CBHE: Corner-based Building Height Estimation for Complex Street Scene Images

Yunxiang Zhao  
The University of Melbourne & NUDT  
yunxiangz@student.unimelb.edu.au

Jianzhong Qi  
The University of Melbourne  
jianzhong.qi@unimelb.edu.au

Rui Zhang∗  
The University of Melbourne  
rui.zhang@unimelb.edu.au

ABSTRACT

Building height estimation is important in many applications such as 3D city reconstruction, urban planning, and navigation. Recently, a new building height estimation method using street scene images and 2D maps was proposed. This method is more scalable than traditional methods that use high-resolution optical data, LiDAR data, or RADAR data which are expensive to obtain. The method needs to detect building rooflines and then compute building height via the pinhole camera model. We observe that this method has limitations in handling complex street scene images in which buildings overlap with each other and the rooflines are difficult to locate. We propose CBHE, a building height estimation algorithm considering both building corners and rooflines. CBHE first obtains building corner and roofline candidates in street scene images based on building footprints from 2D maps and the camera parameters. Then, we use a deep neural network named BuildingNet to classify and filter corner and roofline candidates. Based on the valid corners and rooflines from BuildingNet, CBHE computes building height via the pinhole camera model. Experimental results show that the proposed BuildingNet yields a higher accuracy on building corner and roofline candidate filtering compared with the state-of-the-art open set classifiers. Meanwhile, CBHE outperforms the baseline algorithm by over 10% in building height estimation accuracy.

CCS CONCEPTS

• Information systems → Location based services; • Computing methodologies → Neural networks; Camera calibration; Image representations.

KEYWORDS

Building Height Estimation; Camera Location Calibration; Open Set Classification

1 INTRODUCTION

Building height plays an essential role in many applications, such as 3D city reconstruction [3, 30], urban planning [29], navigation [14, 32], and geographic knowledge bases [50]. For example, in navigation, knowing the height of buildings helps identify those standing out in a city block, which can then be used to facilitate navigation by generating instructions such as "turn left before an 18 meters high (five-story) building".

Previous studies for building height estimation are mainly based on high-resolution optical data [18, 48], synthetic aperture radar (SAR) images [7, 41], and Light Detection and Ranging (LiDAR) data [33, 44]. Such data, however, are expensive to obtain and hence the above approach is difficult to apply at a large scale, e.g., to all the buildings on earth. Moreover, such data is usually proprietary and not available to the public and research community. Recently, street scene images (or together with 2D maps) have been used for building height estimation [11, 46], which can be easily obtained at large scale (e.g., via open source mapping applications such as Google Street View [2] and OpenStreetMap [16]). Estimating building height via street scene images relies on accurate detection of building rooflines from the images, which then enables building height computation using camera projection. However, existing methods for roofline detection check the roofline segments only, which may be confused with overlapping buildings. As shown in Fig. 1, the roofline of building B may be detected as the roofline of building A because the rooflines of different buildings may be in parallel with each other, and the buildings have similar colors and positions in the street scene images.

Figure 1: Building A's rooflines are hard to detect due to the overlapping with building B, while building A's corners can help figure out the true rooflines. Red arrows: building rooflines; Blue arrows: building corners.
street scene image to detect building corners in the image. Corners
of different buildings do not share the same coordinates, and it is
easier to associate them with different buildings, as shown in Fig. 1.

CBHE works as follows. It starts with an upper-bound estimation
of the height of a building, which is computed as the maximum
height that can be captured by the camera. It then tries to locate a
line (i.e., a roofline candidate) at this height and repeats this process
by iteratively reducing the height estimation. Following a similar
procedure, CBHE also locates a set of building corner candidates.
Next, CBHE filters the roofline candidates with the help of the
corner candidates (i.e., a roofline candidate needs to be connected
to a corner of the same building to be a true roofline). When the
true roofline is identified, CBHE uses the pinhole camera model to
compute the building height.

In the process above, when locating the roofline and corner can-
didates, we fetch two sets of image segments that may contain
building rooflines or corners, respectively. To filter each set of ob-
jects and identify the true roofline and corner images, we propose
a deep neural network model named BuildingNet. The key idea
of BuildingNet is as follows. Building corner has a limited number
of patterns (e.g., “”, “”, “”, “”), while non-corner images may
have any pattern. The same applies to the building rooflines. Thus,
we model building corner (roofline) identification as an open set
image classification problem, where each corner (roofline) pattern
forms a class, while non-corner (non-roofline) images should be
differentiated from them when building the classifier. To do so,
BuildingNet learns embeddings of the input images, which mini-
mize the intra-class distance and maximize the inter-class distance,
and then differentiates different classes using a Support Vector Clas-
sification (SVC) model on the learned embeddings. When a new
image comes, the trained SVC model will tell whether it falls into
any corner (roofline) classes. If the image does not fall into any
corner (roofline) classes, it is a non-corner image and can be safely
discarded.

When estimating building height via the pinhole camera model,
the result highly relies on the accuracy of the camera location due to
GPS errors. Therefore, CBHE calibrates the camera location before
roofline detection. To calibrate the camera location, CBHE detects
candidate regions of all building corners in street scene images by
matching buildings in street scene images with their footprints in
2D maps based on the imprecise camera location from GPS. Then, it
uses BuildingNet to classify the corner candidates and remove those
images classified as non-corner. From the remaining corner after
the classification through BuildingNet, CBHE selects two corners
with the highest score (detailed in Section 4.2) to calibrate the
camera location via the pinhole camera model. We summarize our
contributions as follows:

- We propose a corner-based building height estimation algorithm
  named CBHE, which uses an entropy-based algorithm to select
  the roofline among all candidates from BuildingNet. The entropy-
based algorithm considers both building corner and roofline fea-
tures and yields higher robustness for the overlapping problem in
complex street scene images. Experiments on real-world datasets
show that CBHE outperforms the baseline method by over 10%
regarding the estimation error within two meters.

- We propose a camera location calibration method with an analyt-
cal solution when given the locations of two building corners in
a 2D map, which means highly accurate result can be guaranteed
with the valid building corners from BuildingNet.

We organize the rest of this paper as follows. We review related
work in Section 2 and give an overview of CBHE in Section 3. The
BuildingNet and entropy-based ranking algorithm are presented in
Section 4, and the building height estimation method is detailed in
Section 5. We report experimental results in Section 6 and conclude
the paper in Section 7.

