Fully Convolutional Line Parsing

Xili Dai
UESTC & UC Berkeley
daixili_cs@berkeley.edu

Xiaojun Yuan
UESTC
xjyuan@uestc.edu.cn

Haigang Gong
UESTC
hggong@uestc.edu.cn

Yi Ma
UC Berkeley
yima@eecs.berkeley.edu

Abstract

*We present a one-stage **F**ully **C**onvolutional **Li**ne **P**arsing network (F-Clip) that detects line segments from images. The proposed network is very simple and flexible with variations that gracefully trade off between speed and accuracy for different applications. F-Clip detects line segments in an end-to-end fashion by predicting them with each line's center position, length, and angle. Based on empirical observation of the distribution of line angles in real image datasets, we further customize the design of convolution kernels of our fully convolutional network to effectively exploit such statistical priors. We conduct extensive experiments and show that our method achieves a significantly better trade-off between efficiency and accuracy, resulting in a real-time line detector at up to 73 FPS on a single GPU. Such inference speed makes our method readily applicable to real-time tasks without compromising any accuracy of previous methods. Moreover, when equipped with a performance-improving backbone network, F-Clip is able to significantly outperform all state-of-the-art line detectors on accuracy at a similar or even higher frame rate. Source code https://github.com/Delay-Xili/F-Clip.*

1. Introduction

A holistic 3D representation aims to model and reconstruct a scene with high-level geometric primitives/structures such as lines, planes, and layouts [36]. Unlike representations based on local features that are usually noisy and incomplete, a holistic counterpart is arguably more compact, robust, and easy to use. This belief has motivated a line of recent works on recognizing geometric structures from image observations [9, 38, 39, 16, 15, 19, 40, 32].

Among all the geometric primitives mentioned above, lines are arguably the most important and fundamental one. An accurate line detection system is essential for many downstream vision tasks such as vanishing point detection [37], camera pose estimation [5], camera calibration [34], stereo matching [31], and even full 3D reconstruction [2, 39].

Recently, significant progress has been made in the field of line detection due to the introduction of a large-scale dataset [9], effective learning methods [38], and the community's continuing effort to develop better algorithms [29, 14, 30]. Nevertheless, most of the existing methods focus primarily on *accuracy*, and their performance drops significantly if modified for *efficiency*. In this work, our goal is to develop a flexible algorithm that can achieve the best speed-accuracy trade-off (Figure 1).

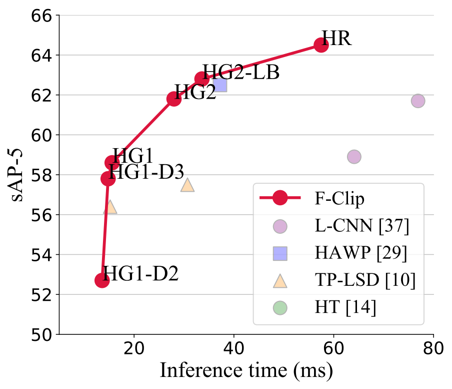


Figure 1: Speed (ms) versus accuracy (sAP^5) trade-off of state-of-the-art algorithms on the ShanghaiTech wireframe dataset.

We argue that the unsatisfactory speed-accuracy trade-off of existing methods mostly comes from the nature of the typical "two-stage" model design [38, 30]. In the first stage, thousands of line candidates are extracted. After that, based on the proposed lines, one extracts the corresponding image features and trains a small sub-network to determine whether each proposed line is correct. This approach has shown to be effective and achieves state-of-the-art accuracy so far. However, such a two-stage method sacrifices efficiency since it needs to process a large number of line candidates with an extra sub-network. Structure-wise, such a two-stage model also lacks flexibility in case we need to change the network to achieve a graceful trade-off between speed and accuracy.

As described in the seminal work [38], the "two-stage" line detection methods are motivated by "two-stage" object detection [7, 6, 24]. Beyond the prevalent two-stage methods, there has been another line of work in object detection known as "one-stage" methods. One-stage object detection

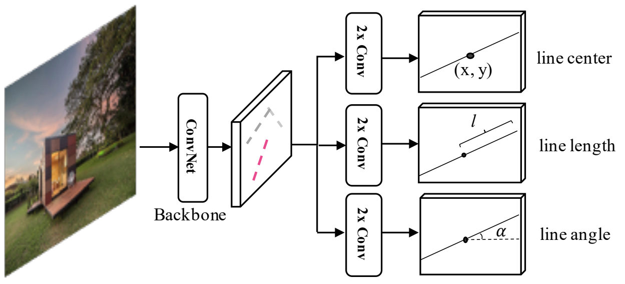


Figure 2: The overall architecture of the F-Clipnetwork. Taken an image as input, a backbone convolutional network is followed by three convolution heads that predict the line segment center, length, and angle respectively as the output.

is done using a dense sliding window, implemented by a fully convolutional network. Such one-stage methods are believed to be more flexible and efficient in certain tasks, and they can achieve more than 200 FPS with decent accuracy [21, 17, 22, 23]. Hence, in this work we take on the following question:

> *Can we achieve a better speed-accuracy trade-off in line detection by leveraging the successful ideas of one-stage methods in object detection?*

Contributions of This Paper. A key observation that motivates this work is that the extensive line proposal of the first stage used in most existing methods is not entirely necessary. Instead, a line segment can be viewed as an object and can be conveniently represented by its center, length, and angle. Hence, we can formulate the prediction of each of the parameter as a pixel-wise classification/regression problem. To this end, we propose a Fully Convolutional Line Parsing (F-Clip) network, which realizes the above idea via a fully convolutional network.

