End-to-End Wireframe Parsing

Yichao Zhou
UC Berkeley
zyc@berkeley.edu

Haozhi Qi
UC Berkeley
hqi@berkeley.edu

Yi Ma
UC Berkeley
yima@berkeley.edu

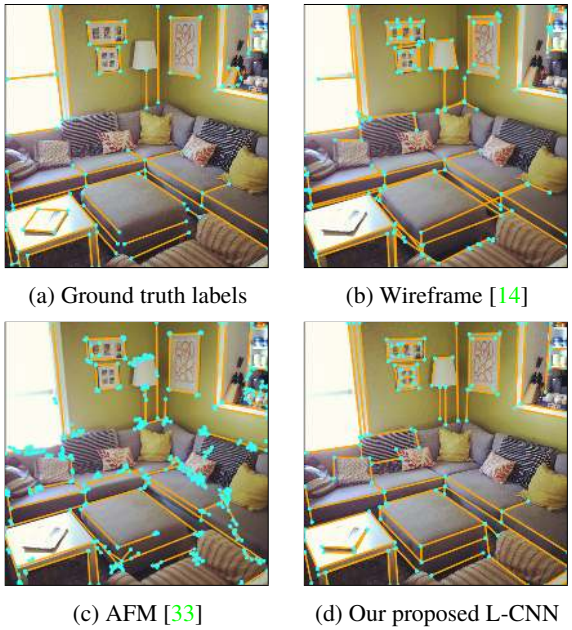
Abstract

We present a conceptually simple yet effective algorithm to detect wireframes [14] in a given image. Compared to the previous methods [14, 33] which first predict an intermediate heat map and then extract straight lines with heuristic algorithms, our method is end-to-end trainable and can directly output a vectorized wireframe that contains semantically meaningful and geometrically salient junctions and lines. To better understand the quality of the outputs, we propose a new metric for wireframe evaluation that penalizes overlapped line segments and incorrect line connectivities. We conduct extensive experiments and show that our method significantly outperforms the previous state-of-the-art wireframe and line extraction algorithms [14, 33, 32]. We hope our simple approach can be served as a baseline for future wireframe parsing studies. Code has been made publicly available at https://github.com/zhou13/lcnn.

1. Introduction

Recent progress in object recognition [17, 31, 29, 13] and large-scale datasets [28, 3, 2, 1] has made it possible to recognize, extract, and utilize high-level geometric features or global structures of a scene for image-based 3D reconstruction. Unlike local features (SIFT [21], ORB [27], etc.) used in conventional 3D reconstruction systems such as structure from motion (SfM) and visual SLAM, high-level geometric features provide more salient and robust information about the global geometry of the scene. This line of research has drawn interests on the exploration of extracting structures such as lines and junctions (wireframes) [14], planes [34, 20], surfaces [11], and room layouts [37].

Among all the high-level geometric features, straight lines and their junctions (together called a wireframe [14]) are probably the most fundamental elements that can be used to assemble the 3D structures of a scene. Recently, works such as [14] encourages the research of wireframe parsing by providing a well-annotated dataset, a learning-based framework, as well as a set of evaluation metrics. Nevertheless, existing wireframe parsing systems are intricate and still inadequate for complex scenes with complicated line connectivity. The goal of this paper is to explore a clean and effective solution to this challenging problem.



(a) Ground truth labels (b) Wireframe [14] (c) AFM [33] (d) Our proposed L-CNN

Figure 1: Demonstration of the wireframe representation of a scene and the results produced by Wireframe [14], AFM [33], and our proposed L-CNN.

Existing researches [14, 33] address the wireframe parsing problem with two stages. First, an input image is passed through a deep convolutional neural network to generate pixel-wise junction and line heat maps (or their variants [33]). After that, a heuristic algorithm is used to search through the generated heat map to find junction positions, vectorized line segments, and their connectivity. While these methods are intuitive and widely used in the current literature, their vectorization algorithms are often complex and rely on a set of heuristics, and thus sometimes lead to inferior solutions. Inspired by [4, 12, 9] in which the end-to-end pipelines outperform their stage-wise counterparts, we hypothesis that making wireframe parsing systems end-to-end trainable could also push the state-of-the-arts.

Therefore, in this paper we address the following problem:

> *How to learn a vectorized representation of wireframes in an end-to-end trainable fashion?*

To this end, we propose a new network called *L-CNN*, an algorithm that performs end-to-end wireframe parsing using a single and unified neural network. Our network can be split into four parts: a feature extraction backbone, a junction proposal module, and a line verification module bridged by a line sampling module. Taken an RGB image as the input, the neural network directly generates a vectorized representation without using heuristics. Our system is fully differentiable and can be trained end-to-end through back-propagation, enabling us to fully exploit the power of the state-of-the-art neural network architectures to parse the scenes.

Besides, current wireframe evaluation metrics treat a line as a collection of independent pixels, so it cannot take the correctness of line connectivity into consideration, as discussed in Section 4.3. To evaluate such structural correctness of a wireframe, we introduce a new evaluation metric. Our new proposed metric uses line matching to calculate the precision and recall curves on vectorized wireframes. We perform extensive experiments on wireframe datasets [14] and carefully do the ablation study on the effects of different system design choices.

2. Related work

Line Detection: Line detection is a widely studied problem in computer vision. It aims to produce vectorized line representation from images. Traditional methods such as [30, 32] detect lines based on local edge features. Recently, [33] combines the deep learning-based features with the line vectorization algorithm from [32]. Unlike the wireframe representation, traditional line detection algorithms do not provide the information about junctions and how lines and junctions are connected to each other, which limits its application in scene parsing and understanding.

