



# TrimBot2020 Deliverable 5.1

# Dynamic Reconstruction Progress Report

Principal Author: ALUF
Contributors: ALUF
Dissemination: PU

**Abstract:** A major goal in WP5 is an accurate scene flow estimation software that can run at interactive framerates. In this report, we present the current state of our research and demonstrate that it is on a good way to achieve this goal. The chosen approach formulates disparity estimation and optical flow estimation as end-to-end learning tasks that are solved via deep learning. The learning approach will allow us to adapt to the specific garden scenario and the specific camera type used in the project by training on data that has been specifically created for this purpose. Challenges in the scope of plants, such as heavy self-occlusion and fine details, can be treated elegantly within such formulation. We present results that aim for the first project demonstrator, which is supposed to trim a boxwood bush. However, the approach we have been chosen is very generic and will be applicable also to other plants in the later project stages. Moreover, it can support 3D reconstruction in WP3.

Deliverable due: Month 12

# Contents



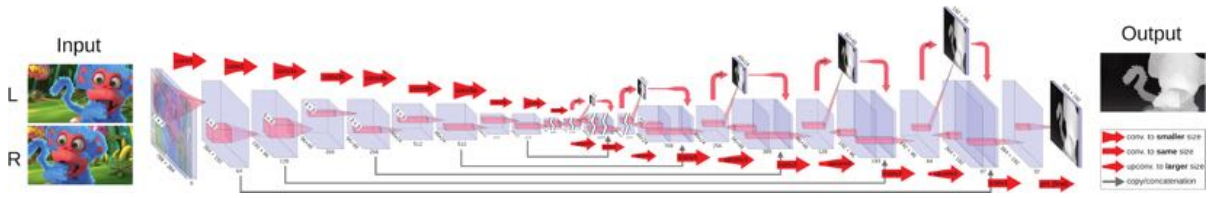



Figure 1: **DispNet architecture.** The network has a straightforward design and needs no pre- or postprocessing. Shown here is the *simple* variant; the slightly more complex two-stream *correlation* variant achieves better scores by making explicit use of the input images' "rectified stereo" property (we refer to our paper for details [1]). In the graphic, data flows from left to right. The input images traverse a number of *convolution layers* in which many small convolutional filters with learned weights gradually transform the data from image space into an abstract representation within the network's learned deep feature space. The blue shapes visually represent this change by becoming smaller in the "spatial" dimensions (width, height) but deeper. The first part of the network, up to the narrowest point, is called the "analysis", "encoder", or "contracting" part. Afterwards, the "synthesis", "decoder", or "expanding" part uses *upconvolution layers* to reassemble the network output back into image space.

# 1 DispNet: disparity estimation with a deep network

DispNet is a convolutional network architecture that is trained end-to-end to solve the task of disparity estimation; see Figure 1. It receives a rectified stereo image pair as input and outputs a disparity map for the left view of the pair. The learning approach requires a large amount of training data. To this end, we created a large set of randomized rendered data and found that training on such data generalizes well also to real imagery.

One potential advantage of a learning framework over previous work on disparity estimation is the capability of deep networks to learn special priors for a certain type of data. In case of Trimbot, this will be vegetation. The data we trained on so far, was not specific to vegetation. Section 4 describes our ongoing work on generating data specific to the garden scenes in Trimbot. We already verified the concept of specialization to a task on the public KITTI benchmark[1] [2], which specializes on an automotive scenario in a car. Finetuning the network on this kind of data largely improved the results in this particular scenario. We conjecture that the specialization for vegetation in garden scenes will be similarly successful.

Another large advantage of a learning approach is the fast runtime of deep networks on GPUs. DispNet already provides interactive framerates (about 15Hz) at this early stage of the project, although this capability was only planned for month 24. DispNet is currently the fastest technique among the highest-scoring stereo disparity methods on the KITTI 2015 benchmark and still yields very high accuracy.

For known camera parameters, depth can be directly derived from the disparities. Depth is critical for the fine control tasks of the TrimBot project as it is an important cue for 3D scene understanding. The bush cutting task will directly use depth from stereo to establish a target surface for trimming. The later rose cutting task will use it indirectly while infering plant structure, and successively cutting points on the plant.

[1] `http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo`

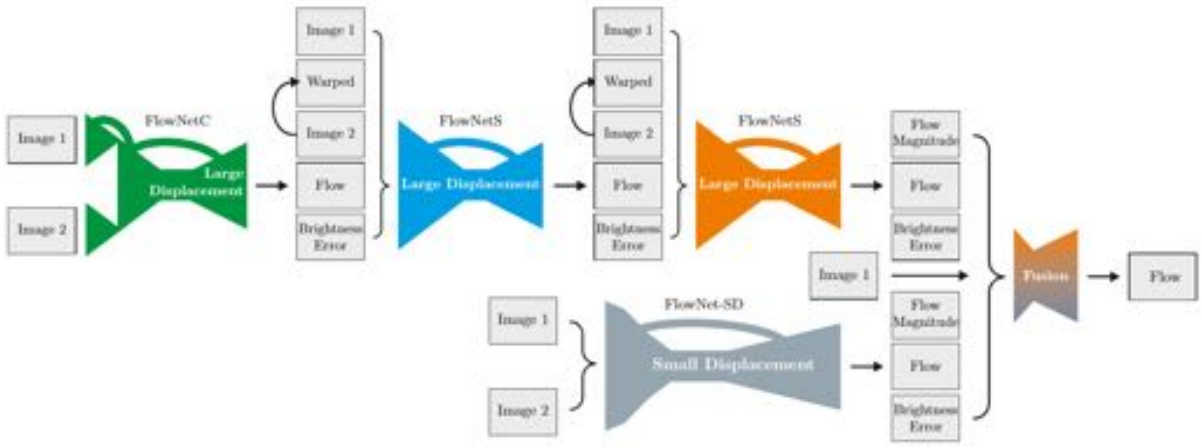



Figure 2: **FlowNet2 architecture.** The multistream approach couples a general multistage network which refines its own predictions and a specialized small-displacements network. Each colored shape is a complete encoder-decoder network.

*DispNet* has been published in a conference paper [1], and we refer to this paper for technical details about the method and the datasets on which it has been trained.

# 2 Optical flow estimation: FlowNet2

The optical flow estimator *FlowNet2* is part of a work which is still under peer review for CVPR 2017 [3]. Like DispNet, FlowNet2 is a convolutional architecture, but it follows a multistage approach in which it iteratively refines its own predictions; see Fig. 2. FlowNet2 can run at interactive and realtime framerates, as well: we provide a number of networks for a wide range of speed/accuracy tradeoff choices, but even the best and slowest networks run at 8 Hz. DispNet and FlowNet2 together yield *scene flow*.

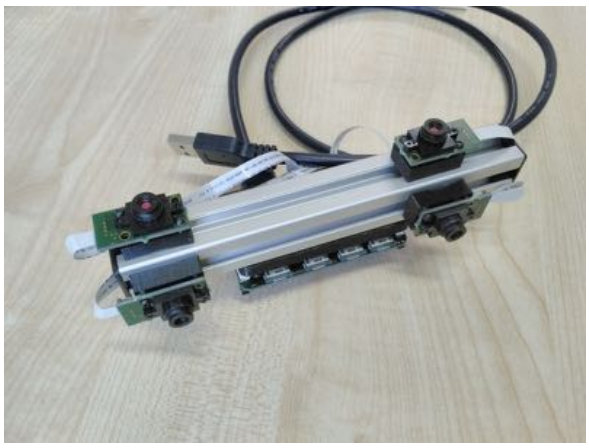
# 3 Camera

We have purchased the multi-camera system that is intended to be used on the robot and the arm throughout the project[2]. The system can produce synchronized images from multiple attached camera modules, with a resolution of $752 \times 480$ pixels (using the provided modules) at standard framerates. Calibration was done using the Kalibr[3] software on Apriltag[4] targets. After undistortion and rectification, slightly smaller images remain (see Figure 7).

[2] `https://github.com/PX4/uvc_ros_driver/wiki`
[3] `https://github.com/ethz-asl/kalibr`
[4] `https://people.csail.mit.edu/kaess/apriltags/`

Camera schematic with four stereo pairs



Real camera with two stereo pairs

Figure 3: **Camera setup.** We did not use the camera's builtin SGM stereo function.