F-Clip has a surprisingly simple architecture as illustrated in Figure 2. It does not require any heavy network engineering or carefully designed training samplers as in [38]. To detect a line, our system simply applies a convolutional neural network to extract the image features and uses two additional convolution layers to regress the center, length, and angle score map. Then, for each line center with a high score, we directly output a line segment by associating the length and angle values in the same location. Due to its simple one-stage design, the network is amenable to variations and modifications for different speed-accuracy trade-off, as we will discuss more detail in Section 3.2.

Through extensive experiments on large real-world image datasets, we will see that the proposed simple method/network achieves a surprisingly good speed-accuracy trade-off. In the latency sensitive setting, F-Clip with a simple hour-glass backbone [20] can achieve 52.7 sAP^5 at 73 FPS, nearly 5 times faster than [38] at a similar accuracy. Equipped with a performance-improving backbone network [26], F-Clip can achieve 64.3 sAP^5 at 17.4 FPS, better than the state-of-the-art method [10].

2. Related Work

Line Detection. The classical approach for line detection can be dated back to the 70s'. Hough transform [4] detects lines by aggregating the pixel intensity in the parameter space of lines and output detected straight lines via a voting procedure. Modern methods such as [25, 27] detect lines based on local edge filtering. Recently, [29] utilizes deep neural networks to improve the performance of the conventional LSD line detection algorithm [27]. In our experiment, we will compare ours with this enhanced LSD algorithm [29].

Wireframe Parsing. The wireframe parsing task was first proposed in [9]. The work provided a large-scale dataset with wireframe annotations, a baseline method, and a set of evaluation metrics. After that, [38] proposed an end-to-end solution and significantly improved the performance. [33] is a counterpart work of [38] which also introduces a new annotated dataset. Meanwhile, [30] was a follow-up work of [29] which holds the state-of-the-art result. [14] designed a hough-transform based convolutional operator for the line detection task. In order to handle the topology of junctions and lines, a graph neural network based method [18] was proposed to tackle the wireframe task. Recently, LETR, [28] a transformer based approach, was proposed for line segment without heuristics-driven intermediate stages for edge and junction proposal generation. Furthermore, [39] proposed a pipeline for reconstructing 3D wireframes from 2D images. Strictly speaking, line detection is not wireframe parsing as it does not detect junctions of multiple line segments. Nevertheless, we will use the same metric proposed for the wireframe to evaluate the quality of our line segments with the endpoints being viewed as the junctions. In particular, we will compare with the state-of-the-art method in this category [30] in our experiments.

Object Detection. Recently, the performance of line detection and wireframe parsing has all been boosted by an improvement in methods for object detection. Specifically, [38] has inspired [24] and many other two-stage detection methods such as [6]. After that, [30] combined [29] with [38] has pushed the performance further. [30] and [38] are two state-of-the-art methods from 2019 and 2020, respectively. Both of them drew inspiration from the object detection community and adopted a two stage strategy. Notice that the object detection community has evolved from two-stage detector [8, 24] to one-stage detector [17, 21] or anchor free detector [35, 12]. Motivated by these work, in this paper, we propose a F-Clip network that detects line segments from images in one stage (see Figure. 2), which aims to achieve a better trade-off between speed and accuracy. During the preparation of this paper, we become aware of a very recent work [10] that casts the line detection problem as a similar learning problem of predicting three parameters for each line segment. However, they have adopted a different parameterization and a rather different network design than ours. We will discuss the differences in the next section as well as compare their algorithm with ours in the experiments.

3. Method

The overall structure of the proposed network is illustrated in Figure 2. Given an input image, we first use a convolutional neural network to extract a shared feature map. Then the map is forwarded to separate sub-networks to predict three line representation maps: the line center map, the line length map, and the line angle map. These three maps are supervised by the pixel-wise loss between prediction and ground-truth. The network is optimized end-to-end with stochastic gradient descent. Below we give a detailed description and justification for each component.

3.1. Line Representation

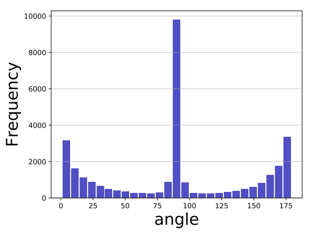
We represent a line segment by its center, length, and angle. Denote $\mathbf{p}_c \in \mathbb{R}^2$ as the center of line in the image coordinate, $\mathbf{p}_l$ and $\mathbf{p}_r \in \mathbb{R}^2$ as the left and right endpoints, respectively. We have the following relationship:

$$\mathbf{p}_l = \mathbf{p}_c + \frac{1}{2}(l\cos\alpha, l\sin\alpha) \ \in \mathbb{R}^2, \quad (1)$$

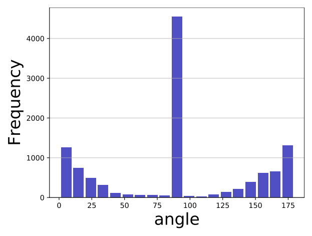
$$\mathbf{p}_r = \mathbf{p}_c - \frac{1}{2}(l\cos\alpha, l\sin\alpha) \ \in \mathbb{R}^2, \quad (2)$$

where l is the length and α is the angle between line and the horizontal direction. Any three of the above five quantities can uniquely determine a line segment. This leads to many mathematically equivalent representations of a line segment. For instance, the recent work [10] uses the center $\mathbf{p}_c$ and the x and y offsets of an end point to parameterize a segment.

