ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Instance Segmentation of Geometrical Shapes in Aerial Images

Master's Thesis

Zuoyue Li

April 2018

Supervisors: Dr. A. Lucchi, Dr. J. D. Wegner, Prof. Dr. T. Hofmann
Department of Computer Science, ETH Zürich

Abstract

Deep learning methods have recently demonstrated remarkable achievements on image semantic segmentation. However, their architectures are still limited to pixel-wise labeling, which makes it hard to exploit high-level topology.

The goal of this project is to develop novel deep learning architectures for exploiting geometrical shapes of arbitrary structure. We want to depart from the standard paradigm that labels pixels but instead directly exploit and learn the geometry of the objects. In practice, we use the dataset of aerial images and aiming at extracting buildings' polygon shapes.

Two recent models, PolygonRNN and FPN (Feature Pyramid Network), draw our attention. The former one can well exploit geometrical shape of a single object in an image. The latter one performs well on the multi-scale object detection. Since our goal can be divided into two parts, objects detection and geometrical segmentation, the basic idea of the proposed solution would utilize these two models in two steps.

Specifically, in order to address the problem, we propose a new model, R-PolygonRNN (Region-based PolygonRNN), which is the integration of FPN and PolygonRNN. The model has three different versions, the two-step version, hybrid version and hybrid version with RoIAlign. We also introduce beam search to PolygonRNN in order to find the best polygon proposal.

Experiments show that our proposed model outperforms two previous works in the dataset of Chicago. The model can well localize each building within an aerial image, and for each building, it can well extract the building's geometrical shape.

Acknowledgment

Foremost, I would like to express my sincere gratitude to Dr. Aurelien Lucchi and Dr. Jan Dirk Wegner for supervising my Master's thesis project, supporting me continuously and contributing many useful ideas. Their guidance helped me in all the time of discussing project and writing thesis, and I have learned a lot in the field of object detection and geometrical segmentation.

I would also like to thank Prof. Thomas Hofmann for providing me with the opportunity of this interesting project, as well as his suggestions about beam search. I would say doing Master's thesis project at Data Analytics Lab is an unforgettable experience for me.

Besides my supervisors, I would like to thank Tianhao Wei, a junior to me at Zhejiang University, for giving me many suggestions for the implementation details about PolygonRNN.

My sincere thank also goes to my friends, Jingxuan He, Xiaojuan Wang, Canxi Chen, Jie Huang, Junlin Yao, and Renfei Liu, for all their help, supports and companionship.

Last but not the least, I would like to thank my parents Haiyan Dai and Fasheng Li, for their spiritual supports and understandings throughout my studying life in Switzerland.

Contents

Chapter 1

Introduction

This chapter mainly provides a brief introduction to the entire project. Section 1.1 presents the background and some fundamental concepts in order to give readers a basic understanding of this field. Section 1.2 defines the problems of this project to be solved. Section 1.3 gives a brief introduction to our contribution and our proposed new model. Section 1.4 illustrates the structure of this thesis for the convenience of readers.

1.1 Background

In this section, the background of this project is introduced. Subsection 1.1.1 focus on aerial image and its application. Subsection 1.1.2 presents the concept of image segmentation, and its commonly used methods. Subsection 1.1.3 introduces the idea of segmentation with geometry, which is the main point of our project.

1.1.1 Aerial Image

In our project, an aerial image generally refers to the "optical overhead imagery" [1] acquired by aircraft. This kind of image has an extremely wide range of applications in the field of geographical surveying and mapping, which can not only clearly depict the terrain, but can also show the structure and the layout of the city. Furthermore, it can also provide services such as land use status and remote sensing monitoring.

Especially in the metropolis, many buildings are constantly being updated with the expansion of the city, and many landscapes are changing with human's activities. Therefore, the city's electronic map needs to be updated accordingly with the change of city appearance as well. In this case, the acquisition of aerial images becomes necessary, as those images can provide significant detailed visual information from above of the city, which is typically

unaccessible from the human's perspective. Figure 1.1 shows two examples of aerial images.

(a) a tile of city Zurich (b) a tile of city Chicago

Figure 1.1: Example of two aerial images. This kind of images can provide people with important reference of the city layout from above.

Nowadays, huge volumes of aerial images are captured with airborne or spaceborne platforms. The increasing volume makes manual interpretation prohibitive [1]. Hence, we should employ appropriate ideas and methods from the field of computer vision to utilize this kind of data. In order for the machine to better understand aerial images, we can perform image segmentation.

1.1.2 Image Segmentation

In the field of computer vision, image segmentation is the process of partitioning an image into multiple sets of pixels. The goal of segmentation is to simplify and change the representation of an image and make the image more meaningful and easier to analyze [2].

In our project, image segmentation mainly refers to the so-called semantic segmentation, which is a process of labeling each pixel of the image. Hence the segmentation is exactly a pixel-wise classification problem. Note that the labels between two adjacent pixels are not independent, but related to each other. Pixels with the same label are generally similar in the metric of certain visual characteristics, such as color, brightness or texture.

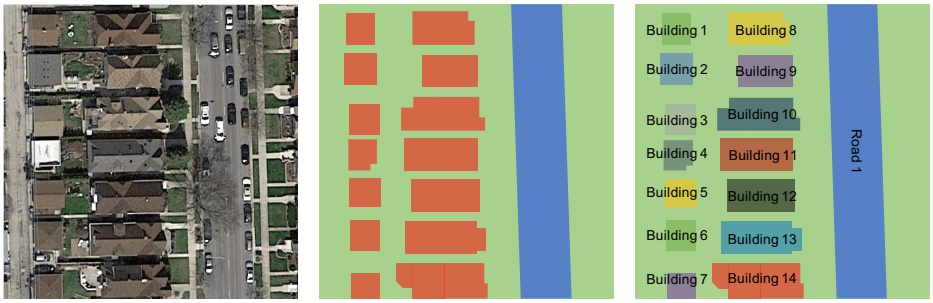
Another kind of segmentation is the so-called instance segmentation. It not only does semantic segmentation, but also distinguishes between the object instances, even they have the same label. That is to say, object instances with

same labels are required to have different IDs. Figure 1.2 shows the difference between semantic and instance segmentation.

(a) original image (b) semantic segmentation (c) instance segmentation

Figure 1.2: Examples of semantic segmentation and instance segmentation. Image copyright owned by [3]. The original image (a) shows a room with several chairs. (b) is the semantic segmentation result of (a), where blue color denotes the chairs, and other colors such as white and gray denote irrelevant background. (c) is instance segmentation result of (a), where all chairs have the same label, but different colors which are used to indicate different instances.

Specifically, for semantic segmentation in aerial images, usually what we do is to label each pixel as buildings, roads or background, which can be seen in figure 1.3(b). Its instance segmentation result is shown in figure 1.3(c).



(a) original aerial image (b) semantic segmentation (c) instance segmentation

Figure 1.3: Examples of semantic segmentation and instance segmentation in an aerial image. The original aerial image (a) is a tile of city Chicago. (b) is the semantic segmentation result of (a), where red color denotes the buildings, blue color denotes the roads, green color denotes the background. (c) is instance segmentation result of (a), where all buildings have the same label, but different "ID" numbers which can be used to indicate they are different instances.

As a matter of fact, for an image where the objects (buildings) are separated like figure 1.3(a), we can easily obtain the instance segmentation result based on the pixel connectivity information from the semantic segmentation result. However, if objects overlap in the image like the chairs in figure 1.2(a), it is difficult to do such a thing based on the semantic segmentation result. Thus, in this case, instance segmentation can show its advantages.

Traditional semantic segmentation methods include clustering method [4],

histogram-based method [2], compression-based methods [5], region-growing method [6] and so on. Recently, deep learning methods have demonstrated remarkable achievements in image segmentation tasks. For those methods, please refer to section 2.1 for more details.

1.1.3 Geometrical Shape

As introduced in subsection 1.1.2, the output format of either semantic or instance segmentation, is the per-pixel mask. Although it is currently the mainstream choice in the field of image segmentation, it is undeniable that the per-pixel mask has limitations on the representation of geometrical shapes. The geometrical shape here generally refers to a polygon represented by a series of ordered vectors or coordinates.

Indeed, we can undergo more processing steps to obtain geometrical shapes based on the instance segmentation result. However, we want to get rid of the pixel-wise labeling rigidity and directly describe the geometrical shape of each object in an image. Specifically, in our project, deviating from the standard paradigm of labeling pixels and aiming to directly learn the geometry of the buildings in aerial images have following advantages: (1) Polygon representation has much less redundancy and relatively less storage than pixel-wise labeling; (2) Polygon representation is a kind of vector illustration, thus can be used at arbitrary levels or scales; (3) Buildings with polygon representation can be modelled more naturally; (4) Polygon representation can be directly marked in the electronic map, but per-pixel mask can not. The polygon representation is therefore a more compact, useful and structure-aware representation of object silhouettes in the segmentation of buildings in aerial images.

As mentioned in subsection 1.1.2, deep learning methods have shown significant progress in tasks of image segmentation. However, the architectures used in these methods are still limited to conventional grid structure diagrams and their output is still pixel-wise. The rigidity of these networks makes it difficult to exploit high-level priors about the geometrical shapes of objects in an image.

Therefore, the purpose of this project is to develop novel deep learning methods for the geometrical shapes of arbitrary structures, which means that we want to introduce object geometrical shapes into deep learning techniques.

1.2 Problem Definition

Given an aerial image, our goal is to extract the polygon shape for each building in the image. The input is an aerial image, denoting

$$I = \{I_{ijk}\}_{i\in\{1,2,\ldots,h\},j\in\{1,2,\ldots,w\},k\in\{1,2,3\}}, \tag{1.1}$$

where I_{ijk} denotes the pixel value of i-th row, j-th column and k-th channel in the image, w and h denote the width and height of the input image. The output is polygons, denoting

$$P = \{P^{(n)}\}_{n \in \{1,2,\ldots,N\}}, \tag{1.2}$$

$$P^{(n)} = \{(i_t^{(n)}, j_t^{(n)})\}_{t \in \{1,2,\ldots,L_n\}}, \tag{1.3}$$

where N denotes the number of complete buildings (or polygons) in the image, $P^{(n)}$ and L_n denote the n-th polygon and its number of vertices, $(i_t^{(n)}, j_t^{(n)})$ denote the image coordinates of t-th vertex of the polygon $P^{(n)}$ in the original input image.

In short, our goal is to achieve instance segmentation of geometrical shapes for buildings in aerial images. Figure 1.4 shows an example of desired input and output.

(a) aerial image as input (b) expected polygons as output

Figure 1.4: Example of instance segmentation of geometrical shapes in an aerial image. (a) is the input aerial image containing 3 complete buildings. (b) is the visualized result for output polygons.

1.3 Focus of This Work

The problem defined in section 1.2 is challenging because it not only requires the correct detection (or localization) for all buildings in an aerial image, but also needs to precisely segment each building in the representation of polygon rather than per-pixel mask. Therefore, it is exactly a combination of two tasks of computer vision. The first one is object detection, where the goal is to detect (or localize) each individual object in the image using a bounding box.

The second one is the geometrical instance segmentation, where the goal is to extract polygon for a single instance.

In order to solve this problem, we propose a new model, R-PolygonRNN (Region-based PolygonRNN), which is a combination of the FPN (Feature Pyramid Network) [7] and PolygonRNN [8]. In our new model, FPN is used to localize buildings, i.e. to detect RoIs (Regions of Interest) in the image, and PolygonRNN is used to find the geometrical shape for a single object. Experiments show that the new proposed model can successfully find geometrical shapes for multiple buildings in an aerial image.

In summary, our contributions are as follows:

- We combine FPN [7] and PolygonRNN [8], and propose the model R-PolygonRNN, with 3 versions. The proposed model tackles the shortcoming of PolygonRNN, which can only segment single object;
- We redesign the lateral connection of FPN with VGG-16 [9] backbone in order for the better combination with PolygonRNN;
- We learn the idea from `seq2seq` [10] and introduce the beam search algorithm to the prediction phase of PolygonRNN, which addresses the false vertex problem and significantly improves the prediction result.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 reviews related work, focusing on the deep learning methods for image segmentation, previous work related to segmentation in aerial images, as well as the two recent new models, Mask R-CNN [11] and PolygonRNN. In chapter 3, the architectures of PolygonRNN and FPN are presented, and the structure of our proposed model is also explained in detail. Chapter 4 describes the ground truth dataset we use and gives the experiment configurations and results. Finally, chapter 5 makes conclusions, points out the problems which exist in our project and gives future direction.

In addition, the common terminologies used in our project and its corresponding detailed explanations are shown as follows:

- **Object Detection (Object Localization)**:
  Detection or localization via bounding boxes instead of masks;
- **Semantic Segmentation**:
  Pixel-wise classification without differentiating instances;
- **Instance Segmentation (Semantic Instance Segmentation)**:
  Both semantic and a form of detection;

- **Segmentation of Geometrical Shape (Geometrical Segmentation)**:
  Extraction of the polygon outlining **single** object;
- **Instance Segmentation of Geometrical Shapes (Geometrical Instance Segmentation)**:
  Both geometrical and a form of detection.

Chapter 2

Related Work

In this chapter, some related work is illustrated in detail. Section 2.1 presents several deep learning networks for semantic and instance segmentation. Section 2.2 introduces methods related to geometrical shapes extraction with and without machine learning frameworks. Section 2.3 gives the summary of some of these models, points out their respective deficiencies with regard to our problem, and further discusses the feasibility of applying those models to our problem.

2.1 Deep Learning for Image Segmentation

This section mainly introduces some deep learning frameworks used in image segmentation. Subsection 2.1.1 and subsection 2.1.2 focus on semantic and instance segmentation, respectively.

2.1.1 Frameworks for Semantic Segmentation

Traditional CNN-based semantic segmentation approach is usually done by classifying a pixel using the small neighboring patch centered on this pixel as input to the CNN. This method has several disadvantages such as the large storage, inefficient computing (patches of adjacent pixels are basically repeated) and the limitation of the sensing area (only local features can be extracted). However, the application of FCN (Fully Convolutional Networks) to semantic segmentation of images, which originates from this far-reaching paper [12], breaks with precedent. Figure 2.1 shows the model architecture of FCN, and the core techniques used by FCN are also shown as follows.

Convolutionalization The CNN used for image classification generally has fully connected layers at the end. It compresses the two-dimensional matrix into a one-dimensional vector, thus losing the spatial information. In contrast,

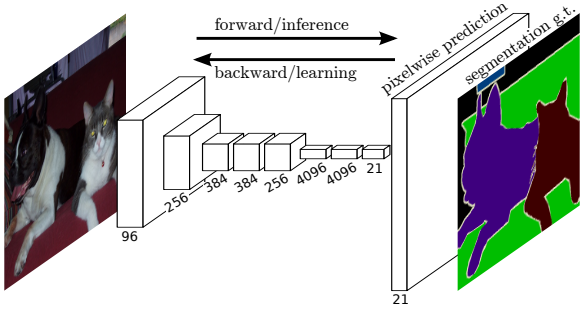


Figure 2.1: The model architecture of FCN. Image copyright owned by [12].

FCN discards those fully connected layers and replaces it with convolutional layers. This replacement is so-called convolutionalization. The spatial output maps of these convolutionalized models make them a natural choice for dense problems like semantic segmentation. Figure 2.2 shows the convolutionalization process.

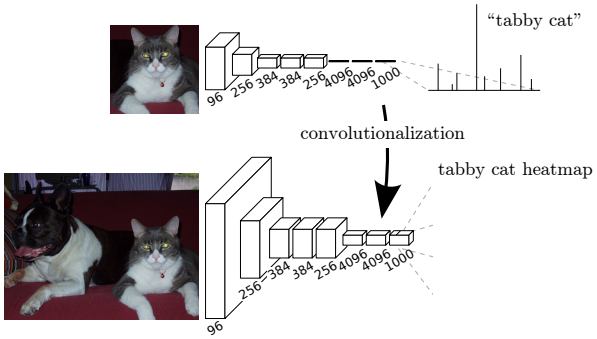


Figure 2.2: Convolutionalization in FCN. Image copyright owned by [12].

Upsampling In-network upsampling layers enable pixel-wise prediction and learning in nets with subsampled pooling. As we know, CNN uses several pooling layers to reduce the image size (usually the size is reduced by half for each pooling layer). However, in semantic segmentation, the mask with the same size as the original image is required. Hence, we need to perform upsampling, and the corresponding layer is so-called deconvolutional (transposed convolutional) layer. It shows that learning dense prediction through upsampling is more effective and efficient, especially when combined with the skip layer fusion [12].

Skip Connection Skip architecture combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentations. Skip connection, which is

shown in figure 2.3, can take advantage of this feature spectrum and refine the spatial precision of the output.

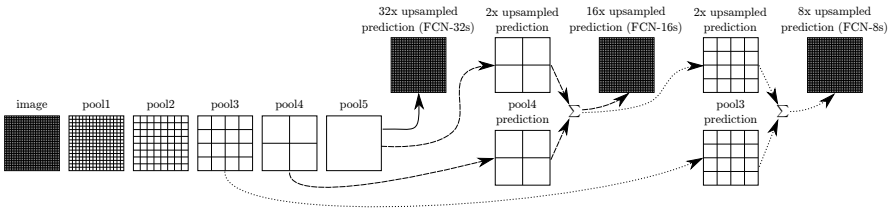
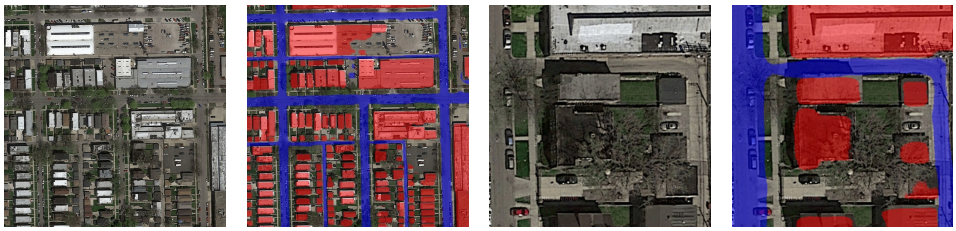


Figure 2.3: Skip connection in FCN. Image copyright owned by [12].

Compared with previous conventional method of semantic segmentation using CNN, FCN has two major advantages: (1) It can accept input image of arbitrary size, without stretching the image to a fixed size; (2) FCN is more efficient as it avoids the problem of duplicate in both storage and computation.

At the same time, the disadvantages of FCN are also obvious: (1) The results obtained are not precise enough, since upsampling makes the mask blurry and smooth, and it is not sensitive to the details in the image; (2) Pixel-wise classification does not fully consider the relationship between neighboring pixels, ignoring the spatial regularization used in general per-pixel segmentation methods, thus lacking spatial consistency.

