ELSD: Efficient Line Segment Detector and Descriptor

Haotian Zhang[1] Yicheng Luo[1] Fangbo Qin[2] Yijia He Xiao Liu[1]*
[1]Megvii Technology [2]Institute of Automation, CAS
{zhanghaotian,luoyicheng}@megvii.com qinfangbo2013@ia.ac.cn
heyijia2016@gmail.com liuxiao@megvii.com

Abstract

We present the novel Efficient Line Segment Detector and Descriptor (ELSD) to simultaneously detect line segments and extract their descriptors in an image. Unlike the traditional pipelines that conduct detection and description separately, ELSD utilizes a shared feature extractor for both detection and description, to provide the essential line features to the higher-level tasks like SLAM and image matching in real time. First, we design a one-stage compact model, and propose to use the mid-point, angle and length as the minimal representation of line segment, which also guarantees the center-symmetry. The non-centerness suppression is proposed to filter out the fragmented line segments caused by lines' intersections. The fine offset prediction is designed to refine the mid-point localization. Second, the line descriptor branch is integrated with the detector branch, and the two branches are jointly trained in an end-to-end manner. In the experiments, the proposed ELSD achieves the state-of-the-art performance on the Wireframe dataset and YorkUrban dataset, in both accuracy and efficiency. The line description ability of ELSD also outperforms the previous works on the line matching task.

1. Introduction

Image perception for low-level visual patterns is an essential issue for many computer vision tasks such as SLAM, Structure-from-Motion (SfM), and image matching. Local point features [4, 21, 24] are widely used in these tasks, and recently the researchers have been exploring the usage of structural features for better geometric representation[9, 10, 13, 30, 34]. Line segments are the most widely seen structural features in man-made environments. The reliable extraction of line segments and the matching across frames are important for the aforementioned tasks.

Recently, the convolutional neural networks (CNN) based line segment detection models have significantly outperformed the traditional methods. The models[33, 36, 38] consist of two stages. They first detect junctions and then generate line segment proposals and finally feed the embedding of each line segment into a classifier. Although these two-stage methods can achieve high performance, their running speed cannot satisfy real-time applications. TP-LSD[12] first realizes the compact one-stage detection by introducing the Tri-points (a root-point and two end-points) representation of line segment. However, TP-LSD predicts the two end-points separately and does not leverage the center-symmetric characteristics of the line segment. Thus, the predicted root-point might not be the exact mid-point of the two predicted end-points, and even the three points might be not co-linear. Moreover, the prediction of the root point is ambiguous especially when the lines intersect with each other so that many false root-points belonging to the fragmented line segments are detected. Besides, TP-LSD does not differentiate hard and easy examples during training. Some hard root points of line segments may not be properly detected.

Line segment descriptor is required to represent the line segment in a high-dimensional metric space, and the same line in two adjacent frames should be close in this metric

*Corresponding author. This paper is supported by the National Key R&D Plan of the Ministry of Science and Technology (Project No. 2020AAA0104400).

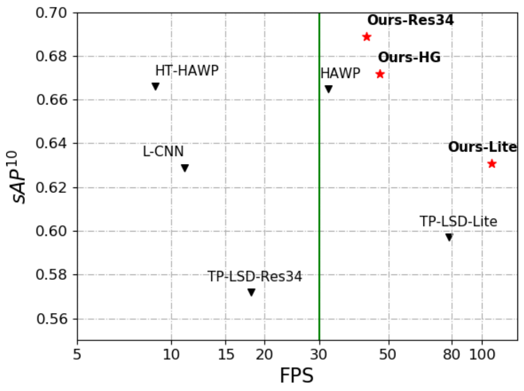

Figure 1. Inference speed (FPS) and accuracy (sAP^{10}) on Wireframe dataset.

space. There exist some CNN-based line descriptors[15, 16, 27]. However, these line descriptors are designed individually, and not yet tightly coupled with the line segment detector. It is also time-consuming to execute detection and description separately.

To this end, we propose ELSD that simultaneously predicts line segments and inferences line descriptors in an end-to-end fashion. 1) We introduce the one-stage architecture that utilizes the Center-Angle-Length (CAL) representation to vectorize a line segment. Our line detector consists of two module: (i) localization module and (ii) regression module. 2) Since the mid-points might be ambiguous for detection when lines intersect, as shown in Figure 3b, we introduce the line-centerness to filter the false mid-points belonging to fragmented line segments and adopt modified focal loss[19] to focus more on the mid-points of hard cases. 3) In the regression module, the geometric maps are predicted to provide the rotation angles and lengths. Moreover, we refine the position of the midpoints by predicting the fine offsets to compensate for the localization accuracy. 4) In the line descriptor branch, we obtain the descriptor of each predicted line segment by line pooling. The descriptor is learned by random homography-based self-supervision. The pipeline of ELSD is shown in Figure 2.

In summary, the main contributions are as follows:

- We present a pipeline that simultaneously detects line segments and inferences line descriptors in an end-to-end fashion. To the best of our knowledge, this is the first work that unifies line detector and descriptor in a compact neural network. The major computation in the backbone is shared by the two tasks, and the two task branches can be jointly training, with negligible loss on detection performance.
- We utilize the Center-Angle-Length (CAL) representation to encode a line segment that has only four parameters to predict. To overcome the detection ambiguity when lines intersect, we proposed the non-centerness suppression mechanism to remove the mid-points of fragmented line segments. The midpoint position is further refined by using the offset regression so that the line segment localization is more precise.
- Our ELSD obtains state-of-the-art performance in both accuracy and efficiency on the Wireframe and YorkUrban datasets. Moreover, the light version of our model achieves the speed of 107.5 FPS on a single GPU (RTX2080Ti) with comparable performance.

