

Scalable Building Height Estimation from Street Scene Images

Yunxiang Zhao[✉], Jianzhong Qi[✉], Flip Korn[✉], and Xiangyu Wang[✉]

Abstract—Building height estimation plays an essential role in many applications such as 3D city rendering, urban planning, and navigation. Recently, a new building height estimation method was proposed using street scene images and 2D maps, which is more scalable than traditional methods that use high-resolution optical images, RADAR, or LiDAR data, which are proprietary or expensive to obtain. The method needs to detect building rooflines to compute building height via the pinhole camera model. We observe that this method has limitations in handling complex street scene images where buildings occlude each other or are blocked by other objects such as trees since rooflines can be difficult to locate. To address these limitations, we propose a robust building height estimation method that computes building height simultaneously from street scene images with an orientation along the street and images facing the building with an upward-looking view. We first detect roofline candidates from both types of images. Then, we use a deep neural network called RoofNet to classify and filter these candidates and select the best candidate via an entropy-based ranking algorithm. When the true roofline is identified, we compute building height via the pinhole camera model. Experimental results show that the proposed RoofNet model yields a higher accuracy on building corner and roofline candidate filtering compared with state-of-the-art open-set classifiers. Our overall building height estimation method outperforms the baseline by up to 11.9% in accuracy and achieves 92.8% in height estimation error within 4 meters on the collected data set.

Index Terms—Building Height Estimation, Camera Location Calibration, Open Set Classification, Deep Neural Networks

I. INTRODUCTION

Building height is important in many applications, such as 3D city rendering for VR/AR applications [1], urban planning [2], and navigation [3], [4]. For example, building heights are useful to identify prominent buildings on a block, which can then be used to facilitate navigation via instructions such as “Turn left at the five-story building.”

Previous approaches to building height estimation are mainly based on high-resolution optical images [5], [6], synthetic aperture radar (SAR) [7], [8], [9], and Light Detection and Ranging (LiDAR) [10], [11], [12]. Such data are expensive to obtain, and hence the above approaches are not feasible at a large scale let alone for all buildings on earth [10]. Moreover, such data is often proprietary and publicly unavailable.

Corresponding authors: Jianzhong Qi, Flip Korn, and Xiangyu Wang.

Yunxiang Zhao is with Beijing Institute of Biotechnology, Beijing, China (e-mail: zhaoyx1993@163.com).

Jianzhong Qi is with The University of Melbourne, Melbourne, Australia (e-mail: jianzhong.qi@unimelb.edu.au).

Flip Korn is with Google Research, NYC, USA (e-mail: flip@google.com).

Xiangyu Wang is with East China Jiao Tong University, Jiangxi, China, and Curtin University, Perth, Australia (e-mail: xiangyu.wang@curtin.edu.au).

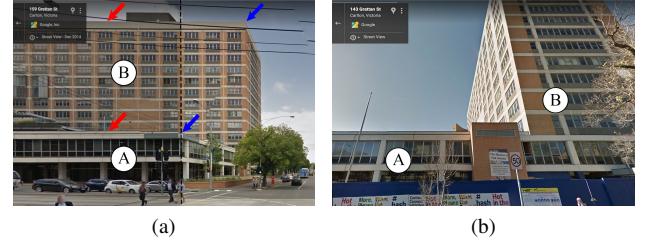


Fig. 1. (a) Building A’s rooflines are hard to detect due to the overlapping with building B while building A’s corners can help figure out the true roofline. Red arrows: building rooflines; Blue arrows: building corners. (b) Building A’s roofline is easier to detect but the estimated height is sensitive to GPS errors due to its close distance to the camera.

Recently, lower-resolution street scene images, such as Google Street View [13], together with 2D maps, such as OpenStreetMap [14], have become ubiquitous and have been applied to building height estimation. Estimating building height from street scene images relies on accurate detection of building rooflines, which then enables computation based on camera projection. However, the existing methods for roofline detection are not robust to occluded buildings since they consider only roofline segments [15], [16]. As Fig. 1a shows, the roofline of building B may be detected as the roofline of building A because the rooflines of adjacent buildings may be parallel to each other and have similar colors.

In this paper, we present a robust and scalable algorithm to estimate building height using *multi-sampled street scene images*. Given a 2D map such as OpenStreetMap, our algorithm automatically fetches all the street scene images from a map API such as Google Maps. To estimate the height of each building, we simultaneously consider two types of street scene images containing the building:

- Images with an orientation along the street (cf. Fig. 1a). We detect both building corners and rooflines from such images to compute the building height. Corners of different buildings do not share the same coordinates, and it is easier to associate them with different buildings. Building heights computed from these images tend to be accurate if the true roofline is detected. However, due to the building overlapping and blocking (e.g., by trees), the true roofline of a building may not be detected from such images.
- Images facing the building with an upward-looking view (cf. Fig. 1b). We capture both “slanted” building features and rooflines to compute the building height. Building heights estimated from these images can avoid drastic errors due to overlapping or blocking objects. However, the estimated heights may be inaccurate due to GPS errors compounded

by the close distance of the camera location to the building.

To take advantage of both types of images while avoiding their limitations, we jointly learn the building height based on both types of images. The heights from images facing the building help to eliminate large estimation errors, while the heights from images with an orientation along the street help to obtain an accurate result.

In the process above, when locating the roofline and corner candidates, we fetch two sets of image segments that contain building rooflines and corners, respectively. To filter each set of objects and identify the true roofline and corner images, we propose a deep neural network model named **RoofNet**. Building corners have a limited number of patterns (e.g., “ \nearrow ”, “ \nwarrow ”, “ $\nearrow\swarrow$ ”, and “ $\nwarrow\nearrow$ ”), while non-corner images (e.g., trees, lamps, or power lines) may have any pattern, which should be filtered out. The same applies to the building rooflines. Classifying images into different building corner (roofline) classes and non-corner (non-roofline) class is an *open-set* classification problem, since the non-corner (non-roofline) images do not have a unified pattern and may contain unseen patterns, where traditional deep neural networks may wrongly recognize it as one of the known classes. To solve this problem, RoofNet learns embeddings of the input images, which minimize the intra-class distance and maximize the inter-class distance. Following the design of FaceNet [17], it then differentiates different classes using a Support Vector Classification (SVC) model on the learned embeddings. When a new image segment comes, the trained SVC model tells whether it falls into any corner (roofline) classes. If the image does not fall into any corner (roofline) classes, it is a non-corner (non-roofline) image and can be discarded.

When estimating building height via the pinhole camera model, the GPS errors on the camera locations have a strong negative impact on the estimation accuracy. We thus calibrate the camera location before roofline detection. To achieve this, our model first detects candidate regions of all building corners in street scene images by matching buildings in street scene images with their (latitude and longitude) footprints in 2D maps based on the imprecise camera location from GPS. Then, it uses RoofNet to classify the corner candidates and remove those images classified as non-corners. From the surviving corners, our model selects two that are most likely the true corners of a building (detailed in Section IV-B) to triangulate the camera location via the pinhole camera model.

We summarize our contributions as follows:

- C₁. We propose an algorithm for large-scale street scene image collection. The algorithm automatically detects all the locations within a given 2D map that have street scene images taken and fetches such images from street scene image service APIs.
- C₂. We propose a building height estimation algorithm that learns the height from roofline candidates from two corresponding street scene images: (a) with an orientation along the street and (b) facing the building with an upward-looking view. Experiments on real-world data sets show that our model outperforms the state-of-the-art building height estimation method via street scene images by up to 11.9% in

building height estimation accuracy, and enables (accurate) estimation of tall buildings, which was not possible before. We model building corner and roofline detection as an open-set classification problem and propose a novel deep neural network named **RoofNet** to solve the problem. **RoofNet** learns embeddings of the input images, which minimize the intra-class distance and maximize the inter-class distance. Experimental results show that **RoofNet** achieves higher accuracy on building corner and roofline identification compared with existing open-set classifiers.

- C₃. We propose an entropy-based ranking algorithm to select the roofline candidate for building height estimation. For images with an orientation along the street, the algorithm considers features of both building corners and roofline candidates. For images facing the building with an upward-looking view, the algorithm considers features of roofline candidates and the slanted and partially visible building.
- C₄. We propose a camera location calibration algorithm with an analytical solution when given the locations of two building corners in a 2D map (a highly accurate result can be guaranteed when given valid building corners from **RoofNet**).

This paper is an extended version of our previous conference paper [18]. In this journal extension, we make new contributions: C₁, presented in Section III-B, plus C₂ and the latter part of C₄, both in Section V-B. In addition, we provide algorithm cost analysis in Section V-E, experiments on the large-scale street scene image collection algorithm in Section VI-B, a new open-set baseline [19] to be compared with **RoofNet** in Section VI-C, experiments on the improved building height estimation algorithms over larger data sets in Section VI-D, error analysis in Section VI-E, and sensitivity analysis in Section VI-F.

In particular, street scene images in our conference paper were manually collected. In this extension, we propose a street scene image collection algorithm for large-scale processing (C₁). To detect building rooflines, our conference paper considers street scene images with an orientation along the street, which may still miss the roofline of a building occluded by other buildings or blocked by objects such as trees. In this extension, we additionally consider rooflines from images facing the buildings. Since buildings in such images are slanted, our previous algorithm does not apply. We thus extend contribution C₄ to find rooflines. We then jointly learn building height from both street scene images with an orientation along the street and facing the building with an upward-looking view (C₂). Our new experiments show that the optimized algorithm can better handle the blocking and overlapping problems when estimating building height from street scene images.

II. RELATED WORK

In this section, we review studies on camera location calibration and building height estimation. We also detail our baseline method [15].

A. Camera Location Calibration

Camera location calibration aims to refine the camera location of the taken images, given a rough camera position from

GPS devices or the image localization [20], [21].

To calibrate device location in urban environments, various approaches have been proposed using 2.5D maps (2D maps with building height information), which can yield good localization results [22], [23], [24], [25], [26]. Here, the hurdle is the requirement of building height information for generating 2.5D maps, which may not be available for every building. [27], [28] extract the position of *building corner lines* (the vertical lines from the corner to the ground) and then find the camera location and orientation by matching the extracted position with building coordinates in 2D maps. This method cannot handle buildings overlapping with each other or having non-uniform patterns on their facades.

B. Building Height Estimation

Building height estimation has been studied using geographical data such as high-resolution images, SAR images, and LiDAR data. Studies [29], [5], [6], [30] based on high-resolution images (e.g., satellite or optical stereo images) estimate building height via methods such as elevation comparison and shadow detection, which may be impacted by the lighting and weather condition when the images were taken. Similarly, methods using synthetic aperture radar (SAR) images [7], [31], [8], [32] are mainly based on shadow or layover analysis. Methods based on aerial images and aerial LiDAR data [10], [33] usually segment, cluster, and then reconstruct building rooftop planar patches according to predefined geometric structures or shapes [34]. LiDAR data are expensive to analyze and have a limited operating altitude because the pulses are only effective between 500 and 2,000 meters [11]. A common limitation shared by the methods above is that their input data is expensive to collect, which significantly constrains the scalability of these methods.

[15] propose a method for building height estimation that uses street scene images facilitated by 2D maps. Street scene images are available from Google Street View [13], Bing StreetSide [35], and Baidu Maps [36]. Building height estimation based on such data is easier to scale. The method proposed by Yuan and Cheriyadat 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. Buildings are represented as “Shape” objects in 2D maps. Given a region of interest, they download all object data of the region (an XML file), where each building corresponds to a Shape and each corner of the Shape corresponds to a building corner. Here, the camera location may be imprecise due to GPS errors [37], [38]. (ii) Triangulate 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 rooflines via edge detection algorithms. (iv) Compute building height via camera projection with camera parameters, calibrated camera location, the height of building rooflines in the street scene image, and building footprints in the 2D map.

