PRELIMINARY PREPRINT VERSION: DO NOT CITE
The AAAI Digital Library will contain the published
version some time after the conference.

Towards Light-weight and Real-time Line Segment Detection

Geonmo Gu*, **Byungsoo Ko***, **SeoungHyun Go, Sung-Hyun Lee, Jingeun Lee, Minchul Shin**
NAVER/LINE Vision
{korgm403, kobiso62, powflash, shlee.mars, jglee0206, min.stellastra}@gmail.com

Abstract

Previous deep learning-based line segment detection (LSD) suffers from the immense model size and high computational cost for line prediction. This constrains them from real-time inference on computationally restricted environments. In this paper, we propose a real-time and light-weight line segment detector for resource-constrained environments named Mobile LSD (M-LSD). We design an extremely efficient LSD architecture by minimizing the backbone network and removing the typical multi-module process for line prediction found in previous methods. To maintain competitive performance with a light-weight network, we present novel training schemes: Segments of Line segment (SoL) augmentation, matching and geometric loss. SoL augmentation splits a line segment into multiple subparts, which are used to provide auxiliary line data during the training process. Moreover, the matching and geometric loss allow a model to capture additional geometric cues. Compared with TP-LSD-Lite, previously the best real-time LSD method, our model (M-LSD-tiny) achieves competitive performance with 2.5% of model size and an increase of 130.5% in inference speed on GPU. Furthermore, our model runs at 56.8 FPS and 48.6 FPS on the latest Android and iPhone mobile devices, respectively. To the best of our knowledge, this is the first real-time deep LSD available on mobile devices. Our code is available [1].

1 Introduction

Line segments and junctions are crucial visual features in low-level vision, which provide fundamental information to the higher level vision tasks, such as pose estimation (Přibyl, Zemčík, and Čadík 2017; Xu et al. 2016), structure from motion (Bartoli and Sturm 2005; Micusik and Wildenauer 2017), 3D reconstruction (Denis, Elder, and Estrada 2008; Faugeras et al. 1992), image matching (Xue et al. 2017), wireframe to image translation (Xue, Zhou, and Huang 2019) and image rectification (Xue et al. 2019b). Moreover, the growing demand for performing such vision tasks on resource constraint platforms, like mobile or embedded devices, has made real-time line segment detection (LSD) an essential but challenging task. The difficulty arises from the limited computational power and model size when finding the best accuracy and resource-efficiency trade-offs to achieve real-time inference.

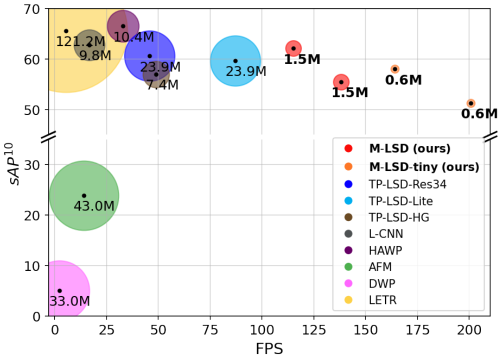


Figure 1: Comparison of M-LSD and existing LSD methods on Wireframe dataset. Inference speed (FPS) is computed on Tesla V100 GPU. Size and value of circles indicate the number of model parameters (Millions). M-LSD achieves competitive performance with the lightest model size and the fastest inference speed. Details are in Table 2.

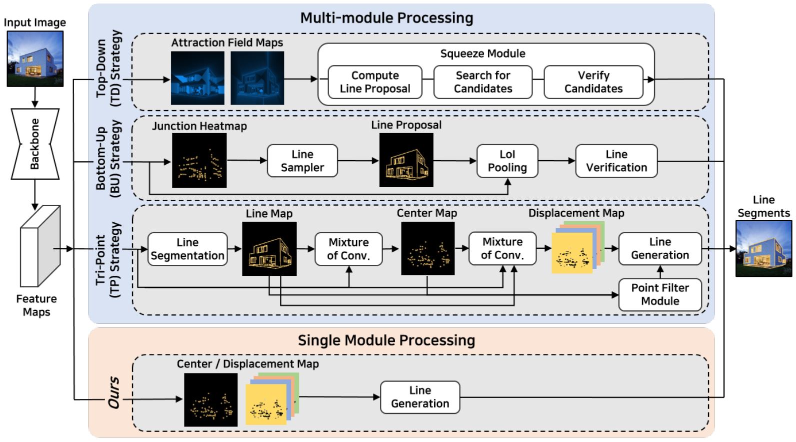
With the advent of deep neural networks, deep learning-based LSD architectures have adopted models to learn various geometric cues of line segments and have proved to show improvements in performance. As described in Figure 2, we have summarized multiple strategies that use deep learning models for LSD. The top-down strategy (Xue et al. 2019a) first detects regions of line segment with attraction field maps and then squeezes these regions into line segments to make predictions. In contrast, the bottom-up strategy first detects junctions, then arranges them into line segments, and lastly verifies the line segments by using an extra classifier (Zhou, Qi, and Ma 2019; Xue et al. 2020; Zhang et al. 2019) or a merging algorithm (Huang and Gao 2019; Huang et al. 2018). Recently, (Huang et al. 2020) proposes Tri-Points (TP) representation for a simpler process of line prediction without the time-consuming steps of line proposal and verification.

Although previous efforts of using deep networks have made remarkable achievements, real-time inference for LSD on resource-constraint platforms still remains limited. There

*Authors contributed equally.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] https://github.com/navervision/mlsd



(a) Different strategies for LSD.

