SemARFlow: Injecting Semantics into Unsupervised Optical Flow Estimation for Autonomous Driving (Appendix)

Shuai Yuan, Shuzhi Yu, Hannah Kim, and Carlo Tomasi
Duke University
{shuai, shuzhiyu, hannah, tomasi}@cs.duke.edu

Contents

A. Network structure

Our detailed network structure is shown in Fig. 1 and Fig. 2. The dimension of each convolutional layer and each tensor in the network is marked in the figure.

In the semantic encoder, we have six levels of layer groups, and each layer group consists of two convolutional layers with leaky ReLU non-linearity. The layers in the same group have the same number of output channels marked in the figure. We use 3×3 convolutional kernels everywhere. The dilation value is set as 2 whenever we need to reduce the dimension by half.

The iterative decoder has a flow estimator, a context network, and a upmask net, which are shared across all levels. Some 1×1 convolutional layers are applied to transform the input features of different levels to features with the same number of channels so that they can be fed into the same shared flow estimator.

Our upmask net outputs a 144-channel mask for upsampling. We first unfold the 144 channels to 16 groups, each of which has 9 values. Since we are upsampling four times, each original value needs to correspond to 16 values in the output, and each output value is computed as a convex linear combinition of the 3×3 input window, so each group of 9 values in the mask are used as the coefficients here. We apply a softmax transform to make sure these 9 coefficients sum up to be one.

B. Supplementary results

B.1. Full data tables

We provide the full data table of all our experiments on the validation set in Tabs. 1 to 4. For test sets, we have submitted test results to the benchmark website, so please refer to the website for full evaluations.

B.2. More qualitative results

More qualitative results on the KITTI-2015 test set are shown in Fig. 3.

B.3. Time efficiency

Our network runs very efficiently thanks to its small size. Our network with semantic encoder and learned upsampler only has 2.6M parameters in total, so the model parameter size is only around 10MB.

Training We run 200k iterations in total. For our basic network with a semantic encoder and a learned upsampler, it takes around 44-48 hours on 2 NVIDIA GeForce RTX 2080 Ti GPUs. After adding semantic augmentation, it takes longer because we add a third forward pass of the network, but since we only use semantic augmentation for the last 50k iterations, the running time is still feasible: 54-58 hours on 2 NVIDIA GeForce RTX 2080 Ti GPUs.

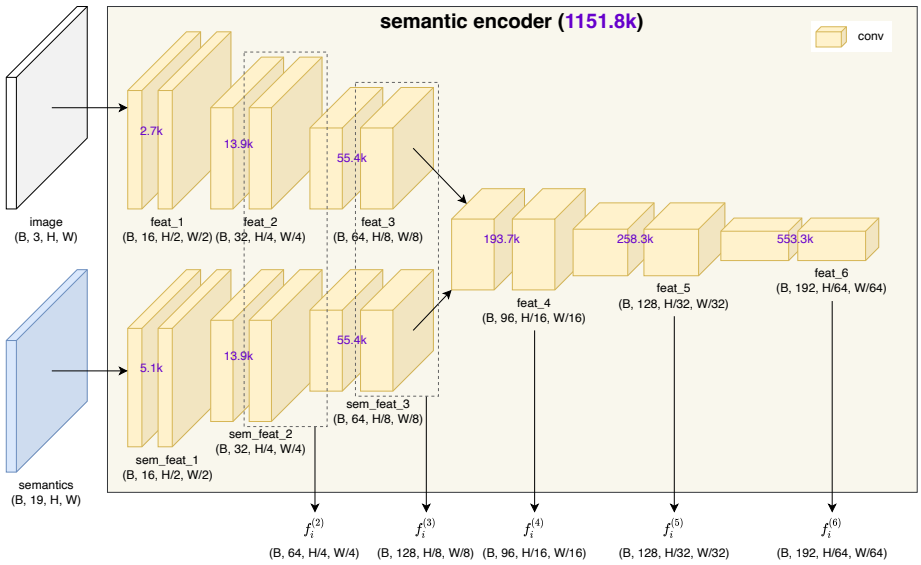


Figure 1. Semantic encoder network structure. Purple numbers show the number of parameters. B is batch size; (H, W) is the input resolution.