2 RELATED WORK

In this section, we review studies on camera location calibration and
building height estimation. We also detail our baseline method [46].

2.1 Camera Location calibration

Camera location calibration aims to refine the camera location of
the taken images, given rough camera position information from
GPS devices or image localization [1, 25].

Existing work uses 2.5D maps (2D maps with height information)
to calibrate camera locations. Arth et al. [4] present a mobile device
localization method that calibrates the camera location by matching
building facades in street scene images with their footprints in 2.5D
maps. Armagan et al. [3] train a convolutional neural network (CNN)
to predict the camera location based on a semantic segmentation
of the building facades in input images. Their method iteratively
applies CNN to compute the camera’s position and orientation
until it converges to a location that yields the best match between
building facades and 2.5D maps. Camera location calibration using
2.5D maps can produce good results. The hurdle is the requirement
of building height information for generating 2.5D maps, which
may not be available for every building. Chu et al. [9] extract the
position features of building corner lines (the vertical line of a corner)
and then find the camera location and orientation by matching the
extracted position features with building footprints in 2D maps.
However, their method cannot handle buildings overlapping with
each other or having non-uniform patterns on their facades.

2.2 Building Height Estimation

Building height estimation has been studied using geographical
data such as high-resolution images, synthetic aperture radar (SAR)
images, and Light Detection and Ranging (LiDAR) data.

Studies [18, 23, 31, 39, 48] based on high-resolution images (such as
satellite or optical stereo images) estimate building height via
methods such as elevation comparison and shadow detection, which
may be impacted by lighting and weather condition when the im-
ages are taken. Similarly, methods based on height estimation is
synthetic aperture radar (SAR) images [7, 37, 41] are mainly based
on shadow or layover analysis. Methods based on aerial images and aerial LiDAR data [33, 36] usually segment, cluster and then reconstruct building rooftop planar patches according to predefined geometric structures or shapes [48]. LiDAR data is expensive to analysis and has a limited operating altitude because the pulses are only effective between 500 and 2,000 meters [44]. A common limitation shared by the methods above is that the data that they use are expensive to collect, which significantly constrains the scalability of these methods.

**Method based on street scene image and 2D map.** Yuan and Cheriyadat propose a method for building height estimation uses street scene images facilitated by 2D maps [46]. Street scene images are widely available from Google Street View [2], Bing Street-Side [21] and Baidu Map [5], which makes building height estimation based on such data easier to scale. Yuan and Cheriyadat’s method has four main steps: (i) Match buildings in a street scene image with their footprints in a 2D map via camera projection based on the camera location that comes with the image. Here, the camera location may be imprecise due to GPS error [15, 47]. (ii) Calibrate the camera location via camera projection with the extracted building corner lines from street scene images. (iii) Re-match buildings from a 2D map with those in the street scene image based on the calibrated camera location, and then detect building rooftlines through edge detection methods. (iv) Compute building height via camera projection with camera parameters, calibrated camera location, the height of building rooftlines in the street scene image, and the building footprint in the 2D map.

Our proposed CBHE differs from Yuan and Cheriyadat’s method in the following two aspects: (A) In Step (ii) of their method, they calibrate camera location by matching building corner lines in the street scene image with building footprints in the 2D map. Such a method cannot handle images in urban areas where the corner lines of different buildings are too close to be differentiated, or the buildings have non-uniform patterns/colors on their facades which makes corner lines difficult to recognize. CBHE uses building corners instead of corner lines, which puts more restriction on the references for camera location calibration, and thus yields more accurate results. (B) In Step (iv) of their method, they use a local spectral histogram representation [26] as the edge indicator to capture building rooftlines, which can be ineffective when buildings overlap with each other. CBHE uses the proposed deep neural network named BuildingNet to learn a latent representation of building rooftlines, which has been shown to be more accurate in the experiments.

### 3 OVERVIEW OF CBHE

We present the overall procedure of our proposed CBHE in this section. We also briefly present the process of camera projection, which forms the theoretical foundation of building height estimation using street scene images.

#### 3.1 Solution Overview

We assume a given street scene image of buildings that comes with geo-coordinates and angle of the camera by which the image is taken. Here, the geo-coordinates may be imprecise due to GPS errors. Google Street View images are examples of such images, and we aim to compute the height of each building in the image. As illustrated in Fig. 2, CBHE contains three stages:

- **Stage 1 – Preprocessing:** In the first stage, we pre-process the input image by recognizing the buildings and computing their sketches. There are many methods for these purposes. We use two existing models RefineNet [24] and Structured Forest [12] to identify the buildings and compute their sketches, respectively. After this step, the input image will be converted into a grayscale image with each pixel valued from 0 to 255 that contains building sketches, which enables identifying rooftlines and computing the height of the building via camera projection.

- **Stage 2 – Camera location calibration:** Before computing building height by camera projection, in the second stage, we calibrate the camera location. This is necessary because a precise camera location is required in the camera projection, while the geo-coordinates that come with street scene images are imprecise due to GPS errors. To calibrate the camera location, we first detect building corner candidates in street scene images according to the geo-coordinates and angle of the camera by which the image is taken. Here, the geo-coordinates may be imprecise due to GPS errors. Google Street View images are examples of such images, and we aim to compute the height of each building in the image. As illustrated in Fig. 2, CBHE contains three stages:

- **Stage 3 – Building height computation:** In the third stage, we compute the height of each building in the image via camera projection with camera parameters, calibrated camera location, the height of building rooftlines in the 2D map, and the building footprint in the 2D map.
to their footprints in 2D maps and their relative position to the camera. Then, by comparing the locations and the projected positions of building corners (two corners), we calibrate the camera location via camera projection. In this stage, we propose a deep neural network named BuildingNet to determine whether an image segment contains a valid building corner. The BuildingNet model and the process of selecting two building corners for the calibration are detailed in Section 4.

- **Stage 3 – Building height computation**: In this stage, we obtain the roofline candidates of each building via weighted Hough transform and filter out those invalid roofline candidates via BuildingNet. Then we rank all valid rooflines by an entropy-based ranking algorithm considering both corner and roofline features and select the best one for computing building height via camera projection. The detailed process is provided in Section 5.

