Supplementary Material for
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

1. Overview

In this supplementary material, we provide further details on YOLinO. In Section 2, the base architecture is explained in detail. Here, we also provide training parameters. An estimation of the discretization errors can be found in Section 3. For further detail on the non-maximum suppression (NMS) or the TuSimple-specific post-processing, please refer to Section 4. Following, we describe the calculation of our evaluation metrics in Section 5, followed by an overview of all experiments of the ablation study in Section 6. Qualitative results from all three datasets can be found in Section 7, Section 8 and Section 9.

2. Architecture

For the backbone we chose YOLO9000 with Darknet-19 [4]. 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 1 shows the full architecture with one upsampling block, resulting in cells of $16 \times 16\,\mathrm{px}$.

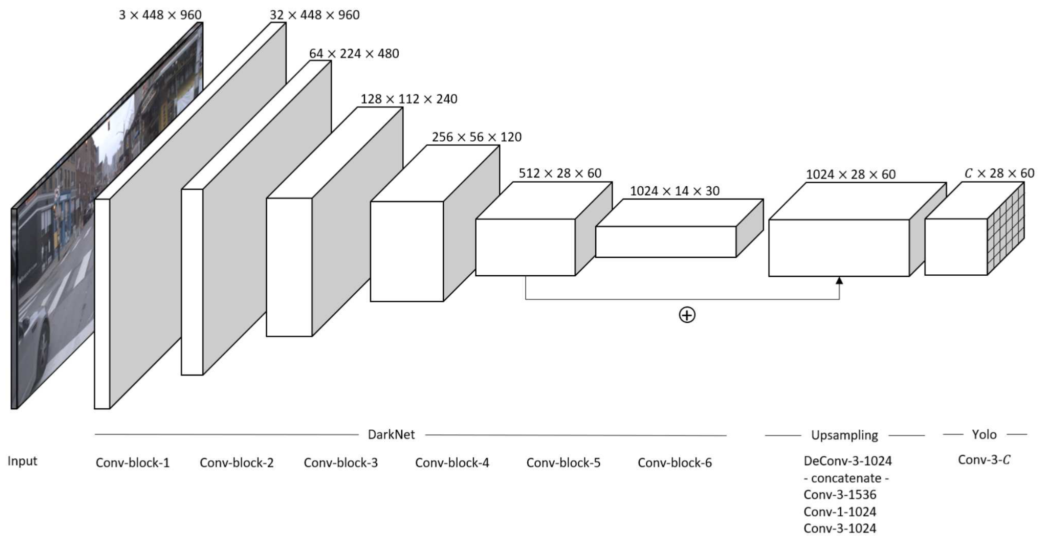


Figure 1. YOLinO's feed-forward architecture. Here shown for the Argoverse input images and with a single upsampling stage, leading to grid cells of $16 \times 16\,\mathrm{px}$. Each grid cell predicts up to 12 line segments with a geometric definition g, a confidence value c and an optional classification l, all together resulting in the output channels C.

The training is implemented in Pytorch [2] and parameterized according to the configuration in Table 1.

[*]equally contributed

	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

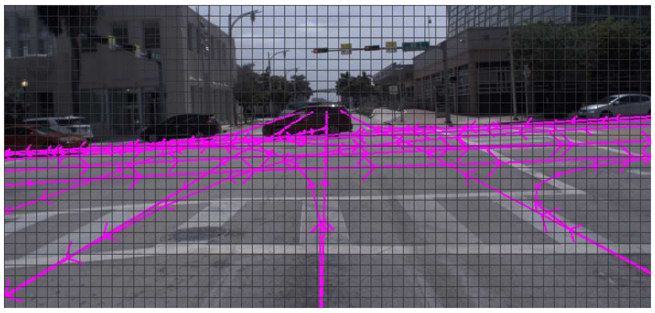
Table 1. List of training parameters of each experiment shown in qualitative results. Epochs until convergence are for unbound Cartesian points with eight predictors and 16 px resolution.

3. 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 2. As expected, the introduced error is smaller the higher the grid resolution. We also see higher values for the Karlsruhe Aerial Images (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 conclude that the most suitable grid resolution depends on the specific problem setting as finer grids also introduce significantly higher computational costs.

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


(a) Original Argoverse ground truth

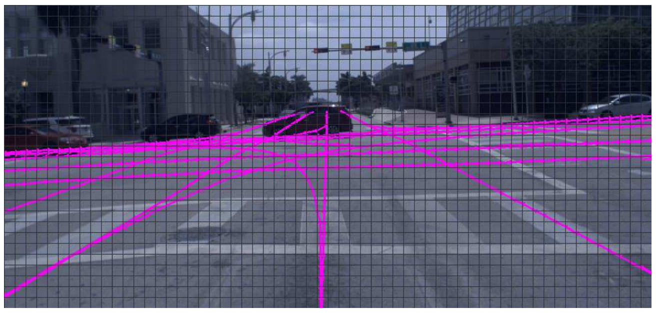

(b) Discretized ground truth

Figure 2. Comparison of the original Argoverse ground truth center lines with the discretized grid lines used for training.

