of a positive and a negative subset, containing images with or without the objects we are looking for respectively. In step (2), false positives are manually removed from the automatically collected positive subset.

This dataset is then used to retrain a neural network (step (3)), which can classify images into the different dataset categories (5). The images to classify are obtained by moving a sliding window over the area of interest (4).

The midpoints of the windows where objects of the class of interest have been recognized by the neural network are aggregated (to eliminate duplicates) and treated as the object locations. After removing locations corresponding to instances already in the OpenStreetMap data set (6), we use the coordinates to generate tasks and challenges for MapRoulette. MapRoulette users then review the detected objects and add the genuine ones into OpenStreetMap (7).

Acquiring training data

A huge amount of data is needed to train a large neural network like a deep convolutional neural network. Significantly less training data is needed for re-training a pre-trained deep neural network (see next section), but the number of examples should still be in the thousands (Chollet, 2016). To efficiently obtain a training set of sufficient size, we combine existing OpenStreetMap data with aerial imagery. Using the fact that *some* pedestrian crosswalks are already mapped in OpenStreetMap and that many of these are visible in aerial imagery, we are able to extract sections from the orthophotos at the positions of crosswalks that are already present in the dataset (see Figure 5 and (1) in Figure 4) These sections are likely to show a crosswalk and thus are promising to provide a solid, positive training set. To obtain the crosswalk positions in OpenStreetMap, we query Overpass (OpenStreetMap Wiki contributors, 2017b), an easy-to-use application programming interface (API) for accessing OpenStreetMap data.

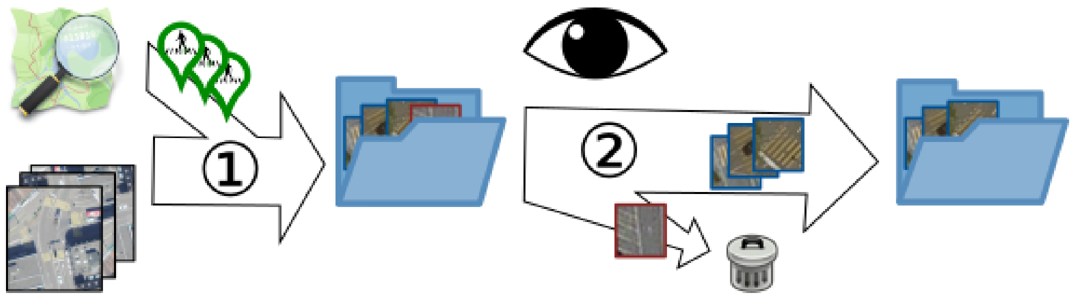

Figure 4: Positive samples are obtained by (1) using orthophotos at known crosswalk locations, and (2) manually removing false positives and unusable images

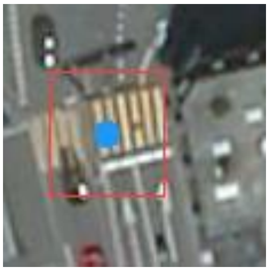

Figure 5: Visualization of the training data selection. The position of a crosswalk in OpenStreetMap is indicated by a blue dot. The red square is the border for the 50×50-pixel section to be cropped out of the aerial imagery

Unfortunately, a manual check of the collected images ((2) in Figure 4) is still needed before they can be used as training data, since the result is not perfect. The object in question might not be visible on the image at all, due to either OpenStreetMap or the aerial imagery being out of date, or because of different translational offsets placing the object outside the cropped-out section. For our example of detecting crosswalks, we also decided to discard images where a car, a shadow or a tree covered the target, as is the case in Figure 6.

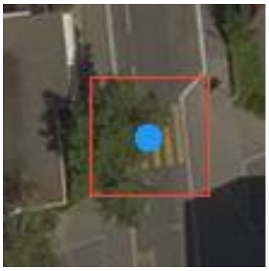

Figure 6: Example of a crosswalk half covered by a tree, which should not be used in the training set

Having collected the positive samples with the target object on them, to complete our training set we also needed negative samples. Because the search algorithm is more likely to encounter areas without the target object, and because these areas will vary more broadly than the areas containing the target object, we needed even more (and more varied) negative samples than positive ones.

To gather negative samples efficiently, we created an application which takes small screenshots along the user's cursor movement. Using an aerial map, these samples can be collected simply by moving the cursor over the map at the correct resolution (zoom level) (illustrated in Figure 7). Of course, hovering over the target objects must be avoided.