Strategy	Method	Input	Inference speed (FPS)		
			Backbone	Prediction	Total
TD	AFM	320	77.1	17.3	14.1
BU	L-CNN	512	55.2	23.8	16.6
	L-CNN-P	512	55.2	0.4	0.4
	HAWP	512	55.0	82.2	32.9
TP	TP-LSD-Lite	320	138.4	234.6	87.1
	TP-LSD-Res34	320	129.0	71.0	45.8
	TP-LSD-Res34	512	128.8	23.7	20.0
	TP-LSD-HG	512	64.7	200.5	48.9
Ours	M-LSD-tiny	320	**241.1**	**1202.8**	**200.8**
	M-LSD-tiny	512	201.6	881.9	164.1
	M-LSD	320	156.3	1194.7	138.2
	M-LSD	512	132.8	883.4	115.4

(b) Inference speed on GPU.

Figure 2: (a) Previous LSD methods exploit multi-module processing for line segment prediction. In contrast, our method directly predicts line segments from feature maps with a single module. (b) Our method shows superior speed on backbone and line prediction by employing a light-weight network with a single module of line prediction.

have been attempts to present real-time LSD (Huang et al. 2020; Meng et al. 2020; Xue et al. 2020), but they still depend on server-class GPUs. This is mainly because the models that are used exploit heavy backbone networks, such as dilated ResNet50-based FPN (Zhang et al. 2019), stacked hourglass network (Meng et al. 2020; Huang et al. 2020), and atrous residual U-net (Xue et al. 2019a), which require large memory and high computational power. In addition, as shown in Figure 2, the line prediction process consists of multiple modules, which include line proposal (Xue et al. 2019a; Zhang et al. 2019; Zhou, Qi, and Ma 2019; Xue et al. 2020), line verification networks (Zhang et al. 2019; Zhou, Qi, and Ma 2019; Xue et al. 2020) and mixture of convolution module (Huang et al. 2020, 2018). As the size of the model and the number of modules for line prediction increase, the overall inference speed of LSD can become slower, as shown in Figure 2b, while demanding higher computation. Thus, increases in computational cost make it difficult to deploy LSD on resource-constraint platforms.

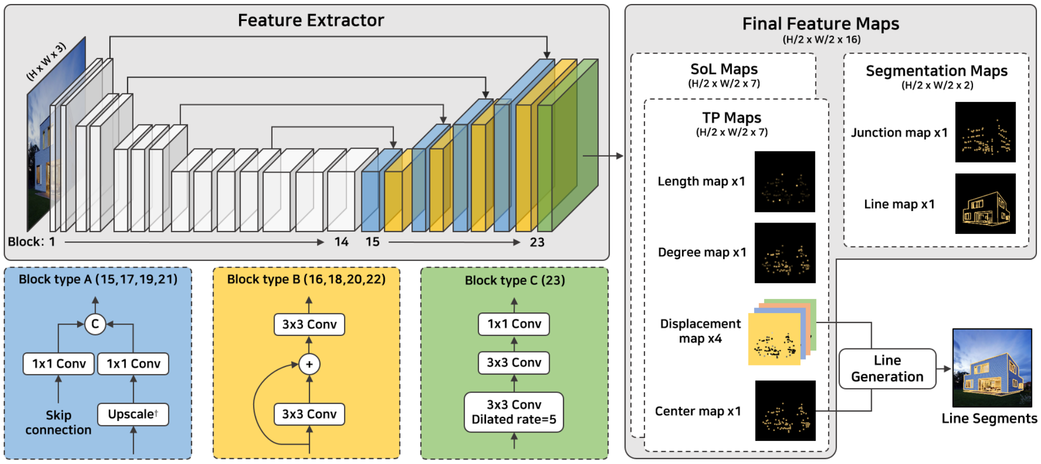
In this paper, we propose a real-time and light-weight LSD for resource-constrained environments, named Mobile LSD (M-LSD). For the network, we design a significantly efficient architecture with a single module to predict line segments. By minimizing the network size and removing the multi-module process from previous methods, M-LSD is extremely light and fast. To maintain competitive performance even with a light-weight network, we present novel training schemes: SoL augmentation, matching and geometric loss. SoL augmentation divides a line segment into subparts, which are further used to provide augmented line data during the training phase. Matching and geometric loss train a model with additional geometric information, including relation between line segments, junction and line segmentation, length and degree regression. As a result, our model is able to capture extra geometric information during training to make more accurate line predictions. Moreover, the proposed training schemes can be used with existing methods to further improve performance in a plug-and-play manner.

As shown in Figure 1, our methods achieve competitive performance and faster inference speed with a much smaller model size. M-LSD outperforms previously the real-time method, TP-LSD-Lite (Huang et al. 2020), with only 6.3% of the model size but gaining an increase of 32.5% in inference speed. Moreover, M-LSD-tiny runs in real-time at 56.8 FPS and 48.6 FPS on the latest Android and iPhone mobile devices, respectively. To the best of our knowledge, this is the first real-time LSD method available on mobile devices.