Our proposed model differs from [15] in two aspects: (1) In Step (ii) of their method, they calibrate camera location

by matching building corner lines (from the corner down to the ground) in the street scene image with building footprints on the 2D map. Such a method cannot handle images in dense/urban areas, where the corner lines of different buildings are too close to be differentiated, or when the buildings have non-uniform patterns/colors on their facades, which makes corner lines difficult to recognize. Our model uses building corners instead of corner lines, which puts more restrictions on the references for camera location calibration, and thus yields more accurate results. (2) In Step (iv) of their method, they use a local spectral histogram representation [39] as the edge indicator to capture building rooflines and then compute the building height, which can be ineffective when buildings overlap with each other. Our model uses RoofNet, a deep neural network that we designed to learn a latent representation of building rooflines for filtering invalid roofline candidates. Moreover, instead of only using street scene images with an orientation along the street to estimate the height of a building, we further use images facing buildings with an upward-looking view and learn the building height from two types of images.

III. OVERVIEW OF THE PROPOSED MODEL

We present the overall procedure of our proposed model in this section. To make the proposed model ready for large-scale deployment, we then present the algorithm to harvest street scene images automatically. We also summarize the process of camera projection, which is the theoretical foundation of building height estimation via street scene images.

A. Solution Overview

Given a 2D map such as a map segment from OpenStreetMap, we first retrieve all the locations where street scene images have been captured (e.g., the location of an image captured by a Google Street View car), which is detailed in Section III-B. For each such location, we obtain the corresponding street scene image using its geo-coordinates. The geo-coordinates that come with the image may be imprecise due to GPS errors. Google Street View image is an example of such images, and we aim to compute the height of each building in such images. As illustrated in Fig. 2, our model contains three stages:

- **Stage 1 – Preprocessing:** In the first stage, we preprocess an input image by recognizing the buildings and computing their sketches. There are many methods for these purposes. We use two existing models RefineNet [40], [41] and Structured Forest [42], [43] to identify the buildings and compute their sketches, respectively. After this step, the input image will be converted into a grayscale image that contains building sketches with each pixel valued from 0 to 255, which enables identifying building rooflines.
- **Stage 2 – Camera location calibration:** Before computing building height via 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 camera location, we detect building corner candidates in street scene

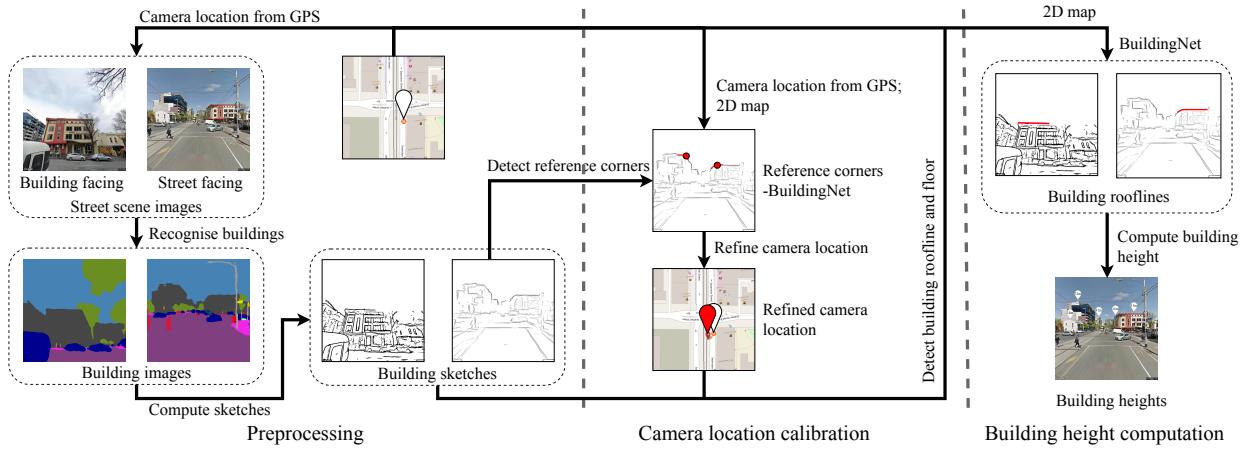


Fig. 2. Solution overview

images according to their footprints in 2D maps and their relative positions to the camera. Then, by comparing the locations and projected positions of (two) building corners in the image, we calibrate the camera location via camera projection. In this stage, we propose RoofNet to determine whether an image segment contains a valid building corner. RoofNet and the process of selecting two building corners for the calibration are detailed in Section IV.

Stage 3 – Building height computation: In this stage, we compute roofline candidates of each building from street scene images with an orientation along the street and those facing the building with an upward-looking view (a novel contribution of this journal extension). Among the roofline candidates detected, we filter out invalid ones via RoofNet and then rank all valid roofline candidates by an entropy-based ranking algorithm. The top-ranked roofline candidate is most likely to be the true roofline, based on which the building height is computed. The roofline and building height computation process above is detailed in Section V.

Since Stage 1 is relatively straightforward, we focus on Stages 2 and 3 in Sections IV and V, respectively. Before diving into these two stages, we discuss our algorithm to collect street scene images on a large scale (another contribution of this journal extension) and the idea of camera projection.

B. Large-scale Street Scene Image Collection

Street scene images (both street-orientation and upward-facing) are available from various sources such as Google Street View, Bing StreetSide, and Baidu Maps. We take Google Street View as an example to illustrate our image collection algorithm, which is summarized in Algorithm 1. Google Street View images are taken by Google Street View cars with fixed camera parameters. To download street scene images together with their camera geo-coordinates, we first identify the set S of all streets within a given rectangle M (line 1) such as the blue line segments in Fig. 3a; this involves a standard API function call in OpenStreetMap. For each $street$ in S , we traverse it with a step length of 3 meters – the typical distance between image captures in Street View – to obtain a set of sample points P (lines 2 and 3). For each point $p \in P$, we send a

Algorithm 1 Large-scale street scene image collection

```

Input: 2D map  $M$ 
Output: street scene images  $\mathcal{I}$ 
1:  $S \leftarrow \text{GetAllStreets}(M);$ 
2: for street  $s$  in  $S$  do
3:    $P \leftarrow \text{TraverseStreet}(s);$ 
4:   for point  $p$  in  $P$  do
5:      $o \leftarrow \text{GetNearestCameraLocation}(p);$ 
6:      $\langle l, r \rangle \leftarrow \text{GetTwoImagesAlongStreet}(o, s);$ 
7:      $\mathcal{I}.\text{add}(l);$ 
8:      $\mathcal{I}.\text{add}(r);$ 
9:     if  $\text{IsFacingBuilding}(o, M)$  then
10:       $F \leftarrow \text{GetImagesFacingBuilding}(o, M);$ 
11:      for image  $f$  in  $F$  do
12:         $\mathcal{I}.\text{add}(f);$ 
13: return  $\mathcal{I};$ 
```

request to the Google Maps API with its geo-coordinates. The API returns the geo-coordinates of location o (with a street scene image) that is the nearest to the requested point p (lines 4 and 5). The geo-coordinates of the camera location o may contain GPS errors, and we address this issue in Section IV.

At each location o as specified by the geo-coordinates returned by Google Maps API, we request two street scene images, one for each side of the street (line 6, cf. the red arrows in Fig. 3b). This is feasible as Google Maps API allows setting the camera orientation when requesting street scene images. Further, if o is in front of a building, we request one more image where the camera orientation is perpendicular to the building with an upward-looking view (line 9, cf. the blue arrows in Fig. 3b, detailed in Section V-B). In Fig. 3b, location o_1 is in front of both buildings B_1 and B_2 . We request one more image for both buildings B_1 and B_2 at o_1 . Location o_2 is in front of B_2 only. We only request one more image for B_2 at o_2 . All the images obtained, denoted by \mathcal{I} , will be used for roofline detection and building height estimation, which is detailed in Section V.

In our instances, the time required for image collection was small enough that we did not optimize the traversal order

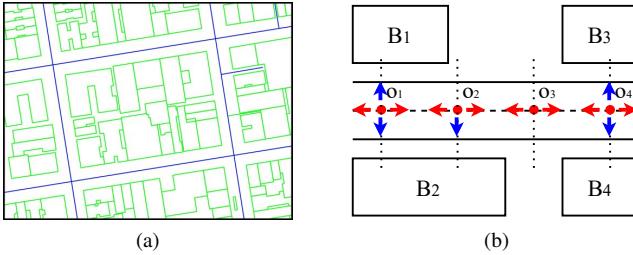


Fig. 3. (a) Streets (blue lines) and buildings (green lines) within a 2D map. (b) Camera locations from Google Maps API along a street.

of points. However, an order that preserves spatial locality, e.g., via space-filling curves, should be more efficient given caching and prefetching strategies employed by most maps API backends. Furthermore, while parallelism was not needed in our instances, this could be exploited to speed up image collection where a balanced partitioning of points on the map should yield the best performance.

C. Camera Projection

We use Fig. 4 to illustrate the idea of camera projection and the corresponding symbols. In Fig. 4, there are two coordinate systems, i.e., the camera coordinate system and the image plane coordinate system. Specifically, $\{o', x', y', z'\}$ represents 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, and plane xy is parallel to plane $x'y'$.

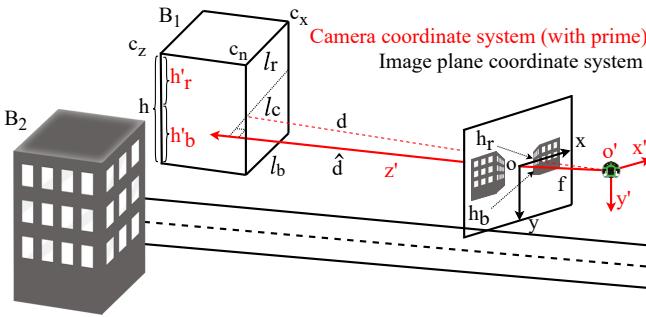


Fig. 4. Geometric variables in the camera and the image coordinate systems.

In Fig. 4, there are two buildings B_1 and B_2 projected onto the image. For each building, we use l_r , l_b , and l_c to represent the roofline, the floor, and the line on the building projected to the x -axis (centerline) 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). The height h of the building is the sum of the distance between l_r and l_c and the distance between l_c and l_b . 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 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 image, which can be regarded as a constant.

Let d be the horizontal distance from the camera location o' to corner c_n , \hat{d} be the projected length of d onto the z' -axis, 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 is:

$$h = h'_r + h'_b = h_r \cdot \hat{d} / f + h'_b \quad (1)$$

In this equation, the focal length f comes with the input image as its metadata. The distance \hat{d} is computed based on the geo-coordinates of the building and the camera, as well as the orientation of the camera from Google Street View. The geo-coordinates of the building are obtained from an open-sourced digital map, OpenStreetMap. The geo-coordinates and the camera orientation come with the input image from Google Street View. Due to GPS errors, we describe how to calibrate the camera location in Section IV. Due to GPS errors, we describe how to calibrate the camera location in Section IV, which takes a building height value h'_r computed based on the position of the roofline l_r , as discussion in Section V. The frequently used symbols in the rest of the discussion can be found in Appendix A.

IV. CAMERA LOCATION CALIBRATION

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

A. Key Idea

We use two building corners in a street scene image with known real-world locations for camera location calibration. To illustrate the process, we project Fig. 4 to a 2D plane, as shown in Fig. 5a, and assume that corners c_n of buildings B_1 and building B_2 are the two (closest) reference corners.