# 4 Synthetic garden data

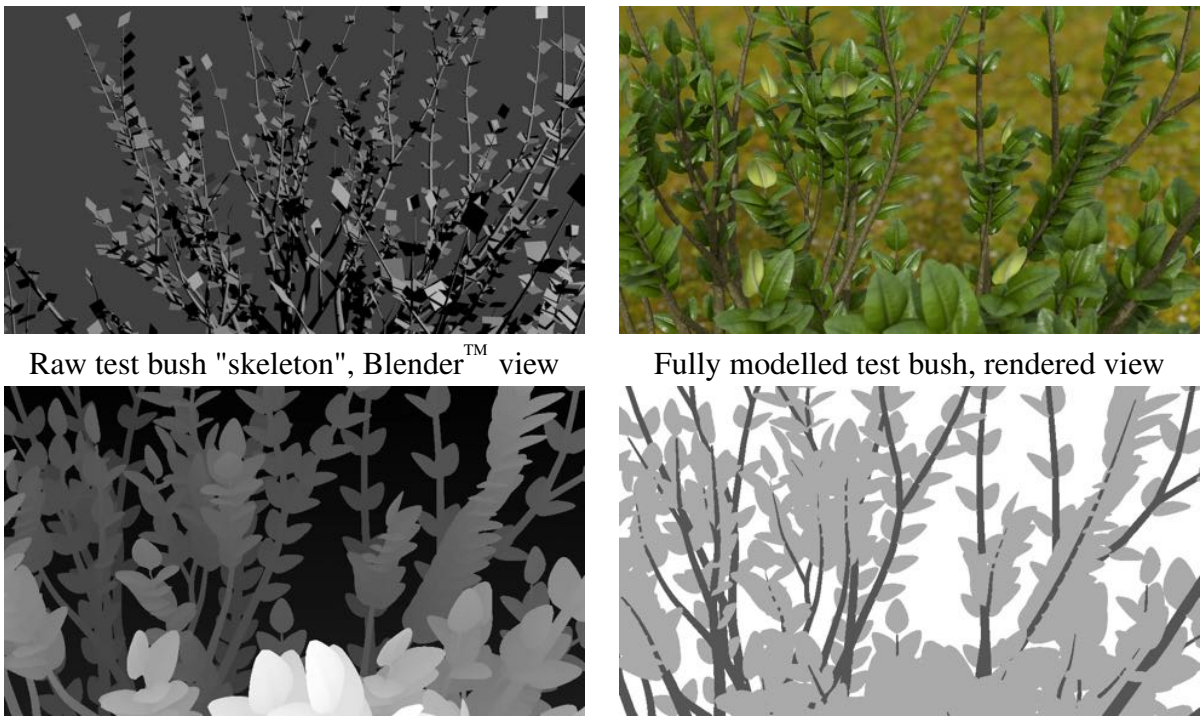
Our central approach to the reconstruction problems of TrimBot is to use *machine learning*, i.e. to "train" self-learning systems by example and guidance to perform tasks, instead of explicitly formulating explicit rules on how to process a specific input. There is no available training dataset available for our purpose. We will follow the approach to generate our own data by rendering garden scenes at a sufficient level of detail and with sufficient diversity to generalize to other gardens.

## 4.1 Task-specific training data

The working environment of the project's robot contains plants: intricate structures with complex foliage, articulated and elastic motion, nonlambertian surfaces and unordered multiple self-occlusion. These aspects present significant challenges to the robot and its software; they break assumptions often made by vision algorithms, such as "the scene is not moving" or "the appearance of a certain object does not change". Our machine learning approach has the advantage of not relying on any explicit assumptions. However, assumptions could occur implicitly: we design the scenes from which the training data is derived. A system trained exclusively on data with Lambertian surface properties may well fail when presented with specular objects during testing (i.e., in the real world).

Learning algorithms have a certain ability to *generalize*, i.e. to transfer their knowledge to cope with data which differs somewhat from that seen during training, but this effect has limits: training on data in which all leaves are 2cm long will not stop the system from coping with 3cm long leaves, but training on blue leaves could mean that green leaves are not understood to be the same kind of object. Unfortunately, it is not generally possible to predict which data characteristics will be important. Thus, the training data for this project should contain as many of the relevant intricacies as possible. We are planning for at least the following: nonrigidly moving plants; lighting effects such as specular reflection, translucency, and global illumination; realistic "garden"-style scenes; weather effects (sunny/overcast sky); shadowing by other objects, self-shadowing, and complex self-occlusion within objects. Key to the data



Raw test bush "skeleton", Blender™ view

Fully modelled test bush, rendered view

Disparity groundtruth

Segmentation into "branch" and "leaf" classes

Figure 4: **Plant data.** Modeling appearance and dynamics of real plants ensures that the machine learning system will receive the relevant training data. Note that segmentation data is not needed for the bush trimming task, but will be useful for the rose plants.

setup is the automated generation of plant structures for which we use a combination of publicly available software (Blender[5] and plugins, e.g. Sapling[6]) and our own extensions. Currently, we are at the state of setting up first garden scenes manually. For a large scale, diverse dataset, however, we plan for a partially automated software to create such scenes.

## 4.2 Camera-specific training data

Conventional vision algorithms work with manually defined descriptors, thresholds, tuned keypoint detectors etc. Ideally, such a system should work for all input data in all environments, but this is an illusory goal for all but the most trivial tasks: for example, an object segmenter will fail on blurry images if it has not been designed for this specific data fault, because blurriness weakens or even removes information-carrying boundaries between objects which are important segmentation cues.

Learning algorithms are afflicted by this as well: our experiment in Figure 6 demonstrates how a disparity estimation system trained on "good" (clean, sharp, good colors) data performs well on good data, but not as good on "bad" (blurry, overexposed) data. The advantage here is that there is no need to manually design robustness against e.g. blurriness into the system: it

[5] www.blender.org

[6] https://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Curve/Sapling_Tree

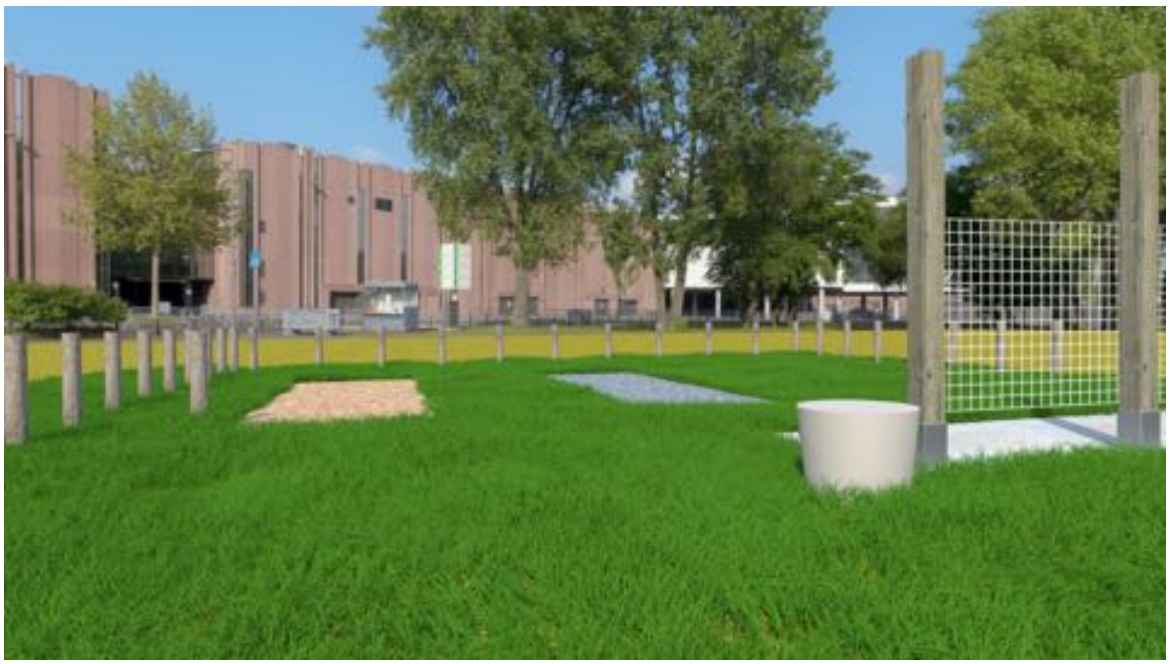


Figure 5: **Environment data.** Rendering synthetic plants within a simulated garden is an intuitive way to be sure that not only the plants themselves but also secondary effects such as lighting are natural. This experimental "garden" study was done manually, but we will explore automated generation techniques.

suffices to *train* it using blurry images; the necessary invariance is learned autonomously.

The concept behind this is fundamentally the same as in Section 4.1, but the required workloads are disjoint and independent; one happens before and the other after rendering. We are exploring data post-processing steps to match the imaging characteristics of the project camera (c.f. Figure 6).

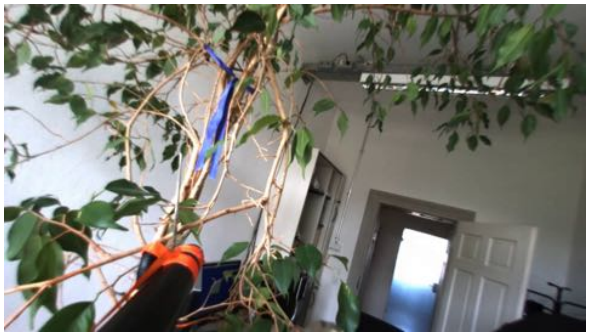
# 5 Summary

The research work on dynamic reconstruction is on a good way. We have developed deep networks for disparity estimation and optical flow estimation that show state-of-the-art performance on regular benchmarks and run at interactive framerates. This software is already up and running. Currently we are in the process to fully exploit the learning approach by creating training data that is optimized for the used camera and garden scenes. A network trained on such data is expected to specialize on the task and perform even much better. Apart from achieving higher accuracy and robustness, this could also allow for smaller versions of our networks that would run at interactive framerates on embedded devices.

