A Appendix - implementation details

A.1 Training

All training sessions consisted of 30,500 iterations (batches) with constant learning rate and no weight decay. In case of adversarial training, on each iteration, both discriminator and generator were updated. Test results reported were averaged over results obtained with model snapshots at iterations 30,100, 30,200, 30,300, 30,400 and 30,500.

Batch size. For all supervised methods (optimizing $\mathcal{L}_{task}$), including the supervision over translated images in the **image translation** method, we used batch size 24. In cases in which supervised training is with two domains (e.g. **small + syn. supervision**) each batch consisted of 12 examples from each domain. For the other domain adaptation methods, **CMD**, **autoencoder**, **embedding GAN** and **self-supervision** methods, each batch consisted of 16 source domain images and 16 target ones.

learning rate. For all weight updates the learning is 10^{-4} except for the discriminator weights updated with learning rate $5 \cdot 10^{-4}$.

Loss balancing. For all wgan-gp implementation the gradient penalty loss weight was 10. In adversarial training the generator loss weight is 0.2. For the **CMD** the regularization loss weight is 10^{-3}. For the autoencoder the weight of the reconstruction loss $\alpha = 10$.

DA method combination. Whenever DA methods are combined (e.g. **AE+S**), the same target domain batch is used multiple times (one per method), each time for optimizing a different loss.

A.2 CMD

Moment matching aims at matching the distributions of the intermediate feature maps of the source and target domain by minimizing the distance between their first order moments. We follow the method described in [11]. Given a batch of source and domain examples their corresponding feature maps f_s, f_t with C channels in each are computed. For each example, each spatial entry in the feature map is considered as a single C dimensional sample. The collection of all samples from each domain $z_s, z_t \in \mathbb{R}^C$ is then the input to the central moment discrepancy loss [11].

A.3 Evaluation

Segment-based evaluation To complete the description of the proposed segment-based evaluation described in Section 4 we define the distance $seg\_dist(\mathbf{p}, \mathbf{q})$ between two segments in the plane. The idea was to try to match lane segments that belong to the same lane, and therefore there is an emphasis on their orientation, and different treatment of the distance *along* the segment direction and *perpendicular* to it.

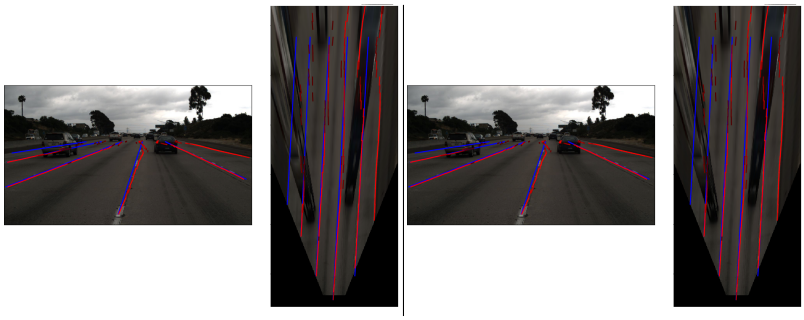

Fig. 5. Two examples of the forgivingness of the tuSimple evaluation towards the lane segment output. Ground truth marked in blue and detected segments in red. In both examples the per-image tuSimple score was high: 0.93 (left), 0.94 (right), while as can be observed there are many erroneous segments. In the right example, at the far range there are clearly large lateral offsets between the detections and the gt. However, since the tuSimple evaluation is in the image plane, these offsets become very small. In the right example, there are many small false segments on the far left and far right sides which the clustering manages to filter out.

Let l_p, l_q be the corresponding infinite lines on which p, q reside, and let $((p^1, p^2), (q^1, q^2) \in \mathbb{R}^2$ be the segment endpoints. We project each endpoint to the opposite line (i.e. p^1 is projected to l_q and denoted by p^1_q) to generate four projected points: $(p^1_q, p^2_q), (q^1_p, q^2_p) \in \mathbb{R}^2$. We start by eliminating some of the matches if the projected segment doesn't sufficiently overlap with the opposite segment: $\frac{|p^1_q p^2_q|}{|q^1 q^2|} > 0.5$ or $\frac{|q^1_p q^2_p|}{|p^1 p^2|} > 0.5$. Matches are eliminated by setting $seg\_dist(\mathbf{p}, \mathbf{q}) = \infty$. Finally, for the remaining matching pairs, distance is computed as the maximum distance between the end-points and their projected counterparts: $seg\_dist = \max(|p^1 p^1_q|, |p^2 p^2_q|, |q^1 q^1_p|, |q^2 q^2_p|)$.

tuSimple evaluation As mentioned in Section 4 for the tuSimple evaluation we need to cluster the segments. For this purpose we apply a heuristic clustering algorithm, operating on the tile representation output. Initially, in each row we apply a 1D non-maxima suppression with kernel size 20cm to suppress redundant detections of the same lane in neighboring tiles.