In order to tackle the shortcomings mentioned above, many expansions of FCN are proposed such as DeepLab [13], U-Net [14] and SegNet [15]. Due to space limitations, in this thesis, we will not elaborate on all of these models. Please refer to the original paper for more details.



(a) (b) (c) (d)

Figure 2.4: Semantic segmentation in aerial images. Image copyright owned by [1]. (a) and (c) are two aerial images. In the semantic segmentation result (b) and (d), the pixels covered by red color refer to buildings, while the pixels covered by blue color refer to roads.

FCN for Aerial Images In paper [1], FCN is employed in semantic segmentation in aerial images. In this work, each pixel in the aerial image is classi-

fied as different labels, i.e. buildings, roads or background. The output of the model is a per-pixel labeled mask. An example is shown in figure 2.4. We can see that the model can well distinguish between buildings and roads. However, there are also some limitations in this method with regard to our problem: (1) There is no instance segmentation here, and the buildings cannot be detected unless we do further pixel connectivity detection; (2) There is no segmentation of geometrical shapes of the objects in this model. Therefore, this model cannot be fully used to solve our problems.

2.1.2 Frameworks for Instance Segmentation

As mentioned in subsection 1.1.2, instance segmentation stands at a higher level than semantic segmentation, which additionally combines object detection as a kind of "preprocessing". In fact, current object detection methods using deep learning networks, such as Faster R-CNN [16], YOLO [17], SSD [18], are very mature already.

Driven by the effectiveness of these object detection networks mentioned above, many approaches to instance segmentation are based on segment proposals. Earlier methods such as DeepMask [19] and its following work such as paper [20] and [21] learn to propose segment candidates, which are then classified. Similarly, paper [22] comes up with a complex multiple-stage cascade that predicts segment proposals from bounding box proposals, also followed by classification. In these methods, "segmentation precedes recognition, which is slow and less accurate," [11].

Most recently, paper [23] integrates the segment proposal system and object detection system for FCIS (Fully Convolutional Instance Segmentation). The core idea for this combined model is to "predict a set of position-sensitive output channels fully convolutionally," [11]. These changes make the whole system faster and more efficient as these channels can predict object classes, boxes and masks simultaneously. But FCIS does not perform well on overlapping instances and creates odd edges, showing that "it is challenged by the fundamental difficulties of segmenting instances," [11].

Instead, here we would like to introduce a state-of-the-art framework for instance segmentation with classification, Mask R-CNN [11]. It is based on "parallel prediction of masks and class labels, which is simpler and more flexible," [11].

Mask R-CNN is a general framework for object instance segmentation and classification. It is the latest model in the R-CNN family, which includes R-CNN [24], Fast R-CNN [25], Faster R-CNN [16] as well. Mask R-CNN can efficiently detect and classify objects in an image and generate a high-quality segmentation mask for each instance simultaneously. Figure 2.5 shows example results of Mask R-CNN. We can see from the figure that this model

achieves excellent results. Figure 2.6 shows the simplified model architecture of Mask R-CNN. Each part of the structure is illustrated in detail as follows.

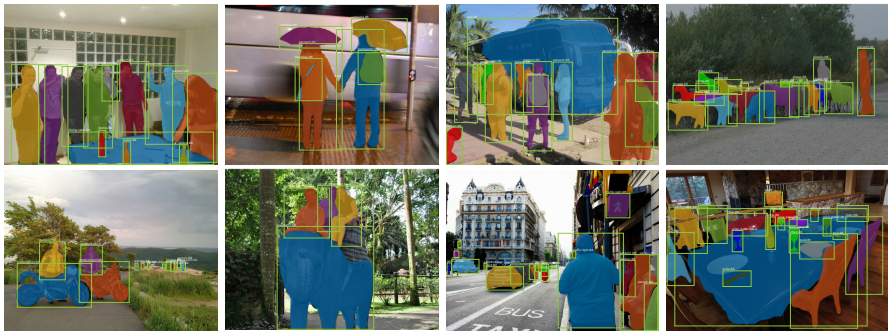


Figure 2.5: Example results of Mask R-CNN. Image copyright owned by [11].

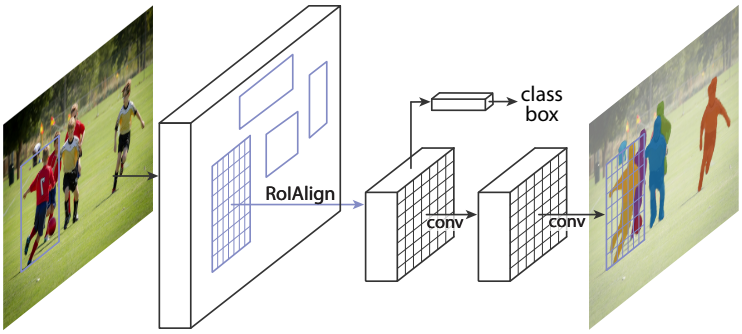


Figure 2.6: The simplified model architecture of Mask R-CNN. Image copyright owned by [11]. Feature Pyramid Network locates at the first layer in the figure.

Feature Pyramid Network The original image is first passed through FPN, which locates at the first layer in figure 2.6. FPN aims at finding multiple bounding boxes of the objects (known as RoIs) in the image. Actually, FPN can be regarded as an upgraded version of RPN (Region Proposal Network) used in Faster R-CNN [16]. Compared with RPN, FPN introduces feature pyramid and solves the problem of multi-scale detection. Especially when the target object is relatively small, usually FPN would give better results than RPN, and thus improve the accuracy of detection. For more details about the model architecture of FPN, please refer to section 3.2.

Classification Branch The classification branch exists from the earliest R-CNN [24] to the present Mask R-CNN [11]. It classifies each object in RoI found by FPN (or RPN in Faster R-CNN) as different classes and gives the probability distribution, using fully connected layers and softmax function. Note that this branch is not relevant to our problem, since currently we only

interested in the segmentation of buildings rather than kinds of different objects in an aerial image.

Mask Branch Mask R-CNN extends Faster R-CNN by adding the mask branch for predicting an object mask on each RoI in parallel with the existing classification branch. This branch uses small FCN for the pixel-wise semantic segmentation, just like what we have already mentioned in subsection 2.1.1. The mask branch should be relevant to our problem, but the mask it predicts is still at the pixel level instead of geometrical shape.

In short, Mask R-CNN can surpass prior instance segmentation results, but it cannot give any geometrical information of the objects in an image. Therefore, we consider that FPN is the only part in Mask R-CNN, which can be utilized to solve our problems. We hope that FPN can make correct detection of all buildings and localize each using a bounding box. What we need to do next is to find appropriate methods for extracting geometrical shapes for each object detected.

2.2 Frameworks Related to Geometrical Shapes

In this section, several frameworks which can find geometrical shapes are presented, including graph-based approach (see subsection 2.2.1), polygonal partitioning approach (see subsection 2.2.2), bounding box covering approach (see subsection 2.2.3) and PolygonRNN [8] (see subsection 2.2.4) approach.

2.2.1 Graph-based Approach

The graph-based approach is proposed in the thesis work [26], of which the pipeline is illustrated in figure 2.7. In this approach, buildings are regarded as polygons defined by corners and edges. The process of segmentation contains following steps: (1) Detect corners; (2) Compute the probability of two arbitrary different corners belonging to the same building; (3) Assemble over-complete set of potential corners and edges to a graph; (4) Segment graph into individual buildings.

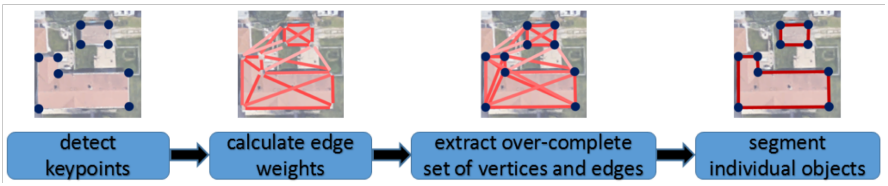


Figure 2.7: Pipeline of graph-based approach for extracting polygons. Image copyright owned by [26].

Actually, each stage of the graph-based approach has its own unsupervised solutions. For corner detection, we have Harris [27], SIFT [28], SURF [29] and so on. For the similarity of two image patches, we have histogram-based [30] and BoW-based (Bag-of-Words-based) methods [31]. For graph segmentation, we have graph's minimum cut, and unsupervised learning approaches like spectral clustering.

However, each stage can also employ supervised learning methods, thus the parameters of each stage become learnable and trainable. Specifically, the thesis work [26] uses U-Net [14] as the corner detector, uses adapted siamese network (called "triamese network" in the thesis work) described in [32] to compute the probabilities for all potential corner pairs, and uses normalized cuts [33] with its learning method [34] to segment individual objects. Figure 2.7 shows two example results of the model, where the segmentation effect is just passable.

In addition, the thesis work does not provide a more complicated situation, such as the number of building corners larger than four or the building not horizontally or vertically aligned. Besides, this model has a fatal flaw. In theory, even if we find the vertices of an instance, it is also very difficult to determine their orders in order to form a polygon, which requires further processing.

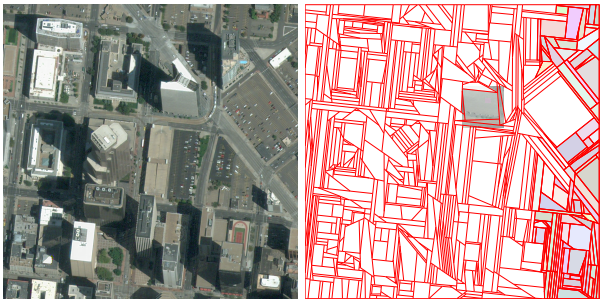
(a) (b)

Figure 2.8: Example results of graph-based approach. Image copyright owned by [26].

2.2.2 Polygonal Partitioning Approach

Many traditional image segmentation algorithms use superpixels. The paper [35] points out that floating polygons can be an interesting alternative, especially for analyzing scenes with strong geometric signatures such as man-made environments (cityscapes or aerial images of a city). The paper also shows that existing algorithms such as [36] and [37] produce homogeneously-sized polygons that fail to capture thin geometric structures and over-partition large uniform areas.

In order to tackle these problems, a kinetic approach, called KIPPI (KInetic Polygonal Partitioning of Images), is proposed in the paper, which brings more flexibility on polygon shape and size. The key idea consists in "progressively extending pre-detected line-segments until they meet each other," [35]. Figure 2.9 shows an input-output example of this algorithm.



(a) original image (b) KIPPI result

Figure 2.9: Example result of KIPPI. Image copyright owned by [35]. The KIPPI algorithm decomposes the original image (a) into a partition of convex polygons shown in (b).

From the figure we can see that, different from general superpixel-based methods imposing homogeneously-sized regions, the polygons generated by KIPPI are "more meaningful, capturing both large components and thin lineic structures that compose, for instance, urban scenes," [35]. The paper's experiments demonstrate that output partitions both contain less polygons and better capture geometric structures than those delivered by existing methods.

Figure 2.10: Segmenting buildings in an aerial image using KIPPI. Image copyright owned by [35].

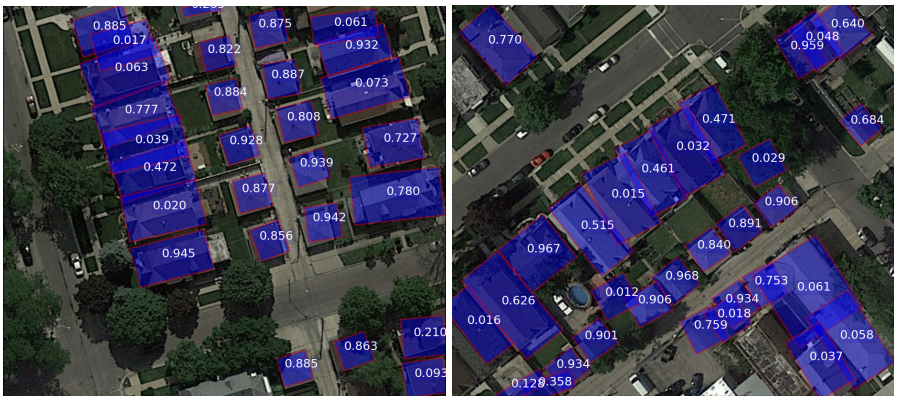
Paper [35] also show the applicative potential of the method when used as preprocessing in object contouring. To achieve polygonal object contouring from its partition, the paper associates each polygon with a binary activation variable indicating if it belongs to the objects of interest or not. The output

polygonal contours correspond to the set of edges separating active polygons from inactive ones, which ensures that the contours are closed by construction. Figure 2.10 shows its application in segmenting buildings in an aerial image.

2.2.3 Bounding Box Covering Approach

Additionally, the thesis work [26] proposes another approach, which we call bounding box covering approach. It utilizes the RPN in Faster R-CNN [16] and additionally adds a branch for calculating the orientation degree. It predicts the rotated bounding box to best cover each building instance in an aerial image. Thus, the problem here becomes a parameters regression problem, which is to find out the center coordinates, width, height, and the rotation degrees of the bounding box.

The model used here is the adapted RPN. As mentioned in subsection 2.1.2, the role of RPN in Faster R-CNN is to find RoIs, which are generally represented as bounding boxes or rectangles. This kind of representation can be very suitable for horizontally or vertically aligned buildings. But we know that buildings' orientations are not always regular, many of them are inclined to the image. Therefore, in this thesis project, RPN is adapted and another branch for rotating the bounding box is added. This can make the boundary of the bounding box closer to the building.



(a) (b)

Figure 2.11: Example result of bounding box covering approach. Image copyright owned by [26]. The numbers shown in the figure indicate the probability that the rotated bounding box contains an object.

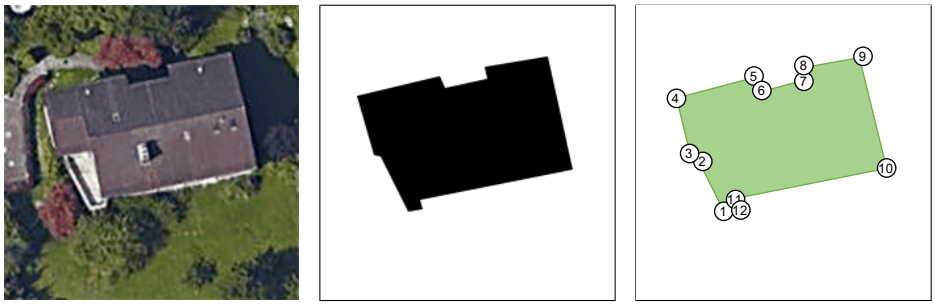
Figure 2.11 shows an example output. We can see from the figure that each building instance corresponds to a rotated bounding box, and is also limited tightly to the box. Results show that this method can detect buildings very

well, but the geometrical shapes found are limited to rectangles and cannot be more precise for buildings that are not rectangular in shape.

2.2.4 PolygonRNN

PolygonRNN [8] aims at exploiting geometrical shape for single object instance within an image. The model is originally proposed for speeding up manually contouring the object since it can achieve semi-automatic annotation of object instances, but the model can also be used for geometrical segmentation for single instance as well.

Most current methods regard instance segmentation problem as a pixel-wise classification problem, labeling each pixel as object or background (see figure 2.12(b) for example). Different from this traditional way, the paper treats the segmentation task as a polygon prediction problem. In particular, PolygonRNN takes the image of an object instance as input and sequentially produces vertices of the polygon outlining the object (see figure 2.12(c) for example).



(a) an example building (b) per-pixel mask (c) polygon representation

Figure 2.12: Comparison of pixel-wise mask and polygon. (a) is the original image containing a building. (b) is the target mask of traditional pixel-wise instance segmentation, (c) is the desired prediction of a polygon, of which the vertices are numbered.

We know that predicting a polygon is equivalent to predicting each of its vertices. Thus, the paper regards polygon as a series of vertices, and uses RNN as the model to make a coherent prediction. RNN is very powerful when data is related to time series as it can carry complex information about the history. In this case, the prediction of each vertex is dependent on the position of its two previous vertices. The paper also mentioned that another advantage over traditional methods is that RNN can capture the shape of the object even in ambiguous cases like shadows and saturation. In short, PolygonRNN can directly learn and predict the geometry of an object. For more details of the model architecture, please refer to section 3.1.

2.3 Motivation

Table 2.1 makes a summary of all models introduced in this chapter.

Table 2.1: Summary of related work.

Model	Detection	Geometry	Classification	End-to-end
FCN [12, 1]	Limited[1]	No	**Yes**	**Yes**
Mask R-CNN [11]	**Yes**	No	**Yes**	**Yes**
RPN [16] or FPN [7]	**Yes**	No	No	**Yes**
Graph-based [26]	**Yes**	Limited[2]	No	No[3]
KIPPI [35]	Limited[4]	**Yes**	No	No[5]
Adapted RPN [26]	**Yes**	Limited[6]	No	**Yes**
PolygonRNN [8]	No	**Yes**	No	**Yes**
What We Want	**Yes**	**Yes**	N/A[7]	**Yes**

From the table we can conclude that none of these models meets our requirements. Thus, combination of two or even more models becomes necessary. After observation, we propose a possible approach, simply replacing the mask branch in Mask R-CNN with PolygonRNN and removing the classification branch, so that the adapted model can predict polygon rather than per-pixel mask for each RoI. This model takes advantages from both models, thus can be applied to our problem. In practice, we combine FPN and PolygonRNN and name it a new model R-PolygonRNN (Region-based PolygonRNN), of which the architecture is shown in detail in section 3.3.

[1] As mentioned in subsection 1.1.2, object detection requires further processing of pixel connectivity detection.

[2] As mentioned in subsection 2.2.1, the thesis work does not provide the result for more complicated geometry, and in theory, even if we find the vertices of an instance, it requires further processing to determine their orders to form a polygon.

[3] As mentioned in paper [26], the whole model is a step-by-step process, each step is trained separately, so here we do not consider it is an end-to-end training.

[4] As mentioned in subsection 2.2.2, the object outline is form by edges separating active and inactive polygons. However, differentiating multiple instances requires further detection of edge connectivity in the graph.

[5] Since detection and geometrical segmentation are both based on the polygonal partition result, here we do not regard it as an end-to-end training.

[6] As mentioned in section 2.2.3, the shape is only limited to rectangle.

[7] Segmentation for roads is currently not considered in our project, thus the cell here shows "N/A", meaning not applicable.

Chapter 3

Model Architecture

In this chapter, the architectures of several models are presented. Section 3.1 gives a comprehensive explanation of the structure of PolygonRNN, while section 3.2 gives basic concepts in both single and multiple bounding box(es) regression and looks into FPN.

Considering our problem, we combine PolygonRNN and FPN together and come up with a new model, which is called R-PolygonRNN (Region-based PolygonRNN, see section 3.3). In theory, the proposed model can find out the bounding boxes for buildings within an aerial image and give geometrical shape for each building.