Since Stage 1 is relatively straightforward, we focus on Stages 2 and 3 in the following Sections 4 and 5, respectively. Before diving into these two stages, we briefly discuss the idea of camera projection and present the frequently used symbols.

### 3.2 Camera projection

We use Fig. 3 to illustrate the idea of camera projection and the corresponding symbols. In this figure, there are two coordinate systems, i.e., the camera coordinate system and the image plane coordinate system. Specifically, \( \{o',x',y',z'\} \) represent the camera coordinate system, where origin \( o' \) represents the location of the camera. The camera is set horizontal to the sea level, which means that plane \( x'z' \) is vertical to the building facades while the \( y' \)-axis is horizontal to the building facades. We use \( \{o,x,y\} \) to represent the image plane coordinate system, where origin \( o \) is the center of the image, and plane \( xy \) is parallel to plane \( x'y' \).

In Fig. 3, there are two buildings \( B_1 \) and \( B_2 \) that have been projected onto the image. For each building, we use \( l_r \), \( l_f \), and \( l_c \) to represent the roofline, the floor, and the line on the building projected to the \( x \)-axis (center line) of the image plane \( xy \). Corners \( c_n, c_x, \) and \( c_z \) are the corner nearest to the camera, the corner farthest to the \( y \)-axis of the image plane when projected to the image plane (along the \( x \)-axis), and the corner closest to the \( y \)-axis of the image plane when projected to the image plane (along the \( z \)-axis), respectively. The height \( h \) of the building is the sum of the distance between \( l_r \) and \( l_c \) and the distance between \( l_r \) and \( l_f \). These two distances are denoted as \( h'_r \) and \( h'_b \), and the projected length of \( h'_r \) in the image plane \( xy \) is denoted by \( h_r \). Since the camera is set horizontal to the sea level, the height of \( h'_b \) is the same as the height of the car or human beings who captured the street scene image, which can be regarded as a constant.

Let \( d \) be the distance from the camera \( o' \) to corner \( c_n \), \( d' \) be the projected length of \( d \) onto the \( x' \)-axis, and \( f \) be the focal length of the camera (i.e., the distance between the image center \( o \) and the camera center \( o' \)). Based on the pinhole camera projection, the height of a building can be computed as follows:

\[
    h = h'_r + h'_b = h_r \cdot d/f + h'_b 
\]

(1)

In this equation, the focal length \( f \) comes with the input image as its metadata. The distance \( d' \) is computed based on the geographic coordinates of the building and the camera, as well as the orientation of the camera. The geographic coordinates of the building are obtained from an open-sourced digital map, OpenStreetMap, while the geographic coordinates and orientation of the camera come with the input image from Google Street View. Due to GPS errors, we describe how to calibrate the location of the camera in Section 4. The height \( h_r \) is computed based on the position of the roofline \( l_r \) which is discussed in Section 5. Table 1 summarizes the symbols that are frequently used in the rest of the discussion.

### 4 Camera location calibration

When applying camera projection for building height estimation, we need the distance \( d \) between the building and the camera. Computing this distance is based on the locations of both the building and the camera. Due to the error of GPS, we calibrate the camera location in this section.

#### 4.1 Key Idea

We use two building corners in the street scene image with known real-world locations for camera location calibration. To illustrate the process, we project Fig. 3 to a 2D plane, as shown in Fig. 4a, and assume that corner \( c_n \) of building \( B_1 \) and building \( B_2 \) are two reference corners.

We consider a coordinate system with corner \( c_n \) of building \( B_1 \) as the origin, and the camera orientation as the \( y \)-axis. Let \( \theta_1 \) and \( \theta_2 \) be the angles of corner \( c_n \) of \( B_1 \) and corner \( c_n \) of \( B_2 \) from the orientation of the camera, respectively. Then the ratio of \( d_2/d_1 \) is determined by the position of these two reference corners in the image. \( \theta_2 \) represents the angle between the line connecting corner \( c_n \) of \( B_1 \) and corner \( c_n \) of \( B_2 \) and \( x \)-axis, and it can be computed.
according to the camera’s orientation and the relative locations of the two reference corners in 2D maps. Therefore, we can compute the coordinates \((x, y)\) of corner \(c_n\) of building \(B_2\) in the coordinate system. With \(\theta_1, \theta_2\), and the coordinate \((x, y)\), we compute the \(y\) coordinate of the camera as follows:

\[
y' = \frac{x - y \cdot \tan \theta_1}{\tan \theta_1 + \tan \theta_2}
\]

(2)

Since the \(x\) coordinate of the camera is equal to \(y' \cdot \tan \theta_2\), we obtain the relative position of the camera to the corner \(c_n\) of building \(B_1\). Thus, camera location calibration becomes the problem of matching two building corners with their positions in the image.

The real-world location of the building corners can be obtained from 2D maps, and we need to locate their corresponding positions in the street scene image based on the (inaccurate) geo-coordinates of the camera. For a pinhole camera, matching a 3D point in the real world to a 2D point in the image is determined by a 3x4 camera projection matrix as follows:

\[
\alpha \cdot p = \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} f_0 & 0 & 0 \\ 0 & f_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \quad I = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

(3)

where a real-world point \(p' = (x', y', z')^T\) can be projected to its position \(p = (x, y, 1)^T\) in the image plane; \(\alpha\) is the parameter that transfers pixel scale to millimeter scale [28]; \([I|0]_3\) is the camera matrix determined by focal length \(f\); \(R\) is the camera rotation matrix, while \(t\) is a 3-dimensional translation vector that describes the transformation from the real-world coordinates to the camera coordinates.

Since the image geo-coordinates may be inaccurate, we can only compute rough locations of the building corners. Based on the rough position of each corner, we then iteratively assume a height \(h_n\) for each building to obtain the gradient of its rooflines, as shown in Fig. 4b. We use 120 × 120 sub-images with the horizontal position and the assumed height of the corner as their center for building corner detection. A building corner consists of two rooflines or a roofline with a building corner line, as shown in Fig. 4b. For each building, we only consider their corners \(c_{n}\) and \(c_{z}\). There are three types of formation for corner \(c_{n}\) as illustrated by the red lines on the left-hand side building in Fig. 4b, and there is one type of formation for corner \(c_{z}\) as illustrated by the blue lines. Based on the detected building corner candidates, we use BuildingNet described Section 4.2 to filter out non-corner image segments, and then select the two reference corners which is discussed in Section 4.2.