2. Related Works

2.1. Line Segment Detection

Deep learning-based line segment detection methods has attracted great attention due to the remarkable performances[11, 12, 33, 38, 39]. AFM[32] presented regional partition maps and attraction field maps of line segment maps, followed by a squeeze module to generate line segments. L-CNN[38] first proposed a two-stage pipeline for wireframe parser. It predicts junction map to generate line proposals and utilize the LoI-pooling to gather feature of the proposals. Then a line verification network classifies proposals and removes false lines. PPGNet[36] used a graph formulation to represent the relation between junctions. HAWP[33] proposed a 4-D holistic attraction field map for generating line proposals and refine the proposals with junction heat maps. HT-HAWP[20] combined Hough transform and HAWP model, obtaining excellent results in line segment detection. As the first one-stage line segment detector, TP-LSD[12] proposed a Tri-Points representation to encode line segments and predicted two endpoints of each line segment in an end-to-end manner. LETR[31] applied transformers for line segment detection from coarse-to-fine grained. Our ELSD has a similar pipeline with TP-LSD. We encode a line segment by CAL representation, and can directly detect possible semantic line segments in the image without additional classification.

2.2. Object Detection

The recent surge of some keypoint-based object detectors has achieved remarkable performance. CornerNet[17] formulated each object by a pair of corner keypoints and grouped all the detected corner keypoints to form the final detected bounding box, which requires more complicated post-processing. CenterNet[37] models an object by the center point of its bounding box, and uses keypoint estimation method to find center points and regresses to its size. FCOS[26] treats all the pixels with an object as candidate position and proposed center-ness to represent the importance of all the candidate positions. PolarNet[29] learns corner pairs based on polar coordinates and avoids the large variance of learned offsets in Cartesian coordinate. Such keypoint-based methods have good detection capabilities with a fast speed and brief structure. Motivated by these, we proposed a new line segment representation and further designed a keypoint-based line segment detector.

2.3. Line Description

Like descriptor-based keypoint matching[4, 21, 24], line matching is also based on comparing the descriptors of the same line segments in two frames. MSLD[28] constructs the line descriptors by counting the mean and variance of the gradients of pixels in the neighbor region of a line segment. LBD[35] proposes a line-band descriptor that computes gradient histograms over bands with more robustness and efficiency. Recently, some deep learning-based methods such as LLD[27], DLD[16] and WLD[15] use the convolution neural network to learn the line descriptors and achieve remarkable performance.

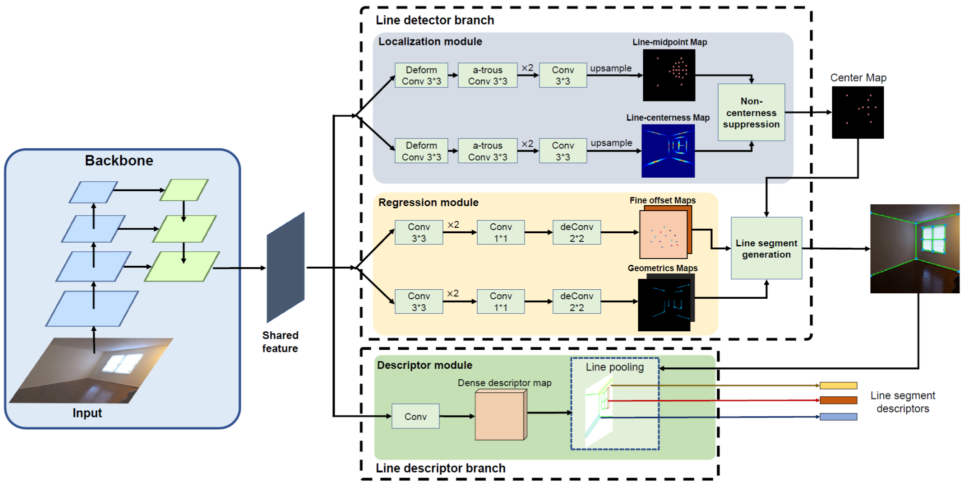

Figure 2. Illustration of the architecture of our proposed ELSD. It consists of three components: backbone, line detector branch and line descriptor branch. See text for details.

3. Methods

3.1. Line Representation

Line segments have two characteristics: 1) Due to the center-symmetry, the mid-point determines the location of the line segment, then the geometric feature is determined by the angle and length. 2) Since a line segment is straight, its direction can be consistently measured from a local part of it, which is easier to learn and requires a small receptive field. Therefore, we propose the Center-Angle-Length (CAL) representation to vectorize a line segment, which only has four parameters: 2D coordinates, rotation angle, and total length. In comparison, the Tri-points representation in TP-LSD[12] has six parameters to predict, which is redundant, and the prediction results might not satisfy the center-symmetry.

With angle θ, length ρ, and center point $\begin{bmatrix} x_c \\ y_c \end{bmatrix}$, the two endpoints of the line segment are given by,

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} + \frac{\rho}{2} \begin{bmatrix} cos\ \theta \\ sin\ \theta \end{bmatrix}$$
$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} - \frac{\rho}{2} \begin{bmatrix} cos\ \theta \\ sin\ \theta \end{bmatrix} \quad (1)$$

.