3.1 PolygonRNN

PolygonRNN is the core model for exploiting geometrical shapes in this thesis project. Figure 3.1 shows its simplified architecture, which consists of two parts. The CNN part (see subsection 3.1.1) can capture image features through multilayer convolutions and maximum pooling. Those captured features are then fed into the RNN part (see subsection 3.1.2), which sequentially predicts the spatial location of the new vertex with the highest probability at each time step.

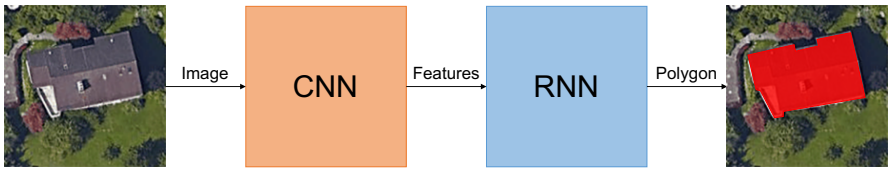


Figure 3.1: The simplified model architecture of PolygonRNN.

3.1.1 CNN Part

The CNN part of PolygonRNN uses VGG-16 [9]. Actually, VGG-16 is a form of VGGNet [9], which is very structured and focusing on deepening the neural network without a large number of parameters. It generally believes that deeper networks have stronger expressive capabilities than shallow networks, and can accomplish more complex tasks. It is also proven in practice that VGGNet has made great progress in performance compared to its previous network architecture (e.g. AlexNet [38]).

The "16" in VGG-16 means that it is a VGGNet with 16 layers containing parameters (13 convolutional layers and 3 fully connected layers). It has around 138 million parameters in total. Figure 3.2 shows its detailed network structure. From the figure we can see that VGG-16 continuously does a convolution with 3×3 small kernels and makes 2×2 maximum pooling. As the network deepens, the width and height of the image are reduced by half after each maximum pooling, and the number of channels is also doubly increasing after some convolutional layers.

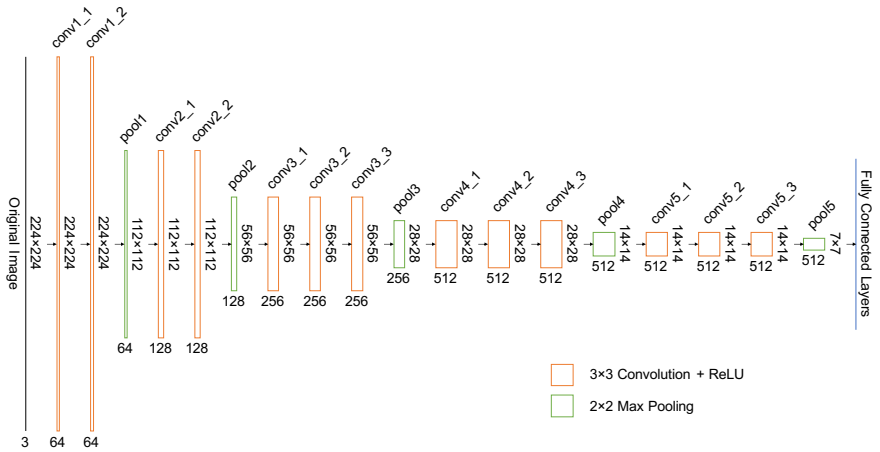


Figure 3.2: VGG-16 architecture. Only convolutional layers and maximum pooling layers are presented.

VGG-16 was mostly used for image classification before. Hence, after the convolutional layers and maximum pooling layers, there are fully connected layers and softmax layer for the class labels. However, these two kinds of layers are not required for VGG-16 used in PolygonRNN, because the CNN here is working as a feature extractor and mask predictor. Layer `pool5` is omitted as well because of the too low resolution.

Feature Extraction The modified VGG-16 provides RNN with useful features, which are taken from different convolutional and maximum pooling

layers. Thus, similar to FCN mentioned in subsection 2.1.1, the network here also uses skip connection. Specifically, the features are extracted from layers `pool2`, `pool3`, `conv4_3` and `conv5_3`. Note that since the resolution of final features is fixed, when taking features from layer `pool2` and `conv5_3`, it requires another maximum pooling and upsampling respectively. All of these processes can be seen in figure 3.3.

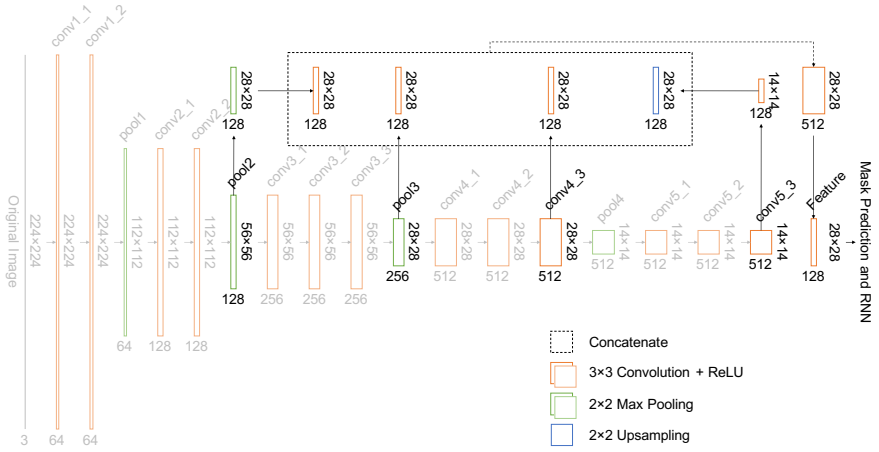


Figure 3.3: Modified VGG-16 architecture in PolygonRNN. In this figure, the highlight layers are used to extract features.

Mask Prediction Another function of the CNN part is to predict masks of boundary and vertices in a low resolution (one-eighth of the original). Figure 3.4 shows the mask prediction phase. Different from the ReLU (Rectified Linear Units) [39] non-linearity used in the former convolutional layers, the activation function used here is the general sigmoid function.

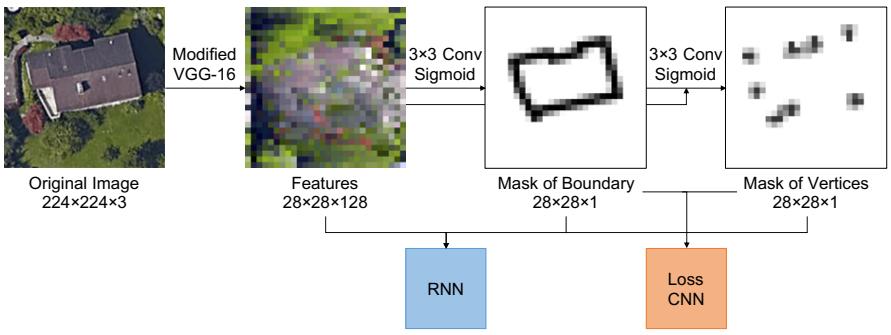


Figure 3.4: Mask prediction of VGG-16. Note that the mask of vertices is obtained by the convolution on the concatenation of the features and the mask of boundary.

Each entry of the boundary or the vertices mask indicates the probability that

the pixel is located in the boundary or is a vertex, respectively. The features and two masks are then sent into RNN together. In addition, the two masks are used to calculate the loss of CNN part. For the loss function used here, please refer to subsection 3.1.3.

3.1.2 RNN Part

The basic model used for predicting polygon vertices is RNN. We have already mentioned in subsection 2.2.4 that RNN is very powerful when data is related to time series, and in our case, we regard the polygon as a series of vertices.

As we know, given two adjacent vertices on a polygon in a specific order (either clockwise or anticlockwise), the third vertex after the two points can be uniquely determined. What RNN here can do is to predict the probability distribution of the next vertex's position when given the history information about the two vertices before. The calculation process also requires the image features and the position of the starting vertex, which is formulated as follows.

$$P(v_t \mid v_{t-1}, v_{t-2}) = F(v_{t-1}, v_{t-2}, v_0, f), \forall t \in \{2, 3, \ldots, T\}, \tag{3.1}$$

where f denotes the extracted features by CNN (including both masks of boundary and vertices as well), $F(\cdot)$ denotes a function (actually the simulation of RNN) for computing the conditional probability, T denotes the number of vertices of the polygon, v_t denotes the vertex position at t-th prediction. Specifically, the vertex prediction problem can be formulated as a classification problem. The position of v_t can be then quantized to the resolution of output grid, and we can thus use one-hot encoding for v_t's representation.

End Signal Note that the positions for v_t are not only limited to the general output grid, the end signal for the closure detection of the polygon is also embedded in v_t's assignments, just like the "end of sequence" token `<eos>` or `</s>` in RNN language model. Thus, the number possible assignments of v_t equal to the resolution of the output grid plus one ($28 \times 28 + 1 = 785$ in our case). In order to correctly predict the end signal, the starting vertex v_0 is required for the conditional probability (equation 3.1) calculation, as it tells the model when to finish the prediction phase. If the current prediction is the same as, or very close to the starting vertex v_0, v_t will be forced to raise the end signal, indicating that the entire polygon is close, and the prediction phase is therefore complete. So generally, the end signal works when $t = T$.

Starting Vertex In equation 3.1, two special cases $P(v_0)$ and $P(v_1 \mid v_0)$ are not included yet. These two cases are different from the general case, and should be considered in addition. In particular, we can directly regard the

mask of vertices (for example, the rightmost image in figure 3.4) predicted by the CNN as v_0's unnormalized probability distribution $\tilde{P}(v_0)$, and choose the position with the highest probability for v_0's assignment. As v_0 is known, there are typically two options for v_1, one is the next vertex on its left direction and another on the right direction. To tackle the problem of vertices ordering, we can simply specify the order of polygon vertices to be fixed, so that v_1 can be uniquely determined. In our project, the order of polygon is set to be anticlockwise, and in practice, we simply use equation 3.2 to get the conditional probability distribution of v_1 given v_0.

$$P(v_1 \mid v_0) = F(v_0, v_0, v_0, f). \tag{3.2}$$

ConvLSTM The RNN uses ConvLSTM (Convolutional LSTM) [40] cells as the polygon decoder to sequentially predict vertices. For simplicity, we can regard ConvLSTM as the function $F(\cdot)$ in equations 3.1 and 3.2. In fact, the structure of a ConvLSTM cell is almost the same as that of an ordinary LSTM (Long Short-Term Memory) [41] cell, except that it employs convolution in a 2D image with multiple channels instead of multiplication of matrices and vectors. The introduction of ConvLSTM can significantly reduce the number of parameters when compared with a fully connected RNN. The following equations define the computation process within a ConvLSTM cell, which is also visualized in figure 3.5.

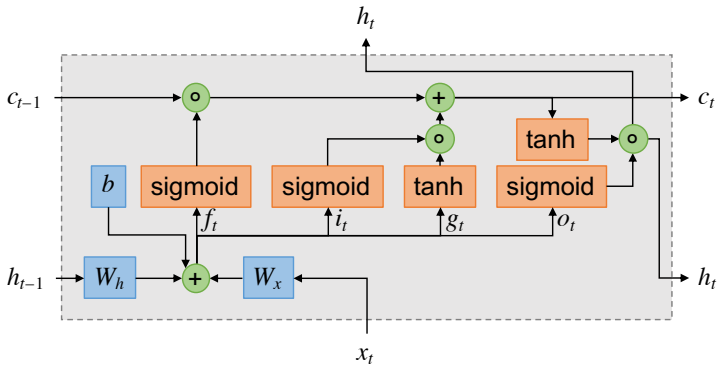


Figure 3.5: Visualization for an LSTM cell.

$$\begin{bmatrix} f_t \\ i_t \\ g_t \\ o_t \end{bmatrix} = \begin{bmatrix} W_{hf} \\ W_{hi} \\ W_{hg} \\ W_{ho} \end{bmatrix} * h_{t-1} + \begin{bmatrix} W_{xf} \\ W_{xi} \\ W_{xg} \\ W_{xo} \end{bmatrix} * x_t + \begin{bmatrix} b_f \\ b_i \\ b_g \\ b_o \end{bmatrix} = W_h * h_{t-1} + W_x * x_t + b, \tag{3.3}$$

$$c_t = \sigma(f_t) \circ c_{t-1} + \sigma(i_t) \circ \tanh(g_t), \tag{3.4}$$

$$h_t = \sigma(o_t) \circ \tanh(c_t), \tag{3.5}$$

where x_t, h_t, c_t denote the input, the hidden state (or cell output), and the cell state of the cell at time step t respectively, i_t, o_t, f_t denote the states of input, output, and forget gate at time step t respectively, g_t denotes an intermediate variable, $\sigma(\cdot)$, $*$, $\circ$ denote the sigmoid function, convolution, and Hadamard (element-wise) product respectively, W_x and W_h denotes two convolution kernels for x_t and h_t respectively, W_{xf}, W_{xi}, W_{xg}, W_{xo} denote the four components of W_x in the dimension of channels and W_{hf}, W_{hi}, W_{hg}, W_{ho} denote the four components of W_h in the dimension of channels.

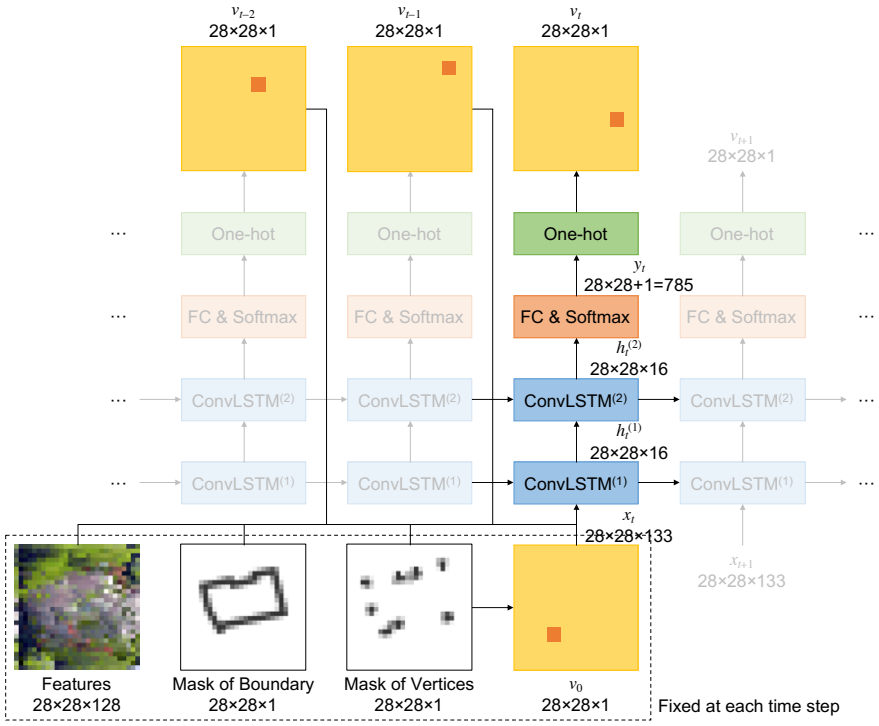


Figure 3.6: Visualization of RNN decoder in PolygonRNN with the time step. In this figure, the outputs of the end signal are omitted, and the configuration of the ConvLSTM cell is the same as what is used in the original PolygonRNN paper [8]. Specifically, the paper sets the number of RNN layers to 2 and uses ConvLSTM cells with 3×3 kernel size and 16 channels.

Multilayer RNN PolygonRNN uses multilayer RNN with ConvLSTM cells. The connection between two neighboring layers can be described as follows, which means that the cell input of the current layer comes from the cell output of the previous layer.

$$x_t^{(k)} = h_t^{(k-1)}, \forall k \in \{2, 3, \ldots, n\}, \tag{3.6}$$

where n is the number of RNN layers or ConvLSTM cells, k in the superscript denotes the k-th layer. Recall that the initial cell input (i.e. the cell input of the first layer) at time step t consists of the spatial information f from CNN (including the masks of boundary and vertices), the two previous vertices v_{t-1} and v_{t-2}, and starting vertex v_0, here it is formulated as follows.

$$x_t = x_t^{(1)} = v_{t-1} \oplus v_{t-2} \oplus v_0 \oplus f, \tag{3.7}$$

where $\oplus$ denotes the concatenation operation in the dimension of channels. Thus, the equation 3.1 can be written as follows.

$$y_t = P(v_t \mid v_{t-1}, v_{t-2}) = \text{softmax}(W_y \text{vec}(h_t^{(n)})), \tag{3.8}$$

where W_y denotes the weights of the final fully connected layer, $\text{vec}(\cdot)$ denotes the vectorization function for a matrix. Figure 3.6 shows the entire structure of the RNN part and highlights the process at time step t, using the same notation as the equations 3.6, 3.7 and 3.8.

Note that at each time step, the features, the masks of boundary and vertices as well as the one-hot encoding of starting vertex are fixed. Only v_{t-1} and v_{t-2} change over time.

3.1.3 Loss Function

The loss of PolygonRNN consists of two parts, loss of CNN and loss of RNN. The loss function of CNN part uses log loss, since the mask prediction is equivalent to binary classification for each pixel. We also notice that the non-boundary pixels or the non-vertex pixels occupy the majority, the loss function should be further weighted.

$$L(M, M^*) = \sum_{i=1}^{H} \sum_{j=1}^{W} \left[w_p \mathbb{1}[M_{ij} = 1] \log \frac{1}{M_{ij}^*} + w_n \mathbb{1}[M_{ij} = 0] \log \frac{1}{1 - M_{ij}^*} \right], \tag{3.9}$$

$$w_p = \frac{1}{2 \sum_{i,j} \mathbb{1}[M_{ij} = 1]}, w_n = \frac{1}{2 \sum_{i,j} \mathbb{1}[M_{ij} = 0]}, \tag{3.10}$$

where $L(M, M^*)$ denotes the loss function for the ground truth mask M and predicted mask M^*, the subscript i and j denotes the i-th row and j-th column, W and H denote the width and height of mask, $\mathbb{1}[\cdot]$ denotes the indicator function, w_p and w_n denote the weights for positive and negative pixels, respectively. Thus, the loss of CNN part L_{CNN} can be written as follows.

$$L_{\text{CNN}} = L(M_b, M_b^*) + L(M_v, M_v^*), \tag{3.11}$$

where subscript b and v denote masks of boundary and vertices, respectively. As for the loss of RNN, the loss function used is the cross-entropy loss, because the polygon prediction is equivalent to multiclass classification at every

time step. Thus, the loss of RNN part L_{RNN} can be written as follows.

$$L_{\text{RNN}} = \frac{1}{TK} \sum_{t=1}^{T} \sum_{k=1}^{K} \mathbb{1}[v_t = k] \log \frac{1}{y_{tk}}, \tag{3.12}$$

$$y_{tk} = P(v_t = k \mid v_{t-1}, v_{t-2}), \tag{3.13}$$

where $y_t k$ denotes the k-th component of y_t, i.e. the conditional probability of v_t at the position k, given its two previous vertices, T denotes the number of vertices in the polygon, K denotes the number of possible assignments (positions) for vertex v_t including the end signal. Here we do not consider the loss for the first vertex v_0, because it is randomly chosen from the polygon vertices when training. Finally, the total loss of PolygonRNN $L_{\text{PolygonRNN}}$ is defined as

$$L_{\text{PolygonRNN}} = L_{\text{CNN}} + \gamma L_{\text{RNN}}, \tag{3.14}$$

where γ is a self-defined weight.

