FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, Thomas Brox
University of Freiburg, Germany
{ilg,mayern,saikiat,keuper,dosovits,brox}@cs.uni-freiburg.de

Abstract

The FlowNet demonstrated that optical flow estimation can be cast as a learning problem. However, the state of the art with regard to the quality of the flow has still been defined by traditional methods. Particularly on small displacements and real-world data, FlowNet cannot compete with variational methods. In this paper, we advance the concept of end-to-end learning of optical flow and make it work really well. The large improvements in quality and speed are caused by three major contributions: first, we focus on the training data and show that the schedule of presenting data during training is very important. Second, we develop a stacked architecture that includes warping of the second image with intermediate optical flow. Third, we elaborate on small displacements by introducing a sub-network specializing on small motions. FlowNet 2.0 is only marginally slower than the original FlowNet but decreases the estimation error by more than 50%. It performs on par with state-of-the-art methods, while running at interactive frame rates. Moreover, we present faster variants that allow optical flow computation at up to 140fps with accuracy matching the original FlowNet.

1. Introduction

The FlowNet by Dosovitskiy *et al.* [10] represented a paradigm shift in optical flow estimation. The idea of using a simple convolutional neural network (CNN) architecture to directly learn the concept of optical flow from data was completely disjoint from all the established approaches. However, first implementations of new ideas often have a hard time competing with highly fine-tuned existing methods, and FlowNet was no exception to this rule. It is the successive consolidation that resolves the negative effects and helps us appreciate the benefits of new ways of thinking.

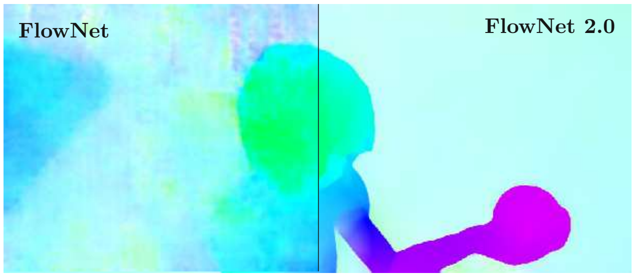

Figure 1. We present an extension of FlowNet. FlowNet 2.0 yields smooth flow fields, preserves fine motion details and runs at 8 to 140fps. The accuracy on this example is four times higher than with the original FlowNet.

The present paper is about a consolidation of the FlowNet idea. The resulting FlowNet 2.0 inherits the advantages of the original FlowNet, such as mastering large displacements, correct estimation of very fine details in the optical flow field, the potential to learn priors for specific scenarios, and fast runtimes. At the same time, it resolves problems with small displacements and noisy artifacts in estimated flow fields. This leads to a dramatic performance improvement on real-world applications such as action recognition and motion segmentation, bringing FlowNet 2.0 to the state-of-the-art level.

The way towards FlowNet 2.0 is via several evolutionary, but decisive modifications that are not trivially connected to the observed problems. First, we evaluate the influence of dataset schedules. Interestingly, the more sophisticated training data provided by Mayer *et al.* [18] leads to inferior results if used in isolation. However, a learning schedule consisting of multiple datasets improves results significantly. In this scope, we also found that the FlowNet version with an explicit correlation layer outperforms the version without such layer. This is in contrast to the results reported in Dosovitskiy *et al.* [10].

As a second contribution, we introduce a warping operation and show how stacking multiple networks using this operation can significantly improve the results. By varying the depth of the stack and the size of individual components we obtain many network variants with different size and runtime. This allows us to control the trade-off between accuracy and computational resources. We provide networks for the spectrum between 8fps and 140fps.

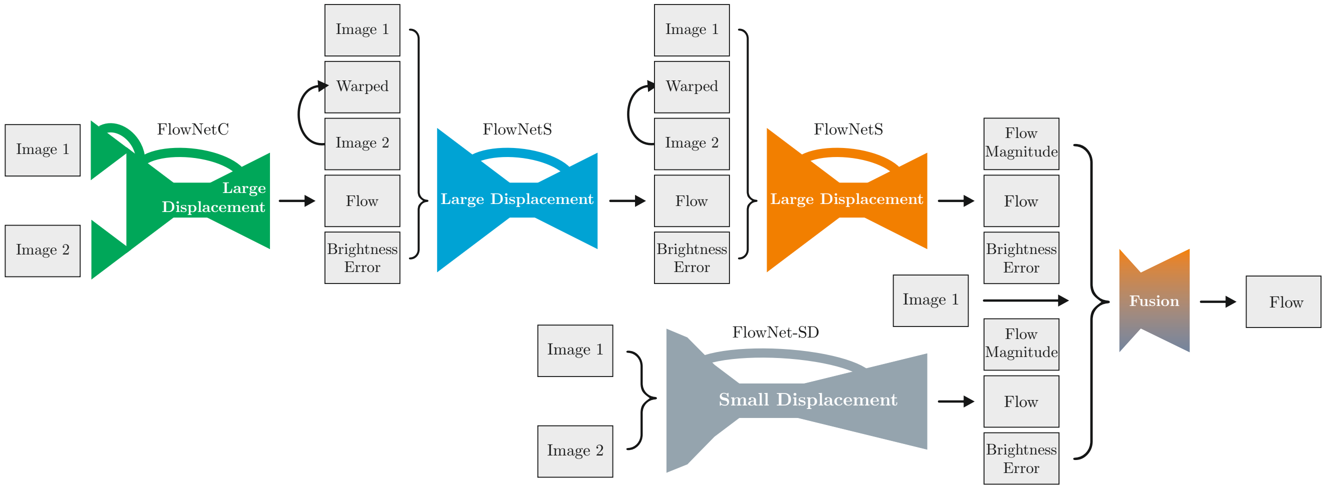

Figure 2. Schematic view of complete architecture: To compute large displacement optical flow we combine multiple FlowNets. Braces indicate concatenation of inputs. *Brightness Error* is the difference between the first image and the second image warped with the previously estimated flow. To optimally deal with small displacements, we introduce smaller strides in the beginning and convolutions between upconvolutions into the FlowNetS architecture. Finally we apply a small fusion network to provide the final estimate.