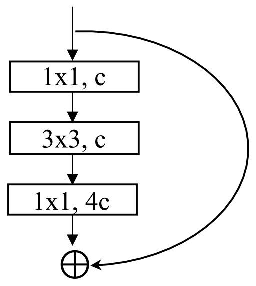
Among different choices, in this work, we choose to use the center $\mathbf{p}_c$, length l, and angle α for the following reasons: Firstly, the angle is the easiest one to predict since it can be identified accurately even from a local patch. It also has strong statistical priors (see Figure 3) that one can exploit to design more effective filters as we will elaborate more in the next subsection. In contrast to angle prediction, the network needs to perceive the whole line before making accurate predictions for length, center, and the endpoint (offsets). Secondly, since the relationship between line angle and the line's ending points is not one-to-one (one end point may be shared by multiple line segments), we choose to use line center and line length rather than the endpoints to simplify the inference. The resulting line representation gives several advantages:



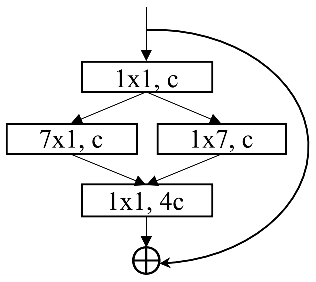
(a) angle histogram [9]



(b) angle histogram [3]



(c) conventional conv. block



(d) customized conv. block

Figure 3: The top two figures are the histograms of line angles in the ShanghaiTech dataset [9] and the YorkUrban dataset [3], respectively. The bottom two are the convolutional blocks used in our method, conventional and customized, respectively.

1. It naturally converts the line parsing problem to a pixel-wise classification/regression problem. Such transformation enables us to build a fully-convolutional structure for this task, which is both accurate and efficient (Section 3.2).
2. Due to the pixel-wise formulation, it is not necessary to sample different kinds of lines as in [38], which significantly reduces the number of hyperparameters to tune (Section 3.3).
3. The inference algorithm is straightforward. Given a predicted center location, we can directly use the corresponding predicted length and angle to get a line segment (Section 3.4).

3.2. Fully Convolutional Line Parsing Networks

Designs of Backbone Networks. There are many advantages for our one-stage network. First, a one-stage network usually can improve the efficiency. Second, the simpler architecture allows easy customization. Line detection is often used in applications with very different requirements in speed and accuracy: e.g. the real-time localization or mapping versus offline 3D modeling from images. Some applications process one image at a time whereas others may process images in batches. Hence, in this work, we provide customized versions of our F-Clip along two lines of improvement. Along the first line, in order to speed-up the network, we simplify the backbone without sacrificing much performance. Along the second line, we show how to push the overall accuracy without compromising much speed (at least in the batch processing mode).

First of all, following the work of [20, 30, 38], we adopt an hourglass network with two stack modules as our default version F-Clip (HG2). We simplify the hourglass with one stack module to get a fast version F-Clip (HG1), then reduce the number of hierarchical structure in the hourglass block, from 4 to 3 and 2, to obtain two even faster versions F-Clip (HG1-D3) and F-Clip (HG1-D2).

The second line of improvement is to increase the accuracy without sacrifice too much speed. We observe from the datasets that most line segments in man-made environments are close to being vertical or horizontal (see Figure 3 (a) and (b)). Actually, this is typically the case for most real-world line detection tasks. Based on this strong statistical prior of the angle distribution, we customize the design of the line detection blocks (see Figure 3 (d)) to exploit such priors with similar computational cost (compared to that in Figure 3 (c)). This results in a more accurate version of F-Clip (HG2-LB) (where LB is short for line block). Moreover, to further improve the performance, we exploit multiple resolutions of the input through a parallel structure such as that in the high resolution (HR) network [26], which exhibits state-of-the-art performance for many vision tasks such image segmentation, detection, and recognition. This leads to a high-performance version of F-Clip (HR).

So overall, we have 6 different versions of F-Clip: 1) F-Clip (HG1-D2); 2) F-Clip (HG1-D3); 3) F-Clip (HG1); 4) F-Clip (HG2); 5) F-Clip (HG2-LB); and 6) F-Clip (HR). Their relative accuracy and speed are illustrated in Figure 1, and their quantitative and qualitative evaluations will be given in Table 3 and Figure 5 in the experiment section.

Line Score Maps. We design the size of network output to be the same as the feature map size (128×128) to make inference efficient. However, such a design will inevitably introduce quantization errors. We address this issue by introducing a local offset parameter associated with each line center. Specifically, for a ground-truth line center $\mathbf{p}_c$ in the original image, the line center score map C is 1 only in the $\lfloor \mathbf{p}_c/s \rfloor$ coordinates. For each center location, we also predict an offset value $\mathbf{p}_c/s - \lfloor \mathbf{p}_c/s \rfloor$, a line length value l, and a line angle value α.

Prediction Head. In contrast to [10] where different network structures are used for different tasks, we intentionally minimize the design effort of each prediction head. For each branch, given the shared feature map of channel dimension c, we use two 3×3 convolutions with 256 output channels to refine the feature. Then we use a 1×1 convolution with task-specific output channel(s) to predict each quantity. The number of output channels is 2, 2, 1, 1 for center, offset, length, and angle prediction, respectively.

3.3. Training

Loss Function. We treat the problem of prediction C as a classification problem and use the focal loss [13] to tackle the unbalanced positive/negative data samples:

$$\mathcal{L}_C \doteq -\frac{1}{N}\sum_{i,j}\begin{cases} C_{i,j}(1-\hat{C}_{i,j})^{\beta}\log(\hat{C}_{i,j}) & \text{if } C_{i,j}=1, \\ (1-C_{i,j})\hat{C}_{i,j}^{\beta}\log(1-\hat{C}_{i,j}) & \text{otherwise,} \end{cases} \tag{3}$$

where β is the hyper-parameter of the focal loss, N is the number of pixels in the score map, and $\hat{C}$ is the probability of each bin after softmax operation.