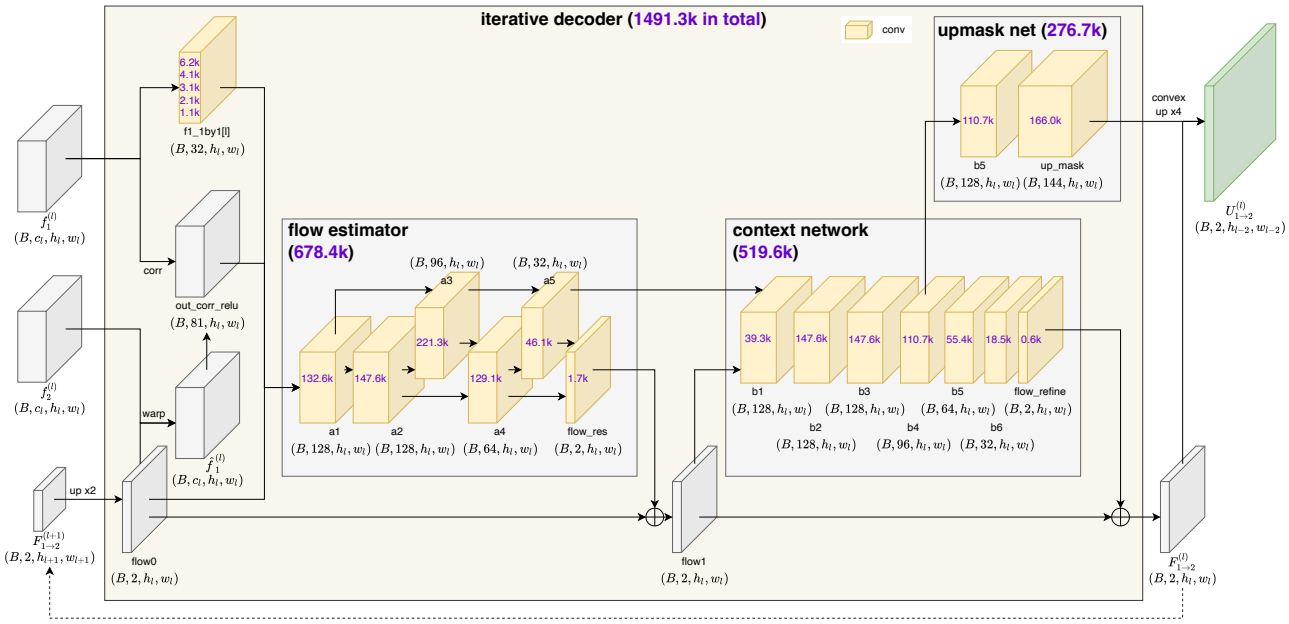


Figure 2. Iterative decoder network structure. Purple numbers show the number of parameters. The 1×1 convolution layers are not shared, so we show the number of parameters at each level. B is batch size; (h_l, w_l) is the resolution of the l-th level; c_l is the number of channels.

Inference For inputs of size 256×832, inferring the forward flow of each sample takes $0.0168(\pm 0.0005)$ second, *i.e.* 60 frames per second, on a single NVIDIA GeForce RTX 2080 Ti GPU.

Tips on how to process semantic inputs efficiently We need to one-hot encode the semantic map input before feeding into the network. We do one-hot transformation *after* data augmentation to save time because otherwise, doing horizontal flip or bilinear interpolation for a 19-channel map is very time-consuming. We also avoid flipping or rescaling these 19-channel semantic maps when we copy and paste occluder objects across samples for semantic augmentation.