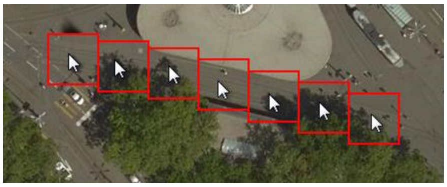
Figure 7: A visualization of the sample acquisition process with a cursor moving over satellite imagery. Every red square represents a cropped-out image

The combination of these two methods to collect images allows us to produce a dataset within a reasonable time and is applicable to a wide variety of objects.

Re-training a pre-trained dCNN

Deep convolutional neural networks are suitable for our detection task, but very deep networks with many hidden layers require a lot of training data. With parameter counts of around 20 million (Chollet, 2016), these networks are prone to over-fitting; that is, if the training set is not large enough, the networks are unable to generalize enough to handle unknown situations. The large amounts of time and effort needed to acquire these quantities of training data, and the long processing time required to train a network with them, can be avoided by resorting to an already-trained neural network, simply re-training the last layer for the new classification task (Figure 8). This process is called transfer learning (Yosinski, Clune, Bengio, & Lipson, 2014; Oquab, Bottou, Laptev, & Sivic, 2014)) and is possible because the first layers of convolutional neural networks perform very generic tasks like edge and shape detection (Zeiler & Fergus, 2013). The 'How to Retrain Inception's Final Layer for New Categories' guideline from TensorFlow (Google, 2016) advises how to do this. We found that the re-train approach reduces the training time on a Tesla K40m GPU by up to four hours.

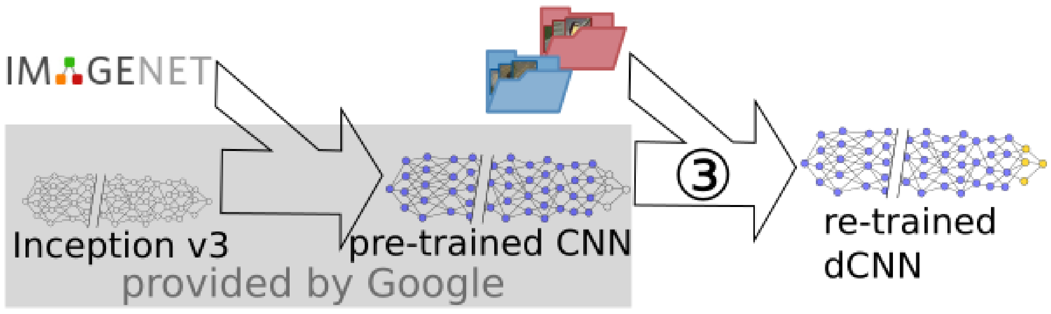

Figure 8: Re-training (3) of the Inception v3 network already pre-trained on ImageNet data

To handle the detection part of the process, we used TensorFlow[4], an open source software library for numerical computation, machine learning and simplification of GPU usage. TensorFlow provides various pre-trained neural network models for image classification. For our task, we used a pre-trained Inception-v3 model with 23,853,833 parameters (Chollet, 2016). The network is learned from the images of the ImageNet academic competition, which has about 1,000,000 pictures divided into 1,000 categories. Inception-v3 achieves a 'top-5' error of just 3.58% on the validation set on this competition. That means that the network had to predict the five classes most likely to describe the image, and that the image's real class was amongst these five predicted ones in 96.42% of all cases. To put this into perspective, Andrej Karpathy went through a subset of the validation set himself (Karpathy, 2014) and scored a human top-5 error rate of 5.1% (1.52% worse than Inception-v3). As a proof of concept, we re-trained the Inception-v3 model (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2015) on 8,242 images of crosswalks and 40,463 images of non-crosswalks, which reached 98.5% accuracy on our validation set.

Object detection: Searching for crosswalks

The detection procedure for the search for crosswalks is structured as follows: the area to be analysed is passed to the system as a large bounding box; this large bounding box is then split into smaller bounding boxes, which are added to the queue of jobs which have to be processed.

If a so-called 'worker' starts processing a job, it collects all streets from OpenStreetMap for that job's (small) bounding box ((4) in Figure 9). If there are no streets in the current small bounding box, the job is already finished, or downloading the corresponding orthophotos (Figure 10 (a)) is still incomplete.