Following [38], we use ℓ_2 regression to predict the offset map $\mathbf{O}$. The loss on $\mathbf{O}$ is averaged over the the number of line center points in the score map:

$$\mathcal{L}_{\mathbf{O}} \doteq -\frac{1}{N}\sum_{i,j}\|\hat{\mathbf{O}}_{i,j} - \mathbf{O}_{i,j}\|_2^2, \tag{4}$$

For line length and line angle prediction, the score map is normalized by sigmoid activation. Then we use the ℓ_1 loss between the predicted and ground-truth value (as we empirically found that ℓ_1 loss is better than ℓ_2):

$$\mathcal{L}_L \doteq -\frac{1}{N}\sum_{i,j}|\hat{L}_{i,j} - L_{i,j}|,\ \mathcal{L}_\alpha \doteq -\frac{1}{N}\sum_{i,j}|\hat{A}_{i,j} - A_{i,j}|. \tag{5}$$

The final loss used to train the network is:

$$\mathcal{L} = \lambda_C\mathcal{L}_C + \lambda_O\mathcal{L}_O + \lambda_l\mathcal{L}_l + \lambda_\alpha\mathcal{L}_\alpha. \tag{6}$$

Data Augmentation. To make the model more robust to various viewpoints and scale, we perform the following data augmentations. In the first step, an image is processed with one of the following operations with equal probability:

1. keep the original input image;
2. flip it horizontally or vertically or simultaneously;
3. rotate it by 90° clockwise or counterclockwise.

After that, we use the random expansion augmentation in [17]. Specifically, we choose a $k \times k$ region in the 512×512 zero input, and resize the image to fit inside the $k \times k$ region. We randomly sample k from $[256, 512]$. This augmentation is to enhance detection accuracy for short lines.

3.4. Inference

During inference, we firstly apply a non-maximum suppression (NMS) on the line center score map to remove duplicate line detection results [38]. Different from [38], we leverage the SoftNMS [1] from object detection to enhance the performance. Specifically,

$$C'_{i,j} = \begin{cases} C_{i,j} & \text{if } C_{i,j} = \max_{(i',j')\in\mathcal{N}(i,j)} C_{i',j'} \\ \delta \cdot C_{i,j} & \text{otherwise,} \end{cases} \tag{7}$$

where $\mathcal{N}(i,j)$ represents the 8 nearby bins around the location (i,j). Such non-maximum suppression can be implemented with a max-pooling operator. After using SoftNMS, we use the top K line centers according to their classification score. We use the corresponding predicted length and angle values to form a line according to Equation 2.

The previous step only performs NMS on the point-level, without considering the effect of length and angle of a line. We hence propose a new structural NMS (StructNMS) that removes duplicate lines with the whole line structure. Starting from the line with the highest line center score (assume its index is i), we calculate the ℓ_2 distance between its two endpoints and those of another line j:

$$\begin{aligned} d = \min\Big(&\|\mathbf{p}_l^i - \mathbf{p}_l^j\|_2^2 + \|\mathbf{p}_r^i - \mathbf{p}_r^j\|_2^2, \\ &\|\mathbf{p}_l^i - \mathbf{p}_r^j\|_2^2 + \|\mathbf{p}_r^i - \mathbf{p}_l^j\|_2^2\Big). \end{aligned} \tag{8}$$

Then we remove all the lines with d less than a predefined threshold τ. The procedure is applied for all the remaining line candidates.

4. Experiments

In this section, we present our experiment results to analyze the performance of F-Clip, as well as compare with many other state-of-the-art line detection or wireframe parsing methods.

4.1. Implementation Details

Details for Backbone Networks. In order to make our neural network adaptable to various time efficiency requirement, F-Clip uses two different frameworks for the backbone networks: the stacked hourglass network [20] for efficiency, and the HRNet [26] for performance. The stacked houglass network is a simple and elegant U-shape network that was previously used for wireframe parsing, such as in L-CNN [38] and HAWP [30]. The configuration of our stacked houglass backbones are similar to the one in [38]. The main difference is that we provide five different settings to meet different efficiency needs: 1) one stack of the hourglass network with 2 hierarchical structure in hourglass block (F-Clip (HG1-D2)); 2) one stack of the hourglass network with 3 hierarchical structure in hourglass block (F-Clip (HG1-D3)); 3) one stack of the hourglass network with 4 hierarchical structures in the hourglass block (F-Clip (HG1)); 4) two stacks of the hourglass network with 4 hierarchical structures in hourglass block, while the one in L-CNN [38] only provide the models containing 2 stacks of the hourglass networks with 4 hierarchical structures in hourglass block (F-Clip (HG2)); 5) two stacks of the hourglass network with 4 hierarchical structures in the hourglass block which residual blocks are all replaced with line block (see Figure 3 (d), F-Clip (HG2-LB)).

To further push the performance of F-Clip, we also employ the recent HRNet [26] as our backbone feature extractor (F-Clip (HR)). HRNet is originally designed for human pose estimation tasks. HRNet uses a more complex architecture design. It starts with a high-resolution subnetwork (performing convolution on high-resolution feature maps) in its initial stages, and gradually add some low-resolution subnetworks. HRNet is designed to preserve more high-resolution details. We use the HRNet-W32 variant [26] and find it performs better in F-Clip in term of accuracy, but it is much slower than the 2-stack hourglass network when batch size is equal to one.