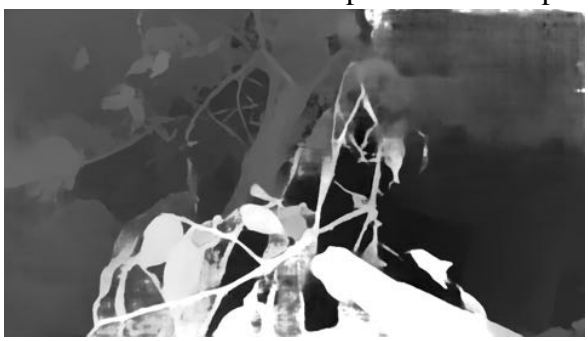
# References

[1] 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 es-

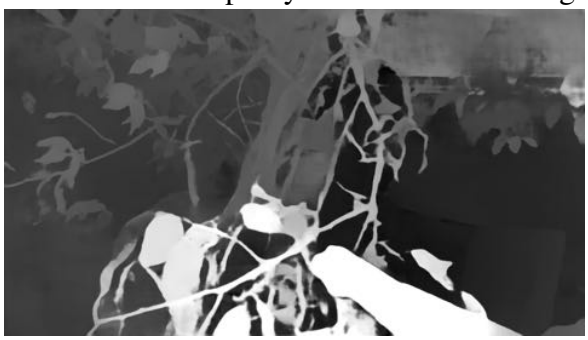
 

Stereo frame from an experimental setup using a Point Grey Bumblebee 2 stereo camera




Disparity estimate and training sample using original "clean" data




Disparity estimate and training sample using modified "degraded" data

Figure 6: **Train-test data discrepancy.** A DispNet [1] trained on clean synthetic data does not perform well out of the box; the input images from a Point Grey Bumblebee 2 stereo camera (https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-systems) show significant radial blur after undistortion, as well as general blur and other faults). Training on data which has been postprocessed to look more like the Bumblebee images leads to significantly more detailed estimates, especially far away from the image center where the radial blur is the strongest. Note that the CNN makes sensible "guesstimates" for areas which cannot actually be measured from the data due to the necessarily differing views of the two camera eyes (left border of left image).

timation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, arXiv:1512.02134, 2016. [Online]. Available: `http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16`.

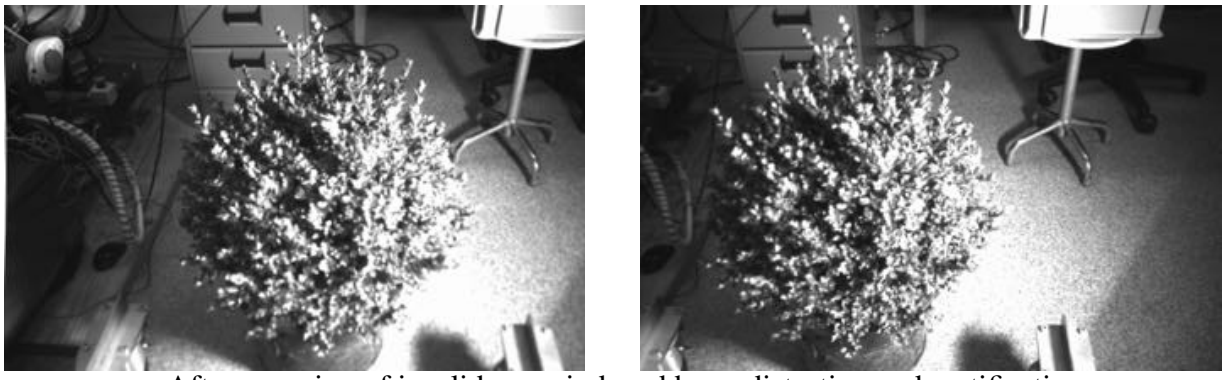
[2] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[3] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: evolution of optical flow estimation with deep networks," arXiv:1612.01925, Tech. Rep., Dec. 2016. [Online]. Available: `http://lmb.informatik.uni-freiburg.de//Publications/2016/IMKDB16`.

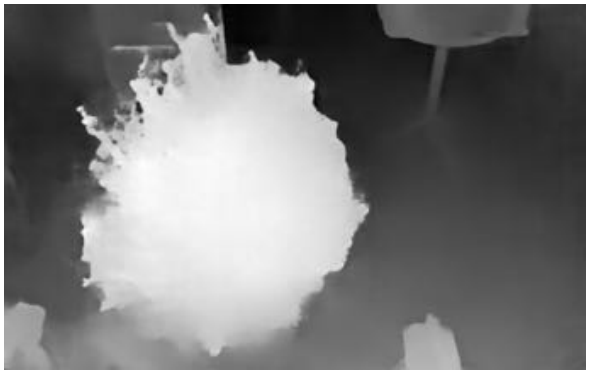
Raw camera outputs

After removal of lens distortion, stereo-rectified



After cropping of invalid areas induced by undistortion and rectification

Figure 7: **Disparity estimation on project camera.** An unmodified DispNet [1] produces sensible but low-quality disparity maps on the ETHZ camera due to discrepancies between its training data and this test data (beside the less-than-perfect image quality). Noteworthy points include: Grayscale instead of color images; excessive contrast and simulantenous over- and underexposure; vignetting; defocus blur (which even differs between the left and the right camera).



Output of disparity network (for right view)

# A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation

Nikolaus Mayer*[1], Eddy Ilg*[1], Philip Häusser*[2], Philipp Fischer*[1†]
[1]University of Freiburg [2]Technical University of Munich
[1]{mayern,ilg,fischer}@cs.uni-freiburg.de [2]haeusser@cs.tum.edu

Daniel Cremers
Technical University of Munich
cremers@tum.de

Alexey Dosovitskiy, Thomas Brox
University of Freiburg
{dosovits,brox}@cs.uni-freiburg.de

## Abstract

*Recent work has shown that optical flow estimation can be formulated as a supervised learning task and can be successfully solved with convolutional networks. Training of the so-called FlowNet was enabled by a large synthetically generated dataset. The present paper extends the concept of optical flow estimation via convolutional networks to disparity and scene flow estimation. To this end, we propose three synthetic stereo video datasets with sufficient realism, variation, and size to successfully train large networks. Our datasets are the first large-scale datasets to enable training and evaluation of scene flow methods. Besides the datasets, we present a convolutional network for real-time disparity estimation that provides state-of-the-art results. By combining a flow and disparity estimation network and training it jointly, we demonstrate the first scene flow estimation with a convolutional network.*

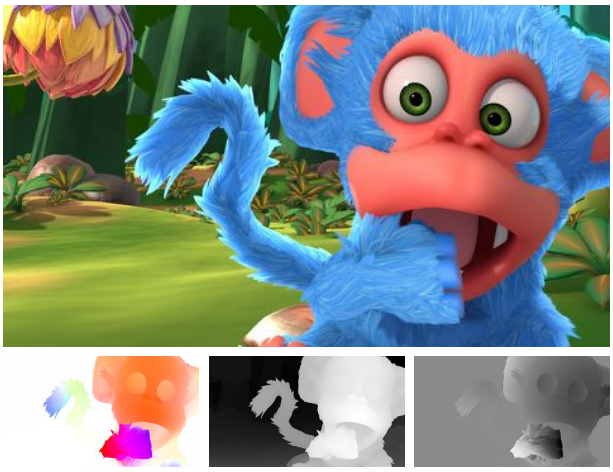


Figure 1. Our datasets provide over 35000 stereo frames with dense ground truth for *optical flow*, *disparity* and *disparity change*, as well as other data such as object segmentation.

## 1. Introduction

Estimating scene flow means providing the depth and 3D motion vectors of all visible points in a stereo video. It is the "royal league" task when it comes to reconstruction and motion estimation and provides an important basis for numerous higher-level challenges such as advanced driver assistance and autonomous systems. Research over the last decades has focused on its subtasks, namely disparity estimation and optical flow estimation, with considerable success. The full scene flow problem has not been explored to the same extent. While partial scene flow can be simply assembled from the subtask results, it is expected that the joint estimation of all components would be advantageous with regard to both efficiency and accuracy. One reason for scene flow being less explored than its subtasks seems to be a shortage of fully annotated ground truth data.

The availability of such data has become even more important in the era of convolutional networks. Dosovitskiy et al. [4] showed that optical flow estimation can be posed as a supervised learning problem and can be solved with a large network. For training their network, they created a simple synthetic 2D dataset of flying chairs, which proved to be sufficient to predict accurate optical flow in general videos. These results suggest that also disparities and scene flow can be estimated via a convolutional network, ideally jointly, efficiently, and in real-time. What is missing to implement this idea is a large dataset with sufficient realism and variability to train such a network and to evaluate its performance.