3.2 Feature Pyramid Network

FPN [7] is the core model for object detection in this project. Object detection problem is exactly multiple bounding boxes regression problem, which finds out multiple RoIs within an image. Subsection 3.2.1 first illustrates the problem of single bounding box regression. Subsection 3.2.2 introduces some basic concepts related to anchor, and subsection 3.2.3 further illustrates multiple bounding boxes regression. Finally, subsection 3.2.4 looks into the backbone of FPN, presents how feature pyramid is generated and how FPN can tackle the multi-scale detection problem.

3.2.1 Single Bounding Box Regression

Single bounding box regression can be used for such a scenario, where the image contains only one object. Generally, a bounding box can be uniquely determined by four parameters, either box center and box size, denoting (x, y) and (w, h) or the coordinates of the upper left and lower right corners of the box, denoting (l, u) and (r, d). They have following relationships.

$$\begin{bmatrix} x \\ y \\ w \\ h \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 \end{bmatrix} \begin{bmatrix} l \\ u \\ r \\ d \end{bmatrix}, \tag{3.15}$$

$$\begin{bmatrix} l \\ u \\ r \\ d \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 \\ 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \\ h \end{bmatrix}. \tag{3.16}$$

Therefore, we can regard the problem of finding out object's bounding box as a parameter regression problem. With the image as input, either set of the four parameters, (x, y, w, h) or (l, u, r, d), can be selected as the target for the regression. Figure 3.7 shows the architecture of the network for single bounding box regression, using box center and size as outputs.

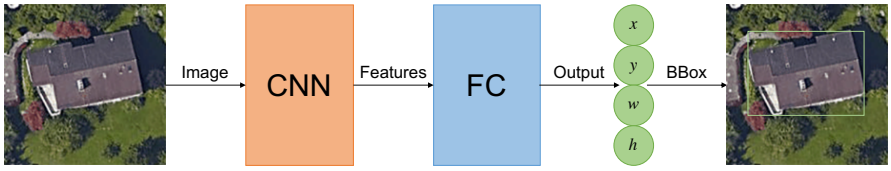


Figure 3.7: Simplified structure of the network for single bounding box regression. In this figure, the output of FC layer refers to the center and size of the bounding box.

In practice, usually the normalized parameters of box center and size rather than the coordinates of two corners are used as training targets. As for the normalization functions, please see subsection 3.2.3 for more details.

3.2.2 Anchor and Its Properties

The anchor is a basic concept in multiple bounding boxes regression. The following paragraphs introduce the idea of the anchor and its properties in detail.

Anchor An anchor refers to an artificially specified bounding box in the original image, regardless of the number of RoIs or where these RoIs locate. An arbitrary rectangular area in the image can be an anchor. For an image with width w and height h, the total number of possible anchors n can be calculated as

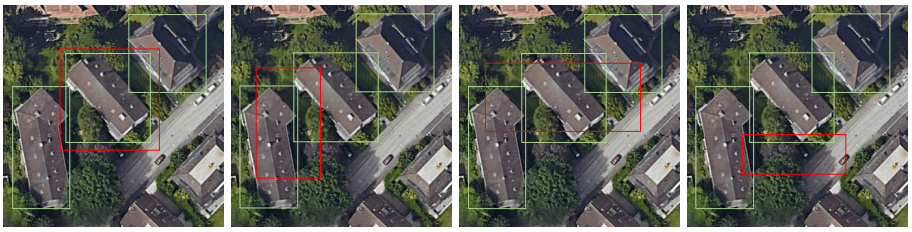
$$n = \frac{1}{4}w(w+1)h(h+1). \tag{3.17}$$

Thus, even for a very small image patch with size 224×224, n is already very large, reaching 635.04 million. Figure 3.8 illustrates some examples of anchors in an image with multiple RoIs.

IoU Score Usually IoU (Intersection over Union) score is used to evaluate the distance between an anchor and a ground truth bounding box. The following equation shows its calculation. Note that the score's computation is irrelevant to the actual image content in either anchor or the ground truth bounding box.

$$s = \frac{|A \cap T|}{|A \cup T|} \in [0, 1], \tag{3.18}$$

where s denotes the IoU score, $|\cdot|$ denotes the size of a set, A and T refer to the sets of pixels covered by the anchor and the ground truth bounding box



(a) positive (b) natural (c) natural (d) negative

Figure 3.8: Example anchors in an image with multiple RoIs. (a)–(d) show four possible anchors with each showing one. The red and light green rectangles refer to anchors and ground truth bounding boxes, respectively. The polarities of these anchors are also presented in the sub-captions, using the typical threshold.

respectively. $s = 1$ means that the anchor and the ground truth bounding box overlap perfectly. Figure 3.9 shows the coverage level under different IoU scores.

(a) IoU $= 0$ (b) IoU $= 0.3$ (c) IoU $= 0.7$ (d) IoU $= 1$

Figure 3.9: Coverage level of anchor and ground truth bounding box under different IoU scores.

Anchor Polarity Typically, there are several buildings in an aerial image, thus multiple RoIs exist. Based on the IoU scores of an anchor with these ground truth bounding boxes, we can make a judgment on the polarity of this anchor, which is defined as follows.

$$s_i = \max_j \text{IoU}(a_i, b_j), \tag{3.19}$$

$$p_i = \begin{cases} 1, & s_i > t_h, \\ 0, & t_l \leqslant s_i \leqslant t_h, \\ -1, & s_i < t_l, \end{cases} \tag{3.20}$$

where a_i and b_j denote the i-th anchor and j-th ground truth bounding box, s_i denotes the highest IoU score that a_i can get, i.e. the IoU score of a_i and its "closest" ground truth bounding box, p_i denotes the polarity of a_i, t_l and t_h denote two manually set thresholds for polarity judgment, and typically,

$t_h = 0.7$, $t_l = 0.3$. An anchor is labeled as positive, natural or negative in case of $p_i = 1$, $p_i = 0$ or $p_i = -1$, respectively. Figure 3.8 also shows anchors with different polarities using typical thresholds.

3.2.3 Multiple Bounding Boxes Regression

Multiple bounding boxes regression can be very different from the single one. Since the number of RoIs is unknown, if we still directly use the idea of parameters regression as what we do in case of the single bounding box, then the number of nodes of the FC output layer will not be fixed. In addition, the RoIs in the image usually reflect the local information, thus we cannot directly use all global features obtained by CNN, either.

Anchor Assignment The basic idea for multiple bounding boxes regression is to regress them on positive anchors. Specifically, each RoI is regressed on the anchors which are assigned to it. There are two rules for the assignment: (1) Each ground truth bounding box should have at least one anchor (either positive, natural or negative) assigned to it; (2) Each positive anchor should be assigned to its "closest" ground truth bounding box. For those non-positive anchors that have assignments, we also treat them as positive anchors. For details of the matching algorithm, please refer to section B.4.

Classification and Regression on Anchors As a matter of fact, the single bounding box regression can be also regarded as regression on the anchor. Here the anchor is exactly the whole image. Similar to this, we can simply increase the number of nodes in the FC output layer (see figure 3.7) according to the number of anchors. Generally, each anchor requires 6 output nodes, 2 for binary classification and 4 for parameters regression. Specifically, 2 classification nodes output the logits, denoting l_p and l_n, indicating how close the anchor is to a RoI. Thus, the predicted probability of an anchor referring to a RoI p^* can be calculated as follows, using softmax function.

$$p^* = \frac{e^{l_p}}{e^{l_p} + e^{l_n}}, 1 - p^* = \frac{e^{l_n}}{e^{l_p} + e^{l_n}}. \tag{3.21}$$

The rest 4 regression nodes output the information of the refined box, for recovering the corresponding RoI. Suppose we have a fixed positive anchor, which is assigned to a ground truth bounding box. The parameters of refined (or normalized) box are used for regression target, which can be calculated as

$$r_x = \frac{x_g - x_a}{w_a}, r_y = \frac{y_g - y_a}{h_a}, r_w = \log \frac{w_g}{w_a}, r_h = \log \frac{h_g}{h_a}, \tag{3.22}$$

where (x_a, y_a, w_a, h_a) and (x_g, y_g, w_g, h_g) denotes the four parameters of the anchor and the ground truth bounding box respectively, (r_x, r_y, r_w, r_h) denotes the refined parameters for regression target. The following equations

define the recovery phase.

$$x_g^* = x_a + r_x^* w_a, y_g^* = h_a + r_y^* h_a, w_g^* = w_a e^{r_w^*}, h_g^* = h_a e^{r_h^*}, \tag{3.23}$$

where $(r_x^*, r_y^*, r_w^*, r_h^*)$ and $(x_g^*, y_g^*, w_g^*, h_g^*)$ denote the predicted refinement parameters and the parameters of the final predicted bounding box respectively. The reason why the refined parameters are used for training, instead of original box, is that the former one eliminates the shape and location information of original box, hence it is more suitable for calculating the loss.

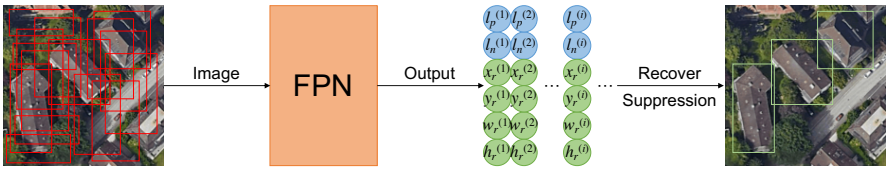


Figure 3.10: Simplified structure of multiple bounding box regression.

Loss As for training, only positive and negative anchors are used, where classification uses both and regression uses positive anchors only. The loss function used for classification is log loss, which is described as follows (for a single anchor).

$$L_c = \mathbb{1}[p = 1] \log \frac{1}{p^*} + \mathbb{1}[p = -1] \log \frac{1}{1 - p^*}, \tag{3.24}$$

where L_c is the classification loss, p is the polarity of the anchor, p^* is the predicted probability, $\mathbb{1}[\cdot]$ is the indicator function. Additionally, the loss used for regression is the smooth L_1 loss, which is described as

$$f(x) = \begin{cases} \frac{1}{2}x^2, & |x| < 1, \\ |x| - \frac{1}{2}, & |x| \geqslant 1, \end{cases} \tag{3.25}$$

$$L_r = f(x_g^* - x_g) + f(y_g^* - y_g) + f(w_g^* - w_g) + f(h_g^* - h_g), \tag{3.26}$$

where L_r is the regression loss. The total loss of FPN L_{FPN} is defined as

$$L_{\text{FPN}} = \sum_i L_c^{(i)} + \lambda \sum_i \mathbb{1}[p_i = 1] L_r^{(i)}, \tag{3.27}$$

where the superscript i denotes the i-th anchor, λ denotes a self-defined parameter when training.

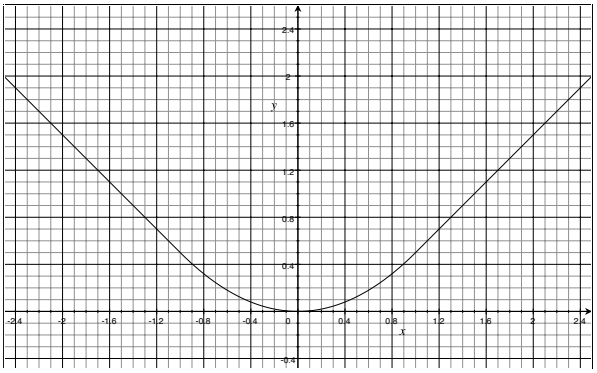


Figure 3.11: Smooth L_1 function.

3.2.4 Feature Pyramid and Backbone

From equation 3.17 we know that even a small image can have a huge number of possible anchors. However, it is impossible to involve all anchors in the training phase, since the output layer would have too many nodes and the number of parameters in the network would explode. Thus, how to select appropriate number of anchors in the original image remains a problem. But there is no doubt that the selected anchors should not "miss" any ground truth bounding box (i.e. should "cover" the ground truth bounding boxes as much as possible). Under this guideline, selected anchors should have different kinds of shapes, in different scales and evenly distributed in the image.

Anchor Selection in RPN In fact, in RPN from Faster R-CNN [16], there is a simple way to select anchors. For each entry of the convoluted feature map (or for each pixel of the original image with the corresponding stride), we generate k different shapes of anchors with different scales, which is shown in figure 3.12. For example, if we select anchors from an image with size 320×320, choosing stride 8×8 (corresponding to the feature map with size 40×40), 3 different shapes (1:1, $\frac{\sqrt{2}}{2}$:$\sqrt{2}$, $\sqrt{2}$:$\frac{\sqrt{2}}{2}$) and 4 scales (16, 32, 64, 128), then we have $40^2 \times 3 \times 4 = 19200$ anchors.

But this method produces two problems: (1) Large anchors are very close to each other, resulting in redundancy; (2) The stride may be too large for small anchors, thus a typically small ground truth bounding box, which is sandwiched between two small anchors, is very likely to be "missed". Actually, the second one is the reason why RPN cannot well detect small objects. However, both problems are tackled with the introduction of feature pyramid.

Anchor Selection in FPN Feature pyramid is a basic component in recognition systems for detecting objects. It actually refers to semantic feature maps at different scales. FPN exploits the inherent multi-scale, pyramidal hi-

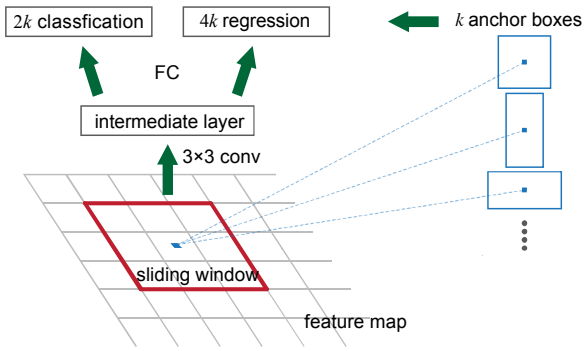


Figure 3.12: Anchor selection and regression part of RPN/FPN. Image copyright owned by [16].

erarchy of deep convolutional networks to construct feature pyramids. The network uses a top-down architecture with lateral connections, which is developed for constructing an in-network feature pyramid from a single-scale input. Figure 3.13 shows its architecture.

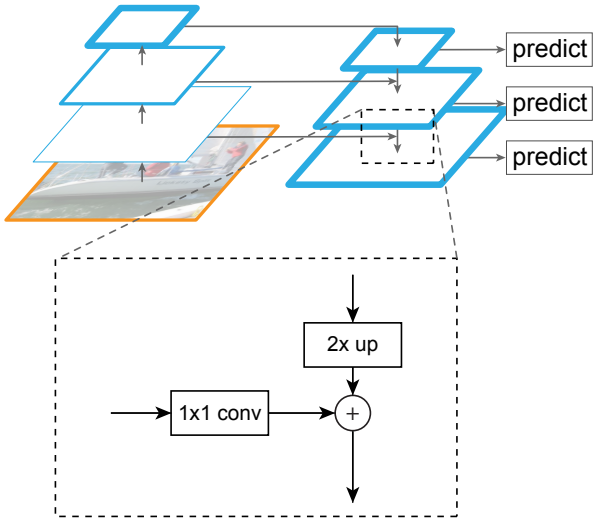


Figure 3.13: Structure of Feature Pyramid Network. Image copyright owned by [7]. The upper left part is the general convolutional process. The upper right part shows the feature pyramid. The lower part shows the lateral connections in detail.

The relationship between the feature pyramid and anchors is that each level of the feature pyramid corresponds to a single scale of anchors, which can be seen in figure 3.14. In general, we have

$$S_i = 2^{i-1} S_1, i \in \{1, 2, \ldots\}, \tag{3.28}$$

where S_1 denotes the smallest anchor scale (typically around 20, e.g. 16–24), S_i denotes the anchor scale corresponding to the feature map with 2^{-i} res-

olution. For example, for an image with size 320×320, if the smallest anchor scale is set to be 16, the anchor scale for the feature map with resolution 40×40 would be 128.

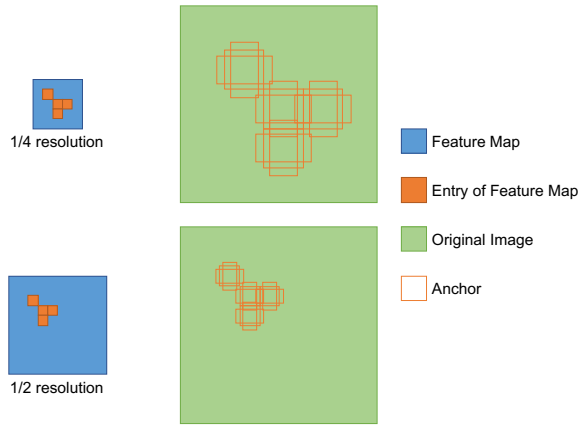


Figure 3.14: Generation of anchors from different layers of feature pyramid.

Therefore, in FPN, the feature map with larger resolution corresponds to smaller anchor scale, with smaller stride with regard to the original image. Similarly, the feature map with smaller resolution corresponds to larger anchor scale, with larger stride with regard to the original image. This is the core reason why the problems mentioned in figure 3.12 can be addressed. On the other hand, the total number of anchors n_l has an upper bound $\Theta(wh)$, which can be derived from

$$n_l = \sum_{i=1}^{l} \frac{w}{2^i}\frac{h}{2^i}k = whk\sum_{i=1}^{l} 4^{-i} = \frac{1}{3}whk(1-4^{-l}) < \frac{1}{3}whk, \tag{3.29}$$

where the subscript l denotes the number of layers (or feature maps) in the feature pyramid, i.e. the depth of feature pyramid, w and h denote the image width and height, k denotes the number of manually designed anchor shapes. Compared with equation 3.17, the total number of anchors is reduced by two orders of magnitude.

The Backbone of FPN The upper left part of figure 3.13 is the so-called backbone of FPN, which is exactly a general CNN. In Mask R-CNN [11], the backbone used for feature extraction is ResNet-101 [42], which can give excellent gains in both accuracy and speed. However, in our project, VGG-16 [9] is chosen as the backbone of FPN, because we want FPN and PolygonRNN share the same VGG-16. For more details about the shared VGG-16, please refer to subsection 3.3.2.

