

YOLinO: Generic Single Shot Polyline Detection in Real Time

Annika Meyer Philipp Skudlik* Jan-Hendrik Pauls* Christoph Stiller
 Institute of Measurement & Control Systems, Karlsruhe Institute of Technology
 annika.meyer@kit.edu

Abstract

The detection of polylines is usually either bound to branchless polylines or formulated in a recurrent way, prohibiting their use in real-time systems.

We propose an approach that builds upon the idea of single shot object detection. Reformulating the problem of polyline detection as a bottom-up composition of small line segments allows to detect bounded, dashed and continuous polylines with a single head. This has several major advantages over previous methods. Not only is the method at 187 fps more than suited for real-time applications with virtually any restriction on the shapes of the detected polylines. By predicting multiple line segments for each cell, even branching or crossing polylines can be detected.

We evaluate our approach on three different applications for road marking, lane border and center line detection. Hereby, we demonstrate the ability to generalize to different domains as well as both implicit and explicit polyline detection tasks.

1. Introduction

Polylines are ubiquitous in many applications. They enable a generic representation of real world objects like lane markings, curbs etc. However, the recognition of polylines in images is usually bound to branchless polylines or formulated in a recurrent way. Moreover, many approaches can only detect a single polyline, while the majority of applications would need a multitude (*e.g.* lane markings in Figure 1).

Learning the detection of polylines so far required recurrent architectures, because the number of polylines is often unknown and every polyline contains an indefinite number of vertices (*e.g.* [18]). At the same time, recurrent architectures are difficult to train and are comparably slow. Feed-forward neural networks, however, need a well-defined rigid output format that seems incompatible with the variable nature of polylines at first sight.



Figure 1. YOLinO detects generic polylines as a bottom-up composition. The image is split into spatial cells each predicting multiple line segments. As one advantage, dashed and solid road markings can be detected using the same head. Aerial image: © City of Karlsruhe | Liegenschaftsamt

We propose an approach that is inspired by single-shot object detection [34, 35] and is able to detect bounded, dashed and continuous polylines in real time without a recurrent architecture. Instead of recurrent node proposals, we reformulate the problem of polyline detection as bottom-up composition of small line segments. First, the image is fed into a neural network that predicts multiple line segment proposals for independent spatial cells (cf. Figure 1). Later, the line segments can be combined into polylines of virtually any shape or topology.

The explicit encoding of start and endpoints allows predicted lines to carry directional information. Furthermore, they can be enhanced with class labels to differentiate between different line types where such a classification is useful (*e.g.* lane marking types in Figure 1). Lastly, we can represent overlapping polylines in the image. In fact, our parametrized system is able to even estimate polylines that head in opposite directions while occupying the exact same pixel space. Facilitating polylines for robotic tasks is intuitive as most ground truth data is provided as polylines in modern datasets [7, 31].

*equally contributed

Contributions and outline We propose a bottom-up polyline detection approach that is inspired by single shot object detection but serves the same needs as recurrent approaches. For the first time, this allows to obtain a polyline detector that is not only easy and stable to train. With an inference speed of 187 fps, it also allows to use polyline detections in real-time applications such as automated driving. Thereby, we close a gap that is left in previous works which are reviewed in the next section.

In Appendix A and Section 4, we describe our approach and the corresponding non-maximum suppression, respectively. Using three datasets around autonomous driving, in Section 5 and Section 6, we demonstrate the generality of our approach for implicitly and explicitly visible polylines that range from short dashes to continuous and even crossing or branching polyline structures. Still, for the TuSimple benchmark as an exemplary application, we show that our approach can reach the performance of networks that are tailored just to that very task.

2. Related Work

Related approaches can be separated into three previously unrelated groups. First, we review single shot object detection approaches that our idea is based on. The second group is line segment detection (LSD) which detects line segments assuming they are straight, clearly delimited and independent of each other. Finally, we present recurrent approaches.

An approach that fits neither category, but has conceptual similarity to our 1D border points representation is [26]. Based on points as input, object surfaces are estimated. A transfer to the image domain is generally conceivable, but a suitable encoder is probably as complex as our whole approach.

Single shot detection Object detection can mainly be divided into one- and two-stage approaches. However, alternatives like multi-stage approaches [5] exist.

Two-stage approaches have a region proposal step that predicts sparse regions of interest. They are usually more accurate, but also more complex and less efficient. Representatives of this group are the anchor-based R-CNN family [14, 15, 37] and R-FCN [10], but there are also anchor-free two-stage approaches [48].

Motivated by real-time capable inference times and simplicity, single shot object detection approaches have been proposed. They skip the region proposal step and densely predict objects directly on the image. Most famously, SSD [29] and the recently extended YOLO family [4, 34–36], but also RetinaNet [27] and EfficientDet [41] are based on static anchors. However, as for two-stage approaches, anchor-free, but slower alternatives have been proposed [12, 22, 42].

While both, one- and two-stage approaches are continuously being improved, we wanted to build upon a foundation that is both real-time capable and well-known to be applicable to other tasks than object detection. Thus, we evaluated both YOLO9000 [35] and YOLOv3 [36] with YOLOv3 showing no measurable improvement over YOLO9000 for our task. For highest performance, we suggest to transfer our idea to recent architectures like YOLOv4 or EfficientDet, but claim that YOLO9000 is sufficient to show the overall idea and give an estimate of its performance.

Line segment detection The problem of detecting (parts of) straight lines has been widely covered under the term line segment detection (LSD). For most approaches that do not employ deep learning, we refer to the work by von Gioi [16, 44] that is still often used as baseline. In [44], they also provide a well-written and comprehensive review of multiple classical approaches and key ideas. [2] and [8] not only proposed the probably most significant improvements compared to [44], they also provided two of the main datasets for evaluating LSD quantitatively, both based on the YorkUrban images [11]. However, by concept, all those approaches are limited to lines along edges in the image and cannot detect implicit lines, such as conceptually connected dashes or lane center lines.

Given the aforementioned and the Wireframes [19] dataset, deep learning approaches could tackle the problem of LSD. [19] proposed to detect both junction points and lines connecting them. [45, 46] proposed to use attraction fields for LSD, significantly improving the state of the art. While those three approaches used heuristics to finally extract line segments, [49] proposed the first end-to-end trainable approach for LSD. Recently, [47] reformulated the idea of attraction fields in an end-to-end trainable fashion.

Still, these approaches focus on more or less explicit lines. Also, in contrast to polyline detection, LSD methods generally target wireframes or similar edges that are densely distributed in the image, without any semantic meaning and usually straight. This makes them considerably different from the polylines that we tackle, but a domain transfer could be tried.

Polyline detection This brings us to approaches that aim for the very same generic polylines or polygons that we are trying to detect. They can be found in two fields: instance segmentation and road network/road boundary extraction from bird’s eye view imagery.

For highly accurate, but automated instance segmentation, a series of recurrent neural networks (RNNs) has been proposed [1, 6]. They each expect a bounding box/crop around an object and then predict the polyline vertices node

by node with optional refinement using gated graph neural networks [1]. Not only are these RNNs usually slow and difficult to train, they also need special care to predict the initial vertex. [25, 28] rely on a more generic initialization that fits well for the instance segmentation case, but not at all to our exemplary applications. While being faster than the recurrent approaches, with around 30 ms [28] is still far away from the speed of single shot detectors. Also, extra steps would need to be added to care about branches and merges in the graph.

For the extraction of road network information, multiple approaches predict a graph vertex by vertex. [3] proposed to predict either a direction or a STOP token using a convolutional neural network (CNN). This enables the automatic tracing of the road network given an initial vertex and allows cycles in the network using a distance-based loop closure. In [18], two RNNs are predicting both direction and position of the next vertex that leads along the road boundary. In addition, the RNN estimates the branching degree for each node, allowing to extract directed acyclic road boundary graphs. In contrast, [24] directly predicts the position of the next vertex using a RNN, proposing to convert a graph into multiple bidirectional branchless polygons. Finally, [9] predicts all neighboring nodes with a recurrent decoder and checks for overlap with the existing graph to close loops.

To the best of our knowledge, all polyline detection approaches have in common that they are either recurrent and, thus, hard to train and slow, or very limited in their topology. This motivated our search for a generic, yet fast method to detect polylines of virtually any shape.

3. Generic polyline detection

YOLinO is inspired by the YOLO object detector family [34–36, 38], but refined for the purpose of polyline detection. The overall idea is to split the image into several spatial cells and predict multiple line proposals for each cell. Thus, we enable real-time capability with a shallow and simple network, while being flexible to generalize to any kind of polyline without any restriction on the shape.

Architecture While any backbone with sufficient downsampling capabilities may be used, we chose YOLO9000 [35]. We also ran experiments with YOLOv3 [36], but found it to be outperformed by the smaller version when applied to polyline detection.

DarkNet-19 is a backbone consisting of 19 repeated convolutions and five maxpooling layers to produce a fixed size feature map which represents the cell grid. It is then expanded with an additional convolution to predict the parameters for a given number of line segments within each cell. The full architecture can be found in the supplementary material.

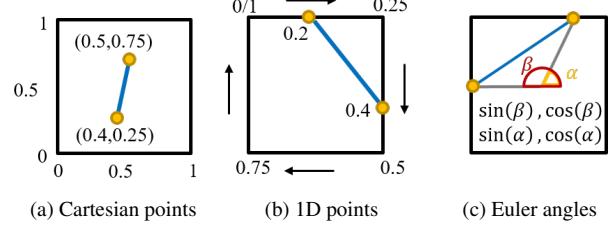


Figure 2. Representations for line segments in a spatial cell. Unbound Cartesian Points (Po) are the most flexible variant, whereas 1D border points (1D) and Euler angles (Eu) converge faster, although the former suffer from a discontinuity in the top left corner.

Predictors Each predictor is constructed as $P = (g, l, c)$ with geometric definition g , a class confidence l and the confidence of the predictor as c . The number of predictors per cell consequentially puts an upper limit on the number of line segments that can be predicted per cell.

Grid resolution In its vanilla version, DarkNet-19 reduces an input image by a factor of 32. An input image of 640×640 for example, translates to a grid size of 20×20 and a cell size of 32×32 pixels. As such, the grid resolution is tightly bound to the network architecture.

As different applications may benefit from finer grid resolutions, we add a custom network layer for rescaling the output of the feature extractor backbone. This new upsampling block consists of a transposed convolution and three regular convolutional layers. When upsampling, we also employ skip connections in order to keep fine-grained features in the process. We evaluate three different upsampling configurations resulting in a grid cell of either 32×32 pixels, 16×16 pixels or 8×8 pixels.