2 Related Works

Deep Line Segment Detection. There have been active studies on deep learning-based LSD. In junction-based methods, DWP (Huang et al. 2018) includes two parallel branches to predict line and junction heatmaps, followed by a merging process. PPGNet (Zhang et al. 2019) and L-CNN (Zhou, Qi, and Ma 2019) utilize junction-based line segment representations with an extra classifier to verify whether a pair of points belongs to the same line segment. Another approach uses dense prediction. AFM (Xue et al. 2019a) predicts attraction field maps that contain 2-D projection vectors representing associated line segments, followed by a squeeze module to recover line segments. HAWP (Xue et al. 2020) is presented as a hybrid model of AFM and L-CNN. Recently, (Huang et al. 2020) devises the TP line representation to remove the use of extra classifiers or heuristic post-processing found in previous methods and proposes TP-LSD network with two branches: TP extraction and line segmentation branches. Other approaches include the use of transformers (Xu et al. 2021) or Hough transform with deep networks (Lin, Pintea, and van Gemert 2020). However, it is commonly observed that the aforementioned multi-module processes restrict existing LSD to run on resource-constrained environments.

Real-time Object Detectors. Real-time object detection



† denotes that block 15 skips upscale operation.

Figure 3: The overall architecture of M-LSD. In the feature extractor, block 1 ∼ 14 are parts of MobileNetV2, and block 15 ∼ 23 are designed as a top-down architecture. The predicted line segments are generated with center and displacement maps.

has been an important task for deep learning-based object detection. Object detectors proposed in earlier days, such as RCNN-series (Girshick et al. 2014; Girshick 2015; Ren et al. 2015), consist of two-stage architecture: generating proposals in the first stage, then classifying the proposals in the second stage. These two-stage detectors typically suffer from slow inference speed and difficulty in optimization. To handle this problem, one-stage detectors, such as YOLO-series (Redmon et al. 2016; Redmon and Farhadi 2017, 2018) and SSD (Liu et al. 2016), are proposed to achieve GPU real-time inference by reducing backbone size and simplifying the two-stage process into one. This one-stage architecture has been further studied and improved to run in real-time on mobile devices (Howard et al. 2017; Sandler et al. 2018; Wang, Li, and Ling 2018; Li et al. 2018). Motivated by the transition from two-stage to one-stage architecture in object detection, we argue that the complicated multi-module processing in previous LSD can be disregarded. We simplify the line prediction process with a single module for faster inference speed and enhance the performance by the efficient training strategies; SoL augmentation, matching and geometric loss.

3 M-LSD for Line Segment Detection

In this section, we present the details of M-LSD. Our design mainly focuses on efficiency while retaining competitive performance. Firstly, we exploit a light-weight backbone and reduce the modules involved in processing line predictions for better efficiency. Next, we apply additional training schemes, including SoL augmentation, matching and geometric loss, to capture extra geometric cues. As a result, M-LSD is able to balance the trade-off between accuracy and efficiency to be well suited for mobile devices.

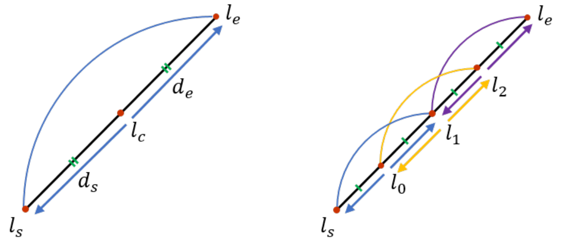
3.1 Network Architecture

We design light (M-LSD) and lighter (M-LSD-tiny) models as popular encoder-decoder architectures. In efforts to build a light-weight LSD model, our encoder networks are based on MobileNetV2 (Sandler et al. 2018) which is well-known to run in real-time on mobile environments. The encoder network uses parts of MobileNetV2 to make it even lighter. As illustrated in Figure 3, the encoder of M-LSD includes an input to 96-channel of bottleneck blocks. The number of parameters in the encoder network is 0.56M (16.5% of MobileNetV2), while the total parameters of MobileNetV2 are 3.4M. For M-LSD-tiny, a slightly smaller yet faster model, the encoder network also uses parts of MobileNetV2, including an input to 64-channel of bottleneck blocks which results in a number of 0.25M (7.4% of MobileNetV2). The decoder network is designed using a combination of block types A, B, and C. The expansive path consists of concatenation of feature maps from the skip connection and upscale from block type A, followed by two 3 × 3 convolutions with a residual connection in-between from block type B. Similarly, block type C performs two 3 × 3 convolutions, the first being a dilated convolution, followed by a 1 × 1 convolution. Please refer to the supplementary material for further details on the network architectures.

As shown in Figure 2b, we observe that one of the most critical bottlenecks in inference speed has been the prediction process, which contains multi-module processing from previous methods. In this paper, we argue that the complicated multi-module can be disregarded. As illustrated in Figure 3, we generate line segments directly from the final feature maps in a single module process. In the final feature maps, each feature map channel serves its own purpose: 1) TP maps have seven feature maps, including one length map, one degree map, one center map, and four displacement maps. 2) SoL maps have seven feature maps with the same configuration as TP maps. 3) Segmentation maps have two feature maps, including junction and line maps.

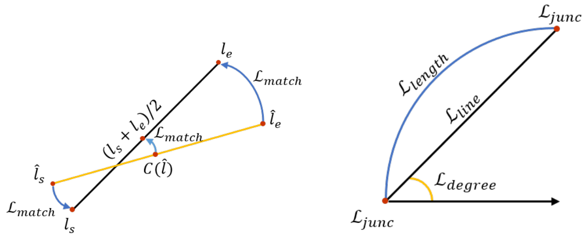
3.2 Line Segment Representation

Line segment representation determines how line segment predictions are generated and ultimately affects the ef-



(a) TP representation (b) SoL augmentation

Figure 4: Tri-Points (TP) representation and Segments of Line segment (SoL) augmentation. l_s, l_c, and l_e denote start, center, and end points, respectively. d_s and d_e are displacement vectors to start and end points. $l_0 \sim l_2$ indicates internally dividing points of the line segment $\overline{l_s l_e}$.