4. 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 3). 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, $\tilde{g} = (m_x, m_y, \ell, d_x, d_y)^\intercal$ 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_e, g_s 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:

$$\begin{pmatrix} m_x \\ m_y \end{pmatrix} = \lambda_m \kappa \frac{\hat{g}_e + \hat{g}_s}{2} \tag{1}$$

$$\Delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \hat{g}_e - \hat{g}_s \tag{2}$$

$$\ell = \lambda_\ell \left\| \Delta \right\| \tag{3}$$

$$d_x = \lambda_d \frac{\delta_x}{\ell} \tag{4}$$

$$d_y = \lambda_d \frac{\delta_y}{\ell} \tag{5}$$

The factors λ scale the coordinates appropriately and are determined empirically for each dataset (cf. Table 3). 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.

		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 3. List of NMS parameters for each dataset.

Next, we apply the DBSCAN clustering by Scikit-learn [3] 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 [1]. The overall speed of 230 fps holds for all three steps, conversion, DBSCAN and averaging taken together. 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 topmost predictors in each "tree"), but are possibly successor to multiple predictors, we start a breadth-first search starting from those root nodes (cf. Figure 3). At

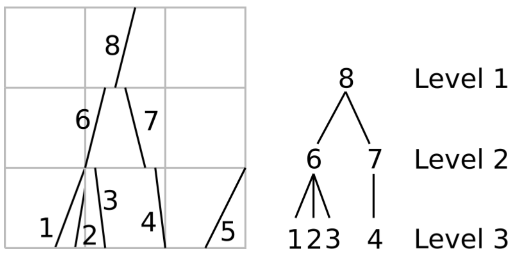
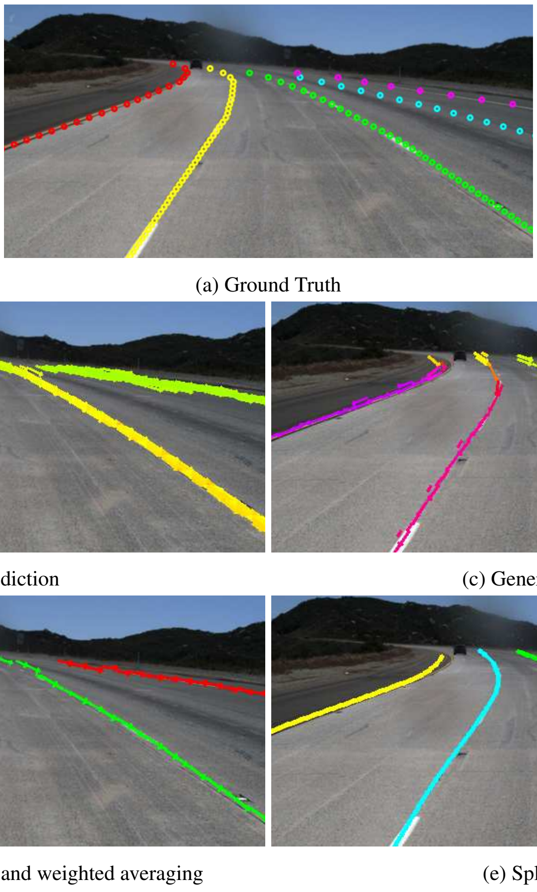


Figure 3. Illustration of the breadth-first search (BFS) and averaging used for polyline extraction, here using $\tau_s = 3$. Segments 1, 2, and 3 share the same successor, as do segments 6 and 7. 8 is a root node and will initiate one BFS procedure leading to the tree on the right with three levels. 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.

each level, we 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 [5] `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 4.



(a) Ground Truth

(b) Prediction

(c) Generic NMS

(d) Breadth-first search and weighted averaging

(e) Spline fit

Figure 4. 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.

5. 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

$$R = \frac{|TP|}{|GT|}. \tag{6}$$

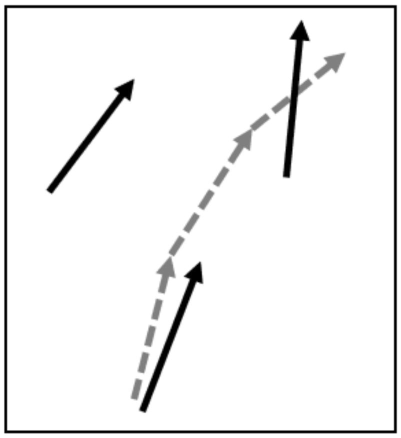
Comparing the true positives with the number of predictions $|PD|$, we get a precision with

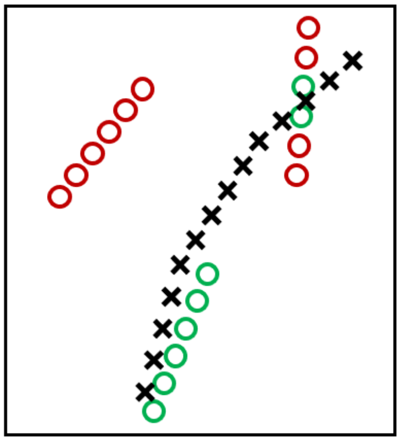
$$P = \frac{|TP|}{|PD|}. \tag{7}$$

Combining both scores leads to

$$F1 = 2 * \frac{P \cdot R}{P + R}. \tag{8}$$

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 $1\,\mathrm{px}$. 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 5 visualizes this for the 2D case without regarding the orientation α. An example from the TuSimple dataset is depicted in Figure 6.