Prediction Head. The prediction head transforms the feature maps from the backbone network into the final representations. We simply use two 3×3 convolution layers followed by a 1×1 convolution to match the output dimension. All the convolution layers are activated with ReLU non-linearity. The channel sizes of the middle 3×3 convolution are 128.

Training. We set a different initial learning rate 4×10^{-4} and 4×10^{-3} for the stack hourglass network and the high resolution network, respectively. Meanwhile, the parameter β in focal loss is also different for the two backbones ($\beta=5$ for the stack hourglass network and $\beta=4$ for the high resolution network). We choose the weights of the four loss terms in Equation (6) to be $\lambda_{C,O,l,\alpha}=\{1,0.25,3,1\}$. We train our neural network for 300 epochs, in which we decay the learning rate 10 times at the 240th epoch and the 280th epoch. All the experiments are conducted on a single NVIDIA GTX 2080Ti GPU. We use the ADAM optimizer [11]. The weight decay is set to be 1×10^{-4}. We use a batch size that maximizes the occupancy of available GPU memory.

Inference. There are two hyper-parameters in the inference stage that need to be determined (δ in Equation (7) and τ in Equation (8)). For δ, we experimented with 10 numbers in the range from 0 to 1 (with a step 0.1) and chose the best $\delta=0.8$. For τ, we experimented with 6 values (1, 2, 4, 8, 16, 32) and chose $\tau=2$ eventually.

4.2. Experimental Settings

Datasets. We train and test F-Clip on the ShanghaiTech wireframe dataset [9], which contains 5,000 training images and 462 testing images of man-made scenes. We also include York Urban dataset [3], a small dataset containing 102 images, as the testing dataset to evaluate the generalizability of different methods.

Baselines. We compare F-Clip with six baseline approaches: LSD [27], DWP [9], AFM [29], L-CNN [38], HAWP [30], and TP-LSD [10]. Five approaches are supervised deep learning-based methods. To our best knowledge, they represent the state-of-the-art in their respective category of methods. We use the pre-trained models provided by the authors of each paper for evaluation, which are also trained on the

	Backbone	$\mathcal{L}_C$	DataAug			sAP^5	sAP^{10}	sAP^{15}
			Flip	Rotate	Expand			
(a)	Hourglass	CE	✓			56.2	61.4	63.2
(b)	Hourglass	FL	✓			57.3	62.3	64.4
(c)	Hourglass	FL	✓	✓		58.4	63.3	65.3
(d)	Hourglass	FL	✓	✓	✓	61.2	65.8	67.8
(e)	HRNet	FL	✓	✓	✓	63.5	67.4	69.2

focal 焦点的

Table 1: Ablation study for our training details. CE and FL stand for cross-entropy loss and focal loss, respectively. DataAug refers to the data augmentation we used: flip, rotation, and expansion. We apply SoftNMS on the above models.

ShanghaiTech wireframe dataset.

Metric. The structural average precision (sAP) [38], proposed for evaluating accuracy in wireframe detection, uses the sum of squared error between the predicted end-points and their ground truths as evaluation metric. The predicted line segment will be counted as a true positive detection when its sum of squared error is less than a threshold, such as $\epsilon = 5$, 10, 15. AP^H was used in wireframe parsing [9]. Instead of directly using the vectorized representation of line segments, we use heatmaps generated by rasterizing line segments for both parsing results and the groundtruth.

4.3. Ablation Study

In this section, we verify the effectiveness of our proposed method through extensive experiments. All of the experiments are performed on the ShanghaiTech dataset [9] and structural AP are reported.

In Table 1, we analyze the choices of different training designs. Firstly, we show that using focal loss can improve the performance by about 1 point for all metrics, by comparing Table 1 row (a) and (b). This is because the line centers only occupy a small portion of the image, thus the ratio between positive and negative samples is very small. In this case, focal loss is effective on addressing this problem. Secondly, we show the effectiveness of our proposed rotation and expansion data augmentation. In Table 1 (c) and (d), adding rotation and expansion augmentation leads to an improvement of about 1 point and 3 points, respectively. These results show that by augmenting the data with different geometric transformations, one can obtain a more effective line detector that generalizes better.

Next, we show the influence of different inference strategies. The results are shown in Table 2. In entry (a), our method achieves 61.5 sAP^5 using the original hard NMS in [38]. Next, we apply the SoftNMS in Equation (7) and the result improves by 2 points to 63.5. This is because at this stage, only point information is utilized. Thus there may be different lines with close center locations that are mistakenly removed. Setting a lower confidence instead of completely deleting such lines maintains the potential to recover such

	SoftNMS	StructNMS	sAP^5	sAP^{10}	sAP^{15}
(a)			61.5	65.6	67.1
(b)	✓		63.5	67.4	69.2
(c)		✓	62.5	66.7	68.3
(d)	✓	✓	64.3	68.3	70.1

Table 2: Ablation study for inference details. The SoftNMS and StructNMS columns are two operators mentioned in methods. The model is an HRNet with focal loss and all the data augmentation.

mistakes. Next, we show that using the StructNMS can further improve the performance by 1 point since such mechanism takes the whole line into consideration. Combining these two new NMS mechanisms improves 3 points over the original pipeline.

4.4. Comparison with Other Methods

To ensure fair comparison with the previous state-of-the-art method HAWP [30], we have re-implemented their method with slightly better results. We have adopted the same hyper-parameter settings as our best performing model F-Clip (HR), including the focal loss, backbone, longer training epoch, and data augmentation etc. The experiment details can be found in the Appendix.