Line representation A suitable set of geometric parameters g must be defined to describe the position, dimension and direction of a line segment. Furthermore, the representation must enable an efficient and meaningful calculation of distances $d(g, \hat{g})$ between line segments for the loss function. We evaluate several line representations and run extensive evaluations with three variants: unbound Cartesian points (Po), 1D border points (1D) and Euler angles (Eu). Note that for all line representations, the start and endpoints are defined by their positions in the network output and, thus, a directional information is encoded implicitly.

The *unbound Cartesian points* (Po) define a line segment intuitively with two Cartesian points that are not bound to any restriction in length or position within a grid cell (see Figure 2a): $g_{\text{Po}} = (g_s, g_e)$ with $g_s, g_e \in [0, 1] \times [0, 1]$. We later encountered this concept to be quite similar to the object detection representation presented in [22].

A positional loss can be calculated as distance between two line segments g and \hat{g} with the sum of Euclidean dis-

tances between both start and endpoint of the two lines.

$$d_{\text{Po}}(g, \hat{g}) = \|g_s - \hat{g}_s\|_2 + \|g_e - \hat{g}_e\|_2 \quad (1)$$

As an alternative, we investigate two other line representations that restrict line segments to have their start and endpoints on the border of a cell. We expect both to converge faster and limit the search space in order to predict connected polylines.

The minimal parametric *1D border points* (1D) regard the cell border as a one-dimensional line, starting in the top left corner and wrapping around the cell clockwise (see Figure 2b). In this setting, a line is described by any tuple of two positions: $g_{\text{1D}} = (g_s, g_e)$ with $g_s, g_e \in [0, 1]$ along the border. We calculate the distance between two line segments g and \hat{g} as follows:

$$d_{\text{1D}}(g, \hat{g}) = \sum_{i \in \{s, e\}} \min(|g_i - \hat{g}_i|, |1 - (|g_i - \hat{g}_i|)|) \quad (2)$$

The *Euler angles* (Eu) solve the discontinuity problem of *1D border points* (1D) at points close to the top left corner [38]. We represent a line segment as cosine and sine components of the start and endpoint angles, α and β : $g_{\text{Eu}} = (\cos(\alpha), \sin(\alpha), \cos(\beta), \sin(\beta))^T$.

Like for the *1D border points* (1D), start and endpoints both lie on the cell border. The angles α and β are then measured between the y -axis of the image and the line between the cell center and the respective border point (see Figure 2c). Due to the given periodicity of the representation, the distance is continuously defined between the individual components. Similarly to [38], we make use of the mean squared error (MSE) between all components.

$$d_{\text{Eu}}(g, \hat{g}) = \frac{1}{4} \left\| \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \\ \cos(\beta) \\ \sin(\beta) \end{pmatrix} - \begin{pmatrix} \cos(\hat{\alpha}) \\ \sin(\hat{\alpha}) \\ \cos(\hat{\beta}) \\ \sin(\hat{\beta}) \end{pmatrix} \right\|_2 \quad (3)$$

Training objective One of the most crucial ideas to conceptualize when dealing with single shot detection systems is the notion of responsibility [34]. When calculating the error term for a given network output, each of the ground truth lines is only matched with a single predictor. To find the predictor, that is considered responsible for a given ground truth line segment, we opt for an iterative greedy approach:

Within each cell, we match the closest pair of ground truth line and predicted line until all ground truth lines are assigned a responsible predictor. Ultimately, not all predictors will be matched to a ground truth line. However, this is intended as each predictor should be responsible only for a specific class/type of line segments that might not be present in the current image. Our matching process is order-invariant, computationally manageable and enables a

responsibility concept for different classes and geometric types of line segments to be learned by different predictors.

We formulate a loss function for this purpose as weighted sum of four sub loss terms. These loss terms are defined in Equations (4) to (7).

$$\mathcal{L}_{\text{loc}} = \sum_{i=0}^S \sum_{j=0}^L \sum_{k=0}^P \mathbb{1}_{ijk}^{\text{resp}} d(g_{ij}, \hat{g}_{ik}) \quad (4)$$

$$\mathcal{L}_{\text{resp}} = \sum_{i=0}^S \sum_{k=0}^P \mathbb{1}_{ik}^{\text{resp}} (c_{ik} - 1)^2 \quad (5)$$

$$\mathcal{L}_{\text{noresp}} = \sum_{i=0}^S \sum_{k=0}^P (1 - \mathbb{1}_{ik}^{\text{resp}}) (c_{ik} - 0)^2 \quad (6)$$

$$\mathcal{L}_{\text{class}} = \sum_{i=0}^S \sum_{j=0}^L \sum_{k=0}^P \mathbb{1}_{ijk}^{\text{resp}} \sum_{c=0}^C (l_{ik}(c) - \hat{l}_j(c))^2 \quad (7)$$

Here, S denotes the number of cells within the output grid, L the number of ground truth lines, P the number of predictors and C the number of classes.

Equation 4 defines the loss \mathcal{L}_{loc} on the geometric distance $d(g, \hat{g})$ between two line segments and encapsulates the localization error of all assignments. $\mathbb{1}_{ijk}^{\text{resp}}$ indicates if in cell i , ground truth line j was assigned to predictor k by the above-mentioned matching process.

Equation 5 and Equation 6 build the confidence error for any predicted line segment. c_{ik} is the confidence score of the k^{th} predictor with the i^{th} cell. $\mathbb{1}_{ik}^{\text{resp}}$ denotes if the k^{th} predictor in cell i , was assigned to any ground truth line.

Equation 7 calculates the classification error $\mathcal{L}_{\text{class}}$ of the classification $l_{ik}(c)$ for the k^{th} ground truth line in cell i for class c .

4. Non-maximum suppression

Since our method outputs a collection of line segments, we propose a simple, yet effective post-processing. Initially, it is important to set a suitable confidence threshold to keep only relevant predictors. While the resulting output may already be useful for some downstream applications, the precision can drastically be improved by adding appropriate non-maximum suppression (NMS). The objective of NMS is to reduce the number of predictions to the correct density and remove duplicates. While this is rather straight-forward for object detections, the detection of polylines bears some ambiguities. Especially for implicit labels like centerlines, many close predictions might be accounted as correct by a human and, thus, manually annotated ground truth labels are noisy by definition. Furthermore, the neighbor cells which construct a polyline influence the correctness of a predictor and, thus, need to be regarded in the

process. Hence, for NMS, we propose to compare the predictions with a confidence score $c_{ik} > \tau_c$ across multiple spatial cells in the image domain.

Independent of the chosen line representation, redundant predictions are clustered using DBSCAN [13] on $(m_x, m_y, l, d_x, d_y)^\top$ where m_x, m_y are the center point image coordinates, l is the length and d_x, d_y are normalized directions of each predicted line segment. We calculate the mean representative of each cluster as result of the NMS. Here, the confidences are used as weights for both clustering and averaging each cluster. This process drastically increases the precision of the estimate while hardly impairing recall and, at 230 fps, not impairing the inference speed. All details on NMS can be found in the supplementary material.

5. Datasets

Our method for image based polyline estimation is designed to be applicable to a wide range of applications. To prove the method’s adaptability, we present results on three different datasets covering two domains and three distinct tasks from photogrammetry and robotics.

TuSimple: Lane boundary estimation The TuSimple lane detection benchmark¹ presents the task of estimating lane boundaries on a highway setting. The popular dataset features more than 3000 front view images for training, 358 for validation and 2782 for testing. The polylines are continuous, each describing the boundary between two lanes, indifferent of the type of lane marking.

Karlsruhe Aerial Images (KAI): Lane marking detection We use aerial images of a highway section near Karlsruhe, Germany, for which the actual lane markings have been annotated manually as polylines. This dataset [31] contains more than 1400 bird’s eye view (BEV) images for training and the task is to identify the explicit white lane marking instead of the implicit lane boundaries of TuSimple. It is important to note here that the dataset provides both continuous as well as dashed markings, requiring the detection to cope with both continuous and bounded labels as well as a classification of these.

Argoverse: Lane centerline estimation As third dataset, we work with Argoverse [7], using its underlying high definition map information to project lane centerlines into more than 2600 front view images. This requires the model to estimate *implicit* polylines that do not have direct visual cues within the image. Given that lane centerlines are often ambiguous and prone to overlaps and occlusions, this problem

¹https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection

setting is considered the most difficult of the three. Unfortunately, there is neither in Argoverse nor anywhere else a benchmark regarding centerline estimations. That’s why we state our results for future comparison.

Data augmentation To enlarge the datasets and reduce the susceptibility for overfitting, we employ several data augmentation techniques during training. For all datasets, random color jitter on brightness ($\pm 50\%$), contrast ($\pm 50\%$), saturation ($\pm 50\%$) and hue ($\pm 20\%$), random erasing (2 %–15 %) and normalization of the colors within the datasets are applied. A small random rotation of up to 36° and random crop up to 18 % of the image size is applied to the TuSimple and Argoverse datasets, while full 360° rotation, but the same cropping are used for the KAI dataset.

6. Experiments

We evaluate three major parameter choices on the TuSimple dataset, but also show example results for the detection on Argoverse and Karlsruhe Aerial Images later on. At first, we examine the influence of the line representation. Secondly, we investigate how different numbers of predictors for each spatial cell affect the performance. And, lastly, we train networks for the different available grid resolutions (32 px, 16 px, 8 px) and compare the results.

For comparison, we trained all TuSimple networks for 80 epochs with a batch size of 32 on a Nvidia GTX 1080Ti GPU. The initial learning rate is set to 10^{-3} . For unbound Cartesian points (Po) a learning rate of 10^{-4} was necessary for convergence. Further details on the experiment can be found in the supplementary material.

Pre-processing Both 1D border points and Euler angles force all line segments to start and end on the border of a single grid cell. Naturally, bounded polylines do not always follow this restriction. Consequently, we need to convert arbitrarily starting and ending polylines into line segments that are restricted to start and end on cell borders (see Figure 3). First, we calculate the intersection points of the polylines with the cell grid (see Figure 3a) and remove points within the cells (see Figure 3b). Next, the start and end points of the polylines are extrapolated (see Figure 3c) if they cover more than half of the grid cell. Otherwise they are dropped (see Figure 3d).