(a) Exemplary predicted line segments (solid) and a single ground truth polyline (dashed).


(b) We calculate a precision of $P = \frac{8}{18}$. Here, red circles indicate false positives (FP) and green circles are accounted as true positives (TP).

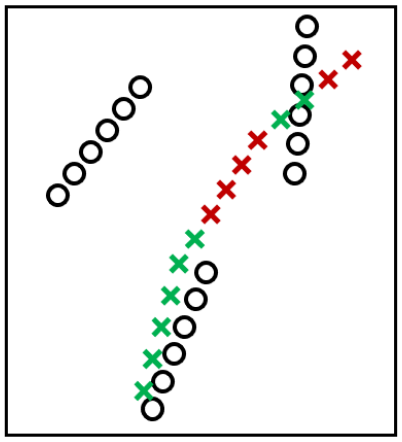
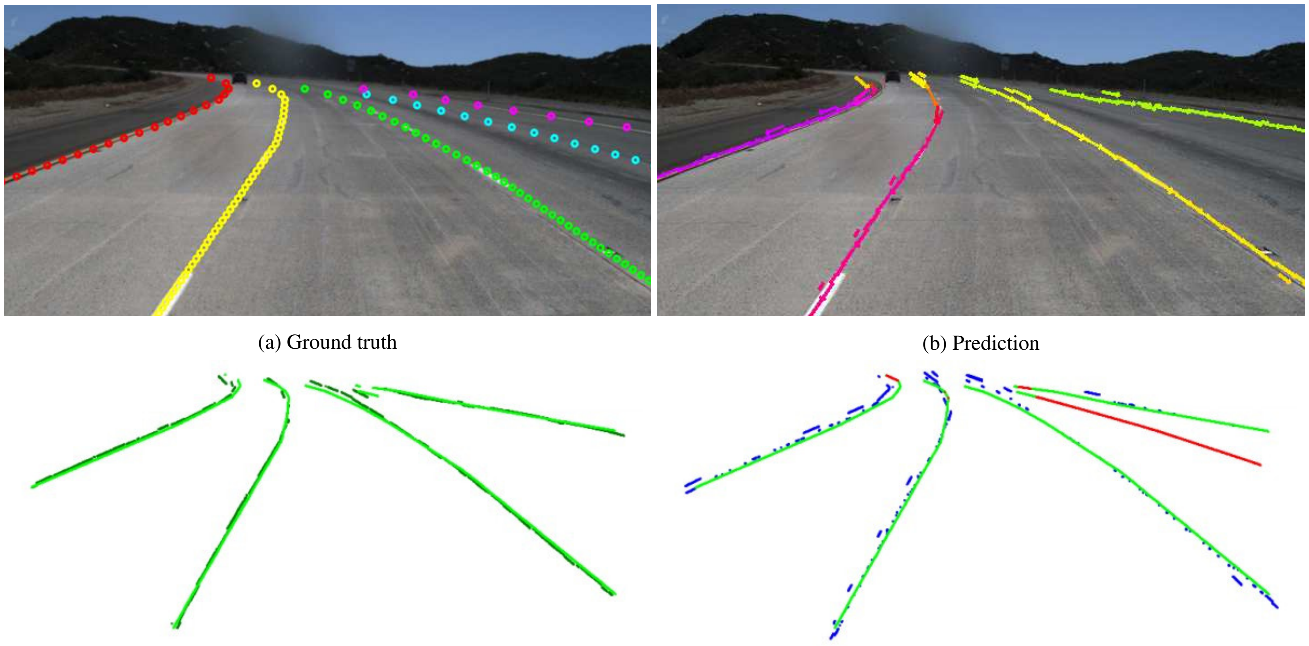

(c) The recall sums up to $R = \frac{8}{14}$. We depicted false negatives (FN) as red crosses and true positives (TP) as green crosses.

Figure 5. 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.



(a) Ground truth

(b) Prediction

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

6. Relevant experiments

For better comparison between the several ablation studies, we provide all results in Table 4. 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.