3.2. Overall Network Architecture

As shown in Figure 2, our proposed ELSD consists of a backbone, a line detector branch, and a line descriptor branch. Our backbone is a U-shape network that consists of an encoder and two decoder blocks. The backbone takes an image of size $3 \times 512 \times 512$ as input and outputs the shared feature with a size of $128 \times 128 \times 128$. After the backbone, the architecture splits into two parts: one for line detector and the other for line descriptor. The line detector branch can predict line segments from an image. We can further obtain line descriptors by feeding both shared feature and predicted line segments into the line descriptor branch. ELSD can produce line segments and further extract fixed dimensional descriptors of the line segments in a single forward pass. Moreover, unlike the traditional pipeline that first detects line segments, then computes line descriptors, ELSD shares most of the parameters between these two tasks, which reduces the computation cost and improves the compactness.

3.3. Line Detector Branch

Our line detector branch takes the shared feature from the backbone as input and splits into two modules: 1) Localization module, which consists of a line-midpoint detection head and a line-centerness detection head. In Non-Centerness-Suppression (NCS), the two heads are combined to get a more accurate center detection; 2) Regression module, which contains a geometrics regression head and a fine offset regression head. The outputs of the regression module are a pair of geometrics maps that consists of (ρ, θ) and a pair of fine offset maps. Finally, the outputs of two modules are combined together to generate the mid-points with two symmetrical endpoints as the line segment detection results.

3.3.1 Localization Module

Similar to TP-LSD[12], we use a deformable convolution, two a-trous convolution (dilation rate=2) and a standard convolution layers to obtain the adaptive spatial sampling and a large receptive field, to predict the mid-point map. Furthermore, we leverage the line-centerness, i.e. how close an on-line point lies to the mid-point, to distinct the mid-

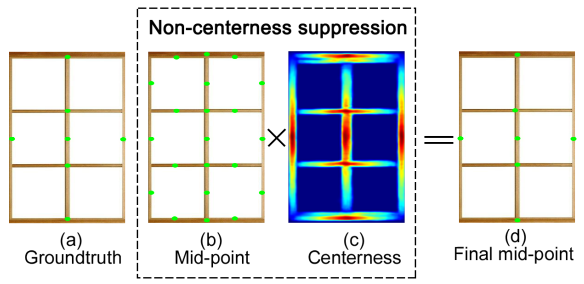

Figure 3. Illustration of Non-Centerness-Suppression (NCS). (b) and (c) show the predicted mid-point map and centerness map, respectively.

points of the entire lines and the fragmented lines. The line-centerness is calculated by,

$$P_{centerness} = \sqrt{\frac{min(d_1, d_2)}{max(d_1, d_2)}} \tag{2}$$

where d_1, d_2 are the distances from a point on the line segment to the two end-points, respectively. Apparently, $P_{centerness}$ equals 1 when the point is midpoint and decreases to 0 when the point approximates the end-points. We use a ground-truth (GT) of line segment, whose width is 3 pixel, to generate the GT of centerness map, using Eq. 2. The pixels out of the GT of line segment are assigned zeros. Note that if two line segments intersect, only the larger centerness value is assigned to the intersection pixel.

The line-centerness module has the same architecture as the localization module. Denote the predicted line-midpoint map and line-centerness map as $\widehat{P}_{mid}$ and $\widehat{P}_{centerness}$, respectively. As shown in Figure 3, we propose the Non-Centerness Suppression (NCS) to filter false local midpoints belonging to fragmented line segments, and obtain a more accurate center confidence map $\widehat{P}$, as given by,

$$\widehat{P} = \widehat{P}_{mid} \times \widehat{P}^{0.5}_{centerness} \tag{3}$$

The effectiveness of NCS is explained as follows. The midpoint detection is to obtain the exact positions but is prone to false detection caused by lines intersection. As shown in Figure 3, when a line segment is intersected with another line, its two endpoints and the intersection point form two shorter fragmented line segments. Although the mid-points of these fragmented line segments are not annotated as ground truths and are not expected to be detected, the detector tends to detect them because the fragmented line segments satisfy the definition of line segment. Differently, as visualized in Figure 3, line-centerness is not exact but provides a non-local distribution along the global line segment. The non-local distribution is more significant to inference and contains the global structure information of the potentially intersected lines. Namely, the midpoints can only mark a line segment without the awareness of the global structure, and the line-centerness map can further encode the global structure information with a non-local non-linear multi-peak 2D distribution. Therefore, the line mid-point map and line-centerness map are fused by Eq. 3 to suppress the false detection and get the final mid-points. Thus the ambiguity problem met by TP-LSD is effectively alleviated.

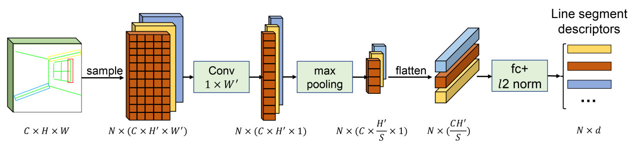
Line segment descriptors
sample
Conv $1 \times W'$
max pooling
flatten
fc+ $l2$ norm
...
$C \times H \times W$
$N \times (C \times H' \times W')$
$N \times (C \times H' \times 1)$
$N \times (C \times \frac{H'}{S} \times 1)$
$N \times (\frac{CH'}{S})$
$N \times d$

Figure 4. Illustration of Line Pooling. See text for more details.