up	no sm	enc	aug	KITTI-2015					KITTI-2012				
				Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
				10.360	8.528	2.901	2.068	6.967	5.707	3.741	1.406	0.886	4.417
		1		9.920	8.087	2.685	1.885	6.568	5.484	3.606	1.354	0.861	4.205
		2		9.830	7.951	2.646	1.857	6.396	5.450	3.592	1.339	0.860	4.115
		3		9.745	7.977	2.608	1.853	6.543	5.359	3.511	1.328	0.850	4.098
		4		9.745	7.951	2.644	1.852	6.553	5.437	3.564	1.343	0.852	4.183
✓				10.220	8.221	2.825	1.970	6.748	5.728	3.725	1.420	0.888	4.495
✓	✓			8.871	7.142	2.605	1.849	6.037	5.314	3.374	1.386	0.876	4.347
✓	✓	3		8.260	6.577	2.415	1.729	5.794	4.964	3.111	1.291	0.825	**4.007**
✓	✓		✓	8.801	6.748	2.484	1.595	6.642	5.421	3.281	1.411	0.852	4.653
✓	✓	3	✓	**7.788**	**5.963**	**2.179**	**1.399**	**5.635**	**4.872**	**2.932**	**1.284**	**0.788**	4.175

Table 1. Full validation results for Table 1, 2, & 3 in the main paper (EPE/px and Fl/%). Metrics evaluated at 'all' (all pixels, default for EPE), 'noc' (non-occlusions), 'occ' (occlusions), 'bg' (background), and 'fg' (foreground). Key metrics (used in official ranking) are underlined. 'Ours (baselinse)'= up + no sm; 'Ours (+enc)'= up + no sm + enc=3; 'Ours (+enc +aug)' = 'Ours (final)' = up + no sm + enc=3 + aug. For all metrics, lower is better.

Options of aug	KITTI-2015					KITTI-2012				
	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
Ours (final)	**7.788**	**5.963**	**2.179**	1.399	5.635	**4.872**	**2.932**	**1.284**	**0.788**	**4.175**
start from 100k	7.916	6.013	2.186	**1.396**	**5.406**	4.984	2.998	1.302	0.803	4.984
vehicles only	7.940	6.110	2.212	1.435	5.781	4.950	3.041	1.304	0.804	5.571
loss on new occ	8.149	6.105	2.272	1.418	5.620	5.167	3.083	1.361	0.810	5.892

Table 2. Full validation results for Table 4 in the main paper (EPE/px and Fl/%).

	road	**car**	**terrain**	**vegetation**	**sidewalk**	**building**	**wall**	**pole**	**fence**
Proportion	42.4%	17.6%	14.0%	11.6%	6.2%	4.1%	1.3%	1.1%	0.9%
ARFlow [3]	4.573	15.791	9.420	15.339	4.606	6.000	16.336	10.745	46.297
Ours (final)	3.674	10.171	8.737	13.470	3.085	4.275	11.180	9.437	39.120
Rel. impr.	19.7%	35.6%	7.3%	12.2%	33.0%	28.8%	31.6%	12.2%	15.5%
Rew. contri.	19.0%	49.4%	4.8%	10.9%	4.7%	3.5%	3.4%	0.7%	3.3%
	traffic sign	**truck**	**bicycle**	**traffic light**	**person**	**rider**	**motorcycle**	**sky**	**bus**	**train**
Proportion	0.4%	0.2%	0.1%	0.1%	0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%
ARFlow [3]	5.254	10.461	5.421	2.290	3.852	20.811	13.385	28.547	17.179	12.564
Ours (final)	4.681	9.526	4.945	2.060	2.627	19.581	3.620	38.521	17.747	9.615
Rel. impr.	10.9%	8.9%	8.8%	10.0%	31.8%	5.9%	73.0%	-34.9%	-3.3%	23.5%
Rew. contri.	0.1%	0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%

Table 3. Full data for Table 5 in the main paper. We show the relative improvements and reweighted contributions of all 19 classes.

Method	KITTI-2015					KITTI-2012				
	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
ARFlow (our impl.)	13.210	10.190	4.081	2.878	9.398	7.360	4.434	1.713	0.994	5.806
Ours (baseline)	12.270	8.642	3.809	2.433	9.907	7.165	4.104	1.730	0.997	5.915
Ours (+enc)†	11.280	7.749	3.327	2.121	8.749	6.583	3.596	1.561	0.912	5.267
Ours (+enc +aug)†	**10.320**	**6.950**	**2.640**	**1.558**	**7.121**	**6.204**	**3.488**	**1.489**	**0.855**	**5.150**

Table 4. Full validation results for Table 6 in the main paper (EPE/px and Fl/%). † denotes models with semantic inputs.