Wireframe Parsing: [14] proposes the wireframe parsing task. The authors train two separate neural networks to predict junction and line heat maps from an input image. After that, the two predictions are combined using a heuristic wireframe fusion algorithm to produce the final vectorized output. Although it is intuitive and can produce reasonable results, such two-stage process prevents the benefits of end-to-end training. In contrast, our framework is based on a single end-to-end trainable neural network, which directly delivers a vectorized wireframe representation as the output.

Instance-level Recognition: At the technical level, our method is inspired by instance-level recognition frameworks such as Fast R-CNN [9], Faster R-CNN [25], CornerNet [18], Extremenet [35]. Our pipeline and LoI pooling (Section 3.6) are conceptually similar to the RoI pooling in Faster R-CNN and Fast R-CNN. Both methods first generate a set of proposals and extract features to classify these proposes. The difference is that in [25, 9], the candidate proposals are generated by a sliding window fashion while our proposals are generated by connecting salient junctions (line sampler module Section 3.5). In this sense, the proposal generation procedure is also similar to what is used in point-based object detection [18, 35]. The difference lies in how to discriminate between true lines and false positives. They use either similarity between points feature embedding [18] or the classification score in the geometric center of several salient points [35] while ours extracts features to feed into a small neural network (line verification network Section 3.6).

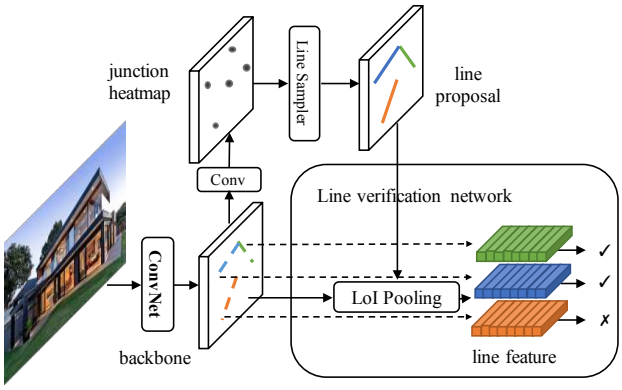


Figure 2: An overview of our network architecture.

3. Methods

3.1. Data Representation

Our representation of wireframes is based on the notation from graph theory. It can also be seen as a simplified version of the wireframe definition in [14]. Let $\mathsf{W} = (\mathsf{V}, \mathsf{E})$ be the wireframe of an image, in which the V is the set of junction indices and $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ is the set of lines represented by the pair of junction endpoints in V. For each $i \in \mathsf{V}$, we use $\mathbf{p}_i \in \mathbb{R}^2$ to represent the (ground truth) coordinate of the junction i in the image space.

3.2. Overall Network Architecture

Figure 2 illustrates the L-CNN architecture. It contains four modules: 1) a feature extraction backbone (Section 3.3) that takes a single image as the input and provides shared intermediate feature maps for the successive modules; 2) a junction proposal module (Section 3.4) which outputs the candidate junctions; 3) a line sampling module (Section 3.5) that outputs line proposals based on the output junctions from the junction proposal module; 4) a line verification module (in Section 3.6) which classifies the proposed lines. The output of L-CNN are the positions of junctions and the connectivity matrix among those junctions. Our system is fully end-to-end trainable with stochastic gradient descent.

3.3. Backbone Network

The function of the backbone network is to extract semantically meaningful features for the successive modules of L-CNN. We choose stacked hourglass network [23] as our backbone for its efficiency and effectiveness. Input images are resized into squares. The stacked hourglass network first downsamples the input images twice in the spatial resolution via two 2-strided convolution layers. After that, learned feature maps are gradually refined by multiple U-Net-like modules [26] (the hourglass modules) with intermediate supervision imposed on the output of each module. The total loss of the network is the sum of the loss on those modules.

3.4. Junction Proposal Module

Junction Prediction: We use a simplified version of [14] to estimate the candidate junction locations in the wireframe. An input image with resolution $W \times H$ is first divided into $W_b \times H_b$ bins. For each bin, the neural network predicts whether there exists a junction inside it, and if yes, it also predicts the its relative location inside this bin. Mathematically, the neural network outputs a junction likelihood map J and an offset map $\mathbf{O}$. For each bin b, we have

$$J(b) = \begin{cases} 1 & \exists i \in \mathsf{V} : \mathbf{p}_i \in b, \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathbf{O}(b) = \begin{cases} (\mathbf{b} - \mathbf{p}_i)/W_b & \exists i \in \mathsf{V} : \mathbf{p}_i \in b, \\ 0 & \text{otherwise.} \end{cases}$$

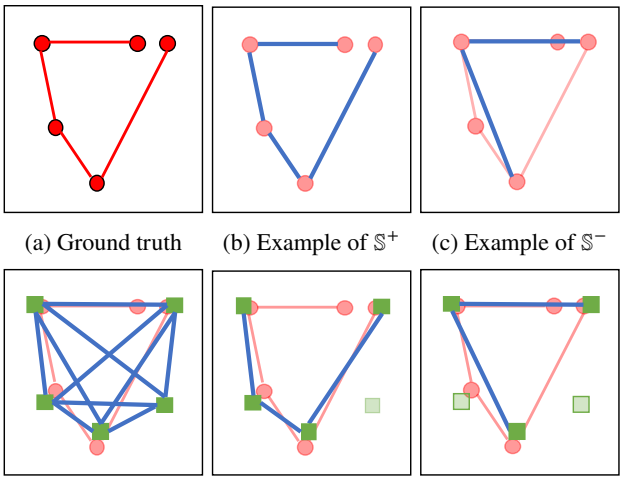
where $\mathbf{b}$ represents the location of bin b's center and $\mathbf{p}$ represents the location of a vertex in V.