3.3.2 Regression Module

Our regression module consists of two heads: a fine offset regression head and a geometrics regression head. The fine offset regression head is used to predict the offset of the center caused by the downsampling ratio. The refined sub-pixel mid-point can be obtained by just add the corresponding offset to the position of the predicted mid-point. The geometrics regression head can predict angle and length with respect to the midpoint. Both of our regression heads contain two 3×3, a 1×1 convolutional layers, and a deconvolutional layer. The deconvolutional layer is used to restore the size of the output map to 256×256. We can index the related angle θ and length ρ by the center position (x_c, y_c) on the output map. Then a line segment can be obtained by Eq. 1.

We utilize the CAL representation rather than Cartesian coordinates representation because the angle belongs to the geometric attributes of the line segment itself. Since the angle information can be perceived from a local part of the line segment, it is easier and more precise to predict the angle than the coordinates. We have done experiments to compare CAL representation and Cartesian coordinate representation under the same settings in Section 4.3.

3.4. Line Descriptor Branch

Given a set of line segments, the purpose of the line descriptor branch is to learn a fixed-length descriptor for each line segment, which is used to distinguish different line segments according to the distance between their descriptors. We first apply two 3×3 stride-1 convolution on the shared feature map from backbone. Then this intermediate feature map is resized to 256×256 by bilinear interpolation. The resulting feature map named dense descriptor map is used in the following Line Pooling.

Line Pooling: Similar to RoIPool[6] and RoIAlign[7] used in object detection, the Line Pooling is used to squeeze the rotated narrow ROI to a descriptor vector. As shown in

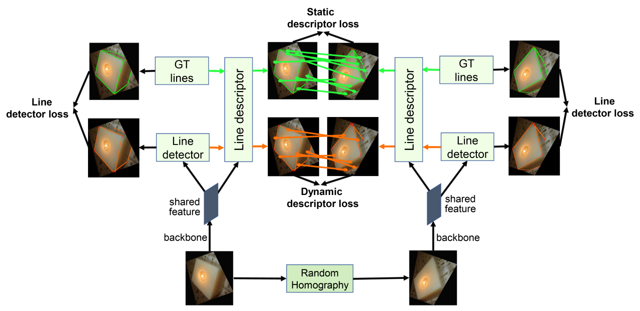

Figure 5. Training framework. The random homography is used to realize self-supervised training. See text for details.

Figure 4, the RoI of a line segment is defined as a rotated bounding box centered at the line segment, with the same length and angle as the line segment. The width of the RoI is a hyperparameter that depends on the desired size of the receptive field. Then we crop a fixed-size line feature map by sampling from the dense descriptor map using bilinear interpolation. Assuming there exist N candidate line segments and each line feature map has the size of $C \times H' \times W'$, in which C is the channel dimension of the dense descriptor map and H', W' represent the height and width of the line feature map respectively. We further apply a $1 \times W'$ stride-1 depth-wise convolution as well as a stride-S max pooling to the line feature map. Finally, the resulting feature vector is flattened and fed into a fully connected layer and then normalized, producing the final descriptor with a fixed length denoted as d.

Self-supervised learning: Similar to [4], we apply random homographies on an image to produce a paired image with different views of the same scene, assuming that planar scenes or distant scenes are common in the real environment. The homography transformation that we used is composed of a set of transformations such as translation, scale, rotation, and perspective distortion, covering most of the viewpoint change caused by camera motion. After applying random homography on the input image, we can obtain the exact image-to-image transformation. So we can label matched or unmatched line segments just by transforming the endpoint of the line segment from one image to another and checking whether the distance of two corresponding endpoints is close enough.

When training from scratch, inspired by the Line Sampling Module of L-CNN[38] that adopts static line sampler and dynamic line sampler to train the classifier, we use static line segments and dynamic line segments to train the descriptors. In the training stage, the static line segments are the annotated ground-truths, and the dynamic line segments are those predicted by the detection branch which changed as the model training proceeds. Because the line segment detection are not confident at the early training stage, we only use the detected line segments that are close enough to ground-truths as the dynamic line segments. Note that for training line descriptor branch, the proposed ELSD is trained on mini-batches of image pairs. We can obtain the ground truth correspondence of a pair of image's static line segments set during data preparation. The ground truth correspondence of a pair of dynamic line segments can be given by its closest static line segments. If the closest static line segments of a pair of dynamic line segments are matched, we then label this dynamic pair as a match and otherwise non-match. The whole training process of ELSD is shown in Figure 5. To sum up, the training with the static line segments helps to cold-start the training of descriptors at the beginning. The training with dynamic line segments helps to couple the descriptor with the actual prediction of the detector.

3.5. Loss Functions

3.5.1 Total Loss

The total loss to train ELSD is composed of the line detector loss $\mathcal{L}_p$ and line descriptor loss $\mathcal{L}_d$. Note that the input of ELSD is a pair of images with random homographies, which have both ground truth line segments, as well as the ground truth correspondence of ground truth line segments and predicted line segments. This allows us to optimize the two losses simultaneously. Given a pair of image, (I^A, I^B), and the total loss can be represented as:

$$\mathcal{L}(A,B) = \lambda_p(\mathcal{L}_p(A) + \mathcal{L}_p(B)) + \lambda_d \mathcal{L}_d(A,B) \tag{4}$$

We empirically set $\lambda_p = 0.9$, $\lambda_d = 0.1$ in this work.