In order to make fair comparison on speed, we compute the frames per second (FPS) for different methods with their latest released code using a single GPU (RTX 2080Ti). Most of the previous methods focus on the *latency* of the algorithm (FPS while batch-size=1) but not *throughput* (FPS while batch-size=max) which is a more important metric for offline batch processing. We show the throughput metric in the last column of Table 3 which illustrates the better parallel performance of single stage methods compare with two stage methods.

Table 3 summarizes our results. Our F-Clip obtains state-of-the-art performance both in terms of *efficiency* and *accuracy*. Under the very challenging sAP^5 metric, using the same backbone network, our F-Clip (HG2) achieves comparable performance with previous state-of-the-art methods while being 1.4x faster when batch size is equal to one. By changing the backbone from HG2 to HG2-LB, we can get another 1.5 points gain without sacrifice too much speed. Moreover, ourF-Clip (HR) achieves state-of-the-art with a decent speed (17.4 FPS) and achieves 1.8 times higher throughput than HAWP [30]. Specifically, our F-Clip (HG1) is 1 points higher than another one-stage method TP-LSD, while is 1.8 times faster. When we reduce the hierarchical structure in hourglass block to 2, the speed further increases to 73 FPS. Our method not only achieves comparable speed with LSD, but is more than 8 times accurate in terms of sAP^5. Finally, F-Clip also achieves the state-of-the-art results on YorkUrban dataset which shows its generalizability.

The precision and recall curves of sAP^{10} and AP^H are shown in Figure 4. For the ShanghaiTech dataset, our method

Method	ShanghaiTech			YorkUrban			FPS	FPS
	sAP^5	sAP^{10}	AP^H	sAP^5	sAP^{10}	AP^H	batch-size=1	batch-size=10
Two-stage methods								
LSD (320) [27]	6.7	8.8	52.0	7.5	9.2	51.0	100	/
AFM [29]	18.5	24.4	69.2	7.3	9.4	48.2	13.5	/
DWP [9]	3.7	5.1	67.8	1.5	2.1	51.0	2.24	/
L-CNN [38]	58.9	62.9	80.3	24.3	26.4	58.5	15.6	16.4
HAWP [30]	62.5	66.5	84.5	26.1	28.5	60.6	26.9	81.3
HAWP (re-trained)	63.1	66.9	84.9	26.9	29.2	61.4	12.3	60.6
One-stage methods								
TP-LSD (Res34-Lite) [10]	56.4	59.7	/	24.8	26.8	/	65.9	327.6
TP-LSD (Res34) [10]	57.5	60.0	/	25.3	27.4	/	32.6	194.2
F-Clip (HG1-D2)	52.7	57.2	76.7	23.9	26.1	56.3	**73.4**	**363.1**
F-Clip (HG1-D3)	57.8	62.7	82.0	25.6	28.2	60.6	67.9	339.8
F-Clip (HG1)	58.6	63.6	83.0	24.9	27.4	60.3	64.1	324.5
F-Clip (HG2)	61.3	65.8	84.0	27.2	29.4	62.0	35.7	205.2
F-Clip (HG2-LB)	62.6	66.8	85.1	27.6	29.9	62.3	29.8	148.5
F-Clip (HR)	**64.3**	**68.3**	**85.7**	**28.5**	**30.8**	**65.0**	17.4	107.5

Table 3: Quantitative Results and Comparisons. The proposed F-Clip achieves state-of-the-art results consistently. Overall, the FPS of our F-Clip's is still significantly better than or on par with that of the six existing methods. Note that for fair apple-to-apple comparison, we have retrained the HAWP model using their latest released code and the same settings used in our F-Clip (HR). See text for details.

achieves higher recall and performs better in the higher recall regime, similarly for the YorkUrban Dataset.

4.5. Visualization

We visualize the output of our F-Clip and other three methods L-CNN, HAWP, and TP-LSD in Figure 5. The junctions are marked cyan and lines are marked orange. Wireframes from L-CNN and HAWP are post processed using the method from Appendix A.1 in [38]. Since TP-LSD and F-Clipdo not explicitly output junctions, we treat the endpoints of lines as junctions.

Both L-CNN and HAWP rely highly on the junction detection and line feature sampling, which might be prone to missing junctions or texture variations. In comparison, TP-LSD and and F-Clip are capable of detecting line segments in complicated even low-contrast environments (see third row of Figure 5). An obvious draw-back of TP-LSD is that it captures many redundant lines (see details in Figure 5).

5. Conclusions

In this work, we have introduced a one-stage fully convolutional line detection network F-Clip that directly outputs parameters of all line segments from an image. We formulate line segment detection as an end-to-end prediction of the center-point, length, and angle for each line segment. We show that by simply adjusting the backbone network, we are able to obtain a family of line detection networks that achieve state-of-the-art trade-off between accuracy and speed. We have conducted extensive experiments on large real-world datasets and demonstrated that this method outperforms the previous state-of-the-art wireframe parsing and line detection methods, by either improving the accuracy by a wide margin at the same frame rate or improving the speed by multi-fold at the same accuracy.