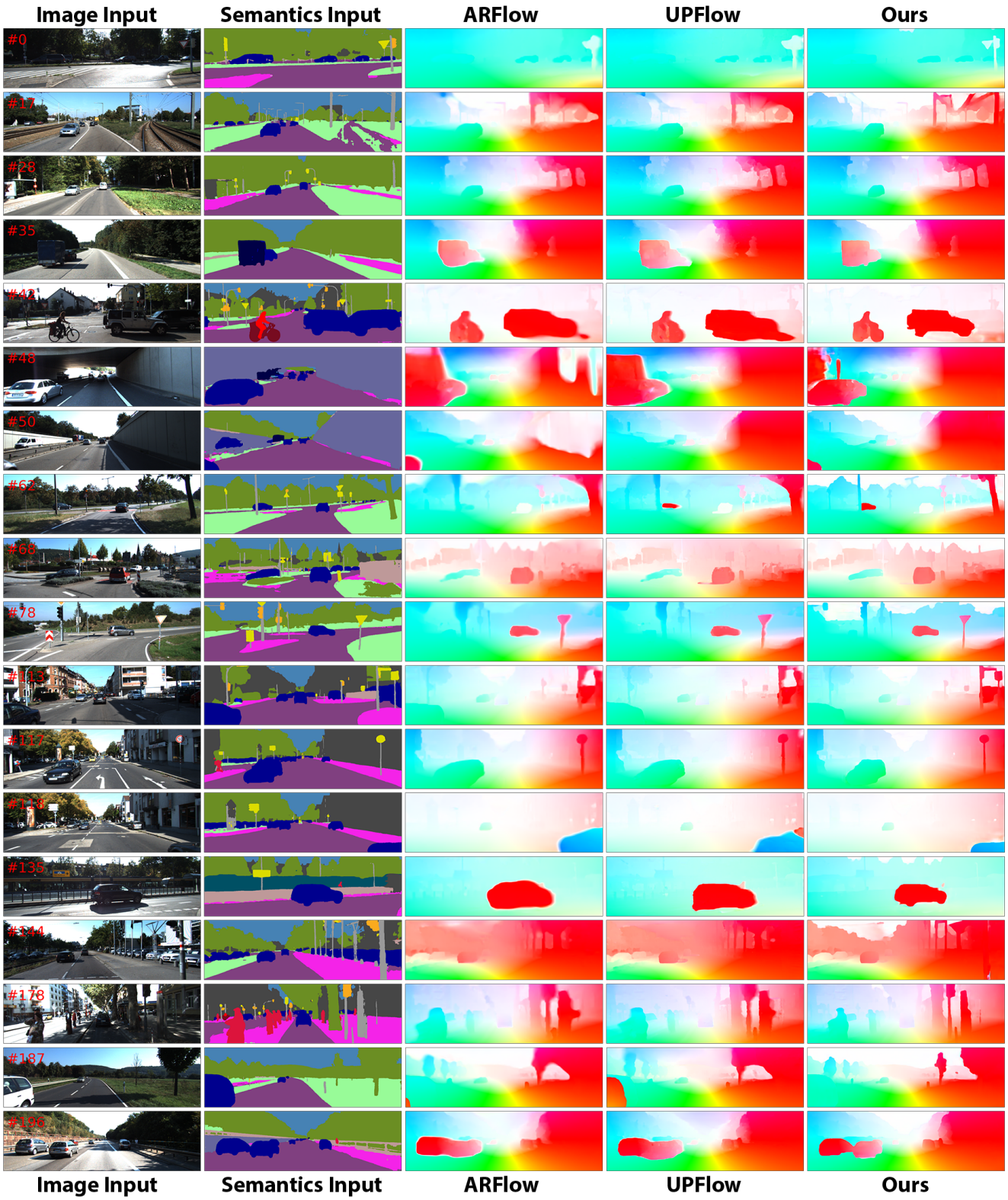


Figure 3. More qualitative results from the KITTI-2015 test set. Sample IDs are shown on the top left corners of the images.