This introduces slight discretization errors which are most noticeable in sharp turns and on the ends of a polyline. Our evaluation accounts for these errors as we compare the predictions to the raw ground truth lines. However, for clarity, we provide the average ground truth deviation for the different datasets and grid resolutions in the supplementary material.

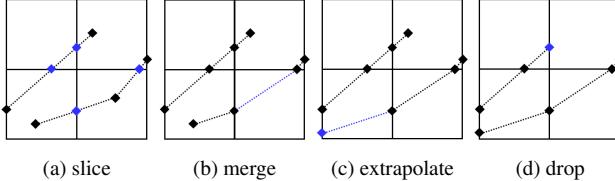


Figure 3. Converting original polylines to discretized cells on a 2×2 grid. Black lines visualize the ground truth lines, that are modified from steps a to d. The modifications are highlighted in blue.

Line segment evaluation metrics To enhance comparability between the different problem settings, we also develop an evaluation metric that allows us to quantify the correctness of a collection of line segments with respect to a set of ground truth polylines. We provide recall, precision and F1 score for all experiments in order to provide an insight into the quality of the actual predictions. Other works resort to a pixel-based IoU metric, which is often more difficult to interpret and neglects the direction of line segments [30].

For evaluation, we run our NMS and sample evaluation points $\hat{p} = (x, y)$ along the ground truth and the predicted line segments (with a sample distance of 1 px). For all points the orientation α of the line segment is calculated in order to compare also the orientation.

Having this representation, we treat predicted points as true positive whenever they are the closest prediction to a ground truth point and lie within a certain radius τ of all three dimensions (x, y, α) .

TuSimple benchmark In order to compare our results to related work, we also present the TuSimple benchmark metrics with average accuracy, false negative and false positive rates. The TuSimple benchmark provides evaluation scripts and labeled test data for the TuSimple dataset. Here, lane boundary instances are expected and thus further post-processing is required. First, we propose to find connected components in the generic NMS output. At each level we calculate the average representative and then fit a cubic spline into each line instance.

We assume two NMS predictions a and b to be connected if the start point of a is within a certain distance from b 's endpoint (here: 75 % of the cell size). This is done using a breadth-first search with a loop detection, resulting in a distinct tree-like structure for each component. Next, for each tree, predictions are averaged across the same depth level. Finally, to increase fidelity, each connected component is interpolated with a cubic spline. Further details about the TuSimple post-processing can be found in the supplementary material.

Line representations One of the major modeling decisions of our approach lies in the choice of a suitable line

Line	Acc	FP	FN	F1	Rec.	Prec.
1D	.887	.157	.160	.616	.955	.455
Eu	.877	.137	.168	.685	.956	.533
Po	.914	.159	.112	.740	.950	.607

Table 1. Results for different line representations on the TuSimple validation set. We evaluate 1D border points, Euler angles and unbound Cartesian points. The architecture has eight predictors per 32 px cell.

Line Pred	Acc	FP	FN	F1	Rec.	Prec.
Eu 4	.878	.168	.180	.546	.971	.380
Eu 8	.877	.137	.168	.685	.956	.533
Eu 12	.885	.154	.159	.717	.940	.580
Po 4	.922	.157	.099	.713	.952	.571
Po 8	.914	.159	.112	.740	.948	.607
Po 12	.903	.135	.120	.777	.941	.662

Table 2. Results on different number of predictors for both unbound Cartesian points (Po) and Euler angles (Eu) on the TuSimple validation set without any upsampling (32 px).

representation. It is worth mentioning that the 1D border points (1D) converge rather slowly compared to Euler angles (Eu) due to the fact that the network has to overcome the discontinuity between points before and after the top left corner of the grid cell, where the mathematical distance is huge, but the actual difference is tiny. Further, in order to achieve convergence with unbound Cartesian points (Po), it appears to be useful to initialize a smaller learning rate compared to the other two experiments.

We present three experiments comparing the representations in Table 1. For each, we used eight predictors per cell, a cell size of 32 px (i.e. no upsampling). Unbound Cartesian Points (Po) have the highest false positive rate for TuSimple, but are the most convincing regarding both generic F1 score and TuSimple accuracy. We assume this to be due to the flexibility that comes with this representation. While we liked the idea of the minimal representation of 1D border points, it proved to be unstable during training. Hence, we will perform further experiments only on unbound Cartesian points and Euler angles.

Number of predictors Another important aspect in designing single shot detection models is the allowed number of predictions per cell. Therefore, we trained networks with 4, 8 and 12 predictors per cell at a default resolution of 32×32 pixels. The results are presented in Table 2. They suggest that fewer predictors work better for TuSimple accuracy while more predictors work better for the more generic F1 score. Hence, we recommend 12 predictors for more complex applications while using four or eight predictors for the TuSimple lane boundary estimation task.

Line Grid	Acc	FP	FN	F1	Rec.	Prec.
Eu 32 px	.877	.137	.168	.685	.956	.533
Eu 16 px	.899	.343	.201	.658	.959	.500
Eu 8 px	.839	.481	.331	.712	.919	.581
Po 32 px	.914	.159	.112	.740	.948	.607
Po 16 px	.930	.258	.095	.739	.951	.604
Po 8 px	.875	.405	.196	.776	.930	.665

Table 3. Results for different cell sizes on the TuSimple validation set. We compare Euler angles and unbound Cartesian points with eight predictors. 16 px is best in TuSimple accuracy, 8 px has the highest generic F1 score.

Grid resolution The third important choice is whether to use the default resolution of 32×32 pixels or add one (16×16 px) or two upsampling modules (8×8 px). We evaluate all grid resolutions on both Euler angles (Eu) and unbound Cartesian points (Po) with eight predictors per cell (see Table 3). Using one upsampling module (16 px) seems to be best for TuSimple accuracy while two upsampling modules (8 px) is the winner regarding generic F1 score.

Comparison with the state of the art As there are no benchmarks for semantic polyline detection, we settled for the TuSimple lane boundary estimation benchmark as simple example application. Thus, our generic framework competes with approaches that are tailored, but also limited to the very detection of a few mostly vertical line instances.

For evaluation on the test set, the previous experiments suggested two candidate configurations: unbound Cartesian points at a resolution of 16 px with either four or eight predictors. Since, on the validation set, four predictors only scored 88% accuracy, we discarded this option.

We did not expect to win, but rather wanted to evaluate performance w.r.t. far more specialized networks. With 94.2% accuracy on the test set (see Table 4), our most promising candidate gets within three percentage points of the best specialized competitor, which is a lot better than we expected. In Figure 13, we show sample results.

Hence, while not setting a new state of the art for highway lane boundary detections, YOLinO is so far the only choice for many, especially more complex applications which do not have a multitude of existing, specialized solutions. Still, we get close to the best possible performance without requiring any network specialization.

Other applications To show the wide range of applications for our approach, we present further results on Karlsruhe Aerial Images and Argoverse.

The Argoverse dataset is by far the most challenging task in the evaluation. Lane centerlines are already difficult for

Method	Acc	FP	FN	fps
LineCNN [23]	.969	.044	.020	17
PINet [21]	.958	.059	.033	40
LaneATT [39]	.967	.036	.018	250
ResNet-18 [33]	.961	.019 ²	.040 ²	312
YOLinO (ours)	.942	.188	.076	187

Table 4. Our best-performing configuration (Po, eight predictors, 16 px) compared to selected related work that are specifically tailored towards lane boundary detection. While YOLinO is not tailored to this task, we are able to reach comparable accuracy on the TuSimple test set. Note that regarding inference speed, YOLinO is among the fastest approaches.

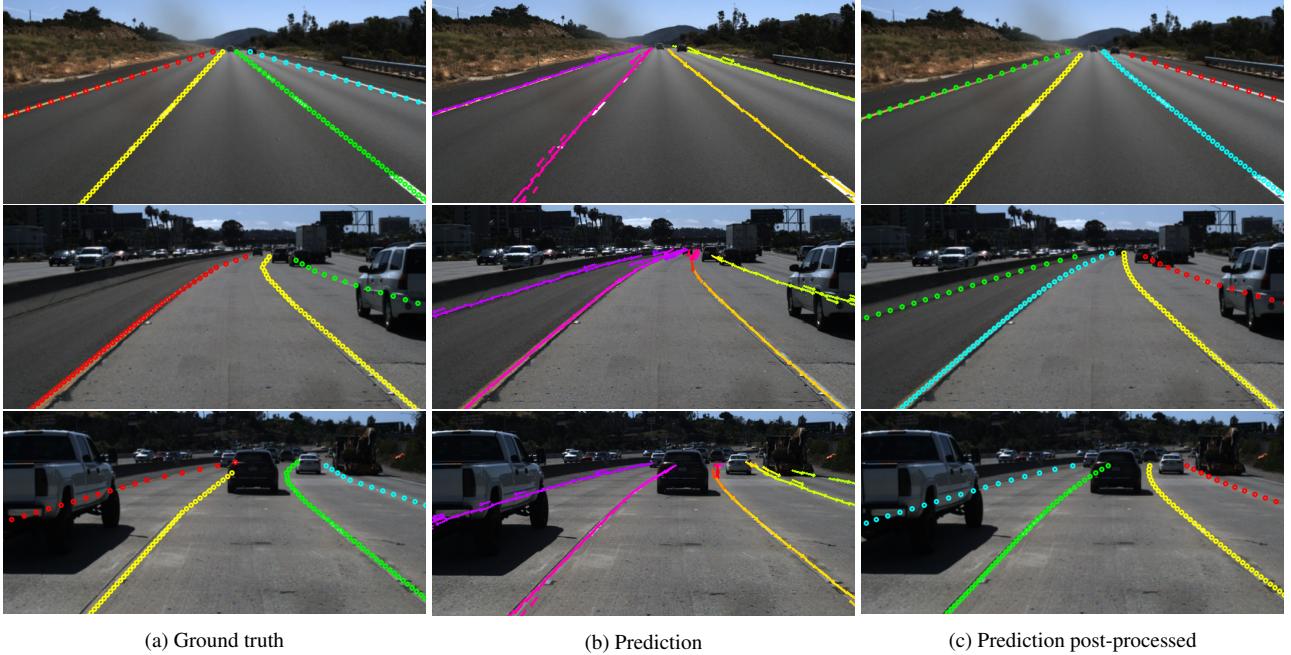
humans to determine and, thus, labels are also prone to variations. The detected centerlines have a precision of 41 %, recall of 69 % leading to an F1 score of 52 %. Examples can be found in Figure 17. Regarding the complexity of lane structures, this is already a great result.