The clustering progresses row by row from bottom to top. For each segment in a top row, it is connected to its 3 closest neighbors in the previous (bottom) row. Per connection, an affinity score is computed. The affinity measures the likelihood that the two segments belong to the same lane, using the following parameters. θ - the orientation difference between the segments, d_{min} - the minimum euclidean distance between their endpoints, b_{curr} - confidence score for the current segment and b_{prev} - confidence score of the connected neighbor from the previous row. If $\theta > 45$ then the affinity $a = 0$, otherwise, it is computed as: $a = b_{curr} \cdot b_{prev} \cdot \cos(\theta) \cdot \frac{8 - d_{min}}{8}$. We then cluster the current segment with

its highest scoring neighbor. For each cluster b_{max} is set as the maximum score over segments in the cluster.

We next apply a filtering stage dropping clusters with less than four segments, and clusters with $b_{max} < 10^{-2}$. Finally, we loop over all remaining cluster pairs and merge a pair if one cluster starts on the same row as the other one ends on horizontally-adjacent tiles. We continue untill no pairs can be merged. As can be seen in Figure 5, due to the clustering, and the forgivingness of the tuSimple evaluation, especially at the farther ranges, the final metric is not very indicative of the quality of the raw tile-representation output.

A.4 Additional examples

In Figure 6 we show examples of additional results before and after domain adaptation in the UDA setting for the three leading methods, namely, image translation, self-supervision and our auto-encoder approach.

A.5 Architectures

Tables 2-7 in this section specify the CNN architectures. Wherever input_name is empty it is the output of the line above. "+" in the input means concatenation along channel dimension. All ReLUs are leaky relu with factor 0.1.

type	Cin	Cout	Kernel	Stride	input_name	Output name
ConvBNRelu	3	32	3	1	bev image	
ConvBNRelu	32	32	3	1		
maxpool			2	2		embed_2
ConvBNRelu	32	64	3	1	embed_2	
ConvBNRelu	64	64	3	1		
maxpool			2	2		embed_4
ConvBNRelu	64	128	3	1	embed_4	
ConvBNRelu	128	128	3	1		
ConvBNRelu	128	128	3	1		
maxpool			2	2		embed_8
ConvBNRelu	128	128	3	1	embed_8	
ConvBNRelu	128	128	3	1		
ConvBNRelu	128	128	3	1		
maxpool			2	2		embed_16

Table 2. Base architecture. Image Embedding network, ϕ.

type	Cin	Cout	Kernel	Stride	input_name
ConvBNRelu	128	64	3	1	embed_16
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	10	1	1	

Table 3. Base architecture. Embedding to tile-representation network, ψ

type	Cin	Cout	Kernel	Stride	input_name	output_name
ConvBNRelu	128	256	3	1	embed_16	
ConvBNRelu256	256	256	3	1		
ConvBNRelu	256	256	3	1		
maxpool			2	2		embed_32
ConvBNRelu	128	64	3	1	embed_16	embed_16_reduced
ConvBNRelu	128	64	3	1	embed_8	embed_8_reduced
ConvBNRelu	64	32	3	1	embed_4	embed_4_reduced
Nearest Upsample			2		embed_32	
ConvBNRelu	256	64	3	1		
ConvBNRelu	64	64	3	1		
ConvBNRelu	64	64	3	1		embed_16_up
Nearest Upsample			2		embed_16_reduced + embed_16_up	
ConvBNRelu	128	64	3	1		
ConvBNRelu	64	64	3	1		
ConvBNRelu	64	64	3	1		embed_8_up
Nearest Upsample			2		embed_8_reduced + embed_8_up	
ConvBNRelu	128	32	3	1		
ConvBNRelu	32	32	3	1		
ConvBNRelu	32	32	3	1		embed_4_up
ConvBNRelu	64	64	3	1	embed_4_reduced + embed_4_up	
ConvBNRelu	64	64	3	1		
ConvBNRelu	64	64	3	1		
ConvBNRelu	64	1	3	1		l - lanes image

Table 4. Autoencoder. embedding to skeleton network, δ

type	Cin	Cout	Kernel	Stride	input_name	output_name
convSpectralNormRelu	10	64	4	2	lanes image	
convSpectralNormInstanceNormRelu	64	128	4	2		
convSpectralNormInstanceNormRelu	128	256	4	2		
convSpectralNormInstanceNormRelu	256	512	4	2		
Conv	512	128	4	1		d_internal
Minibatch discrimination layer	128	128			d_internal	md
Conv	256	1	1	1	d_internal + md	