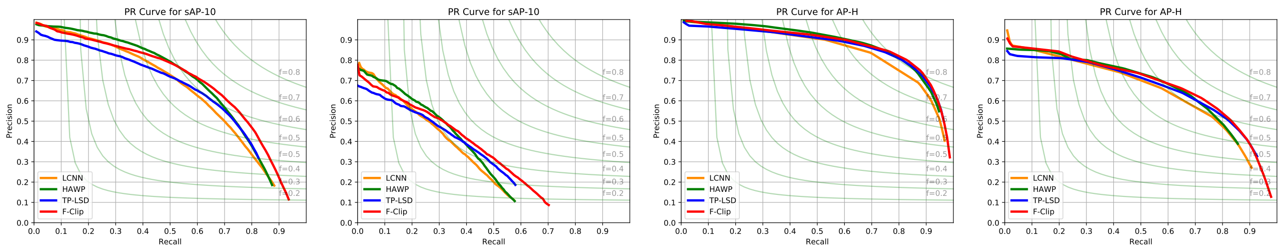
6. Acknowledgement

We thank Yichao Zhou and Haozhi Qi of Berkeley for their help during all processes include ideas, experiments, and paper writing. This paper would not have been possible without their help. We also thank Kenji Tashiro of Sony for his helpful discussions.

7. Supplementary Material

To make a fair comparison with the previous state-of-the-art method HAWP [30], we adopt the hyper-parameter settings including the 1) backbone, 2) longer training epoch, 3) data augmentation, and 4) focal loss on HAWP same as our best performance model F-Clip (HR).

Backbone. Our method employ a strong backbone network HRnet [26] (short for HR in Table 4). As shwon in Table 4, the HRnet do not bring significantly performance improvement for HAWP.



(a) sAP^{10} on ShanghaiTech (b) sAP^{10} on YorkUrban (c) AP^{H} on ShanghaiTech (d) AP^{H} on YorkUrban

Figure 4: Precision-Recall (PR) curves of sAP^{10} and AP^{H} for TP-LSD [10], L-CNN [38], HAWP [30] and F-Clip (HR) on the ShanghaiTech and the YorkUrban benchmarks respectively.

Training Epochs. Our method needs more training iterations to converge because we use a strong backbone network. As shown in Table 4 below, additional training epochs do not improve significantly the performance of state-of-the-art two-stage methods HAWP.

Data Augmentation. As shown in single-stage object detection methods [17, 35], applying a more complex data augmentation does not improve the performance of two-stage networks. We also see the same phenomena, as shown in Table 4.

Focal Loss. Focal loss [13] is designed to handle the balance between positive and negative samples. We apply the focal loss on the junction detector of HAWP. As shown in Table 4 below, the focal loss makes a bad effect on the performance of HAWP.

Analysis. Both two-stage wireframe detection methods LCNN [38] and HAWP are junction based methods. The performance of junction detection will dominate the performance of overall wireframe detection. Our F-Clip is a single stage method which skip the detection of junction and predicts the line directly. Compare with line detection, the local feature is enough for the detection of junction. Hence, hourglass backbone with a short training epoch (30 epochs) is enough for converging to a good result, a strong backbone HRnet with a longer training epoch (300 epoch) does not bring significant performance improvement for HAWP. Meanwhile, focal loss on the junction detector even makes a bad effect on the performance of HAWP.

Method	backbone	epoch	DataAug	focal-loss	sAP^{5}
HAWP	HG	30			62.5
	HG	300			62.8
	HR	300			63.1
	HR	300	✓		63.0
	HR	300	✓	✓	62.4
Ours	HR	300	✓	✓	64.5

Table 4: HAWP with longer training epochs, hrnet backbone, focal loss, and the same data augmentation as ours.

References

[1] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-NMS–improving object detection with one line of code. In *ICCV*, 2017.

[2] Patrick Denis, James H Elder, and Francisco J Estrada. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *ECCV*, 2008.

[3] Patrick Denis, James H Elder, and Francisco J Estrada. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *ECCV*, 2008.

[4] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *CACM*, 1972.

[5] Ali Elqursh and Ahmed Elgammal. Line-based relative pose estimation. In *CVPR*, 2011.

[6] Ross Girshick. Fast R-CNN. In *ICCV*, 2015.

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

[8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask-RCNN. In *ICCV*, 2017.

[9] 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.

[10] Siyu Huang, Fangbo Qin, Pengfei Xiong, Ning Ding, Yijia He, and Xiao Liu. TP-LSD: Tri-points based line segment detector. In *ECCV*, 2020.