C. Other explorations

We have also explored many other methods to apply semantics in the unsupervised optical flow network. Although these trials are not very successful and thus are not proposed in our final model, we briefly discuss our findings for the readers' reference.

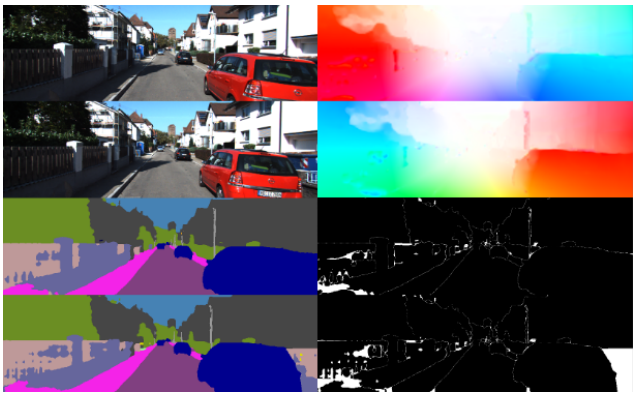


Figure 4. Left: two input frames and the semanic inputs; top right: forward and backward flow; bottom right: forward and backward semantic consistency loss

C.1. Adding semantic inputs to the decoder

We also tried adding semantic inputs to the decoder. This is because the current semantic maps are used as encoder inputs, which are somewhat distant from our final output, so we were wondering whether adding semantic cues to the decoder directly could help it decode a better flow.

We used two shallow convolutional layers to extract a feature map from semantic input and downsample that feature to different resolutions for different levels of the iterative decoder. We found that such direct injection of semantic input into the decoder did improve the vanilla ARFlow without semantic encoder. However, it helped little for our adapted ARFlow model *with* semantic encoder.

C.2. Adding a semantic consistency loss

As applied in some previous work [1], the semantic consistency loss enforces the output correspondence to have consistent semantic classes. This is similar to the photometric loss, but we use the output flow to warp the semantic inputs instead of the image inputs.

After experimenting this semantic consistency loss, unfortunately, we did not find it very helpful to our network. There are mainly two reasons.

Firstly, unlike the image input, which consists of roughly continuous RGB values, the semantic input is categorical, and it is common to have large areas of the same semantic class. Thus, one may understand the semantic map as a highly texture-less channel of input, which cannot help differentiate different regions of the same semantic class. Also, since a large area has the same semantic class, tuning flow in that region makes no difference to the semantic consistency loss, which means the gradient will be zero for most pixels except for those near the semantic boundaries.

One solution to the aforementioned problem is to use a continuous semantic class distribution as input. For example, we can use the softmax values from the semantic segmentation network as our semantic input. However, as we mentioned earlier in Appendix B.3, doing augmentations on a 19-channel semantic input is very time-consuming.

Secondly, semantic consistency loss also does not work on occlusion regions, where photometric loss has issues, so we have to mask out the occlusion regions for both losses. For non-occluded regions, the current photometric loss is already good enough to find semantically consistent output by itself, so semantic consistency does not add much here. As illustrated in Fig. 4, we trained a model with only photometric loss for only 50k iterations, and most part of the frame is already very consistent on semantics. The inconsistent parts are mostly either on semantic boundaries or in the occlusion region.

C.3. Using semantic boundaries for the boundary-aware smoothness loss

Most previous methods use smoothness loss to constrain a smooth flow output. However, motion is not smooth across motion boundaries, where motion changes abruptly. Motion boundaries usually coincide with object boundaries, so object boundaries can be a good approximation to indicate where smoothness loss should not be imposed.

Due to lack of semantic information, current methods use image edges instead to generate a weight map to reweigh smoothness loss at different pixels. In our case, since semantic maps are available, we use this information to create much clearer object boundaries.

We start from the same smoothness loss in ARFlow [3]. As visualized in Fig. 5, the weights based on image edges are computed as the sum of the second-order image derivatives on both x and y-axis, *i.e.*, the Laplacian of the 2D optical flow field. In comparison, our semantic boundaries are much cleaner.