Finally, we focus on small, subpixel motion and real-world data. To this end, we created a special training dataset and a specialized network. We show that the architecture trained with this dataset performs well on small motions typical for real-world videos. To reach optimal performance on arbitrary displacements, we add a network that learns to fuse the former stacked network with the small displacement network in an optimal manner.

The final network outperforms the previous FlowNet by a large margin and performs on par with state-of-the-art methods on the Sintel and KITTI benchmarks. It can estimate small and large displacements with very high level of detail while providing interactive frame rates. We make code and data available under `http://lmb.informatik.uni-freiburg.de`.

2. Related Work

End-to-end optical flow estimation with CNNs was proposed by Dosovitskiy *et al.* in [10]. Their "FlowNet" model directly outputs the flow field for a pair of images. Following FlowNet, several papers have studied optical flow estimation with CNNs: featuring a 3D CNN [30], unsupervised learning [1, 33], carefully designed rotationally invariant architectures [28], or a pyramidal approach based on the coarse-to-fine idea of variational methods [20]. None of these significantly outperform the original FlowNet.

An alternative approach to learning-based optical flow estimation is to use CNNs to match image patches. Thewlis *et al.* [29] formulate Deep Matching [31] as a CNN and optimize it end-to-end. Gadot & Wolf [12] and Bailer *et al.* [3] learn image patch descriptors using Siamese network architectures. These methods can reach good accuracy, but require exhaustive matching of patches. Thus, they are restrictively slow for most practical applications. Moreover, methods based on (small) patches are inherently unable to use the larger whole-image context.

CNNs trained for per-pixel prediction tasks often produce noisy or blurry results. As a remedy, off-the-shelf optimization can be applied to the network predictions (e.g., optical flow can be postprocessed with a variational approach [10]). In some cases, this refinement can be approximated by neural networks: Chen & Pock [9] formulate their reaction diffusion model as a CNN and apply it to image denoising, deblocking and superresolution. Recently, it has been shown that similar refinement can be obtained by stacking several CNNs on top of each other. This led to improved results in human pose estimation [17, 8] and semantic instance segmentation [22]. In this paper we adapt the idea of stacking networks to optical flow estimation.

Our network architecture includes warping layers that compensate for some already estimated preliminary motion in the second image. The concept of image warping is common to all contemporary variational optical flow methods and goes back to the work of Lucas & Kanade [16]. In Brox *et al.* [6] it was shown to correspond to a numerical fixed point iteration scheme coupled with a continuation method.

The strategy of training machine learning models on a series of gradually increasing tasks is known as curriculum learning [5]. The idea dates back at least to Elman [11], who showed that both the evolution of tasks and the network architectures can be beneficial in the language processing scenario. In this paper we revisit this idea in the context of computer vision and show how it can lead to dramatic performance improvement on a complex real-world task of optical flow estimation.

3. Dataset Schedules

High quality training data is crucial for the success of supervised training. We investigated the differences in the quality of the estimated optical flow depending on the presented training data. Interestingly, it turned out that not only the kind of data is important but also the order in which it is presented during training.

The original FlowNets [10] were trained on the FlyingChairs dataset (we will call it Chairs). This rather simplistic dataset contains about 22k image pairs of chairs superimposed on random background images from Flickr. Random affine transformations are applied to chairs and background to obtain the second image and ground truth flow fields. The dataset contains only planar motions.

The FlyingThings3D (Things3D) dataset proposed by Mayer *et al.* [18] can be seen as a three-dimensional version of Chairs: 22k renderings of random scenes show 3D models from the ShapeNet dataset [23] moving in front of static 3D backgrounds. In contrast to Chairs, the images show true 3D motion and lighting effects and there is more variety among the object models.

We tested the two network architectures introduced by Dosovitskiy *et al.* [10]: FlowNetS, which is a straightforward encoder-decoder architecture, and FlowNetC, which includes explicit correlation of feature maps. We trained FlowNetS and FlowNetC on Chairs and Things3D and an equal mixture of samples from both datasets using the different learning rate schedules shown in Figure 3. The basic schedule S_{short} (600k iterations) corresponds to Dosovitskiy *et al.* [10] except for minor changes[1]. Apart from this basic schedule S_{short}, we investigated a longer schedule S_{long} with 1.2M iterations, and a schedule for fine-tuning S_{fine} with smaller learning rates. Results of networks trained on Chairs and Things3D with the different schedules are given in Table 1. The results lead to the following observations:

The order of presenting training data with different properties matters. Although Things3D is more realistic, training on Things3D alone leads to worse results than training on Chairs. The best results are consistently achieved when first training on Chairs and only then fine-tuning on Things3D. This schedule also outperforms training on a mixture of Chairs and Things3D. We conjecture that the simpler Chairs dataset helps the network learn the general concept of color matching without developing possibly confusing priors for 3D motion and realistic lighting too early. The result indicates the importance of training data schedules for avoiding shortcuts when learning generic concepts with deep networks.