(a) Matching loss (b) Geometric loss

Figure 5: Matching and geometric loss. (a) Given a matched pair of a predicted line $\hat{l}$ and a GT line l, matching loss ($\mathcal{L}_{match}$) optimizes the predicted start, end, and center points. (b) Given a line segment, M-LSD learns various geometric cues: junction ($\mathcal{L}_{junc}$) and line ($\mathcal{L}_{line}$) segmentation, length ($\mathcal{L}_{length}$) and degree ($\mathcal{L}_{degree}$) regression.

ficiency of LSD. Hence, we employ the TP representation (Huang et al. 2020) which has been introduced to have a simple line generation process and shown to perform real-time LSD using GPUs. TP representation uses three key-points to depict a line segment: start, center, and end points. As illustrated in Figure 4a, the start l_s and end l_e points are represented by using two displacement vectors (d_s, d_e) with respect to the center l_c point. The line generation process, which is to convert center point and displacement vectors to a vectorized line segment, is performed as:

$$\begin{aligned}(x_{l_s}, y_{l_s}) &= (x_{l_c}, y_{l_c}) + d_s(x_{l_c}, y_{l_c}),\\ (x_{l_e}, y_{l_e}) &= (x_{l_c}, y_{l_c}) + d_e(x_{l_c}, y_{l_c}),\end{aligned} \quad (1)$$

where (x_α, y_α) denotes coordinates of an arbitrary α point. $d_s(x_{l_c}, y_{l_c})$ and $d_e(x_{l_c}, y_{l_c})$ indicate 2D displacements from the center point l_c to the corresponding start l_s and end l_e points. The center point and displacement vectors are trained with one center map and four displacement maps (one for each x and y value of the displacement vectors d_s and d_e). In the line generation process, we extract the exact center point position by applying non-maximum suppression on the center map. Next, we generate line segments with the extracted center points and the corresponding displacement vectors using a simple arithmetic operation as expressed in Equation 1; thus, making inference efficient and fast.

3.3 Matching Loss

Following (Huang et al. 2020), we use the weighted binary cross-entropy (WBCE) loss and smooth L1 loss as center loss $\mathcal{L}_{center}$ and displacement loss $\mathcal{L}_{disp}$, which are for training the center and displacement map, respectively. The line segments under the TP representation are decoupled into center points and displacement vectors, which are optimized separately. However, the coupled information of the line segment is under-utilized in the objective functions.

To resolve this problem, we present a matching loss, which leverages the coupled information w.r.t. the ground truth. As illustrated in Figure 5a, matching loss considers relation between line segments by guiding the generated line segments to be similar to the matched GT. We first take the endpoints of each prediction, which can be calculated via the line generation process, and measure the Euclidean distance $d(\cdot)$ to the endpoints of the GT. Next, these distances are used to match predicted line segments $\hat{l}$ with GT line segments l that are under a threshold γ:

$$d(l_s, \hat{l}_s) < \gamma \text{ and } d(l_e, \hat{l}_e) < \gamma, \quad (2)$$

where l_s and l_e are the start and end points of the line l, and γ is set to 5 pixels. Then, we obtain a set $\mathbb{M}$ of matched line segments (l, $\hat{l}$) that satisfies this condition. Finally, the L1 loss is used for the matching loss, which aims to minimize the geometric distance of the matched line segments w.r.t the start, end, and center points as follows:

$$\begin{aligned}\mathcal{L}_{match} \;=\; & \frac{1}{\mid \mathbb{M} \mid} \sum_{(l,\hat{l}) \in \mathbb{M}} \parallel l_s - \hat{l}_s \parallel_1 + \parallel l_e - \hat{l}_e \parallel_1 \\ & + \parallel \tilde{C}(\hat{l}) - (l_s + l_e)/2 \parallel_1,\end{aligned} \quad (3)$$

where $\tilde{C}(\hat{l})$ is the center point of line $\hat{l}$ from the center map. The total loss function for the TP map can be formulated as $\mathcal{L}_{TP} = \mathcal{L}_{center} + \mathcal{L}_{disp} + \mathcal{L}_{match}$.

3.4 SoL Augmentation

We propose Segments of Line segment (SoL) augmentation that increases the number of line segments with wider varieties of length for training. Learning line segments with center points and displacement vectors can be insufficient in certain circumstances where a line segment may be too long to manage within the receptive field size or the center points of two distinct line segments may be too close to each other. To address these issues and provide auxiliary information to the TP representation, SoL explicitly splits line segments into multiple subparts with overlapping portions of each other. An overlap between each split is enforced to preserve connectivity among the subparts.