To predict J and $\mathbf{O}$, we design a network head that consists of two 1×1 convolution layers to transform the feature maps into J and $\mathbf{O}$. We treat the problem of prediction J as a classification problem and use the average binary cross entropy loss. We use ℓ_2 regression to predict the offset map $\mathbf{O}$. As the range of offset $\mathbf{O}(b)$ is bounded by $[-1/2, 1/2) \times [-1/2, 1/2)$, we append a sigmoid activation with offset -0.5 after the head to normalize the output. The loss on $\mathbf{O}$ is averaged over the bins that contain ground-truth junctions for each input image.

Non-Maximum Suppression: In instance-level recognition, non-maximum suppression (NMS) is applied to remove duplicate around correct predictions. We use the same mechanism for remove blurred score map around correct predictions and get $J'(b)$ as:

$$J'(b) = \begin{cases} J(b) & J(b) = \max_{b' \in \mathcal{N}(b)} J(b') \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{N}(b)$ represents the 8 nearby bins around b. Here, we suppress the pixel values that are not the local maxima on the junction map. Such non-maximum suppression can be implemented with a max-pooling operator. The final output of the junction proposal network is the top K junction positions $\{\hat{\mathbf{p}}_i\}_{i=1}^K$ with the highest probabilities in J'.



(a) Ground truth (b) Example of $\mathbb{S}^+$ (c) Example of $\mathbb{S}^-$

(d) Example of $\mathbb{D}^*$ (e) Example of $\mathbb{D}^+$ (f) Example of $\mathbb{D}^-$

Figure 3: Illustration of our sampling methods. Red circles represent the ground truth junctions, red lines represent the ground truth lines, green squares represent the predicted junctions, and blue lines represent the candidate lines in the static and dynamic samplers.

3.5. Line Sampling Module

Given a list of K best candidate junctions $\{\hat{\mathbf{p}}_i\}_{i=1}^K$ from the junction proposal module, the purpose of the line sampling module is to generate a list of line candidates $\{L_j\}_{j=1}^M = \{(\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)\}_{j=1}^M$ during the training stage so that the line verification network can learn to predict the existence of a line. Here $\tilde{\mathbf{p}}_j^1$ and $\tilde{\mathbf{p}}_j^2$ represents the coordinates of two endpoints of the jth candidate line segment. In this task, the amount of positive samples and negatives samples are extremely unbalanced, we address this issue by carefully design the sampling mechanism as stated below.

Static Line Sampler: For each image, the static line sampler returns $N_{\mathbb{S}^+}$ positive samples and $N_{\mathbb{S}^-}$ negative samples that are directly derived from the ground truth labels. We call them static samples since they are irrelevant to the predicted candidate junction positions. Positive line samples are uniformly sampled from all the ground truth lines, denoted by $\mathbb{S}^+$, with the ground truth coordinate of the corresponding junctions. The number of total negative line samples is $O(|\mathsf{V}|^2)$, which is huge compared to the number of positive samples $O(|\mathsf{E}|)$. To alleviate the problem, we sample the negative lines from $\mathbb{S}^-$, a set of negative lines that are potentially hard to classify. We use the following heuristic to compute the $\mathbb{S}^-$: we first rasterize all the ground truth lines onto a 64×64 low-resolution bitmap. Then, for each possible connections formed by a pair of ground truth junctions that is not a ground truth line, we define its *hardness score* to be the average pixel density on the bitmap along this line.

For each image, $\mathbb{S}^-$ is set to be the top 2000 lines with the highest hardness scores.

Dynamic Line Sampler: In contrast to the static line sampler, the dynamic line sampler samples the lines using the predicted junctions from the junction proposal module. The sampler first matches all the predicted junctions to the ground truth junction. Let $m_i := \arg\min_j \|\hat{\mathbf{p}}_i - \mathbf{p}_j\|_2$ be the index of the best matching ground truth junction for the ith junction candidates. If the ℓ_2-distance between $\hat{\mathbf{p}}_i$ and $\mathbf{p}_{m_i}$ is less than the threshold η, we say that the junction candidate $\hat{\mathbf{p}}_i$ is *matched*. For each line candidate line $(\hat{\mathbf{p}}_{i_1}, \hat{\mathbf{p}}_{i_2})$ in which $i_1, i_2 \in \{1, 2, \ldots, K\}$ and $i_1 \neq i_2$, we put it into line sets $\mathbb{D}^+$, $\mathbb{D}^-$, and $\mathbb{D}^*$ according to the following criteria:

- if both $\hat{\mathbf{p}}_{i_1}$ and $\hat{\mathbf{p}}_{i_2}$ are matched, and $(m_{i_1}, m_{i_2}) \in \mathsf{E}$, we add this line to the *positive sample set* $\mathbb{D}^+$;
- if both $\hat{\mathbf{p}}_{i_1}$ and $\hat{\mathbf{p}}_{i_2}$ are matched, and $(m_{i_1}, m_{i_2}) \in \mathbb{S}^-$, we add this line to the *hard negative sample set* $\mathbb{D}^-$;
- the *random sample set* $\mathbb{D}^*$ includes all the line candidates from the predicted junctions, regardless of their matching results.