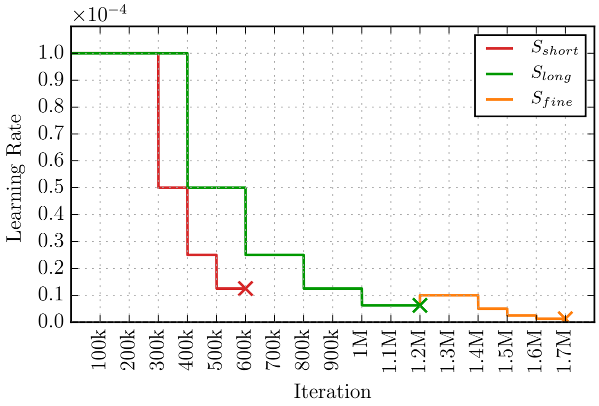

Figure 3. Learning rate schedules: S_{short} is similar to the schedule in Dosovitskiy *et al.* [10]. We investigated another longer version S_{long} and a fine-tuning schedule S_{fine}.

Architecture	Datasets	S_{short}	S_{long}	S_{fine}
FlowNetS	Chairs	4.45	-	-
	Chairs	-	4.24	4.21
	Things3D	-	5.07	4.50
	mixed	-	4.52	4.10
	Chairs→Things3D	-	4.24	**3.79**
FlowNetC	Chairs	3.77	-	-
	Chairs→Things3D	-	3.58	**3.04**

Table 1. Results of training FlowNets with different schedules on different datasets (one network per row). Numbers indicate endpoint errors on Sintel train *clean*. *mixed* denotes an equal mixture of Chairs and Things3D. Training on Chairs first and fine-tuning on Things3D yields the best results (the same holds when testing on the KITTI dataset; see supplemental material). FlowNetC performs better than FlowNetS.

FlowNetC outperforms FlowNetS. The result we got with FlowNetS and S_{short} corresponds to the one reported in Dosovitskiy *et al.* [10]. However, we obtained much better results on FlowNetC. We conclude that Dosovitskiy *et al.* [10] did not train FlowNetS and FlowNetC under the exact same conditions. When done so, the FlowNetC architecture compares favorably to the FlowNetS architecture.

Improved results. Just by modifying datasets and training schedules, we improved the FlowNetS result reported by Dosovitskiy *et al.* [10] by $\sim$ 25% and the FlowNetC result by $\sim$ 30%. In this section, we did not yet use specialized training sets for special scenarios. The trained network is rather supposed to be generic and to work well everywhere. An additional optional component in dataset schedules is fine-tuning of a generic network to a specific scenario, such as the driving scenario, which we show in Section 6.

[1] (1) We do not start with a learning rate of $1e-6$ and increase it first, but we start with $1e-4$ immediately. (2) We fix the learning rate for 300k iterations and then divide it by 2 every 100k iterations.

Stack architecture	Training enabled		Warping included	Warping gradient enabled	Loss after		EPE on Chairs test	EPE on Sintel train *clean*
	Net1	Net2			Net1	Net2		
Net1	✓	–	–	–	✓	–	3.01	3.79
Net1 + Net2	✗	✓	✗	–	–	✓	2.60	4.29
Net1 + Net2	✓	✓	✗	–	✗	✓	2.55	4.29
Net1 + Net2	✓	✓	✗	–	✓	✓	2.38	3.94
Net1 + W + Net2	✗	✓	✓	–	–	✓	1.94	**2.93**
Net1 + W + Net2	✓	✓	✓	✓	✗	✓	1.96	3.49
Net1 + W + Net2	✓	✓	✓	✓	✓	✓	**1.78**	3.33

Table 2. Evaluation of options when stacking two FlowNetS networks (Net1 and Net2). Net1 was trained with the Chairs→Things3D schedule from Section 3. Net2 is initialized randomly and subsequently, Net1 and Net2 together, or only Net2 is trained on Chairs with S_{long}; see text for details. The column "Warping Gradient Enabled" indicates whether the warping operation produces a gradient during backpropagation. When training without warping, the stacked network overfits to the Chairs dataset. The best results on Sintel are obtained when fixing Net1 and training Net2 with warping.

4. Stacking Networks

4.1. Stacking Two Networks for Flow Refinement

All state-of-the-art optical flow approaches rely on iterative methods [7, 31, 21, 2]. Can deep networks also benefit from iterative refinement? To answer this, we experiment with stacking multiple FlowNetC/S architectures.

The first network in the stack always gets the images I_1 and I_2 as input. Subsequent networks get I_1, I_2, and the previous flow estimate $w_i = (u_i, v_i)^\top$, where i denotes the index of the network in the stack.

To make assessment of the previous error and computing an incremental update easier for the network, we also optionally warp the second image $I_2(x, y)$ via the flow w_i and bilinear interpolation to $\tilde{I}_{2,i}(x, y) = I_2(x+u_i, y+v_i)$. This way, the next network in the stack can focus on the remaining increment between I_1 and $\tilde{I}_{2,i}$. When using warping, we additionally provide $\tilde{I}_{2,i}$ and the error $e_i = ||\tilde{I}_{2,i} - I_1||$ as input to the next network; see Figure 2. Thanks to bilinear interpolation, the derivatives of the warping operation can be computed (see supplemental material for details). This enables training of stacked networks end-to-end.

