Supplementary Material for "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks"

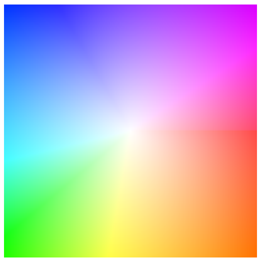

Figure 1. Flow field color coding used in this paper. The displacement of every pixel in this illustration is the vector from the center of the square to this pixel. The central pixel does not move. The value is scaled differently for different images to best visualize the most interesting range.

1. Video

Please see the supplementary video for FlowNet2 results on a number of diverse video sequences, a comparison between FlowNet2 and state-of-the-art methods, and an illustration of the speed/accuracy trade-off of the FlowNet 2.0 family of models.

Optical flow color coding. For optical flow visualization we use the color coding of Butler *et al.* [3]. The color coding scheme is illustrated in Figure 1. Hue represents the direction of the displacement vector, while the intensity of the color represents its magnitude. White color corresponds to no motion. Because the range of motions is very different in different image sequences, we scale the flow fields before visualization: independently for each image pair shown in figures, and independently for each video fragment in the supplementary video. Scaling is always the same for all methods being compared.

2. Dataset Schedules: KITTI2015 Results

In Table 1 we show more results of training networks with the original FlowNet schedule S_{short} [4] and the new FlowNet2 schedules S_{long} and S_{fine}. We provide the endpoint error when testing on the KITTI2015 train dataset. Table 1 in the main paper shows the performance of the same networks on Sintel. One can observe that on KITTI2015, as well as on Sintel, training with $S_{long} + S_{fine}$ on the combination of Chairs and Things3D works best (in the paper referred to as Chairs→Things3D schedule).

Architecture	Datasets	S_{short}	S_{long}	S_{fine}
FlowNetS	Chairs	15.58	-	-
	Chairs	-	14.60	14.28
	Things3D	-	16.01	16.10
	mixed	-	16.69	15.57
	Chairs→Things3D	-	14.60	**14.18**
FlowNetC	Chairs	13.41	-	-
	Chairs→Things3D	-	12.48	**11.36**

Table 1. Results of training FlowNets with different schedules on different datasets (one network per row). Numbers indicate endpoint errors on the KITTI2015 training dataset.

3. Recurrently Stacking Networks with the Same Weights

The bootstrap network differs from the succeeding networks by its task (it needs to predict a flow field from scratch) and inputs (it does not get a previous flow estimate and a warped image). The network after the bootstrap network only refines the previous flow estimate, so it can be applied to its own output recursively. We took the best network from Table 2 of the main paper and applied Net2 recursively multiple times. We then continued training the whole stack with multiple Net2. The difference from our final FlowNet2 architecture is that here the weights are shared between the stacked networks, similar to a standard recurrent network. Results are given in Table 2. In all cases we observe no or negligible improvements compared to the baseline network with a single Net2.

4. Small Displacements

4.1. The ChairsSDHom Dataset

As an example of real-world data we examine the UCF101 dataset [9]. We compute optical flow using LDOF [4] and compare the flow magnitude distribution to the synthetic datasets we use for training and benchmarking, this is shown in Figure 3. While Chairs are similar to Sintel, UCF101 is fundamentally different and contains much more small displacments.

	Training of Net2 enabled	Warping gradient enabled	EPE
Net1 + 1×Net2	✗	–	**2.93**
Net1 + 2×Net2	✗	–	2.95
Net1 + 3×Net2	✗	–	3.04
Net1 + 3×Net2	✓	✗	**2.85**
Net1 + 3×Net2	✓	✓	2.85

Table 2. Stacked architectures using shared weights. The combination in the first row corresponds to the best results of Table 2 from the paper. Just applying the second network multiple times does not yield improvements. In the two bottom rows we show the results of fine-tuning the stack of the top networks on Chairs for 100k more iterations. This leads to a minor improvement of performance.

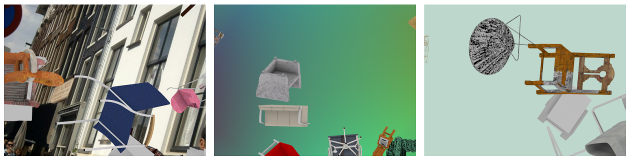
Figure 2. Images from the ChairsSDHom (Chairs Small Displacement Homogeneous) dataset.