*These authors contributed equally

†Supported by the Deutsche Telekom Stiftung

| Dataset | MPI Sintel [2] | KITTI Benchmark Suite [16] | | SUN3D[26] | NYU2[21] | Ours | | |
|---|---|---|---|---|---|---|---|---|
| | | 2012 | 2015 | | | FlyingThings3D | Monkaa | Driving |
| #Training frames | 1064 | 194 | 200† | 2.5M | 1449 | 21818 | 8591 | 4392 |
| #Test frames | 564 | 195 | 200† | — | — | 4248 | — | — |
| #Training scenes | 25 | 194 | 200 | 415 | 464 | 2247 | 8 | 1 |
| Resolution | 1024×436 | 1226×370 | 1242×375 | 640×480 | 640×480 | 960×540 | 960×540 | 960×540 |
| Disparity/Depth | ✓ | sparse | sparse | ✓ | ✓ | ✓ | ✓ | ✓ |
| Disparity change | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Optical flow | ✓ | (sparse) | (sparse) | ✗ | ✗ | ✓ | ✓ | ✓ |
| Segmentation | ✓ | ✗ | ✗ | (✓) | ✓ | ✓ | ✓ | ✓ |
| Motion boundaries | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Naturalism | (✓) | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | (✓) |

Table 1. Comparison of available datasets: our new collection offers more annotated data and greater data variety than any existing choice. All our data has fully contiguous, dense, accurate ground truth. †Note that in KITTI 2015, a scene is a sequence of 21 stereo pairs, but groundtruth is only provided for a single frame.

In this paper, we present a collection of three such datasets, made using a customized version of the open source 3D creation suite Blender[3]. Our effort is similar in spirit to the Sintel benchmark [2]. In contrast to Sintel, our dataset is large enough to facilitate training of convolutional networks, and it provides ground truth for scene flow. In particular, it includes stereo color images and ground truth for bidirectional disparity, bidirectional optical flow and disparity change, motion boundaries, and object segmentation. Moreover, the full camera calibration and 3D point positions are available, i.e. our dataset also covers RGBD data. The datasets are freely available online[4].

We cannot exploit the full potential of this dataset in a single paper, but we already demonstrate various usage examples in conjunction with convolutional network training. We train a network for disparity estimation, which yields competitive performance also on previous benchmarks, especially among those methods that run in real-time. Finally, we also present a network for scene flow estimation and provide the first quantitative numbers on full scene flow on a sufficiently sized test set.

## 2. Related work

**Datasets.** The first significant efforts to create standard datasets were the Middlebury datasets for stereo disparity estimation [20] and optical flow estimation [1]. While the stereo dataset consists of real scenes, the optical flow dataset is a mixture of real scenes and rendered scenes. Both datasets are very small in today's terms. Especially the small test sets have led to heavy manual overfitting. An advantage of the stereo dataset is the availability of relevant real scenes, especially in the latest high-resolution version from 2014 [19].

MPI Sintel [2] is an entirely synthetic dataset derived from a short open source animated 3D movie. It provides dense ground truth for optical flow. Since very recently, a beta testing version with disparities is available for training. With 1064 training frames, the Sintel dataset is the largest dataset currently available. It contains sufficiently realistic scenes including natural image degradations such as fog and motion blur. The authors put much effort into the correctness of the ground truth for all frames and pixels. This makes the dataset a very reliable test set for comparison of methods. However, for training convolutional networks, the dataset is still too small.

The KITTI dataset was produced in 2012 [8] and extended in 2015 [16]. It contains stereo videos of road scenes from a calibrated pair of cameras mounted on a car. Ground truth for optical flow and disparity is obtained from a 3D laser scanner combined with the egomotion data of the car. While the dataset contains real data, the acquisition method restricts the ground truth to static parts of the scene. Moreover, the laser only provides sparse data up to a certain distance and height. For the most recent version, 3D models of cars were fitted to the point clouds to obtain a denser labeling and to also include moving objects. However, the ground truth in these areas is still an approximation.

Dosovitskiy et al. [4] trained convolutional networks for optical flow estimation on a synthetic dataset of moving 2D chair images superimposed on natural background images. This dataset is large but limited to single-view optical flow and does *not* contain any 3D motions.

Both the latest Sintel dataset and the KITTI dataset can be used to estimate scene flow with some restrictions. In occluded areas (visible in one frame but not in the other), ground truth for scene flow is not available. On KITTI, the most interesting component of scene flow, namely the 3D motion of foreground points, is missing or approximated via fitted CAD models of cars. A comprehensive overview of the most important comparable datasets and their features is given in Table 1.

**Convolutional networks.** Convolutional networks [15] have proven very successful for a variety of recognition

[3] https://www.blender.org/

[4] http://lmb.informatik.uni-freiburg.de/resources/datasets/

tasks, such as image classification [14]. Recent applications of convolutional networks include also depth estimation from single images [6], stereo matching [27], and optical flow estimation [4].

The FlowNet of Dosovitskiy et al. [4] is most related to our work. It uses an encoder-decoder architecture with additional crosslinks between contracting and expanding network parts, where the encoder computes abstract features from receptive fields of increasing size, and the decoder reestablishes the original resolution via an expanding upconvolutional architecture [5]. We adapt this approach for disparity estimation.

The disparity estimation method in Žbontar et al. [27] uses a Siamese network for computing matching distances between image patches. To finally estimate the disparity, the authors then perform cross-based cost aggregation [28] and semi-global matching (SGM) [10]. In contrast to our work, Žbontar et al. have no end-to-end training of a convolutional network on the disparity estimation task, with corresponding consequences for computational efficiency and elegance.

**Scene flow.** While there are hundreds of papers on disparity estimation and optical flow estimation, there are only a few on scene flow. None of them uses a learning approach.

Scene flow estimation was popularized for the first time by the work of Vedula et al. [22] who analyzed different possible problem settings. Later works were dominated by variational methods. Huguet and Devernay [11] formulated scene flow estimation in a joint variational approach. Wedel et al. [25] followed the variational framework but decoupled the disparity estimation for larger efficiency and accuracy. Vogel et al. [24] combined the task of scene flow estimation with superpixel segmentation using a piecewise rigid model for regularization. Quiroga et al. [17] extended the regularizer further to a smooth field of rigid motion. Like Wedel et al. [25] they decoupled the disparity estimation and replaced it by the depth values of RGBD videos.

The fastest method in KITTI's scene flow top 7 is from Cech et al. [3] with a runtime of 2.4 seconds. The method employs a seed growing algorithm for simultaneous disparity and optical flow estimation.

## 3. Definition of scene flow

Optical flow is a projection of the world's 3D motion onto the image plane. Commonly, *scene flow* is considered as the underlying 3D motion field that can be computed from stereo videos or RGBD videos. Assume two successive time frames $t$ and $t+1$ of a stereo pair, yielding four images ($I_L^t$, $I_R^t$, $I_L^{t+1}$, $I_R^{t+1}$). Scene flow provides for each visible point in one of these four images the point's 3D position and its 3D motion vector [23].

These 3D quantities can be computed only in the case of known camera intrinsics and extrinsics. A camera-independent definition of scene flow is obtained by the separate components *optical flow*, the *disparity*, and the *disparity change* [11], cf. Fig. 2. This representation is complete in the sense that the visible 3D points and their 3D motion vectors can be computed from the components if the camera parameters are known.

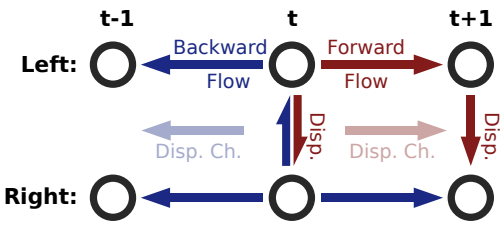



Figure 2. Given stereo images at times $t-1$, $t$ and $t+1$, the arrows indicate disparity and flow relations between them. The red components are commonly used to estimate scene flow. In our datasets we provide all relations including the blue arrows.

Given the disparities at $t$ and $t+1$, the disparity change is almost redundant. Thus, in the KITTI 2015 scene flow benchmark [16], only optical flow and disparities are evaluated. In this case, scene flow can be reconstructed only for surface points that are visible in both the left and the right frame. Especially in the context of convolutional networks, it is particularly interesting to estimate also depth and motion in partially occluded areas. Moreover, reconstruction of the 3D motions from flow and disparities is more sensitive to noise, because a small error in the optical flow can lead to a large error in the 3D motion vector.

## 4. Three rendered datasets

We created a synthetic dataset suite that consists of three subsets and provides the complete ground truth scene flow (incl. disparity change) in forward and backward direction. To this end, we used the open source 3D creation suite Blender to animate a large number of objects with complex motions and to render the results into tens of thousands of frames. We modified the pipeline of Blender's internal render engine to produce – besides stereo RGB images – three additional data passes per frame and view. These provide 3D positions of all visible surface points, as well as their future and past 3D positions. The pixelwise difference between two such data passes for a given camera view results in an "image" of 3D motion vectors – the complete scene flow ground truth as seen by this camera. Note that the information is complete even in occluded regions since the render engine always has full knowledge about all (visible and invisible) scene points.