We assume the camera location error from Google Street View API to be less than three meters due to its camera location optimization [20]. If the camera location we compute is more than three meters away from the one provided by Google Street View API, we use the camera location from Google Street View API directly. We further improve the estimation accuracy by a multi-sampling strategy, which uses the median height among results from different images of the same building taken at different distances.

### 4.2 BuildingNet

We formulate building corner detection as an object classification problem, which first detects candidate corner regions for a specific building by a heuristic method, and then classifies them into different types of corners or non-corners.

We classify images that may contain building corners into five classes. The first four classes correspond to images containing one of the four types of building corners, i.e., corner \(c_n\), \(c_z\) of the left-hand side buildings, and corner \(c_n\), \(c_z\) of the right-hand side buildings ("n", "s", "f", "h"). The last class corresponds to non-corner images which may contain any pattern except the above four types of corners (e.g., they could contain trees, lamps or power lines), and should be filtered out. Such a classification problem is an open set problem in the sense that the non-corner images do not have a unified pattern and will encounter unseen patterns. To solve this classification problem, we build a classifier that only requires samples of the first four classes in the training stage (can also take advantage of non-corner images), while can handle all five classes in the testing stage. To enable such a classifier, we first propose the BuildingNet model based on LeNet 5 [22] and triplet loss functions, which learns embeddings that map potential corner region image segments to a Euclidean space where the embeddings have small intra-class distances and large inter-class distances.

#### 4.2.1 Triplet Relative Loss Function

As shown in Fig 5, an input of BuildingNet contains three images. Two of them \((x^f, x')\) contain the same type of corner, and we name them the target \((x^f)\) and positive \((x')\), respectively. The other image \(x^p\) contains another type of corners (or a non-corner image if available), and we name it negative. BuildingNet trains its inputs to \(d\)-dimensional embeddings based on a triplet relative loss function inspired by Triplet-Center Loss and FaceNet [17, 35, 40, 43], which minimizes the distances within the same type of corners, and maximizes the distances between different types of corners as follows:

\[
I = \sum_{i=1}^{N} \alpha \cdot ||f(x_i^f) - f(x_i^p)||^2_2 + (1 - \alpha) \cdot \frac{||f(x_i^f) - f(x_i^f)||^2_2}{||f(x_i^f) - f(x_i^f)||^2_2}
\]

(4)

where \(\alpha \in [0, 1]\) is the weight of intra-class distance in the \(d\)-dimensional Euclidean space; \((1 - \alpha)\) is the weight of the ratio between intra-classes distance and inter-class distance, which aims to separate different classes in the \(d\)-dimensional Euclidean space; \(N\) is the cardinality of all input triplets. Function \(f\) computes the \(d\)-dimensional embedding of an input image, and we normalize it to \(||f(x)||^2_2 = 1\). Different from existing loss function based on triplet selection [35, 42], triplet relative loss function can minimise the intra-class distance and maximize the inter-class distance by means of their relative distance.
4.2.2 Hard Triplet Selection. Generating all possible image triplets for each batch during the training process will result in a large amount of unnecessary training data (e.g., $x^t$ and $x^b$ are too similar, while $x^n$ is way different). It is crucial to select triplets that contribute more to the training phase. In BuildingNet, we accelerate convergence by assigning a higher selection probability to triplets that may contribute more to the training process. The probability of selecting a negative image $x^n_i$ to form a training triplet is:

$$p(x^n_i) = \frac{e^{||f(x^n_i)-f(x^t)||_2^2-m}}{\sum_{j=1}^k e^{||f(x^n_i)-f(x^t)||_2^2-m}}, \quad i = [1,k]$$

$$m = \min(||f(x^n_i)-f(x^t)||_2^2 - ||f(x^n_i)||_2^2), \quad i = [1,k]$$

Here, $k$ is the total number of negative images in a batch. After randomly choosing $x^t$ and $x^b$ for a triplet, we compute the Euclidean distance between $x^t$ and $x^b$, as well as the distances between $x^t$ and the $k$ negative images $x^n_i$ in the batch. Let $m$ be the minimum Euclidean distance between $x^t$ and any $x^n_i$, which can be positive or negative. Then, the negative image $x^n_i$ similar to $x^t$ will have a higher probability to be selected.

After the training process, we obtain a $d$-dimensional embedding for each input image. We then learn a support vector classifier [8] based on these embeddings for corner region image classification.

4.3 Entropy-based Ranking

BuildingNet can filter out non-corner images. Among the remaining corner candidates, we select the two corners with the highest score as the reference corners. Reference corner selection relies on multiple factors: the length and edgeness (detailed in Section 5) of the lines forming the corner, the number of other corner candidates ($c_n$, $c_x$, and $c_z$) of the same building with the same assumed height, and the position of the corner in the image. We use the position of the corner into consideration because, empirically, corners close to one quarter or three-quarters (horizontally) of the image yield more accurate matching between their positions in the image and their footprints in 2D maps. We also consider their real-world locations because a corner close to the camera will be clearer and has higher accuracy when matching them to their footprints in 2D maps. Therefore, we define the score of each corner candidate as:

$$[s_{c_1}, ..., s_{c_k}] = \begin{bmatrix} \lambda, \omega, \tau, \rho, d|c_1 \end{bmatrix} \cdot \begin{bmatrix} w_\lambda \\ \vdots \\ \lambda, \omega, \tau, \rho, d|c_k \\ w_d \end{bmatrix}$$

where $k$ is the number of corner candidates from all buildings; $c_i$ is the $i$th corner candidate; $s_i$ is the score of the $i$th corner candidate; $\lambda$ is the detected length of the two lines that form a corner, while $\omega$ is the sum of the edgeness of the two lines; $\tau$ is the number of other corner candidates of the same building with the same assumed height; $\rho$ is the minimum distance from the corner to a quarter or three-quarters of the image, and $d$ is the distance from the corner to the camera; $w_\lambda, w_\omega, w_\tau, w_\rho, w_d$ are the weight of these parameters. Parameters $\lambda, \omega, \tau, \rho$ correlate with the score positively, while parameter $d$ correlates with the score negatively.