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

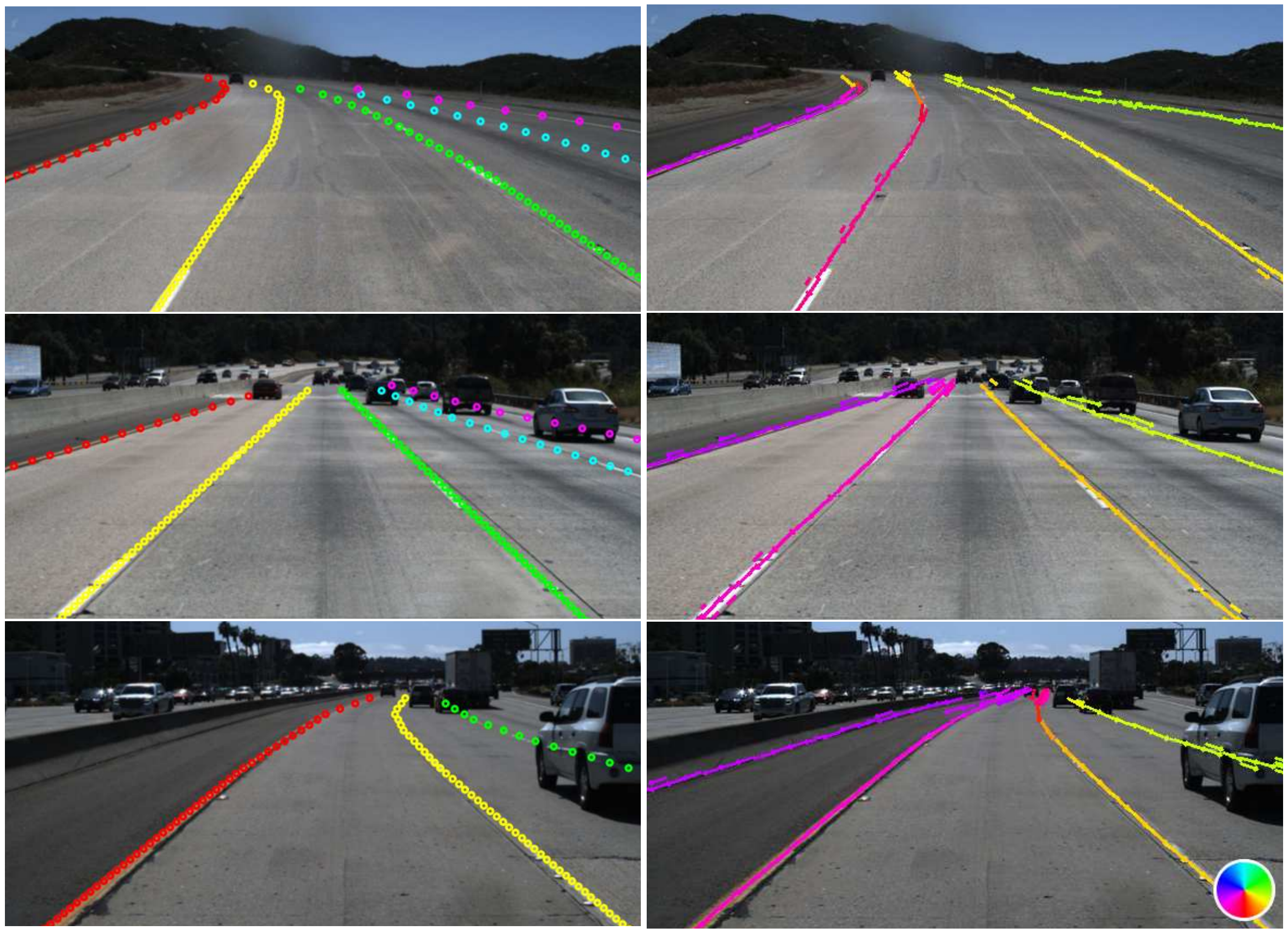
Table 4. Results for the ablation studies on important hyperparameters on the Tusimple validation set. Best result in each group is bold, second best result is underlined.

7. Qualitative results for TuSimple

We depicted further examples for the TuSimple dataset in Figure 7 and Figure 8. 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.

