Stop Sign Detection in Google Street View Images Using the Haar Cascade Classifier

Richard Wen

Department of Civil Engineering, Ryerson University, Toronto, ON

June 27, 2017

Abstract

The success of Google Street View (GSV) has enabled millions of users to visualize particular locations without being present at the locations. This is possible due to the large scale data collection of panorama imagery using vehicular and portable camera systems. The imagery collected by GSV provides a unique source of visual data that can be used to extract valuable and unique geospatial information. The images provide a great opportunity for object detection methods to extract objects of interest from images with geolocation metadata with algorithms such as the popular Haar Cascade Classifier (HCC) proposed by Viola and Jones. This paper used the HCC algorithm to create a stop sign detection model for GSV images as a small experiment in extracting street level geospatial data. The experiment revealed that the HCC algorithm was strongly affected by grayscale image intensities of images, which may change greatly with different distortions, viewing angles, and image qualities. However, the rapid efficiency of the algorithm enables the potential for it to be combined with multiple HCC models to better improve performance. This experiment demonstrated that it was possible to extract unique and valuable geospatial data such as street level stop signs from images with geolocation metadata, which are expensive or impractical to obtain with other standard methods such as Light Detection and Ranging, remote sensing, and manual in-situ surveying.

1 Introduction

Google Street View (GSV) has enabled millions of users to view street level imagery without having to visit locations in person. GSV has covered hundreds of cities in 20 countries over four continents by mounting camera systems on to vehicles that are able to capture panoramic images of an immediate surrounding [Anguelov et al.2010]. These images provide a valuable source of geospatial data that can be difficult or expensive to obtain using aerial methods such as remote sensing, drones, or Light Detection and Ranging (LiDAR) [Watts et al.2012].

Objects such as civilians, poles, and traffic signs viewed at the street level may be unidentifiable from an aerial view, but easily identified by users of GSV through visual inspection of street level imagery at particular locations. However, this task of manual inspection is time-consuming, tedious, and impractical for the massive quantities of imagery that are available. Computer vision techniques, namely object detection, can be used to automate the inspection of GSV image for the extraction of geo-located objects on a wide scale without requiring large amounts of manual labour.

This paper detailed an experiment with a popular object detection method known as the Haar Cascade Classifier (HCC) on small subset of GSV images. The goal of the experiment was to extract the approximate location of stop signs in Toronto, Ontario for randomly selected GSV images. The experiment was done by first developing understanding of the HCC through a background review. A package was then developed to extract GSV images using an Application Programming Interface (API). A performance evaluation was done by manually labeling the small subset of extracted GSV images to observe the detection results. Finally, a discussion of challenges, proposed improvements, and future directions were provided to address the implications of the experiment.

The proceeding sections are organized as follows:

- Section 2 details the HCC algorithm
- Section 3 details the methods used for extracting GSV images and performance evaluation
- Section 4 presents the results of the performance evaluation
- Section 5 discusses the experiment's challenges, proposed improvements, and future directions
- Section 6 provides concluding remarks and implications

2 Background

This section provides background on the HCC algorithm. The HCC was based on the Viola and Jones algorithm, which provided an efficient object detection method for images using a machine learning approach [Viola and Jones2001]. The algorithm involves three main steps:

1. **Feature creation**: abstraction of pixels into simple rectangular features
2. **Learning classifiers**: learn function for feature selection based on lowest misclassification
3. **Cascading classifiers**: negative sub-window rejection for efficient computation

2.1 Feature Creation

The feature creation process involves creating Haar-like features from pixels in an image. Haar-like features are simple rectangular features computed from individual image pixels. Integral images are used to efficiently create Haar-like features from images by computing rectangular sums in constant time. This section provides details on Haar-like features and integral images for object detection in images.

2.1.1 Haar-like Features

Haar-like features form abstract representations of individual pixels from a grayscale image. For example, a rectangular region of an image can contain an abstract representation such as a line or an edge, which is defined by the gradient changes between light and dark pixels. The dimensions of each rectangular region are defined by the user, where 24 by 24 regions are suggested as optimal through empirical observation [Viola and Jones2001]. In addition to the rectangular regions, Haar-like features consist of additional rectangular sub-regions with aggregated pixel values to capture a simplified structure. Each sub-region containing individual pixels can be either white or black, where the sum of the white regions subtracted from the sum of the black regions produce a single value to be used for object detection. A sliding window, representing a rectangular region, is used to capture these sub-regions throughout the image to compute single values representing features for each image.