Finally, we randomly choose $N_{\mathbb{D}^+}$ lines from the positive sample set, $N_{\mathbb{D}^-}$ lines from the hard negative sample set, $N_{\mathbb{D}^*}$ lines from the random line sample set, and return their union as the dynamic line samples.

On one hand, the static line sampler helps cold-start the training at the beginning when there are few accurate positive samples from the dynamic sampler. It also complements the dynamic sampler by adding ground truth positive samples and hard negative samples to help the joint training process. On the other hand, the dynamic line sampler improves the performance of line detection by adapting the line endpoints to the predicted junction locations.

3.6. Line Verification Network

The line verification network takes a list of candidate lines $\{L_j\}_{j=1}^M = \{(\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)\}_{j=1}^M$ along with the feature maps of the image from the backbone network as the input and predicts whether or not each line is in the wireframe of the scene. During training, L is computed by the line sampling modules, while during the evaluation, L is set to be every pair of the predicted junctions $\{\hat{\mathbf{p}}_i\}_{i=1}^K$.

For each candidate line segment $L_j = (\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)$, we feed the coordinates of its two endpoints into a *line of interest (LoI) pooling layer* (introduced below), which returns a fixed-length feature vector. Then, we pass the concatenated feature vector into a network head composed of two fully connected layers and get a logit. The loss of the line is the sigmoid binary cross entropy loss between the logit and the label of this line, i.e., a positive sample or a negative sample. To keep the loss balanced between positive and negative samples, the loss on each image for the line verification network is the sum of two separated loss, averaged over the positive lines and the negative lines, respectively.

LoI Pooling: To check whether a line segment exists in an image, we first turn the line into a feature vector. Inspired by the RoIPool and RoIAlign layers from the object detection community [10, 9, 25, 12], we propose the LoI pooling layer to extract line features while it can back-propagate the gradient to the backbone network.

Each LoI is defined by the coordinates of its two endpoints, i.e., $\tilde{\mathbf{p}}_j^1$ and $\tilde{\mathbf{p}}_j^2$. The LoI pooling layer first computes the coordinates of N_p uniform spaced middle points along the line with linear interpretation

$$\mathbf{q}_k = \frac{k}{N_p - 1}\tilde{\mathbf{p}}_j^1 + \frac{N_p - k}{N_p - 1}\tilde{\mathbf{p}}_j^2, \quad \forall k \in \{0, 1, \ldots, N_p - 1\}.$$

Then, it calculates the feature values at those N_p points in the backbone's feature map using bilinear interpretation to avoid quantization artifacts [4, 15, 6, 12]. The resulting feature vector has a spatial extent of $C \times N_p$, in which C is the channel dimension of the feature map from the backbone network. After that, the LoI Pooling layer reduces the size of the feature vector with a 1D max pooling layer. The result feature vector has shape $C \times \lceil \frac{N_p}{s} \rceil$, where s is the size of stride of the max pooling layer. This vector is then flattened and returned as the output of LoI pooling layer.

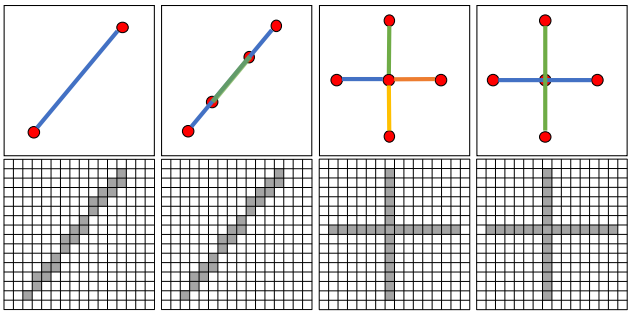
4. Experiments

4.1. Implementation Details

We use a stacked hourglass network [23] as our backbone. Given an input image, we first apply a 7×7 stride-2 convolution, three residual blocks with channel dimension 64, and append a stride-2 max pooling on it. Then this intermediate feature representation is fed into two stacked hourglass modules. In each hourglass, the feature maps are down-sampled with 4 stride-2 residual blocks and then up-sampled with nearest neighbour interpolation. The dimensions of both the input channel and the output channel of each residual block are 256. The network heads for J and $\mathbf{O}$ contain a 3×3 convolutional layer that reduces the number of channels to 128 with the ReLU non-linearity, followed by a 1×1 convolutional layer to match the output dimension.

We reduce the feature dimension from 256 to 128 using a 1×1 convolution kernels before feeding the feature map into the line verification network. For the LoIPool layer, we pick $N_p = 32$ points along each line as the features of the line, resulting a 128×32 feature for each line. After that, we apply a one-dimensional stride-4 max pooling to reduce the spatial dimension of line features from 32 to 8. Our final line feature has dimension 128×8. The head of the line verification network then takes the flattened feature vector and feeds it into two fully connected layers with ReLU non-linearity, in which the middle layer has 1024 neurons.

All the experiments are conducted on a single NVIDIA GTX 1080Ti GPU for neural network training. We use the ADAM optimizer [16]. The learning rate and weight



(a) Overlapped lines (b) Incorrect connectivity

Figure 4: Demonstration of cases that heat map-based metric is not ideal for wireframe quality evaluation. The upper part shows the detected lines and their heat maps, and the lower part shows the ground truth lines and their heat maps.

decay are set to 4×10^{-4} and 1×10^{-4}, respectively. The batch size is set to 6 for maximizing the GPU memory occupancy. We train the network for 10 epochs and then decay the learning rate by 10. We stop the training at 16 epochs as the validation loss no longer decreases. The total training time is about 8 hours. All the input images are resized to $(H, W) = (512, 512)$ and we use $H_b \times W_b = 128 \times 128$ bins for J and $\mathbf{O}$. The junction proposal proposal network outputs the best $K = 300$ junctions. For the line sampling module, we use $N_{\mathbb{S}^+} = 300$, $N_{\mathbb{S}^-} = 40$, $N_{\mathbb{D}^+} = 300$, $N_{\mathbb{D}^-} = 80$, and $N_{\mathbb{D}^*} = 600$. The loss weights of multi-task learning for J, $\mathbf{O}$, and line verification network are set to 8, 0.25, and 1, respectively. Those weights are adjusted so that the magnitudes of the losses are in a similar scale.