To create a training dataset similar to UCF101, following [4], we generated our ChairsSDHom (Chairs Small Displacement Homogeneous) dataset by randomly placing and moving chairs in front of randomized background images. However, we also followed Mayer *et al.* [6] in that our chairs are not flat 2D bitmaps as in [4], but rendered 3D objects. Similar to Mayer *et al.*, we rendered our data first in a "raw" version to get blend-free flow boundaries and then a second time with antialiasing to obtain the color images. To match the characteristic contents of the UCF101 dataset, we mostly applied small motions. We added scenes with weakly textured background to the dataset, being monochrome or containing a very subtle color gradient. Such monotonous backgrounds are not unusual in natural videos, but almost never appear in Chairs or Things3D. A featureless background can potentially move in any direction (an extreme case of the aperture problem), so we kept these background images fixed to introduce a meaningful prior into the dataset. Example images from the dataset are shown in Figure 2.

4.2. Fine-Tuning FlowNet2-CSS-ft-sd

With the new ChairsSDHom dataset we fine-tuned our FlowNet2-CSS network for smaller displacements (we denote this by *FlowNet2-CSS-ft-sd*). We experimented with different configurations to avoid sacrificing performance on large displacements. We found the best performance can be achieved by training with mini-batches of 8 samples: 2 from Things3D and 6 from ChairsSDHom. Furthermore, we applied a nonlinearity of $x^{0.4}$ to the endpoint error to emphasize the small-magnitude flows.

Name	Kernel	Str.	Ch I/O	In Res	Out Res	Input
conv0	3×3	1	6/64	512×384	512×384	Images
conv1	3×3	2	64/64	512×384	256×192	conv0
conv1_1	3×3	1	64/128	256×192	256×192	conv1
conv2	3×3	2	128/128	256×192	128×96	conv1_1
conv2_1	3×3	1	128/128	128×96	128×96	conv2
conv3	3×3	2	128/256	128×96	64×48	conv2_1
conv3_1	3×3	1	256/256	64×48	64×48	conv3
conv4	3×3	2	256/512	64×48	32×24	conv3_1
conv4_1	3×3	1	512/512	32×24	32×24	conv4
conv5	3×3	2	512/512	32×24	16×12	conv4_1
conv5_1	3×3	1	512/512	16×12	16×12	conv5
conv6	3×3	2	512/1024	16×12	8×6	conv5_1
conv6_1	3×3	1	1024/1024	8×6	8×6	conv6
pr6+loss6	3×3	1	1024/2	8×6	8×6	conv6_1
upconv5	4×4	2	1024/512	8×6	16×12	conv6_1
rconv5	3×3	1	1026/512	16×12	16×12	upconv5+pr6+conv5_1
pr5+loss5	3×3	1	512/2	16×12	16×12	rconv5
upconv4	4×4	2	512/256	16×12	32×24	rconv5
rconv4	3×3	1	770/256	32×24	32×24	upconv4+pr5+conv4_1
pr4+loss4	3×3	1	256/2	32×24	32×24	rconv4
upconv3	4×4	2	256/128	32×24	64×48	rconv4
rconv3	3×3	1	386/128	64×48	64×48	upconv3+pr4+conv3_1
pr3+loss3	3×3	1	128/2	64×48	64×48	rconv3
upconv2	4×4	2	128/64	64×48	128×96	rconv3
rconv2	3×3	1	194/64	128×96	128×96	upconv2+pr3+conv2_1
pr2+loss2	3×3	1	64/2	128×96	128×96	rconv2

Table 3. The details of the FlowNet2-SD architecture.

Name	Kernel	Str.	Ch I/O	In Res	Out Res	Input
conv0	3×3	1	6/64	512×384	512×384	Img1+flows+mags+errs
conv1	3×3	2	64/64	512×384	256×192	conv0
conv1_1	3×3	1	64/128	256×192	256×192	conv1
conv2	3×3	2	128/128	256×192	128×96	conv1_1
conv2_1	3×3	1	128/128	128×96	128×96	conv2
pr2+loss2	3×3	1	128/2	128×96	128×96	conv2_1
upconv1	4×4	2	128/32	128×96	256×192	conv2_1
rconv1	3×3	1	162/32	256×192	256×192	upconv1+pr2+conv1_1
pr1+loss1	3×3	1	32/2	256×192	256×192	rconv1
upconv0	4×4	2	32/16	256×192	512×384	rconv1
rconv0	3×3	1	82/16	512×384	512×384	upconv0+pr1+conv0
pr0+loss0	3×3	1	16/2	512×384	512×384	rconv0

Table 4. The details of the FlowNet2 fusion network architecture.