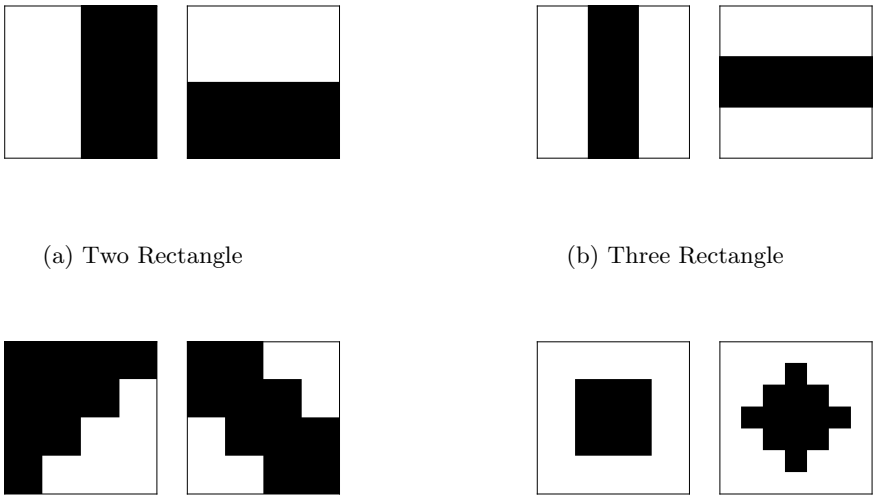
Several haar-like features have been proposed to be effective for object detecton. Simple two-rectangle features [Papageorgiou et al.1998], three rectangle features in [Viola and Jones2001], rotated features, and center-surround features [Lienhart and Maydt2002] are seen in Figure 1. These features can be used to abstract details in images to represent identifying features of objects such as noses, where a three-rectangle feature can be used to measure the grayscale contrast of the lighter region on a nose, and darker regions to both sides of a nose.

2.1.2 Integral Images

Integral images (also known as summed area tables) are pre-computed to efficiently calculate sums in rectangular regions of an image [Crow1984]. An integral image is created by inclusively summing each pixel in the original image by the pixels that are to the left and above as seen in Equation 1.

$$ii(x, y) = \sum_{x_i \leq x, y_i \leq y} i(x_i, y_i) \tag{1}$$

where $ii(x, y)$ is the integral image at row x and column y, and $i(x_i, y_i)$ is the original image pixels that are to the left and above $ii(x, y)$. The sum of rectangular regions can be then be efficiently calculated using the pixel values of the pre-computed integral image.



(a) Two Rectangle

(b) Three Rectangle

(c) Rotated

(d) Center-surround

Figure 1: Haar-like features

A rectangular region is defined by a pair of top-left and bottom-right corner coordinates seen in Equation 2:

$$(x_1, y_1), (x_2, y_2) \tag{2}$$

where (x_1, y_1) are the top-left coordinates of the rectangular region, and (x_2, y_2) are the bottom-right coordinates of the rectangular region.

Four integral image pixel values relative to the top-left and bottom-right coordinates of the rectangular region are defined by Equations 3 to 6:

$$A = ii(x_1 - 1, y_1 - 1) \begin{cases} 0 : \text{if} \quad x_1 - 1 \notin \{x_{min} \dots x_{max}\} \\ 0 : \text{if} \quad y_1 - 1 \notin \{y_{min} \dots y_{max}\} \end{cases} \tag{3}$$

$$B = ii(x_2, y_1 - 1) \begin{cases} 0 : \text{if} \quad x_2 \notin \{x_{min} \dots x_{max}\} \\ 0 : \text{if} \quad y_1 - 1 \notin \{y_{min} \dots y_{max}\} \end{cases} \tag{4}$$

$$C = ii(x_1 - 1, y_2) \begin{cases} 0 : \text{if} \quad x_1 - 1 \notin \{x_{min} \dots x_{max}\} \\ 0 : \text{if} \quad y_2 \notin \{y_{min} \dots y_{max}\} \end{cases} \tag{5}$$

$$D = ii(x_2, y_2) \begin{cases} 0 : \text{if} \quad x_2 \notin \{x_{min} \dots x_{max}\} \\ 0 : \text{if} \quad y_2 \notin \{y_{min} \dots y_{max}\} \end{cases} \tag{6}$$

where if the coordinates x_1, y_1, x_2, y_2 are outside the $x_{min}, y_{min}, x_{max}, y_{max}$ dimensions of the integral image ii, then the value will be 0, otherwise the respective pixel values from ii of A, B, C, D are assigned.