However, both image edges and semantic boundaries have issues. Image edges usually provide boundaries within the same object, such as the edge of the shadow on the road, and they also have fewer boundaries in the dark region where image values are similar. Meanwhile, semantic boundaries are computed from semantic segmentation, where different instances of the same semantic class are not differentiated. This causes big issues when, for example, the semantic map of multiple cars merge into one.

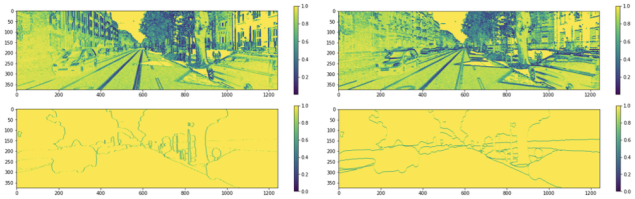


Figure 5. Top: weights based on image edges; bottom: weights based on semantic boundaries

To fix these issues, we attempted to use a combination of both image edges and semantic boundaries. We find image edges in the vehicle (car, truck, bus, train), people (person, rider), and small vehicle (motorcycle, bicycle) regions, combined with semantic boundaries else where, as shown in Fig. 6.

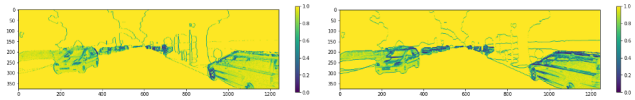


Figure 6. The combined boundary weight

We tried all these boundaries and tuned the weight of the smoothness loss term in the total loss, and found little difference in the evaluated results. Moreover, after we add the learned upampler module in the network, applying smoothness loss is doing more harm than help. This is understandable because the goal of the learned upsampler is to make the network decide which part of the flow output should be smooth and where should not be smooth. Smoothness loss imposes a preference that the motion should be smooth in the form of zero second-order derivatives, which is not data-driven and may not be precise for real-world motion fields. Therefore, we ended up turning off the smoothness loss in our final model.

C.4. Learning the initial flow in the decoder

Driving scenes usually have similar scene layouts (sky is on the top, and road is on the bottom, *etc*.). In addition, the motion of the camera is also mostly moving forward with some occasional slight rotations. These two effects together create a *looming* motion, where most objects are moving closer to the camera. To explain in 2D image frames, the left part of the image tends to move left, and the right part tends to move right. The lower part (mostly roads and sidewalks) also tends to move downwards until they are out of sight. These observations indicate a strong motion prior knowledge that can be used to better initialize our flow estimate.

The current iterative decoder as in ARFlow [3] initialize the flow estimate as zero motion, which gets refined later iteration by iteration. However, we can apply the looming motion prior instead by parameterizing the initial flow using some learnable parameters. Specifically, for a 256×832 input, the highest (6th) level feature has dimension 4×13, so we use a learnable $19 \times 4 \times 13 \times 2$ tensor as the prior. We condition the motion prior on the 19 semantic classes, so that we can refer to the semantic map input to generate its initial flow prior based on semantics. Note that we define the prior specifically forward flow only, and we need to flip the prior when computing backward flow in our network.

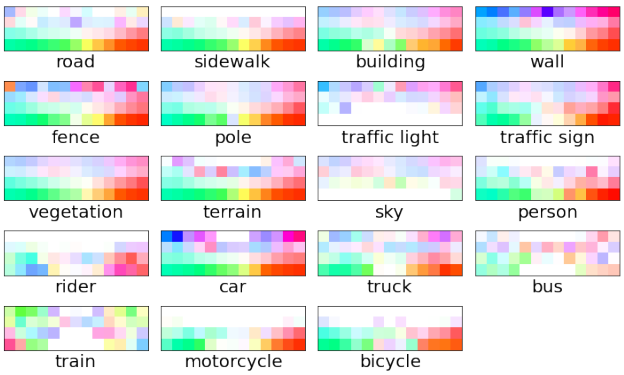


Figure 7. The learned init flow prior for each semantic class

We trained the network with learned initial flow conditioned on semantics, and the learned prior is visualized in Fig. 7. Overall, we can see that the looming motion pattern is learned by our network. However, for each semantic class, the network can only learn prior for places where that class frequently appears. For instance, the upper part of the road prior is not learned well because there are few road pixels on the top of the frame.