As described in Figure 4b, we compute k internally dividing points $(l_0, l_1, \cdots, l_k)$ and separate the line segment $\overline{l_s l_e}$ into k subparts ($\overline{l_s l_1}$, $\overline{l_0 l_2}$, $\cdots$, $\overline{l_{k-1} l_e}$). Expressed in TP representation, each subpart is trained as if it is a typical line segment. The number of internally dividing points

M	Schemes	F^H	sAP^{10}	LAP
1	Baseline	74.3	48.9	48.1
2	+ Matching loss	75.4 (+1.1)	52.2 (+3.3)	52.5 (+4.4)
3	+ Geometric loss	76.2 (+0.8)	55.1 (+2.9)	55.3 (+2.8)
4	+ SoL augmentation	**77.2** (+1.0)	**58.0** (+2.9)	**57.9** (+2.6)

Table 1: Ablation study of M-LSD-tiny on Wireframe. The baseline is M-LSD-tiny trained with only TP representation. M denotes model number.

k is determined by the length of the line segment as $k = \lfloor r(l)/(\mu/2) \rceil - 1$, where $r(l)$ denotes the length of line segment l, and μ is the base length of subparts. Note that when $k \leq 1$, we do not split the line segment. The resulting length of each subpart can be similar to μ with small margins of error due to the rounding function $\lfloor \cdot \rceil$, and we empirically set $\mu = input\_size \times 0.125$. The loss function of $\mathcal{L}_{SoL}$ follows the same configuration as $\mathcal{L}_{TP}$, while each subpart is treated as an individual line segment. Note that the line generation process is only done in TP maps, not in SoL maps.

3.5 Learning with Geometric Information

To boost the quality of predictions, we incorporate various geometric information about line segments which helps the overall learning process. In this section, we present learning LSD with junction and line segmentation, and length and degree regression for additional geometric information.

Junction and Line Segmentation Center point and displacement vectors are highly related to pixel-wise junctions and line segments in the segmentation maps of Figure 3. For example, end points, derived from the center point and displacement vectors, should be the junction points. Also, center points must be localized on the pixel-wise line segment. Thus, learning the segmentation maps of junctions and line segments works as a spatial attention cue for LSD. As illustrated in Figure 3, M-LSD contains segmentation maps, including a junction map and a line map. We construct the junction GT map by scaling with Gaussian kernel as the center map, while using a binary map for line GT map. The total segmentation loss is defined as $\mathcal{L}_{seg} = \mathcal{L}_{junc} + \mathcal{L}_{line}$, where we use WBCE loss for both $\mathcal{L}_{junc}$ and $\mathcal{L}_{line}$.

Length and Degree Regression As displacement vectors can be derived from the length and degree of line segments, they can be additional geometric cues to support the displacement maps. We compute the length and degree from the ground truth and mark the values on the center of line segments in each GT map. Next, these values are extrapolated to a 3×3 window so that all neighboring pixels of a given pixel contain the same value. As shown in Figure 3, we maintain predicted length and degree maps for both TP and SoL maps, where TP uses the original line segment and SoL uses augmented subparts. As the ranges of length and degree are wide, we divide each length by the diagonal length of the input image for normalization. For degree, we divide each degree by 2π and add 0.5. The total regression loss can be formulated as $\mathcal{L}_{reg} = \mathcal{L}_{length} + \mathcal{L}_{degree}$, where we use smooth L1 loss for both $\mathcal{L}_{length}$ and $\mathcal{L}_{degree}$.

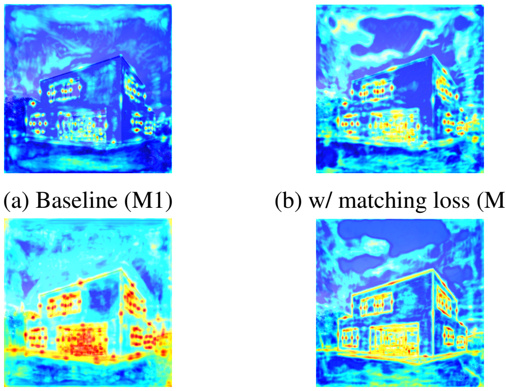

(a) Baseline (M1)
(b) w/ matching loss (M2)
(c) w/ geometric loss (M3)
(d) w/ SoL augmentation (M4)

Figure 6: Saliency maps generated from TP center map. Model numbers (M1∼4) are from Table 1.

3.6 Final Loss Functions

The geometric loss function is defined as the sum of segmentation and regression loss:

$$\mathcal{L}_{Geo} = \mathcal{L}_{seg} + \mathcal{L}_{reg}. \tag{4}$$

The loss function for SoL maps $\mathcal{L}_{SoL}$ follows the same formulation as $\mathcal{L}_{TP}$ but with SoL augmented GT. Finally, we obtain the final loss function to train M-LSD as follows:

$$\mathcal{L}_{total} \quad = \quad \mathcal{L}_{TP} + \mathcal{L}_{SoL} + \mathcal{L}_{Geo}. \tag{5}$$

Please refer to the supplementary material for further details on the feature maps and losses.

4 Experiments

In this section, we conduct extensive ablation studies, quantitative and qualitative analysis of the proposed method. For better understanding, we add extended experiments in the supplementary material, including ablation study of architecture, SoL augmentation, application example and so on.

4.1 Experimental Setting

Dataset and Evaluation Metrics. We evaluate our model with two famous LSD datasets: *Wireframe* (Huang et al. 2018) and *YorkUrban* (Denis, Elder, and Estrada 2008). The Wireframe dataset consists of 5,000 training and 462 test images of man-made environments, while the YorkUrban dataset has 102 test images. Following the typical training and test protocol (Huang et al. 2020; Zhou, Qi, and Ma 2019), we train our model with the training set from the Wireframe dataset and test with both Wireframe and YorkUrban datasets. We evaluate our models using prevalent metrics for LSD (Huang et al. 2020; Zhang et al. 2019; Meng et al. 2020; Xue et al. 2019a; Zhou, Qi, and Ma 2019) that include: heatmap-based metric F^H, structural average precision (sAP), and line matching average precision (LAP).