3.5.2 Line Detector Loss

In the training stage of line detector branch, the outputs of four heads include line midpoint map, line-centerness map, geometrics maps, and fine offset maps. The ground truth of these maps is generated from the raw line segments label. The total loss of line segment detection is shown in Eq. (5)

$$\begin{aligned}\mathcal{L}_p(A) = &\lambda_{mid}\mathcal{L}_{mid}(A) + \lambda_{cen}\mathcal{L}_{cen}(A) + \\ &\lambda_{geo}\mathcal{L}_{geo}(A) + \lambda_{off}\mathcal{L}_{off}(A)\end{aligned} \tag{5}$$

where weights $\lambda_{mid,cen,geo,off} = \{25, 10, 1, 3\}$

Localization loss: Given an image I^A, for each ground truth midpoint p with continuous value, we construct the midpoint confidence map $P \in [0,1]^{H \times W \times 1}$ with four pixels near the midpoint by flooring and ceiling and we denote the selected pixels set by v. The 2D Gaussian kernel $G_{xy} = \exp(-\frac{(x-p_x)^2 + (y-p_y)^2}{2\sigma^2})$ is then used to compute each confidence of the pixels in v. Then we normalize these confidence by dividing the max value of v. If the confidence of a pixel is assigned more than one time, we keep the max value of it. The overall process is described by,

$$P_{xy} = max(\frac{G_{xy}}{\max\limits_{(i,j)\in v} G_{ij}}, P_{xy}) \tag{6}$$

Then we followed CornerNet[17] to use a variant of focal loss:

$$\mathcal{L}_{mid}(A) = \frac{-1}{N}\sum_{xy}^{HW}\begin{cases}(1-\widehat{P}_{xy})^{\alpha}\log(\widehat{P}_{xy}), & \text{if } P_{xy}=1\\ (1-P_{xy})^{\beta}(\widehat{P}_{xy})^{\alpha} & \text{otherwise}\\ \quad\log(1-\widehat{P}_{xy}), & \end{cases} \tag{7}$$

where α and β are hyper-parameters and N is the number of midpoints in an image. We set $\alpha = 2$ and $\beta = 4$.

According to Eq. (2), we can obtain ground truth centerness map. Then we use weighted Binary Cross Entropy (BCE) loss denoted as $\mathcal{L}_{cen}$ to supervise the learning process of the centerness.

Regression loss: Suppose the ground truth angle, length is (θ, ρ) and the corresponding predicted angle, length is $(\widehat{\theta}, \widehat{\rho})$. We use $L1$ loss and smooth $L1$ loss as the geometrics regression loss which is defined as

$$\mathcal{L}_{geo}(A) = \lambda_{ang}L1(\theta,\widehat{\theta}) + \lambda_{len}SmoothL1(\rho,\widehat{\rho}) \tag{8}$$

where $\lambda_{ang,len} = \{300, 10\}$. Besides, to recover the discretization error of midpoint coordinate caused by downsampling with ratio s, we additionally predict the fine offset maps $\widehat{O}$ for each midpoint. The offset is trained with loss.

$$\mathcal{L}_{off}(A) = \frac{1}{N}\sum_{k=1}^{N}|\widehat{O} - (\frac{p}{s} - \left\lfloor\frac{p}{s}\right\rfloor)| \tag{9}$$

where $p \in \mathbb{R}^2$ is the 2D position of GT midpoint. Note that only the midpoints where the confidence score of the ground truth equals 1 are involved in regression loss calculation.

3.5.3 Line Descriptor Loss

We utilize the triplet loss proposed in Facenet[25] to learn a line descriptor. Since the descriptors are regularized by l_2 normalization, the cosine similarity of two descriptors can be represented as $cos(d_i, d_j) = d_i^T d_j$, where d_i, d_j are two descriptors. Given image pair (I^A, I^B) and their line segments set L^A, L^B, let L_i^A, d_i^A be the i-th line segment of image I^A and its corresponding descriptor, d_i^+ be the descriptor of its matched line segment in image I^B, d_i^- be the descriptor of its unmatched line segment in image I^B with the maximal cosine similarity. Then the hard-negative triplet loss from image I^A to I^B can be represented as:

$$\mathcal{T}(A,B) = \frac{1}{N}\sum_{i=1}^{N}[m - d_i^T d_i^+ + d_i^T d_i^-]_+ \tag{10}$$

where $[x]_+ = \max\{0, x\}$. N is the number of line segments in A, m is the margin that simultaneously enhances the consistency of matched line segments and the discrepancy of unmatched line segments. As mentioned in Section 3.4, we have both static and dynamic line segments, so the overall loss of descriptor loss is:

$$\begin{aligned}\mathcal{L}_d(A,B) =& \lambda_D(\mathcal{T}^D(A,B) + \mathcal{T}^D(B,A)) +\\ & \lambda_S(\mathcal{T}^S(A,B) + \mathcal{T}^S(B,A))\end{aligned} \tag{11}$$

where $\mathcal{T}^D, \mathcal{T}^S$ represent the dynamic and static descriptor loss according to Eq. (10). We set $\lambda_D = \sqrt{\frac{e}{E}}$ and $\lambda_S = 1 - \sqrt{\frac{e}{E}}$ in this paper, where the E denotes the total epochs of entire training process and e denotes the current epoch. Briefly, we expect to rely more on static loss at the early training stage, and rely more on dynamic loss after the detector is well trained to adapt the descriptor to the actual detection result.

4. Experiments

4.1. Experiment Setting

Implementation details: We use ResNet34[8] and optionally Hourglass Network[23] as the backbone, respectively. We conduct standard data augmentation for the training set, including horizontal/vertical flip and random rotate. Input images are resized to 512×512. Our model is trained using ADAM[14] optimizer with a total of 170 epochs on four NVIDIA RTX 2080Ti GPUs and an Inter Xeon Gold 6130 2.10 GHz CPU. The initial learning rate, weight decay, and batch size are set to $1e-3$, $1e-5$, and 16 respectively. The learning rate is divided by 10 at the 100th and 150th epoch.