4.3. Network Architectures

The architectures of the small displacement network and the fusion network are shown in Tables 3 and 4. The input to the small displacement network is formed by concatenating both RGB images, resulting in 6 input channels. The network is in general similar to FlowNetS. Differences are the smaller strides and smaller kernel sizes in the beginning and the convolutions between the upconvolutions.

The fusion network is trained to merge the flow estimates of two previously trained networks, and this task dictates the input structure. We feed the following data into the network: the first image from the image pair, two estimated flow fields, their magnitudes, and finally the two squared Euclidean photoconsistency errors, that is, per-pixel squared Euclidean distance between the first image and the second image warped with the predicted flow field. This sums up to 11 channels. Note that we do not input the second image directly. All inputs are at full image resolution, flow field estimates from previous networks are upsampled with nearest neighbor upsampling.

"

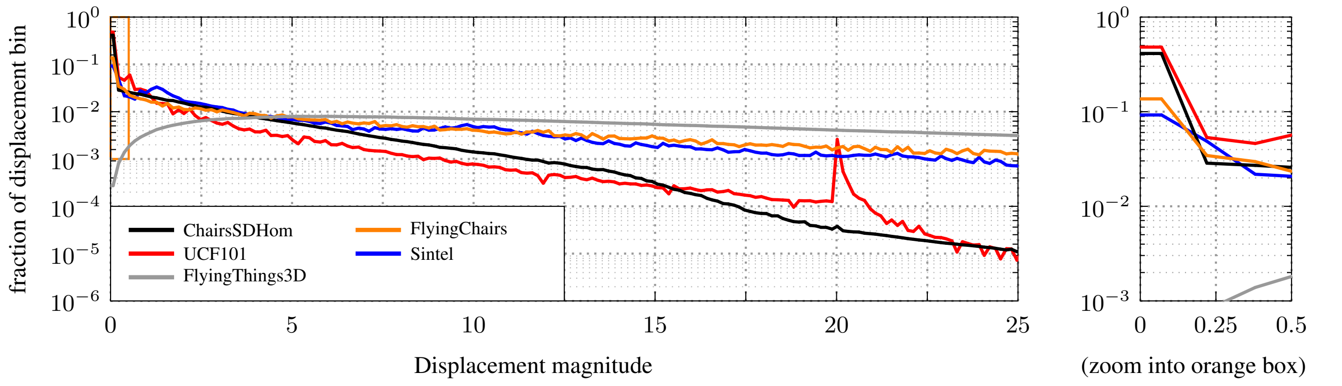

Figure 3. **Left:** histogram of displacement magnitudes of different datasets. y-axis is logarithmic. **Right:** zoomed view for very small displacements. The Chairs dataset very closely follows the Sintel dataset, while our ChairsSDHom datasets is close to UCF101. Things3D has few small displacements and for larger displacements also follows Sintel and Chairs. The Things3D histogram appears smoother because it contains more raw pixel data and due to its randomization of 6-DOF camera motion.

5. Evaluation

5.1. Intermediate Results in Stacked Networks

The idea of the stacked network architecture is that the estimated flow field is gradually improved by every network in the stack. This improvement has been quantitatively shown in the paper. Here, we additionally show qualitative examples which clearly highlight this effect. The improvement is especially dramatic for small displacements, as illustrated in Figure 4. The initial prediction of FlowNet2-C is very noisy, but is then significantly refined by the two succeeding networks. The FlowNet2-SD network, specifically trained on small displacements, estimates small displacements well even without additional refinement. Best results are obtained by fusing both estimated flow fields. Figure 5 illustrates this for a large displacement case.

5.2. Speed and Performance on KITTI2012

Figure 6 shows runtime vs. endpoint error comparisons of various optical flow estimation methods on two datasets: Sintel (also shown in the main paper) and KITTI2012. In both cases models of the FlowNet 2.0 family offer an excellent speed/accuracy trade-off. Networks fine-tuned on KITTI are not shown. The corresponding points would be below the lower border of the KITTI2012 plot.

5.3. Motion Segmentation

Table 5 shows detailed results on motion segmentation obtained using the algorithms from [7, 5] with flow fields from different methods as input. For FlowNetS the algorithm does not fully converge after one week on the training set. Due to the bad flow estimations of FlowNetS [4], only very short trajectories can be computed (on average about 3 frames), yielding an excessive number of trajectories. Therefore we do not evaluate FlowNetS on the test set. On all metrics, FlowNet2 is at least on par with the best optical flow estimation methods and on the VI (variation of information) metric it is even significantly better.