Table 2 shows the effect of stacking two networks, of warping, and of end-to-end training. We take the best FlowNetS from Section 3 and add another FlowNetS on top. The second network is initialized randomly and then the stack is trained on Chairs with the schedule S_{long}. We experimented with two scenarios: keeping the weights of the first network fixed, or updating them together with the weights of the second network. In the latter case, the weights of the first network are fixed for the first 400k iterations to first provide a good initialization of the second network. We report the error on Sintel train *clean* and on the test set of Chairs. Since the Chairs test set is much more similar to the training data than Sintel, comparing results on both datasets allows us to detect tendencies to over-fitting.

We make the following observations: (1) Just stacking networks without warping yields better results on Chairs, but worse on Sintel; the stacked network is over-fitting. (2) Stacking *with* warping always improves results. (3) Adding an intermediate loss after Net1 is advantageous when training the stacked network end-to-end. (4) The best results are obtained by keeping the first network fixed and only training the second network after the warping operation.

Clearly, since the stacked network is twice as big as the single network, over-fitting is an issue. The positive effect of flow refinement after warping can counteract this problem, yet the best of both is obtained when the stacked networks are trained one after the other, since this avoids over-fitting while having the benefit of flow refinement.

4.2. Stacking Multiple Diverse Networks

Rather than stacking identical networks, it is possible to stack networks of different type (FlowNetC and FlowNetS). Reducing the size of the individual networks is another valid option. We now investigate different combinations and additionally also vary the network size.

We call the first network the *bootstrap* network as it differs from the second network by its inputs. The second network could however be repeated an arbitray number of times in a recurrent fashion. We conducted this experiment and found that applying a network with the same weights multiple times and also fine-tuning this recurrent part does not improve results (see supplemental material for details). As also done in [17, 9], we therefore add networks with different weights to the stack. Compared to identical weights, stacking networks with different weights increases the memory footprint, but does not increase the runtime. In this case the top networks are not constrained to a general improvement of their input, but can perform different tasks at different stages and the stack can be trained in smaller pieces by fixing existing networks and adding new networks

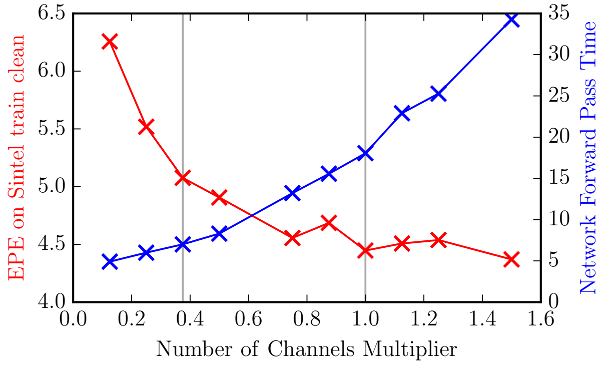

Figure 4. Accuracy and runtime of FlowNetS depending on the network width. The multiplier 1 corresponds to the width of the original FlowNet architecture. Wider networks do not improve the accuracy. For fast execution times, a factor of $\frac{3}{8}$ is a good choice. Timings are from an Nvidia GTX 1080.

	Number of Networks			
	1	2	3	4
Architecture	s	ss	sss	–
Runtime	**7ms**	14ms	20ms	
EPE	4.55	3.22	3.12	
Architecture	S	SS	–	–
Runtime	18ms	37ms		
EPE	3.79	2.56		
Architecture	c	cs	css	csss
Runtime	17ms	24ms	31ms	36ms
EPE	3.62	2.65	2.51	2.49
Architecture	C	CS	CSS	–
Runtime	33ms	51ms	69ms	
EPE	3.04	2.20	**2.10**	

Table 3. Results on Sintel train *clean* for some variants of stacked FlowNet architectures following the best practices of Section 3 and Section 4.1. Each new network was first trained on Chairs with S_{long} and then on Things3D with S_{fine} (Chairs→Things3D schedule). Forward pass times are from an Nvidia GTX 1080.

one-by-one. We do so by using the Chairs→Things3D schedule from Section 3 for every new network and the best configuration with warping from Section 4.1. Furthermore, we experiment with different network sizes and alternatively use FlowNetS or FlowNetC as a bootstrapping network. We use FlowNetC only in case of the bootstrap network, as the input to the next network is too diverse to be properly handeled by the Siamese structure of FlowNetC. Smaller size versions of the networks were created by taking only a fraction of the number of channels for every layer in the network. Figure 4 shows the network accuracy and runtime for different network sizes of a single FlowNetS. Factor $\frac{3}{8}$ yields a good trade-off between speed and accuracy when aiming for faster networks.

Notation: We denote networks trained by the Chairs→Things3D schedule from Section 3 starting with *FlowNet2*. Networks in a stack are trained with this schedule one-by-one. For the stack configuration we append upper- or lower-case letters to indicate the original FlowNet or the thin version with $\frac{3}{8}$ of the channels. E.g: *FlowNet2-CSS* stands for a network stack consisting of one FlowNetC and two FlowNetS. *FlowNet2-css* is the same but with fewer channels.

Table 3 shows the performance of different network stacks. Most notably, the final FlowNet2-CSS result improves by ~30% over the single network FlowNet2-C from Section 3 and by ~50% over the original FlowNetC [10]. Furthermore, two small networks in the beginning always outperform one large network, despite being faster and having fewer weights: FlowNet2-ss (11M weights) over FlowNet2-S (38M weights), and FlowNet2-cs (11M weights) over FlowNet2-C (38M weights). Training smaller units step by step proves to be advantageous and enables us to train very deep networks for optical flow. At last, FlowNet2-s provides nearly the same accuracy as the original FlowNet [10], while running at 140 frames per second.