Datasets: We train and evaluate our model on Wireframe Dataset[11], which contains 5000 images for training and 462 images for testing. We further evaluate on YorkUrban dataset[3] with 102 test images from both indoor scenes and outdoor scenes to validate the generalization ability.

Structural Average Precision Metric[38]: The structural average precision (sAP) of the line segment is based on the L2-distance between the predicted end-points and the ground truths. The predicted line segments will be counted as True Positive if the distance is less than a certain threshold ϑ and otherwise False Positives. We set the threshold $\vartheta = 5, 10, 15$ and report the corresponding results, denote by sAP^5, sAP^{10}, sAP^{15}. For more details see [38].

Heatmap based Metric[38]: Heatmap-based F-score and average precision, F^H and AP^H are typical metrics used in wireframe parsing and line segment detection. We first convert the predicted line and ground truth line to two heatmaps by rasterizing the lines respectively. Then we can calculate the pixel-level precision and recall (PR) curves. Finally, we can compute F^H and AP^H with the PR curves.

4.2. Comparison Experiments on Line Detection

We compare our proposed ELSD with line segment detection methods and wireframe parsing methods. Our

Method	Input size	Backbone	Wireframe					YorkUrban					FPS
			sAP^5	sAP^{10}	sAP^{15}	AP^H	F^H	sAP^5	sAP^{10}	sAP^{15}	AP^H	F^H	
LSD[5]	320	/	/	/	/	55.2	62.5	/	/	/	50.9	60.1	100
AFM[32]	320	U-Net	18.5	24.4	27.5	69.2	77.2	7.3	9.4	11.1	48.2	63.3	12.8
DWP[11]	512	Hourglass	3.7	5.1	5.9	67.8	72.2	1.5	2.1	2.6	51	61.6	2.2
LETR[31]	512	ResNet101	/	65.2	67.7	86.3	**83.3**	/	29.4	31.7	62.7	66.9	/
TP-LSD-Lite[12]	320	ResNet34	56.4	59.7	/	/	80.4	24.8	26.8	/	/	**68.1**	78.2
TP-LSD[12]	512	ResNet34	57.6	57.2	/	/	80.6	27.6	27.7	/	/	67.2	18.1
L-CNN[38]	512	Hourglass	58.9	62.9	64.9	80.3 82.8†	76.9 81.3†	24.3	26.4	27.5	58.5 59.6†	63.8 65.3†	11.1
HT-HAWP[20]	512	Hourglass	62.9	66.6	/	/	/	25	27.4	/	/	/	8.9
HAWP[33]	512	Hourglass	62.5	66.5	68.2	84.5 86.1†	80.3 83.1†	26.1	28.5	29.7	60.6 61.2†	64.8 66.3†	32.1
Ours-Lite	256	ResNet34	57.4	63.1	65.5	85.6	80.2	24.3	27.4	29.3	**63.2**	63.3	**107.5**
Ours-HG	512	Hourglass	62.7	67.2	69.0	84.7	80.3	23.9	26.3	27.9	57.8	62.1	47
Ours-Res34	512	ResNet34	**64.3**	**68.9**	**70.9**	**87.2** 87.3†	82.3 83.1†	**27.6**	**30.2**	**31.8**	62.0 62.6†	63.6 64.8†	42.6

Table 1. Comparison experiments on line segment detection. '/' means the values are not reported in the related paper. '†' means the post-processing scheme proposed in L-CNN[38] is used.

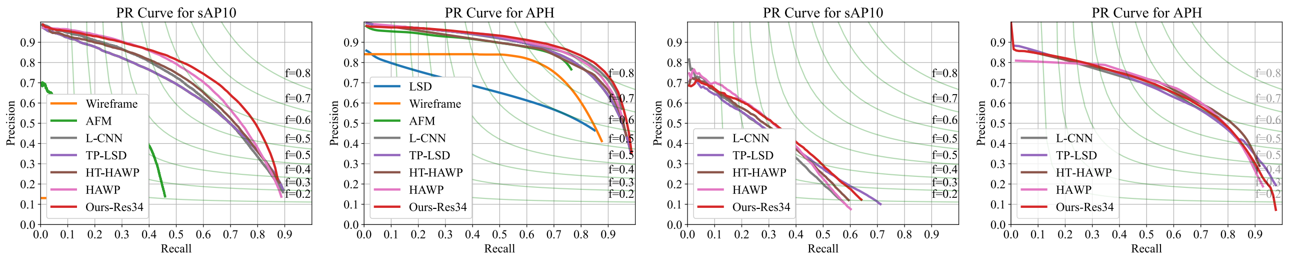

Figure 6. PR curves of sAP^{10} and AP^H on Wireframe datasets (the left two figures) and YorkUrban datasets (the right two figures). The curve of our model is depicted in red. The results of DWP, AFM, and LSD on YorkUrban datasets are not displayed since they are slightly lower than the current methods.

model use ResNet34 as backbone and for a fair comparison with other methods, we also alter the backbone with Hourglass denote by Ours-HG. Ours-Lite is a faster version of our model. In Ours-Lite, we resize the input image to 256×256 and add a decoder in backbone. Therefore the outputs maps of each head is 256×256. Table 1 shows quantitative results based on sAP, AP^H, F^H, and FPS of line segment detection.