To better fix this issue, we try to parameterize the motion prior by its four corners. Specifically, we learn a 2×2 prior for each class, and bilinear interpolate this 2×2 prior to 4×13 before using it to construct initial flow based on semantics. The results are then visualized in Fig. 8. The network has learned a very smooth and more or less similar pattern prior for each class.

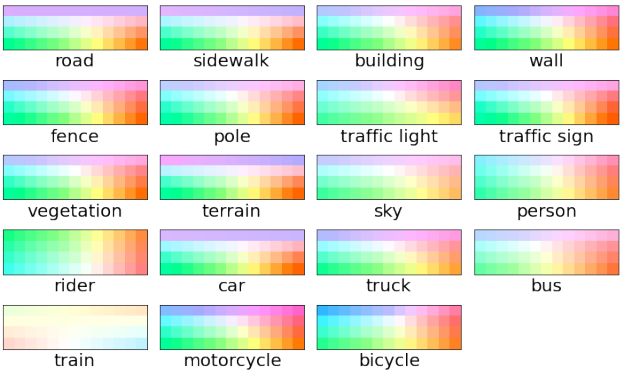


Figure 8. The learned init flow prior for each semantic class if we parameterize by coners

In terms of evaluation metrics, both methods improve slightly by themselves, but we found the improvements became negligible after we apply the semantic segmentation module. One question here is whether good initialization matters a lot for our unsupervised flow networks. Since we refine the flow by many iterations in the decoder, the initial estimate may not change the results significantly, if the following refinement units are effective enough.

C.5. Reweighing losses at different semantic regions

Current models usually have different scales of error at different semantic regions. Semantic classes such as car are harder to track and thus incur larger errors than other classes. Also, some classes, such as car and person, are practically more important for autonomous driving applications. Therefore, we tried to reweigh the photometric loss by their semantic class. We give higher weights to classes like car and person so that the network can focus on improving those classes more.

However, the results are hard to evaluate numerically. The current ground-truth labels are mostly concentrated on the lower part of the frame, so many practically important objects such as traffic lights, traffic signs and poles only account for a very small amount of the evaluation. Also, classes like person, rider, and bicycles do not have flow labels because reliable CAD models are not available for these dynamic objects. Based on these issues, finding a better evaluation method may be more important.

C.6. Using the epipolar constraint to post-process the static region flow

Following earlier traditional methods [2, 4], we also explored the possibility of using the static scene epipolar constraint to post-process static optical flow. Since most part of the frame is static, we can use our correspondences found to estimate the fundamental matrix between two frames, and then use epipolar constraints to refine our flow.

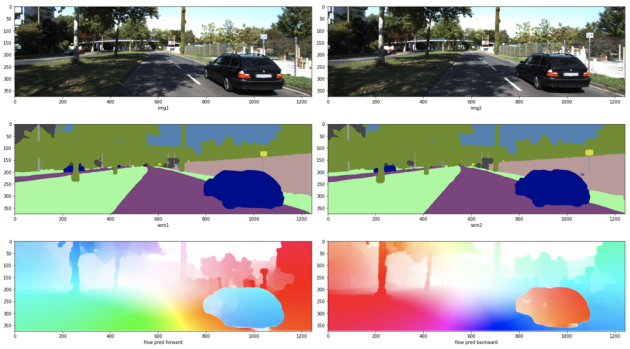
This method is mostly targeted on refining optical flow for those occluded static pixels. For example, a part of the road or background may be occluded by the moving cars, or they may simply move out of the frame, so their correspondence is not visible in the other frame. However, we still want the network to have a best "guess" on where the correspondence is. Most current methods rely on smoothness to generate those "guesses". However, given the epipolar constraint, we limit the searching range of correspondence to one epipolar line, which may help us "guess" more informatively. Our semantic map inputs tell us where those static region is, so we can get a more reliable fundamental matrix estimate.