The sum of the pixel values inside the original image is given by Equation 7:

$$S_i(x_1, y_1, x_2, y_2) = D - B - C + A \tag{7}$$

where $S_i(x_1, y_1, x_2, y_2)$ is the sum of all pixel values of the original image i inside the rectangular region, which is defined by the top-left coordinates x_1, y_1 and bottom-right coordinates x_2, y_2.

The brute force method for the sum of all pixels inside the rectangular region requires the access of n pixels inside the rectangular region ($O(n)$), while the integral image enables the sum to be computed by at most 4 pixel accesses in the integral image at constant time ($O(1)$). The integral images can be used to process Haar-like features efficiently as two-rectangle features require at most 8 pixel accesses, and three-rectangle features require at most 12 accesses for features of any dimension. A detailed example of calculating integral images is provided in Appendix A.

2.2 Learning Classifiers

A machine learning algorithm known as Adaboost was used to learn classifiers for feature selection. Machine learning algorithms seek to optimize performance measures by learning from examples, known as training data, to generalize on new examples not seen in the learning process [Domingos2012]. Classifiers are models generated by machine learning algorithms to determine whether the input data belongs to one or more known categories (classes) [Kotsiantis et al.2006]. Adaboost combines several weak classifiers, using a process called boosting, to create a strong classifier [Freund and Schapire1997, Schapire1999]. Weak classifiers are algorithms that perform slightly better than random guessing (for example, classifying a data sample correctly more than 50% of the time). These weak classifiers are combined to produce a strong classifier, which performs better than the individual weak classifiers, using a process called *boosting*. Boosting refers to iteritively learning weak classifiers using a vector of weights, where the weights are re-calculated based on performances of previous weak classifiers.

A set of example images, their feature values, and their label (positive or negative) are given by Equation 8:

$$(x_i, f_{i,j}, y_i) \text{ for } i \ldots n_i \text{ and } j \ldots n_j, \text{ such that } y_i \in \{0, 1\} \tag{8}$$

where x_i is example image i, $f_{i,j}$ is feature j for image i, y_i are the labels for example image i (0 being negative and 1 being positive), n_i is the number of example images, and n_j is the number of features for xi.

A set of weights are initialized for each of the example images seen in Equation 9:

$$w_{1,i} = \begin{cases} \frac{1}{2m} : \text{if } y_i = 0 \\ \frac{1}{2l} : \text{if } y_i = 1 \end{cases} \tag{9}$$

where m is the number of negative example images, and l is the number of positive images.

The HCC is based on a modified version of Adaboost, in which the following equations are applied for t number of iterations until a specified criteria is met (such as reaching the desired classification accuracy or optimizing the misclassification error).

Create a probability distribution of normalized weights $w_{t,i}$, for each t, such that $w_{t,i}$ is in the range of 0 to 1 and sums to 1 as seen in Equation 10:

$$w_{t,i} = \frac{w_{t,i}}{\sum_{k=1}^{n_i} w_{t,k}} \tag{10}$$

where n_i is the number of example images, and $\sum_{k=1}^{n_i} w_{t,k}$ is the sum of the weights for all images.

Train a weak classifier $h_j(x)$ for each feature j using example images xi and their labels yi. Viola and Jones uses a decision stump as the weak classifier defined in Equation 11:

$$h(x, f, p, \theta) = \begin{cases} 1 : \text{if } p\ f(x) p\theta \\ 0 : \text{otherwise} \end{cases} \tag{11}$$

where $h(x)$ is the weak classifier for a feature given example image x defined by a sub-window (such as a 24 by 24 rectangular sub-region), p is the parity representing the direction of the inequality sign (positive or negative), $f(x)$ is the single value for the rectangular feature of x, and θ is a threshold. The threshold θ is learned by training a decision stump to first order the feature values f from lowest to highest for each image x, followed by selecting the feature value that optimizes h for the lowest error measure defined by Equation 12:

$$e = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+)) \tag{12}$$

where S^+ is the sum of positive weights below the current feature value f, S^- is the sum of negative weights below the current feature value f, T^+ is the total sum of all positive example weights, and T^- is the total sum of all negative example weights. This error can be interpreted as a measure of the minimum error of labeling examples below a feature value and above a feature value given the weights [Viola and Jones2004]. The feature value with the minimum error e is then selected as the threshold θ_j.