Optimization. We train our model on Tesla V100 GPU. We use the TensorFlow (Abadi et al. 2016) framework for model training and TFLite [2] for porting models to mobile

[2] www.tensorflow.org/lite

Methods	Input	Wireframe				YorkUrban				Params(M)	FPS
		F^H	sAP^5	sAP^{10}	LAP	F^H	sAP^5	sAP^{10}	LAP		
LSD (Von Gioi et al. 2008)	320	64.1	6.7	8.8	18.7	60.6	7.5	9.2	16.1	-	100.0[†]
DWP (Huang et al. 2018)	512	72.7	3.7	5.1	6.6	65.2	2.8	2.6	3.1	33.0	2.2
AFM (Xue et al. 2019a)	320	77.3	18.3	23.9	36.7	66.3	7.0	9.1	17.5	43.0	14.1
LGNN (Meng et al. 2020)	512	-	-	62.3	-	-	-	-	-	-	15.8[‡]
LGNN-lite (Meng et al. 2020)	512	-	-	57.6	-	-	-	-	-	-	34.0[‡]
TP-LSD-Lite (Huang et al. 2020)	320	80.4	56.4	59.7	59.7	**68.1**	24.8	26.8	31.2	23.9	87.1
TP-LSD-Res34 (Huang et al. 2020)	320	81.6	57.5	60.6	60.6	67.4	25.3	27.4	31.1	23.9	45.8
TP-LSD-Res34 (Huang et al. 2020)	512	80.6	57.6	57.2	61.3	67.2	**27.6**	27.7	**34.3**	23.9	20.0
TP-LSD-HG (Huang et al. 2020)	512	82.0	50.9	57.0	55.1	67.3	18.9	22.0	24.6	7.4	48.9
LETR (Xu et al. 2021)	1100*	**82.6**	59.2	65.6	**65.1**	66.6	24.0	27.6	32.5	121.2	5.4
L-CNN (Zhou, Qi, and Ma 2019)	512	77.5	58.9	62.8	59.8	64.6	25.9	28.2	32.0	9.8	16.6
HAWP (Xue et al. 2020)	512	80.3	62.5	66.5	62.9	64.8	26.1	**28.5**	30.4	10.4	32.9
HT-L-CNN (Lin, Pintea, and van Gemert 2020)	512	-	60.3	64.2	-	-	25.7	28.0	-	9.3	7.5[‡]
HT-HAWP (Lin, Pintea, and van Gemert 2020)	512	-	62.9	66.6	-	-	25.0	27.4	-	10.5	12.2[‡]
L-CNN + *M-LSD-s*	512	80.7	59.4	63.7	63.8	66.5	27.5	28.1	31.7	9.8	16.6
HAWP + *M-LSD-s*	512	82.5	**63.3**	**67.1**	64.2	66.7	27.5	**28.5**	32.4	10.4	32.9
M-LSD-tiny	320	76.8	43.0	51.3	50.1	61.9	17.4	21.3	23.7	**0.6**	**200.8**
M-LSD-tiny	512	77.2	52.3	58.0	57.9	62.4	22.1	25.0	28.3	**0.6**	164.1
M-LSD	320	78.7	48.2	55.5	55.7	63.4	20.2	23.9	27.7	1.5	138.2
M-LSD	512	80.0	56.4	62.1	61.5	64.2	24.6	27.3	30.7	1.5	115.4

Table 2: Quantitative comparisons with existing LSD methods. FPS is evaluated in Tesla V100 GPU, where [†] denotes CPU FPS and [‡] denotes the values from the corresponding paper due to no published or incomplete implementation. $*$ denotes resizing the image with the shortest side at least 1100 pixels. M-LSD-s indicates the proposed training schemes. The best scores among previous methods, our models, and all together are marked in blue, red, and **bold**, respectively.

devices. Input images are resized to 320 × 320 or 512 × 512 in both training and testing, which are specified in each experiment. The input augmentation consists of horizontal and vertical flips, shearing, rotation, and scaling. We use ImageNet (Deng et al. 2009) pre-trained weights on the parts of MobileNetV2 (Sandler et al. 2018) in M-LSD and M-LSD-tiny. Our model is trained using the Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.01. We use linear learning rate warm-up for 5 epochs and cosine learning rate decay (Loshchilov and Hutter 2016) from 70 epoch to 150 epoch. We train the model for a total of 150 epochs with a batch size of 64.

4.2 Ablation Study and Interpretability

We conduct a series of ablation experiments to analyze our proposed method. M-LSD-tiny is trained and tested on the Wireframe dataset with an input size of 512 × 512. As shown in Table 1, all the proposed schemes contribute to a significant performance improvement. In addition, we include saliency map visualizations generated from each feature map to analyze networks learned from each training scheme in Figure 6 using GradCam (Selvaraju et al. 2017). The saliency map interprets important regions and importance levels on the input image by computing the gradients from each feature map.

Matching Loss. Integrating matching loss shows performance boosts on both pixel localization accuracy and line prediction quality. We observe weak attention on center points from the baseline saliency maps in Figure 6a, while w/ matching loss amplifies the attention on center points in Figure 6b. This demonstrates that training with coupled information of center points and displacement vectors allows the model to learn with more line-awareness features.

Geometric Loss. Adding geometric loss gives performance boosts in every metric. Moreover, the saliency map of Figure 6c shows more distinct and stronger attention on center points and line segments as compared to that of saliency maps w/ matching loss in Figure 6b. It shows that geometric information work as spatial attention cues for training.