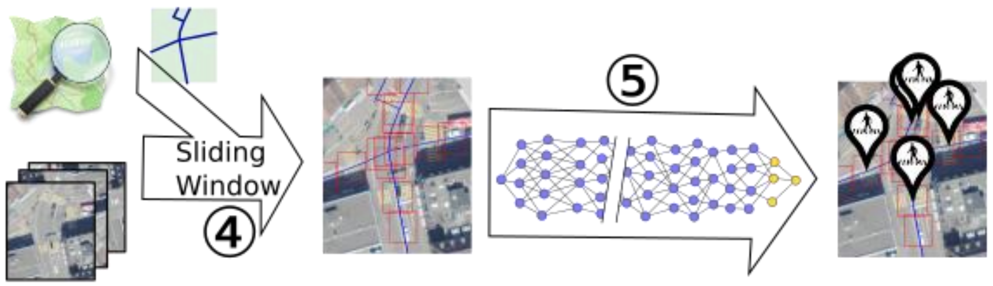

Figure 9: A sliding window along OSM roads (4) determines the orthophoto excerpts to be used as inputs for the classification by the dCNN (5)

4 https://www.tensorflow.org/

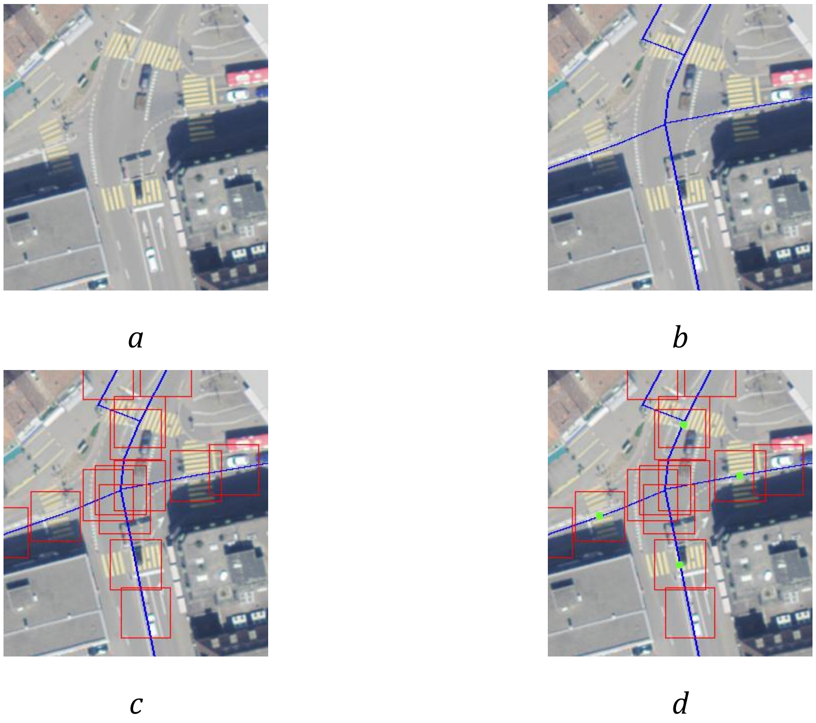

Figure 10: The steps for object detection using sliding windows along streets. (a) Aerial imagery for one job's bounding box; (b) Corresponding street data from OSM (overlaid as blue lines); (c) 50×50-pixel cutouts (shown as red squares) along the streets; (d) Some of the cutouts (centres marked by green dots) have been classified as containing crosswalks.

Figure 10 (b) shows the orthophotos overlaid by the street data. Along the streets, the worker has to cut out 50×50-pixel images from the orthophotos, which is the size the convolutional neural network needs as its input parameter (Figure 10 (c)). Finally, the detection algorithm decides for each image whether it contains a crosswalk or not. Detected crosswalks are shown as a green dot in Figure 10 (d) and as icons in Figure 9.

When the classification has been completed, all detected crosswalks within a radius of 5 metres are merged (part of (6) in Figure 11), because it is possible that some crosswalks are larger than one cut-out image and may appear in neighbouring or overlapping images. Furthermore, our mode for searching along streets isn't yet sophisticated enough to avoid searching areas twice where streets cross each other.