Choose the classifier h_j with the lowest error ϵ_j defined by Equation 13:

$$\epsilon_j = \sum_{i}^{n} w_i \left| h_j(x_i, f_j, p, \theta_j) - y_i \right| \tag{13}$$

where $h_j(x_i, f_j, p, \theta_j)$ is the predicted label (0 negative or 1 for positive), and y_i is the true label.

The classifier with the lowest error h_t minimizes ϵ_j by optimizing θ_j and p shown in Equation 14:

$$\min_{\theta_j \in R,\ p \in \{+1,-1\}} \epsilon_j(x_i, f_j, p, \theta_j) \tag{14}$$

The weights are updated after selecting the classifier h_j using the Equation 15:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_j} \tag{15}$$

where β_t is $\frac{\epsilon_t}{1-\epsilon_t}$.

After t iterations, a final strong classifier $C(x)$ can then be formed using Equation 16:

$$C(x) = \begin{cases} 1 : \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 : \text{otherwise} \end{cases} \tag{16}$$

where $\alpha_t = \log \frac{1}{B_t}$, and h_t are the weak classifiers learned from t iterations. This strong classifier is the combination of the weak classifiers such that the predictions are the weighted majority votes.

2.3 Cascading Classifiers

A cascade of classifiers was used in the Viola and Jones algorithm to reduce the number of rectangular regions (sub-windows) that need further processing, and to improve the detection rate [Viola and Jones2004]. A cascade of classifiers is constructed using the Adaboost algorithm. A cascade consists of several stages that increase the detection rate and false positive rate with a higher number of stages. In order for a rectangular region to be further processed, it has to pass each stage of the cascade. If the rectangular region is classified as negative at any stage, it is rejected immediately and further processing is stopped. Each stage of a cascade classifier uses the previous stage's examples that were successfully passed as positive, which makes subsequent stages inherit increasingly difficult examples that form classifiers with with rules that are more strict. The number of stages depend on the desired detection or false positive rate set manually by the user.

The detection rate of a cascade D is given by Equation 17:

$$D = \prod_{t=1}^{K} d_i \tag{17}$$

where K is the number of classifiers and d_i is the detection rate of classifier i on examples that are accepted by it.

The false positive rate of a cascade F is given by Equation 18:

$$F = \prod_{t=1}^{K} f_i \tag{18}$$

where K is the number of classifiers and f_i is the false positive rate of classifier i on examples that are accepted by it.

3 Methods

This section describes the procedures taken to perform the experiment of a stop sign detection HCC for GSV images. A set of positive and negative images were required to evaluate a Haar Cascade Classifier for stop signs. Positive images are example images that contain the object to be detected, and negative images are example images that do not contain the object to be detected. The positive and negative images were extracted by locating stop signs from OpenStreetMap, and querying Google Street View using those locations [Haklay and Weber2008]. The data was then manually inspected to be labelled as positive or negative, and was used for performance evaluation. The procedure involved four major steps:

1. **Locating stop signs**: extraction of stop sign locations using OpenStreetMap
2. **Extracting Google Street View Images**: extraction of GSV images using stop sign locations
3. **Stop Sign Detection Model**: utilize a pre-trained HCC model for performance evaluation of the GSV images
4. **Performance Evaluation**: evaluate correct detections and false positives using the GSV images

3.1 Locating Stop Signs

```
[out:json];
node[highway=stop]
  (43.458297,-79.639219,43.8554579,-79.002481);
out;
```

Figure 2: Overpass API query for stop sign locations

The location of stop signs were extracted using the Overpass API through a Uniform Resource Location (URL) link with a data query string [OpenStreetMap2017,

Lee et al.1994]. The Overpass API provided OpenStreetMap data by enabling users to send requests made of queries to filter desired data subsets. Details on the Overpass API URL structure is available in Appendix B.