Karlsruhe Aerial Images contain mostly homogeneous images, but only few examples of corner cases like on-ramps or bridges. Still, we are able to cope with these challenges and predict markings with an F1 score of 89 %, a recall of 90 % and a precision of 88 %. Examples can be found in Figure 6.

7. Conclusion

We presented *YOLinO*, a single shot polyline detector enabling to build real-time applications using polylines. While being a generic approach, YOLinO is comparable to state-of-the-art TuSimple lane detection algorithms. Besides being extremely fast, our approach can detect a wide range of polylines, even if they are not explicitly visible, branching or even crossing. Moreover, both dashed and continuous lines can be detected with the same head. This enables robotic applications such as road marking detection or lane centerline estimation, but is also conceivable for many others, e.g. blood vessel detection. While we demonstrated the general idea using YOLO9000, using a modern backend such as EfficientNet [40] and incorporating ideas from YOLOv4 [4] is expected to improve performance even more. This also holds for adding a learned NMS module, e.g. exploiting the grid structure with a Graph Neural Network.

²reported by [39]



(a) Ground truth

(b) Prediction

(c) Prediction post-processed

Figure 4. Results for the TuSimple benchmark using unbound Cartesian points, eight predictors and a grid resolution of 16 px. Colors in (a) and (c) indicate instances, but do not need to be similar for prediction and ground truth. In (b) the colors visualize the orientation of the predicted line segments.

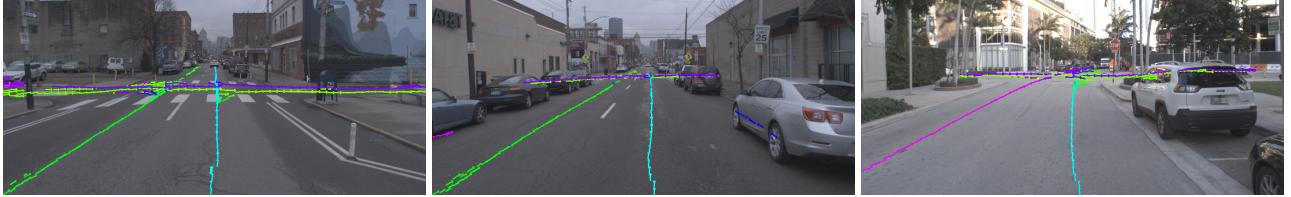


Figure 5. Results for the Argoverse lane centerline task. Colors encode the driving direction of a lane. In the first two images, YOLinO is able to correctly detect a one-way street with both lanes driving in the same direction. In all three images, detecting intersections does not pose a problem as well.



Figure 6. Results for the KAI dataset. The network has eight predictors, 16 px cells and unbound Cartesian points. Colors indicate the different classes: dashed and solid road marking. Aerial images: © City of Karlsruhe | Liegenschaftsamt

Appendix

In this supplementary material, we provide further details on YOLinO. In Appendix A, the base architecture is explained in detail. Here, we also provide training parameters. An estimation of the discretization errors can be found in Appendix B. For further detail on the NMS or the TuSimple-specific post-processing, please refer to Appendix C. Following, we describe the calculation of our evaluation metrics in Appendix D, followed by an overview of all experiments of the ablation study in Appendix E. Qualitative results from all three datasets can be found in Appendix F, Appendix G and Appendix H.

A. Architecture

For the backbone we chose YOLO9000 with Darknet-19 [35]. Depending on the required resolution of the output grid, we employ zero, one or two upsampling blocks. In order to pass local features, we further implement a skip connection between the downsampling and upsampling layers. Figure 7 shows the full architecture with one upsampling block, resulting in cells of 16×16 px.

The training is implemented in Pytorch [20] and parameterized according to the configuration in Table 5.

B. Discretization

Especially for 1D border points and Euler angles binding start and endpoint to the cell borders introduces slight errors. These deviations are most noticeable in sharp turns and on the ends of a polyline. Our recall already accounts for these errors as we compare the predictions to the full ground truth lines. However, for clarity we provide the average ground truth deviation for the different datasets and grid resolutions in Table 6. As expected, the introduced error is smaller the higher the grid resolution. We also see higher values for the KAI dataset as it contains many short polylines that do not initially align with the grid. Similarly, the Argoverse dataset features many splitting polylines and contains shorter polylines in the more distant road areas. Contrastingly, the polylines in TuSimple are mostly continuous which makes the overall deviation less distinctive. We

	Argoverse	KAI	TuSimple
Batch Size	16	16	32
Input Image Size	448x960	640x640	320x640
Decay Rate	10^{-4}	10^{-4}	10^{-4}
Optimizer	Adam	Adam	Adam
Epochs	55	30	40

Table 5. List of training parameters of each experiment shown in qualitative results. Epochs until convergence are for unbound Cartesian points with eight predictors and 16 px resolution.

Size (px)	TuSimple	KAI	Argoverse
32×32	1.40	1.85	2.57
16×16	0.42	0.55	0.79
8×8	0.14	0.16	0.25
Input Size (px)	320×640	640×640	448×960

Table 6. Ground truth deviation induced by pre-processing with respect to different grid resolutions, given as average pixel difference for all line segments.

conclude that the most suitable grid resolution depends on the specific problem setting as finer grids also introduce significantly higher computational costs.

C. Post-processing

As our representation in terms of raw predictors is generic, but the approach is applicable to many concrete applications, we distinguish two kinds of post-processing. First, we describe a generic non-maximum suppression (NMS) that is beneficial for almost any application since it suppresses redundant predictors. Second, we show how our generic output can be post-processed such that the output is comparable with that of neural networks specialized to these very applications. For this, we chose the TuSimple lane boundary estimation benchmark.

Generic NMS For the generic NMS, we expect predictors $P = (g, l, c)$. First, we discard all predictors with $c \leq \tau_c$ (cf. Table 7). Next, we convert the various geometric representations of a predictor to a common one that is more suitable for DBSCAN, hereafter called *NMS coordinates* \tilde{g} . Here, $\tilde{g} = (m_x, m_y, \ell, d_x, d_y)^\top$ consists of the midpoint of the line segments in image coordinates m_x, m_y , allowing to cluster even across neighboring cells. ℓ is the length of the predictors in image coordinates. Finally, d_x, d_y are the *normalized* directions for each predictor. For instance, for the unbound Cartesian points (Po) representation, we have $g_{\text{Po}} = (g_s, g_e)$ with $g_s, g_e \in [0, 1] \times [0, 1]$. We first convert g_s, g_e to image coordinates, yielding $\hat{g}_s, \hat{g}_e \in [0, M] \times [0, N]$. These can then be used to calculate the common NMS coordinates:

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \lambda_m \kappa \frac{\hat{g}_e + \hat{g}_s}{2} \quad (8)$$

$$\Delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \hat{g}_e - \hat{g}_s \quad (9)$$

$$\ell = \lambda_\ell \|\Delta\| \quad (10)$$

$$d_x = \lambda_d \frac{\delta_x}{\ell} \quad (11)$$

$$d_y = \lambda_d \frac{\delta_y}{\ell} \quad (12)$$

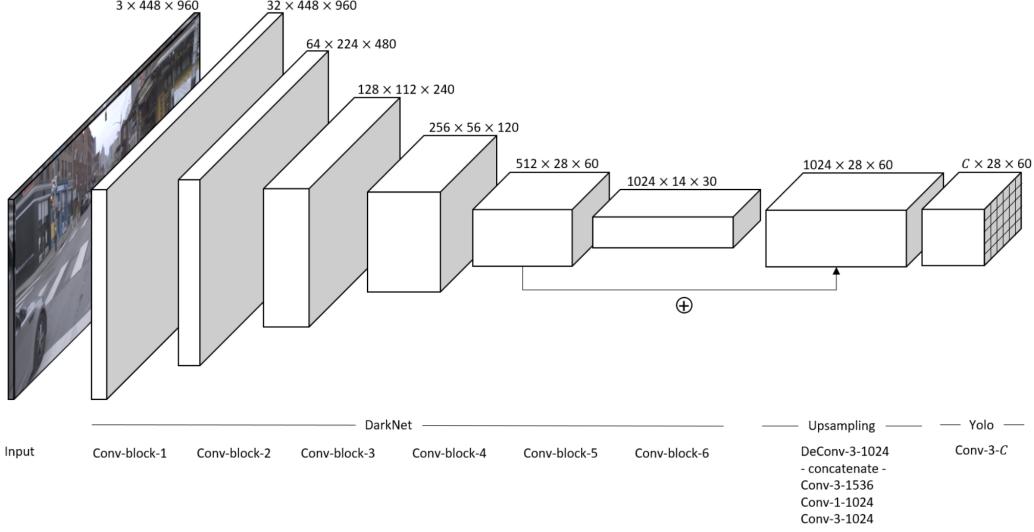


Figure 7. YOLinO’s feed-forward architecture. Here shown for the Argoverse input images and with a single upsampling stage, leading to grid cells of 16×16 px. Each grid cell predicts up to 12 line segments with a geometric definition g , a confidence value c and an optional classification l , all together resulting in the output channels C .

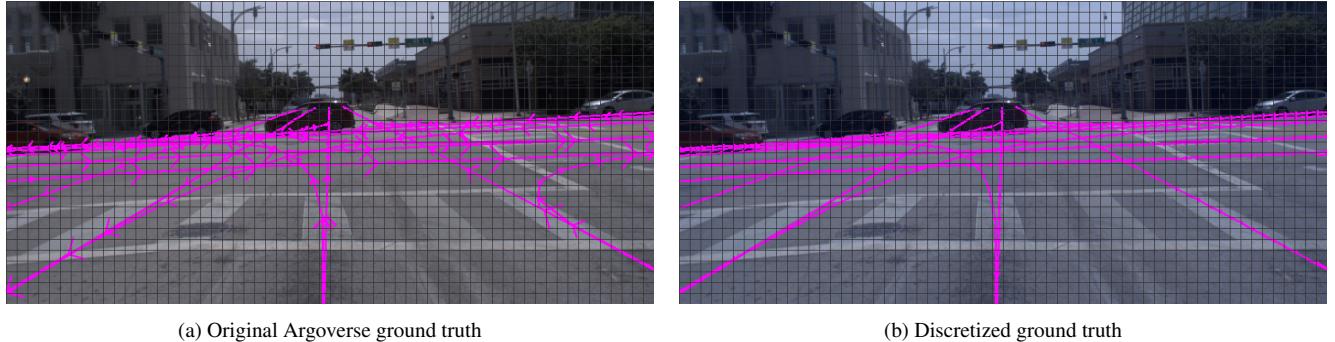


Figure 8. Comparison of the original Argoverse ground truth center lines with the discretized grid lines used for training.