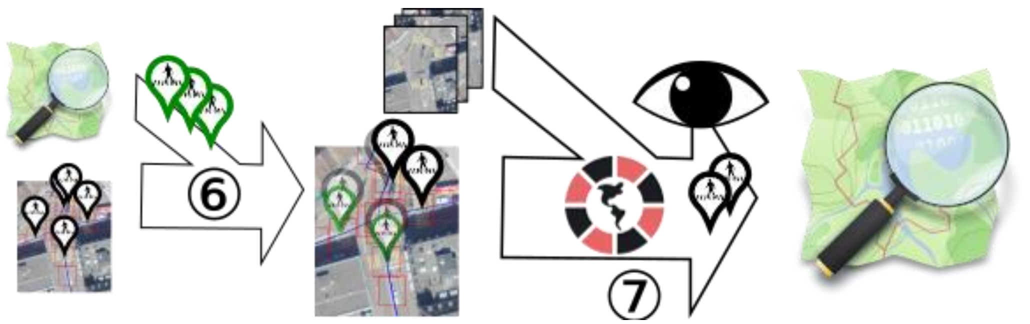

Figure 11: Detected crosswalks close to each other are merged, and those close to existing ones in OSM are eliminated (6). The remaining crosswalks are vetted by MapRoulette volunteers and manually added to OSM if deemed genuine (7).

Optimizations

When faced with the task of finding all crosswalks in Switzerland, we are confronted with an area of 41,285 km², which would have corresponded to roughly 400 million images of 50×50 pixels at a resolution of 0.3 m per pixel. To speed up the process, we parallelized the image recognition and also reduced the amount of data that needs to be classified.

TensorFlow already makes use of vector processing and multiple cores within the classification of a single image and for training. However, to use our hardware to full capacity, we also analysed several images in parallel by using a message queue[5] from which multiple processes (called 'workers') pulled their tasks.

Because our detection algorithm is confronted with a large number of images to classify, we looked for opportunities to reduce that number. One technique was to analyse images along streets only. This is sufficient when looking for crosswalks or roundabouts, as these can only occur along streets. Thus, this approach decreases the number of images significantly without deterioration of the detection rate. When this option is enabled, our algorithm checks for known streets before downloading the orthophoto tiles, so that only tiles along streets are downloaded. To get the street positions, we again use OpenStreetMap with the help of the Overpass API (OpenStreetMap Wiki contributors, 2017b).

The step size for moving the sliding window along the streets or over the complete orthophoto tile also heavily influences the number of images to classify. We chose a translation shift of 0.66 of the window's side length and found that this gives enough overlap not to miss too many features.

One way to reduce the data further is to work on the lowest resolution level of the orthophotos that is still sharp enough to represent the important features of the objects to be detected, resulting in fewer and smaller images to be analysed. For crosswalks, this turned out to be zoom level 19. It seems that for roundabouts, zoom level 18 might suffice.

5 specifically, RQ (http://python-rq.org/)

Crowd-sourced verification and integration

The goal of our work is not only to detect objects on orthophotos, but also to make the newly acquired information available to the general public, and specifically to the OpenStreetMap community. An obvious way to achieve this is to integrate the newly-found crosswalks into the OpenStreetMap dataset. However, OpenStreetMap only accepts data which are verified by humans, so just dumping the data into an automated import software was out of the question. Even though the detection rate is very high, there are still false positives in our detected crosswalk data and we do not want them to contaminate the OpenStreetMap data. Further, in OpenStreetMap, a crosswalk is tagged on the individual node (i.e. point) of the way (linestring) representing the section of the street that it crosses (OpenStreetMap Wiki contributors, 2017c), information which is not available in our results dataset.[6]

There are gamification tools that integrate data into OpenStreetMap or help users to edit data. One of them, MapRoulette, is an open source project and web platform on http://maproulette.org/ which allows everyone to create challenges with multiple tasks. These challenges can then be solved by other users (or 'the crowd') by editing OpenStreetMap using their own accounts. In contrast to some other OpenStreetMap gamification tools, the editing functionality is not built into the MapRoulette platform itself. Instead, the user can choose which generic OpenStreetMap editor to launch amongst several popular ones with which MapRoulette is able to interact.

MapRoulette is meant to be used from home rather than in the field, so tasks should be solvable using only the information provided by available imagery. Deciding from the imagery whether there is a crosswalk in a small area (step (7) in Figure 11) is a suitable task for MapRoulette, because crosswalks usually only appear along paved streets (which are themselves usually easy to discern in aerial imagery), and have a distinctive pattern and a strong colour contrast with the environment. If a task doesn't require map editing to be completed (e.g. because our software detected a crosswalk where there is none – i.e. a false positive), or when users can't solve a task (e.g. because they cannot discern whether there is a crosswalk in the picture, or exactly where), users can indicate this on the MapRoulette platform.