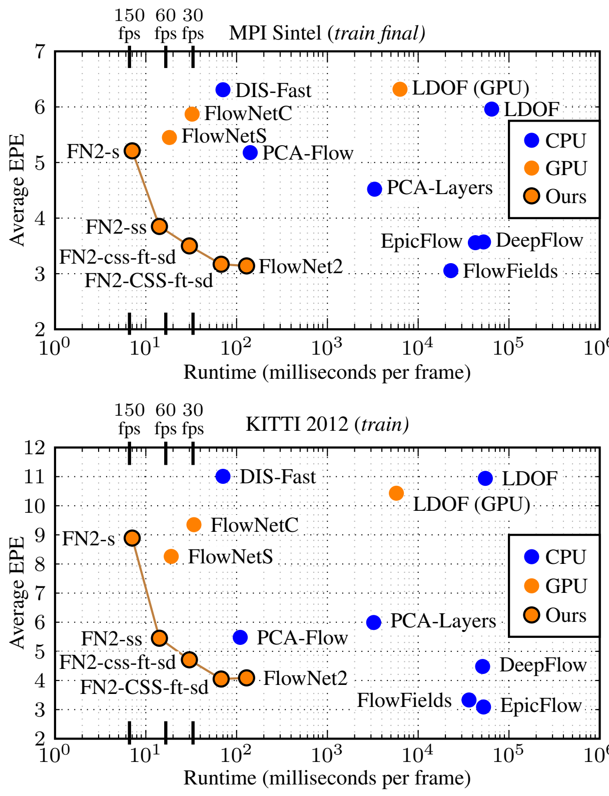

Figure 6. Runtime vs. endpoint error comparison to the fastest existing methods with available code. The FlowNet2 family outperforms other methods by a large margin.