4.2. Datasets

We conduct most of our experiments on the ShanghaiTech dataset [14]. It contains 5,462 images of man-made environments, in which 5000 images are used as the training set and 462 images are used as the testing set. The wireframe annotation of this dataset includes the positions of the salient junctions V and lines E. We also test the models trained with the ShanghaiTech dataset on the York Urban dataset [7] to evaluate the generalizability of all the methods.

4.3. Evaluation Metric

Previously, researchers use two metrics to evaluate the quality of the detection wireframes: the heat map-based AP^{H} to evaluate lines and junction AP to evaluate junctions. In this section, we first give a brief introduction to these metrics and discuss the reason why they are not proper for the wireframe parsing tasks. Then we give a new metric, named *structural AP*, a more reasonable way to evaluate the structural quality of wireframes.

Precision and Recall of Line Heat Maps: The precision and recall curve over line heap maps is often used to evaluate the performance of wireframe and line detection [14, 33]. Given a vectorized representation (lines or wireframes), it first generates a confidence heat map by rasterizing the lines. To compare it with the ground truth heat map, a bipartite matching that treats each pixel independently as a graph node is ran to match between two heat maps. Then precision and recall curve is computed according to the matching and confidence of each pixel. In our experiment, we provide analysis of different methods using this metric. We show both the F-score (as in [33]) and the area under the PR curve (similar to [8]) as the quantitative measure, and write the them as F^{H} and AP^{H}, respectively.

These metrics were originally designed for evaluating boundary detection [22] and they work well for that purpose. However they are problematic in wireframe detection since

1. they do not penalize for overlapped lines (Figure 4a);
2. they do not properly evaluate the connectivity of the wireframe (Figure 4b).

For example, if a long line is broken into several short line segments, the resulted heat map is almost the same as the ground truth heat map, as shown in Figure 4. A good performance on the above two properties is vital for downstream tasks that rely on the correctness of line connectivity, such as inferring the 3D geometry through lines [24, 36].

Structural AP: To overcome those drawbacks, we propose a new evaluation metric defined on vectorized wireframes rather than on a heat map. We call our metric *structural average precision (sAP)*. This metric is inspired by the mean average precision commonly used in object detection [8]. Structural AP is defined to be the area under the precision recall curve computed from a scored list of the detected line segments on all test images. Recall is the proportion of the correctly detected line segments (up to a cutoff score) to all the ground truth line segments, while precision is the proportion of the correctly detected line segments above that cutoff to all the detected line segments.

A detected line segment $L_j = (\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)$ is considered to be a true positive (correct) if and only if

$$\min_{(u,v)\in\mathsf{E}} \|\tilde{\mathbf{p}}_j^1 - \mathbf{p}_u\|_2^2 + \|\tilde{\mathbf{p}}_j^2 - \mathbf{p}_v\|_2^2 \le \vartheta,$$

where ϑ is a user-defined number represents the strictness of the metric. In this experiment section, we evaluate the structural AP at $\vartheta = 5$, $\vartheta = 10$, and $\vartheta = 15$ under the resolution of 128×128. We abbreviate them as sAP^5, sAP^{10}, and sAP^{15}, respectively. In addition, each ground truth line segment is not allowed to be matched more than once in order to penalize double-predicted lines. That is to say if there exists a line L_i that is ranked above the line L_j and

$$\begin{aligned} &\arg\min_{(u,v)\in\mathsf{E}} \|\tilde{\mathbf{p}}_i^1 - \mathbf{p}_u\|_2^2 + \|\tilde{\mathbf{p}}_i^2 - \mathbf{p}_v\|_2^2 \\ =\ &\arg\min_{(u,v)\in\mathsf{E}} \|\tilde{\mathbf{p}}_j^1 - \mathbf{p}_u\|_2^2 + \|\tilde{\mathbf{p}}_j^2 - \mathbf{p}_v\|_2^2, \end{aligned}$$

then the line L_j will always be marked as a false positive.

	sampler					head		metric		
	$\mathbb{D}^*$	$\mathbb{S}^+$	$\mathbb{S}^-$	$\mathbb{D}^+$	$\mathbb{D}^-$	fc+fc	conv+fc	sAP^5	sAP^{10}	sAP^{15}
(a)	✓					✓		43.7	48.2	50.2
(b)		✓	✓			✓		38.5	41.9	43.8
(c)	✓	✓	✓			✓		47.8	51.7	53.6
(d)	✓	✓	✓	✓	✓		✓	55.7	59.8	61.7
(e)	✓	✓		✓		✓		57.4	61.4	63.2
(f)	✓	✓	✓	✓	✓	✓		**58.9**	**62.9**	**64.7**

Table 1: Ablation study of L-CNN. The columns labeled with "sampler" represent whether a specific sampler is used during the training stage, as introduced in Section 3.5. The columns labeled with "head" represent the network head structured used in the line verification network. "fc + fc" is the network structure introduced in Section 3.6, while in "conv + fc" we replace the middle fully connected layer with a 1D Bottleneck layer [13].