We use an entropy-based ranking method to compute the weights of parameters ($w_\lambda, ..., w_d$) [8]. Shannon entropy is a commonly used measurement of uncertainty in information theory [38]. The main idea of the entropy-based ranking algorithm is to compute the objective weights of different parameters according to their data distribution. If the samples of a parameter vary greatly, the parameter should be considered as a more important feature and thus should be given a larger weight.

For building corner classification, there are $n = 5$ parameters and $m = k$ samples. We denote the decision matrix as $r$, where $r_{ij}$ is the value of the $i$th sample under the $j$th parameter. Before applying the entropy-based ranking algorithm, we pre-process these parameters by Min-max scaling as follows:

$$r_{ij} = \begin{cases} \frac{(r_{ij} - \min(r_{ij}))}{(\max(r_{ij}) - \min(r_{ij}))}, & \text{iff positve} \\ \frac{(r_{ij} - \min(r_{ij}))}{(\max(r_{ij}) - \min(r_{ij}))} + 1, & \text{iff negative} \end{cases}$$

where positive and negative mean that the $j$th parameter is positively/negatively correlated with the value of $r$. After Min-max scaling, the entropy of each parameter based on the normalized decision matrix $r'$ is defined as:

$$e_j = -\ln(m)^{-1} \sum_{j=1}^m r'_{ij} \cdot \ln(r'_{ij}), \quad r'_{ij} = \frac{r_{ij}}{m} \sum_{j=1}^m r_{ij}$$

where $r'$ is the standardized $r$. Based on the entropy of each parameter, the weight of each parameter is computed by:

$$w_j = (1-e_j)/(n-\sum_{j=1}^m e_j), \quad j = [1,n]$$

After computing the weight of each parameter, we apply them to all corner candidates and rank all the candidates by their scores to obtain the best two as the reference corners.

5 ROOFLINE DETECTION

Building height estimation requires detecting the roofline of each building. In this section, we present our method for roofline candidate detection in Section 5.1, and our method for the true roofline selection in Section 5.2. We further present a strategy for handling tall building (over 100 meters) in Section 5.3.

5.1 Roofline Candidate Detection

We consider the rooflines from corner $c_n$ to the corner next to $c_n$, along the positive direction of the $x'$-axis in the camera coordinate system, and the one from corner $c_z$ to the corner next to corner $c_z$.
along the negative direction of the $z'$-axis in the camera coordinate system. The corner between corner $c_x$ and corner $c_x$ is corner $c_y$ if they are adjacent to each other, as shown in Fig. 3, and we take this situation to simplify the explanation.

Similar to corner candidate detection, as shown in Fig. 6a, we find all roofline candidates of each building by a heuristic method, which projects the rooflines of each building according to its relative location to the camera in the real world, together with the camera's parameters. To do so, we first assume $h_r$ of a building to be the maximum height that can be captured, which means that at least a roof corner $(c_x, c_y, z_r)$ is visible in the image. If corner $c_y$ is visible, the maximum height computed via camera projection is:

$$h_r = d \cdot (h_f/2f)$$ (10)

where $h_f$ is the height of the street scene image; $d$ is the distance from corner $c_y$ to the camera projected to the $z'$-axis of the camera coordinate system. If corner $c_y$ is invisible, we use $c_x$ as the reference corner when computing the maximum height of a building in the same way. With the maximum height of the building, we compute the position of corner $c_x$, $c_y$, and $c_z$ in the image. We then apply Hough transform to the input edge map in Fig. 2 to detect roofline candidates, and the roofline candidates from $c_x$ to $c_z$ need to match the computed position of $c_x$ and $c_z$. Similarly, the roofline candidates from $c_z$ to $c_y$ need to match the computed position of $c_z$ and $c_x$. Instead of using the typical Hough transform for line detection, which takes binarized images as the input, we use $M$, the mask of detected buildings.

5.2 Roofline Classification and Ranking

There are three types of rooflines: (i) Roofline from $c_x$ to $c_y$; (ii) Roofline from $c_y$ to $c_x$ of the left hand side buildings; (iii) Roofline from $c_z$ to $c_y$ of the right hand side buildings, as shown in Fig. 4c. We use BuildingNet to filter these candidates and find the true roofline, which is similar to the corner candidate validation process in Section 4.2. Based on the valid roofline candidates from BuildingNet, we weight each roofline candidate $l_r$ by its detected length $\lambda$, edgeness $\omega$, and the number of corners $\tau$ with the same assumed height of the same building. We rank all roofline candidates via the entropy-based ranking algorithm in Section 4.3, as follows:

$$[s_{l_{r_1}}, ..., s_{l_{r_k}}]' = \begin{bmatrix} \lambda, \omega, \tau | l_{r_1} \rvert \\ \vdots \\ \lambda, \omega, \tau | l_{r_k} \rvert \\ \lambda, \omega, \tau | l_{r_k} \rvert \\ w_{\lambda} | l_{r_1} \rvert \\ \vdots \\ w_{\lambda} | l_{r_k} \rvert \\ w_{\omega} | l_{r_1} \rvert \\ \vdots \\ w_{\omega} | l_{r_k} \rvert \\ w_{\tau} | l_{r_1} \rvert \\ \vdots \\ w_{\tau} | l_{r_k} \rvert \end{bmatrix}$$ (11)

where $k$ is the number of roofline candidates for a specific roofline; $l_{r_i}$ is the $i$th roofline candidate; $s_{l_{r_i}}$ is the score of the $i$th roofline candidate. $w_\lambda$, $w_\omega$, and $w_\tau$ are the weight of these parameters based on all candidates of a specific roofline of a building, and all three parameters are positively correlated with the score $s$. The value of $\tau$ depends on the number of corners $(c_x, c_y, z_r)$ with the same assumed height as the roofline candidate, and its value is $\{0, 1, 2, 3\}$. We discussed how to detect references corners in Section 4.2, and the difference in detecting the corners of a specific building is that we do not consider $\rho$ and $d$ in Equation 6 and all corner candidates here are those of a specific building corner.

Different from building corners, which can only be visible or invisible, rooflines can also be partially blocked by other objects (trees in particular). Therefore, before we apply the ranking algorithm, we pre-process the length $\lambda$ and edgeness $\omega$ which are affected by the blocking via Algorithm 1 as follows:

When estimating building height, we first separate buildings into two classes: (i) with at least one valid corner; (ii) without any valid corner. Then, we process buildings in class (i) according to their distance to the camera. After all the buildings in class (i) have been processed, we process the buildings in class (ii) according to their distance to the camera. After we obtain the height of a building, we mark the scope of the building in the street scene image, as shown in Fig. 6b (i.e., height has been obtained).