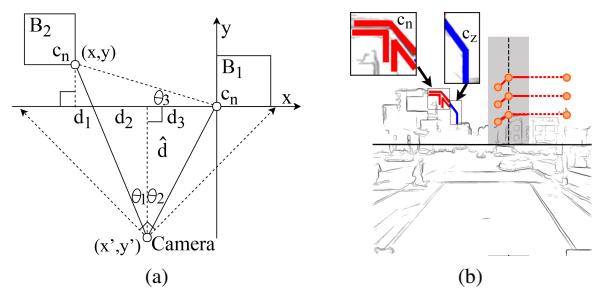


Fig. 5. (a) Geometric variables of Fig. 4 in plan-view. (b) The left building shows the formation of corner c_n and c_z , while the right building illustrates how to find corner candidates.

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 θ_1 and θ_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_3 is determined by the position of these two reference corners in the image. Let θ_3 be the angle between the line connecting corner c_n of B_1 and B_2 and the x -axis. This angle can be computed according to the camera's orientation and the relative locations of the two reference corners in 2D maps. Thus, we can compute the coordinates (x, y) of corner c_n of building B_2 in the coordinate system. With θ_1 , θ_2 , and the coordinate (x, y) , we compute the y -coordinate of the camera:

$$|y'| = \frac{|x| - |y| \cdot \tan\theta_1}{\tan\theta_1 + \tan\theta_2} \quad (2)$$

Further, the x -coordinate of the camera is $x' = y' \cdot \tan\theta_2$. Coordinates (x', y') yield the relative position of the camera to corner c_n of building B_1 . Thus, to compute (x', y') for camera location calibration, the key is to match two building corners with their positions in the image.

The real-world locations of building corners are obtained from 2D maps, and we need to locate their 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 3×4 camera projection matrix as follows:

$$\alpha \cdot p = [I|0_3] \begin{bmatrix} R_o & t \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} p' \\ 1 \end{bmatrix}, I = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

where a real-world point $p' = (x', y', z')^T$ is projected to its position $p = (x, y, 1)^T$ in the image plane; α is a scalar that translates pixel scale into millimeter scale [44]; $[I|0_3]$ is the camera matrix determined by focal length f , where 0_3 is $[0, 0, 0]^T$; R_o is the camera rotation matrix; and t is a 3-dimensional translation vector that describes the transformation from real-world coordinates to camera coordinates.

Since 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_r for each building to obtain the gradient of its rooflines, as shown in Fig. 5b. 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. 5b. For each building, we only consider its 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. 5b, 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 RoofNet described in Section IV-B to filter out non-corner image segments, and then select the two reference corners which are discussed in Section IV-C.

We assume the camera location error from Google Maps API to be less than three meters due to its camera location optimization [45]. If the camera location that we compute is more than three meters away from the one provided by Google Maps API, we use the camera location from Google Maps API.

B. RoofNet

We formulate building corner detection as an object classification problem [46], which first detects candidate corner

regions for a 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., corners c_n and c_z of left-hand side and right-hand side buildings ("L", "R", "T", and "B"), respectively. The last class corresponds to non-corner images that may contain any pattern except the above four types of corners (e.g., trees, lamps, or power lines), which should be filtered out. Such a classification problem is an *open-set classification* problem in the sense that the non-corner images do not have a unified pattern and will contain unseen patterns, whereas traditional deep neural networks may wrongly recognize it as one of the known classes. To solve this 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 it can classify all five classes in the testing phase. To enable such a classifier, we propose the *RoofNet* model based on LeNet-5 [47] architecture (which consists of 7 convolution layers for recognition of handwritten and machine-printed characters) plus our proposed triplet relative loss function. More details of the RoofNet architecture can be found in Appendix B. RoofNet 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.

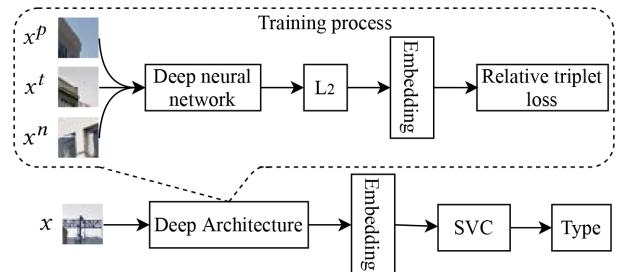


Fig. 6. RoofNet architecture: x^t , and x^p are corner images within the same type; x^n is an image of another type or non-corner; x is a testing image.

1) Triplet Relative Loss Function: As shown in Fig. 6, an input of RoofNet contains three images. Two of them (x^p and x^t) contain the same type of corner, and we name them the target (x^t) and a positive sample (x^p), respectively. The third image x^n contains another type of corners (or a non-corner if available), and we name it a negative sample. RoofNet learns to map its inputs to d_R -dimensional (a system parameter) embeddings based on our proposed *triplet relative loss* function inspired by the Triplet-Center Loss and the FaceNet [17], [48], [49], [50], which minimizes the distances within the same type of corners and maximizes the distances between different types of corners as:

$$\mathcal{L}_R = \sum_{i=1}^N (\beta \|f_R(x_i^t) - f_R(x_i^p)\|_2^2 + (1-\beta) \frac{\|f_R(x_i^t) - f_R(x_i^n)\|_2^2}{\|f_R(x_i^t) - f_R(x_i^n)\|_2^2}) \quad (4)$$

where $\beta \in [0, 1]$ is the weight of intra-class distance in the d_R -dimensional (Euclidean) embedding space; $(1 - \beta)$ is the weight of the ratio between intra-class distance and

inter-class distance, which aims to separate different classes in the embedding space; N is the cardinality of all input triplets. Function $f_R(\cdot)$ computes the d_R -dimensional embedding of an input image, and we normalize the embedding to $\|f_R(x)\|_2^2 = 1$. Different from existing loss functions based on triplet selection [51], [17], our *triplet relative loss* can minimise the intra-class distance and maximize the inter-class distance by means of their relative distance.

2) *Hard Triplet Selection*: Generating all possible image triplets for each batch during the training process will result in a large volume of unnecessary training data (e.g., x^t and x^p are too similar, while x^n is way different). It is crucial to select triplets that contribute more to the training phase. We assign a higher selection probability to such triplets and thus accelerating the convergence. The probability of selecting a negative sample x_i^n is:

$$p(x_i^n) = \frac{e^{m - \|f_R(x_i^n) - f_R(x^t)\|_2^2}}{\sum_{i=1}^{k_n} e^{m - \|f_R(x_i^n) - f_R(x^t)\|_2^2}}, i = [1, k_n]$$

$$m = \min\{\forall i \in [1, k_n] \|f_R(x_i^n) - f_R(x^t)\|_2^2 - \|f_R(x^t) - f_R(x^p)\|_2^2\}$$

where k_n is the total number of negative samples in a batch. After randomly choosing x^t and x^p for a triplet, we compute the Euclidean distance between (the embeddings of) x^t and x^p , as well as that between x^t and the k negative samples x^n in the batch. Let m be the minimum Euclidean distance between (the embeddings of) x^t and any x^n , and the Euclidean distance between x^t and x^p , which can be positive or negative. Then, a negative sample x_i^n that is more similar to x^t will have a higher probability to be selected.

After the training process, we obtain a d_R -dimensional embedding for each input image. We then learn a support vector classifier [52] based on these embeddings for corner region image classification.

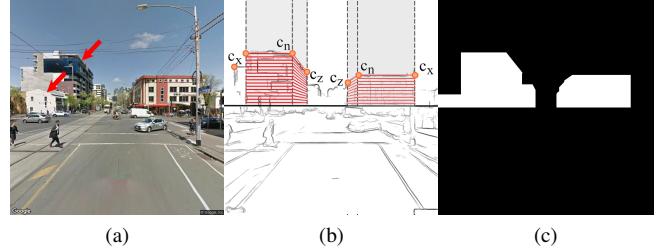
C. Entropy-based Ranking

RoofNet can filter out non-corner images. Among the remaining corner candidates, we select two corners as the reference corners. Reference corner selection relies on multiple factors: the length and edgeness (i.e., the thickness of a line, detailed in Section V) 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 consider the position of corners 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 yield higher accuracy when matching them to their footprints in 2D maps. Therefore, we score the corner candidates by:

$$[s_{c_1}, \dots, s_{c_k}]^T = \begin{bmatrix} \lambda_1, \omega_1, \tau_1, \rho_1, d_1 | c_1 \\ \vdots \\ \lambda_k, \omega_k, \tau_k, \rho_k, d_k | c_k \end{bmatrix} \cdot \begin{bmatrix} w_\lambda \\ \vdots \\ w_d \end{bmatrix} \quad (6)$$

where k is the number of corner candidates from all buildings; c_i is the i th corner candidate; s_{c_i} is the score of c_i ; λ_i is the

detected length of the two lines that form c_i , while ω_i is the sum of the edgeness of the two lines; τ_i is the number of other corner candidates of the same building with the same assumed height as c_i ; ρ_i is the minimum distance from c_i to a quarter or three-quarters of the image, and d_i is the distance from c_i to the camera. $w_\lambda, w_\omega, w_\tau, w_\rho, w_d$ are the weights of these parameters. Parameters $\lambda_i, \omega_i, \tau_i$ and ρ_i correlate with score s_{c_i} positively, while d_i correlates with s_{c_i} negatively.



(5) Fig. 7. (a) Two buildings (pointed to by the two red arrows) overlapping with each other. (b) A heuristic method for roofline candidate detection. (c) The mask of detected buildings.

We use an entropy-based ranking algorithm to compute the weights $(w_\lambda, \dots, w_d)^T$. Shannon entropy is a commonly used measurement of uncertainty in information theory [53]. The main idea of the entropy-based ranking is to compute the objective weights of different parameters according to their value distribution. In particular, if the values of a parameter among all samples vary greatly, then the parameter is considered an important one. Otherwise, the parameter is considered less important because it cannot provide much information for distinguishing different samples.

The input of entropy-based ranking is a decision matrix with the size of $n_s \times n_p$, where n_p is the number of indicators and n_s is the number of samples. For building corner classification, there are $n_p = 5$ indicators ($\lambda, \omega, \tau, \rho, d$) and $n_s = k$ samples. We denote the decision matrix as R , where R_{ij} is the value of the i th sample under the j th indicator. Before applying the entropy-based ranking algorithm, we preprocess R by Min-max scaling as follows:

$$R_{ij} = \begin{cases} (R_{ij} - \min_j(R_{ij})) / (\max_j(R_{ij}) - \min_j(R_{ij})), & \text{iff P} \\ (R_{ij} - \min_j(R_{ij})) / (\max_j(R_{ij}) - \min_j(R_{ij})) + 1, & \text{iff N} \end{cases} \quad (7)$$

where ‘P’ and ‘N’ denote that the j th indicator is positively and negatively correlated with score s , respectively. After scaling, the entropy of each indicator is computed as:

$$e_j = -\ln(n_s)^{-1} \cdot \sum_{j=1}^{n_s} R'_{ij} \cdot \ln(R'_{ij}), \quad R'_{ij} = R_{ij} / \sum_{j=1}^{n_s} R_{ij} \quad (8)$$

where R' is a normalized version of R . We then compute the weight of each indicator based on its entropy by:

$$w_j = (1 - e_j) / (n_p - \sum_{j=1}^{n_s} e_j), \quad j = [1, n_p] \quad (9)$$

After computing the weight of each indicator, we apply them to Equation 6 and rank all corner candidates by their scores to obtain the best two as the reference corners.