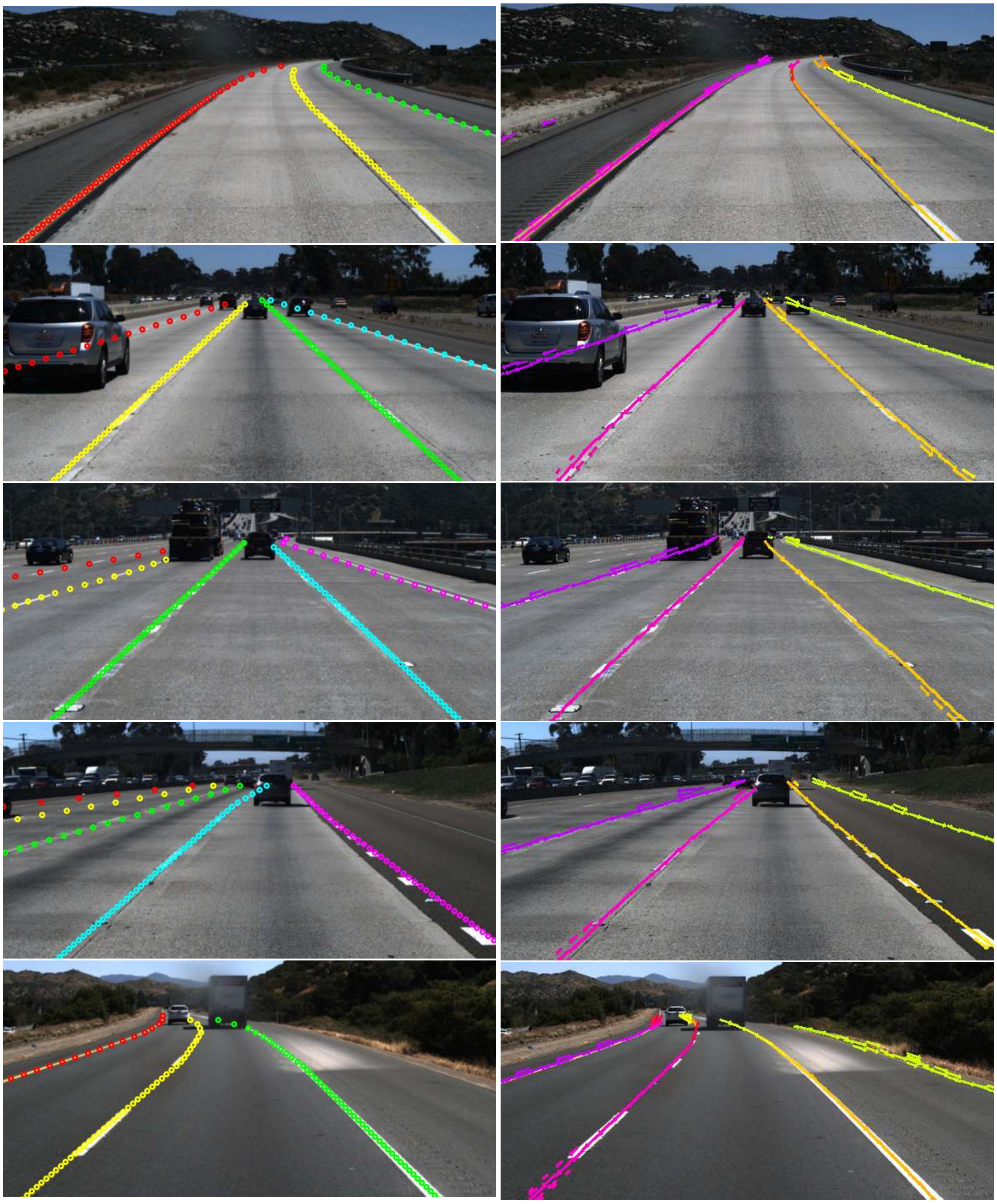
Similarly, predicting false positive and false negative lane boundaries happens (FP rate of .188 and FN rate of .076) as there also exist quite some ambiguities in the labeling. E.g. considering the first row in Figure 7, the blue lane boundary (in the ground truth) does not have any visual cue and is very doubtful even for humans. Thus, our approach does not recognize this as a valid boundary. Comparing the second and third row of Figure 7, 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 8, 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 8.



(a) Ground Truth (b) Prediction after generic NMS

Figure 7. 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.



(a) Ground Truth

(b) Prediction after generic NMS

Figure 8. 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.

8. Qualitative results for Karlsruhe Aerial Images

We depicted further examples for the KAI dataset in Figure 9 and Figure 10. 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 9, 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 10, 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 9 and in the first and third row of Figure 10, 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 10)