After detecting the roofline candidates of a building, we refine the $\lambda_{l_r}$ of each roofline candidate using the following equation:
where \( h \) is the input edge map of the original image, \( \lambda_{l_r}^{(ini)} \) and \( \lambda_{l_r} \) are the initial and prolonged length of the roofline, respectively.

### 5.3 Tall Building Preprocessing

Height estimation for tall buildings (over 100 meters) requires the camera to be placed far away from the buildings with an upward-looking view to capture the building roof. For images with an upward-looking view, all building corner lines will become slanted. Typically, we take the upward-looking view as 25 degrees as an example to show the strategy that we use for handling tall buildings.

We first compute a plane-to-plane homography \([51]\), which maps an image with an upward-looking view to the corresponding image with a horizontal view. Here, we use the homogeneous estimation method \([10]\), which solves a \(3 \times 3\) homogeneous matrix \( h \) that matches a point in an upward-looking image (Fig. 7a) to a horizontal-view image (Fig. 7b) using the Equation 14:

\[
A \cdot h = \begin{bmatrix}
  x_1 & y_1 & 1 & 0 & 0 & 0 & -x_2 & y_2 & x_1 \\
  0 & 0 & x_1 & y_1 & 1 & -x_3 & y_3 & -y_1 & x_1 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  x_n & y_n & 1 & 0 & 0 & 0 & -x_n & y_n & x_n \\
  0 & 0 & x_n & y_n & 1 & -x_n & y_n & -y_n & x_n
\end{bmatrix} \cdot h = 0
\]  

where the homogeneous matrix \( h(\|h\| = 1) \) is represented in the vector form as \( h = (h_{11}, h_{12}, h_{13}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^T \); \( n \) is the number of point pairs, which should be no less than four to validate the homogeneous equation; \((X_t, Y_t)\) represents a point in the upward-looking image and \((x_i, y_i)\) represents the corresponding point in the resultant image with a horizontal view. Vector \( h \) minimizes the algebraic residuals, and \( A \cdot h \) is a standard result of linear algebra. Subject to \( h = 1 \), the least eigenvalue of \( A \cdot A^T \) is given by the eigenvector, and this eigenvector can be obtained from the singular value decomposition (SVD) of \( A \).

### 6 EXPERIMENTS

In this section, we first evaluate our proposed BuildingNet model for building corner and roofline classification and then evaluate our proposed CBHE algorithm for building height estimation.

#### 6.1 Datasets

In our experiments, we obtain building footprints (geo-coordinates) from OpenStreetMap and building images from Google Street View, respectively. For the experiments on building height estimation, we use two datasets:

- **City Blocks**, which contains 128 buildings in San Francisco. We collect all Google Street View images (640×640 pixels) with camera orientation along the street. We set the view of the camera is 90 degrees, and the focal length can be derived via the camera parameters provided by Google. We do not need to consider the camera rotation matrix \( R \) and the translation vector \( t \) due to the image preprocessing of Google Street View. We obtain the building height ground truth from high-resolution aerial maps (e.g., NearMap \([13]\)).

- **Tall Buildings**, which contains 37 buildings taller than 100 meters in San Francisco, Melbourne, and Sydney collected by us via Google Street View API. We set the camera with an upward-looking view (25 degrees) to capture their rooflines. The building height ground truth comes from Wikipedia pages of these buildings or derived from NearMap \([13]\).

For building corner classification, we crop images from City Blocks dataset. We generate the corner dataset semi-automatically, where we crop 28×28 pixels image segments from street scene images, and then manually label whether an image segment contains a building corner (and the type of corner). The training dataset that we collected contains 10,400 images, including 1,300 images of each type of building corner (i.e., a total of 5,200 building corner images) and 5,200 non-corner images. The testing dataset contains 1,280 images, including 160 images for each type of building corners and 640 non-corner images. The training data and testing data come from different city blocks.

Following a similar approach, we collect a roofline dataset. For each roofline candidate, we extend the upper and lower 10 pixels of the roofline to obtain a 21×W image segments, where \( W \) is the length of the roofline, and we further resize (rotate if the roofline is not a horizontal line) the image to 28×28 to generate same-size inputs for BuildingNet. The training dataset includes 7,800 images, including 1,300 images for each type of rooflines (i.e., a total of 3,900 images for the each building roofline).

![Figure 7: (a) an image with an upward-looking view. (b) the corresponding image with a horizontal view.](image)

![Figure 8: Examples of four types of corners, three types of rooflines, and the corresponding unlabelled images.](image)
we pre-train LeNet 5 with the MNIST dataset and fine-tune it with the default setting in Caffe [19]. We then apply the last fully connected layer to OpenMax for classification. It simplifies the open set classification problem by testing the probability of whether an input comes from unknown classes based on the last fully connected layer of a neural network. Further, we use two loss functions based on triplet selection to illustrate the effectiveness of our proposed triplet relative loss function. The loss function in FaceNet [35] makes the intra-class distance smaller than inter-class distance by adding a margin, and the one in MSML [45] optimizes the triplet selection process towards selecting hard triplets in each in training.

Hyperparameters. For the OpenMax, we use the LeNet 5 model to train on the building corner and roofline dataset for 10K iterations with the default setting in Caffe [19]. We then apply the last fully connected layer to OpenMax for classification. For our BuildingNet, we pre-train LeNet 5 with the MNIST dataset and fine-tune it with our collected building corner and roofline images. Further, since BuildingNet can also take advantage of unlabeled data (known unknown [34]) during training, we also pre-train a LeNet 5 model based on MNIST dataset (0 to 4 as the labeled data and 5 to 9 as the unlabeled data) and fine-tune it with our data. We set the learning rate as 0.1 with the decay rate of 0.95 after each 1K iterations (50K iterations in total). The batch size is 30 images for each class, the embeddings that BuildingNet learns are 128-dimensional, and the margin in the triplet relative loss function is 0.5. We perform 10-fold cross-validation on the models tested, and then compute the accuracy, precision, recall, and $F_1$ score of different models, which are summarized in Figure 9.