5. Small Displacements

5.1. Datasets

While the original FlowNet [10] performed well on the Sintel benchmark, limitations in real-world applications have become apparent. In particular, the network cannot reliably estimate small motions (see Figure 1). This is counter-intuitive, since small motions are easier for traditional methods, and there is no obvious reason why networks should not reach the same performance in this setting. Thus, we compared the training data to the UCF101 dataset [25] as one example of real-world data. While Chairs are similar to Sintel, UCF101 is fundamentally different (we refer to our supplemental material for the analysis): Sintel is an action movie and as such contains many fast movements that are difficult for traditional methods, while the displacements we see in the UCF101 dataset are much smaller, mostly smaller than 1 pixel. Thus, we created a dataset in the visual style of Chairs but with very small displacements and a displacement histogram much more like UCF101. We also added cases with a background that is homogeneous or just consists of color gradients. We call this dataset *ChairsSDHom*.

5.2. Small Displacement Network and Fusion

We fine-tuned our FlowNet2-CSS network for smaller displacements by further training the whole network stack on a mixture of Things3D and ChairsSDHom and by applying a non-linearity to the error to downweight large displacements[2]. We denote this network by *FlowNet2-CSS-ft-sd*. This improves results on small displacements and we found that this particular mixture does not sacrifice performance on large displacements. However, in case of subpixel motion, noise still remains a problem and we conjecture that the FlowNet architecture might in general not be perfect for such motion. Therefore, we slightly modified the original FlowNetS architecture and removed the stride 2 in the first layer. We made the beginning of the network deeper by exchanging the 7×7 and 5×5 kernels in the beginning with multiple 3×3 kernels[2]. Because noise tends to be a problem with small displacements, we add convolutions between the upconvolutions to obtain smoother estimates like in [18]. We denote the resulting architecture by *FlowNet2-SD*; see Figure 2.

Finally, we created a small network that fuses FlowNet2-CSS-ft-sd and FlowNet2-SD (see Figure 2). The fusion network receives the flows, the flow magnitudes and the errors in brightness after warping as input. It contracts the resolution twice by a factor of 2 and expands again[2]. Contrary to the original FlowNet architecture it expands to the full resolution. We find that this produces crisp motion boundaries and performs well on small as well as on large displacements. We denote the final network as *FlowNet2*.

6. Experiments

We compare the best variants of our network to state-of-the-art approaches on public benchmarks. In addition, we provide a comparison on application tasks, such as motion segmentation and action recognition. This allows benchmarking the method on real data.

6.1. Speed and Performance on Public Benchmarks

We evaluated all methods[3] on an Intel Xeon E5 at 2.40GHz with an Nvidia GTX 1080. Where applicable, dataset-specific parameters with best error scores were used. Endpoint errors and runtimes are given in Table 4.

Sintel: On Sintel, FlowNet2 consistently outperforms DeepFlow [31] and EpicFlow [21] and is on par with FlowFields [2]. All methods with comparable runtimes have clearly inferior accuracy. We fine-tuned FlowNet2 on a mixture of Sintel *clean+final* training data (FlowNet2–ft-sintel). On the benchmark, in case of *clean* data this slightly degraded the result, while on *final* data FlowNet2–ft-sintel is on par with the currently published state-of-the art method DeepDiscreteFlow [13].

KITTI: On KITTI, the results of FlowNet2-CSS are comparable to EpicFlow [21] and FlowFields [2]. Fine-tuning on small displacement data degrades the result, likely due to KITTI containing very large flows in general. Fine-tuning on a combination of the KITTI2012 and KITTI2015 training sets reduces the error by a factor of ~3 (FlowNet2-ft-kitti). Among non-stereo methods we obtain the best EPE on KITTI2012 and the first rank on KITTI2015. This shows how well and elegantly the learning approach can integrate the prior of the driving scenario.

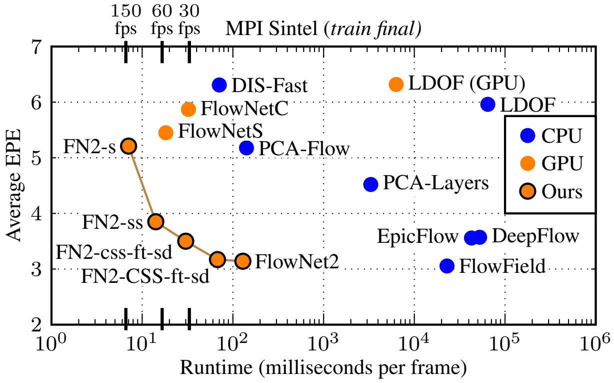

Figure 5. Runtime vs. endpoint error comparison to the fastest existing methods with available code. The FlowNet2 family outperforms other methods by a large margin. The behaviour for the KITTI dataset is the same; see supplemental material.

Middlebury: On the Middlebury training set FlowNet2 performs comparable to traditional methods. The results on the Middlebury test set are unexpectedly a lot worse. Still, there is a large improvement compared to FlowNetS [10].

Endpoint error vs. runtime evaluations for Sintel are provided in Figure 5. The FlowNet2 family outperforms the best and fastest existing methods by large margins. Depending on the type of application, a FlowNet2 variant between 8 to 140 frames per second can be used.

6.2. Qualitative Results

Figures 6 and 7 show example results on Sintel and on real-world data. While the performance on Sintel is similar to FlowFields [2], on real-world data FlowNet 2.0 is more robust to homogeneous regions (rows 2 and 5) and image and compression artifacts (rows 3 and 4), and it yields smooth flow fields with sharp motion boundaries.