V. BUILDING HEIGHT ESTIMATION

Building height estimation requires detecting the roofline of each building. We first present an algorithm to detect rooflines from street scene images taken with an orientation along the street (*images facing the street* for short hereafter) in Section V-A. With the help of building corners, rooflines detected from such images often offer building height estimations with high accuracy. However, buildings in such images can overlap with each other (cf. Fig. 7a), where the rooflines may be shadowed. To address the overlapping issue, we detect rooflines from street scene images facing the buildings in Section V-B, which are less likely to have overlapping. We then present the method for learning the building height via the rooflines computed from both types of images in Section V-C. We also present a strategy to handle tall buildings (over 100 meters) in Section V-D.

A. Rooflines from Images Facing the Street

We first present the roofline detection algorithm for street scene images facing the street, followed by the corresponding roofline classification and ranking algorithm.

Algorithm 2 Roofline preprocessing

Input: buildings B , tree area T , edge map E

Output: updated buildings B

```

1:  $M = \text{null};$ 
2: for building  $b \in B$  do
3:   for roofline candidate  $l_{r_i}$  of  $b.l_r$  do
4:     // traverse all roofline candidates of  $b.l_r$ ;
5:      $l_{r_i}(\text{ini}) = l_{r_i};$ 
6:      $\lambda_{r_i}(\text{ini}) = \text{len}(l_{r_i});$ 
7:     for pixel  $p \in l_{r_i}$  do
8:       if  $\text{Mask}(p)$  then
9:          $l_{r_i}.\text{delete}(p);$ 
10:      for pixel  $p \in \hat{l}_r \cap p \notin l_r \cap !\text{Mask}(p)$  do
11:        if  $\text{Connected}(p, l_{r_i})$  and  $T(p)$  then
12:           $l_{r_i}.\text{add}(p);$ 
13:        // update the length and edgeness of  $l_{r_i};$ 
14:         $\lambda_i = \text{len}(l_{r_i});$ 
15:         $\omega_i = \lambda_i / \lambda_{r_i}(\text{ini}) \cdot E(l_{r_i}(\text{ini}));$ 
16:      ComputeScore( $b$ ); // compute the score of all roofline
        candidates of building  $b$  via Equation 11.
17:      ComputeHeight( $b$ );
18:       $M.\text{add}(b);$  // add the scope of building  $b$ .
19: return  $B;$ 

```

1) *Roofline Candidate Detection:* We consider the roofline 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 roofline from corner c_z to the corner next to c_z along the negative direction of the z' -axis in the camera coordinate system. The corner between corner c_z and corner c_x is corner c_n if they are adjacent to each other, as shown in Fig. 4, and we take this situation to simplify the explanation. Similar to corner candidate detection, as shown in Fig. 7b, we detect all roofline candidates of each building by a heuristic method,

which projects the rooflines of each building to the image 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 one corner (c_n , c_x , or c_z) is visible in the image. If corner c_n is visible, the maximum height computed via pinhole camera projection is:

$$h_r = \hat{d} \cdot (h_I/2f) \quad (10)$$

where h_I is the height of the street scene image; \hat{d} is the distance from corner c_n to the camera projected to the z' -axis of the camera coordinate system. If corner c_n is invisible, we use c_z as the reference corner when computing the maximum height of a building in the same way. Using the maximum height of the building, we compute the position of corner c_n , c_x , and c_z in the image. We then apply Hough transform to the input building sketches in Fig. 2 to detect roofline candidates. The roofline candidates from c_n to c_x need to match the computed position of c_n and c_x . Similarly, the roofline candidates from c_n to c_z need to match the computed position of c_n and c_z . Instead of using the typical Hough transform for line detection, which takes binarized images as the input, each pixel in our algorithm has a value between 0 and 255 and we sum the value of all pixels within a line as its weight. We name the summed value as **edgeness** of a roofline candidate, which reflects the visibility of a line in the edge map. We iteratively reduce the assumed height with a step length of 0.5 meters until $h'_r = 0$ and collect all candidates.

Similar to reference corner detection, we formulate the roofline detection as open-set classification and ranking.

2) *Roofline Candidate Classification and Ranking:* There are three types of rooflines: (i) Roofline from c_n to c_x ; (ii) Roofline from c_n to c_z of the left hand side buildings; (iii) Roofline from c_n to c_z of the right hand side buildings, as shown in Fig. 5b. We use RoofNet to filter these candidates and find the valid roofline candidates, which is similar to the corner candidate validation process described in Section IV-B. For each roofline l_r , we then weight each of its candidate l_{r_i} by its detected length λ_i , edgeness ω_i , and the number of corners τ_i of the same assumed height from the same building. We rank all valid roofline candidates via the entropy-based ranking algorithm in Section IV-C as:

$$[s_{l_{r_1}}, \dots, s_{l_{r_k}}]^T = \begin{bmatrix} \lambda_1, \omega_1, \tau_1 | l_{r_1} \\ \vdots \\ \lambda_k, \omega_k, \tau_k | l_{r_k} \end{bmatrix} \cdot \begin{bmatrix} w_\lambda | l_r \\ w_\omega | l_r \\ w_\tau | l_r \end{bmatrix} \quad (11)$$

where k is the number of roofline candidates; l_{r_i} is the i th roofline candidate; $s_{l_{r_i}}$ is the score of the i th roofline candidate. $w_\lambda | l_r$, $w_\omega | l_r$ and $w_\tau | l_r$ are the weights of these parameters based on all candidates of l_r , and all three parameters are positively correlated with score s . The value of τ_i depends on the number of corner candidates of c_n , c_x , and c_z of the same assumed height as l_r , i.e., $\tau_i \in \{0, 1, 2, 3\}$.

Different from building corners, which can only be either visible or invisible, rooflines can be partially blocked by other objects (e.g., trees). Therefore, before applying the ranking algorithm, we preprocess the length λ_i and the edgeness ω_i of

a roofline candidate l_r which may be affected by the blocking via Algorithm 2, where buildings are ordered by whether they have a detectable corner, and then their distance to the camera.

When estimating building height, we first separate buildings into two classes: (i) buildings with at least one valid corner candidate, which we have a higher confidence in detecting its true outline. (ii) buildings without any valid corner candidate. Then, we process buildings in Class i according to their distances 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. 7c (i.e., when its height has been obtained). For each detected candidate l_{r_i} of roofline l_r of a building, we refine l_{r_i} by (i) removing each pixel $p \in l_{r_i}$ that falls outside the building scope (lines 7 to 9 of Algorithm 2) and (ii) extending (i.e., adding pixels to) l_{r_i} to \hat{l}_{r_i} until \hat{l}_{r_i} reaches the boundary of the building (lines 10 to 12). This refinement process filters false positive pixels and adds back pixels that might have been blocked by trees. We then use the updated \hat{l}_{r_i} to compute its length λ_i (line 14) and its edgeness ω_i (line 15). Formally,

$$\begin{aligned} \lambda_i &= \lambda_{i(ini)} - \sum_{p \in l_{r_i}} \text{Mask}(p) + \sum_{p \in \hat{l}_{r_i}} \text{T}(p) \\ \omega_i &= \lambda_i / \lambda_{r_i(ini)} \cdot E(l_{r_i(ini)}) \\ &= (\lambda_i / \lambda_{i(ini)}) \cdot \sum_{p \in l_{r_i}} E(p) \cdot (1 - \text{Mask}(p)) \end{aligned} \quad (12)$$

where $\lambda_{i(ini)}$ denotes the initially detected length of l_{r_i} . Function $\text{Mask}(p)$ checks whether a pixel p within l_{r_i} goes beyond the building's scope. Function $\text{T}(p)$ checks whether a pixel p , which is in the extended line \hat{l}_{r_i} , is blocked by trees. If there exists such pixels and they connect to the detected roofline segment, we add them to l_{r_i} . Function E calculates the number of pixels on a line to overlap with the detected building sketches (cf. Fig. 2).

B. Rooflines from Images Facing the Building

Next, we present an algorithm to detect rooflines from images facing a building with an upward-looking view. Such images are taken close to a building. Buildings in such images are slanted and maybe only partially visible, which pose new challenges in roofline detection.

1) *Roofline Candidate Detection:* To obtain roofline candidates from images facing the buildings, we fetch street scene images by setting the camera with an upward-looking view to reach a building's roof. The default upward-looking angle is 25 degrees, as shown in Fig. 8a. If this fails to capture the full building (as flagged by the building recognizer shown in Fig. 2), we then set the upward-looking angle as 45 degrees. Google's devices capture a view range of 90 degrees. Thus, an upward-looking angle of 45 degrees (Fig. 8a), can reach the roof of buildings of any height. Given images with an upward-looking view, the roofline is shrunk, and the building corner lines are slanted (Fig. 8b). For roofline candidate detection in such images, we only compute the roofline candidates that connect c_n and c_z because c_x is invisible.

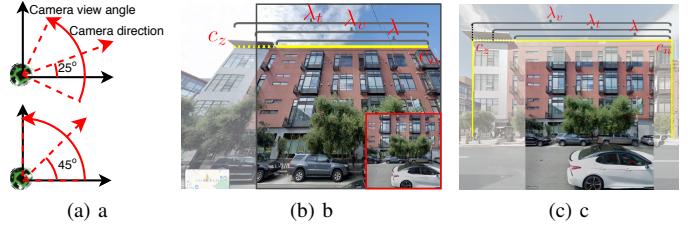


Fig. 8. (a) Camera upward-looking views of 25 degrees and 45 degrees. (b) A street scene image with an upward-looking view of 25 degrees facing the building, where the yellow solid line denotes a candidate of the roofline. The red box at the right bottom denotes the street scene image with the same camera location but with a horizontal view, where the building's roof is invisible. (c) The corresponding image of Fig. 8b after being transformed to a horizontal view.

As discussed in Section V-A1, roofline candidate detection is based on an assumed height h'_r of the building. The pinhole camera projection defined by Equation 3 is inapplicable for a slanted building directly. Therefore, we first compute a plane-to-plane homography [54], which maps an image of an upward-looking view to an image of a horizontal view, as shown in Fig. 8c. Here, we use the homogeneous estimation method [55], which solves a 3×3 homogeneous matrix H that matches a point in an upward-looking image to a horizontal view image as:

$$A \cdot H = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 X_1 & -y_1 X_1 & -X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 Y_1 & -y_1 Y_1 & -Y_1 \\ \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n X_n & -y_n X_n & -X_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_n Y_n & -y_n Y_n & -Y_n \end{bmatrix} \cdot H = 0 \quad (13)$$

where the homogeneous matrix H ($|H| = 1$) is represented in vector form as $H = (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T$; (X_i, Y_i) represents a point in the upward-looking image, and (x_i, y_i) represents the corresponding point in the resultant image of a horizontal view. Parameter n denotes the number of point pairs $((X_i, Y_i)$ and (x_i, y_i)), and $n \geq 4$; By linear algebra, the vector H that minimizes the algebraic residuals $|AH|$, subject to $|H| = 1$, is given by the eigenvector of least eigenvalue of $A \cdot A^T$.

Using the homogeneous matrix H , we adapt the roofline candidate detection method in Section V-A1 as follows. We first transform the image into a horizontal view (cf. Fig. 8c), and we treat the height h'_r of a building to be 0 in the image, which is the same as that of Google's device (e.g., 3 meters for the camera on a Google Street View car). We then compute the roofline's position as detailed in Section V-A1. Next, we use the homogeneous matrix H to map the roofline's position back to that in the original image with an upward-looking view. Similar to Section V-A1, we iteratively increase the assumed height with a step length of h_c / \hat{d} meters until the roofline reaches the sky and collects all roofline candidates. Here, h_c denotes the currently assumed height of the building. Note that when h_c increases, even though the step length in the transformed image is a constant, that in the original image is increasing. This is because, when the iterative process gets closer to the top of the image, each pixel in the transformed image represents a larger physical distance (Fig. 8b).