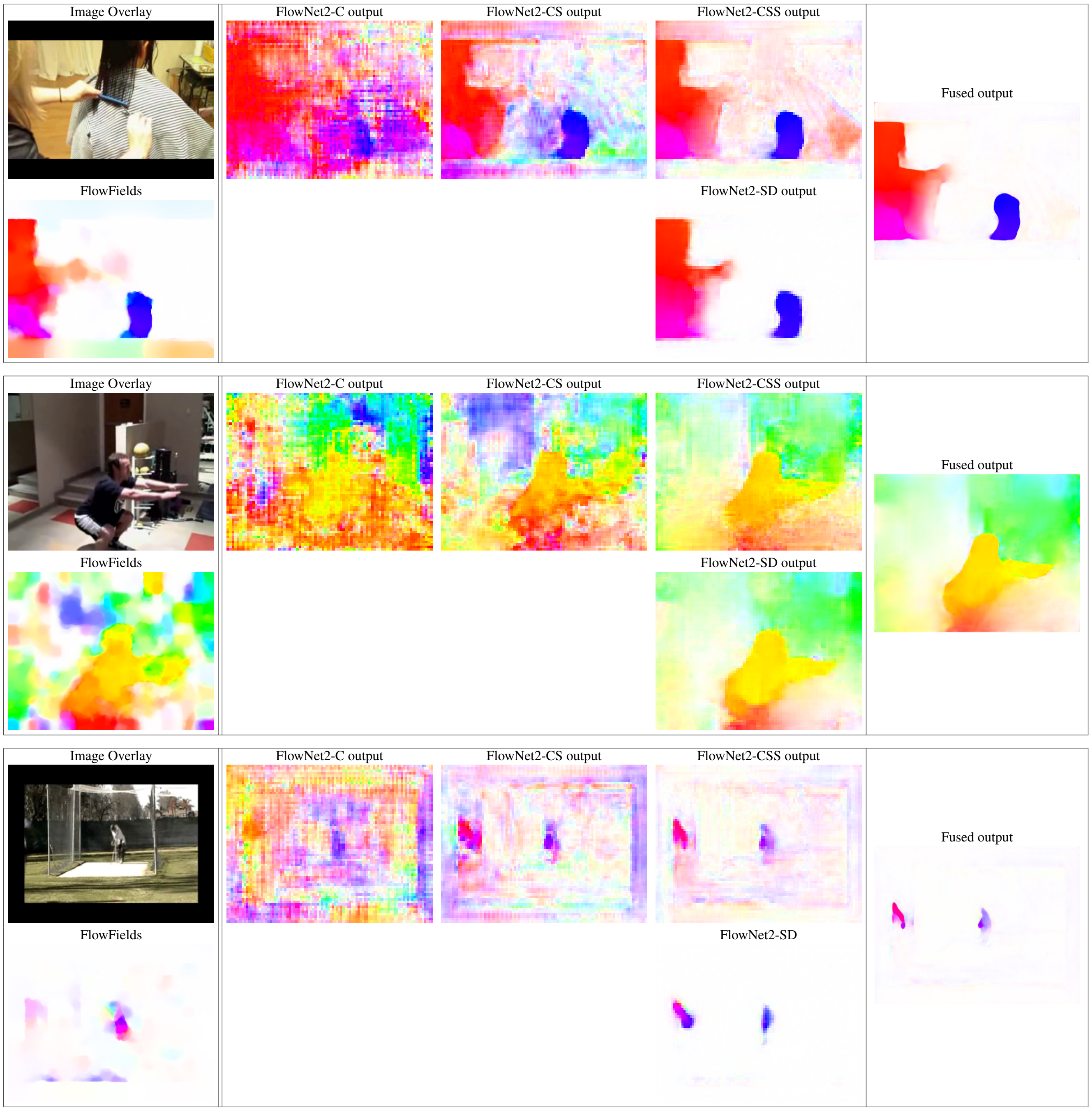

Figure 4. Three examples of iterative flow field refinement and fusion for small displacements. The motion is very small (therefore mostly not visible in the image overlays). One can observe that FlowNet2-SD output is smoother than FlowNet2-CSS output. The fusion correctly uses the FlowNet2-SD output in the areas where FlowNet2-CSS produces noise due to small displacements.

5.4. Qualitative results on KITTI2015

Figure 7 shows qualitative results on the KITTI2015 dataset. FlowNet2-kitti has not been trained on these images during fine-tuning. KITTI ground truth is sparse, for better visualization we interpolated the ground truth bilinearly. FlowNet2-kitti significantly outperforms competing approaches both quantitatively and qualitatively.

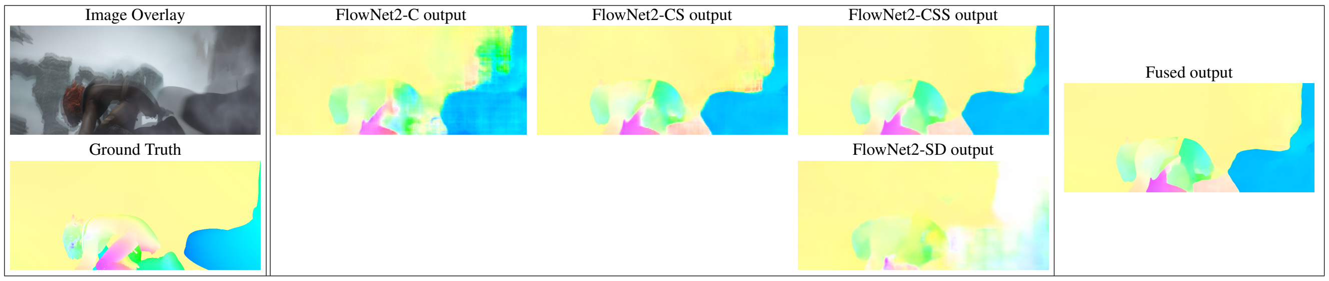

Figure 5. Iterative flow field refinement and fusion for large displacements. The large displacements branch correctly estimates the large motions; the stacked networks improve the flow field and make it smoother. The small displacement branch cannot capture the large motions and the fusion network correctly chooses to use the output of the large displacement branch.