All non-opaque materials – notably, most car windows – were rendered as fully transparent to avoid consistency problems in the 3D data. To prevent layer blending artifacts,

we rendered all non-RGB data without antialiasing.

Given the intrinsic camera parameters (focal length, principal point) and the render settings (image size, virtual sensor size and format), we project the 3D motion vector of each pixel into a 2D pixel motion vector coplanar to the imaging plane: the optical flow. Depth is directly retrieved from a pixel's 3D position and converted to disparity using the known configuration of the virtual stereo rig. We compute the *disparity change* from the depth component of the 3D motion vector. Examples are shown in Fig. 1,3,8.

In addition, we rendered object segmentation masks in which each pixel's value corresponds to the unique index of its object. Objects can consist of multiple subparts, of which each can have a separate *material* (with own appearance properties such as textures). We make use of this and render additional segmentation masks, where each pixel encodes its material's index. The recently available beta version of Sintel also includes this data.

Similar to the Sintel dataset, we also provide object and material segmentations, as well as *motion boundaries* which highlight pixels between at least two moving objects, if the following holds: the difference in motion between the objects is at least 1.5 pixels, and the boundary segment covers an area of at least 10 pixels. The thresholds were chosen to match the results of Sintel's segmentation.

For all frames and views, we provide the full camera intrinsics and extrinsics matrices. Those can be used for structure from motion or other tasks that require camera tracking. We rendered all image data using a virtual focal length of 35mm on a 32mm wide simulated sensor. For the *Driving* dataset we added a wide-angle version using a focal length of 15mm which is visually closer to the existing KITTI datasets.

Like the Sintel dataset, our datasets also include two distinct versions of every image: the *clean pass* shows colors, textures and scene lighting but no image degradations, while the *final pass* additionally includes postprocessing effects such as simulated depth-of-field blur, motion blur, sunlight glare, and gamma curve manipulation.

To handle the massive amount of data (2.5TB), we compressed all RGB image data to the lossy but high-quality WebP[5] format (we provide both WebP and lossless PNG versions). Non-RGB data was compressed losslessly.

### 4.1. FlyingThings3D

The main part of the new data collection consists of everyday objects flying along randomized 3D trajectories. We generated about 25000 stereo frames with ground truth data. Instead of focusing on a particular task (like KITTI) or enforcing strict naturalism (like Sintel), we rely on randomness and a large pool of rendering assets to generate orders of magnitude more data than any existing option, without running a risk of repetition or saturation. Data generation is fast, fully automatic, and yields dense accurate ground truth for the complete scene flow task. The motivation for creating this dataset is to facilitate training of large convolutional networks, which should benefit from the large variety.

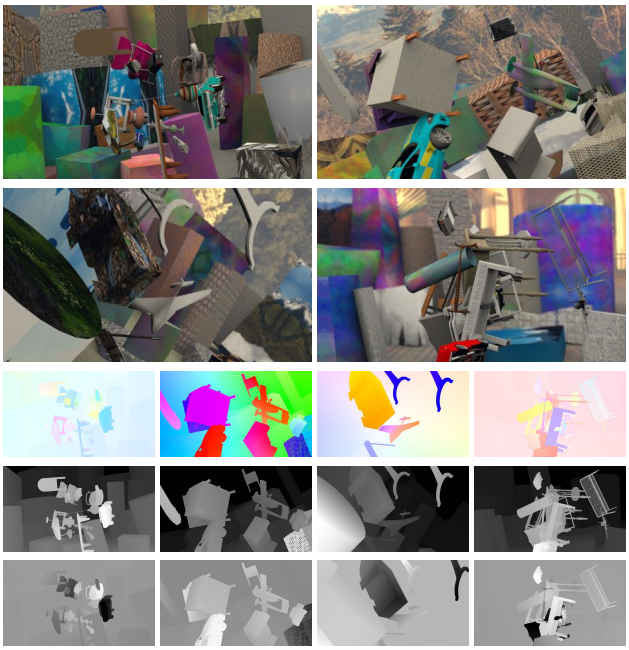


Figure 3. Example scenes from our *FlyingThings3D* dataset. **3rd row:** Optical flow images, **4th row:** Disparity images, **5th row:** Disparity change images. Best viewed on a color screen in high resolution (data images normalized for display).

The base of each scene is a large textured ground plane. We generated 200 static background objects with shapes that were randomly chosen from cuboids and deformed cylinders. Each object was randomly scaled, rotated, textured and then placed on the ground plane.

To populate the scene, we downloaded 35927 detailed 3D models from Stanford's ShapeNet[6] [18] database. From these we assembled a training set of 32872 models and a testing set of size 3055 (model categories are disjoint).

We sampled between 5 and 20 random objects from this object collection and randomly textured every material of every object. The camera and all ShapeNet objects were translated and rotated along linear 3D trajectories modeled such that the camera can see the objects, but with randomized displacements.

The texture collection was a combination of procedural images created using ImageMagick[7], landscape and cityscape photographs from Flickr[8], and texture-style pho-

[5] https://developers.google.com/speed/webp/

[6] http://shapenet.cs.stanford.edu/

[7] http://www.imagemagick.org/script/index.php

[8] https://www.flickr.com/ Non-commercial public license. We used the code framework by Hays and Efros [9]

KITTI 2015 *Driving* (ours)

Figure 4. Example frames from the 2015 version of the KITTI benchmark suite [16] and our new *Driving* dataset. Both show many static and moving cars from various realistic viewpoints, thin objects, complex shadows, textured ground, and challenging specular reflections.

tographs from Image*After[9]. Like the 3D models, also the textures were split into disjoint training and testing parts.

For the final pass images, the scenes vary in presence and intensity of motion blur and defocus blur.

### 4.2. Monkaa

The second part of our dataset is made from the open source Blender assets of the animated short film Monkaa[10]. In this regard, it resembles the MPI Sintel dataset. Monkaa contains nonrigid and softly articulated motion as well as visually challenging fur. Beyond that, there are few visual similarities to Sintel; the Monkaa movie does not strive for the same amount of naturalism.

We selected a number of suitable movie scenes and additionally created entirely new scenes using parts and pieces from Monkaa. To increase the amount of data, we rendered our selfmade scenes in multiple versions, each with random incremental changes to the camera's rotation and motion path.

### 4.3. Driving

The *Driving* scene is a mostly naturalistic, dynamic street scene from the viewpoint of a driving car, made to resemble the KITTI datasets. It uses car models from the same pool as the *FlyingThings3D* dataset and additionally employs highly detailed tree models from 3D Warehouse[11] and simple street lights. In Fig. 4 we show selected frames from *Driving* and lookalike frames from KITTI 2015.

Our stereo baseline is set to 1 Blender unit, which together with a typical car model width of roughly 2 units is comparable to KITTI's setting (54cm baseline, 186cm car width [8]).

[9] http://www.imageafter.com/textures.php
[10] https://cloud.blender.org/bi/monkaa/
[11] https://3dwarehouse.sketchup.com/

## 5. Networks

To prove the applicability of our new synthetic datasets to scene flow estimation, we use them to train convolutional networks. In general, we follow the architecture of the FlowNet [4]: each network consists of a contractive part and an expanding part with long-range links between them. The contracting part contains convolutional layers with occasional strides of 2, resulting in a total downsampling factor of 64. This allows the network to estimate large displacements. The expanding part of the network then gradually and nonlinearly upsamples the feature maps, taking into account also the features from the contractive part. This is done by a series of up-convolutional and convolutional layers. Note that there is no data bottleneck in the network, as information can also pass through the long-range connections between contracting and expanding layers. For an illustration of the overall architecture we refer to the figures in Dosovitskiy et al. [4].

For disparity estimation we propose the basic architecture *DispNet* described in Table 2. We found that additional convolutions in the expanding part yield smoother disparity maps than the FlowNet architecture (see Fig. 6).

We also tested an architecture that makes use of an explicit correlation layer [4], which we call *DispNetCorr*. In this network, the two images are processed separately up to layer conv2 and the resulting features are then correlated horizontally. We consider a maximum displacement of 40 pixels, which corresponds to 160 pixels in the input image. Compared to the 2D correlation in Dosovitskiy et al. [4], 1D correlation is computationally much cheaper and allows us to cover larger displacements with finer sampling than in the FlowNet, which used a stride of 2 for the correlation.

We also train a FlowNet and a joint SceneFlowNet for scene flow estimation by combining and fine-tuning pretrained networks for disparity and flow. This is illustrated in Figure 5. A FlowNet predicts flow between the left and right image and two DispNets predict the disparities at $t$ and $t+1$. The networks in this case do not contain correlation layers and convolutions between up-convolutions. We then fine-tune the large combined network to estimate flow, disparity, and additionally disparity change.

**Training.** All networks are trained end-to-end, given the images as input and the ground truth (optical flow, disparity, or scene flow) as output. We employ a custom version of Caffe [12] and make use of the Adam optimizer [13]. We set $\beta_1 = 0.9$ and $\beta_2 = 0.999$ as in Kingma et al. [13]. As learning rate we used $\lambda = 0.0001$ and divided it by 2 every 200k iterations starting from iteration 400k.