2) *Roofline Candidate Ranking* : In images facing the building, the line candidates become less visible (thinner)

with the increase of buildings' height, since the buildings are slanted in such images. The entropy-based ranking in Section IV-C does not yield strong results in this case. To address this issue, based on the roofline candidates from c_n to c_z for each building, we adapt the entropy-based ranking in Section IV-C for roofline candidate ranking as follows.

As shown in Fig. 8b, for each candidate l_{r_i} , we again denote its detected length as λ_i (yellow solid line), edgeness as ω_i . We further consider the visible length of the building's roofline according to the assumed height h'_r of the candidate, denoted as λ_{vi} (yellow solid line plus the extended dashed line to the left boundary of the image). We denote the true length of the building's roofline computed according to the assumed height of the candidate as λ_{ti} (yellow solid line plus the extended dashed line that exceeds the image), and the proportion of λ_{vi} close to the sky as λ_{si} (λ_{si} equals to λ_{vi} in Fig. 8 because all pixels in line λ_{vi} are connecting to the sky area). We rank all candidates by computing their scores:

$$[s_{l_{r_1}}, \dots, s_{l_{r_g}}]^T = \begin{bmatrix} \omega_1/\lambda_1, \lambda_1/\lambda_{v1}, \lambda_{v1}/\lambda_{t1}, \lambda_{s1}|l_{r_1} \\ \vdots \\ \omega_g/\lambda_g, \lambda_g/\lambda_{vg}, \lambda_{vg}/\lambda_{tg}, \lambda_{sg}|l_{r_g} \end{bmatrix} \begin{bmatrix} w_{\omega/\lambda}|l_r \\ \vdots \\ w_{\lambda_s}|l_r \end{bmatrix} \quad (14)$$

Here, g is the number of candidates; l_{r_i} is the i th roofline candidate; $s_{l_{r_i}}$ is the score of the i th roofline candidate; ω_i/λ_i denotes the averaged edgeness; λ_i/λ_{vi} denotes the ratio of the detected length over the visible length of the roofline in the image (fully detected rooflines are preferred), and $\lambda_{vi}/\lambda_{ti}$ denotes the ratio of the true roofline visible in the street scene image (a larger value denotes a more fully visible roofline candidate which is preferred). $w_{\omega/\lambda}|l_{r_i}$, ..., $w_{\lambda_s}|l_{r_i}$ are the weights of these parameters based on all candidates of l_r , and they are positively correlated with score $s_{l_{r_i}}$.

C. Estimating Building Height from Rooflines

After obtaining the roofline candidates from both images facing the street and facing the building, we compute a set of heights for each building according to Equation 1. We denote the heights from images facing the street as $H_a = h_{a_1}, h_{a_2}, \dots, h_{a_k}$ and those from images facing the building as $H_f = h_{f_1}, h_{f_2}, \dots, h_{f_g}$.

The heights in H_a tend to be close to the true building height when a true roofline has been detected. This is because the camera location of images facing the street is far away from the building, and hence Equation 1 is less sensitive to the GPS errors of the camera location. However, as discussed earlier, buildings in such images may overlap with other buildings, and the true roofline may be shadowed and hence some heights in H_a are far from the true building height. On the other hand, an image facing the building is more likely to capture the true roofline as there are fewer overlaps. Thus, H_f is more likely to contain heights computed from the true roofline. However, a limitation is that they are computed based on a camera location that is closer to the building, where Equation 1 is more sensitive to GPS errors. Thus, these heights may be less accurate when the buildings are high.

To obtain an accurate building height estimation from H_a and H_f , we merge the two sets of heights into a single set

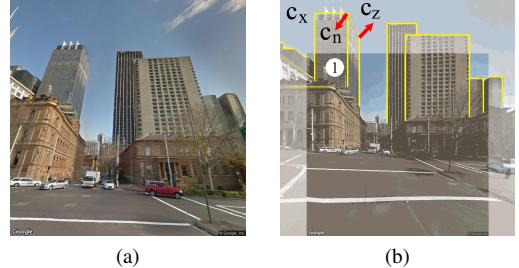


Fig. 9. (a) An image with an upward-looking view. (b) The corresponding image with a horizontal view.

\mathcal{H} . Compared with the distribution center of H_a , that of \mathcal{H} is expected to be closer to the real height due to the inclusion of H_f . Let the median of \mathcal{H} be h_m . We check whether there is a value $h_{f_i} \in H_f$ that is close to h_m . If $|h_{f_i} - h_m| \leq 3 \cdot h_m/\hat{d}$, we say that h_{f_i} and h_m are close. The value 3 here is the upper bound of GPS errors from Google's devices as assumed in Section IV-A after optimization. We weigh it by h_m/\hat{d} , which denotes the error of the height estimation for a building (if the ground truth is h_m) with a 3-meter GPS error.

If there is a $h_{f_i} \in H_f$ that is close to h_m , we return h_m as the final height estimation result. Otherwise, we return the height $h_{a_j} \in H_a$ that falls between h_m and its closest value $h_{f_i} \in H_f$. If there are multiple values in H_a that satisfy this requirement, we return the one that is the closest to h_{f_i} .

We note that advanced algorithms such as Variational Bayes Inference [56], [57] may offer a more accurate estimation given a large set of heights. We did not use such algorithms because our set \mathcal{H} is relatively small ($|\mathcal{H}| < 20$ in most cases) due to the limited availability of street scene images of the same building. Our experiments show that the heuristic algorithm above yields highly accurate height estimations with a small set of \mathcal{H} .

D. Tall Building Preprocessing

For tall buildings, the camera needs to be placed far away with an upward-looking view to capture the building roofline. Also, if the camera is too close to the building, the computed building height is highly sensitive to the GPS error, especially for tall buildings. Therefore, we capture the street scene images 250 meters away from the buildings facing the street with an upward-looking view via Google Maps API. Fig. 9a presents an example of the street scene image for tall building height estimation.

Similar to Section V-B1, with a tall building's footprint in a 2D map and the camera location of a street scene image that contains the tall building, we first transform the image to one that has a horizontal view (Fig. 9b). We then compute the building corners' positions c_n , c_x , and c_z , as those of building ① shown in Fig. 9b, where the corners' positions may exceed the image. Based on the upward-looking angle, we apply the homogeneous matrix H and map the computed corners' positions of a tall building back to the original image with an upward-looking view (Fig. 9a). We then apply the building height estimation algorithm for images facing the street as described in Section V-A to compute the building

height. We do not apply the height estimation algorithm in Section V-B because tall buildings are usually fully visible in the street-facing images already due to their large distances to the camera location.

E. Algorithm Cost Analysis

Our proposed method has four main steps: image segmentation, building sketch computation, camera location calibration, and building height estimation.

- The image segmentation step applies a batch-processing strategy (process multiple street scene images at once) using RestNet [58]. We use the number of parameters to represent its running cost, which is around 120 million. The running time of this step is 8.12 seconds per image on average under our experimental setting (detailed in Section VI, same below) using a laptop computer with a CPU, which can be boosted by using GPU machines.
- The building sketch computation step uses a structured forest algorithm [42], [43], which takes $O(I_w I_h \mathcal{M} \mathcal{D})$ time, where I_w and I_h denote the width and the height (in terms of the number of pixels) of an input image, while \mathcal{M} and \mathcal{D} denote the number of trees and the depth of the trees, respectively. This step takes 0.81 seconds for each image on average.
- The camera location calibration step detects potential building corners from image segments within a street scene image, classifies different types of corners, and calibrates camera location via camera projection (which has a constant time cost). Building corners are formed by two line segments detected via the Hough transform (we sample \mathcal{S}_h points only) with an $O(I_h \mathcal{S}_w \mathcal{S}_h)$ time cost, where \mathcal{S}_w and \mathcal{S}_h are the width and the height of the sub-image that we use to compute the rooflines. Corner classification via RoofNet is based on LeNet-5 with 60 thousand parameters. This step takes 1.18 seconds for each image on average.
- The building height estimation step detects roofline candidates for each building based on the calibrated camera location. For this step, roofline detection via the Hough transform takes the same $O(I_h \mathcal{S}_w \mathcal{S}_h)$ time as above. Roofline classification via RoofNet is also based on LeNet-5 with 60 thousand parameters. The roofline preprocessing (Algorithm 2) and the Entropy-based roofline candidate ranking steps have lower time complexity than the Hough transform. Their time costs are absorbed in the big- O notation. This step takes 4.23 seconds for each image on average.

For comparison, we also present the time costs of the state-of-the-art street view image based building height estimation algorithm [15] (i.e., the baseline algorithm in Section VI-D). This algorithm has two main steps, i.e., camera location calibration and building height estimation. The camera location calibration is based on the spectral histogram algorithm, which has an $O(I_h \mathcal{S}_w \mathcal{S}_h)$ time cost. The height estimation is based on a voting strategy, which has an $O(K \mathcal{S}_h^2)$ time cost, where K denotes the number of candidate camera locations computed for each image.

We summarize the time costs of our method and that of [15] in Table I. Empirically, the running time of our method

TABLE I
ALGORITHM TIME COST ANALYSIS

Steps	Ours		Baseline	
	Complexity	Time	Complexity	Time
Image segmentation	120 M	8.12 s	N/A	N/A
Compute sketches	$O(I_w I_h \mathcal{M} \mathcal{D})$	0.81 s	N/A	N/A
Camera calibration	$O(I_h \mathcal{S}_w \mathcal{S}_h)$	1.18 s	$O(I_h \mathcal{S}_w \mathcal{S}_h)$	2.53 s
Height estimation	$O(I_h \mathcal{S}_w \mathcal{S}_h)$	4.23 s	$O(K \mathcal{S}_h^2)$	0.41 s

to process an image is higher. Our method takes $8.12 + 0.81 + 1.18 + 4.23 = 14.34$ seconds on average to process an image, while the method by [15] takes $2.53 + 0.41 = 2.94$ seconds (based on our implementation, as their source code is unavailable). The higher running time is expected, since we consider both building corners and rooflines while [15] only considers rooflines. As will be shown next, we argue that our running time cost is worthwhile, for the consistently (and up to 11.9%) higher building height estimation accuracy obtained by our method.

VI. EXPERIMENTS

In this section, we first show the scalability of our image collection process. Then, we evaluate our RoofNet model for building corner and roofline classification. We also evaluate our model for building height estimation, followed by an error analysis, and a parameter sensitivity analysis.

A. Data Sets

We obtain building footprints (latitude and longitude coordinates) from OpenStreetMap and street scene images from Google Street View, which have been post-processed by Google to share the same parameters (e.g., focal length), respectively. There may be small GPS errors in the coordinates obtained from OpenStreetMap, which may lead to a horizontal shift (deviation) of all the corners of a building. However, this horizontal shift is small because all buildings are aligned by OpenStreetMap based on their relative positions in the real world. Also, some of the horizontal shifts can be addressed by our camera location calibration step. We evaluated our proposed method on the following two data sets:

(i) **City Blocks**, which contains two blocks (from a lot and block survey system) in San Francisco (SF) and one in New York (NY). The two blocks in SF contain 59 and 69 buildings, respectively, with the first city block used in the baseline work [15]. For the blocks in SF, we manually obtained the building height ground truth from high-resolution aerial maps from NearMap [59] by computing the elevation difference between the ground and the building roofline. The block in NY contains 954 buildings, and the building height ground truth is obtained from NYC Open Data¹. We collected all 640×640 -pixel street scene images where the camera has an orientation along the street horizontally or is facing a building with an upward-looking view. The focal length of the camera can be derived via the parameters provided by Google. We do not need to consider the camera rotation matrix R_o or

¹<https://opendata.cityofnewyork.us/>

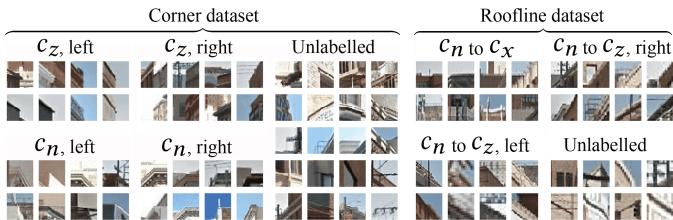


Fig. 10. Examples of four types of corners, three types of rooflines, and the corresponding unlabeled images.

the translation vector t (cf. Equation 3) due to the image preprocessing of Google Street View.