3 Results

This project yielded (a) a workflow for detecting objects on orthophotos and (b) a software application implementing this workflow, as well as (c) positions of 18,036 potential crosswalks that weren't already mapped in OpenStreetMap, found by applying our approach region by region over the complete area of Switzerland.

6 For the OpenStreetMap data model, see Ramm, Topf, & Chilton (2011), pp. 51–56 or OpenStreetMap Wiki contributors (2017a).

We posted 20 challenges on MapRoulette to get these positions vetted and (if genuine) added to OpenStreetMap. We observed a weekly increase in the number of crosswalks in OpenStreetMap within Switzerland, both before and during these challenges, and their positive influence was clearly visible. During 33 weeks before the first challenge started, we had an average growth of 63.2 crosswalks per week. During the first 24 weeks of the challenges, we measured a weekly increase of 370.5 crosswalks. Within those 24 weeks, the number of identified crosswalks in Switzerland rose from 41,788 to 50,584.[7] This means that almost one in five crosswalks present in OpenStreetMap after 24 weeks of our challenges were added during that period.

We are therefore confident that we have reached our aim of helping to increase the completeness of the OpenStreetMap dataset with regard to crosswalks.

4 Discussion

It should be noted that the very high accuracy (98.5%) reported for the training applies to a validation set taken from the input data and stashed apart by TensorFlow's retrain.py script. While this validation data was not used by the script for the actual training, so the high value should not be the result of overfitting to the training set, the input data to the script could have been biased as a whole. Consequently, this result may not correspond to the real accuracy of the actual classification performed after the training in the object-detection steps.

To estimate the actual accuracy, the error rate (false positives and false negatives in relation to the total number of classified images) could be approximated by using the number of crosswalks previously in OpenStreetMap that haven't been re-found by the detection procedure as a lower bound for the false negatives. A lower bound for the number of false positives is more directly available if the assessment by MapRoulette volunteers is treated as the ground truth, which we think should be close enough to reality.

The decision to leave images of partially covered crosswalks out of the training set was made arbitrarily, without being based on actual findings. Whether including these pictures in the training improves or degrades the detection rate should be tested.

5 Conclusion and Outlook

We have demonstrated that software developers and GIS-domain experts can apply state-of-the-art computer vision technology to automate previously manual tasks in their field. The libraries and frameworks available nowadays have enabled us to do so without extensive prior knowledge of AI.

7 See http://zebrastreifen-safari.osm.ch/ for an up-to-date graph of recent development.

While (near) ready-made routines from the framework used were employed for the classification part at the core of our process, our contribution comprises the workflow and steps around that: (a) ways of efficiently and semi-automatically obtaining sufficient training data; (b) framing the object search/detection/coarse-localization problem as a problem of classification of focal data within candidate regions; (c) using domain knowledge for limiting the candidate region within the complete area of interest to decrease the classification expenditure; (d) verification and integration of the results into OpenStreetMap in accordance with the community's rules through use of an existing targeted crowd-editing platform, and (e) some (limited) automation of the individual steps as a re-usable and adaptable open source application, available at https://github.com/geometalab/OSMDeepOD.

With the flexibility and the various configuration parameters of OSMDeepOD, it is possible to search for other objects, such as streets, buildings, swimming pools, solar panels, football fields, tennis courts, agricultural areas, among many other things. Collecting a suitable data set and re-training the convolutional neural network should be all that is needed.

Other varieties of object recognition on satellite imagery include the use of oblique photographs and detecting the number of storeys of buildings, which is very useful information for the 3D visualization of maps. The same techniques could also be used to localize objects on the images and determine their shapes.

For further work to improve detection, a good starting point would be to look at the most recent designs of neural network models (Szegedy, Ioffe, & Vanhoucke, 2016). Advances in machine learning have been so rapid that better networks are created almost every week. For localization tasks, Fast Region-based Convolutional Neural Network (Fast R-CNN; see Girshick, 2015) would seem like the right place to start. The goal of Fast R-CNN is to classify multiple objects on images and to be able to draw a bounding box around them (Uijlings, van de Sande, Gevers, & Smeulders, 2013).