Ours-Res34 model achieves the best sAP on two datasets at a FPS of 42.6. It outperforms HAWP by 2.3% and 1.8% in msAP(mean of sAP) metric on Wireframe and YorkUrban respectively. Besides, when we replace the backbone with an Hourglass network(Ours-HG), it stills reaches a comparable sAP results on Wireframe. Since the HAWP and L-CNN are two stage methods, their inference speeds are limited. Moreover, their line segments rely on a pair of junctions, where junctions are usually local features that contain less global information. On the other hand, benefiting from more accurate midpoint detection and a more compact line representation method, our method is superior to TP-LSD. For further comparison, we evaluate the AP of mid-points similar to Junction AP proposed in L-CNN[38]. The mean AP of mid-point of ELSD is 2.9% higher than TP-LSD, which means the mid-points are predicted more accurately in ELSD.

In terms of the heatmap-based metrics, ELSD shows advanced results in AP^H=87.2 on wireframe dataset and comparable results on F^H. Since the angle prediction error of line segments could produce a lot of incorrect pixels, the error of angle prediction has more influence on heatmap-based metrics compared to sAP. Therefore, the improvement of our model in pixel-based metrics is not as obvious as that of sAP.

Our lightweight model can reach 107.5 FPS, which is 1.4 $\sim$ 48.9 times faster than other learning-based methods while the accuracy drop is limited. We use Ours-Res34 as the representative model and depicted the precision and recall curves on both datasets in Figure 6. Our ELSD outperforms other line segment detection methods especially in sAP metric on Wireframe dataset. Besides, ELSD achieved better generalization ability on YorkUrban dataset than other two-stage methods. More qualitative evaluations are provided in Appendix 1.2.

4.3. Ablation Study for Line Detection

We run ablation experiments on the Wireframe dataset, as reported in Table 2.

NCS: NCS is to suppress the midpoints of fragmented

No.	NCS	Upsample	Focal loss	CAL	Descriptor	sAP^{5}	sAP^{10}	sAP^{15}
1	✓	✓	✓	✓		**64.3**	**68.9**	**70.9**
2	✓	✓	✓	✓	✓	64.2	68.5	70.3
3		✓	✓	✓		63.6	68.0	70.0
4	✓		✓	✓		60.3	65.8	68.2
5	✓	✓		✓		61.8	66.0	68.0
6	✓	✓	✓			62.3	67.9	70.3
7						58.0	62.8	64.8

Table 2. Ablation study of ELSD. See text for details.

Methods	Dimension	Precision(%)	Recall(%)	F-Score(%)
LBD	78	69.3	63.8	66.4
LLD	64	57.5	43.6	49.6
WLD	16	67.0	57.2	61.7
Ours	256	72.6	**77.1**	74.7
	64	**73.5**	76.2	**74.8**
	16	68.4	69.7	69.1

Table 3. Line descriptor evaluation by line matching. We show the precision, recall and F-Score for LBD, LLD, WLD, and Ours with different dimension.

line segments and remain the midpoints of entire segments. It improves the sAP^{10} from 0.680 to 0.689 according to No.3 and No.1 .

Descriptor: The multi-task learning of detection and description leads to slight reduction on the detection accuracy sAP^{10} from 0.689 to 0.685, according to No.1 and No.2 .

Upsample: For detecting line segments in real time, we use the shared feature map with 128 resolution, which is the same setting as L-CNN and HAWP. However, the prediction of the center in 128 resolution is much difficult than higher resolution. We solve this problem by upsampling the midpoint map, centerness map, geometrics maps and fine offset maps to 256 resolution. The sAP^{10} is thus improved from 0.658 to 0.689 according to No.4 and No.1. Since we only upsample once by bilinear interpolation or deconvolution, it has almost no extra cost on inference speed.

Focal loss: We use a variant focal loss instead of standard Binary Cross Entropy (BCE) loss for training the midpoint map. Since we treat the prediction of the midpoints as a binary classification problem, the focal loss that we used can have the ability to focus on the hard classified examples of midpoints. By introducing the focal loss, the sAP^{10} is improved from 0.660 to 0.689 according to No.5 and No.1 .

CAL: The proposed CAL representation is compared to the Tri-points representation in TP-LSD. The sAP^{10} is improved from 0.679 to 0.689 according to No.6 and No.1 by replacing Tri-points with the CAL representation. This is because the Tri-points need to regress more parameters than the CAL representation (4 vs 2), and the angle is easier to learn than the displacements.

4.4. Comparison Experiments on Line Description

To evaluate the line descriptor performance, we compare our method with LBD[35], LLD[27] and WLD[15]. The methods [18, 22] etc, are not involved to be compared, because they leverage the additional geometric characters of lines, other than local appearances. We test all of the algorithms on a subset of ScanNet dataset[2] which is an RGB-D video dataset annotated with 3D camera poses. We select about 1000 image pairs with large viewpoint change, rotation change, and scale change for quantitative evaluation. The GT line matches of the image pairs are obtained by checking if the reprojection error of corresponding lines is less than the certain threshold. We further compute the corresponding line descriptors of line segments detected by our model. The predicted matches of line segments can be obtained by finding the nearest neighbors over descriptors and performing cross-checking. We report the recall, precision and F-score to evaluate different descriptors. In our experiments, we use the OpenCV implementation of LBD descriptors and the official models of LLD/WLD descriptors. Meanwhile, our model is trained with setting the length of the descriptor to 256, 64, and 16 respectively.