The factors λ scale the coordinates appropriately and are determined empirically for each dataset (cf. Table 7). Factor κ denotes the grid scale, i.e. $\kappa = 1.0$ for our raw output with 32×32 pixels cell size, $\kappa = 0.5$ for 16×16 pixels etc.

Next, we apply the DBSCAN clustering by Scikit-learn [32] using exponentially scaled confidences as weights $w_i = c_i^{10}$ and DBSCAN parameters $\epsilon = 0.02$ and at least two predictors per cluster (also called `min_pts` or `min_samples`). Finally, we take the weighted average of each cluster in NMS coordinates using the same exponential weights w_i we used for clustering. For visualization and downstream applications, we found it more intuitive to take the maximum confidence of each cluster instead of the weighted average.

Both, coordinate conversion and averaging are done in NumPy [17]. The overall speed of 230 fps holds for all three steps, conversion, DBSCAN and averaging taken to-

gether. Pipelining the three steps could even roughly double the throughput with DBSCAN being the bottleneck.

TuSimple Post-processing For TuSimple, in addition to the generic NMS, we need to convert predicted line segments into contiguous and smooth, but accurate lane boundary estimates.

First, to prevent cycles, we discard all predictors which point downwards by more than 0.25, i.e. a quarter of a grid cell. For all remaining predictors, we determine a successor by minimal start/endpoint distance such that the successor start point is closer than 0.75 grid cells and not in the bottom half of the bottom row of the grid. Given this adjacency and possible roots that have no successor (i.e. usually the top-most predictors in each “tree”), but are possibly successor to multiple predictors, we start a breadth-first search starting from those root nodes (cf. Figure 9). At each level, we

		Argoverse	KAI	TuSimple
Confidence threshold	τ_c	0.99	0.95	0.9
Weight for segment length	λ_ℓ	0.016	0.013	0.013
Weight for midpoint position	λ_m	1.5	1.5	2
Weight for directions	λ_d	0.05	0.05	0.05

Table 7. List of NMS parameters for each dataset.

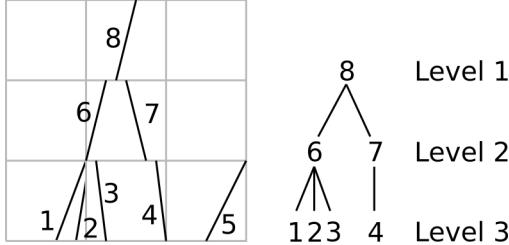


Figure 9. Illustration of the breadth-first search (BFS) and averaging used for polyline extraction, here using $\tau_s = 3$. Segments 1, 2, and 3 share the same successor, as do segments 6 and 7. 8 is a root node and will initiate one BFS procedure leading to the tree on the right with three levels. Each level is averaged, meaning that segments 6 and 7 as well as segments 1-4 are averaged. Segment 5 has no successor in reach and is discarded as the resulting polyline is too short.

calculate the weighted average of all predictors at the same level using confidences as weights. This leads to a polyline that is close to the desired output, but not yet smooth. First, we discard polylines with less than $\tau_s = 10$ segments. Second, we fit a B-spline of degree 3 through the mid points of each predicted line segment using SciPy’s [43] `splprep` function with smoothing parameter $s = 0.05$. These splines are then evaluated, leading to the points that we use for evaluation.

Intermediate results of each step are depicted in Figure 10.

D. Line segment evaluation metrics

For all experiments, we calculate the F1 score, recall and precision, in order to provide a general insight into the prediction independent of the application. We formulate the three scores as follows. With the number of true positive predictions $|TP|$ and the number of ground truth elements $|GT|$, the recall is defined as

$$R = \frac{|TP|}{|GT|}. \quad (13)$$

Comparing the true positives with the number of predictions $|PD|$, we get a precision with

$$P = \frac{|TP|}{|PD|}. \quad (14)$$

Combining both scores leads to

$$F1 = 2 * \frac{P \cdot R}{P + R}. \quad (15)$$

As in our representation, line segments can differ in length and orientation, determining the true positives should regard these parameters. We thus sample all line segments both from the predictions and from the ground truth with a sample distance of 1 px. In addition, we calculate the orientation α of each line segment, leading to a representation of each sample point as $s = (x, y, \alpha)$. Then, we account all predictions as true positive that lie within a certain radius θ from a given ground truth point. Figure 11 visualizes this for the 2D case without regarding the orientation α . An example from the TuSimple dataset is depicted in Figure 12.

E. Relevant experiments

For better comparison between the several ablation studies, we provide all results in Table 8. The first group depicts experiments for different line representations, the second group provides insight into the grid resolution and in the third group, you can compare different numbers of predictors.

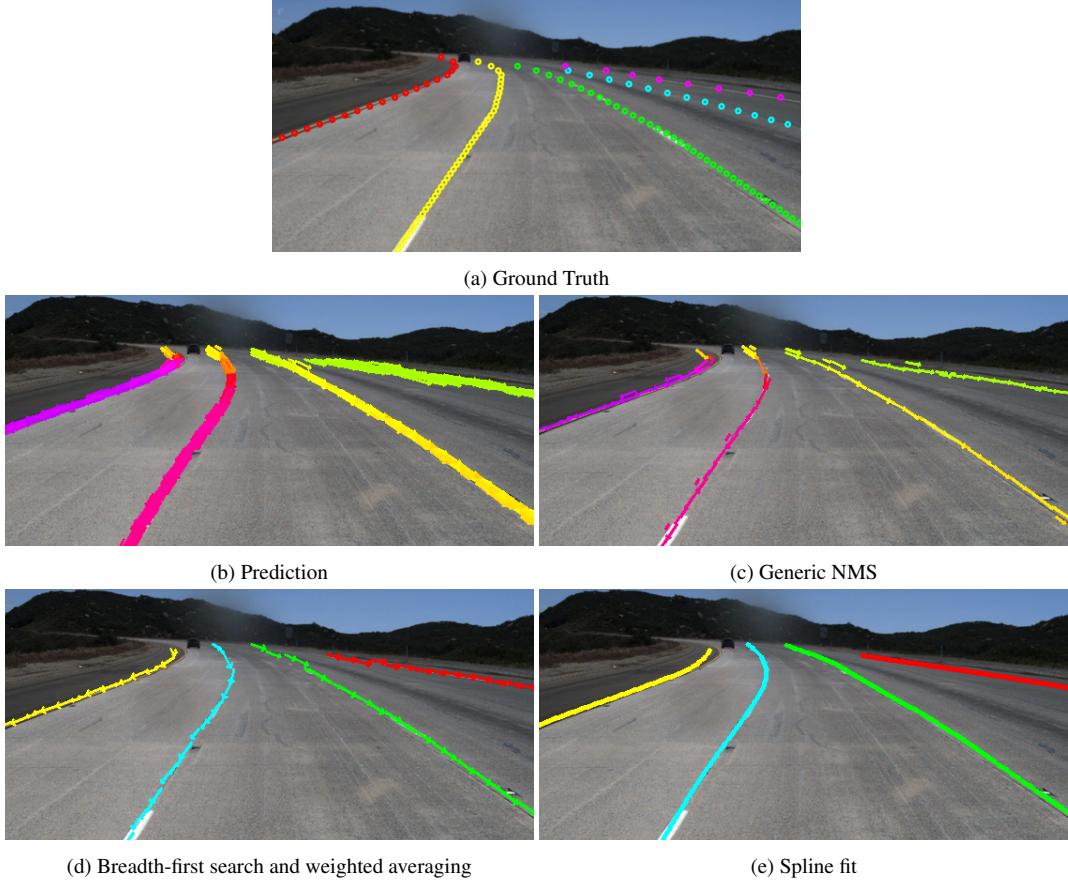


Figure 10. Results of each step in the post-processing for TuSimple predictions. In (a), (d) and (e) the colors indicate instances. In (b) and (c) the colors describe the orientation of the line segments.