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

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

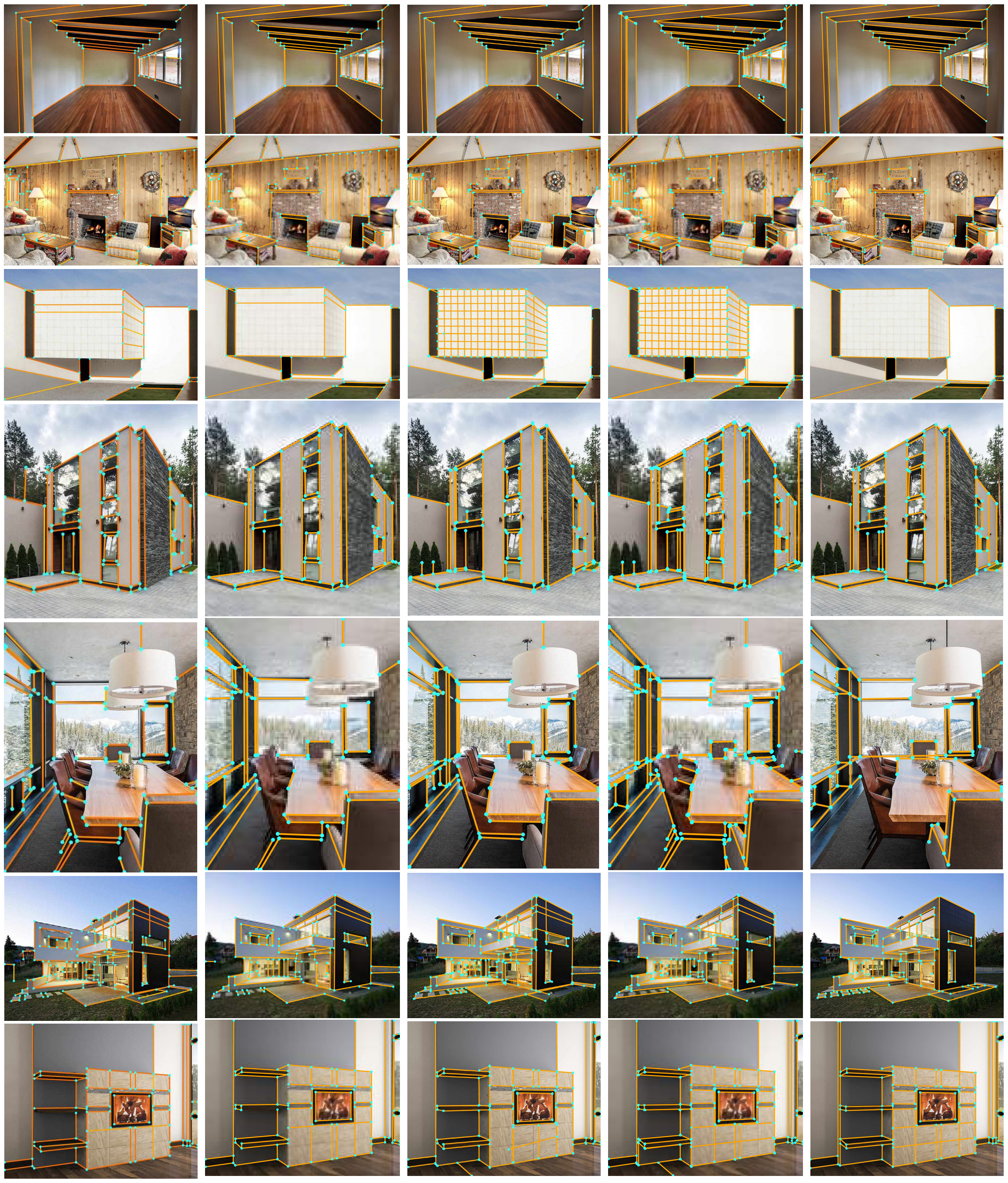


Figure 5: Qualitative evaluation of wireframe and line detection methods. From left to right, the columns correspond to the results from L-CNN [38], HAWP [30], TP-LSD [10], F-Clip (HR), and the ground truth. We also draw the detected junctions from L-CNN and HAWP and the line endpoints from TP-LSD and F-Clip.

[13] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[14] Yancong Lin, Silvia L Pintea, and Jan C van Gemert. Deep hough-transform line priors. In *ECCV*, 2020.

[15] Chen Liu, Kihwan Kim, Jinwei Gu, Yasutaka Furukawa, and Jan Kautz. Planercnn: 3d plane detection and reconstruction from a single image. In *CVPR*, 2019.

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

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *ECCV*, 2016.

[18] Quan Meng, Jiakai Zhang, Qiang Hu, Xuming He, and Jingyi Yu. LGNN: A context-aware line segment detector. In *ACMMM*, 2020.

[19] Arsalan Mousavian, Alexander Toshev, Marek Fišer, Jana Košecká, Ayzaan Wahid, and James Davidson. Visual representations for semantic target driven navigation. In *ICRA*, 2019.

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

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[22] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *CVPR*, 2017.

[23] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

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

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

[26] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, 2019.

[27] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *PAMI*, 2008.

[28] Yifan Xu, Weijian Xu, David Cheung, and Zhuowen Tu. Line segment detection using transformers without edges. *arXiv preprint arXiv:2101.01909*, 2021.

[29] 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.

[30] Nan Xue, Tianfu Wu, Song Bai, Fu-Dong Wang, Gui-Song Xia, Liangpei Zhang, and Philip H.S. Torr. Holistically-attracted wireframe parsing. In *CVPR*, 2020.

[31] Zhan Yu, Xinqing Guo, Haibing Lin, Andrew Lumsdaine, and Jingyi Yu. Line assisted light field triangulation and stereo matching. In *ICCV*, 2013.

[32] Huayi Zeng, Kevin Joseph, Adam Vest, and Yasutaka Furukawa. Bundle pooling for polygonal architecture segmentation problem. In *CVPR*, 2020.

[33] Ziheng Zhang, Zhengxin Li, Ning Bi, Jia Zheng, Jinlei Wang, Kun Huang, Weixin Luo, Yanyu Xu, and Shenghua Gao. PPGNet: Learning point-pair graph for line segment detection. In *CVPR*, 2019.

[34] Fuqiang Zhou, Yi Cui, He Gao, and Yexin Wang. Line-based camera calibration with lens distortion correction from a single image. *Optics and Lasers in Engineering*, 2013.

[35] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv*, 2019.

[36] Yichao Zhou, Jingwei Huang, Xili Dai, Linjie Luo, Zhili Chen, and Yi Ma. Holicity: A city-scale data platform for learning holistic 3d structures. *arXiv*, 2020.

[37] Yichao Zhou, Haozhi Qi, Jingwei Huang, and Yi Ma. NeurVPS: Neural vanishing point scanning via conic convolution. In *NeurIPS*, 2019.

[38] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end wireframe parsing. In *ICCV*, 2019.

[39] 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. In *ICCV*, 2019.

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