Due to the depth of the networks and the direct connections between contracting and expanding layers (see Table 2), lower layers get mixed gradients if all six losses are active. We found that using a loss weight schedule can be beneficial: we start training with a loss weight of 1 assigned

| Method | KITTI 2012 | | KITTI 2015 | | Driving | FlyingThings3D | Monkaa | Sintel | Time |
|---|---|---|---|---|---|---|---|---|---|
| | train | test | train | test (D1) | clean | clean test | clean | clean train | |
| DispNet | 2.38 | — | 2.19 | — | 15.62 | 2.02 | 5.99 | 5.38 | 0.06s |
| DispNetCorr1D | 1.75 | — | 1.59 | — | 16.12 | 1.68 | 5.78 | 5.66 | 0.06s |
| DispNet-K | 1.77 | — | (0.77) | — | 19.67 | 7.14 | 14.09 | 21.29 | 0.06s |
| DispNetCorr1D-K | 1.48 | 1.0† | (0.68) | 4.34% | 20.40 | 7.46 | 14.93 | 21.88 | 0.06s |
| SGM | 10.06 | — | 7.21 | 10.86% | 40.19 | 8.70 | 20.16 | 19.62 | 1.1s |
| MC-CNN-fst | — | — | — | 4.62% | 19.58 | 4.09 | 6.71 | 11.94 | 0.8s |
| MC-CNN-acrt | — | 0.9 | — | 3.89% | — | — | — | — | 67s |

Table 3. Disparity errors. All measures are endpoint errors, except for the *D1-all* measure (see the text for explanation) for KITTI 2015 test. †This result is from a network fine-tuned on KITTI 2012 train.

| Name | Kernel | Str. | Ch I/O | InpRes | OutRes | Input |
|---|---|---|---|---|---|---|
| conv1 | 7×7 | 2 | 6/64 | 768×384 | 384×192 | images |
| conv2 | 5×5 | 2 | 64/128 | 384×192 | 192×96 | conv1 |
| conv3a | 5×5 | 2 | 128/256 | 192×96 | 96×48 | conv2 |
| conv3b | 3×3 | 1 | 256/256 | 96×48 | 96×48 | conv3a |
| conv4a | 3×3 | 2 | 256/512 | 96×48 | 48×24 | conv3b |
| conv4b | 3×3 | 1 | 512/512 | 48×24 | 48×24 | conv4a |
| conv5a | 3×3 | 2 | 512/512 | 48×24 | 24×12 | conv4b |
| conv5b | 3×3 | 1 | 512/512 | 24×12 | 24×12 | conv5a |
| conv6a | 3×3 | 2 | 512/1024 | 24×12 | 12×6 | conv5b |
| conv6b | 3×3 | 1 | 1024/1024 | 12×6 | 12×6 | conv6a |
| pr6+loss6 | 3×3 | 1 | 1024/1 | 12×6 | 12×6 | conv6b |
| upconv5 | 4×4 | 2 | 1024/512 | 12×6 | 24×12 | conv6b |
| iconv5 | 3×3 | 1 | 1025/512 | 24×12 | 24×12 | upconv5+pr6+conv5b |
| pr5+loss5 | 3×3 | 1 | 512/1 | 24×12 | 24×12 | iconv5 |
| upconv4 | 4×4 | 2 | 512/256 | 24×12 | 48×24 | iconv5 |
| iconv4 | 3×3 | 1 | 769/256 | 48×24 | 48×24 | upconv4+pr5+conv4b |
| pr4+loss4 | 3×3 | 1 | 256/1 | 48×24 | 48×24 | iconv4 |
| upconv3 | 4×4 | 2 | 256/128 | 48×24 | 96×48 | iconv4 |
| iconv3 | 3×3 | 1 | 385/128 | 96×48 | 96×48 | upconv3+pr4+conv3b |
| pr3+loss3 | 3×3 | 1 | 128/1 | 96×48 | 96×48 | iconv3 |
| upconv2 | 4×4 | 2 | 128/64 | 96×48 | 192×96 | iconv3 |
| iconv2 | 3×3 | 1 | 193/64 | 192×96 | 192×96 | upconv2+pr3+conv2 |
| pr2+loss2 | 3×3 | 1 | 64/1 | 192×96 | 192×96 | iconv2 |
| upconv1 | 4×4 | 2 | 64/32 | 192×96 | 384×192 | iconv2 |
| iconv1 | 3×3 | 1 | 97/32 | 384×192 | 384×192 | upconv1+pr2+conv1 |
| pr1+loss1 | 3×3 | 1 | 32/1 | 384×192 | 384×192 | iconv1 |

Table 2. Specification of DispNet architecture. The contracting part consists of convolutions *conv1* to *conv6b*. In the expanding part, upconvolutions (*upconvN*), convolutions (*iconvN*, *prN*) and loss layers are alternating. Features from lower layers are concatenated with higher layer features. The predicted disparity image is output by *pr1*.

to the lowest resolution loss *loss6* and a weight of 0 for all other losses (that is, all other losses are switched off). During training, we progressively increase the weights of losses with higher resolution and deactivate the low resolution losses. This enables the network to first learn a coarse representation and then proceed with finer resolutions without losses constraining intermediate features.

**Data augmentation.** Despite the large training set, we chose to perform data augmentation to introduce more diversity into the training data at almost no extra cost[12]. We perform spatial (rotation, translation, cropping, scaling) and chromatic transformations (color, contrast, brightness), and we use the same transformation for all 2 or 4 input images.

[12]The computational bottleneck is in reading the training samples from disk, whereas data augmentation is performed on the fly.

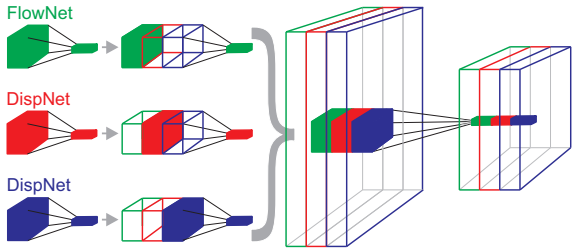



Figure 5. Interleaving the weights of a FlowNet (green) and two DispNets (red and blue) to a SceneFlowNet. For every layer, the filter masks are created by taking the weights of one network (left) and setting the weights of the other networks to zero, respectively (middle). The outputs from each network are then concatenated to yield one big network with three times the number of inputs and outputs (right).

For disparity, any rotation or vertical shift would break the epipolar constraint, and horizontal shifts between stereo views could lead to negative disparities.

## 6. Experiments

**Evaluation of existing methods.** We evaluated several existing disparity methods on our new dataset. Namely, for disparity we evaluate the state-of-the-art method of Žbontar and LeCun [27] and the popular Semi-Global Matching [10] approach with a block matching implementation from OpenCV[13]. Results are shown together with those of our DispNets in Table 3. We use the endpoint error (EPE) as error measure in most cases, with the only exception of KITTI 2015 test set where only the *D1-all* error measure is reported by the KITTI evaluation server (percentage of pixels with estimation error $> 3$px and $> 5\%$ of the true disparity).

**DispNet.** We train DispNets on the FlyingThings3D dataset and then optionally fine-tune on KITTI. The fine-

[13]http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#stereosgbm

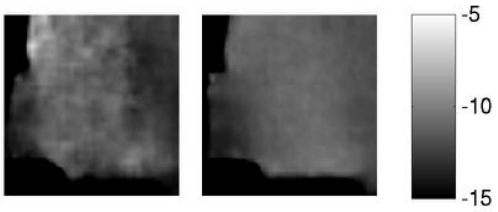



Figure 6. Close-up of a predicted disparity map without (left) and with (right) convolutions between up-convolutions. Note how the prediction on the right is much smoother.

tuned networks are denoted by a '-K' suffix in the table. At submission time, DispNetCorr fine-tuned on KITTI 2015 was second best in the KITTI 2015 top results table, slightly behind MC-CNN-acrt [27] but being roughly 1000 times faster. On KITTI resolution it runs at 15 frames per second on an Nvidia GTX Titan X GPU. For foreground pixels (belonging to car models) our error is roughly half that of [27]. The network achieves an error that is $\sim 30\%$ lower than the best real-time method reported in the table, Multi-Block-Matching [7]. Also on the other datasets DispNet performs well and outperforms both SGM and MC-CNN.

While fine-tuning on KITTI improves the results on this dataset, it increases errors on other datasets. We explain this significant performance drop by the fact that KITTI 2015 only contains relatively small disparities, up to roughly 150 pixels, while the other datasets contain some disparities of 500 pixels and more. When fine-tuned on KITTI, the network seems to lose its ability to predict large displacements, hence making huge errors on these.

We introduced several modifications to the network architecture compared to the FlowNet [4]. First, we added convolutional layers between up-convolutional layers in the expanding part of the network. As expected, this allows the network to better regularize the disparity map and predict smoother results, as illustrated in Fig. 6. The result is a $\sim 15\%$ relative EPE decrease on KITTI 2015.