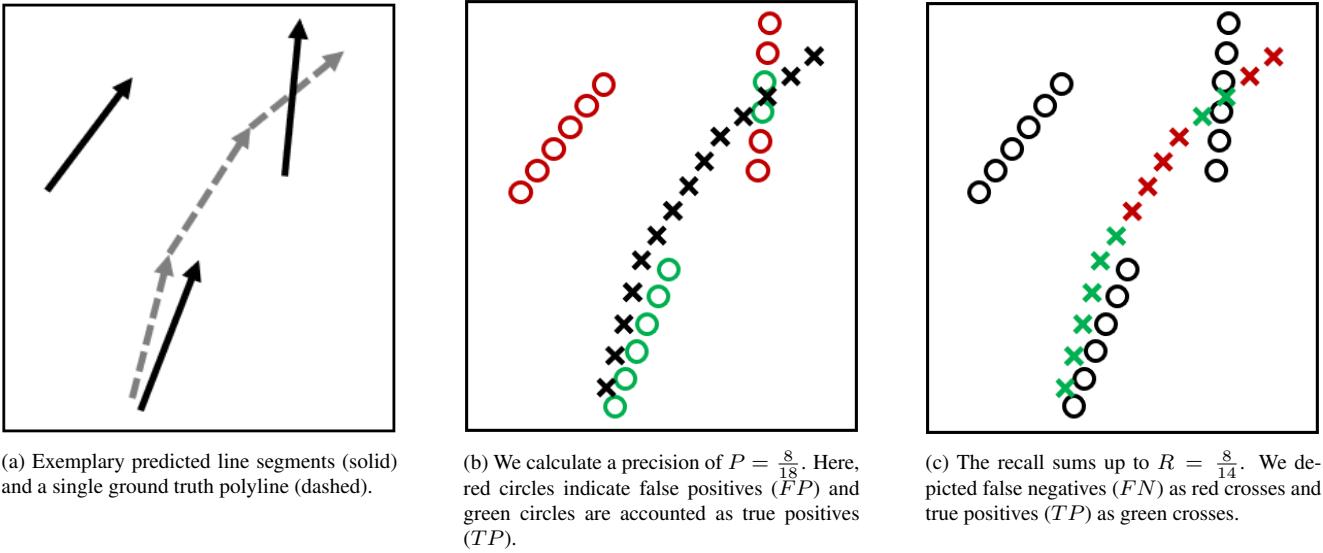
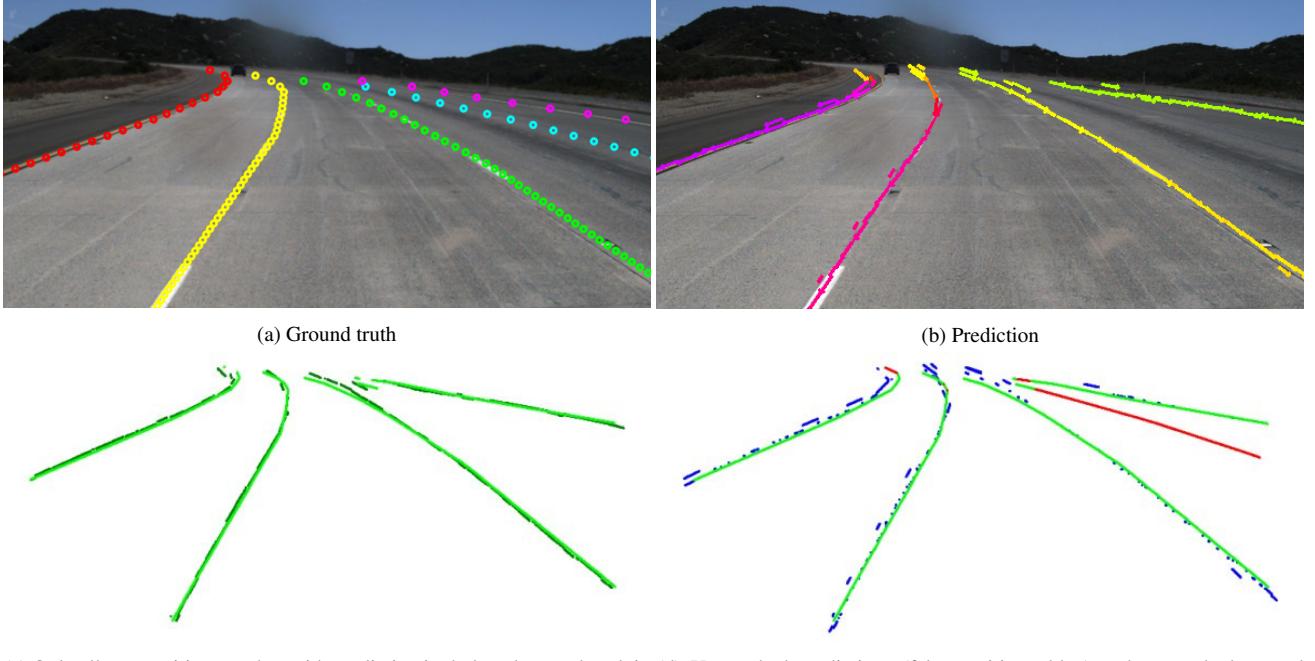


Figure 11. Illustration of our line segment evaluation metric. Note that in the matching, not only the position of each sample, but also the angle at that position is taken into account.



(c) Only all true positive matches with prediction in dark and ground truth in light green.
(d) Unmatched predictions (false positives, blue) and unmatched ground truth (false negatives, red).

Figure 12. Visualization of the true positives, false positives and false negatives for a real prediction. Here, recall $R = 0.81$ and precision $P = 0.86$.

Line	Pred	Grid	Acc	FP	FN	F1	Recall	Prec.
1D	8	32 px	.887	<u>.157</u>	.16	.616	<u>.955</u>	.455
Eu	8	32 px	.877	.137	.168	.685	<u>.956</u>	.533
Po	8	32 px	.914	<u>.159</u>	.112	.740	<u>.950</u>	.607
Eu	8	32 px	.877	.137	.168	.685	<u>.956</u>	.533
Eu	8	16 px	.899	.343	.201	.658	<u>.959</u>	.500
Eu	8	8 px	.839	.481	.331	.712	.919	.581
Po	8	32 px	.914	<u>.159</u>	.112	.74	.948	.607
Po	8	16 px	.930	.258	.095	<u>.739</u>	<u>.951</u>	.604
Po	8	8 px	.875	.405	.196	.776	.930	.665
Eu	4	32 px	.878	.168	.180	.546	.971	.380
Eu	8	32 px	.877	.137	.168	.685	<u>.956</u>	.533
Eu	12	32 px	.885	<u>.154</u>	.159	.717	.940	.580
Po	4	32 px	.922	<u>.157</u>	.099	.713	<u>.952</u>	.571
Po	8	32 px	.914	<u>.159</u>	.112	.74	.948	.607
Po	12	32 px	.903	.135	.120	.777	.941	.662

Table 8. Results for the ablation studies on important hyperparameters on the Tusimple validation set. Best result in each group is bold, second best result is underlined.

F. Qualitative results for TuSimple

We depicted further examples for the TuSimple dataset in Figure 13 and Figure 14. These examples clearly show some peculiarities of the TuSimple dataset and training. First, the prediction often shows two highly confident hypotheses on the lane boundary, which is probably caused by imprecise labels and ambiguities within. As can be seen in the visualization of the main submission, we are easily discarding these false hypotheses in the post-processing.

Similarly, predicting false positive and false negative lane boundaries happens (FP rate of .188 and FN rate of .076) as there also exist quite some ambiguities in the labeling. E.g. considering the first row in Figure 13, the blue lane boundary (in the ground truth) does not have any visual cue and is very doubtful even for humans. Thus, our approach does not recognize this as a valid boundary. Comparing the second and third row of Figure 13, it becomes apparent why a learned architecture might show frequent false positives or false negatives. In the second row, five lane boundaries were labelled whereas the third row only expects three lane boundaries although there exists at least four (+1 next to the right green lane boundary). The same applies to the fourth row of Figure 14, where three lanes to the left are labelled, but the one to the right is not. This leads to false positive predictions by our approach that predicts valid lane boundary hypotheses that are not labelled within TuSimple dataset.

Further, we found scenes, where the ground truth is wrong, but our prediction overcomes those errors e.g. as shown in the third row of Figure 14.

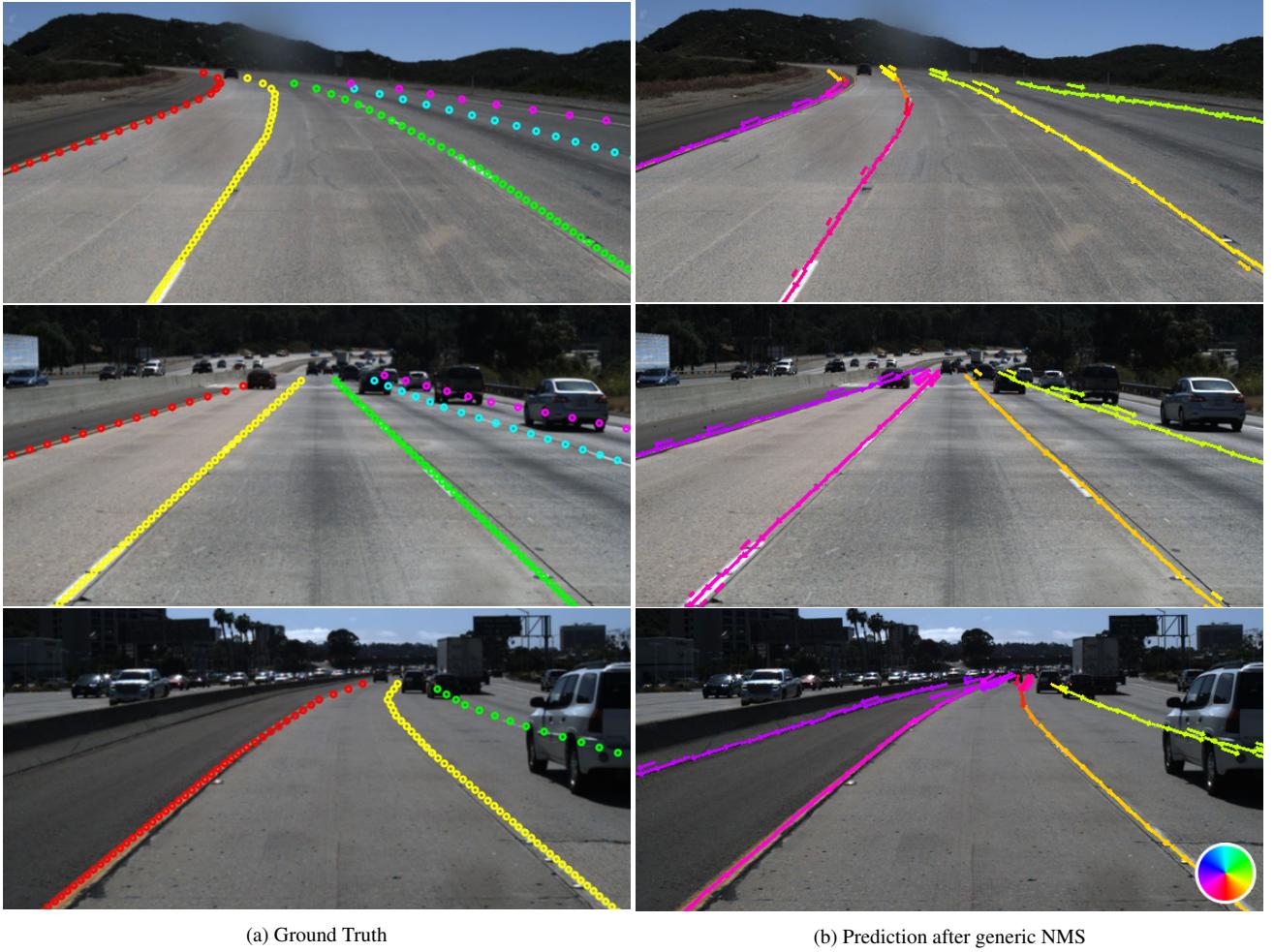


Figure 13. Example results of the TuSimple test set (unbound Cartesian points (Po), eight predictors and a grid resolution of 16×16 px). Colors in (a) indicate instances. In (b) the colors visualize the orientation of the predicted line segments.