The stop sign locations were limited to Toronto, Ontario, by specifying a bounding box defined by 43.458297 and -79.639219 (latitude and longitude of the bottom left corner), and 43.8554579 and -79.002481 (latitude and longitude of the top right corner). Stop signs were tagged as *highway=stop* on OpenStreetMap, and locations were known as *nodes*. The Overpass API query is shown in Figure 2, which was used to produce the final URL to extract the stop sign locations [OpenStreetMap2017]. For the purpose of this experiment, 99 locations were randomly selected to be used as a small subset of data.

3.2 Extracting Google Street View Images

```
size=640x640
location="location_of_stop_sign1..n"
fov=120
key="developer_key"
```

Figure 3: Google Street View API base query

GSV images were collected using the stop sign locations in Section 3.1. A Python package named *google_streetview* was developed to create data query strings for the GSV API and automatically download GSV image requests [Wen2017]. The available GSV API parameters to be set by the user are shown in Table 1. Figure 3 shows the parameters used to extract each GSV image using the stop sign locations. Each location parameter was replaced by a location from the randomly selected stop sign locations. Details on the GSV API URL structure is available in Appendix C.

Parameter	Description
location or pano	coordinates (latitude, longitude) or address
key	the google developer key
heading	compass rotation of camera from 0 to 360 degrees (the default directs it to face the location)
fov	horizontal field of view from 0 to 120 (default is 90)
pitch	vertical angle of camera from -90 to 90 with 0 being in the middle (the default is 0)
size	image size up to 640 by 640

Table 1: Google Street View Paramters

3.3 Stop Sign Detection Model

Figure 4: Example of positive training images

Figure 5: Example of negative training images

A HCC model was trained on 82 images of stop signs, and 1115 images of surrounding street backgrounds[1]. The 82 images of stop signs are known as the positive training data, and the 1115 images of surrounding street backgrounds were known as the negative training data. Additional samples were also created from the training data by iteritively applying distortions to account for stop signs that were viewed from different angles. Figure 4 shows the positive training images, and Figure 5 shows the negative training images.

3.4 Performance Evaluation

The 99 GSV images extracted from the GSV API (described in Section 3.1) were manually labeled to be positive (has a stop sign), and negative (does not have a stop sign). The labels did not count stop signs that were captured from the back (these were labeled negative). The positive and negative images were used to evaluate the number of correctly classified images (positives) and the number of incorrectly classified images (negatives). The negatives can also be interpreted as the false positives or *false alarm* classifications. The GSV images were evaluated based on the detection and false positive ratios.

The detection ratio D_r is defined in Equation 19:

$$D_r = N^+ / N_{positives} \tag{19}$$

where N^+ is the number of images predicted to be positive, and $N_{positives}$ is the number of positive test images. Higher detection ratio values are better.

The false positive ratio F_r is defined in Equation 20:

$$F_r = N^+ / N_{negatives} \tag{20}$$

where N^+ is the number of images predicted to be positive, and $N_{negatives}$ is the number of negative test images. Lower false positive ratio values are better.

[1]The HCC file and training images were made available thanks to user *markgaynor* on Github under the MIT license

4 Results

Figure 6: Map of stop sign locations in Toronto, Ontario, from OpenStreetMap

Figure 7: Map of randomly selected testing stop signs in Toronto, Ontario, from OpenStreetMap

A total of 10401 stop sign locations were extracted from OpenStreetMap using the Overpass API on June 26, 2017. These locations are shown in Figure 6. A total of 99 locations were randomly selected to be used for GSV image extraction and performance evaluation. These randomly selected locations are shown in Figure 7. Using the randomly selected locations, 99 GSV images were extracted using the GSV API. Out of the 99 GSV images, 19 were labeled positive (having a stop sign), and 80 were labeled negative (not having a stop sign). Figure 8 shows a few examples of the positive images, and Figure 9 shows a few examples of the negative images.