SoL Augmentation. Integrating SoL augmentation shows significant performance boost. In the saliency maps of Figure 6c, w/ geometric loss shows strong but vague attention on center points with disconnected line attention for long line segments. This can be a problem because the entire line information is essential to compute the center point. In contrast, w/ SoL augmentation in Figure 6d shows more precise center point attention as well as clearly connected line attention. This demonstrates that augmenting line segments by the number and length guides the model to be more robust in pixel-based and line matching-based qualities.

4.3 Comparison with Other Methods

As shown in Table 2, we conduct experiments that combine the proposed training schemes (SoL augmentation, matching and geometric loss) with existing methods. Finally, we compare our proposed M-LSD and M-LSD-tiny with the previous state-of-the-art methods.

Existing methods with M-LSD Training Schemes. As our proposed training schemes can be used with existing LSD methods, we demonstrate this using L-CNN and HAWP following Deep Hough Transform (HT) (Lin, Pintea, and van Gemert 2020), a recently proposed combinable method. L-CNN + HT (HT-L-CNN) shows a performance boost of 1.4% while L-CNN + M-LSD-s shows a boost of 0.9% in sAP^{10}. HAWP + HT (HT-HAWP) shows 0.1% of performance boost, while HAWP + M-LSD-s shows 0.6% of performance boost in sAP^{10}, which makes the combination one of the state-of-the-art performance. Thus, it demonstrates that the proposed training schemes are flexible and powerful to use with existing LSD methods.

M-LSD and M-LSD-tiny. Our proposed models achieve competitive performance and the fastest inference speed

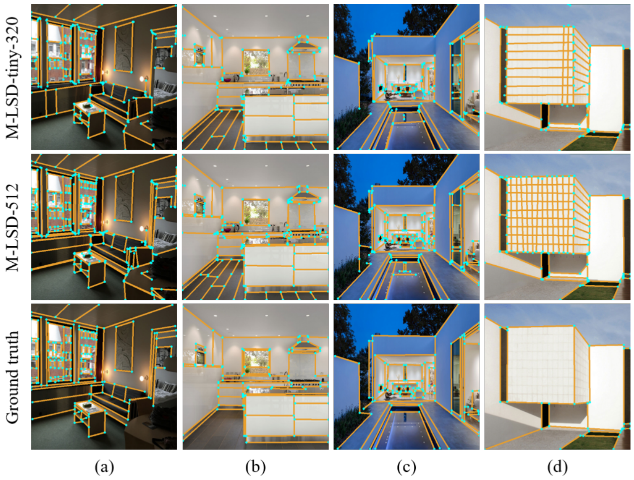


Figure 7: Qualitative evaluation of M-LSD-tiny and M-LSD on WireFrame dataset.

even with a limited model size. In comparison with the previous fastest model, TP-LSD-Lite, M-LSD with input size of 512 shows higher performance and an increase of 32.5% in inference speed with only 6.3% of the model size. Our fastest model, M-LSD-tiny with 320 input size, has a slightly lower performance than that of TP-LSD-Lite, but achieves an increase of 130.5% in inference speed with only 2.5% of the model size. Compared to the previous lightest model TP-LSD-HG, M-LSD with 512 input size outperforms on sAP^5, sAP^{10} and LAP with an increase of 136.0% in inference speed with 20.3% of the model size. Our lightest model, M-LSD-tiny with 320 input size, shows an increase of 310.6% in the inference speed with 8.1% of the model size compared to TP-LSD-HG. Previous methods can be deployed as real-time line segment detectors on server-class GPUs, but not on resource-constrained environments either because the model size is too large or the inference speed is too slow. Although M-LSD does not achieve state-of-the-art performance, it shows competitive performance and the fastest inference speed with the smallest model size, offering the potential to be used in real-time applications on resource-constrained environments, such as mobile devices.

4.4 Visualization

We visualize outputs of M-LSD and M-LSD-tiny in Figure 7. Junctions and line segments are colored with cyan blue and orange, respectively. Compared to the GT, both models are capable of identifying junctions and line segments with high precision even in complicated low contrast environments such as (a) and (c). Although the results of M-LSD-tiny may have a few small line segments missing and junctions incorrectly connected, the fundamental line segments to identify the environmental structure are accurate.

The goal of our model is to detect the structural line segments as (Huang et al. 2018) while avoiding texture and photometric line segments. However, we observe that some are included in our results, such as texture on the floor in (b) and shadow on the wall in (d). We acknowledge this to be a common problem for existing methods, and considering texture and photometric features for training would be great future work. We include more visualizations with a comparison of existing methods in the supplementary material.

Model	Input	Device	FP	Latency (ms)	FPS	Memory (MB)
M-LSD-tiny	320	iPhone	32	30.6	32.7	169
			16	20.6	48.6	111
		Android	32	31.0	32.3	103
			16	**17.6**	**56.8**	**78**
	512	iPhone	32	51.6	19.4	203
			16	36.8	27.1	176
		Android	32	55.8	17.9	195
			16	**25.4**	**39.4**	**129**
M-LSD	320	iPhone	32	74.5	13.4	241
			16	46.4	21.6	188
		Android	32	82.4	12.1	236
			16	**38.4**	**26.0**	**152**
	512	iPhone	32	121.6	8.2	327
			16	90.7	11.0	**261**
		Android	32	177.3	5.6	508
			16	**79.0**	**12.7**	289

Table 3: Inference speed and memory usage on iPhone (A14 Bionic chipset) and Android phone (Snapdragon 865 chipset). FP denotes floating point.