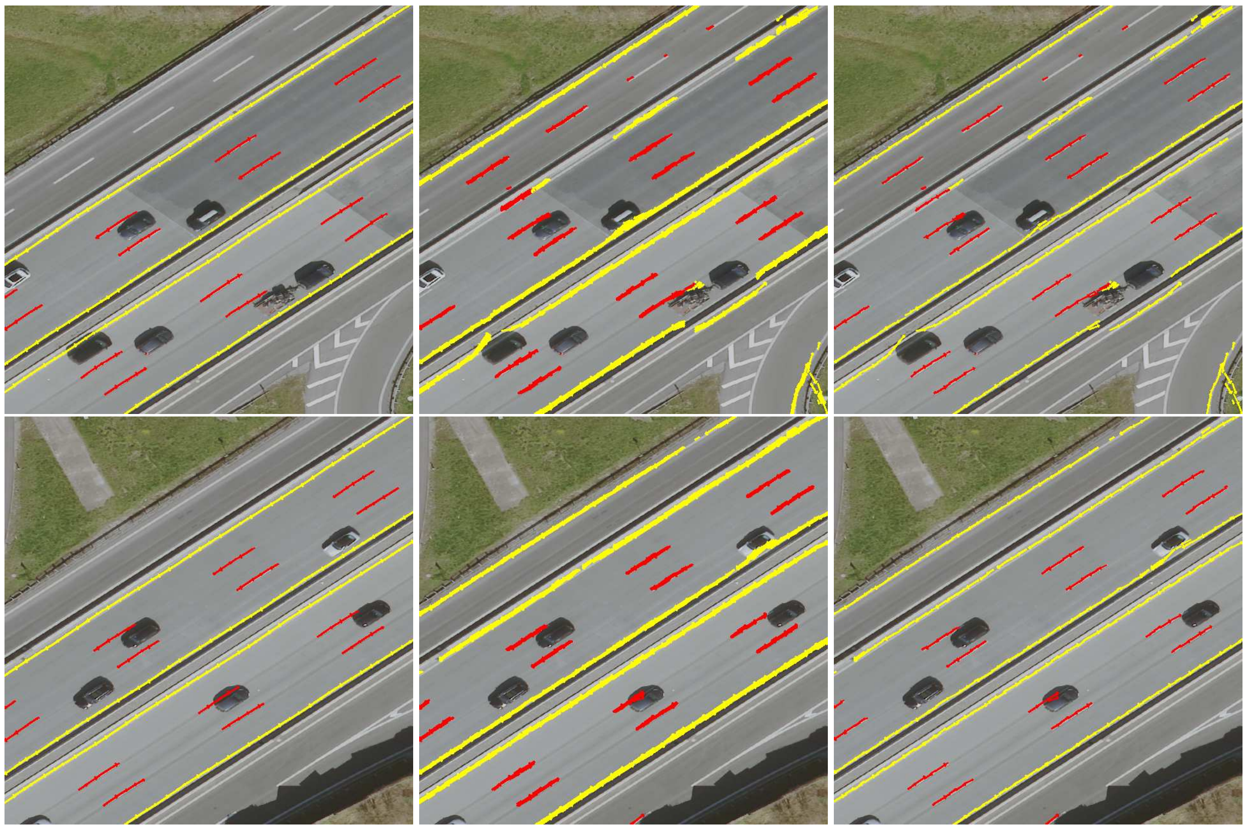


Figure 9. Example results of the KAI dataset (unbound Cartesian points (Po), eight predictors and a grid resolution of 16×16 px).
Aerial images: © City of Karlsruhe | Liegenschaftsamt

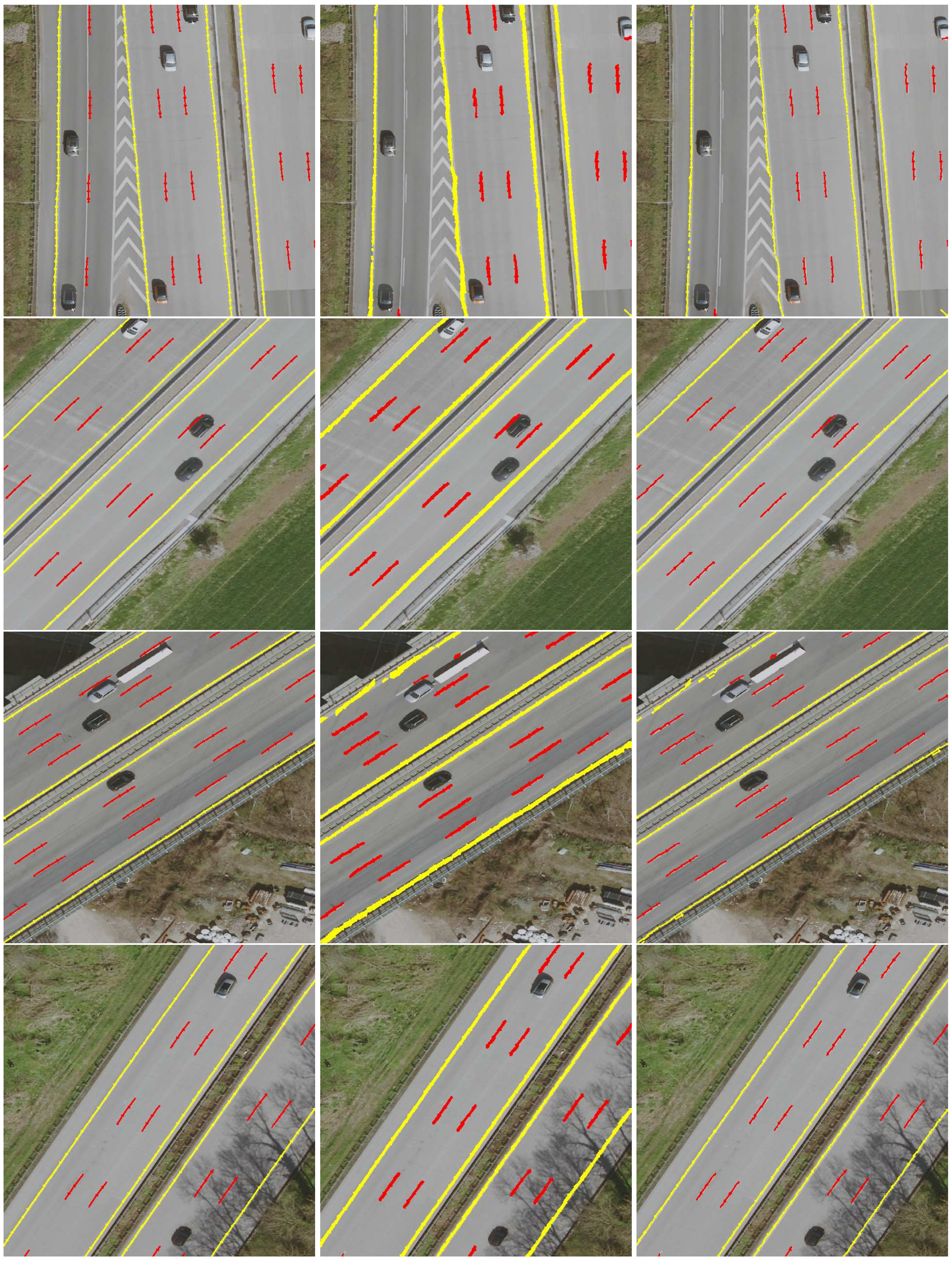

(a) Ground Truth (b) Prediction (c) Prediction after NMS

Figure 10. 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

9. 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 11 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 12 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 11 with the second and fourth rows of Figure 12, it becomes clear that the network is also able to correctly predict the driving direction of lanes, as Figure 12 also shows scenes with a single driving direction.