Figure 3.15 shows the architecture of FPN with VGG-16 backbone used in our project for feature pyramid extraction. Actually, the lateral connections

are not limited to a fixed method, which means that each layer of VGG-16 can be lead out using a reasonable connection. Here we use 4-layer feature pyramid, and features are taken from layers `conv3_3`, `conv4_3` and `conv5_3`, respectively. For the network to make the further prediction of refined box parameters, it is the same as what is shown in figure 3.12.

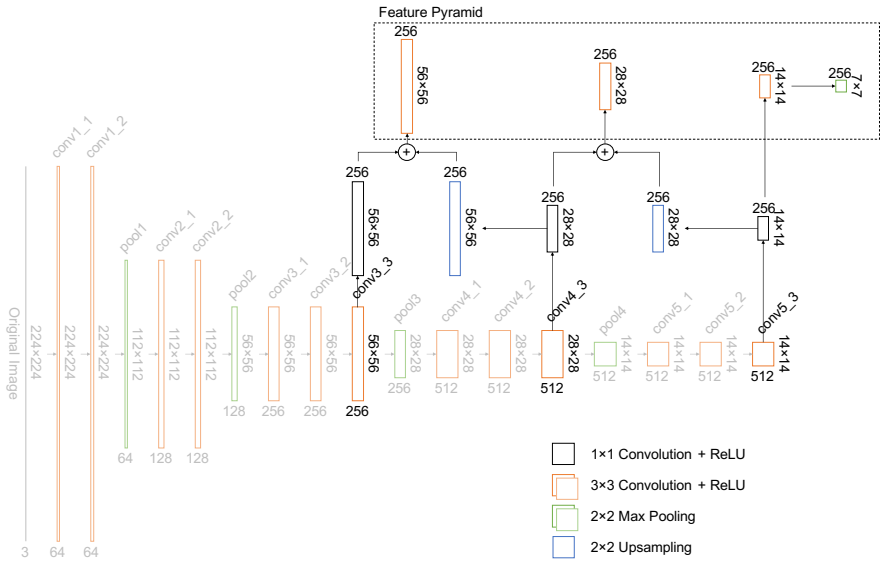


Figure 3.15: FPN with VGG-16 backbone.

3.3 R-PolygonRNN

In this section, the final model, R-PolygonRNN (Region-based PolygonRNN), is derived and introduced in detail. Here we propose 3 possible versions, denoting two-step version, hybrid version and hybrid version with RoIAlign, which are described in subsection 3.3.1, 3.3.2 and 3.3.3, respectively.

3.3.1 Two-step Version

The two-step version is very natural and easy to think of. In the first step, RPN is used to get the RoIs from the aerial image and each RoI contains a building. In the second step, PolygonRNN is used to get the geometrical shape. Thus, our problem is solved after these two steps. Figure 3.16 shows its pipeline.

From the figure 3.16 we can see that RPN and PolygonRNN are completely separate in this version. Therefore, during the training phase, we can train the two models separately and integrate them when making a prediction (also possible to train together).

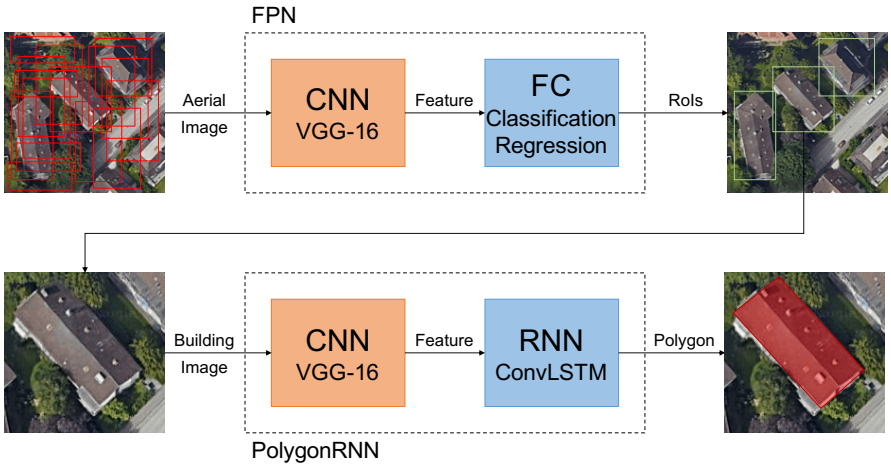


Figure 3.16: R-PolygonRNN, two-step version.

3.3.2 Hybrid Version

Note that in figure 3.16, either when the aerial image is taken as input by FPN or when the building image is taken as input by PolygonRNN, they both pass CNN first, and their CNNs both adopt VGG-16. In fact, regardless of the whole model of FPN or PolygonRNN, focusing on their respective CNN part, both features are extracted from the lateral connection on the basis of VGG-16. Although the features come from different levels, the backbone of the original VGG-16 is exactly the same for both of them. Therefore, they can share the same VGG-16 skeleton, just like what figure 3.17 shows.

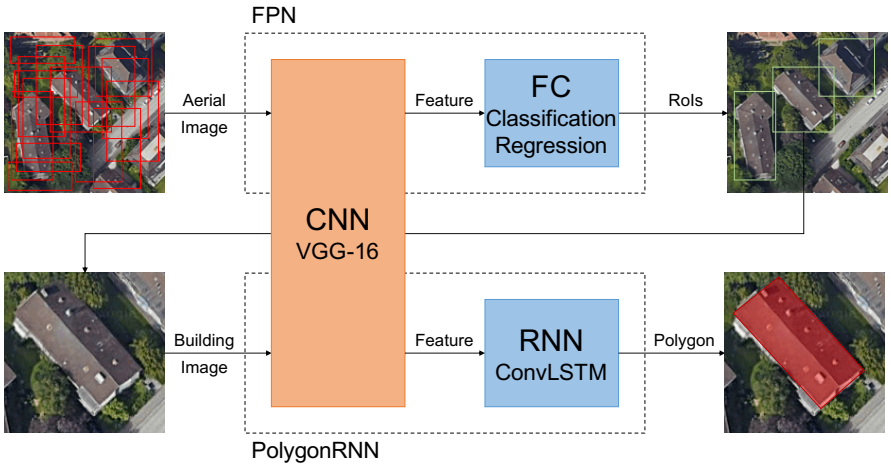


Figure 3.17: R-PolygonRNN, hybrid version.

As a result, the training phase of the whole model R-PolygonRNN cannot be separated, and multi-task training is needed in this version. In general, multi-task training introduces network parameter sharing, which can reduce the model parameters, make the training phase better, and therefore improve the model's prediction effect and accuracy. In addition, the model's generalization ability can be also improved through multi-task training.

3.3.3 Hybrid Version with RoIAlign

Note that both two-step version and hybrid version do convolution for the aerial image and the building image respectively. However, we know that the building image is taken from the aerial image, so there is redundancy in the twice convolutions. Considering that we already have the feature map of the aerial image, it is unnecessary to do a second convolution since we can take the building feature from the corresponding position in the feature map of the aerial image, when given the RoI.

In Mask R-CNN, RoIAlign [11] is proposed to solve this kind of problem. Actually, its previous version, RoIPooling [16] is already used in Faster R-CNN. The role of RoIPooling is to pool the corresponding area in the feature map into a fixed-size tensor, so as to perform subsequent classification phase. The positions of the RoIs are usually obtained by regression of the model, which are generally floating numbers. In order to get a fixed-size feature, RoIPooling has twice process of rounding these floating numbers: (1) Box boundaries are rounded to integers; (2) The region of the box is divided into several cells, boundaries of each cell are rounded to integers. Actually, we can also simply regard RoIPooling as a kind of nearest-neighbor interpolation. After the twice rounding process, each cell has a certain deviation from its initial position, which will have a negative effect on the accuracy of subsequent classification or semantic segmentation (only for Mask R-CNN). This problem is summarized as "misalignment" in Mask R-CNN paper.

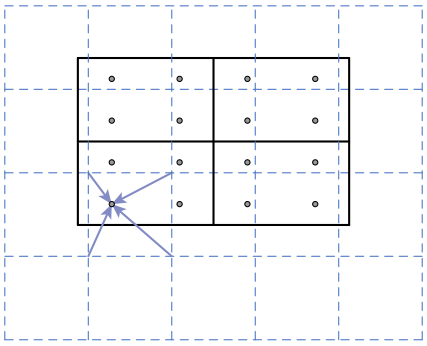


Figure 3.18: Example of bilinear interpolation in RoIAlign. Image copyright owned by [11].

In order to tackle the shortcomings of RoIPooling, Mask R-CNN introduces RoIAlign, using bilinear interpolation instead of rounding numbers. It can be used to obtain the value at the pixel whose coordinates are floating numbers, thereby transforming the pooling process into a continuous operation. Figure 3.18 shows an example of RoIAlign.

Therefore, we can simply apply the bounding boxes obtained from FPN calculation directly on the feature map by RoIAlign and send the pooled features to the RNN part of PolygonRNN. Figure 3.19 shows the model architecture of hybrid version with RoIAlign of R-PolygonRNN.

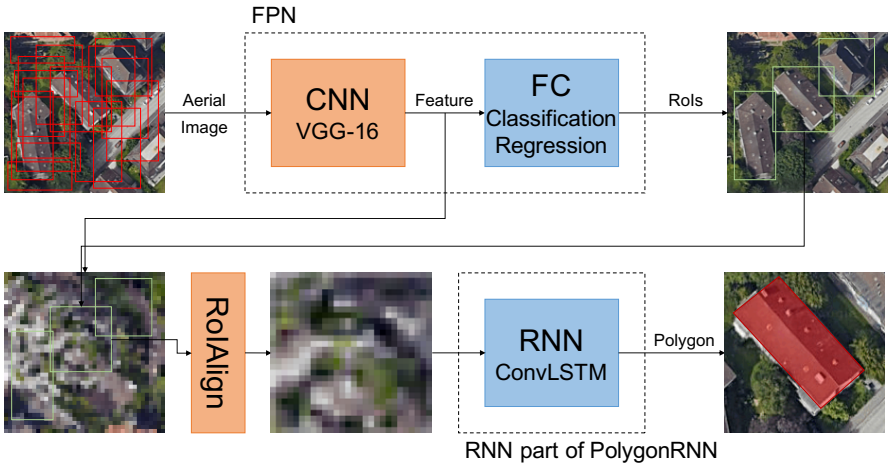


Figure 3.19: R-PolygonRNN, hybrid version with RoIAlign.

As a result, VGG-16 is used only once and therefore it can speed up both training and prediction in theory. However, in practice, we usually resize the aerial image into a low resolution before feeding it to FPN, in order to make the network faster. Thus, the feature pyramid we obtain is at an even lower resolution. If we directly use RoIAlign to get the building features, a lot of semantic information of the building would be lost. Hence, RoIAlign requires the input of aerial image with original resolution, and in this case, FPN would slow down. So here we think that this version will not have much improvement in speed. Therefore, due to the reason mentioned above and the time limitation of this thesis project, this version of R-PolygonRNN is not implemented yet, which would remain as future work.

3.3.4 Beam Search

In the prediction phase of R-PolygonRNN, the beam search algorithm is introduced, which is not used in the original paper [8]. Beam search is a heuristic graph search algorithm, which is usually used when the solution space of

a graph is relatively large. In order to reduce the space and time occupied by the search, at each step of depth expansion, some poor-quality knots are cut off, and higher quality nodes are kept. This process can significantly reduce space consumption and improve time efficiency, but the disadvantage is that, there may be potentially optimal solutions discarded. Therefore, the search result is not necessarily the optimal solution. Experiments show that the introduction of beam search can tackle the problem of the skipped vertex.

Figure 3.20 shows a single step of the beam search algorithm with beam width equaling to 3. For more details about beam search, please refer to paper [10].

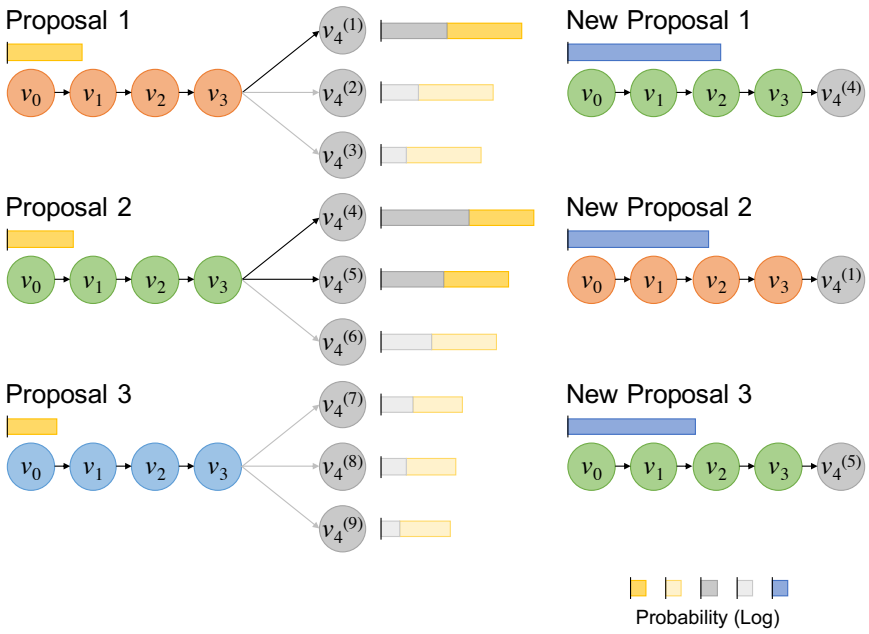


Figure 3.20: Example of a single step of beam search algorithm. Suppose currently we have three proposals for the first four vertices, with its corresponding probability (in yellow). For each proposal, we compute the probability distribution (in gray or light gray) for the fifth vertex, and present top three with the highest probability. Totally we have nine possible choices. For each choice, we compute its probability, which is the current one (in gray) plus (here using log probability, thus plus) the previous one. We take top three of these nine choices as our new proposals with new probability (in blue), and continue to the next loop.

Chapter 4

Experiments and Results

In this chapter, the entire experimental process is presented, from the acquisition of the ground truth dataset (see section 4.1), to the implementation, training and prediction phases of R-PolygonRNN (see section 4.2), and finally to the results evaluation and analysis (see section 4.3).

4.1 Ground Truth

Our ground truth dataset consists of aerial images and buildings' coordinates, which are used as inputs and labels respectively when training. In this project, all of the aerial images are collected from Google Static Maps API and all of the latitude and longitude coordinates of the polygon vertices of buildings are collected from OpenStreetMap. For details of the two APIs mentioned above, please refer to subsections 4.1.1 and 4.1.2.

As mentioned in section 3.3, the training phase of our model R-PolygonRNN requires two different kinds of ground truth dataset, areas with multiple bounding boxes for FPN and buildings with geometrical shapes for PolygonRNN, both of which are illustrated in subsection 4.1.3.

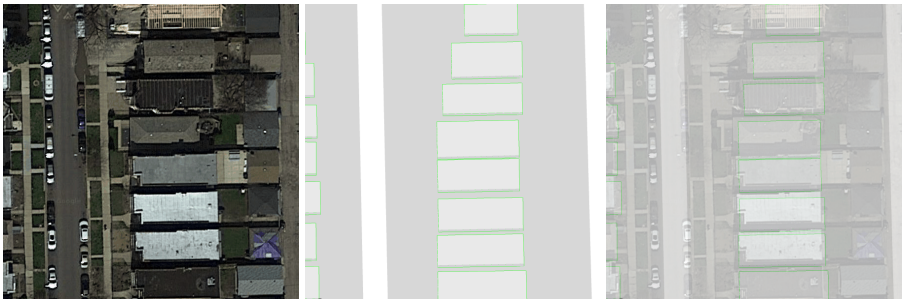
Since the whole dataset is collected from two different sources, the problem of inconsistency may exist. Subsection 4.1.4 describes details of the problems in the ground truth dataset, and subsection 4.1.5 proposes a feasible solution to adjust the shift between buildings' images and polygons.

4.1.1 Google Static Maps API

Google Static Maps API[1] provides an interface that implements maps as high-resolution images. Users can download customized map based on URL with different parameters, which is sent through a standard HTTPS request.

[1] `https://developers.google.com/maps/documentation/static-maps/`

The parameters in URL includes the map type (satellite, roadmap, etc.), latitude and longitude coordinates of the image center, the resolution of the image, the zoom level, and the scale. For the usage of the Google Static Maps API, please refer to section A.1 in appendices. Figure 4.1(a) and 4.1(b) shows two types of images can be obtained by Google Static Maps API.



(a) satellite image (b) roadmap image (c) hybrid image

Figure 4.1: Example images downloaded through Google Static Maps API. (a) and (b) are obtained directly by the API, with 41.9399708 degrees north latitude, 87.7380649 degrees west longitude of the center (located in Chicago), both width and height 600 pixels, zoom level 20 and scale 1, but different in map types. (c) is obtained by overlaying (a) with (b), with 75% opacity. It can be clearly seen that the two images cannot match very well.

We would have liked to obtain buildings' coordinates from Google Static Maps API as well, but we find that the API does not provide such information directly. Instead, it gives the styled roadmap images like figure 4.1(b), where querying coordinates requires further corner detection but the boundaries may be still inaccurate (see figure 4.1(c) for example). Thus, this thesis project, only satellite images from Google Static Maps API are used.

4.1.2 OpenStreetMap

OpenStreetMap[1] provides an interface that implements a map as a XML format `.osm` file. The file contains all the information which can be used to recover the map. When the map's bounding box is given, users can retrieve the latitude and longitude coordinates of the buildings' vertices and roads' central lines within the map. For the usage of OpenStreetMap and the format of the `.osm` file, please refer to section A.2 in appendices.

The coordinates obtained by the API is the original latitude and longitude coordinates of the buildings' vertices, which cannot be directly used for training. We need to convert them to relative pixel coordinates with regards to a given aerial image.

[1] `https://www.openstreetmap.org/`

As a matter of fact, every fixed point on the earth can correspond to a unique pixel on the map at a specific zoom level. This projection process can be regarded as a function. Since we have already known the relative pixel coordinates of the image center (half width and half height), the projection for an arbitrary point with given latitude and longitude coordinates can be therefore computed. This process can be described as

$$(c_x, c_y) = (\frac{w}{2}, \frac{h}{2}) = f(c_{\text{lon}}, c_{\text{lat}}, z) + (\Delta_x, \Delta_y), \tag{4.1}$$

$$(p_x, p_y) = f(p_{\text{lon}}, p_{\text{lat}}, z) + (\Delta_x, \Delta_y), \tag{4.2}$$

where f is the function that projects latitude and longitude coordinates to global pixel coordinates (see B.1 in appendices for more details), c is the image center, p is an arbitrary point, the subscripts x, y, lon, lat mean the relative pixel and longitude and latitude coordinates respectively, z is the zoom level, Δ is the translation constant from the global to the relative pixel coordinate system. Thus, we have

$$(p_x, p_y) = f(p_{\text{lon}}, p_{\text{lat}}, z) - f(c_{\text{lon}}, c_{\text{lat}}, z) + (\frac{w}{2}, \frac{h}{2}), \tag{4.3}$$

for any arbitrary point p.

