BUILDING DETECTION ON ORTHOPHOTO

Kristoffer Petersson
School of Engineering
Jönköping University
Jönköping
pekr19ct@student.ju.se

Zsombor Tóth
School of Engineering
Jönköping University
Jönköping
tozs20pn@student.ju.se

May 28, 2021

ABSTRACT

With the opportunity of machine learning applications getting more widespread, we are assessing the possibility of performing building detection from orthophotos. This is done in collaboration with our stakeholder, the municipality of Jönköping. In this project we are using the Faster R-CNN algorithm to detect buildings in these images. We evaluate our models built through the common metrics used in PASCAL VOC and COCO. An important part in training a working model is to prepare the input data in a way than enables the model to identify the underlying concept of the targeted desired objects. This is done through image filtering and combining these with the raw images. When evaluating our models, the best performing model for our purposes achieved an AP@.50IoU of 0.946. For coming projects this data could be combined with the vector based data maintained by the stakeholder to get the differences between the base map and the "real world" without too much manual labour. One could also use segmentation to actually estimate the size of buildings. This would show not only if buildings are added, but also if they are changed.

***K*eywords** Object detection · Orthophoto · Machine learning · Artificial Intelligence

1 Introduction

Over the last years the research within the area of machine learning has gained a lot of interest. One field that has grown is the field of object detection in images. An image is sent through a Deep Neural Network (DNN) and proposed bounding boxes is given as output. The model can be trained to detect several classes of objects in the same image. With this technology there are many applications to explore, detect pedestrians on the road in systems of self driving cars, counting people in a room is another application that could be used to adapt the room environment.

Our stakeholder, the municipality of Jönköping, uses orthophotos in combination with a database, called base map, which contains vector data of buildings, roads, elevation levels of the ground etc. For the stakeholder it is important to have a base map that is a correct representation of the "real world". They update this whenever there is a construction in the territory of the municipality, for example a new building is built, extended or demolished, which is done by manual labor by the engineers. Here Artificial Intelligence (AI) and machine learning can support the people working on the base map to make the task easier and less tedious. In this report we present our findings with the task to count buildings in orthophotos. This can be a first step towards more automatic checks of difference between the base map and the "real world". Our code with instructions can be found at `https://github.com/Zsombroo/jkpg-building-detection`.

2 Related Work

Much work has been done in the area of finding buildings from aerial images both with regard to detection and segmentation. To have an end-to-end application working properly, several steps are taken. As [1] show the preprocessing step can be vital in getting high performance from the model, and they introduce filtering of images before training of the model occurs. This involves edge detection and denoising filters to enhance the boundaries of buildings and remove noise, where the objective is to help the model in finding clear patterns in the image. Further on [2] reach to the problem of finding buildings of different sizes through hyperparameter tuning of anchor sizes in the algorithm when proposing regions for classification.

3 Materials and Methods

3.1 Hardware

In this project, our desktop computers were used when labelling data, and for preprocessing it and training of models a server is used on hardware specified in Table 2.

3.2 Dataset

The engineers at the municipality are using orthophotos taken at specific times of the year, and try to find the difference between them and the base map's vector data. To be able to identify buildings that are not updated in the municipality's database we have to find them first on the orthophotos, so we worked with these images to train a model that can detect residential houses.

We have received a few images from the municipality with some specific characteristics. They all had 6251x6251 pixels and covered 1 km^2 area. On these orthophotos each pixel corresponds to a 16x16cm surface on the ground. It is important to mention that the images are always taken in favorable weather, meaning that there are no clouds on the sky that would cast shadows, and the sun is around the optimal position so that the buildings also have smaller shadows than they would for example close to sunset. These conditions are always guaranteed, so the model doesn't have to be prepared for bad lighting. We have also received some other meta information alongside the orthophotos, but at this stage we are not interested in them, so we filtered them out and only used the raw pixel values of the images. The first step after receiving the orthophotos was to slice them up into a smaller, more manageable size of 640x640 pixels. This was also the default input shape our base model we used for transfer learning required. We considered these slices to be our starting stage for any further processing. Later on we are going to refer to these slices as raw data. As is shown in Figure 1, with a significant probability, buildings can fall onto the edges or corners of these slices making it more difficult to detect them, but there is an easy solution to this problem, which is producing more slices with an offset that moves these building from the edges to the middle of the image.