In some scenes (2nd row in Figure 12, 2nd row in Figure 11), 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 12 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.

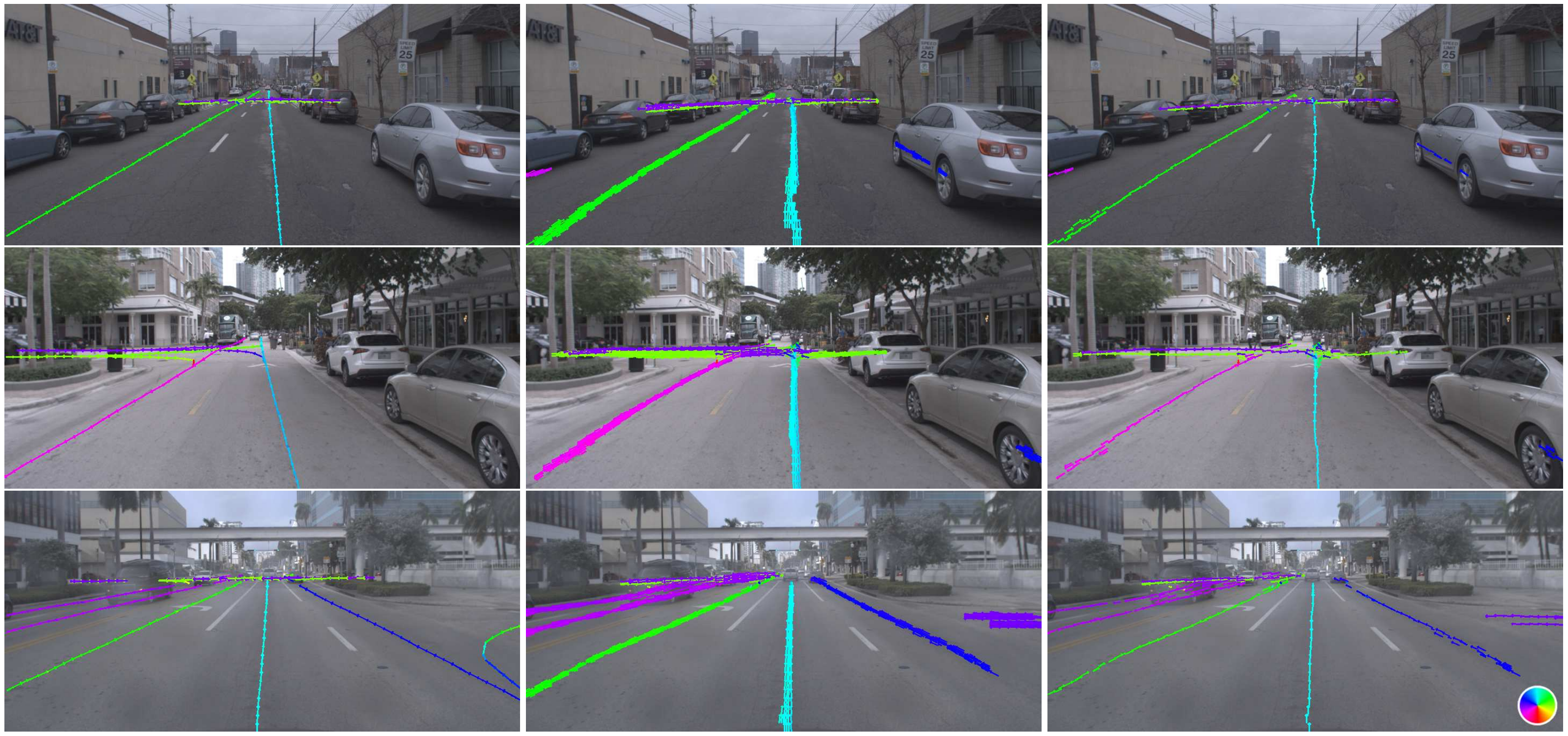
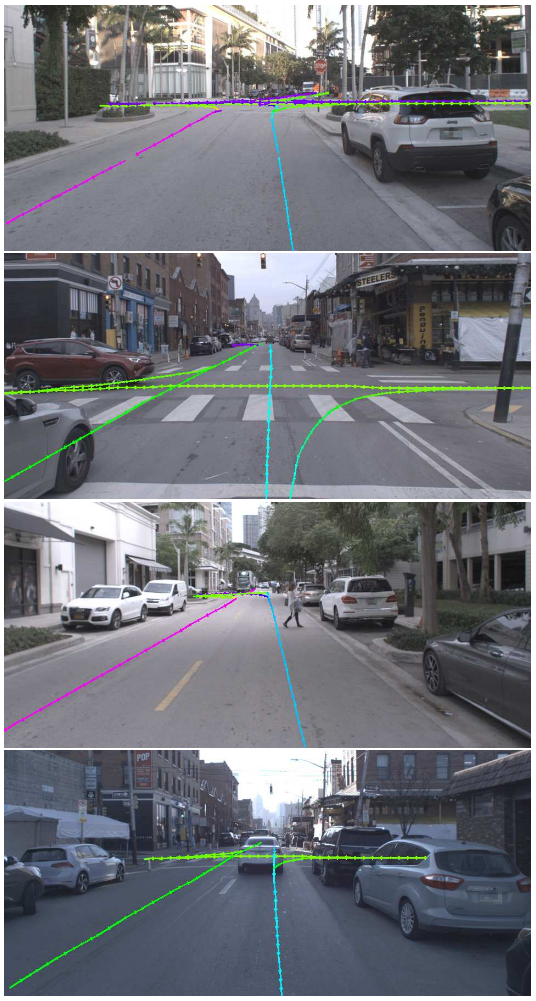
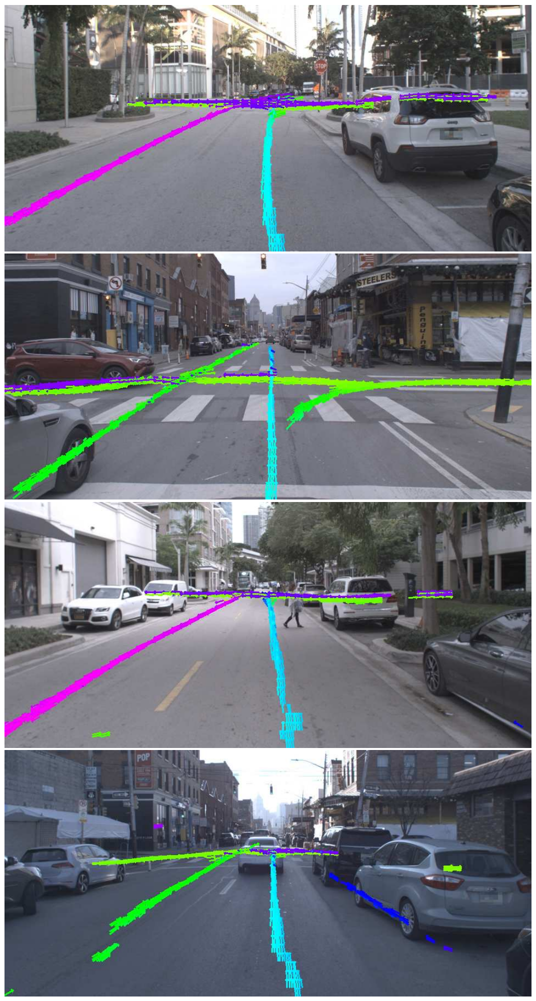