6.3. Performance on Motion Segmentation and Action Recognition

To assess performance of FlowNet 2.0 in real-world applications, we compare the performance of action recognition and motion segmentation. For both applications, good optical flow is key. Thus, a good performance on these tasks also serves as an indicator for good optical flow.

For motion segmentation, we rely on the well-established approach of Ochs *et al.* [19] to compute long

[2] For details we refer to the supplemental material.

[3] An exception is EPPM for which we could not provide the required Windows environment and use the results from [4].

	Method	Sintel *clean*		Sintel *final*		KITTI 2012		KITTI 2015			Middlebury		Runtime	
		AEE		AEE		AEE		AEE	Fl-all	Fl-all	AEE		ms per frame	
		train	*test*	*train*	*test*	*train*	*test*	*train*	*train*	*test*	*train*	*test*	CPU	GPU
Accurate	EpicFlow† [21]	2.27	4.12	3.56	6.29	**3.09**	3.8	9.27	27.18%	**27.10%**	0.31	0.39	42,600	–
	DeepFlow† [31]	2.66	5.38	3.57	7.21	4.48	5.8	10.63	26.52%	29.18%	**0.25**	0.42	51,940	–
	FlowFields [2]	**1.86**	**3.75**	**3.06**	**5.81**	3.33	**3.5**	**8.33**	**24.43%**	–	0.27	**0.33**	22,810	–
	LDOF (CPU) [7]	4.64	7.56	5.96	9.12	10.94	12.4	18.19	38.11%	–	0.44	0.56	64,900	–
	LDOF (GPU) [26]	4.76	–	6.32	–	10.43	–	18.20	38.05%	–	0.36	–	–	6,270
	PCA-Layers [32]	3.22	5.73	4.52	7.89	5.99	5.2	12.74	27.26%	–	0.66	–	**3,300**	–
Fast	EPPM [4]	–	**6.49**	–	**8.38**	–	9.2	–	–	–	–	**0.33**	–	200
	PCA-Flow [32]	**4.04**	6.83	**5.18**	8.65	**5.48**	**6.2**	**14.01**	**39.59%**	–	**0.70**	–	140	–
	DIS-Fast [15]	5.61	9.35	6.31	10.13	11.01	14.4	21.20	53.73%	–	0.92	–	70ꟸ	–
	FlowNetS [10]	4.50	6.96‡	5.45	7.52‡	8.26	–	–	–	–	1.09	–	–	**18**
	FlowNetC [10]	4.31	6.85‡	5.87	8.51‡	9.35	–	–	–	–	1.15	–	–	32
FlowNet 2.0	FlowNet2-s	4.55	–	5.21	–	8.89	–	16.42	56.81%	–	1.27	–	–	**7**
	FlowNet2-ss	3.22	–	3.85	–	5.45	–	12.84	41.03%	–	0.68	–	–	14
	FlowNet2-css	2.51	–	3.54	–	4.49	–	11.01	35.19%	–	0.54	–	–	31
	FlowNet2-css-ft-sd	2.50	–	3.50	–	4.71	–	11.18	34.10%	–	0.43	–	–	31
	FlowNet2-CSS	2.10	–	3.23	–	**3.55**	–	**8.94**	29.77%	–	0.44	–	–	69
	FlowNet2-CSS-ft-sd	2.08	–	3.17	–	4.05	–	10.07	30.73%	–	0.38	–	–	69
	FlowNet2	**2.02**	**3.96**	**3.14**	6.02	4.09	–	10.06	30.37%	–	**0.35**	**0.52**	–	123
	FlowNet2-ft-sintel	(1.45)	4.16	(2.01)	**5.74**	3.61	–	9.84	**28.20%**	–	0.35	–	–	123
	FlowNet2-ft-kitti	3.43	–	4.66	–	(1.28)	**1.8**	(2.30)	(8.61%)	**11.48%**	0.56	–	–	123

Table 4. Performance comparison on public benchmarks. AEE: Average Endpoint Error; Fl-all: Ratio of pixels where flow estimate is wrong by both $\geq$ 3 pixels and $\geq$ 5%. The best number for each category is highlighted in bold. See text for details. For methods with runtimes under 10s we exclude I/O times. †*train* numbers for these methods use slower but better "improved" option. ‡For these results we report the fine-tuned numbers (FlowNetS-ft and FlowNetC-ft). ꟸRGB version of DIS-Fast (we compare all methods on RGB); DIS-Fast grayscale takes 22.2ms with ~2% larger errors.