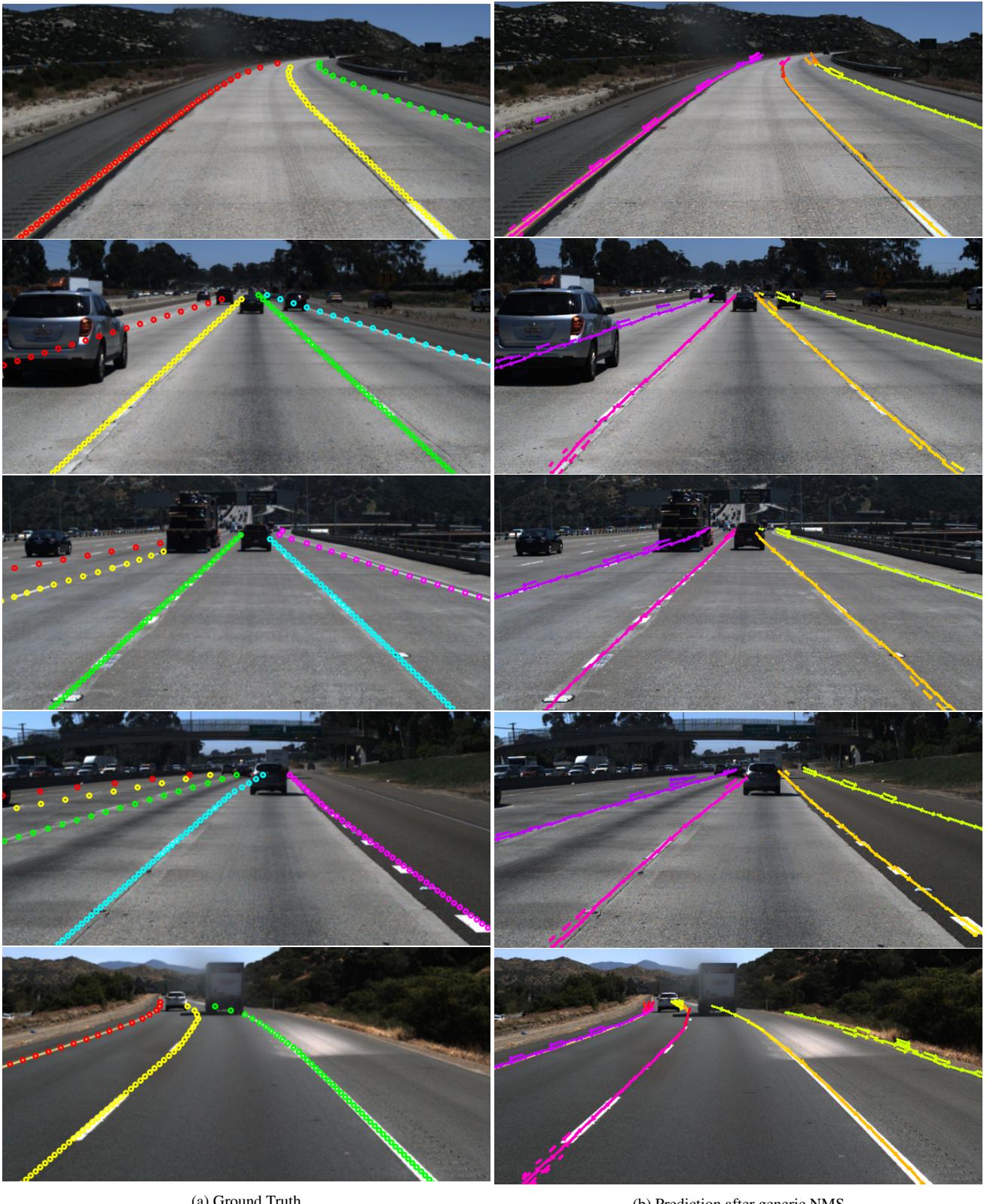


Figure 14. Example results of the TuSimple test set (unbound Cartesian points (Po), eight predictors and a grid resolution of 16×16 px). Colors in (a) indicate instances. In (b) the colors visualize the orientation of the predicted line segments.

G. Qualitative results for Karlsruhe Aerial Images

We depicted further examples for the KAI dataset in Figure 15 and Figure 16. As can be seen, we are able to classify the road markings with 96 % accuracy. However, the geometry of the prediction provides more valuable insights.

As can be seen in Figure 15, we are able to predict unusual side lanes that are not in the ground truth. However, some of the dashed markings are predicted only partly. In the first row of Figure 16, we can see that the dataset bears some unexpected pattern in the labels. For non-drivable areas only the right lane marking is labeled. This is learned by our network, but is not the best in general. In the same image, we can see that unusual road marking combinations (dashed lane marking adjacent to solid lane marking) are not recognized. This can be explained by the small number of occurrences in the training set.

In the first row of Figure 15 and in the first and third row of Figure 16, we can further see that unusual and especially white vehicles confuse the estimation slightly. This might also be due to their small representation in the dataset.

On the contrary, shadows and artifacts on the surface seem to have no effect on the prediction (cf. second and fourth rows of Figure 16)



Figure 15. Example results of the KAI dataset (unbound Cartesian points (Po), eight predictors and a grid resolution of 16×16 px).
Aerial images: © City of Karlsruhe | Liegenschaftsamt

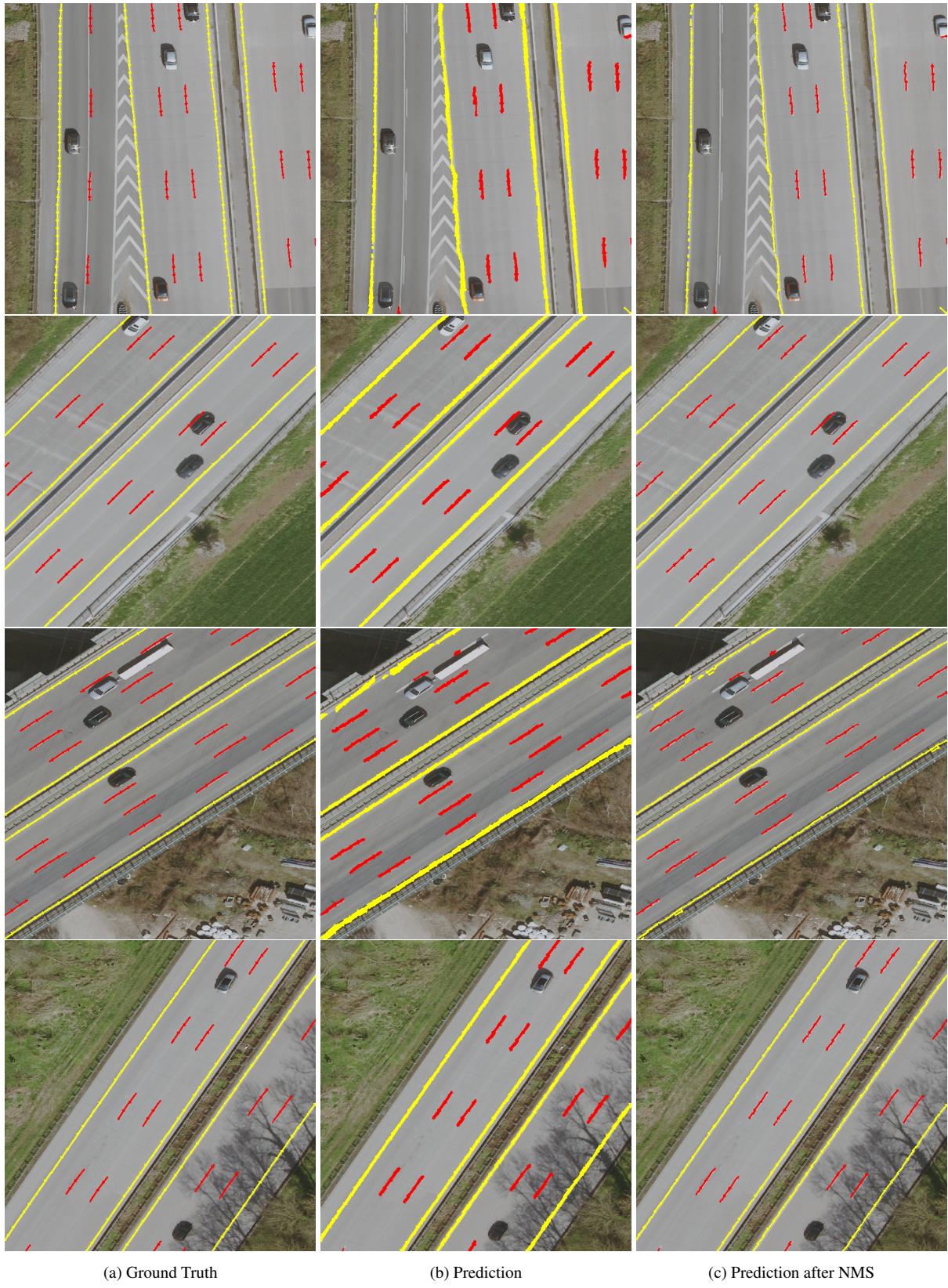


Figure 16. Example results of the KAI dataset (unbound Cartesian points (Po), eight predictors and a grid resolution of 16×16 px).
Aerial images: © City of Karlsruhe | Liegenschaftsamt

H. Qualitative results for Argoverse

The Argoverse datasets provides the most interesting insights into the capabilities of YOLinO. We primarily selected intersection scenes in order to show that YOLinO is able to infer the complex structure. The representation is ideally designed to describe multiple directions and overlapping lanes.

Figure 17 shows several scenes where mainly straight lane centerlines are predicted. Here, no traffic participant is occupying the view, thus the prediction is straight forward. On the contrary, in Figure 18 parts of the road are hidden behind other vehicles or pedestrians are crossing the street. This seems not to be of any problem for the approach. For both, we want to highlight that centerlines on empty streets might already be a tough detection target as they are not determined by direct visual cues.

Comparing the second row of Figure 17 with the second and fourth rows of Figure 18, it becomes clear that the network is also able to correctly predict the driving direction of lanes, as Figure 18 also shows scenes with a single driving direction.

In some scenes (2nd row in Figure 18, 2nd row in Figure 17), the network assumes a three-armed intersection to be a four-way intersection. This only occurs in far distance and might be a result of an imbalance in the dataset. Surprisingly, the scene in the last row of Figure 18 seems to pose more problems than others as the left lane is not fully detected, and more false positives than usual are predicted. This might be due to difficult lighting conditions.