Junction mAP: The major difference between line detection and wireframe detection is that the wireframe representation encodes junction positions. Junctions have physical meaning in 3D (corners or occlusional points) and encodes the line connectivity information. Our junction mean AP (mAP^J) evaluates the quality of vectorized junctions of a wireframe detection algorithm without relying on heat maps as in [14]. To better understand the advantage of explicitly modeling junctions, we also evaluate our method using the junction mAP as described below: for a given ranked list of predicted junction positions, a junction is considered to be correct if the ℓ^2 distance between this junction and its nearest ground-truth is within a threshold. Each ground truth junction is only allowed to be matched once to penalize double-predicted junctions. Using this criteria, we can draw the precision recall curve by counting the number of true and false positives. The junction AP is defined to be the area under this curve. The mean junction AP is defined to be the average of junction AP over difference distance thresholds. In our implementation, we choose to average over 0.5, 1.0, and 2.0 thresholds under 128×128 resolution.

4.4. Ablation Study

In this section, we run a series of ablation experiments on the ShanghaiTech dataset [14] to study our proposed method. We use our structural average precision (sAP) as the evaluation metrics. The results are shown in Table 1.

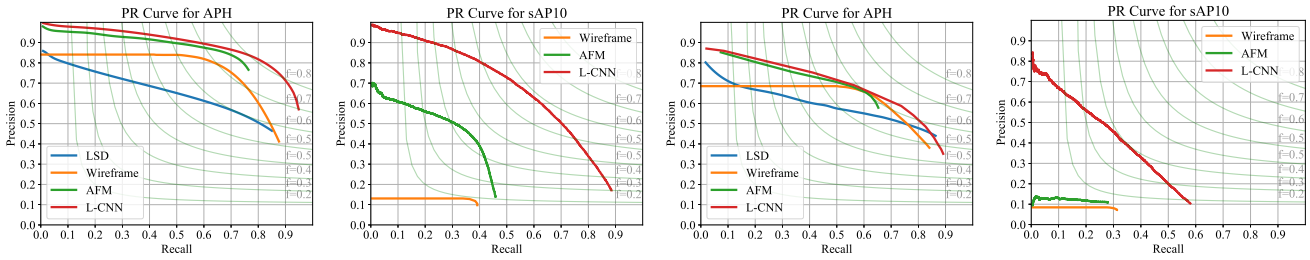
Line Sampling Modules: We compare different design choices for line sampling modules, as shown in Table 1. (a) uses just the random pairs from the dynamic sampler. The sAP^5 is 43.7, which serves as the baseline. (b) only uses the sampled pairs from ground-truth junctions and get much worse performance. The performance gap is even larger when the evaluation criterion is loose. This is because (b) does not consider the imperfect of junction prediction map and cannot tackle when junction is slightly misaligned with the ground truth. After that, we combine the random dynamic sampling and the static sampling, which significantly improves the performance, as shown in Table 1 (c). Then we add dynamic sampler candidate $\mathbb{D}^+$ and $\mathbb{D}^-$, which leads to the best sAP^5 score 58.9 in (f). This experiment indicates that the carefully selected dynamic line candidates are vital to a good performance. Lastly, by comparing (e) and (f), we find that including hard examples $\mathbb{S}^-$ and $\mathbb{D}^-$ is indeed helpful compared to just doing the random sampling in $\mathbb{D}^*$.

Line Verification Networks: Table 1 also shows our ablation on how to design the line verification network. We tried two different designs: In Table 1 (e), we apply two fully-connected layers after the LoI Pooling feature to get the classification results, while in Table 1 (d) we firstly apply a 1D convolution on the features and then use the fully-connected layer on the flattened feature vector to get the final line classification. Experiments show that using convolution largely deteriorates the performance. We hypothesis that this is because line classification requires location sensitivity, which the translation-invariant convolution cannot provide [5, 19].

4.5. Comparison with Other Methods

Following the practice of [14, 33], we compare our method with LSD [32], deep learning-based line detectors [33], as well as wireframe parser from the ShanghaiTech dataset paper [14]. F^H, AP^H, and sAP with different thresholds are used to evaluate those methods quantitatively. All the models are trained on the ShanghaiTech dataset and evaluate on both of the ShanghaiTech [14] and York Urban datasets [7]. The results are shown in Table 2 and Figure 5. We note that the difference of the numbers and curves for AP^H from [14, 33] is due to our more proper implementation of AP^H: 1) In the code provided by [14], they evaluate the precision and recall per image and average them together, while we first sum the number of true positives and false positives over the dataset and then compute the precision and recall. 2) Due to the insufficient number of thresholds, the PR curves in [14, 33] do not cover all the recall that an algorithm can achieve. We evaluate all the methods on more thresholds to extend the curves as long as possible.

Figure 5a shows that our algorithm is better than the state-of-the-art line detector methods under the PR curve of heat map-based line metrics, especially in the high-recall region. This indicates that our method can find more correct lines compared to other methods. We also quantitatively calculate the F-score and the average AP. Table 2 shows that our algorithm performs significantly better than previous state-of-the-art line detectors by 13.3 points in AP^H and 4.0 points in F^H [33]. We also want to emphasize that compared to line detection, it is conceptually harder for the wireframe detection methods to reach the same performance as the line detection methods in term of the heat map-based metrics.