The results are shown in Table 3. Our descriptors outperform the LBD, LLD and WLD significantly, especially in Recall. The LBD descriptor is designed by the human priority that might not be the optimal solution. The LLD descriptor and the similar learning-based descriptors[15, 16] are trained with the line segments given by the line detectors such as Edlines[1]. However, there is a gap between those detected line segments and the annotated line segments in the datasets[3, 11]. In comparison, our descriptors cooperate well with our line detector since they share most of the parameters and representation, and their training is coupled, which can further reduce computation cost. The overall inference speed of ELSD (ResNet34 as backbone) with both line detector and line descriptor can achieve 38 FPS. Moreover, the 64-dimensional descriptor presents the same result as the 256-dimensional descriptor, and is better than the 16-dimensional descriptor in accuracy. More quantitative and qualitative evaluations are provided in Appendix 1.3-1.5.

5. Conclusion

We proposes a fast and accurate model ELSD that simultaneously detects line segments and extracts descriptors in a single forward pass, allowing share computation and representation in the two tasks. To detect line segments, We first utilize the Center-Angle-Length representation to encode line segments that fully exploits the geometric characters of lines. Furthermore, a centerness map is introduced to filter the false line segments by Non-Centerness-Suppression. Our proposed line detector achieves state-of-the-art performance on two benchmarks in both accuracy and efficiency. Moreover, our model also achieves real-time speed with a single GPU. The lite model can reach the high speed of 107.5 FPS while keeping a comparable performance, and thus is useful for many higher-level tasks such as SLAM and SfM that require high real-time performance.

References

[1] C. Akinlar and C. Topal. Edlines: Real-time line segment detection by edge drawing (ed). In *2011 18th IEEE International Conference on Image Processing*, pages 2837–2840, 2011.

[2] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[3] Patrick Denis, James H. Elder, and Francisco J. Estrada. Efficient edge-based methods for estimating manhattan frames in urban imagery. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 197–210, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[4] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018.

[5] Rafael Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32:722–32, 04 2010.

[6] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[9] Yijia He, Ji Zhao, Yue Guo, Wenhao He, and Kui Yuan. Pl-vio: Tightly-coupled monocular visual–inertial odometry using point and line features. *Sensors*, 18(4):1159, 2018.

[10] Ming Hsiao, Eric Westman, Guofeng Zhang, and Michael Kaess. Keyframe-based dense planar slam. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5110–5117. IEEE, 2017.

[11] 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*, June 2018.

[12] Siyu Huang, Fangbo Qin, Pengfei Xiong, Ning Ding, Yijia He, and Xiao Liu. TP-LSD: tri-points based line segment detector. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVII*, Lecture Notes in Computer Science. Springer, 2020.

[13] Pyojin Kim, Brian Coltin, and H Jin Kim. Linear rgb-d slam for planar environments. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 333–348, 2018.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] M. Lange, Claudio Raisch, and A. Schilling. Wld: A wavelet and learning based line descriptor for line feature matching. In *VMV*, 2020.

[16] M. Lange, F. Schweinfurth, and A. Schilling. Dld: A deep learning based line descriptor for line feature matching. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5910–5915, 2019.

[17] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[18] Kai Li, Jian Yao, Xiaohu Lu, Li Li, and Zhichao Zhang. Hierarchical line matching based on line-junction-line structure descriptor and local homography estimation. *Neurocomputing*, 184, 01 2016.

[19] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[20] Yancong Lin, Silvia L. Pintea, and Jan C. van Gemert. Deep hough-transform line priors. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, 2020.

[21] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[22] QuanMeng Ma, Guang Jiang, and DianZhi Lai. Robust line segments matching via graph convolution networks. *arXiv preprint arXiv:2004.04993*, 2020.

[23] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV (8)*, pages 483–499, 2016.

[24] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[25] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[26] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proc. Int. Conf. Computer Vision (ICCV)*, 2019.

[27] A. Vakhitov and V. Lempitsky. Learnable line segment descriptor for visual slam. *IEEE Access*, 7:39923–39934, 2019.

[28] Zhiheng Wang, Fuchao Wu, and Zhanyi Hu. Msld: A robust descriptor for line matching. *Pattern Recognition*, 42(5):941–953, 2009.

[29] Wu Xiongwei, Steven HOI, and Doyen Sahoo. Polarnet: Learning to optimize polar keypoints for keypoint based object detection. In *International Conference on Learning Representations*, 2021.

[30] C. Xu, L. Zhang, L. Cheng, and R. Koch. Pose estimation from line correspondences: A complete analysis and a se-

ries of solutions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1209–1222, 2017.

[31] Yifan Xu, Weijian Xu, David Cheung, and Zhuowen Tu. Line segment detection using transformers without edges. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4257–4266, 2021.

[32] Nan Xue, Song Bai, Fudong Wang, Gui-Song Xia, Tianfu Wu, and Liangpei Zhang. Learning attraction field representation for robust line segment detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[33] Nan Xue, Tianfu Wu, Song Bai, Fudong Wang, Gui-Song Xia, Liangpei Zhang, and Philip H.S. Torr. Holistically-attracted wireframe parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[34] G. Zhang, J. H. Lee, J. Lim, and I. H. Suh. Building a 3-d line-based map using stereo slam. *IEEE Transactions on Robotics*, 31(6):1364–1377, 2015.

[35] Lilian Zhang and Reinhard Koch. An efficient and robust line segment matching approach based on lbd descriptor and pairwise geometric consistency. *Journal of Visual Communication and Image Representation*, 24(7):794–805, 2013.

[36] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[37] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[38] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end wireframe parsing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.