4.5 Deployment on Mobile Devices

We deploy M-LSD on mobile devices and evaluate the memory usage and inference speed. We use iPhone 12 Pro with A14 bionic chipset and Galaxy S20 Ultra with Snapdragon 865 ARM chipset. As shown in Table 3, M-LSD-tiny and M-LSD are small enough to be deployed on mobile devices where memory requirements range between 78MB and 508MB. The inference speed of M-LSD-tiny is fast enough to be real-time on mobile devices where it ranges from a minimum of 17.9 FPS to a maximum of 56.8 FPS. M-LSD still can be real-time with 320 input size, however, with 512 input size, FP16 may be required for a faster FPS over 10. Overall, as all our models have small memory requirements and fast inference speed on mobile devices, the exceptional efficiency allows M-LSD variants to be used in real-world applications. To the best of our knowledge, this is the first and the fastest real-time line segment detector on mobile devices ever reported.

5 Conclusion

We introduce M-LSD, a light-weight and real-time line segment detector for resource-constrained environments. Our model is designed with a significantly efficient network architecture and a single module process to predict line segments. To maintain competitive performance even with a light-weight network, we present novel training schemes: SoL augmentation, matching and geometric loss. As a result, our proposed method achieves competitive performance and the fastest inference speed with the lightest model size. Moreover, we show that M-LSD is deployable on mobile devices in real-time, which demonstrates the potential to be used in real-time mobile applications.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Bartoli, A.; and Sturm, P. 2005. Structure-from-motion using lines: Representation, triangulation, and bundle adjustment. *Computer vision and image understanding*, 100(3): 416–441.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Denis, P.; Elder, J. H.; and Estrada, F. J. 2008. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *European conference on computer vision*, 197–210. Springer.

Faugeras, O. D.; Deriche, R.; Mathieu, H.; Ayache, N.; and Randall, G. 1992. The depth and motion analysis machine. In *Parallel Image Processing*, 143–175. World Scientific.

Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 1440–1448.

Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, K.; and Gao, S. 2019. Wireframe parsing with guidance of distance map. *IEEE Access*, 7: 141036–141044.

Huang, K.; Wang, Y.; Zhou, Z.; Ding, T.; Gao, S.; and Ma, Y. 2018. Learning to parse wireframes in images of man-made environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 626–635.

Huang, S.; Qin, F.; Xiong, P.; Ding, N.; He, Y.; and Liu, X. 2020. TP-LSD: Tri-Points Based Line Segment Detector. *arXiv preprint arXiv:2009.05505*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, Y.; Li, J.; Lin, W.; and Li, J. 2018. Tiny-DSOD: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*.

Lin, Y.; Pintea, S. L.; and van Gemert, J. C. 2020. Deep hough-transform line priors. In *European Conference on Computer Vision*, 323–340. Springer.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*, 21–37. Springer.

Loshchilov, I.; and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Meng, Q.; Zhang, J.; Hu, Q.; He, X.; and Yu, J. 2020. LGNN: A Context-aware Line Segment Detector. In *Proceedings of the 28th ACM International Conference on Multimedia*, 4364–4372.

Micusik, B.; and Wildenauer, H. 2017. Structure from motion with line segments under relaxed endpoint constraints. *International Journal of Computer Vision*, 124(1): 65–79.

Přibyl, B.; Zemčík, P.; and Čadík, M. 2017. Absolute pose estimation from line correspondences using direct linear transformation. *Computer Vision and Image Understanding*, 161: 130–144.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Redmon, J.; and Farhadi, A. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7263–7271.

Redmon, J.; and Farhadi, A. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, 618–626.

Von Gioi, R. G.; Jakubowicz, J.; Morel, J.-M.; and Randall, G. 2008. LSD: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4): 722–732.

Wang, R. J.; Li, X.; and Ling, C. X. 2018. Pelee: A real-time object detection system on mobile devices. *arXiv preprint arXiv:1804.06882*.

Xu, C.; Zhang, L.; Cheng, L.; and Koch, R. 2016. Pose estimation from line correspondences: A complete analysis and a series of solutions. *IEEE transactions on pattern analysis and machine intelligence*, 39(6): 1209–1222.

Xu, Y.; Xu, W.; Cheung, D.; and Tu, Z. 2021. Line segment detection using transformers without edges. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4257–4266.

Xue, N.; Bai, S.; Wang, F.; Xia, G.-S.; Wu, T.; and Zhang, L. 2019a. Learning attraction field representation for robust line segment detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1595–1603.

Xue, N.; Wu, T.; Bai, S.; Wang, F.; Xia, G.-S.; Zhang, L.; and Torr, P. H. 2020. Holistically-attracted wireframe parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2788–2797.

Xue, N.; Xia, G.-S.; Bai, X.; Zhang, L.; and Shen, W. 2017. Anisotropic-scale junction detection and matching for indoor images. *IEEE Transactions on Image Processing*, 27(1): 78–91.

Xue, Y.; Zhou, Z.; and Huang, X. 2019. Neural Wireframe Renderer: Learning Wireframe to Image Translations. *arXiv preprint arXiv:1912.03840*.

Xue, Z.; Xue, N.; Xia, G.-S.; and Shen, W. 2019b. Learning to calibrate straight lines for fisheye image rectification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1643–1651.

Zhang, Z.; Li, Z.; Bi, N.; Zheng, J.; Wang, J.; Huang, K.; Luo, W.; Xu, Y.; and Gao, S. 2019. Ppgnet: Learning point-pair graph for line segment detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7105–7114.

Zhou, Y.; Qi, H.; and Ma, Y. 2019. End-to-end wireframe parsing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 962–971.