Figure 8: Example of positive testing images

Figure 9: Example of negative testing images

The HCC stop sign model was able to only detect 6 of the positive images, and had a misclassification in 1 of the negative images. The detection ratio value was approximately 0.316, and the false positive ratio value was 0.0125. The detected stop signs are shown in Figure 10 and the misclassified stop sign is shown in Figure 11

Figure 10: Detected stop signs in the positive images

Figure 11: Misclassified stop sign in the negative images

5 Discussion

The HCC stop sign model performed well for negative images (false positive ratio value of 0.0125), but had difficulty detecting stop signs in positive images (detection ratio value of 0.316). The HCC algorithm did not account for color, and extreme distortions of the stop signs in the GSV images, as the Haar-like features were calculated on only grayscale images according to the Viola and Jones algorithm. Some stop signs in the positive images were also further away, and may cause reduce image quality and different feature values being calculated. However, the rapid classification times of the HCC stop sign model provides a potential for object detection in real-time and big data. The massive quantities of real-time and big data make it often impractical to train and classify highly accurate algorithms images on demand due to the long training times. The HCC algorithm can train a classifier that can be used to classify images in real-time, and is appropriate when there are massive quantities of data due to the optimized efficiencies of the integral images and cascaded classifiers. In general, the HCC algorithm relies on the patterns of the varying grayscale intensities of images, but if these images are do not have enough pattern variance, then the resulting classifier is expected to perform poorly. Thus, the grayscale intensities are affected by distortions, angle of view, and image quality of the object of interest. For example, when viewing the stop signs from behind, the grayscale intensities differ greatly from the front of the stop sign, and therefore is likely to not be detected. A simple solution for the issue of viewing angles is to train another classifier to detect particular angles in which the object differs greatly, and classify the detection as the same object. This may also be applied to different distortions, and image qualities, but may result in higher false positives. The following points were identified as important to using the HCC algorithm for stop sign detection:

1. **Dynamic appearance**: the object's appearance changes greatly depending on distortions, viewing angles, and image quality, which may lead to the object being undetectable

2. **Rapid detection**: the HCC can detect objects in real-time, which may

be practical for filtering large datasets for objects of interest, before performing manual evaluation

3. **Ensembling**: multiple HCCs can be combined together easily to provide more accurate object detection when dealing with dynamic object appearances due to the fast classification speed

6 Conclusion

This paper detailed a small experiment on stop detection in GSV images using the HCC algorithm. An extensive review of background literature and algorithm implementation was done to improve understanding of the HCC based on the Viola and Jones algorithm. The experiment began with the extraction of stop sign locations to further collect GSV images for testing the performance of a pre-trained HCC stop sign model. A random selection of the collected GSV images was done to reduce the problem set to a more managable size for manual labelling. The performance evaluation measured the HCC stop sign model based on its ability to detect stop signs in GSV images, and to reject GSV images when stop signs did not exist in the image. The results showed that the HCC stop sign model had good performance in rejecting GSV images without stop signs, but poor performance when stop signs existed in GSV images. Although the detection performance was poor for this particular experiment, the advantage of the HCC algorithm lies in its ability to rapidly classify large quantities of images easily. This speed enables it to potentially be able to improve its performance by ensembling multiple HCCs to account for dynamic object appearance issues such as distortions, viewing angles, and image quality. With higher detection rates, there is potential for object detection algorithms to extract valuable and unique geospatial data from images with location metadata such as street level features, civilians, and human activity.

Acknowledgements

I would like to thank Dr. Wai Yeung Yan for providing me the freedom to explore the experiment documented in this paper as an alternative to the term project for the course. I would also like to thank Github user *markgaynor* for open sourcing the Haar Cascade Classifier model and training data used in this paper's experiment under the MIT license.

References

[Anguelov et al.2010] Anguelov, D., Dulong, C., Filip, D., Frueh, C., Lafon, S., Lyon, R., Ogale, A., Vincent, L., and Weaver, J. (2010). Google street view: Capturing the world at street level. *Computer*, 43(6):32–38.

[Crow1984] Crow, F. C. (1984). Summed-area tables for texture mapping. *ACM SIGGRAPH computer graphics*, 18(3):207–212.

[Domingos2012] Domingos, P. (2012). A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87.

[Freund and Schapire1997] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139.

[Haklay and Weber2008] Haklay, M. M. and Weber, P. (2008). Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18.

[Kotsiantis et al.2006] Kotsiantis, S. B., Zaharakis, I. D., and Pintelas, P. E. (2006). Machine learning: a review of classification and combining techniques. *Artif. Intell. Rev.*, 26(3):159–190.