(ii), which contains 37 buildings taller than 100 meters in SF, Melbourne, and Sydney collected by us via Google Maps API. We set the camera orientation along the street with an upward-looking view (25 degrees) to capture their rooflines. The ground truth of building height comes from Wikipedia or is derived from aerial maps.

For building corner classification, we crop images from the City Blocks data set (SF in specific). We generate the corner data set semi-automatically, where we crop 28×28 -pixel image segments from the street scene images. We then manually label whether each image segment contains a building corner and if so, label its corner type. The training data set contains 10,400 image segments, including 1,300 images of each type of building corner (a total of 5,200 corner images) and 5,200 randomly chosen non-corner images. The test data set contains 1,280 images, including 160 images for each type of building corner and 640 randomly chosen non-corner images. There is no overlap between the training and the testing sets.

We follow a similar approach for roofline detection. For each roofline candidate, we extend the upper and lower 10 pixels of the roofline to obtain $21 \times \lambda$ image segments, where λ is the length of the roofline. We then rotate each image if the roofline is not horizontal, and resize them to 28×28 to generate same-size inputs for RoofNet. The training set includes 7,800 image segments from the City Blocks data set (SF in specific), including 1,300 images for each type of roofline (a total of 3,900 roofline images) and 3,900 randomly chosen non-roofline images. The testing set contains 960 images, including 160 images for each type of roofline and 480 randomly chosen non-roofline images. There is no overlap between the training and the testing sets. Fig. 10 provides examples of corner and roofline images.

B. Efficiency of Large-Scale Image Collection

We implement the street scene image collection algorithm described in Section III-B with MATLAB and run it on a laptop computer with a 2.4 GHz Intel Core i5 CPU and 16 GB memory. To showcase the algorithm efficiency and scalability, we download a 2D map for a region in SF from OpenStreetMap. This region covers an area of $250,912 \text{ m}^2$, and we extract all the streets within the region. Based on the streets in this region, we obtain 1,522 street scene images, including both images facing streets and those facing buildings. It takes only 24 seconds to process the map data and obtain the

parameters for downloading images via the Google Maps API. The time for downloading the street scene images is determined by the network speed and the 500-image-per-second limit of Google Maps API. For us, the download time is 99.8 seconds for the 1,522 images (65.6 milliseconds per image). After downloading the images, we send them to our building height estimation algorithm.

C. Effectiveness of RoofNet

Building corner and roofline classification is an open-set classification problem where the non-corner images and the non-roofline images do not have consistent patterns. To benchmark the effectiveness of RoofNet, we use four open-set classifiers as baselines: *SROSR* [60], *OpenMax* [61], *CGDL* [19], and *AMPF++* [62]. *SROSR* handles the open-set classification problem by computing the confidence score for an input to come from known classes based on *reconstruction errors*, where an input is expected to have a low reconstruction error on the true class it comes from. *OpenMax* handles the open-set classification problem by estimating the probability of whether an input comes from unknown classes based on the output of the last fully connected layer of a neural network. *CGDL* uses a conditional Gaussian distribution learning model based on a variational auto-encoder (VAE) for open-set classification. It generates class conditional posterior distributions, where inputs of known classes tend to follow one of the prior distributions and inputs of unknown classes tend to be in lower probability regions. *AMPF++* generates adversarial samples that try to confuse a deep neural network-based classifier and use such samples to train a more robust classifier to differentiate images from known and unknown classes.

To show the effectiveness of our proposed triplet-based loss function, we further compare RoofNet using our proposed loss function with those using the loss functions of *FaceNet* [17] and *MSML* [63], respectively. The original loss function in FaceNet makes the intra-class distance smaller than the inter-class distance by adding a margin, and the one in MSML optimizes the triplet selection process towards selecting hard triplets in the training phase.

TABLE II
EFFECTIVENESS OF ROOFNET

Models	Corner data set				Roofline data set			
	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
CGDL	0.702	0.551	0.509	0.476	0.612	0.495	0.476	0.454
AMPF++	0.729	0.601	0.602	0.601	0.777	0.641	0.594	0.616
SROSR	0.870	0.781	0.787	0.783	0.732	0.597	0.709	0.557
OpenMax	0.876	0.802	0.799	0.798	0.846	0.768	0.812	0.776
FaceNet	0.875	0.799	0.804	0.799	0.828	0.742	0.778	0.750
MSML	0.883	0.812	0.819	0.813	0.836	0.754	0.820	0.765
RoofNet _L	0.887	0.820	0.827	0.818	0.856	0.782	0.812	0.781
RoofNet _U	0.943	0.911	0.912	0.911	0.966	0.928	0.957	0.940

Hyperparameters. For OpenMax, we first pre-train the LeNet-5 model on the MNIST data set. We then train the model on the building corner and roofline data sets and apply the last fully-connected layer to OpenMax for classification.

For CGDL, SROSR, and AMPF++, we use the default settings of their released code.

For our RoofNet, we implement two variants: RoofNet_L and RoofNet_U. Both RoofNet_L and RoofNet_U use LeNet-5 which is first pre-trained on the MNIST data set and then fine-tuned on our building corner and roofline data sets. The difference between the two model variants is that in RoofNet_L, all MNIST data are considered labeled in the pre-training phase, while in RoofNet_U, images in 5 classes are considered labeled while the images in the other 5 classes are considered unlabeled, i.e., from known unknown [64]. For FaceNet and MSML, we apply their loss functions on RoofNet_L. We set the learning rate as 0.1 with the decay rate of 0.95 after every 1,000 iterations. The batch size is 30 images for each class, the embeddings that RoofNet_L and RoofNet_U learn are 128-dimensional, and β in the triplet relative loss function is 0.5.

We perform a 10-fold cross-validation on the models and report the averaged accuracy, precision, recall, and F_1 score of the models, which are summarized in Table II.

On the corner data set, RoofNet achieves the best performance on all four metrics. Compared with CGDL, AMPF++, SROSR, and OpenMax, RoofNet_L improves the accuracy and F_1 score by more than 1.1% and 2%, respectively. When unlabeled data are available at training (RoofNet_U), the advantage of RoofNet is even stronger, i.e., 6% and 10% on the accuracy and F_1 score, respectively. A similar observation is made on the roofline data set, where both RoofNet_L and RoofNet_U outperform the baseline open-set classifiers consistently. Moreover, our proposed triplet relative loss function achieves state-of-the-art performance on both corner and roofline data sets compared with the loss functions from MSML and FaceNet. On the corner data set, our loss function improves the accuracy and F_1 score by more than 0.4% and 0.5%, respectively. On the roofline data set, our loss function achieves the best performance on all evaluation metrics except the recall.

To further illustrate the effectiveness of our triplet relative loss function, we visualize the generated embeddings of different loss functions on the corner data set in Fig. 11. Compared with random triplet selection with margin (FaceNet) and hard triplet selection with margin (MSML), our triplet relative loss function obtains better classification results with smaller average intra-class distance and larger average inter-class distance.

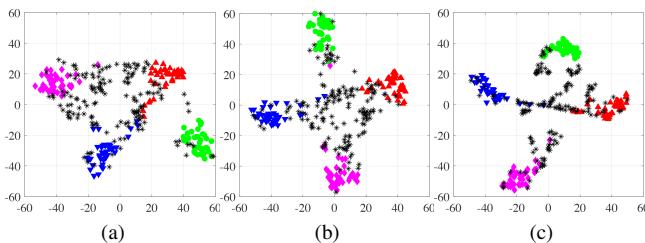


Fig. 11. t-SNE [65] 2D embeddings of four types of corners (colored dots) and the unlabeled data (black dots) after 100 epochs, learned by loss functions of (a) FaceNet, (b) MSML, and (c) the proposed triplet relative loss.

D. Effectiveness of Building Height Estimation

Next, we evaluate the performance of our overall building height estimation method.

1) Building Height Estimation for City Blocks: Results on the full data sets. Fig. 12 shows the building height estimation accuracy, at a variety of error tolerances, over the three city blocks in the City Blocks data set for (a) the existing state-of-the-art method from [15] that uses street scene images only (denoted by “Baseline”); (b) our previous method CBHE [18] that uses only images facing the street (i.e., the median of the heights estimated based on such images, denoted by “Ours-CBHE”), which is the same as those used by [15]; and (c) our full method that uses both images facing the street and images facing the building, as described in Section V-C, denoted by “Ours-full”).

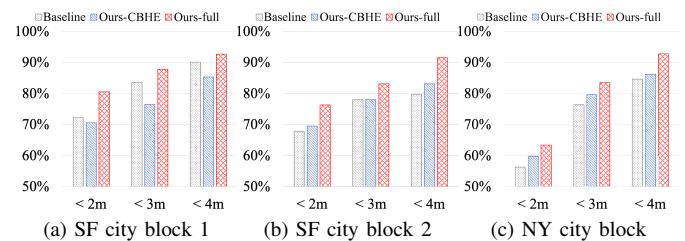


Fig. 12. The accuracy of the baseline method [15], our previous proposed method CBHE [18] denoted as Ours-CBHE, and our full model denoted as Ours-full on three city blocks.

On SF city block 1, Ours-full method has 8.3%, 4.2%, and 2.6% more buildings with height estimation errors within 2, 3, and 4 meters, respectively (cf. Fig. 12a). This confirms the superiority of our proposed method. For Ours-CBHE, we see that using only images facing the street does not yield as strong results as those using both types of images (i.e., Ours-full). This confirms the importance of considering the building-facing images.

We note that Our-CBHE is even worse than the baseline on SF city block 1. This is because the results of the baseline are from [15], while the results of our models are obtained on images newly crawled from the same block as that used by the baseline. The new street scene images for these blocks have become more complex over time, as the buildings have become more occluded due to trees in the images growing larger (cf. Fig. 13)². This makes the building height estimation more challenging.



Fig. 13. Street scene images at the same spots.

²For the same reason, the results of Ours-CBHE reported here are a bit worse than those in our previous conference version [18], which used images collected two years ago.

Fig. 12b shows the result in SF city block 2 (which was not used by [15]). As we are unable to obtain the source code of the baseline method, the result is based on our implementation of their method. Ours-full outperforms Baseline by 11.9%, 5.1%, and 8.5% on height estimation with errors less than 2, 3, and 4 meters, respectively. Also, Ours-CBHE is outperformed by Ours-full, but it is still better than Baseline overall.

Fig. 12c shows the result on the NY city block. Like on SF city block 2, the results here are based on our implementation of all three methods tested. Our full method again outperforms the baseline by 7.2%, 7.1%, and 8.2% on height estimation with errors less than 2, 3, and 4 meters. For this city block, both proposed variants outperform the baseline. Note that the performance of both the baseline and our methods drop on this city block with an error of less than 2 meters. This is because the ground truth of this data set from NYC Open Data is computed by the elevation difference between the ground and the highest point of the building, which is higher than the height of the roofline for most buildings.