3.3 Labelling

Although the base map database contains vector information, that could be used to produce the labels for our images, we decided not to use them, because they contain too many details, and we didn't have enough time to create a preprocessing algorithm that filters out unnecessary information and combines others, e.g., a house was built 2004 but later was expanded in 2006. This change to the building is visible in the base map, but not identifiable on the orthophoto. Since we would only see a single complete building and not the different parts of it built earlier or later. This choice obviously had some drawbacks, but also had some benefits as well. Manual labeling of our data is a time consuming process, but it actually give us freedom over choosing the labeling method that would best support our model in training. Since this was our first project on object detection, experimenting with different methods of labeling gave us some insight on how the model behaves and responds to different label information.

First we trained a model on a lower number of image slices with only a single label that identified all buildings regardless of size, shape, type etc. This experimental model had low precision and often made the mistake of identifying other objects like cars and parking lots as the desired residential houses. We were unsure at this point if the labeling method we used was optimal and we only had to increase the number of images in the training dataset, or train our model longer. Furthermore, if we actually needed to show the model some examples to tell it that the objects it misclassified are not the buildings we are interested in. For the second experimental run we expanded our label set and added cars to it. After this run, we concluded that having more classes to identify does not increase significantly the model's performance to make the extra time needed for the labeling worth it. In the end we chose to return to only label buildings and we followed two strategies that we hoped would produce the best result.

One of them was to draw bounding boxes on houses and other comparably sized buildings like garages, that had traditional roofs trying to make the underlying concept as uniform as possible. At this point we excluded buildings with flat tops, and large-scale industrial buildings and warehouses. The other strategy we used was using three labels based on the main color of the roofs. While working on the images, we noticed that houses usually fit in one of these categories: terracotta colored roof, dark gray colored roofs and white/light gray colored roofs. Using these two labeling strategies, we significantly reduced the complexity of each class making it easier for our model to grasp the concept of a residential house.

3.4 Preprocessing

We have tried 4 different preprocessed versions of the input images. The first version is the raw images, where nothing more than normalization of the pixel values was done.

The second version is the output of the Canny edge detection, with different low and high threshold values. Unfortunately there is no single set of these thresholds that would show all the edges of the buildings without cluttering the background with "noise". Cohen et al. [1] also encountered this problem when they were working on a similar project. We consider every edge that does not belong to an artificial object like trees and bushes to be noise in our images. With some thresholds, edges became visible, but others faded into the background noise. With other parameters, houses are partially or fully disappeared. After trying out some settings, we decided to try two set where in one the lower threshold is 0 and in the other one it is 150, while the higher threshold being 255 in both. Using the prior lower parameter, houses were mostly visible as well as the forests with their busy edges, and with the higher value, the background noise significantly decreased.

To remove the heavily cluttered noisy environment from the images, we developed a method that would directly target these areas. These images were the third version. First we imported the edges that the Canny edge detection produced, then we used the dilate method of the OpenCV [3] python library for 4 iteration to merge the denser areas, applied the erode method for 10 iterations using the same 3x3 kernel of 1s in both of them, and negated the pixel values to get a mask over the problematic areas. Buildings usually did not have any significant noise between their edges, so after applying the mask they were fully visibly just as before, but the the cluttered edges inside the forests mostly disappeared.

The last version of the images we used were the combination of the above mentioned denoised and the raw images, where the edges of the buildings on the raw images were highlighted. We hoped that this combined image will help the model to learn the boundaries of the target objects more easily.