[Lee et al.1994] Lee, B. T., Masinter, L., and Mccahill, M. (1994). RFC 1738: Uniform resource locator (URL). http://www.ietf.org/rfc/rfc1738.txt.

[Lienhart and Maydt2002] Lienhart, R. and Maydt, J. (2002). An extended set of haar-like features for rapid object detection. In *Proceedings. International Conference on Image Processing*, volume 1, pages I–900–I–903 vol.1.

[OpenStreetMap2017] OpenStreetMap (2017). Stop sign locations retrieved from http://overpass-api.de/api/interpreter. https://www.openstreetmap.org .

[Papageorgiou et al.1998] Papageorgiou, C. P., Oren, M., and Poggio, T. (1998). A general framework for object detection. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 555–, Washington, DC, USA. IEEE Computer Society.

[Schapire1999] Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 1401–1406, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Viola and Jones2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE.

[Viola and Jones2004] Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154.

[Watts et al.2012] Watts, A. C., Ambrosia, V. G., and Hinkley, E. A. (2012). Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, 4(6):1671–1692.

[Wen2017] Wen, R. (2017). A command line tool and module for google street view image api. https://pypi.python.org/pypi/google-streetview .

A Integral Image Example

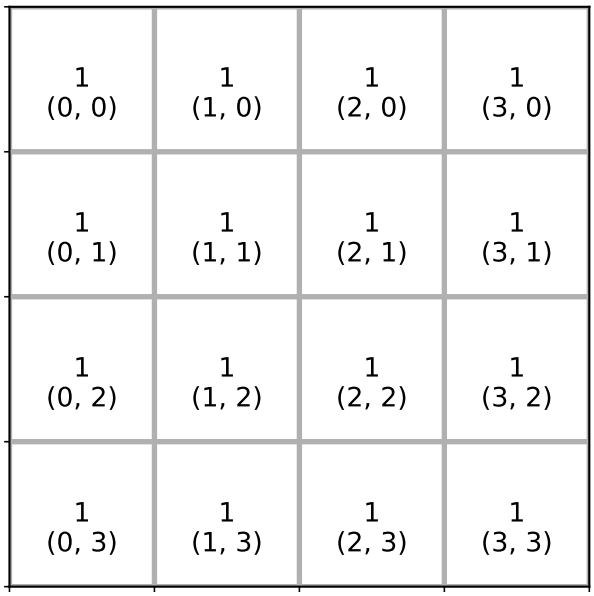


Figure 12: Example of an image i

Given an original image ii with 4 rows by 4 columns where each pixel has a value of 1 and coordinates are defined by $(x, y)(x, y)$ (Figure 12).

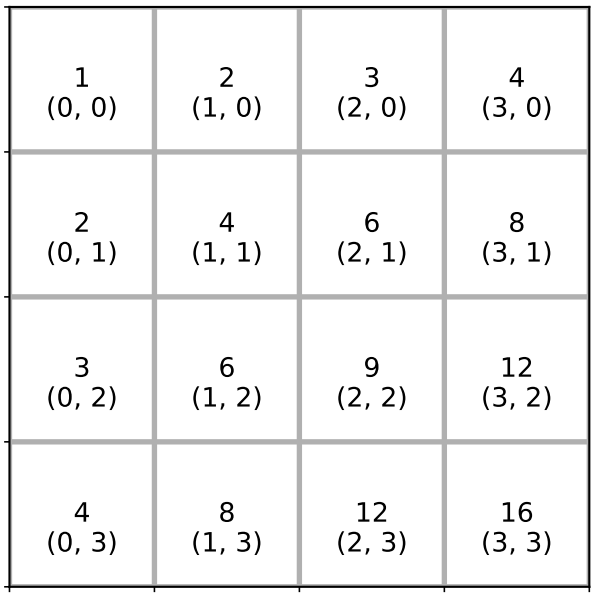


Figure 13: Example of an integral image ii

The integral image ii for the first two rows and columns can be computed as follows (Figure 13):

- Row 1, Column 1: $ii(1,1) = i(1,1) ii(1,1) = i(1,1)$
- Row 1, Column 2: $ii(1,2) = i(1,1) + i(1,2) = 1 + 1 = 2$
- Row 2, Column 1: $ii(2,1) = i(1,1) + i(2,1) = 1 + 1 = 2$
- Row 2, Column 2: $ii(2,2) = i(1,1) + i(1,2) + i(1,3) + i(1,4) = 1 + 1 + 1 + 1 = 4$