(a) Heat map-based PR curves on the ShanghaiTech dataset [14].
(b) Structural PR curves on the ShanghaiTech dataset [14].
(c) Heat map-based PR curves on the York Urban dataset [7].
(d) Structural PR curves on the York Urban dataset [7].

Figure 5: Precision recall curves of multiple algorithms. All the models are training on the ShanghaiTech dataset.

This is because a wireframe detection algorithm requires the positions of junctions, the endpoints of lines, to be correct, while a line detector can start and end a line arbitrarily to "fill" the line heat map. Before evaluating the heat map-based metrics, we post process the lines from L-CNN to remove the overlap, as described in Appendix A.1.

We then evaluate all the methods with our proposed structural AP. The precision recall curve is shown in Figure 5b (LSD is missing here as its scores are too low to be drawn). The gap between our method and previous methods is even larger. Our method achieves 40-point sAP improvement over the previous state-of-the-art method. This is because our line verification network penalizes incorrect structures, while methods such as AFM and wireframe parser use a hand-craft algorithm to extract lines from heat maps, in which the information of junction connectivity gets lost. Furthermore, the authors of [14] mention that their vectorization algorithm will break lines and add junctions to better fit the predicted heat map. Such behaviors can worsen the structure correctness, which might explain its low sAP score.

The mAP^J results are shown in Table 2. For AFM, we treat the endpoints of each line as junctions and use the line NFA score as the score of its endpoints. We note that the inferior junction quality of AFM is not because their method is not well-designed but the end task is different. This shows that one cannot directly apply a line detection algorithm on the wireframe parsing task. In addition, our L-CNN outperforms the previous wireframe parser [14] by a large margin due to the joint training process of the pipeline.

Table 2 and Figures 5c and 5d show that L-CNN also performs the best among all the wireframe and line detection methods when testing on a different dataset [7] without finetune. This indicates that our method is able to generalize to novel scenes and data. We note that the relatively low sAP scores are due to the duplicated lines, texture lines, while missing many long lines in the annotation of the dataset.

4.6. Visualization

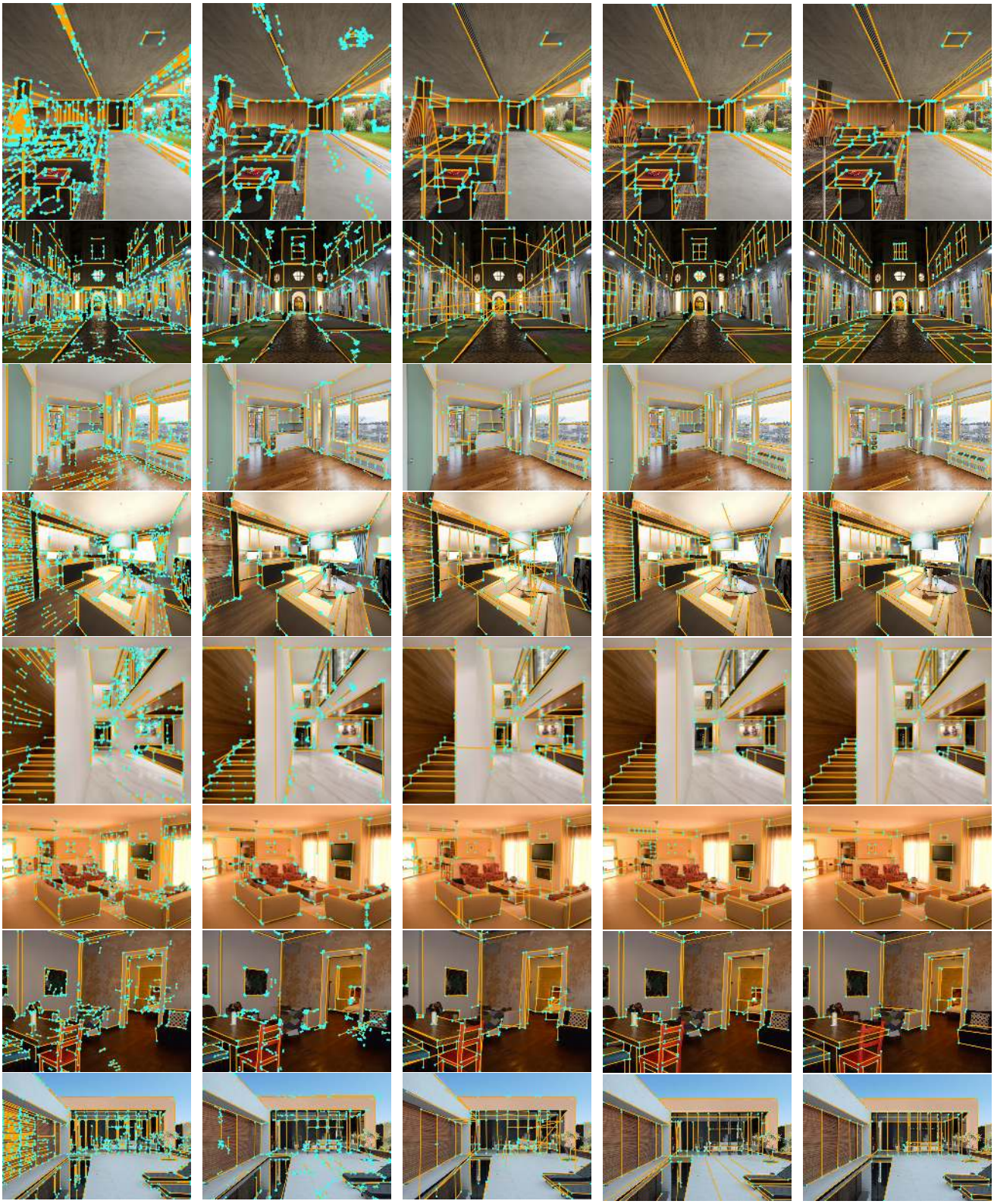
We visualize our algorithm's output in Figure 6. The junctions are marked cyan blue and lines are marked orange. Wireframes from L-CNN are post processed using the method from Appendix A.1. Since LSD and AFM do not explicitly output junctions, we treat the endpoints of lines as junctions. As shown in Figure 6, LSD detects some high-frequency textures without semantic meaning. This is expected as LSD is not a data-driven method. By training a CNN to predict line heat maps, AFM [33] is able to suppress some noise. However, both LSD and AFM still produce a lot of short lines because they do not have an explicit notion of junctions. The wireframe parser [14] utilizes junctions to provide a relatively cleaner result, but their heuristic vectorization algorithm is sub-optimal and leads to crossing lines and incorrectly connected junctions. In contrast, our L-CNN uses powerful neural networks to classify whether a line indeed exists and thus provides the best performance.