3.5 Metrics

When evaluating the result from a prediction in object detection, the value of Intersection over Union (IoU) is used. The ground truth bounding box is compared to the predicted bounding box as an overlay. Their area of intersection is set in relation to their union. A sample is considered a true positive (TP) if this threshold value is higher than 50%, otherwise false positive (FP). Precision in general is referred to the number of TP in relation to the total number of predictions for that class [4].

Mean average precision (mAP) is evaluating the ground truth and predicted bounding boxes. When the detections have been sorted by their confidence score (descending) the precision and recall is calculated by the accumulated values of TP and FP. These values are plotted as the precision-recall curve, under which the area (AUC) is the average precision. There are different common practices that is used when making these calculations. Calculations used for the PASCAL VOC dataset and the COCO dataset is two of them, in which we will use both in this project. PASCAL VOC is using the fixed threshold value of 50% to differ TP from FP, while COCO mAP is using a range of threshold values from 50% to 95% with 5% step size as shown in Equation 1 [5][6]. The PASCAL VOC evaluation is here referred to as AP@.50IoU, and the COCO evaluation as mAP.

$$mAP_{COCO} = \frac{mAP_{0.50} + mAP_{0.55} + ... + mAP_{0.95}}{10} \tag{1}$$

3.6 Faster R-CNN

Faster R-CNN is described in [7] and is an evolution of the earlier algorithms R-CNN and Fast R-CNN. An image is sent through a Convolutional Neural Network (CNN) to propose regions of interest (RoI) from this feature map. The region proposal network is evaluating if the region contains an object or not. The final part of the Faster R-CNN algorithm is the object detector that is classifying what kind of object is present in the proposed region. Compared to its predecessors Faster R-CNN is very fast to run and has a high accuracy [2].

4 Results

4.1 Algorithm

In this project we are using tensorflow 2 object detection model API [8], and have chosen the Faster R-CNN model because of its known accuracy and speed in training [2]. In the model zoo it exists in different versions from where transfer learning is possible. In the elaborative phase different Faster R-CNN algorithms were tested with regard to speed and accuracy. Our choice is the model that is trained on ResNet50.

4.2 Hyperparameters and preprocessing

To create a good performing model both hyperparameters and preprocessing steps has to be evaluated. In the first stage of the project augmentation inside the model were evaluated and found to be best performing with the default settings for our chosen algorithm. Throughout the project the number of steps in training were also evaluated and later chosen to 5000. A great deal of focus is concentrated on how the preprocessing of images is affecting model performance. The categories of preprocessing are shown in Table 1, column "Input data filter". The column "Labelling" is referring to how the labelling was conducted, either all buildings where labelled or only those who have traditional rooftops (with an angle roof). What is shown in the table are that input data that either are raw input or raw input combined with an filter overlay gives the most performing models in our setup.

4.3 Evaluation metrics

We can see from Table 1 that one model performs best with regard to mAP and another model performs the best seen to AP@.50IoU. With our task at hand, the latter is preferred, since we will detect buildings, and are not interested in how accurate the bounding boxes are as long as there is a match. Our model of choice is using raw input with an overlay of the image filtered.

Input data filter	**Channels**	**Labelling**	**mAP**	**AP@.50 IoU**
Raw	RGB	Traditional rooftops	**0.612**	0.904
Raw	RGB	All residential buildings	0.542	0.808
Canny edge filtered (0, 255)	Gray	Traditional rooftops	0.448	0.836
Canny edge filtered (0, 255)	Gray	All residential buildings	0.085	0.259
Canny edge filtered (0, 255) + denoised	Gray	Traditional rooftops	0.431	0.795
Canny edge filtered (0, 255) + denoised	Gray	All residential buildings	0.159	0.341
Raw + overlay (canny edge (0, 255) + denoised)	RGB	Traditional rooftops	0.516	0.816
Raw + overlay (canny edge (0, 255) + denoised)	RGB	All residential buildings	0.252	0.554
Canny edge filtered (150, 255)	Gray	Traditional rooftops	0.413	0.848
Canny edge filtered (150, 255)	Gray	All residential buildings	0.125	0.280
Canny edge filtered (150, 255) + denoised	Gray	Traditional rooftops	0.322	0.724
Canny edge filtered (150, 255) + denoised	Gray	All residential buildings	0.143	0.301
Raw + overlay (canny edge (150, 255) + denoised)	RGB	Traditional rooftops	0.586	**0.946**
Raw + overlay (canny edge (150, 255) + denoised)	RGB	All residential buildings	0.345	0.708