Images

All orthophoto imagery shown in our figures or their backgrounds is taken from

- Federal Office of Topography (SwissTopo): SWISSIMAGE - The Digital Color Orthophotomosaic of Switzerland, https://shop.swisstopo.admin.ch/en/products/images/ortho_images/SWISSIMAGE

All visualizations other than these are our own work (lineart etc.) or screenshots taken by us (mouse pointer).

References

Chen, J., & Zipf, A. (2017). DeepVGI: Deep learning with volunteered geographic information. In Proceedings of the 26th international conference on world wide web companion, Geneva, Switzerland, pp. 771–772. https://doi.org/10.1145/3041021.3054250

Chollet, F. (2016). Building powerful image classification models using very little data. The Keras Blog. Tutorial, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. CoRR, abs/1610.02357. Retrieved from http://arxiv.org/abs/1610.02357

Girshick, R. B. (2015). Fast R-CNN. CoRR, abs/1504.08083. Retrieved from http://arxiv.org/abs/1504.08083

Goodchild, M. F. (2007). Citizens as sensors: The world of volunteered geography. GeoJournal, 69(4), 211–221. https://doi.org/10.1007/s10708-007-9111-y

Google. (2016). How to retrain Inception's final layer for new categories. TensorFlow Guides. Tutorial, https://www.tensorflow.org/versions/r0.11/how_tos/image_retraining/.

Karpathy, A. (2014). What I learned from competing against a ConvNet on ImageNet. Andrej Karpathy blog. Blog, http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/.

Keller, S., Bühler, S., & Kurath, S. (2016). Erkennung von Fußgängerstreifen aus Orthophotos. AGIT Journal, 2, 162–166.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in neural information processing systems 25, pp. 1097–1105. Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Mnih, V., & Hinton, G. E. (2010). Learning to detect roads in high-resolution aerial images. In K. Daniilidis, P. Maragos, & N. Paragios (Eds.), Computer vision – ECCV 2010: 11th European conference on computer vision, Heraklion, Crete, Greece, September 5-11, 2010. Proceedings, part VI (pp. 210–223). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-15567-3_16

Mnih, V., & Hinton, G. E. (2012). Learning to label aerial images from noisy data. In J. Langford & J. Pineau (Eds.), Proceedings of the 29th international conference on machine learning (ICML-12), pp. 567–574. New York: Omnipress. Retrieved from http://icml.cc/2012/papers/318.pdf

Neis, P., Zielstra, D., & Zipf, A. (2012). The street network evolution of crowdsourced maps: OpenStreetMap in Germany 2007–2011. Future Internet, 4(1), 1–21. https://doi.org/10.3390/fi4010001

OpenStreetMap Wiki contributors. (2017a). Elements – OpenStreetMap wiki. Retrieved from http://wiki.openstreetmap.org/w/index.php?title=Elements&oldid=1479648

OpenStreetMap Wiki contributors. (2017b). Overpass API – OpenStreetMap wiki. Retrieved from http://wiki.openstreetmap.org/w/index.php?title=Overpass_API&oldid=1479536

OpenStreetMap Wiki contributors. (2017c). Tag:Highway=crossing – OpenStreetMap wiki. Retrieved from http://wiki.openstreetmap.org/w/index.php?title=Tag:highway%3Dcrossing&oldid=1437852

Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. IEEE conference on computer vision and pattern recognition (CVPR)

Ramm, F., Topf, J., & Chilton, S. (2011). OpenStreetMap: Using and enhancing the free map of the world. Cambridge, UK: UIT Cambridge

Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton project para. Cornell Aeronautical Laboratory

Russell, I. (1996). Brief history of neural networks. Neural Networks Module course notes, http://uhaweb.hartford.edu/compsci/neural-networks-history.html

Szegedy, C., Ioffe, S., & Vanhoucke, V. (2016). Inception-v4, Inception-ResNet and the impact of residual connections on learning. CoRR, abs/1602.07261. Retrieved from http://arxiv.org/abs/1602.07261

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception architecture for computer vision. CoRR, abs/1512.00567. Retrieved from http://arxiv.org/abs/1512.00567

Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., & Smeulders, A. W. M. (2013). Selective search for object recognition. International Journal of Computer Vision, 104(2), 154–171. Retrieved from https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? CoRR, abs/1411.1792. Retrieved from http://arxiv.org/abs/1411.1792

Zeiler, M. D., & Fergus, R. (2013). Visualizing and understanding convolutional networks. CoRR, abs/1311.2901. Retrieved from http://arxiv.org/abs/1311.2901