Figure 11. Example results of the Argoverse dataset (unbound Cartesian points (Po), eight predictors and a grid resolution of $16{\times}16$ px). Colors indicate the orientation of the predicted line segments.


(a) Ground Truth


(b) Prediction

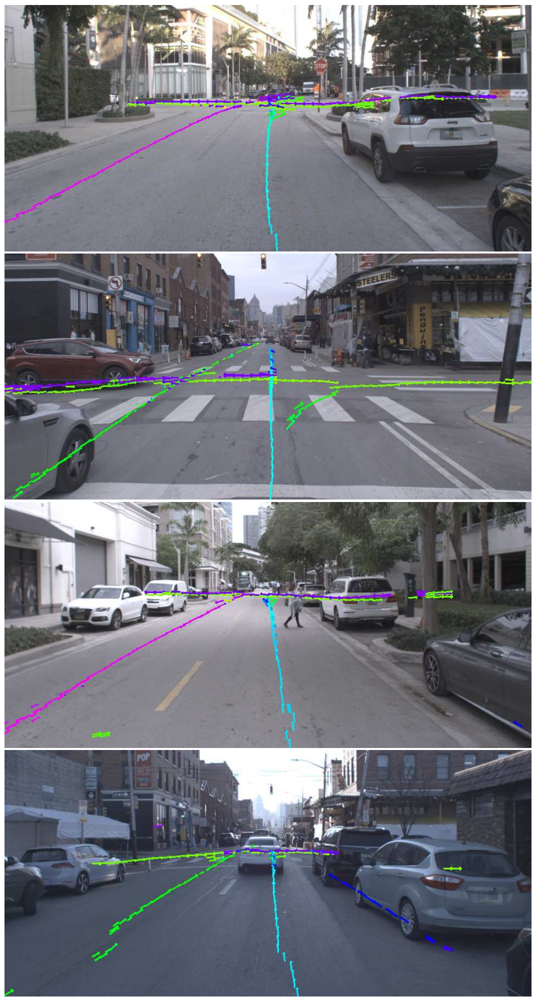

(c) Prediction post-processed

Figure 12. 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.

References

[1] 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 Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-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.

[2] Nikhil Ketkar. *Introduction to PyTorch*, pages 195–208. Apress, Berkeley, CA, 2017.

[3] 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.

[4] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.

[5] 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, Denis 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.