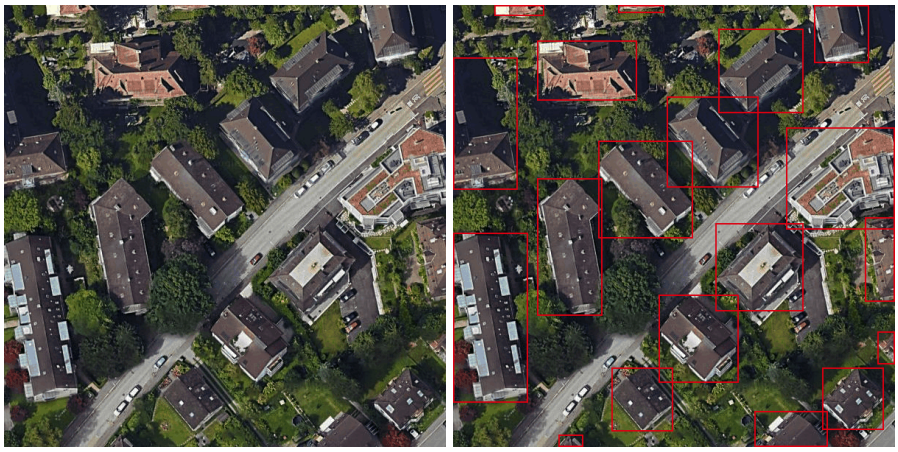
In addition, we keep the order of a polygon to be anticlockwise when training. However, the orders of coordinates of polygon vertices obtained from OpenStreetMap are not fixed. Thus, the reverse operation for those polygons with the clockwise order is required. Please refer to section B.2 in appendices for more details.

4.1.3 Areas and Buildings

As mentioned in section 3.3, R-PolygonRNN is formed by FPN and PolygonRNN. However, the training phase of these two models requires two different kinds of ground truth dataset.

Areas for FPN In subsection 3.2 we have shown that FPN aims at finding rectangular RoIs. Thus, training FPN generally requires a relatively large image with several objects in it. When considering our problem, an example of the ground truth can be an area of a city with several bounding boxes. Each box contains a single building, regardless of its tight polygon outline. Figure 4.2 gives an example area for training FPN and its visualized ground truth label.

Buildings for PolygonRNN In section 3.1 we have mentioned that PolygonRNN can only deal with images with a single object, meaning that the bounding box of the object should be first given. Thus, training PolygonRNN



(a) example area (b) example area with bounding boxes

Figure 4.2: An example area in Zurich for FPN training. (a) is the original aerial image obtained by Google Static Maps API. (b) is (a) covered by multiple bounding boxes of buildings, which is visualized based on the ground truth.

generally requires a relatively small image with a single object, as well as the information of its polygon outline. When it comes to our problem, an example of the ground truth can be a building with some padding and the pixel coordinates of building's vertices. Figure 4.3 gives example buildings for training PolygonRNN and its visualized ground truth label.

(a) example buildings

(b) example buildings with polygon outlines

Figure 4.3: Example buildings in Zurich for PolygonRNN training. Images in (a) are the aerial images obtained by Google Static Maps API. (b) is (a) covered by the corresponding polygon outline, which is visualized based on the ground truth.

4.1.4 Problems

Recall that the aerial images and polygons' coordinates are collected from two different sources, the problem of inconsistency may exist. That is to say, the polygon and the actual building we see in the image may not match well in some cases. This inconsistency is mainly reflected in the following aspects.

(a) general shift

(b) polygon size and shape

(c) angle issue

(d) adjacent buildings

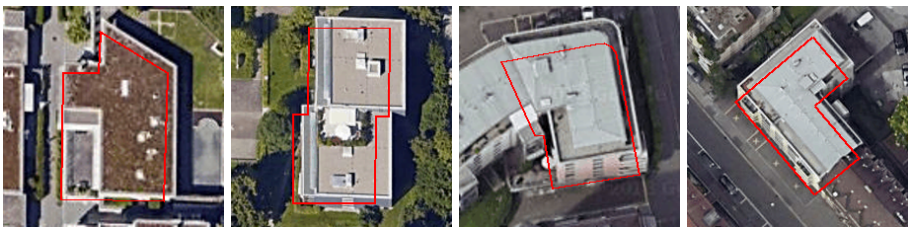
Figure 4.4: Problems existed in the ground truth dataset.

- **General Shift**: Figure 4.4(a) gives two typical shift examples in the dataset. The shift is generally within the range of 30 pixels, either up and down, or left and right.
- **Polygon Size and Shape**: Sometimes the polygon size and shape is different from the actual building size as well. This is typical because OpenStreetMap measures the bottom of the building and the aerial image is taken from the top. The roof of the house we see from the top can be different from the floor area. Figure 4.4(b) shows a building with a smaller polygon and a building with a too rough outline.
- **Angle Issue**: Some tall buildings may suffer from the angle issue. This is because the aircraft that takes the image cannot always be vertically right above the building, so the side of those very high buildings would also be photographed. Figure 4.4(c) gives two examples.
- **Adjacent Buildings**: Some adjacent buildings will be very close together, and some even share a common roof. Thus, from the aerial image, it is generally difficult to distinguish the boundaries between

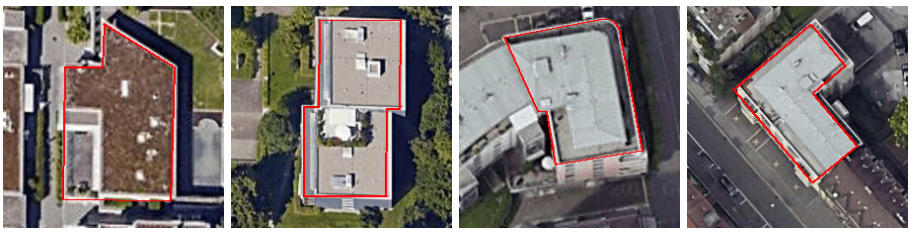
them. However, these buildings are separated in the dataset of OpenStreetMap. It means that where there are originally edges between buildings, it looks like there is no edge from the view of the aerial image. This kind of inaccuracy would have a negative effect on the training phase. Figure 4.4(d) and gives two examples.

4.1.5 Adjustments

Problems such as too rough polygon shape, angle issue and adjacent buildings are unsolvable unless the data sources can correct themselves. What we can improve is to tackle the general shift and the polygon size problem and make the image-polygon pairs more matching. By observation, we can get to some conclusions for a matching image-polygon pair.



(a) before adjustment



(b) after adjustment

Figure 4.5: Adjustment examples. (a) shows the polygons before adjustment, while (b) shows the polygons after adjustment.

- **Color Variance**: The color variance of image pixels covered by polygon generally reaches the local minimum, because the roof of a building usually has only one color or several similar colors.
- **Edges**: The polygon edges generally lie within the output of the edge detection (e.g. Sobel, Canny) performed on the image.
- **Corners**: The polygon vertices can generally match the output of the corner detection (e.g. Harris, SIFT) performed on the image.

Based on these conclusions, we propose a brute force shift adjustment method. Briefly, we resize and translate the initial polygon within a certain range, and

traverse all the cases to see where the minimum color variance, maximum edge and corner coverage are reached. The detailed algorithm, please refer to section B.3 in appendices. Figure 4.5 illustrates some examples, which compare the polygons before and after the adjustment. Results show that this algorithm can well address the shift problem to some extent, which guarantees the correctness of the training set.

4.2 Implementation Details

In this section, several implementation details are given in order to the replicate the model in the future if needed. Subsection 4.2.1 gives information and some notes of the dataset. Subsection 4.2.2 gives the basic configuration of the model. Subsection 4.2.3 and 4.2.4 focus on the training and prediction phase respectively.

4.2.1 Dataset Information

The dataset consists of two cities, Zurich and Chicago, including most of the buildings in the range shown in table 4.1. In addition, aerial images are taken based on the information shown in table 4.2.

Table 4.1: Sampling range of buildings in Zurich and Chicago.

City	Zurich	Chicago
East Boundary	E 8.4092916°	W 87.8039744°
South Boundary	N 47.2879518°	N 41.8799477°
West Boundary	E 8.6729490°	W 87.6721554°
North Boundary	N 47.4664863°	N 41.97799394°
Number of Buildings	96,573	228,074
Size of Training Set	86,915	205,266
Size of Test Set	9,685	22,808

There are some notes for buildings should be specified: (1) Buildings with more than 19 or less than 4 vertices are excluded; (2) All of the building images are squares but in different sizes, and include paddings; (3) As mentioned in subsection 4.1.2, the order of all polygons are set to be anticlockwise; (4) In each polygon, three consecutive vertices cannot be collinear. If so, the second vertex is removed.

There are some notes for areas should be specified: (1) Areas without any buildings are excluded; (2) There is an overlap between neighboring areas because each building is required to appear completely at least once within all areas. For example, in figure 4.2(a), for the incomplete buildings located near the image top edge, we can find them with complete shapes in its northern neighboring area.

Table 4.2: Sampling range of areas in Zurich and Chicago.

City	Zurich	Chicago
Center Longitude	E 8.5468221°	W 87.7380649°
Center Latitude	N 47.3768587°	N 41.9289708°
Step Longitude	$4.023094° \times 10^{-4}$	$4.469772° \times 10^{-4}$
Step Latitude	$2.724221° \times 10^{-4}$	$3.324594° \times 10^{-4}$
Horizontal Step Range	$(-144, 144)$	$(-144, 144)$
Vertical Step Range	$(-144, 144)$	$(-144, 144)$
Zoom Level	19	20
Scale	1	1
Image Size	$(600, 600)$	$(600, 600)$
Number of Areas	57,741	77,716
Size of Training Set	51,966	69,944
Size of Test Set	5,775	7,772

4.2.2 Model Configuration

Table 4.3 shows the parameters used to define the graph of the model. The environment of implementation of our model is Python 3.6.1, with TensorFlow 1.6.0[1].

Table 4.3: Configuration parameters used to define the model graph.

Item	Value
Area Image Resizing	$(256, 256)$
Building Image Resizing	$(224, 224)$
Resolution of Output Grid	$(28, 28)$
RNN Max Sequence Length	20
RNN Number of Layers	3
ConvLSTM Number of Output Channels	$32, 16, 8$
Feature Pyramid Number of Layers	4
Anchor Scale	$16, 32, 64, 128$
Anchor Shape	$\frac{1}{2}$:2, $\frac{\sqrt{2}}{2}$:$\sqrt{2}$, 1:1, $\sqrt{2}$:$\frac{\sqrt{2}}{2}$, 2:$\frac{1}{2}$
Feature Shapes	64×64, 32×32, 16×16, 8×8
Feature Stride	$4, 8, 16, 32$
Total Number of Anchors	27,200

4.2.3 Training Phase

During the training, buildings near the edges of aerial images are typically ignored since they are more likely to be incomplete. In this case, the polygon

[1] `https://www.tensorflow.org/versions/r1.6/`

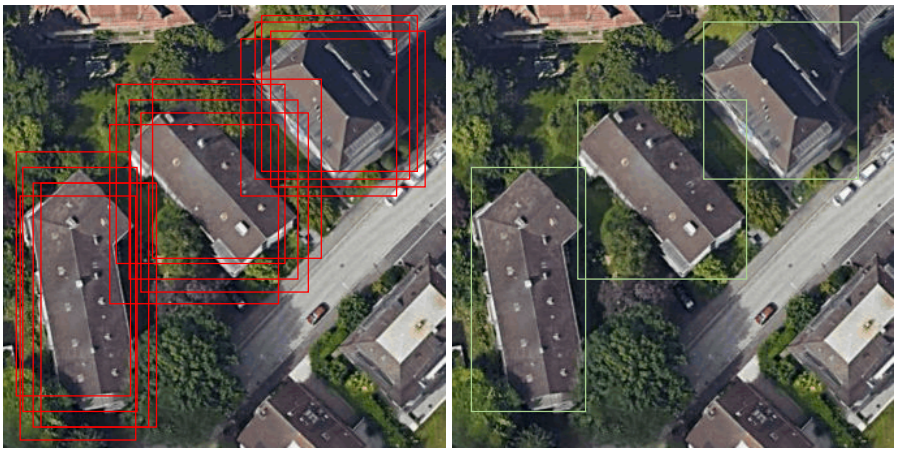
would be clipped by the edges, resulting in new vertices, which can be very hard to compute. Therefore, in this project, two different kinds of ground truth dataset are collected separately, which means that in a specific round of training, the building images come from the dataset instead of patches from the aerial image. Training in this way can keep the batch size of buildings unchanged. In addition, we perform data augmentation on the training set. We rotate the building image and its polygon by 0°, 90°, 180° or 270° in order to create more training set and make model more robust in different situations. Table 4.4 shows the configuration parameters during the training phase of the model.

Table 4.4: Configuration parameters used in the training phase.

Item	Value
Optimizer	Adam
Learning Rate	1×10^{-4}
Area Batch Size	4
Building Batch Size	12
Number of Rounds	25,000

4.2.4 Prediction Phase

Recall subsection 3.2.3, an object bounding box may be regressed on multiple anchors, resulting in duplicate boxes. In order to get the best bounding box of an object, NMS (Non-Maximum Suppression) algorithm is adopted. NMS prunes away boxes that have high IoU overlap with previously selected boxes, thus the "best" boxes can remain.



(a) before NMS (b) after NMS

Figure 4.6: Example of NMS algorithm.

Figure 4.6 shows an example of NMS algorithm. For more details about the algorithm, please refer to the paper [43], which includes both NMS and its improved version Soft-NMS.

In practice, during the prediction, we first select a number of anchors with the highest (e.g. top 500) probability, and then remove anchors with probability lower than a threshold (e.g. 0.99). The selected anchors are then fed into NMS algorithm, which outputs the final RoIs. These RoIs are further taken as input of PolygonRNN to predict their geometrical shapes. Note that too small bounding boxes obtained by FPN are typically ignored and would not be used for polygon prediction. Table 4.5 shows the configuration parameters during the prediction phase of the model.

Table 4.5: Configuration parameters used in the prediction phase.

Item	Value
Beam Width	1 or 6
Top Number of Anchors	500
Anchor Probability Threshold	0.99
NMS Max Output Size	40
NMS IoU Threshold	0.15
Minimum Bounding Box Area	16×16

4.3 Experiment Results

In this section, the experiment results are presented. We evaluate and analyze these results from different aspects. Subsection 4.3.1 focuses on loss decrease of both versions of the model. Subsection 4.3.2 records the time consumption in both training and prediction phases. Subsection 4.3.3 uses kinds of metrics to evaluate the prediction results. Subsection 4.3.4 presents final prediction results of geometrical instance segmentation in aerial images, and makes comparison.

4.3.1 Loss Decrease

Figure 4.7 and figure 4.8 show the loss decrease for all kinds of aspects in dataset of Chicago and Zurich, respectively. Note that in both figure, the loss of anchor classification, anchor regression, per-pixel mask prediction, and polygon vertices prediction are the 4 basic losses. The loss of FPN is calculated as the mean of anchor classification and regression losses, while the loss of PolygonRNN is calculated as the mean of mask and polygon prediction losses. The full loss is defined as the the mean of FPN loss and PolygonRNN loss. In the figures 4.7 and 4.8, the losses for two-step version are in shallow colors, while the losses for hybrid version are in deep colors.

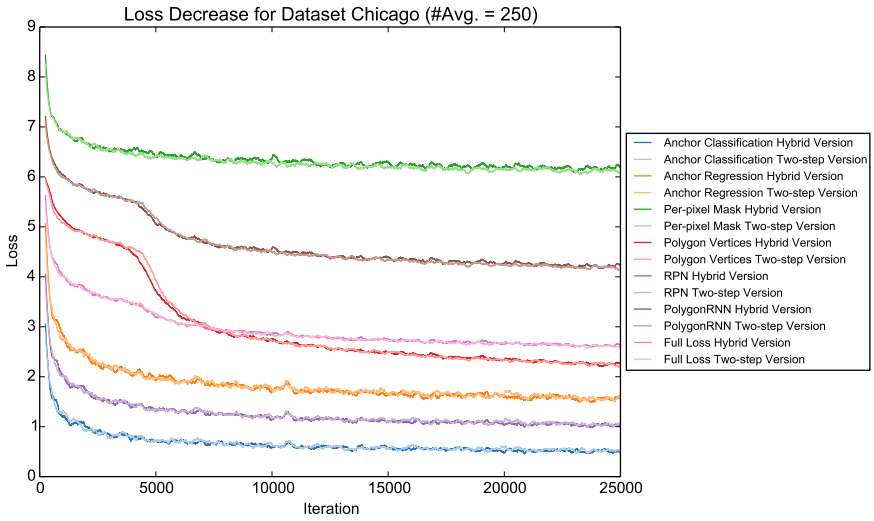


Figure 4.7: Loss decrease of the model in dataset Chicago.

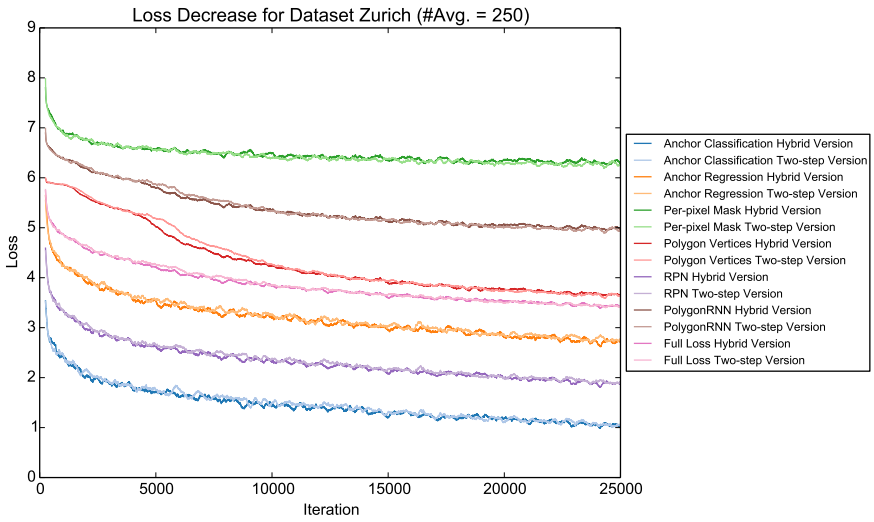


Figure 4.8: Loss decrease of the model in dataset Zurich.