Table 5. Autoencoder. Discriminator network, γ. For **Embedding GAN** the discriinator is identical, and the input is *embed*_16 with channel size is 128. Spectral Norm is described in [44] and the Minibatch discrimination layer in [45]

.

type	Cin	Cout	Kernel	Stride	input_name
ConvBNRelu	1	16	3	1	lane image
ConvBNRelu	16	32	3	1	
ConvBNRelu	32	64	3	1	
Nearest Upsample			2		
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	64	3	1	
Nearest Upsample			2		
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	64	3	1	
ConvBNRelu	64	1	3	1	

Table 6. Autoencoder. Skeleton to gradient image network, λ.

type	Cin	Cout	Kernel	Stride	input_name
ConvBNRelu	128	64	(5, 3)	1	embed_16
maxpool			2	2	
ConvBNRelu	64	64	(5, 3)	1	
maxpool			2	2	
conv	64	3	1	1	

Table 7. Self-supervision. Auxiliary task classifier, ρ.

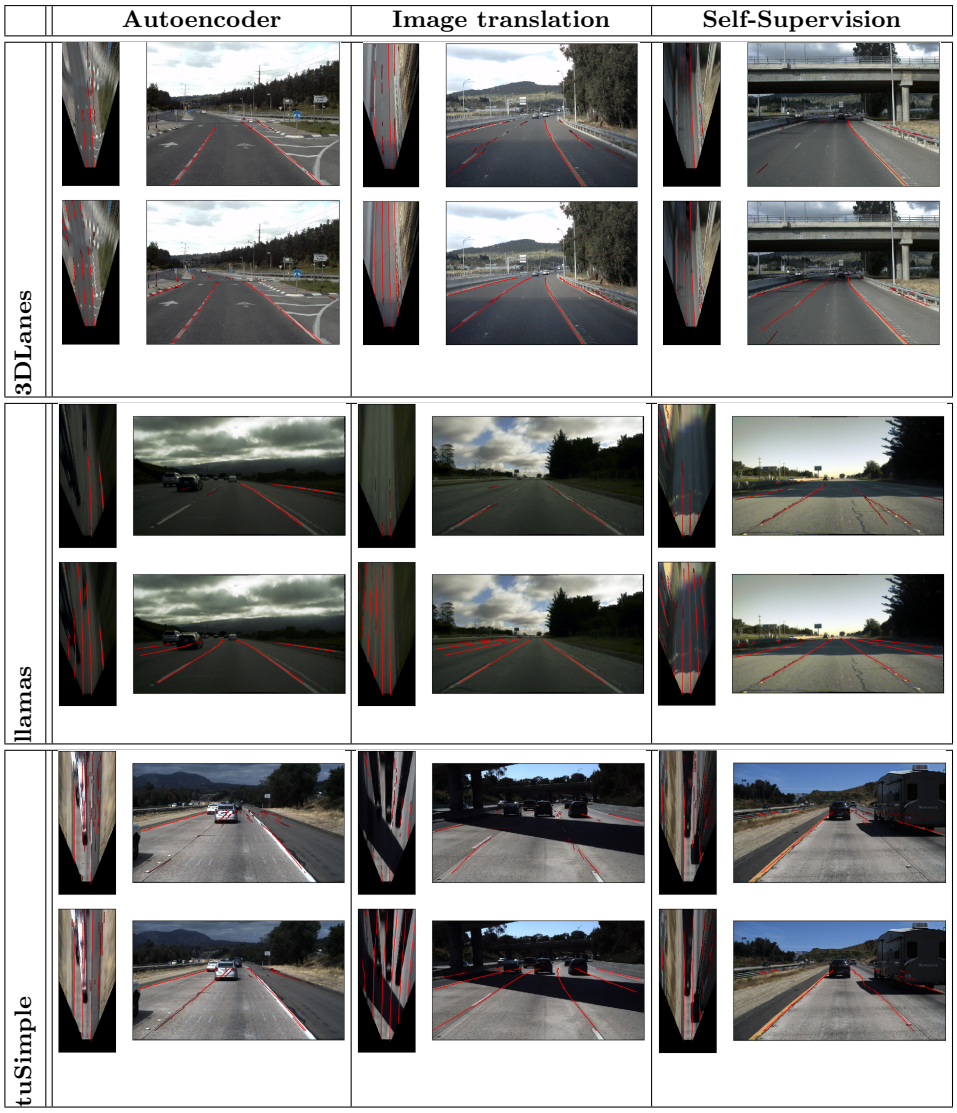

Fig. 6. Additional Results in the unsupervised domain adaptation (UDA) setting. Sample results using the three leading methods (columns) on samples from the three different datasets we tested on (rows). **In each cell** we show the result before (top within cell) and after domain adaptation with the respective method (bottom). Results are shown in each cell in both top-view (right) and regular view (left).