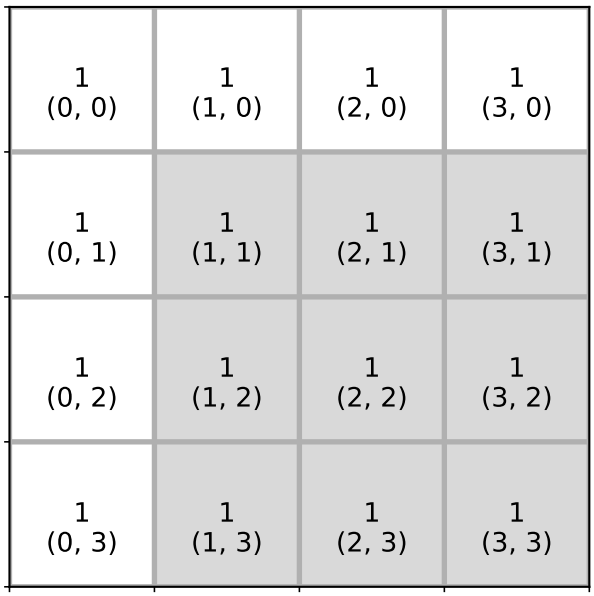


Figure 14: Example of a 3 by 3 rectangular region

A 3 by 3 rectangular region in gray for the original image, which requires 9 array accesses to calculate the sum of 9, is given (Figure 14):

- Bottom-left corner of region: $x_1, y_1 = 1, 1$
- Top-left corner of region: $x_2, y_2 = 3, 3$

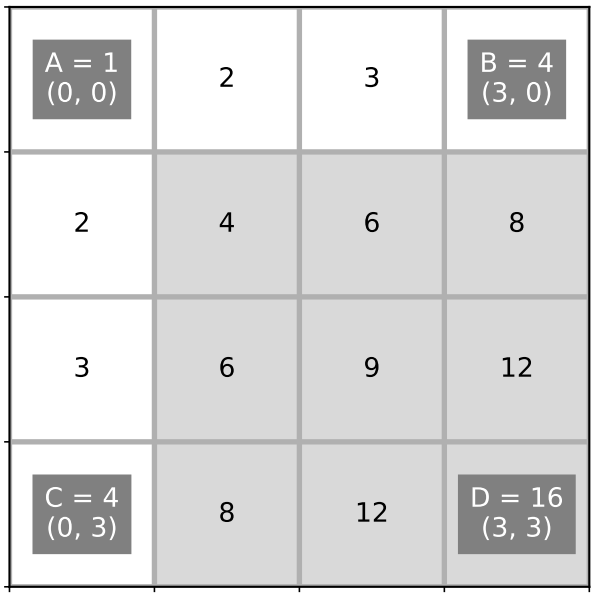


Figure 15: Example of rectangular sum using an integral image

The integral image can be used to calculate the sum of 9 with 4 array accesses instead (Figure 15):

- Recall that $x_1, y_1 = 1, 1$ and $x_2, y_2 = 3, 3$
- $A = ii(x_1 - 1, y_1 - 1) = ii(1 - 1, 1 - 1) = ii(0, 0) = 1$
- $B = ii(x_2, y_1 - 1) = ii(3, 1 - 1) = ii(3, 0) = 4$
- $C = ii(x_1 - 1, y_2) = ii(1 - 1, 3) = ii(0, 3) = 4$
- $D = ii(x_2, y_2) = ii(3, 3) = 16$
- $Si(1, 1, 3, 3) = D - B - C + A = 16 - 4 - 4 + 1 = 9$

B Overpass API URL Structure

An Overpass API request consisted of a base URL:

`http://overpass-api.de/api/interpreter`

and a data query string *ql* after this base URL:

`http://overpass-api.de/api/interpreter?data=ql`

C Google Street View URL API Structure

A GSV API request consisted of a base URL:

`https://maps.googleapis.com/maps/api/streetview`

and a data query string *params* after this base URL:

`https://maps.googleapis.com/maps/api/streetview?params`

The GSV API metadata was checked to see if a requested image exists before downloading using a seperate base URL:

`https://maps.googleapis.com/maps/api/streetview/metadata`