We expect to draw some interesting conclusions from the comparison of the results of different cities or different models. From both figures we can conclude that the introduction of shared VGG-16 does not significantly accelerate the training phase of the network. The only change is that, the loss of RNN part (polygon vertices prediction) of PolygonRNN in the hybrid version falls earlier than the two-step version around the iteration 5,000.

As for different cities, we find that the loss of Chicago decreases faster and finally becomes lower than the loss of Zurich. We consider that it is because

most of the buildings in Chicago are vertically or horizontally aligned, thus it is easier for the machine to learn.

In addition, we found that after completing 25,000 rounds on the training set of Zurich, the loss is still on the decline. However, because it has been trained for a long time and we want to compare it with Chicago under the same criterion, there is no further training for Zurich.

4.3.2 Time Consumption

We use the GPU resources from the Leonhard cluster of ETH Zurich to train our model and make a prediction. The overall training time is around 48 hours for totally 25,000 rounds for a single model in the dataset of each city.

Table 4.6 gives the time consumption for each batch of different models in the different dataset when training. From the table we can conclude that the hybrid model has a negligible increase in training speed.

Table 4.6: Time spent by each batch in training phase.

City	Zurich	Chicago
Two-step Version	7.237s	7.150s
Hybrid Version	7.214s	7.074s

Table 4.7 gives the time consumption of different models in the different dataset when prediction. From the table we can conclude that no matter using what model or which dataset, they do not have an obvious impact on the prediction time.

Table 4.7: Time spent in prediction phase.

Stage		FPN		PolygonRNN[1]	
Model	**BW**[2]	Chicago	Zurich	Chicago	Zurich
Two-step	6	1.414s	1.421s	0.358s	0.358s
Version	1	1.428s	1.386s	0.134s	0.128s
Hybrid	6	1.459s	1.458s	0.358s	0.360s
Version	1	1.451s	1.445s	0.132s	0.131s

4.3.3 Evaluation Metrics

Since the output of the model is an ordered sequence of pixel coordinates, it is difficult to find a suitable evaluation metric to quantize the degree of

[1]Here the prediction time refers to the average time consumption of polygon extraction for a single building.

[2]BW refers to beam width.

fitting between two polygons. Thus, the evaluation metrics we used are still traditional and pixel-based, of which we choose the weighted accuracy and F1 score. We also use IoU score, which can indicate the overlapping extent between two sets of pixels. The evaluation result with internal comparison can be seen in table 4.8.

Table 4.8: Evaluation of models with internal comparison.

Metric	**City**	**Model**			
		BW1[1]		BW6[1]	
		2-Stp.[1]	Hyb.[1]	2-Stp.[1]	Hyb.[1]
Accuracy	Chicago	0.9043	0.8986	**0.9105**	0.9084
	Zurich	0.7856	0.8020	0.8129	**0.8211**
F1 Score	Chicago	0.8672	0.8591	**0.8777**	0.8730
	Zurich	0.6711	0.7033	0.7171	**0.7316**
IoU Score	Chicago	0.7922	0.7822	**0.8037**	0.7985
	Zurich	0.5486	0.5769	0.5937	**0.6069**

From table 4.8 we can see that the introduction of beam search improves the final result in all cases, especially for city Zurich using the IoU score evaluation. We also find that the two-step version does well on the dataset of Chicago, and the hybrid version does well on the dataset of Zurich. However, on the dataset of Chicago, two models have a just slight difference, thus we still consider the hybrid version performs better. The evaluation result with existing methods (external comparison) can be seen in table 4.9.

Table 4.9: Evaluation of models with existing methods.

Metric	**City**	**Model**			
		Hyb., BW6[1]	Paper [8]	Paper [1]	Thesis [26]
Accuracy	Chicago	0.9084	N/A	N/A	N/A
	Zurich	0.8211			
F1 Score	Chicago	**0.8730**		0.837[2]	0.70
	Zurich	0.7316		**0.823**[2]	N/A
IoU Score	Chicago	**0.7985**	0.52–0.72[3]	N/A	
	Zurich	0.6096			

From table 4.9 we can see that our model outperforms the paper [1] and the

[1] BWx refers to beam width x. 2-Stp. refers to "Two-step version". Hyb. refers to "Hybrid version".

[2] The dataset used by paper [1] has lower zoom level than ours, which means that its dataset "is collected from a higher altitude", and does not contain much building details (buildings are very small).

[3] The IoU score here is for the cityscapes dataset (e.g. cars, trains, bicycles) in the original paper [8], **NOT** for the dataset of aerial images of Zurich or Chicago.

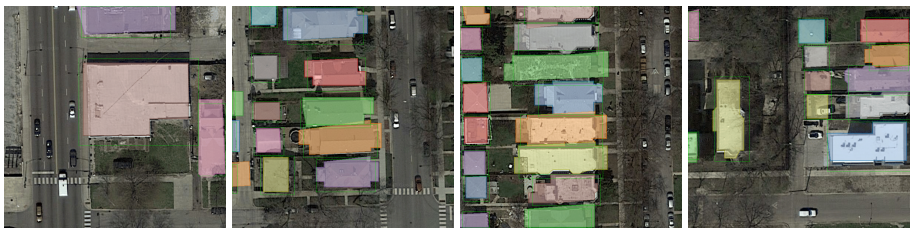
thesis [26] with F1 score evaluation in the dataset of Chicago. As for the IoU evaluation, in the original paper of PolygonRNN, the IoU score typically falls into 0.52–0.72 for the cityscapes, thus our result of dataset Zurich is just normal. The reason why Chicago's IoU score is relatively high is that most of the buildings are vertically or horizontally aligned.

4.3.4 Prediction Results

We first make a comparison in terms of beam search, using the model with the hybrid version, which is presented in figure 4.9. Please zoom in for more details.

(a) Zurich, with beam search

(b) Zurich, without beam search



(c) Chicago, with beam search

(d) Chicago, without beam search

Figure 4.9: Comparison of results in Zurich and Chicago in terms of beam search.

From the figures we can conclude that beam search makes a certain improvement in the dataset of Zurich, indeed, especially for the buildings with complicated shapes. However, such improvement is not obvious for Chicago.

Figure 4.10 and figure 4.11 presents the more prediction results on the test set of Zurich and Chicago respectively, using the hybrid version, beam width 6. Please zoom in for more details.

Figure 4.10: Prediction results on the test set of Zurich. Note that multiple colors are used here to differentiate building instances.

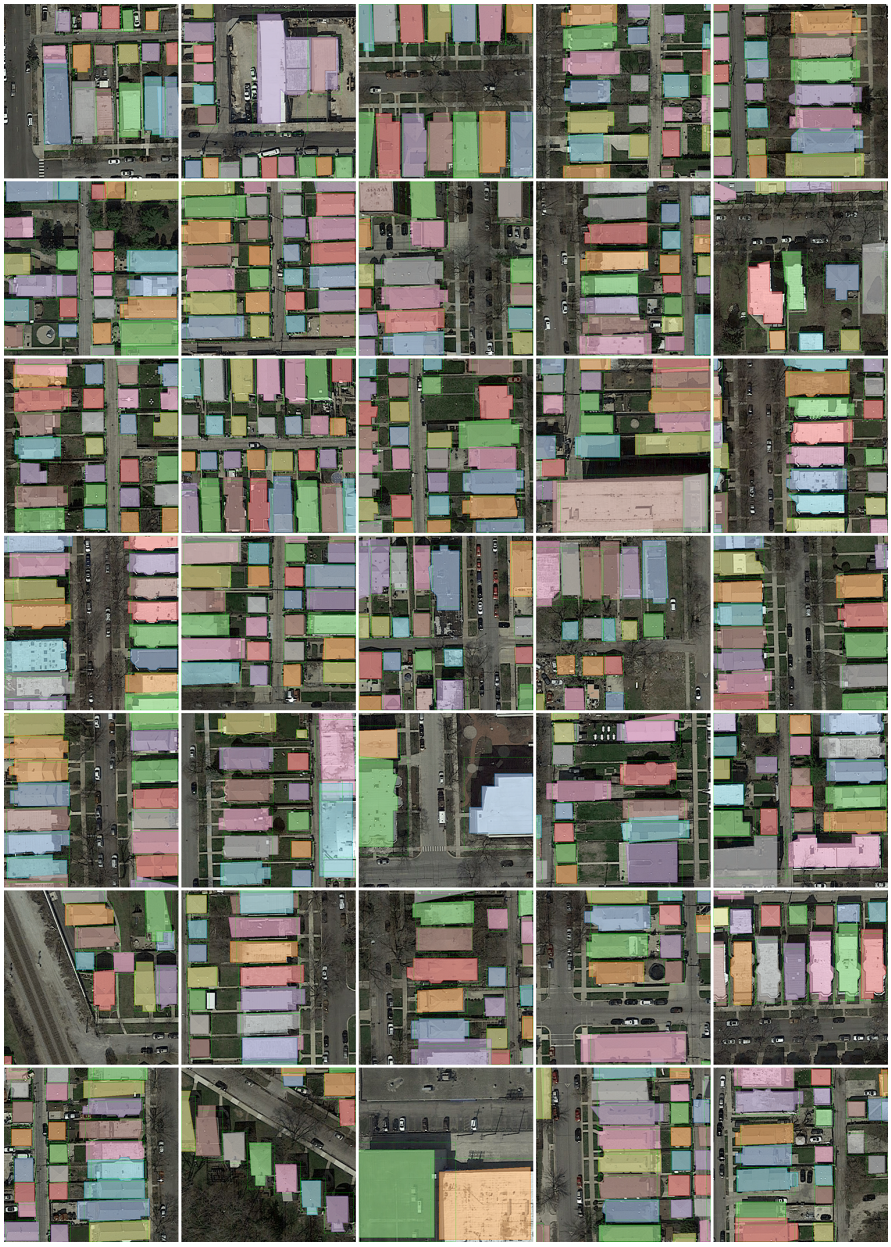


Figure 4.11: Prediction results on the test set of Chicago. Note that multiple colors are used here to differentiate building instances.

In summary, our new proposed model R-PolygonRNN can well outline the buildings in a given aerial image and can give details. For the problems existed in the prediction results, please refer to section 5.1. For more prediction results, please refer to sections C.1 and C.2 in appendices.

Chapter 5

Problems and Future Work

This chapter focuses on problems and future work. Section 5.1 presents some flaws in the prediction results, along with the corresponding possible solutions. Section 5.2 shows the future direction of our work.

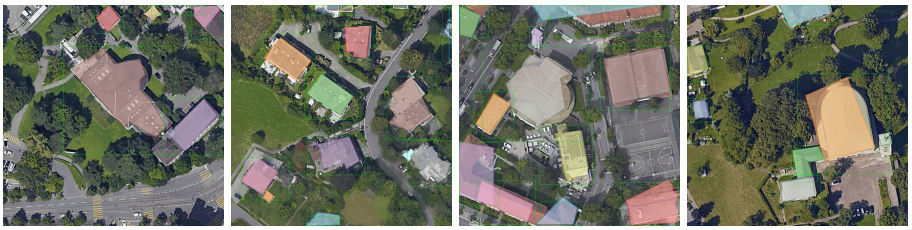
5.1 Problems

In this section, two existed problems in the prediction result are illustrated, and for each problem, we propose potential solutions respectively. Subsection 5.1.1 shows the resolution problem resulting in the inaccurate prediction and subsection 5.1.2 shows the problem of the false vertex.

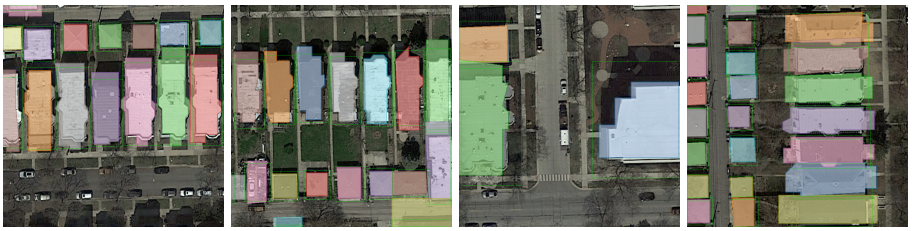
5.1.1 Output Resolution

We know that the output grid for a single vertex has the resolution 28×28, and the resolution of input images of PolygonRNN is 224×224. Thus, a single pixel in the output grid corresponds to an area of 8×8 in the original image. This will result in blurs on the feature map and loss of details, therefore outputting inaccurate vertices. Figure 5.1 gives some examples, where the prediction of those round or semicircular buildings in Zurich is very inaccurate, and the prediction of buildings with small bulges in Chicago cannot well match the edges of buildings in the original image. Please zoom in for more details.

The possible solution is to increase the resolution of the output grid, and to extract features from relatively shallow levels. However, these shallow layers may not have much semantic information, which may further have a negative effect on the prediction accuracy of RNN part. In addition, because of the existence of FC layer at the end of the output of RNN part, the number of its weights would increase at the fourth power, i.e. $\Theta(w^2h^2)$. For example, currently, the FC layer has $(28^2)^2 = 614{,}656$ weights, if we increase the reso-



(a) resolution problem in the dataset of Zurich



(b) resolution problem in the dataset of Chicago

Figure 5.1: Examples of resolution problem in Zurich and Chicago.

lution of the output grid to 56×56, the number of weights would be $(56^2)^2 =$ 9,834,496, 16 times the original. The explosively increasing parameters can make the entire network more difficult to train, and meanwhile, it would also greatly increase the time and space (storage) consumption. Therefore, the conclusion is that if the resolution is increased, we would better adapt the network structure to meet the requirement.

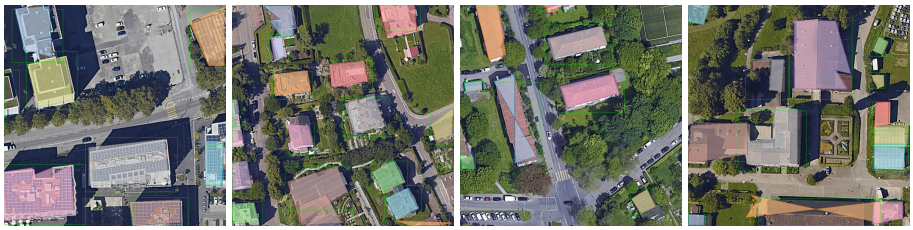
5.1.2 False Vertex

Actually, after the introduction of beam search algorithm, the false vertex problem has been solved a lot and the prediction is greatly improved (see table 4.8), but there are still some in the results, which are shown in figure 5.2.

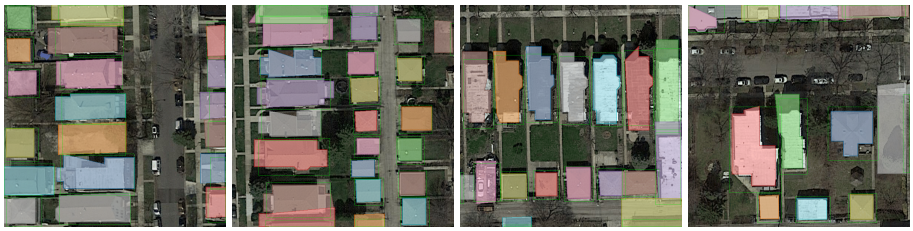
In the dataset of Zurich, the false vertex issue is mainly focused on the order of the vertices. For example, the two adjacent vertices that should be sequential are predicted to be in reverse order, thus the original "L" shape building would show a "zigzag" shape (see figure 5.2(a)).

As for the dataset of Chicago, many buildings are predicted to have bulges and produce sharp corners (see figure 5.2(b)). We think this is mainly influenced by the shadows of the buildings.

The possible solution comes from the point of view of geometry, penalizing the angles with strange degrees in the polygon (e.g. acute angles). Figure 5.3 shows the polygon interior angle degree distribution in the dataset of Zurich and Chicago.

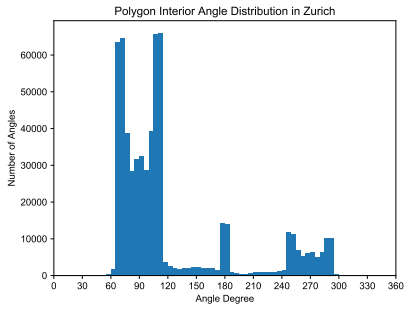


(a) false vertex problem in the dataset of Zurich

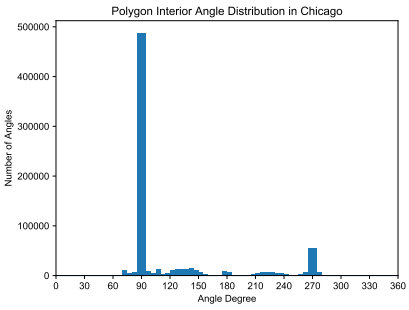


(b) false vertex problem in the dataset of Chicago

Figure 5.2: Examples of false vertex problem in Zurich and Chicago.



(a) angle distribution in Zurich



(b) angle distribution in Chicago

Figure 5.3: Polygon interior angle distribution.

From figure 5.3(b) we can see that in Chicago, most angles are at 90° or 270°, indicating that most of them are right angles, which is easier for a machine to learn. This is also the reason why the dataset of Chicago gets a higher performance. In Zurich's figure 5.3(a), although most angles are around 90° or 270°, they have certain deviations about ±20°. Also, there is a peak near 180° in figure 5.3(a), which means that there are many three-point collinearities in the dataset of Zurich.

In practice of penalizing the angle degree of the polygon, when doing the probability computation of the next vertex, we can use the distribution of the angle θ as a priori to guide the prediction or make a manual intervention. For example, we can simply replace $P(v_t \mid v_{t-1}, v_{t-2})$ in equation 3.1 with

$P(v_t \mid v_{t-1}, v_{t-2}, \theta)$. This requires the function F to take θ as input, which also requires a change of the network structure in order to feed this prior in.

5.2 Future Work

In this section, several future directions of our work are presented, such as the possible direction of roads segmentation, the improvement of the dataset, the methods that are helpful for training and the possible extensions of the model.

Segmentation of Roads At present, our thesis project does not include road segmentation. This is because roads usually cannot be represented as polygons. However, we still hope to add this kind of segmentation in the future, so that the geometrical instance segmentation in aerial images can become complete, and we can also further learn the geometrical layout of the city from aerial images. The basic idea for roads segmentation is to convert roads into centerlines and intersections. Figure 5.4 shows an example.

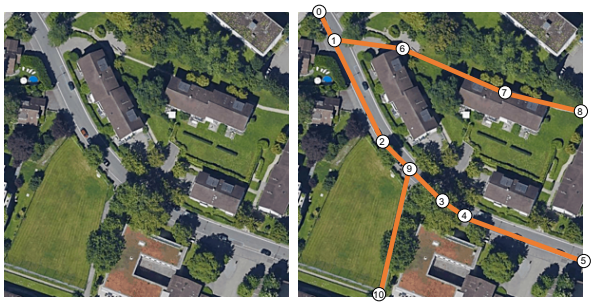


Figure 5.4: A possible approach of road parameterization.