Figure 17. Example results of the Argoverse dataset (unbound Cartesian points (Po), eight predictors and a grid resolution of 16×16 px). Colors indicate the orientation of the predicted line segments.

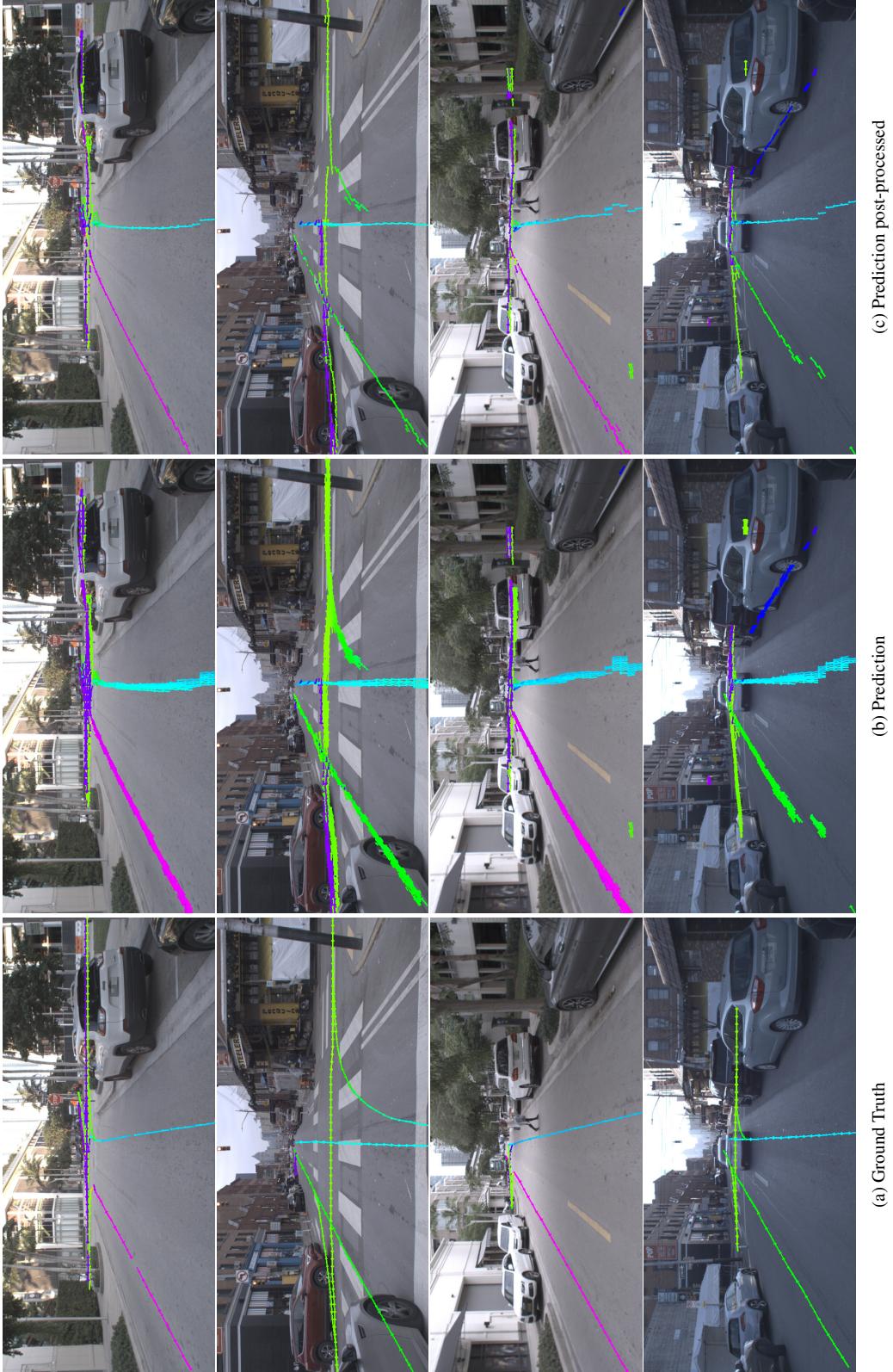


Figure 18. Example results of the Argoverse dataset (unbound Cartesian points (P_o), eight predictors and a grid resolution of 16×16 px). Colors indicate the orientation of the predicted line segments.

References

- [1] David Acuna, Huan Ling, Amlan Kar, et al. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [2] Emilio J. Almazan, Ron Tal, Yiming Qian, et al. Mcmlsd: A dynamic programming approach to line segment detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [3] F. Bastani, S. He, S. Abbar, et al. RoadTracer: Automatic Extraction of Road Networks from Aerial Images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4720–4728, 2018.
- [4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934*, 2020.
- [5] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving Into High Quality Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6154–6162, 2018.
- [6] Lluís Castrejón, Kaustav Kundu, Raquel Urtasun, et al. Annotating object instances with a polygon-rnn. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [7] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8748–8757, 2019.
- [8] N. Cho, A. Yuille, and S. Lee. A Novel Linelet-Based Representation for Line Segment Detection. *Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1195–1208, 2018.
- [9] Hang Chu, Daiqing Li, David Acuna, et al. Neural turtle graphics for modeling city road layouts. In *International Conference on Computer Vision (ICCV)*, October 2019.
- [10] Jifeng Dai, Yi Li, Kaiming He, et al. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *Advances in neural information processing systems*, pages 379–387, 2016.
- [11] Patrick Denis, James H. Elder, and Francisco J. Estrada. Efficient Edge-Based Methods for Estimating Manhattan Frames in Urban Imagery. In *European Conference on Computer Vision (ECCV)*, pages 197–210, 2008.
- [12] K. Duan, S. Bai, L. Xie, et al. CenterNet: Keypoint Triplets for Object Detection. In *International Conference on Computer Vision (ICCV)*, pages 6568–6577, 2019.
- [13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD’96, page 226–231. AAAI Press, 1996.
- [14] R. Girshick. Fast R-CNN. In *International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.
- [15] Ross Girshick, Jeff Donahue, Trevor Darrell, et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.
- [16] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: A Line Segment Detector. *Image Processing On Line*, 2:35–55, 2012.
- [17] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.
- [18] Namdar Homayounfar, Wei-Chiu Ma, Justin Liang, et al. Dagmapper: Learning to map by discovering lane topology. In *International Conference on Computer Vision (ICCV)*, October 2019.
- [19] K. Huang, Y. Wang, Z. Zhou, et al. Learning to Parse Wireframes in Images of Man-Made Environments. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 626–635, 2018.
- [20] Nikhil Ketkar. *Introduction to PyTorch*, pages 195–208. Apress, Berkeley, CA, 2017.
- [21] Yeongmin Ko, Jiwon Jun, Donghwuy Ko, and Moongu Jeon. Key points estimation and point instance segmentation approach for lane detection. *arXiv:2002.06604*, 2020.
- [22] Hei Law and Jia Deng. CornerNet: Detecting Objects as Paired Keypoints. *European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.
- [23] X. Li, J. Li, X. Hu, and J. Yang. Line-cnn: End-to-end traffic line detection with line proposal unit. *IEEE Transactions on Intelligent Transportation Systems*, 21(1):248–258, 2020.
- [24] Zuoyue Li, Jan Dirk Wegner, et al. Topological map extraction from overhead images. In *International Conference on Computer Vision (ICCV)*, October 2019.
- [25] Justin Liang, Namdar Homayounfar, Wei-Chiu Ma, et al. Polytransform: Deep polygon transformer for instance segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [26] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2018.
- [27] T. Lin, P. Goyal, R. Girshick, et al. Focal Loss for Dense Object Detection. In *International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.
- [28] Huan Ling, Jun Gao, Amlan Kar, et al. Fast interactive object annotation with curve-gcn. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, et al. SSD: Single Shot MultiBox Detector. 2016.
- [30] Van Nhan Nguyen, Robert Jenssen, and Davide Roverso. Ls-net: Fast single-shot line-segment detector. *arXiv preprint arXiv:1912.09532*, 2019.
- [31] Jan-Hendrik Pauls, Tobias Strauss, Carsten Hasberg, et al. Can we trust our maps? An evaluation of road changes and a dataset for map validation. In *International Conference*

- on Intelligent Transportation Systems (ITSC)*, pages 2639–2644, 2018.
- [32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [33] Zequn Qin, Hanyu Wang, and Xi Li. Ultra fast structure-aware deep lane detection. In *The European Conference on Computer Vision (ECCV)*, pages 276–291. Springer, 2020.
- [34] Joseph Redmon, Santosh Divvala, Ross Girshick, et al. You only look once: Unified, real-time object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [35] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.
- [36] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018.
- [37] Shaoqing Ren, Kaiming He, Ross Girshick, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.
- [38] Martin Simon, Stefan Milz, Karl Amende, et al. Complex-yolo: Real-time 3d object detection on point clouds. *arXiv:1803.06199*, 2018.
- [39] Lucas Tabelini, Rodrigo Berriel, Thiago M Paixão, Claudine Badue, Alberto F De Souza, and Thiago Olivera-Santos. Keep your eyes on the lane: Attention-guided lane detection. *arXiv:2010.12035*, 2020.
- [40] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.
- [41] M. Tan, R. Pang, and Q. V. Le. EfficientDet: Scalable and Efficient Object Detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020.
- [42] Z. Tian, C. Shen, H. Chen, et al. FCOS: Fully Convolutional One-Stage Object Detection. In *International Conference on Computer Vision (ICCV)*, pages 9626–9635, 2019.
- [43] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Dennis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [44] R. Grompone von Gioi, J. Jakubowicz, J. Morel, et al. LSD: A Fast Line Segment Detector with a False Detection Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, 2010.
- [45] Nan Xue, Song Bai, Fudong Wang, et al. Learning attraction field representation for robust line segment detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [46] N. Xue, S. Bai, F.-D. Wang, et al. Learning Regional Attraction for Line Segment Detection. *Transactions on Pattern Analysis and Machine Intelligence*, 2019.
- [47] Nan Xue, Tianfu Wu, Song Bai, et al. Holistically-attracted wireframe parsing. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [48] Ze Yang, Shaohui Liu, Han Hu, et al. RepPoints: Point Set Representation for Object Detection. In *International Conference on Computer Vision (ICCV)*, pages 9656–9665, 2019.
- [49] Yichao Zhou, Haozhi Qi, and Yi Ma. End-to-end wireframe parsing. In *International Conference on Computer Vision (ICCV)*, October 2019.