Second, we trained a version of our network with a 1D correlation layer. In contrast to Dosovitskiy et al. [4], we find that networks with correlation in many cases improve the performance (see Table 3). A likely plausible explanation is that the 1D nature of the disparity estimation problem allows us to compute correlations at a finer grid than the FlowNet.

**SceneFlowNet.** As mentioned in Sec. 5, to construct a SceneFlowNet, we first train a FlowNet and a DispNet, then combine them as described in Fig. 5 and train the combination. Table 4 shows the results of the initial networks and the SceneFlowNet. We observe that solving the joint task yields better results than solving the individual tasks. The final results on our datasets are given in Table 5 and a qualitative example from FlyingThings3D is shown in Fig. 8.

| | Flow | Disparity | Disp. Ch |
|---|---|---|---|
| FlowNet | 13.78 | | |
| DispNet | | 2.41 | |
| FlowNet +500k | 12.18 | | |
| DispNet +500k | | 2.37 | |
| SceneFlowNet +500k | 10.99 | 2.21 | 0.79 |

Table 4. Performance of solving the single tasks compared to solving the joint scene flow task, trained and tested on FlyingThings3D. FlowNet was initially trained for 1.2M and DispNet for 1.4M iterations, +500k denotes training for 500k more iterations. The SceneFlowNet is initialized with the FlowNet and DispNet. Solving the joint task yields better results in each individual task.

| SceneFlowNet | Driving | FlyingThings3D | Monkaa |
|---|---|---|---|
| Flow | 23.53 | 10.99 | 6.54 |
| Disparity | 15.35 | 2.21 | 6.59 |
| Disp. change | 16.34 | 0.80 | 0.78 |

Table 5. Endpoint errors for the evaluation of our SceneFlowNet on the presented datasets. The Driving dataset contains the largest disparities, flows and disparity changes, resulting in large errors. The FlyingThings3D dataset contains large flows, while Monkaa contains smaller flows and larger disparities.

Although the FlyingThings3D dataset is more sophisticated than the FlyingChairs dataset, training on this dataset did not yield a FlowNet that performs better than training on FlyingChairs. Notwithstanding the fact that FlyingThings3D, in contrast to FlyingChairs, offers the possibility to train networks for disparity and scene flow estimation, we are investigating how 3D datasets can also improve the performance of FlowNet.

## 7. Conclusion

We have introduced a synthetic dataset containing over 35000 stereo image pairs with ground truth disparity, optical flow, and scene flow. While our motivation was to create a sufficiently large dataset that is suitable to train convolutional networks to estimate these quantities, the dataset can also serve for evaluation of other methods. This is particularly interesting for scene flow, where there has been a lack of datasets with ground truth.

We have demonstrated that the dataset can indeed be used to successfully train large convolutional networks: the network we trained for disparity estimation is on par with the state of the art and runs 1000 times faster. A first approach of training the network for scene flow estimation using a standard network architecture also shows promising results. We are convinced that our dataset will help boost deep learning research for such challenging vision tasks as stereo, flow and scene flow estimation.

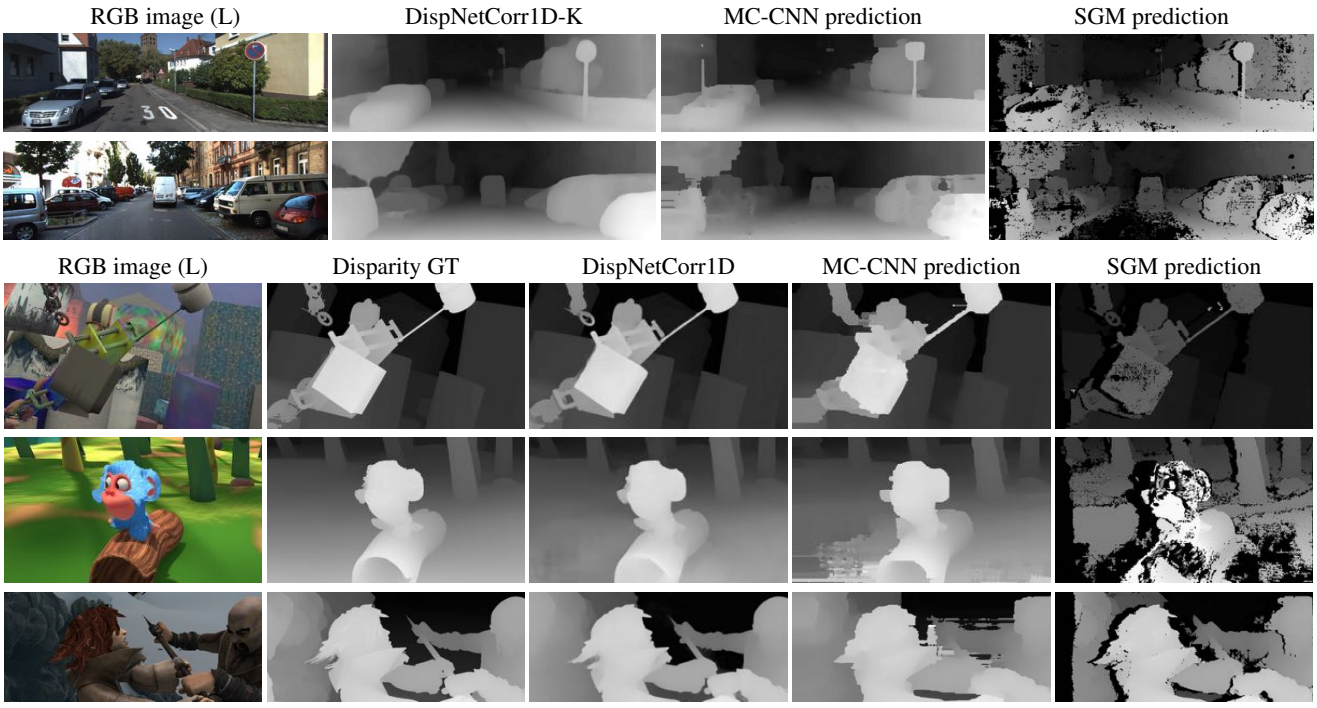



Figure 7. Disparity results. Rows from top to bottom: KITTI 2012, KITTI 2015, FlyingThings3D (clean), Monkaa (clean), Sintel (clean). Note how the DispNet prediction is basically noise-free.

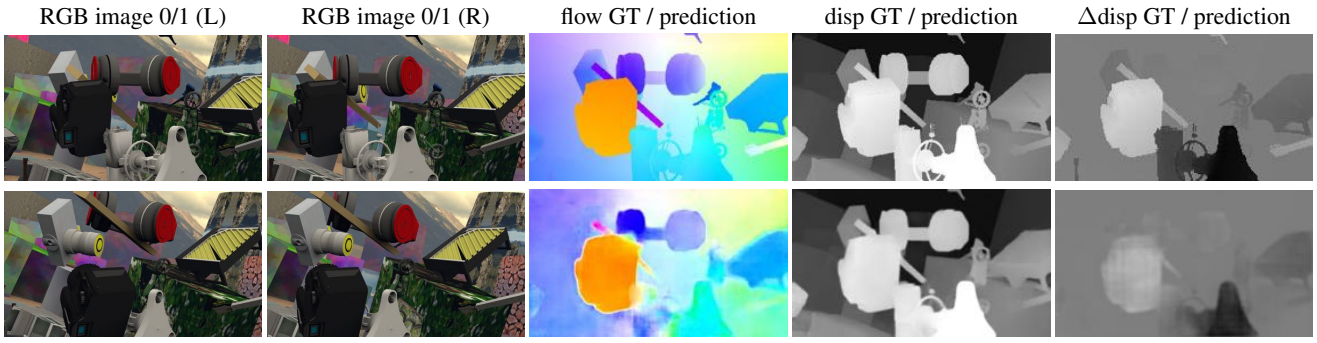



Figure 8. Results of our SceneFlowNet created from pretrained FlowNet and DispNets. The disparity change was added and the network was fine-tuned on FlyingThings3D for 500k iterations.

## 8. Acknowledgements

The work was partially funded by the ERC Starting Grant VideoLearn, the ERC Consolidator Grant 3D Reloaded, and by the DFG Grants BR 3815/7-1 and CR 250/13-1.

## References

[1] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. Technical Report MSR-TR-2009-179, December 2009.

[2] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, Part IV, LNCS 7577, pages 611–625, Oct. 2012.

[3] J. Cech, J. Sanchez-Riera, and R. P. Horaud. Scene flow estimation by growing correspondence seeds. In *CVPR*, 2011.

[4] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *ICCV*, 2015.

[5] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015.

[6] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction

from a single image using a multi-scale deep network. *NIPS*, 2014.

[7] N. Einecke and J. Eggert. A multi-block-matching approach for stereo. In *Intelligent Vehicles Symposium*, pages 585–592, 2015.

[8] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[9] J. Hays and A. A. Efros. im2gps: estimating geographic information from a single image. In *CVPR*, 2008.

[10] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *PAMI*, 30(2):328–341, 2008.

[11] F. Huguet and F. Deverney. A variational method for scene flow estimation from stereo sequences. In *ICCV*, 2007.