Actually, in Mask R-CNN [11], the part of human posture detection is exactly doing this, where a person is structured as several keypoints with edges connecting them. Figure 5.5 shows some human posture detection results of Mask R-CNN.

Figure 5.5: Human posture detection in Mask R-CNN. Image copyright owned by [11].

However, it is difficult to apply this method to our project, because the road is different from the human body, the number of keypoints of roads in each image is not fixed. That is the biggest obstacle to structurize roads.

Ground Truth Correction Subsection 4.1.5 has already introduced a method to correct ground truth. However, there are still some image-polygon pairs inaccurate, especially for those adjacent buildings (see figure 4.4(d)). We hope that the dataset can be collected from a single source, so that the polygons and the buildings are more likely to match each other. As for the adjacent buildings, it is difficult for us to merge their polygons into a single one. Thus, we hope polygons of buildings instead of houses can be provided.

RoIAlign Implementation Subsection 3.3.3 has already introduced the hybrid version with RoIAlign of R-PolygonRNN. We hope that in the future, this kind of version can be implemented and tested. In theory, it can speed up both the training and prediction phases.

Training Method Our model does not use pre-trained VGG-16 during training, neither as an initializer nor as fixed weights. In the future we may apply the pre-trained VGG-16 to our model's training. Besides, we hope that the alternating training method can be introduced, of which the main idea is to train the subnetworks separately, then train them together and fine-tune the whole model. For more details about this kind of training, please refer to the Fast R-CNN paper [16].

Model Generalization In fact, our model can be applied not only to the geometrical instance segmentation in the aerial images, but also to other kinds of images which contain multiple objects. We hope that our model R-PolygonRNN can be generalized to other fields and we can compare the performance under different dataset.

Appendix A

APIs

A.1 Google Static Maps API

The URL usage of Google Static Maps API is as follows.

```
https://maps.googleapis.com/maps/api/staticmap?
        maptype=<MAP TYPE>&
        center=<LATITUDE>,<LONGITUDE>&
        zoom=<ZOOM LEVEL>&
        size=<IMAGE WIDTH>x<IMAGE HEIGHT>&
        scale=<SCALE>&
        key=<GOOGLE MAPS APIS KEY>
```

The corresponding possible values of parameters are illustrated in table A.1. Note that the parameters should be valid.

Table A.1: Parameters of Google Static Maps API.

Item	**Value**
Map Type	satellite / roadmap / hybrid
Latitude	$(-90, 90]$
Longitude	$(-180, 180]$
Zoom Level	1–21
Image Width	1–640
Image Height	1–640
Scale	1 or 2
Google Maps APIs Key	Provided by Google.

A.2 OpenStreetMap

The URL usage of OpenStreetMap is as follows.

```
https://www.openstreetmap.org/api/0.6/map?bbox=
        <LOWER BOUND>%2C
        <LEFT BOUND>%2C
        <UPPER BOUND>%2C
        <RIGHT BOUND>%2C
```

The corresponding possible values of parameters are illustrated in table A.2. Note that the parameters should be valid, and the bounding box should not be too large.

Table A.2: Parameters of OpenStreetMap.

Item	Value
Lower & Upper Bound	$(-90, 90]$
Left & Right Bound	$(-180, 180]$

For the detailed XML structure of the obtained `.osm` file, please refer to its official website[1].

[1] `https://wiki.openstreetmap.org/wiki/OSM_XML`

Appendix B

Algorithms

B.1 Projection

According to the Google Maps JavaScript API [1], the process of projecting longitude and latitude coordinates $(p_{\text{lon}}, p_{\text{lat}})$ in **radian** to its world pixel coordinates (p_x, p_y) is described as

$$p_x = c \cdot 2^z(\frac{1}{2} + \frac{1}{2\pi}p_{\text{lon}}), \tag{B.1}$$

$$p_y = c \cdot 2^z(\frac{1}{2} - \frac{1}{4\pi}\ln\frac{1+\sin p_{\text{lat}}}{1-\sin p_{\text{lat}}}), \tag{B.2}$$

where c is a constant, referring to the so-called tile size, which is 256 in Google Maps APIs.

B.2 Order of Polygon Vertices

The area S of a polygon $P = \{(x_t, y_t)\}_{t\in\{1,2,\ldots,T\}}$ in Cartesian coordinate system can be computed as

$$S = \frac{1}{2}\sum_{t=1}^{T}\begin{vmatrix} x_t & y_t \\ x_{t+1} & y_{t+1} \end{vmatrix}, \tag{B.3}$$

where (x_{T+1}, y_{T+1}) refers to (x_1, y_1). $S > 0$ if and only if the order of the polygon vertices is anticlockwise, and vice versa. Note that the Y-axis in the pixel coordinates is contrary to the Y-axis in the Cartesian coordinate system, thus $S > 0$ refers to a clockwise polygon.

[1] https://developers.google.com/maps/documentation/javascript/coordinates

B.3 Shift and Resizing Adjustment

The shift adjustment is exactly an optimization problem defined as follows. We use brute force to find out the best shift and resizing rate.

$$\min_{i,j,\alpha} \beta_1 \text{var}(I, f_{ij\alpha}) - \beta_2 \text{match}(I_e, e_{ij\alpha}) - \beta_3 \text{match}(I_c, c_{ij\alpha}), \tag{B.4a}$$

$$\text{s. t. } t_1 \leqslant \alpha \leqslant t_2, t_3 \leqslant i \leqslant t_4, t_5 \leqslant j \leqslant t_6, \tag{B.4b}$$

$$\text{match}(I_c, c_{ij\alpha}) \geqslant t_7, \tag{B.4c}$$

$$\text{diff}(I, f_{ij\alpha}, f_{0,0,1}) \leqslant t_8, \tag{B.4d}$$

$$\text{ground}(I, f_{ij\alpha}) \leqslant t_9, \tag{B.4e}$$

where i and j denote the shift of polygon, α denotes the resizing rate of polygon, I, I_e, I_c denote the original image, mask of edges, mask of corners respectively, $f_{ij\alpha}$, $e_{ij\alpha}$, $c_{ij\alpha}$ denote the set of pixels of polygon's face, edges, and corners after the transformation (i, j, α), respectively, $\text{var}(\cdot, \cdot)$, $\text{match}(\cdot, \cdot)$, $\text{diff}(\cdot, \cdot, \cdot)$, $\text{ground}(\cdot, \cdot)$ denote the color variance of pixels, matching degrees, color difference and similarity with ground, $t_{1:9}$ and $\beta_{1:3}$ denote self-defined parameters. Equations B.4d and B.4e are the regularization terms, meaning that the pixels covered by adjusted polygon should not be too different from the original, and should not too similar to the ground.

B.4 Anchor Assignment

As mentioned in subsection 3.2.3, the matching result has two requirements: (1) Each ground truth bounding box should have at least one anchor (either positive, natural or negative) assigned to it; (2) Each positive anchor should be assigned to its "closest" ground truth bounding box. Here gives the matching process.

Firstly, we compute the IoU score matrix $O = \{O_{ij}\}_{i \in \{1,2,\ldots,n_a\}, j \in \{1,2,\ldots,n_b\}}$,

$$O_{ij} = \text{IoU}(a_i, b_j), \tag{B.5}$$

where n_a and n_b denote the number of anchors and the number of ground truth bounding boxes, a_i and b_j denote the i-th anchor and j-th ground truth bounding box.

Secondly, we use the maximum bipartite matching algorithm (e.g. maximum flow), denoting $\text{MaxBiMatchForCol}(\cdot)$, to select the corresponding anchor for each ground truth bounding box.

$$B_{ij} = \text{MaxBiMatchForCol}(O), \tag{B.6}$$

where B_{ij} denotes the primary assignment relationship between a_i and b_j, and we have

$$B_{ij} = \begin{cases} 1, & \text{if } a_i \text{ is primarily assigned to } b_j, \\ 0, & \text{otherwise.} \end{cases} \tag{B.7}$$

Note that an anchor should not be assigned twice, thus we have

$$\sum_{i=1}^{n_a} B_{ij} = 1, \forall j \in \{1, 2, \ldots, n_b\}, \tag{B.8}$$

$$\sum_{j=1}^{n_b} B_{ij} \leqslant 1, \forall i \in \{1, 2, \ldots, n_a\}. \tag{B.9}$$

Thirdly, we compute the final assignment matrix A as

$$A_{ij} = \begin{cases} 1, & \text{if } B_{ij} = 1, \text{ or} \\ & \sum_k B_{ik} = 0 \text{ and } j = \text{argmax}_k O_{ik} \text{ and } O_{ij} \geqslant t_h, \\ 0, & \text{otherwise,} \end{cases} \tag{B.10}$$

and decide the polarity of these anchors as

$$p_i = \begin{cases} 1, & \text{if } \sum_j A_{ij} = 1, \\ -1, & \text{else if } \max_j O_{ij} \leqslant t_l, \\ 0, & \text{otherwise,} \end{cases} \tag{B.11}$$

where the the notations of p_i, t_h and t_l have the same meanings as what we mentioned in subsection 3.2.2.

Appendix C

Prediction Results

C.1 Zurich

Figure C.1 gives more examples of prediction results in the test set of Zurich. Please zoom in for more details.

C.2 Chicago

Figure C.2 gives more examples of prediction results in the test set of Chicago. Please zoom in for more details.

Figure C.1: More prediction results on the test set of Zurich. Note that multiple colors are used here to differentiate building instances.

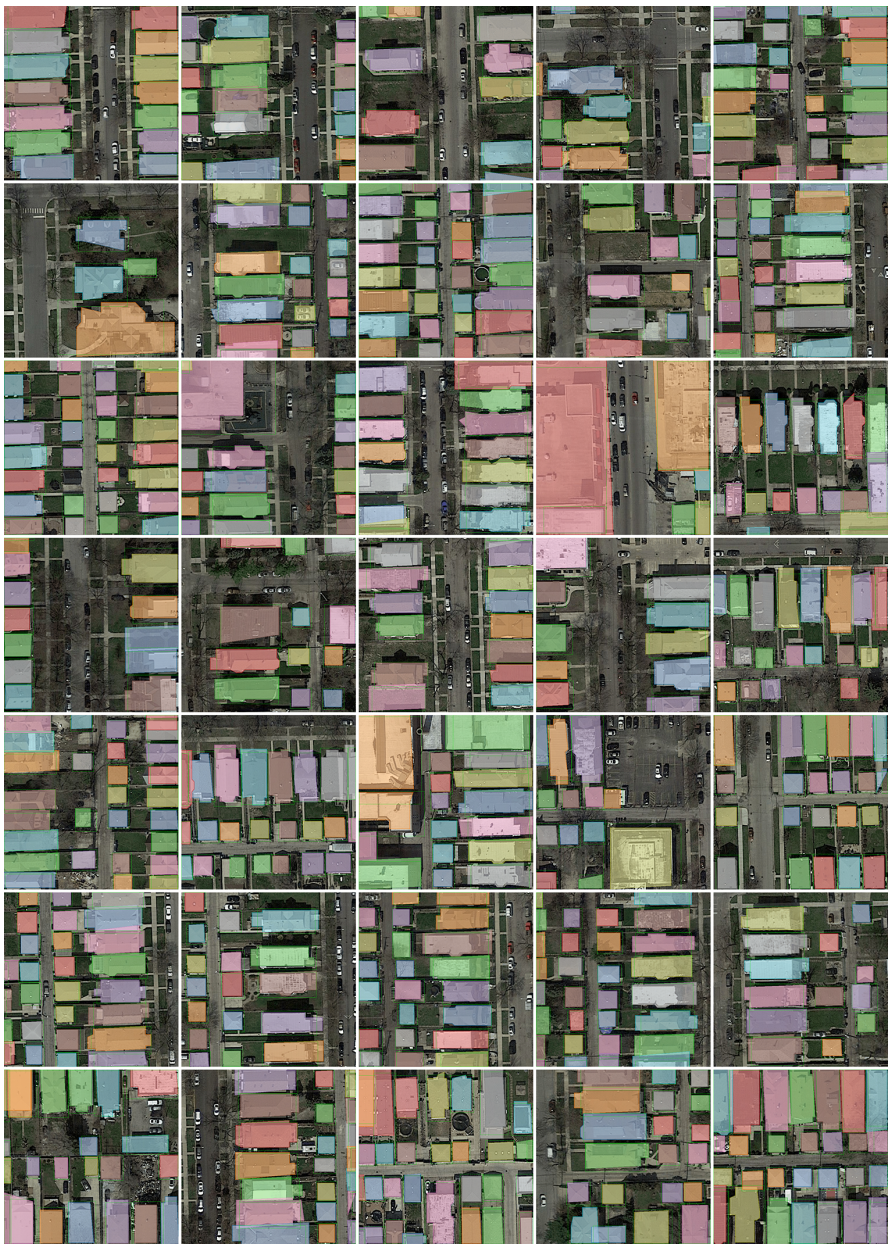


Figure C.2: More prediction results on the test set of Chicago. Note that multiple colors are used here to differentiate building instances.

List of Figures

List of Tables

Bibliography

[1] P. Kaiser, J. D. Wegner, A. Lucchi, M. Jaggi, T. Hofmann, and K. Schindler, "Learning Aerial Image Segmentation from Online Maps," in *IEEE Transactions on Geoscience and Remote Sensing (TGRS)*, 2017.

[2] G. Stockman and L. G. Shapiro, *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1st ed., 2001.

[3] N. Silberman, D. Sontag, and R. Fergus, "Instance Segmentation of Indoor Scenes using a Coverage Loss," in *European Conference on Computer Vision (ECCV)*, 2014.

[4] L. arghout and J. Sheynin, "Real-world Scene Perception and Perceptual Organization: Lessons from Computer Vision," *Journal of Vision*, vol. 13, no. 9, pp. 709–709, 2013.

[5] H. Mobahi, S. Rao, A. Yang, S. Sastry, and Y. Ma, "Segmentation of Natural Images by Texture and Boundary Compression," *International Journal of Computer Vision*, no. 95, pp. 86–98, 2011.

[6] R. Nock and F. Nielsen, "Statistical Region Merging," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 26, no. 11, pp. 1452–1458, 2004.

[7] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature Pyramid Networks for Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] L. Castrejón, K. Kundu, R. Urtasun, and S. Fidler, "Annotating Object Instances with a Polygon-RNN," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2014.

[11] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *International Conference on Computer Vision (ICCV)*, 2017.

[12] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[13] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, p. 66, 2016.

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[15] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2015.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *European Conference on Computer Vision (ECCV)*, 2016.

[19] P. O. Pinheiro, R. Collobert, and P. Dollar, "Learning to Segment Object Candidates," in *International Conference on Neural Information Processing Systems (NIPS)*, 2015.

[20] P. O. Pinheiro, T. Lin, R. Collobert, and P. Dollar, "Learning to Refine Object Segments," in *European Conference on Computer Vision (ECCV)*, 2016.

[21] J. Dai, K. He, Y. Li, S. Ren, and J. Sun, "Instance-sensitive Fully Convolutional Networks," in *European Conference on Computer Vision (ECCV)*, 2016.

[22] J. Dai, K. He, and J. Sun, "Instance-aware Semantic Segmentation via Multi-task Network Cascades," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully Convolutional Instance-aware Semantic Segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[25] R. Girshick, "Fast R-CNN," in *International Conference on Computer Vision (ICCV)*, 2015.

[26] N. Rüegg, "Segmenting Aerial Images into Polygonal Shapes with Deep Learning ," Master's thesis, Swiss Federal Institute of Technology (ETH Zürich), 2016.

[27] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *4th Alvey Vision Conference*, 1988.

[28] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *International Conference on Computer Vision (ICCV)*, 1999.

[29] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded Up Robust Features," in *European Conference on Computer Vision (ECCV)*, 2006.

[30] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Prentice Hall PTR, 3rd ed., 2008.

[31] F.-F. Li and P. Perona, "A Bayesian Hierarchical Model for Learning Natural Scene Categories," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[32] L. Chen, F. Rottensteiner, and C. Heipke, "Invariant Descriptor Learning Using a Siamese Convolutional Neural Network," in *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2016.

[33] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2008.

[34] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Matrix Backpropagation for Deep Networks with Structured Layers," in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[35] J.-P. Bauchet and F. Lafarge, "KIPPI: KInetic Polygonal Partitioning of Images," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[36] J. Forsythe, V. Kurlin, and A. Fitzgibbon, "Resolution-Independent Superpixels Based on Convex Constrained Meshes Without Small Angles," in *International Symposium on Visual Computing (ISVC)*, 2016.

[37] L. Duan and F. Lafarge, "Image Partitioning into Convex Polygons," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2012.

[39] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *International Conference on Machine Learning (ICML)*, 2010.

[40] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[43] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-NMS: Improving Object Detection With One Line of Code," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Instance Segmentation of Geometrical Shapes in Aerial Images

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Zuoyue	Li

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 15 Apr 2018	Li Zuoyue

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

ETH Zurich
ETH-Bibliothek
ETH E-Collection
Rämistrasse 101
8092 Zurich
Tel. +41 44 632 78 64
Fax +41 44 632 13 57
www.library.ethz.ch

Declaration of consent

For the publication of a diploma or master thesis on the ETH E-Collection institutional repository of ETH Zurich

ETH Zurich's institutional repository allows users to access diploma or master theses written at ETH Zurich. This offers academics the opportunity to present their work worldwide. The published theses fulfil the specified formal quality criteria.

Declaration of the author

I hereby declare that I consent to the ETH-Bibliothek making my diploma or master thesis, which I shall submit to the ETH-Bibliothek as a pdf file, available to the public on the institutional repository. The rights of third parties are not infringed by this publication. I consent to any subsequent necessary conversions into other data formats being made.

Author:	Zuoyue Li
Title of the publication:	Instance Segmentation of Geometrical Shapes in Aerial Images
Department:	Department of Computer Science
Publication year:	2018
E-mail/Telephone number:	zuli@student.ethz.ch, +41 (0)78 669 25 86
Private Address:	Röslistrasse 51, 8006 Zürich

Zurich, on 15 Apr 2018 Signature Li Zuoyue

Recommendation of the responsible ETH professor or research supervisor

I have supervised this diploma or master thesis at ETH Zurich and recommend its publication. In particular, I hereby declare that the publication of this thesis does not infringe the rights of third parties and that any rights to confidentiality are protected.

Zurich, on Signature _____

Declaration of the ETH-Bibliothek

The ETH-Bibliothek guarantees the long-term availability of the thesis as well as the integrity of its content and its authenticity and will make the document accessible via the Internet. The intellectual property rights remain with the author.