Table 1: Evaluation metrics for different models, bold face for highest metric

Component	**Specification**
CPU	AMD Ryzen 5 @3.6GHz
RAM	32GB@3.2GHz
GPU	GTX 1660 6GB

Table 2: Hardware specification used in training

4.4 Inference on novel data

Figure 1 is showing inference on a previously unseen image. The buildings that is detected are correct and has a high confidence score (100%). What is missed by our model is the building in the middle.

Buildings on the boundary of the image is sometimes detected by the model, and sometimes not. In application, this would impose a problem, since the buildings not found, probably will keep being unfound in all coming detections. This can be resolved through a second run of detection, where the image slices will be cut with an offset value, just to make the boundary buildings not be split in half as described in 3.2.

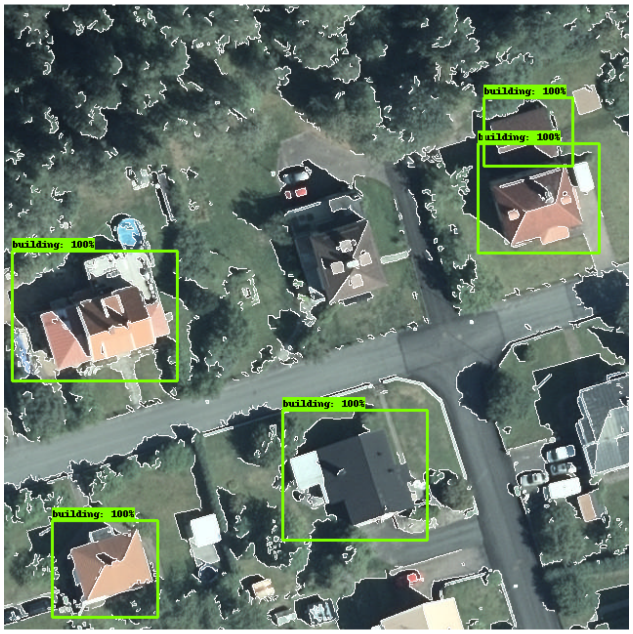

Figure 1: Inference on novel data

5 Discussion, conclusion and future

From this project we have trained a model to perform with an AP@.50IoU of 0.95 when detecting buildings in aerial images. During the project several lessons have been learned. Most specifically how the size of the data and the possible preprocessing steps can help a model to improve in finding useful patterns. The images used are all from the same type of area, outside of the most urban areas. How the model will perform running on other areas is not evaluated at this point. It has been clear that from orthophotos the environment has a great impact on performance. Large shadows or trees might hide parts of buildings, thus making it more hard to detect.

To make practical use of this project for our stakeholder we see several possible future projects, including creating a difference between the detected buildings and the base map. This could help the stakeholder in keeping the base map up-to-date and minimizing manual labour. Other projects for the future could include segmentation of the same buildings, thus being able to estimate a size of a building. That would identify not only buildings, but changes made to the same.

References

[1] Joseph Paul Cohen, Wei Ding, Caitlin Kuhlman, Aijun Chen, and Liping Di. Rapid building detection using machine learning. *Applied Intelligence*, 45(2):443–457, 2016.

[2] Karin Fritz. Instance segmentation ofbuildings in satellite images. 2020.

[3] Opencv project. *https://opencv.org/*.

[4] Evaluation metrics for object detection and segmentation: map. *https://kharshit.github.io/blog/2019/09/20/evaluation-metrics-for-object-detection-and-segmentation*.

[5] Coco - common objects in context. *https://cocodataset.org/#detection-eval*.

[6] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[8] Tensorflow. models/tf2_detection_zoo.md at master - tensorflow/models.