On the corner dataset, BuildingNet achieves an accuracy of 94.34%, and its recall, precision, and $F_1$ score are all over 91% when using both labeled and unlabeled data for training. Compared with SROSR and OpenMax, BuildingNet improves the accuracy and $F_1$ score by more than 6% and 10%, respectively. When trained with labeled data only, BuildingNet still has the highest accuracy and $F_1$ score (i.e., 88.72% and 81.8%), which are 1.1% and 2% higher than OpenMax, respectively. Compared with the two loss functions in MSML and FaceNet which are also based on triplet selection, our proposed loss function can improve the accuracy and $F_1$ score by more than 0.4% and 0.5%, respectively. For the roofline dataset, the proposed BuildingNet again outperforms the baseline models consistently. These confirm the effectiveness of BuildingNet.

To further illustrate the effectiveness of BuildingNet, we visualize the embeddings generated by three triplet based loss functions on the corner dataset, as shown in Fig. 10. Compared with random triplet selection with margin (FaceNet) and hard triplet selection with margin (MSML), our triplet relative loss function obtains better classification result with smaller average intra-class distance and larger average inter-class distance after the same number of epochs.

6.2 Effectiveness of BuildingNet

Building corner and roofline classification is an open set classification problem where the invalid corner or roofline candidates do not have consistent features. To test the effectiveness of BuildingNet, we use two open set classifiers as the baselines: SROSR [49] and OpenMax [6]. SROSR uses the reconstruction errors for classification. It simplifies the open set classification problem into testing and analyzing a set of hypothesis based on the matched and no-matched error distributions. OpenMax handles the open set classification problem by estimating the probability of whether an input comes from unknown classes based on the last fully connected layer of a neural network.

Further, we use two loss functions based on triplet selection to illustrate the effectiveness of our proposed triplet relative loss function. The loss function in FaceNet [35] makes the intra-class distance smaller than inter-class distance by adding a margin, and the one in MSML [45] optimizes the triplet selection process towards selecting hard triplets in each in training.

Hyperparameters. For the OpenMax, we use the LeNet 5 model to train on the building corner and roofline dataset for 10K iterations with the default setting in Caffe [19]. We then apply the last fully connected layer to OpenMax for classification. For our BuildingNet, we pre-train LeNet 5 with the MNIST dataset and fine-tune it with our collected building corner and roofline images. Further, since BuildingNet can also take advantage of unlabeled data (unknown unknown [34]) during training, we also pre-train a LeNet 5 model based on MNIST dataset (0 to 4 as the labeled data and 5 to 9 as the unlabeled data) and fine-tune it with our data. We set the learning rate as 0.1 with the decay rate of 0.95 after each 1K iterations (50K iterations in total). The batch size is 30 images for each class, the embeddings that BuildingNet learns are 128-dimensional, and the margin in the triplet relative loss function is 0.5. We perform 10-fold cross-validation on the models tested, and then compute the accuracy, precision, recall, and $F_1$ score of different models, which are summarized in Figure 9.

On the corner dataset, BuildingNet achieves an accuracy of 94.34%, and its recall, precision, and $F_1$ score are all over 91% when using both labeled and unlabeled data for training. Compared with SROSR and OpenMax, BuildingNet improves the accuracy and $F_1$ score by more than 6% and 10%, respectively. When trained with labeled data only, BuildingNet still has the highest accuracy and $F_1$ score (i.e., 88.72% and 81.8%), which are 1.1% and 2% higher than OpenMax, respectively. Compared with the two loss functions in MSML and FaceNet which are also based on triplet selection, our proposed loss function can improve the accuracy and $F_1$ score by more than 0.4% and 0.5%, respectively. For the roofline dataset, the proposed BuildingNet again outperforms the baseline models consistently. These confirm the effectiveness of BuildingNet.

To further illustrate the effectiveness of BuildingNet, we visualize the embeddings generated by three triplet based loss functions on the corner dataset, as shown in Fig. 10. Compared with random triplet selection with margin (FaceNet) and hard triplet selection with margin (MSML), our triplet relative loss function obtains better classification result with smaller average intra-class distance and larger average inter-class distance after the same number of epochs.

6.3 Effectiveness of CBHE

We evaluate the performance of CBHE on City Blocks and Tall Buildings in this subsection.

6.3.1 Building height estimation on City Blocks. Figure 11 shows the building high estimation errors of the baseline method [46] and CBHE over the City Blocks dataset. It shows the percentage of buildings where the height estimation is greater than 2, 3, and 4 meters, respectively. In both city blocks, CBHE achieves a smaller percentage of buildings than that of the baseline [46].

In particular, in the first city block (Fig. 11a, which has been used in [46]), CBHE has 10.4%, 5.5%, and 1.2% fewer buildings than those of the baseline with height estimation errors greater than 2, 3, and 4 meters, respectively. Note that the results of the baseline method are obtained from their paper [46] since we are unable to obtain
6.3.2 Building height estimation on Tall Buildings. For tall buildings, the camera needs to be placed far away with an upward-look view to capture the building roofline. We capture the building images 250 meters away from the buildings via Google Street View API. Fig. 13 presents examples of the street scene images for tall building height estimation. For each street scene image, we first rotate it to the horizontal view according to Equation 14, and then compute the height of the buildings according to Section 5.

Figure 13: Tall building examples (best view in color).

We proposed a corner-based algorithm named CBHE to learn building height from complex street scene images. CBHE consists of camera location calibration and building roofline detection as its two main steps. To calibrate camera location, CBHE performs camera projection by matching two building corners in street scene images with their physical locations obtained from a 2D map. To identify building rooflines, CBHE first detects roofline candidates according to the building footprints in 2D maps and the calibrated camera location. Then, it uses a deep neural network named BuildingNet that we proposed to check whether a roofline candidate indeed is a valid roofline. Finally, CBHE ranks the valid rooflines based on an entropy-based ranking algorithm, which also involves building corner information as an essential indicator, and then computes the building height through camera projection. Experimental results show that the proposed BuildingNet model outperforms two state-of-the-art classifiers SROSR and OpenMax consistently, and CBHE outperforms the baseline algorithm by over 10% in building height estimation accuracy.

7 CONCLUSIONS

We summarize the challenging cases for CBHE in this section. These challenging scenarios will be explored in future work.