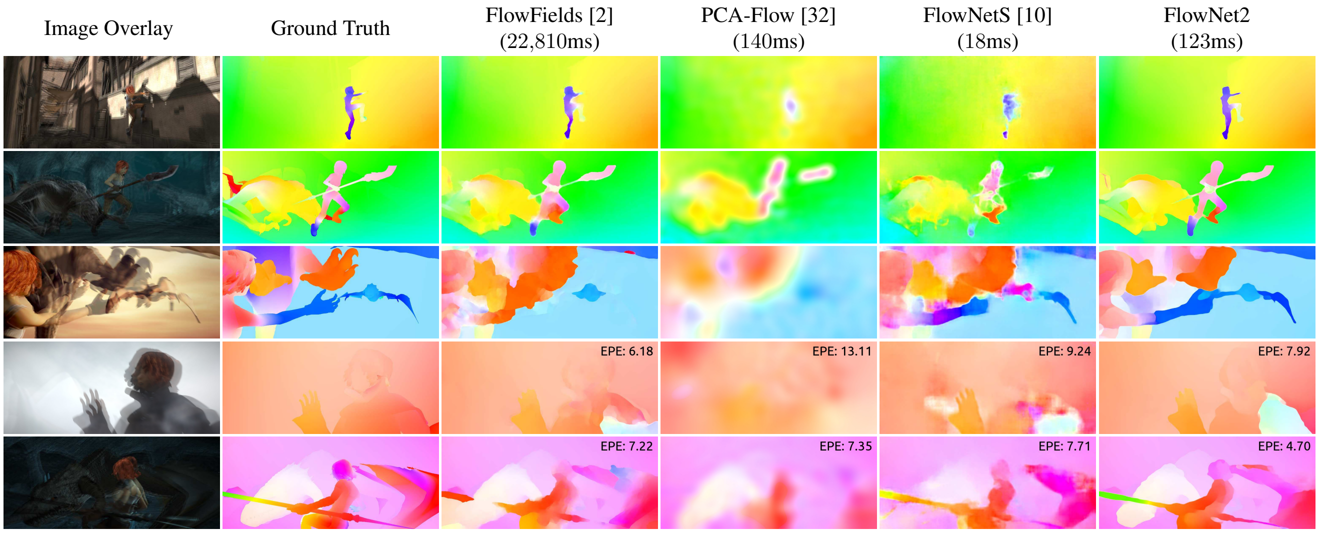

Figure 6. Examples of flow fields from different methods estimated on Sintel. FlowNet2 performs similar to FlowFields and is able to extract fine details, while methods running at comparable speeds perform much worse (PCA-Flow and FlowNetS).

term point trajectories. A motion segmentation is obtained from these using the state-of-the-art method from Keuper *et al.* [14]. The results are shown in Table 5. The original model in Ochs *et al.* [14] was built on Large Displacement Optical Flow [7]. We included also other popular optical flow methods in the comparison. The old FlowNet [10] was not useful for motion segmentation. In contrast, the FlowNet2 is as reliable as other state-of-the-art methods while being orders of magnitude faster.

Optical flow is also a crucial feature for action recognition. To assess the performance, we trained the temporal stream of the two-stream approach from Simonyan *et al.* [24] with different optical flow inputs. Table 5 shows that FlowNetS [10] did not provide useful results, while the flow from FlowNet 2.0 yields comparable results to state-of-the art methods.

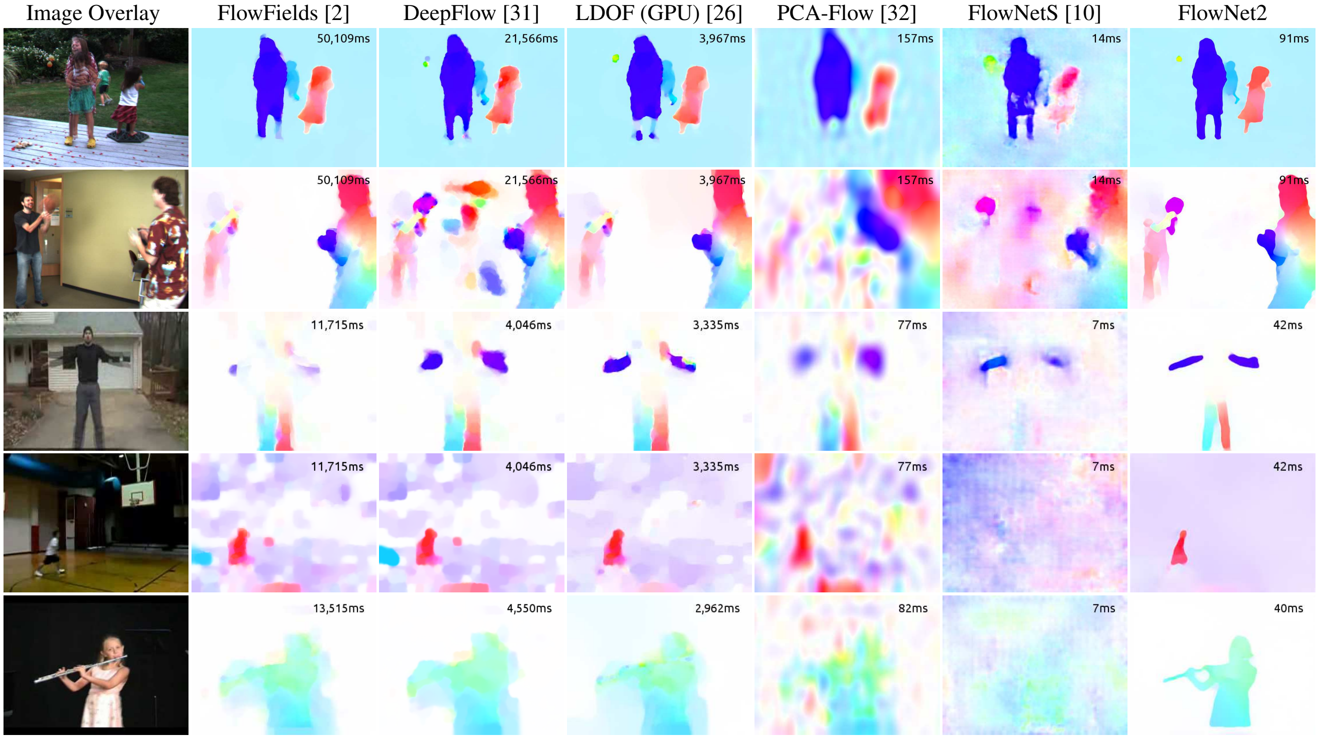

Figure 7. Examples of flow fields from different methods estimated on real-world data. The top two rows are from the Middlebury data set and the bottom three from UCF101. Note how well FlowNet2 generalizes to real-world data, i.e. it produces smooth flow fields, crisp boundaries and is robust to motion blur and compression artifacts. Given timings of methods differ due to different image resolutions.