Estimating the fundamental matrix only requires eight pairs of corresponding points, so we can select only the most reliable correspondences for this computation. Specifically, we define "reliable correspondence" based on three criteria: (1) not in the occlusion region, (2) not on the semantic boundary or image borders, and (3) not from any (possibly) dynamic semantic class or any texture-less semantic class.

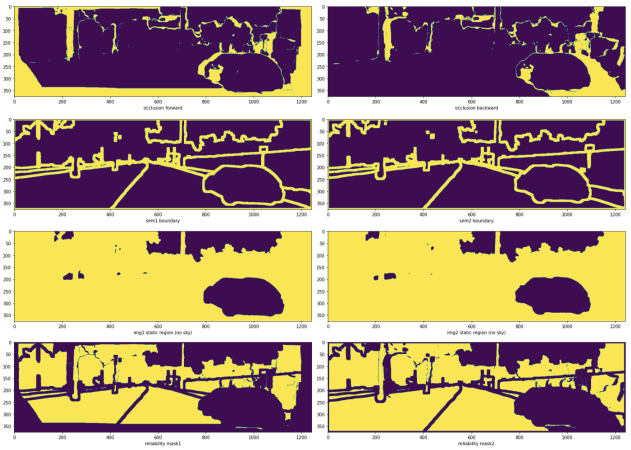
We first compute the occlusion mask through forward-backward consistency check. Then, we find semantic boundaries based on our semantic map input and define all pixels that are ≤ 5 pixels away from any boundary as the boundary pixels. For reliable static semantic classes, we include all classes except vehicles (car, truck, bus, train), people (person, rider), small vehicles (bicycle, motorcycle), and sky (poor texture). One example is shown in Figs. 9(a) and 9(b).

After computing the reliable static regions, we use both forward and backward flow in those regions to create a set of correspondences between two input frames. We then estimate the fundamental matrix $\hat{F}$ using RANSAC. For a typical sample in the KITTI-2015 train set, RANSAC based on correspondeces in our reliable regions mostly produces $>$98% inlier rate.

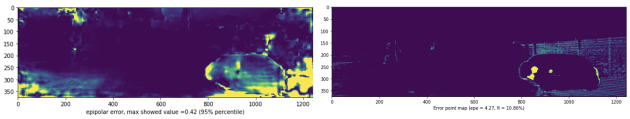
We then find the correspondences in static regions and



(a) Inputs for post-processing



(b) Computing reliable static region masks



(c) Comparing epipolar errors and true errors

Figure 9. An example for semantic post-processing

check whether they conform to epipolar constraint. For each point $\boldsymbol{p}$ in the static region of the first frame, we compute the epipolar error

$$\epsilon(\boldsymbol{p}) = \left((\boldsymbol{p} + U_{1\to 2}(\boldsymbol{p}))^T F \boldsymbol{p}\right)^2,$$

where $U_{1\to 2}(\boldsymbol{p})$ is the input flow estimate. We find the pixels with top 5% epipolar error and refine those by projecting their current estimated correspondences onto their epipolar lines. As shown in Fig. 9(c), our epipolar error successfully detects the part of the frame that has high EPE errors. However, we tested this method on the KITTI-2015 train set but did not see much improvements on the evaluation results. The reason is still under investigation.

References

[1] Min Bai, Wenjie Luo, Kaustav Kundu, and Raquel Urtasun. Exploiting semantic information and deep matching for optical flow. In *Proceedings of the European Conference on Computer Vision*, pages 154–170. Springer, 2016. 5

[2] Junhwa Hur and Stefan Roth. Joint optical flow and temporally consistent semantic segmentation. In *Proceedings of the European Conference on Computer Vision*, pages 163–177. Springer, 2016. 7

[3] Liang Liu, Jiangning Zhang, Ruifei He, Yong Liu, Yabiao Wang, Ying Tai, Donghao Luo, Chengjie Wang, Jilin Li, and Feiyue Huang. Learning by analogy: Reliable supervision from transformations for unsupervised optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6489–6498, 2020. 3, 6

[4] Jonas Wulff, Laura Sevilla-Lara, and Michael J Black. Optical flow in mostly rigid scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4671–4680, 2017. 7