Method	**Training set** (29 sequences)						**Test set** (30 sequences)					
	D	P	R	F	VI	O	D	P	R	F	VI	O
LDOF (CPU) [2]	0.81%	86.73%	73.08%	79.32%	0.267	31/65	0.87%	87.88%	67.70%	76.48%	0.366	25/69
DeepFlow [11]	**0.86%**	88.96%	**76.56%**	**82.29%**	0.296	**33/65**	0.89%	**88.20%**	69.39%	**77.67%**	0.367	26/69
EpicFlow [8]	0.84%	87.21%	74.53%	80.37%	0.279	30/65	**0.90%**	85.69%	69.09%	76.50%	0.373	25/69
FlowFields [1]	0.83%	87.19%	74.33%	80.25%	0.282	31/65	0.89%	86.88%	**69.74%**	77.37%	0.365	**27/69**
FlowNetS [4]	0.45%	74.84%	45.81%	56.83%	0.604	3/65	0.48%	68.05%	41.73%	51.74%	0.60	3/69
FlowNet2-css-ft-sd	0.78%	88.07%	71.81%	79.12%	0.270	28/65	0.81%	83.76%	65.77%	73.68%	0.394	24/69
FlowNet2-CSS-ft-sd	0.79%	87.57%	73.87%	80.14%	0.255	31/65	0.85%	85.36%	68.81%	76.19%	0.327	26/69
FlowNet2	0.80%	**89.63%**	73.38%	80.69%	**0.238**	29/65	0.85%	86.73%	68.77%	76.71%	**0.311**	26/69
LDOF (CPU) [2]	3.47%	86.79%	73.36%	79.51%	0.270	28/65	3.72%	86.81%	67.96%	76.24%	0.361	25/69
DeepFlow [11]	**3.66%**	86.69%	**74.58%**	**80.18%**	0.303	29/65	3.79%	88.58%	68.46%	**77.23%**	0.393	**27/69**
EpicFlow [8]	3.58%	84.47%	73.08%	78.36%	0.289	27/65	**3.83%**	86.38%	**70.31%**	77.52%	0.343	**27/69**
FlowFields [1]	3.55%	87.05%	73.50%	79.70%	0.293	30/65	3.82%	88.04%	68.44%	77.01%	0.397	24/69
FlowNetS [4]*	1.93%	76.60%	45.23%	56.87%	0.680	3/62	–	–	–	–	–	–/69
FlowNet2-css-ft-sd	3.38%	85.82%	71.29%	77.88%	0.297	26/65	3.53%	84.24%	65.49%	73.69%	0.369	25/69
FlowNet2-CSS-ft-sd	3.41%	86.54%	73.54%	79.52%	0.279	30/65	3.68%	85.58%	67.81%	75.66%	0.339	**27/69**
FlowNet2	3.41%	**87.42%**	73.60%	79.92%	**0.249**	**32/65**	3.66%	**87.16%**	68.51%	76.72%	**0.324**	26/69

Table 5. Results on the FBMS-59 [10, 7] dataset on training (**left**) and test set (**right**). Best results are highlighted in bold. **Top**: low trajectory density (8px distance), **bottom**: high trajectory density (4px distance). We report **D**: density (depending on the selected trajectory sparseness), **P**: average precision, **R**: average recall, **F**: F-measure, **VI**: variation of information (lower is better), and **O**: extracted objects with F $\geq 75\%$. (*) FlownetS is evaluated on 28 out of 29 sequences. On the sequence *lion02*, the optimization did not converge after one week. Due to the convergence problems we do not evaluate FlowNetS on the test set.

6. Warping Layer

The following two sections give the mathematical details of forward and backward passes through the warping layer used to stack networks.

6.1. Definitions and Bilinear Interpolation

Let the image coordinates be $\mathbf{x} = (x, y)^\top$ and the set of valid image coordinates R. Let $\mathbf{I}(\mathbf{x})$ denote the image and $\mathbf{w}(\mathbf{x}) = (u(\mathbf{x}), v(\mathbf{x}))^\top$ the flow field. The image can also be a feature map and have arbitrarily many channels. Let channel c be denoted with $I_c(\mathbf{x})$. We define the coefficients:

$$\begin{aligned} \theta_x &= x - \lfloor x \rfloor, & \overline{\theta_x} &= 1 - \theta_x, \\ \theta_y &= y - \lfloor y \rfloor, & \overline{\theta_y} &= 1 - \theta_y \end{aligned} \tag{1}$$

and compute a continuous version $\tilde{\mathbf{I}}$ of $\mathbf{I}$ using bilinear interpolation in the usual way:

$$\begin{aligned} \tilde{\mathbf{I}}(x, y) = \quad & \overline{\theta_x}\,\overline{\theta_y}\mathbf{I}(\lfloor x \rfloor, \lfloor y \rfloor) \\ +\, & \theta_x \overline{\theta_y}\mathbf{I}(\lceil x \rceil, \lfloor y \rfloor) \\ +\, & \overline{\theta_x}\theta_y \mathbf{I}(\lfloor x \rfloor, \lceil y \rceil) \\ +\, & \theta_x \theta_y \mathbf{I}(\lceil x \rceil, \lceil y \rceil) \end{aligned} \tag{2}$$