Fig 11b shows the result in a second city block (which was not used in [46]). As we are unable to obtain the source code of the baseline method, the result is based on our implementation of their method. CBHE again outperforms the baseline. It has 11.5%, 4.8%, and 5% fewer buildings than those of the baseline with height estimation errors greater than 2, 3, and 4 meters, respectively.

6.4 Error Analysis

We summarize the challenging cases for CBHE in this section. These challenging scenarios will be explored in future work.

For those buildings whose rooflines are entirely blocked by other objects such as trees, CBHE will ignore them, or output a wrong estimation. Take Fig. 14a as an example, the trees on the left-hand side of the image block the roof of the green colored building heavily, resulting in a line below the roof to be identified as the roofline. Additionally, if the corners of a building are not detectable, lines from other buildings behind this building may also impact the result. As illustrated in Fig. 14b, the roofline of a building behind the blue colored building was detected as its roofline.

In dense city areas, the buildings may overlap with each other, and it is difficult to match all buildings with their boundaries in a 2D map accurately. Take Fig. 14c as an example, building 2 is blocked by building 1 and building 2's corners have a similar horizontal position to building 3. Therefore, CBHE regards the rooflines of building 3 as the rooflines of building 2, which results in the estimated height of building 2 being 77.41m, although its real height is 24.53m. Moreover, the height of building 3 is also wrong because the incorrect rooflines of building 2 block the rooflines of building 3. In Fig. 14d, the blue shaded building mask on the right-hand side is wrongly assigned to the building 2 (between building 1 and building 3) because it is closer to the camera with the similar position to building 1.

Table 2: CBHE for tall building height estimation.

| Absolute error | Percentage | Relative Error | Percentage |
|----------------|------------|----------------|------------|
| >5m            | 45.9%      | >5%            | 40.5%      |
| >10m           | 27.0%      | >10%           | 13.5%      |

The errors of tall buildings may seem larger due to the camera projection (i.e., the errors are multiplied by a larger multiplier for tall buildings). However, we would like to emphasize that the relative errors are still quite low, e.g., since the tall buildings are taller than 100 meters, even a 10-meter error is less than 10% and is barely notable in reality.
8 ACKNOWLEDGMENTS
We thank the anonymous reviewers for their feedback. We appreciate the valuable discussion with Bayu Distiawan Treseya, Weihao Chen, and Jungmin Son. Yunxiang Zhao is supported by the Chinese Scholarship Council (CSC). This work is supported by Australian Research Council (ARC) Discovery Project DP180102050, Google Faculty Research Award, and the National Science Foundation of China (Project No. 61402155).

REFERENCES
[1] Pratik Agarwal, Wolfram Burgard, and Luciano Spinello. 2015. Metric Localization using Google Street View. In IEEE/RSJ International Conference on Intelligent Robots and Systems. 3111–3118.
[2] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Alijibit欧阳, Luc Vincent, and Josh Weaver. 2010. Google Street View: Capturing the World at Street Level. Computer 43, 6 (2010), 32–38.
[3] Anil Armagan, Martin Hirzer, Peter M. Roth, and Vincent Lepetit. 2017. Learning to Align Semantic Segmentation and 2D Maps for Geolocalization. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 4590–4597.
[4] Clemens Arth, Christian Pirchheim, Jonathan Ventura, Dieter Schmalstieg, and Martin Hirzer. 2015. Deep Learning for Practical Applications. In Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Computing. 554–558.
[5] Jiyan Pan, Martial Hebert, and Tarek Kanade. 2015. Inferring 3D Layout of Building Facades from a Single Image. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2918–2926.
[6] Peng Qi, John Z. Zhao, and Gaifeng Dong. 2015. Building Height Estimation using Google Earth. Energy and Buildings 110 (2016), 123–132.

APPENDIX A: DETAILS
[1] Edward Ng. 2009. Policies and Technical Guidelines for Urban Planning of High-Density Cities–Air Ventilation Assessment (AVA) of Hong Kong. Building and Environment 44, 7 (2009), 1478–1488.
[2] Elkin Díaz and Henry Arguello. 2016. An Algorithm to Estimate Building Heights for Practical Applications V. Vol. 99. 98680A.
[3] Feng Qi, John Z. Zhao, and Gaifeng Dong. 2015. Building Height Estimation using Google Earth. Energy and Buildings 110 (2016), 123–132.
[4] Adam Russell and Alexander Zeng. 2017. Towards a Landmark-Based Pedestrian Navigation Service using OSM Data. ISPRS International Journal of Geo-Information 6, 3 (2017), 64.
[5] Paul A. Zandbergen and Sean J. Barbeau. 2011. Positional Accuracy of Assisted Navigation. In Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Computing. 554–558.
[6] Mohammad Iftit and Parvaneh Saeedi. 2012. Three-Dimensional Polygonal Building Model Estimation from Single Satellite Images. IEEE Transactions on Geoscience and Remote Sensing 50, 6 (2012), 2254–2272.
[7] Xiong He, Yang Zhou, Zhichao Zhou, Song Bai, and Xiang Bai. 2018. Triplet-Center Loss for Multi-View 3D Object Retrieval. arXiv preprint arXiv:1803.06189 (2018).
[8] Mohammad Iftit and Parvaneh Saeedi. 2012. Three-Dimensional Polygonal Building Model Estimation from Single Satellite Images. IEEE Transactions on Geoscience and Remote Sensing 50, 6 (2012), 2254–2272.
[49] He Zhang and Vishal M. Patel. 2017. Sparse Representation-Based Open Set Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39, 8 (2017), 1690–1696.

[50] Rui Zhang. 2017. Geographic Knowledge Base (2017): http://www.ruizhang.info/GKB/gkb.htm.

[51] Zhengyou Zhang. 2000. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 22, 11 (2000), 1330–1334.
Author/s:
Zhao, Y; Qi, J; Zhang, R

Title:
CBHE: Corner-based Building Height Estimation for Complex Street Scene Images

Date:
2019-01-01

Citation:
Zhao, Y; Qi, J; Zhang, R, CBHE: Corner-based Building Height Estimation for Complex Street Scene Images, WEB CONFERENCE 2019: PROCEEDINGS OF THE WORLD WIDE WEB CONFERENCE (WWW 2019), 2019, pp. 2436 - 2447

Persistent Link:
http://hdl.handle.net/11343/225731

File Description:
Published version