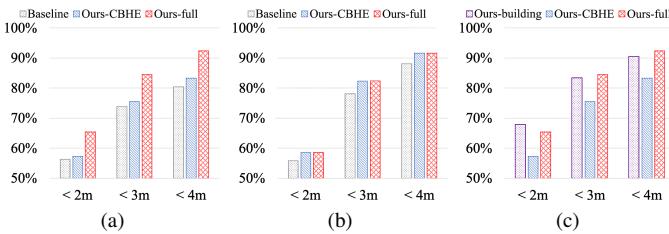


Fig. 14. Accuracy of the baseline method from [15], our previous proposed method from [18] denoted as Ours-CBHE, and our full model denoted as Ours-full on the NY city block for (a) buildings with height estimations from both images facing the street and facing the building, and (b) buildings with height estimation from images facing the street only. (c) accuracy of Ours-building, Ours-CBHE and Ours-full on the NY city block (with only buildings that have estimated heights from both images facing the street and images facing the building).

Results on buildings with both street-facing and building-facing images. In the experiments with the full data sets above, not all buildings have been captured in both street-facing and building-facing images. To show the full benefit of our method using both types of images, we further report results for only the buildings with both types of images, in Fig. 14a. In this case, Ours-full outperforms the baseline by 9.4%, 9.3%, and 12.2% with height estimation errors less than 2, 3, and 4 meters. It also outperforms Ours-CBHE by 8.0%, 7.6%, and 9.7% on height estimation with errors less than 2, 3, and 4 meters, respectively. These larger performance gaps (compared with those reported in Fig. 12c) confirm the strong advantage of our proposed model when both types of images are available for building height estimation.

Note that here we only report results on the NY city block as it has a larger number (954) of buildings, and 49.6% of the buildings in this data set have both types of images. The two SF city blocks only have 59 and 69 buildings, respectively. Comparing the methods on subsets of these data sets did not yield meaningful results.

Results on buildings with street-facing images only. To complement the results above, we also report results for the buildings with street-facing images only (i.e., the other 50.4%

of the buildings in the NY city block), in Fig. 14b. Now Ours-full degenerates to Ours-CBHE, and both methods produce the same results. Even in this case, they both outperform the baseline by 2.7%, 3.9%, and 2.5% on height estimation with errors less than 2, 3, and 4 meters, respectively, confirming the effectiveness of using building corners on top of rooflines for building height estimation.

2) Ablation Study.: To show the performance gains of our proposed method contributed by images facing the street and images facing the buildings, respectively, we run further experiments on the subset of buildings from the NY city block with both types of images (486 out of 954). We summarize the results of our method running on both types of images (denoted as “Ours-full”), our method running on images facing the *street only* (denoted as “Ours-CBHE”, Section V-A), and our method running on images facing the *building only* (denoted as “Ours-building”, Section V-B), in Fig. 14c. In general, Ours-full achieves the best performance compared with Ours-CBHE and Ours-building.

Ours-full outperforms Ours-CBHE by 8.0%, 7.6%, and 9.7% on height estimation with errors less than 2, 3, and 4 meters, respectively, and it outperforms Ours-building by 0.8% and 2.1% on height estimation with errors less than 3 and 4 meters, respectively. These results confirm the importance of using both types of images together for height estimation. Ours-building has even the highest percentage of buildings with a small estimation error (i.e., < 2 meters). This is because the buildings in the NY city block are not too tall, such that their height estimation using Ours-building is less impacted by camera location errors (cf. Section V-C).

3) Building Height Estimation for .: The baseline method cannot be applied to tall buildings because it takes images with a horizontal view along the street only, so here we only show the results from Ours-full (which, again, is the same as Ours-CBHE since it only uses street-facing images). As shown in Table III, 54.1% of the tall buildings have a height estimation error of less than 5 meters, and 73.0% have an error of less than 10 meters. 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). Since skyscrapers are taller than 100 meters, even a 10-meter error is less than 10%.

TABLE III
EFFECTIVENESS ON TALL BUILDING HEIGHT ESTIMATION

Absolute error	Percentage	Relative Error	Percentage
<5m	54.1%	<5%	59.5%
<10m	73.0%	<10%	86.5%

E. Error Analysis

We summarize the challenging cases for our model. For buildings with a small width, roofline detection from images facing the street is sensitive to GPS errors. Although we calibrate the camera location, the camera location may still contain errors, and the detected roofline may be inaccurate. For example, in Fig. 15a, building A is quite narrow. The computed area of the building by our algorithm is shown as

the gray area, which is off the building and will not yield the intended building height.

As for images facing the building with an upward-looking view, the camera location calibration algorithm does not apply because the building corners are close to the camera, and their positions in the images cannot be computed precisely. Fig. 15b shows a possible mistake of height estimation without camera calibration from images facing the building. Building A is close to its two neighboring buildings. When detecting the roofline of building A, our model outputs the roofline of building B (the red line) because the computed area of building A (i.e., the gray area) has a larger overlap with building B, which leads to a wrong height estimation.



Fig. 15. Example of challenging cases.

Further, for buildings with rooflines that are blocked by other objects such as trees in both images facing the street and those facing the building, our model may output a wrong estimation. Take Figs. 15c and 15d as an example. Trees block the roof of building C heavily. The actual roofline is largely blocked and hence has a low roofline score. It is also possible that the unblocked segment of the roofline was considered to be outside the scope of the building (i.e., the gray region) due to GPS errors. As shown in Fig. 15d, the red line below the true roofline may be detected as the result, which leads to a wrong height estimation. We have tried image in-painting [66], [67] to recover the blocked area, but the roofline cannot be clearly in-painted for building height estimation (cf. Fig. 15e).

F. Sensitivity Analysis

The building height estimation is highly related to the camera location error and the corner detection error. In this section, we analyze the building height estimation error ratio with respect to the camera location error (in the real world) and the corner detection error (in the Google Street Scene images). For simplicity, we focus on the height above line l_c in Fig. 4, since the height below l_c is a constant.

As shown in Fig. 4, the height estimation error is irrelevant to the error of the camera location projected on the x' -axis and is only related to that on the z' -axis. If the real camera location has a distance \hat{d}_e to the location obtained from the Google Maps API on the z' -axis, then, according to Equation 1, the height error ratio h'_e of the ground-truth height h'_g above line l_c is computed as:

$$h'_e = \frac{|h' - h'_g|}{h'_g} = \left(\left| \frac{h'_g(\hat{d} + \hat{d}_e)}{d} - h'_g \right| \right) / h'_g = \left| \frac{(\hat{d} + \hat{d}_e)}{\hat{d}} - 1 \right| \quad (15)$$

where h'_e denotes the estimated height, \hat{d} denotes the ground-truth distance between the camera and the building on the

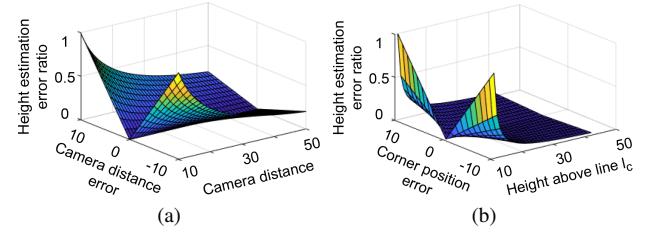


Fig. 16. Height estimation error with respect to camera distance and corner position error.

z' -axis, and \hat{d}_e denotes the camera location error on the z' -axis. We visualize the height estimation error ratio h'_e with respect to \hat{d} and \hat{d}_e in Fig. 16a. For a given value of \hat{d} , h'_e increases linearly with \hat{d}_e . When \hat{d} increases given a fixed \hat{d}_e value, h'_e decreases exponentially.

As also shown in Fig. 4, the height estimation error is irrelevant to the corner detection error on the x -axis and is only related to that on the y -axis. Suppose now the camera location is accurate, while the detected corner position has an error c_e to the corner's ground-truth position on the y -axis. Then, according to Equation 1, the height error ratio h'_e of the ground-truth height h'_g above line l_c is:

$$h'_e = \frac{|h' - h'_g|}{h'_g} = \left(\left| \frac{\hat{d}(h_r + c_e)}{f} - \frac{h_r \cdot \hat{d}}{f} \right| \right) / h'_g = \left| \frac{(h_r + c_e)}{h_r} - 1 \right| \quad (16)$$

We visualize h'_e with respect to h_r and c_e in Fig. 16b. For a given h_r , h'_e increases linearly with c_e . When h_r increases given a fixed c_e value, h'_e decreases exponentially.

Although the building height estimation error increases linearly to the camera distance error and corner position error, large camera distance errors or large corner position errors rarely occur in practice. Since there is no ground truth/labeled data set for analyzing such error distributions, we take the building height estimation error statistics on three data sets as an indirect indicator of these errors (Appendix C).

VII. CONCLUSION

We proposed a scalable algorithm to learn building height from street scene images. Our model consists of camera location calibration and building roofline detection as its two main steps. To calibrate camera location, our model performs camera projection by matching two building corners in street scene images with their physical locations obtained from a 2D map. To compute the building height, we first detect roofline candidates from street scene images facing the street (which is considered in our previous conference version [18]) and those facing the building with an upward-looking view (which is a novel contribution of this journal extension). We filter the roofline candidates via a proposed neural network model named RoofNet. Among the remaining candidates, we select the best roofline candidate via a proposed entropy-based ranking algorithm. When the true roofline is identified, we compute building height via the pinhole camera model. Experimental results show that RoofNet outperforms SOTA classifiers consistently, and our building height estimation method outperforms the baseline by up to 11.9%.

APPENDIX

A. Frequently Used Symbols

TABLE IV
FREQUENTLY USED SYMBOLS

Notation	Description
h_r	the height of a building above the center line of an image capturing the building
h_b	the height of a building below the center line of an image capturing the building
c_n	the corner nearest to the camera
c_x	the corner farthest to o in the image plane
c_z	the corner closest to o in the image plane
d	the horizontal distance from the camera to c_n of a building
\hat{d}	the projected length of d onto the z' -axis
f	the focal length of the camera
l_r	a building roofline

B. RoofNet Architecture

TABLE V
ROOFNET ARCHITECTURE

Layer	Number	Number of kernels	Output size
Conv 5×5	32	28×28	24×24
Maxpool 2×2	-	24×24	12×12
Conv 5×5	64	12×12	8×8
Maxpool 2×2	-	8×8	4×4
FC	-	4×4	$1,024 \times 1$
Dropout	-	$1,024 \times 1$	$1,024 \times 1$
FC	-	$1,024 \times 1$	10×1
L2	-	10×1	10×1

C. Building Height Estimation Error Statistics

TABLE VI

BUILDING HEIGHT ESTIMATION ERROR STATISTICS ON THREE DATA SETS

Data Set	Mean Error	Standard Error	Median Error
NYC	2.43	0.10	1.59
SF city block 1	2.81	0.87	1.27
SF city block 2	1.85	0.27	1.24