	Motion Seg.		Action Recog.
	F-Measure	Extracted Objects	Accuracy
LDOF-CPU [7]	79.51%	28/65	79.91%†
DeepFlow [31]	**80.18%**	29/65	**81.89%**
EpicFlow [21]	78.36%	27/65	78.90%
FlowFields [2]	79.70%	**30/65**	79.38%
FlowNetS [10]	56.87%‡	3/62‡	55.27%
FlowNet2-css-ft-sd	77.88%	26/65	75.36%
FlowNet2-CSS-ft-sd	79.52%	30/65	**79.64%**
FlowNet2	**79.92%**	**32/65**	79.51%

Table 5. Motion segmentation and action recognition using different methods; see text for details. **Motion Segmentation:** We report results using [19, 14] on the training set of FBMS-59 [27, 19] with a density of 4 pixels. Different densities and error measures are given the supplemental material. "*Extracted objects*" refers to objects with $F \geq 75\%$. ‡FlowNetS is evaluated on 28 out of 29 sequences; on the sequence *lion02*, the optimization did not converge even after one week. **Action Recognition:** We report classification accuracies after training the temporal stream of [24]. We use a stack of 5 optical flow fields as input. †To reproduce results from [24], for action recognition we use the OpenCV LDOF implementation. Note the generally large difference for FlowNetS and FlowNet2 and the performance compared to traditional methods.

7. Conclusions

We have presented several improvements to the FlowNet idea that have led to accuracy that is fully on par with state-of-the-art methods while FlowNet 2.0 runs orders of magnitude faster. We have quantified the effect of each contribution and showed that all play an important role. The experiments on motion segmentation and action recognition show that the estimated optical flow with FlowNet 2.0 is reliable on a large variety of scenes and applications. The FlowNet 2.0 family provides networks running at speeds from 8 to 140fps. This further extends the possible range of applications. While the results on Middlebury indicate imperfect performance on subpixel motion, FlowNet 2.0 results highlight very crisp motion boundaries, retrieval of fine structures, and robustness to compression artifacts. Thus, we expect it to become the working horse for all applications that require accurate and fast optical flow computation.

Acknowledgements

We acknowledge funding by the ERC Starting Grant VideoLearn, the DFG Grant BR-3815/7-1, and the EU Horizon2020 project TrimBot2020.

References

[1] A. Ahmadi and I. Patras. Unsupervised convolutional neural networks for motion estimation. In *2016 IEEE Int. Conference on Image Processing (ICIP)*, 2016.

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

[3] C. Bailer, K. Varanasi, and D. Stricker. CNN based patch matching for optical flow with thresholded hinge loss. *arXiv pre-print*, arXiv:1607.08064, Aug. 2016.

[4] L. Bao, Q. Yang, and H. Jin. Fast edge-preserving patchmatch for large displacement optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Int. Conference on Machine Learning (ICML)*, 2009.

[6] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision (ECCV)*, 2004.

[7] 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.

[8] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[9] Y. Chen and T. Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, PP(99):1–1, 2016.

[10] 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 Int. Conference on Computer Vision (ICCV)*, 2015.

[11] J. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[12] D. Gadot and L. Wolf. Patchbatch: A batch augmented loss for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] F. Güney and A. Geiger. Deep discrete flow. In *Asian Conference on Computer Vision (ACCV)*, 2016.

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

[15] T. Kroeger, R. Timofte, D. Dai, and L. V. Gool. Fast optical flow using dense inverse search. In *European Conference on Computer Vision (ECCV)*, 2016.

[16] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th Int. Joint Conference on Artificial Intelligence (IJCAI)*, 1981.

[17] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision (ECCV)*, 2016.

[18] 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.

[19] 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.

[20] A. Ranjan and M. J. Black. Optical Flow Estimation using a Spatial Pyramid Network. *arXiv pre-print*, arXiv:1611.00850, Nov. 2016.

[21] 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.

[22] B. Romera-Paredes and P. H. S. Torr. Recurrent instance segmentation. In *European Conference on Computer Vision (ECCV)*, 2016.

[23] M. Savva, A. X. Chang, and P. Hanrahan. Semantically-Enriched 3D Models for Common-sense Knowledge (Workshop on Functionality, Physics, Intentionality and Causality). In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[24] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Int. Conference on Neural Information Processing Systems (NIPS)*, 2014.

[25] 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.

[26] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. In *European Conference on Computer Vision (ECCV)*, 2010.

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

[28] D. Teney and M. Hebert. Learning to extract motion from videos in convolutional neural networks. *arXiv pre-print*, arXiv:1601.07532, Feb. 2016.

[29] J. Thewlis, S. Zheng, P. H. Torr, and A. Vedaldi. Fully-trainable deep matching. In *British Machine Vision Conference (BMVC)*, 2016.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Deep end2end voxel2voxel prediction (the 3rd workshop on deep learning in computer vision). In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

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

[32] 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.

[33] J. J. Yu, A. W. Harley, and K. G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. *arXiv pre-print*, arXiv:1608.05842, Sept. 2016.