	ShanghaiTech [14]				York Urban [7]			
	sAP^{10}	mAP^J	AP^H	F^H	sAP^{10}	mAP^J	AP^H	F^H
LSD	/	/	52.0	61.0	/	/	51.0	60.0
Wireframe	5.1	40.9	67.8	72.6	2.6	13.4	53.4	63.7
AFM	24.4	23.3	69.5	77.2	3.5	6.9	48.4	63.1
L-CNN	**62.9**	**59.3**	**83.0**	**81.2**	**26.4**	**30.4**	**59.8**	**65.4**

Table 2: Performance comparison of wireframe detection. All the models are trained on the ShanghaiTech dataset and evaluate on both datasets. The columns labelled with "sAP" show the line accuracy with respect to our structural metrics; the columns labelled with "mAP^J" shows the mean average precision of the predicted junctions; the columns labelled with "F^H" and "AP^H" shows the performance metrics related to heat map-based PR curves. Our method L-CNN has the state-of-the-art performance on all of the evaluation metrics.

Acknowledgement

This work is partially supported by Sony US Research Center, Berkeley BAIR, and Bytedance Research Lab. We thank Kenji Tashiro of Sony for helpful discussions. We thank Cecilia Zhang of Berkeley for her helpful comments on the draft of this paper.



(a) LSD (b) AFM (c) Wireframe (d) L-CNN (ours) (e) Ground truth

Figure 6: Qualitative evaluation of wireframe and line detection methods. From left to right, the columns shows the results from LSD [32], AFM [33], Wireframe [14], L-CNN (ours), and the ground truth. We also draw the detected junctions from Wireframe and L-CNN and the line endpoints from LSD and AFM.

References

[1] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2D-3D-semantic data for indoor scene understanding. *arXiv*, 2017. 1

[2] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. ShapeNet: An information-rich 3D model repository. *arXiv*, 2015. 1

[3] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*, 2017. 1

[4] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016. 1, 4

[5] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *NIPS*, 2016. 6

[6] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017. 4

[7] Patrick Denis, James H Elder, and Francisco J Estrada. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *European conference on computer vision*, pages 197–210. Springer, 2008. 5, 6, 7

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL visual object classes (VOC) challenge. *IJCV*, 2010. 5

[9] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, 2015. 1, 2, 4

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 4

[11] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3D surface generation. In *CVPR*, 2018. 1

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 1, 4

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 6

[14] Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. Learning to parse wireframes in images of man-made environments. In *CVPR*, 2018. 1, 2, 3, 5, 6, 7, 8

[15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015. 4

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. 4

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[18] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *ECCV*, 2018. 2

[19] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, 2017. 6

[20] Chen Liu, Jimei Yang, Duygu Ceylan, Ersin Yumer, and Yasutaka Furukawa. PlaneNet: Piece-wise planar reconstruction from a single RGB image. In *CVPR*, 2018. 1

[21] David G Lowe et al. Object recognition from local scale-invariant features. In *ICCV*, 1999. 1

[22] David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *PAMI*, 2004. 5

[23] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016. 3, 4

[24] Srikumar Ramalingam and Matthew Brand. Lifting 3D manhattan lines from a single image. In *ICCV*, 2013. 5

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 2, 4

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015. 3

[27] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R Bradski. ORB: An efficient alternative to SIFT or SURF. In *ICCV*, 2011. 1

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 2015. 1

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[30] Richard S Stephens. Probabilistic approach to the hough transform. *Image and vision computing*, 1991. 2

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1

[32] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: A fast line segment detector with a false detection control. *PAMI*, 2010. 1, 2, 6, 8

[33] Nan Xue, Song Bai, Fudong Wang, Gui-Song Xia, Tianfu Wu, and Liangpei Zhang. Learning attraction field representation for robust line segment detection. In *CVPR*, 2019. 1, 2, 5, 6, 7, 8

[34] Fengting Yang and Zihan Zhou. Recovering 3D planes from a single image via convolutional neural networks. In *ECCV*, 2018. 1

[35] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. Bottom-up object detection by grouping extreme and center points. In *CVPR*, 2019. 2

[36] Yichao Zhou, Haozhi Qi, Yuexiang Zhai, Qi Sun, Zhili Chen, Li-Yi Wei, and Yi Ma. Learning to reconstruct 3D Manhattan wireframes from a single image. 2019. 5

[37] Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. LayoutNet: Reconstructing the 3d room layout from a single RGB image. In *CVPR*, 2018. 1