REFERENCES

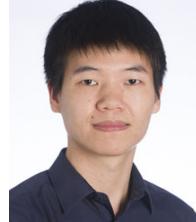
- J. Pan, M. Hebert, and T. Kanade, "Inferring 3D layout of building facades from a single image," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2918–2926.
- E. Ng, "Policies and technical guidelines for urban planning of high-density cities—air ventilation assessment (AVA) of Hong Kong," *Building and Environment*, vol. 44, no. 7, pp. 1478–1488, 2009.
- F. Grabler, M. Agrawala, R. W. Sumner, and M. Pauly, "Automatic generation of tourist maps," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 100:1–100:11, 2008.
- A. Rousell and A. Zipf, "Towards a landmark-based pedestrian navigation service using OSM data," *ISPRS International Journal of Geo-Information*, vol. 6, no. 3, p. 64, 2017.
- F. Qi, J. Z. Zhai, and G. Dang, "Building height estimation using Google Earth," *Energy and Buildings*, vol. 118, pp. 123–132, 2016.
- N. Kadhimi and M. Mourshed, "A shadow-overlapping algorithm for estimating building heights from VHR satellite images," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 1, pp. 8–12, 2017.
- Z. Wang, L. Jiang, L. Lin, and W. Yu, "Building height estimation from high resolution SAR imagery via model-based geometrical structure prediction," *Progress In Electromagnetics Research*, vol. 41, pp. 11–24, 2015.
- Y. Xu, P. Ma, E. Ng, and H. Lin, "Fusion of WorldView-2 stereo and multitemporal terrasar-x images for building height extraction in urban areas," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 8, pp. 1795–1799, 2015.
- D. Brunner, G. Lemoine, L. Bruzzone, and H. Greidanus, "Building height retrieval from vhr SAR imagery based on an iterative simulation and matching technique," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 3, pp. 1487–1504, 2010.
- P. E. Brown, Y. Kanza, and V. Kounev, "Height and facet extraction from LiDAR point cloud for automatic creation of 3D building models," in *ACM International Conference on Advances in Geographic Information Systems*, 2019, pp. 596–599.
- WordPress and HitMag, (2018, Feb.) LiDAR and RADAR information. [Online]. Available: <http://lidarradar.com/category/info>
- Q. Zhang, L. Ge, S. Hensley, G. I. Metternicht, C. Liu, and R. Zhang, "Polgan: A deep-learning-based unsupervised forest height estimation based on the synergy of polinsar and lidar data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 186, pp. 123–139, 2022.
- D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver, "Google street view: Capturing the world at street level," *Computer*, vol. 43, no. 6, pp. 32–38, 2010.
- M. Haklay and P. Weber, "OpenStreetMap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.
- J. Yuan and A. M. Cheriyadat, "Combining maps and street level images for building height and facade estimation," in *ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, 2016, pp. 8:1–8:8.
- E. Díaz and H. Arguello, "An algorithm to estimate building heights from Google street-view imagery using single view metrology across a representational state transfer system," in *Dimensional Optical Metrology and Inspection for Practical Applications V*, vol. 9868, 2016, p. 98680A.
- F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.
- Y. Zhao, J. Qi, and R. Zhang, "CBHE: Corner-based building height estimation for complex street scene images," in *The Web Conference*, 2019, pp. 2436–2447.
- X. Sun, Z. Yang, C. Zhang, K.-V. Ling, and G. Peng, "Conditional Gaussian distribution learning for open set recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 480–13 489.
- P. Agarwal, W. Burgard, and L. Spinello, "Metric localization using Google street view," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015, pp. 3111–3118.
- L. Liu, H. Li, and Y. Dai, "Efficient global 2D-3D matching for camera localization in a large-scale 3D map," in *IEEE International Conference on Computer Vision*, 2017, pp. 2391–2400.
- H. Li, T. Fan, H. Zhai, Z. Cui, H. Bao, and G. Zhang, "BDLoc: Global localization from 2.5D building map," in *IEEE International Symposium on Mixed and Augmented Reality*, 2021, pp. 80–89.
- R. Liu, J. Zhang, S. Chen, and C. Arth, "Towards SLAM-based outdoor localization using poor GPS and 2.5D building models," in *International Symposium on Mixed and Augmented Reality*, 2019, pp. 1–7.
- C. Arth, C. Pirchheim, J. Ventura, D. Schmalstieg, and V. Lepetit, "Instant outdoor localization and SLAM initialization from 2.5D maps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 11, pp. 1309–1318, 2015.
- A. Armagan, M. Hirzer, P. M. Roth, and V. Lepetit, "Learning to align semantic segmentation and 2.5D maps for geolocalization," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4590–4597.
- R. Liu, J. Zhang, S. Chen, T. Yang, and C. Arth, "Accurate real-time visual SLAM combining building models and GPS for mobile robot," *Journal of Real-Time Image Processing*, vol. 18, pp. 419–429, 2021.
- H. Chu, A. Gallagher, and T. Chen, "GPS refinement and camera orientation estimation from a single image and a 2D map," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 171–178.
- B. Semaan, M. Servières, G. Moreau, and B. Chebaro, "Camera pose estimation using collaborative databases and single building image," *Journal of Geographic Information System*, vol. 12, no. 06, pp. 620–645, 2020.
- G. Liassis and S. Stavrou, "Satellite images analysis for shadow detection and building height estimation," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 119, pp. 437–450, 2016.
- H. Hao, S. Baireddy, E. Bartusiak, M. Gupta, K. LaTourette, L. Konz, M. Chan, M. L. Comer, and E. J. Delp, "Building height estimation via satellite metadata and shadow instance detection," in *Automatic Target Recognition XXXI*, vol. 11729, 2021, p. 117290L.
- H. Sportouche, F. Tupin, and L. Denise, "Extraction and three-dimensional reconstruction of isolated buildings in urban scenes from high-resolution optical and SAR spaceborne images," *IEEE Transactions*

- on Geoscience and Remote Sensing*, vol. 49, no. 10, pp. 3932–3946, 2011.
- [32] Y. Sun, S. Montazeri, Y. Wang, and X. X. Zhu, “Automatic registration of a single sar image and gis building footprints in a large-scale urban area,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 170, pp. 1–14, 2020.
- [33] J. Mahmud, T. Price, A. Bapat, and J.-M. Frahm, “Boundary-aware 3D building reconstruction from a single overhead image,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 441–451.
- [34] C. Zeng, J. Wang, W. Zhan, P. Shi, and A. Gambles, “An elevation difference model for building height extraction from stereo-image-derived DSMs,” *International Journal of Remote Sensing*, vol. 35, no. 22, pp. 7614–7630, 2014.
- [35] J. Kopf, B. Chen, R. Szeliski, and M. Cohen, “Street slide: Browsing street level imagery,” *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–8, 2010.
- [36] Baidu. (2018, Oct.) Baidu map. [Online]. Available: <https://map.baidu.com>
- [37] P. A. Zandbergen and S. J. Barbeau, “Positional accuracy of assisted GPS data from high-sensitivity GPS-enabled mobile phones,” *The Journal of Navigation*, vol. 64, no. 3, pp. 381–399, 2011.
- [38] A. Grammenos, C. Mascolo, and J. Crowcroft, “You are sensing, but are you biased? A user unaided sensor calibration approach for mobile sensing,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, pp. 1–26, 2018.
- [39] X. Liu and D. Wang, “A spectral histogram model for texton modeling and texture discrimination,” *Vision Research*, vol. 42, no. 23, pp. 2617–2634, 2002.
- [40] G. Lin, A. Milan, C. Shen, and I. D. Reid, “Refinenet: Multi-path refinement networks for high-resolution semantic segmentation,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5168–5177.
- [41] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, “Image segmentation using deep learning: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2021.
- [42] P. Dollár and C. L. Zitnick, “Structured forests for fast edge detection,” in *IEEE International Conference on Computer Vision*, 2013, pp. 1841–1848.
- [43] H. Kim, J.-M. Dischner, H. Rushmeier, and B. Benes, “Edge-based procedural textures,” *The Visual Computer*, vol. 37, no. 9, pp. 2595–2606, 2021.
- [44] T. Meyer and C. Meyer, *Creating Motion Graphics with After Effects*. Taylor & Francis, 2010.
- [45] B. Klingner, D. Martin, and J. Roseborough, “Street view motion-from-structure-from-motion,” in *IEEE International Conference on Computer Vision*, 2013, pp. 953–960.
- [46] W. Zheng, X. Liu, and L. Yin, “Research on image classification method based on improved multi-scale relational network,” *PeerJ Computer Science*, vol. 7, p. e613, 2021.
- [47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [48] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in *European Conference on Computer Vision*, 2016, pp. 499–515.
- [49] X. He, Y. Zhou, Z. Zhou, S. Bai, and X. Bai, “Triplet-center loss for multi-view 3D object retrieval,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1945–1954.
- [50] X. Wang, R. Zhang, Y. Sun, and J. Qi, “KDGAN: Knowledge distillation with generative adversarial networks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 783–794.
- [51] K. Q. Weinberger, J. Blitzer, and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” in *Advances in Neural Information Processing Systems*, 2006, pp. 1473–1480.
- [52] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM transactions on intelligent systems and technology*, vol. 2, no. 3, pp. 1–27, 2011.
- [53] L.-y. Sun, C.-l. Miao, and L. Yang, “Ecological-economic efficiency evaluation of green technology innovation in strategic emerging industries based on entropy weighted topsis method,” *Ecological Indicators*, vol. 73, pp. 554–558, 2017.
- [54] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.
- [55] A. Criminisi, I. Reid, and A. Zisserman, “A plane measuring device,” *Image and Vision Computing*, vol. 17, no. 8, pp. 625–634, 1999.
- [56] M. J. Beal, “Variational algorithms for approximate bayesian inference,” Ph.D. dissertation, University of London, 2003.
- [57] J. M. Coughlan and A. L. Yuille, “Manhattan world: Compass direction from a single image by bayesian inference,” in *IEEE International Conference on Computer Vision*, 1999, pp. 941–947.
- [58] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [59] Google. (2018, Oct.) Nearmap. [Online]. Available: <http://maps.au.nearmap.com/>
- [60] H. Zhang and V. M. Patel, “Sparse representation-based open set recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 8, pp. 1690–1696, 2017.
- [61] A. Bendale and T. E. Boult, “Towards open set deep networks,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1563–1572.
- [62] Z. Xia, P. Wang, G. Dong, and H. Liu, “Adversarial motorial prototype framework for open set recognition,” *arXiv preprint arXiv:2108.04225*, 2021.
- [63] Q. Xiao, H. Luo, and C. Zhang, “Margin sample mining loss: A deep learning based method for person re-identification,” *arXiv:1710.00478*, 2017.
- [64] W. J. Scheirer, L. P. Jain, and T. E. Boult, “Probability models for open set recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2317–2324, 2014.
- [65] L. V. D. Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.
- [66] A. Li, J. Qi, R. Zhang, X. Ma, and K. Ramamohanarao, “Generative image inpainting with submanifold alignment,” in *International Joint Conference on Artificial Intelligence*, 2019, pp. 811–817.
- [67] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, “Free-form image inpainting with gated convolution,” in *IEEE International Conference on Computer Vision*, 2019, pp. 4471–4480.

Yunxiang Zhao is an assistant professor in Beijing Institute of Biotechnology. He received his Ph.D. degree from The University of Melbourne. His research interests include spatial data analysis, bioinformatics, and deep learning.



Jianzhong Qi is a Senior Lecturer in the School of Computing and Information Systems at The University of Melbourne. He received his Ph.D. degree from The University of Melbourne in 2014. His research interests include machine learning and data management and analytics, with a focus on spatial, temporal, and textual data.



Flip Korn is a Research Scientist at Google Research NYC and, prior to that, was a member of the Database Research Department at AT&T Labs-Research. He received his Ph.D. from the University of Maryland, College Park. His research interests include big data, data mining, and machine learning.



Xiangyu Wang is affiliated with School of Civil Engineering and Architecture of East China Jiaotong University, and Curtin University. He was the Founder and the inaugural Chair of the Curtin Advanced Technology Research and Innovation Alliance. He has been involved in the organization of several international conferences including the Honorary Chair of the Conference on Innovative Production and Construction (IPC).