6.2. Forward Pass

During the forward pass, we compute the warped image by following the flow vectors. We define all pixels to be zero where the flow points outside of the image:

$$\mathbf{J}_{\mathbf{I},\mathbf{w}}(\mathbf{x}) = \begin{cases} \tilde{\mathbf{I}}(\mathbf{x} + \mathbf{w}(\mathbf{x})) & \text{if } \mathbf{x} + \mathbf{w}(\mathbf{x}) \text{ is in } R, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

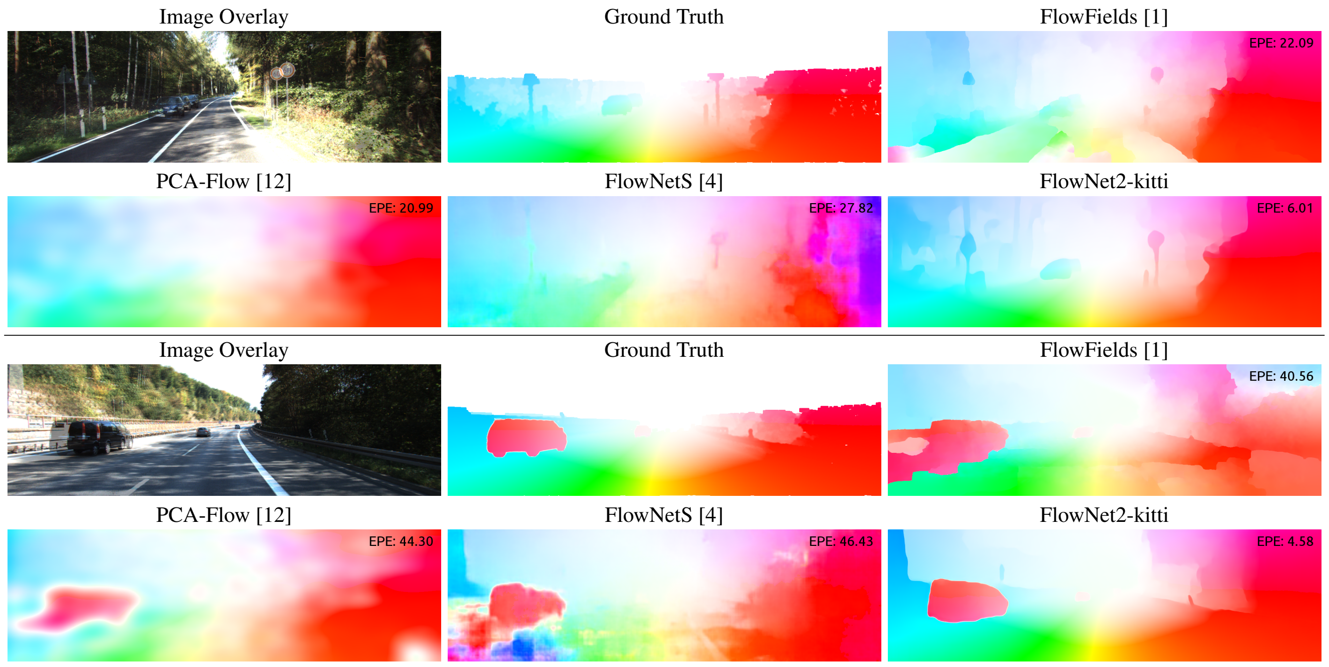

Figure 7. Qualitative results on the KITTI2015 dataset. Flow fields produced by FlowNet2-kitti are significantly more accurate, detailed and smooth than results of all other methods. Sparse ground truth has been interpolated for better visualization (note that this can cause blurry edges in the ground truth).

6.3. Backward Pass

During the backward pass, we need to compute the derivative of $\mathbf{J}_{\mathbf{I},\mathbf{w}}(\mathbf{x})$ with respect to its inputs $\mathbf{I}(\mathbf{x}')$ and $\mathbf{w}(\mathbf{x}')$, where $\mathbf{x}$ and $\mathbf{x}'$ are different integer image locations. Let $\delta(b) = 1$ if b is true and 0 otherwise, and let $\mathbf{x} + \mathbf{w}(\mathbf{x}) = (p(\mathbf{x}), q(\mathbf{x}))^\top$. For brevity, we omit the dependence of p and q on $\mathbf{x}$. The derivative with respect to $I_c(\mathbf{x}')$ is then computed as follows:

$$\begin{aligned}
\frac{\partial J_c(\mathbf{x})}{\partial I_c(\mathbf{x}')} &= \frac{\partial \tilde{I}_c(\mathbf{x}+\mathbf{w}(\mathbf{x}))}{\partial I_c(\mathbf{x}')} \\
&= \frac{\partial \tilde{I}_c(p,q)}{\partial I_c(x',y')} \\
&= \quad \overline{\theta_{x'}}\,\overline{\theta_{y'}}\delta(\lfloor p\rfloor = x')\delta(\lfloor q\rfloor = y') \\
&\quad + \theta_{x'}\overline{\theta_{y'}}\delta(\lceil p\rceil = x')\delta(\lfloor q\rfloor = y') \\
&\quad + \overline{\theta_{x'}}\theta_{y'}\delta(\lfloor p\rfloor = x')\delta(\lceil q\rceil = y') \\
&\quad + \theta_{x'}\theta_{y'}\delta(\lceil p\rceil = x')\delta(\lceil q\rceil = y'). \qquad (4)
\end{aligned}$$

The derivative with respect to the first component of the flow $u(\mathbf{x})$ is computed as follows:

$$\frac{\partial \mathbf{J}(\mathbf{x})}{\partial u(\mathbf{x}')} = \begin{cases} 0 & \text{if } \mathbf{x} \neq \mathbf{x}' \text{ or } (p,q)^\top \notin R \\ \frac{\partial \tilde{\mathbf{I}}(\mathbf{x}+\mathbf{w}(\mathbf{x}))}{\partial u(\mathbf{x})} & \text{otherwise.} \end{cases} \qquad (5)$$

In the non-trivial case, the derivative is computed as follows:

$$\begin{aligned}
\frac{\partial \tilde{\mathbf{I}}(\mathbf{x}+\mathbf{w}(\mathbf{x}))}{\partial u(\mathbf{x})} &= \frac{\partial \tilde{\mathbf{I}}(p,q)}{\partial u} \\
&= \frac{\partial \tilde{\mathbf{I}}(p,q)}{\partial p} \\
&= \quad \frac{\partial}{\partial p}\overline{\theta_p}\,\overline{\theta_q}\mathbf{I}(\lfloor p\rfloor, \lfloor q\rfloor) \\
&\quad + \frac{\partial}{\partial p}\theta_p\overline{\theta_q}\mathbf{I}(\lceil p\rceil, \lfloor q\rfloor) \\
&\quad + \frac{\partial}{\partial p}\overline{\theta_p}\theta_q\mathbf{I}(\lfloor p\rfloor, \lceil q\rceil) \\
&\quad + \frac{\partial}{\partial p}\theta_p\theta_q\mathbf{I}(\lceil p\rceil, \lceil q\rceil) \\
&= \quad - \overline{\theta_q}\mathbf{I}(\lfloor p\rfloor, \lfloor q\rfloor) \\
&\quad + \overline{\theta_q}\mathbf{I}(\lceil p\rceil, \lfloor q\rfloor) \\
&\quad - \theta_q\mathbf{I}(\lfloor p\rfloor, \lceil q\rceil) \\
&\quad + \theta_q\mathbf{I}(\lceil p\rceil, \lceil q\rceil). \qquad (6)
\end{aligned}$$

Note that the ceiling and floor functions ($\lceil\cdot\rceil$, $\lfloor\cdot\rfloor$) are non-differentiable at points with integer coordinates and we use directional derivatives in these cases. The derivative with respect to $v(\mathbf{x})$ is analogous.

References

[1] C. Bailer, B. Taetz, and D. Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[2] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(3):500–513, 2011.

[3] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision (ECCV)*, 2012.

[4] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[5] M. Keuper, B. Andres, and T. Brox. Motion trajectory segmentation via minimum cost multicuts. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[6] N.Mayer, E.Ilg, P.Häusser, P.Fischer, D.Cremers, A.Dosovitskiy, and T.Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] P. Ochs, J. Malik, and T. Brox. Segmentation of moving objects by long term video analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(6):1187 – 1200, Jun 2014.

[8] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv pre-print*, arXiv:1212.0402, Jan. 2013.

[10] T.Brox and J.Malik. Object segmentation by long term analysis of point trajectories. In *European Conference on Computer Vision (ECCV)*, 2010.

[11] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[12] J. Wulff and M. J. Black. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.