[12] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[16] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[17] J. Quiroga, F. Devernay, and J. Crowley. Scene flow by tracking in intensity and depth data. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 50–57. IEEE, 2012.

[18] M. Savva, A. X. Chang, and P. Hanrahan. Semantically-Enriched 3D Models for Common-sense Knowledge. *CVPR 2015 Workshop on Functionality, Physics, Intentionality and Causality*, 2015.

[19] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *Pattern Recognition*, pages 31–42. Springer, 2014.

[20] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[21] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[22] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):475–480, 2005.

[23] S. Vedula, S. Baker, P. Rander, R. T. Collins, and T. Kanade. Three-dimensional scene flow. In *ICCV*, pages 722–729, 1999.

[24] C. Vogel, K. Schindler, and S. Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

[25] A. Wedel, T. Brox, T. Vaudrey, C. Rabe, U. Franke, and D. Cremers. Stereoscopic scene flow computation for 3d motion understanding. *International Journal of Computer Vision*, 95(1):29–51, 2010.

[26] J. Xiao, A. Owens, and A. Torralba. Sun3d: A database of big spaces reconstructed using sfm and object labels. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1625–1632, Dec 2013.

[27] J. Žbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *arXiv preprint arXiv:1510.05970*, 2015.

[28] K. Zhang, J. Lu, and G. Lafruit. Cross-based local stereo matching using orthogonal integral images. *IEEE Trans. Circuits Syst. Video Techn.*, 19(7):1073–1079, 2009.

# 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 subnetwork 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 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.

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.

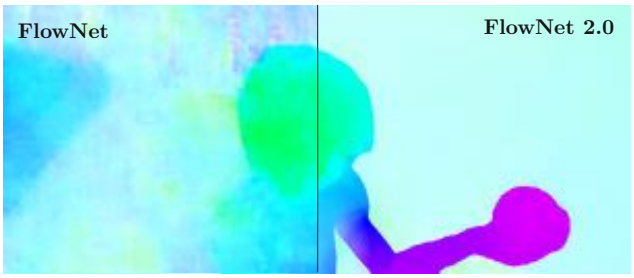



Figure 1. We present an extension of FlowNet. FlowNet 2.0 yields smooth flow fields, preserves fine motion details and runs at 8 to 140fps. The accuracy on this example is four times higher than with the original FlowNet.

The 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.

Finally, we focus on small, subpixel motion and real-

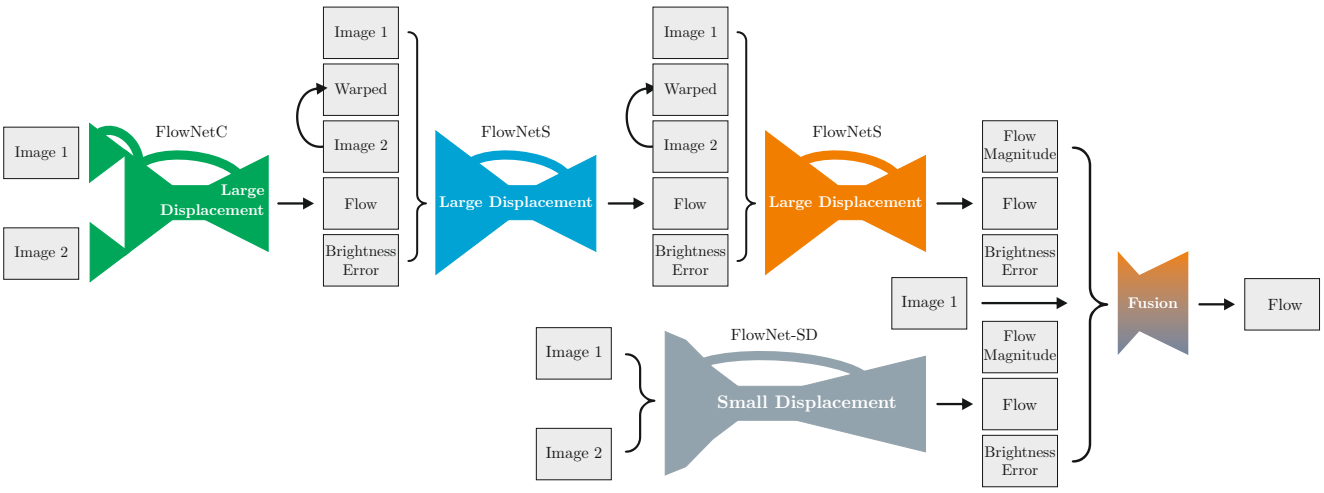



Figure 2. Schematic view of complete architecture: To compute large displacement optical flow we combine multiple FlowNets. Braces indicate concatenation of inputs. *Brightness Error* is the difference between the first image and the second image warped with the previously estimated flow. To optimally 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.

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.

## 2. Related Work

End-to-end optical flow estimation with convolutional networks was proposed by Dosovitskiy *et al*. in [10]. Their model, dubbed FlowNet, takes a pair of images as input and outputs the flow field. Following FlowNet, several papers have studied optical flow estimation with CNNs: featuring a 3D convolutional network [30], an unsupervised learning objective [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 methods significantly outperforms the original FlowNet.

An alternative approach to learning-based optical flow estimation is to use CNNs for matching image patches. Thewlis *et al*. [29] formulate Deep Matching [31] as a convolutional network 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, patch based approaches lack the possibility to use the larger context of the whole image because they operate on small image patches.

Convolutional networks trained for per-pixel prediction tasks often produce noisy or blurry results. As a remedy, out-of-the-box optimization can be applied to the network predictions as a postprocessing operation, for example, optical flow estimates can be refined with a variational approach [10]. In some cases, this refinement can be approximated by neural networks: Chen & Pock [9] formulate 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 convolutional networks 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 multiple 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 the FlyingChairs. The dataset consists of 22k renderings of random scenes showing 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 some 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.

[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.

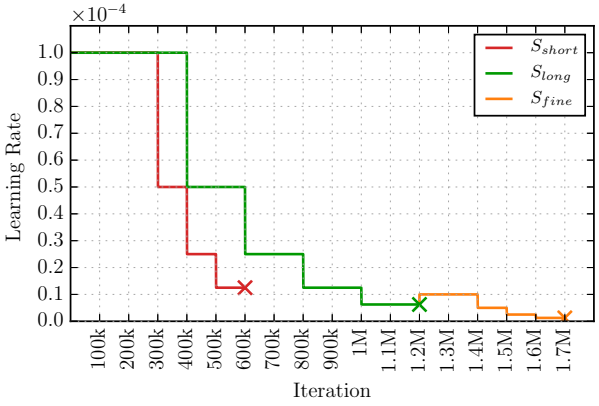



Figure 3. Learning rate schedules: $S_{short}$ is similar to the schedule in Dosovitskiy *et al*. [10]. We investigated another longer version $S_{long}$ and a fine-tuning schedule $S_{fine}$.

| 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 specialized scenarios. The trained network is rather

| 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. 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.

supposed to be generic and to work well in various scenarios. 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.

# 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 FlowNetS and FlowNetC 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, the effect of warping, and the effect 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 improves results on Chairs but decreases performance on Sintel, i.e. the stacked network is over-fitting. (2) With warping included, stacking 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 when 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

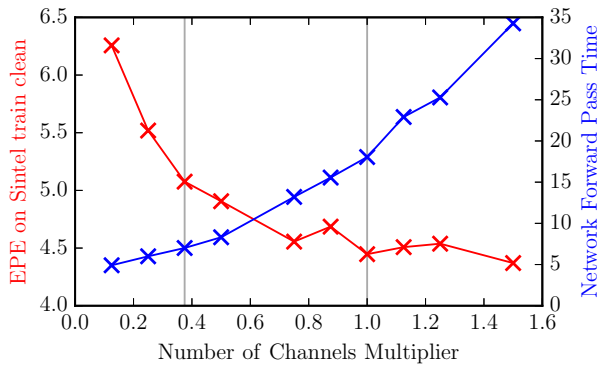



Figure 4. Accuracy and runtime of FlowNetS depending on the network width. The multiplier 1 corresponds to the width of the original FlowNet architecture. Wider networks do not improve the accuracy. For fast execution times, a factor of $\frac{3}{8}$ is a good choice. Timings are from an Nvidia GTX 1080.

| | 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.

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 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 $\sim 30\%$ over the single network FlowNet2-C from Section 3 and by $\sim 50\%$ 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 examined the training data and compared it 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* and will release it upon publication.

### 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 down-weight large displacements[2]. We denote this network by *FlowNet2-CSS-ft-sd*. This increases performance 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 $7\times7$ and $5\times5$ kernels in the beginning with multiple $3\times3$ kernels[2]. Because noise tends to be a problem with small displacements, we add convolutions between the upconvolutions to obtain smoother estimates as 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 two times 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 bechmarks. 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 a system with an Intel Xeon E5 with 2.40GHz and an Nvidia GTX 1080. Where applicable, dataset-specific parameters were used, that yield best performance. 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. 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

[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].

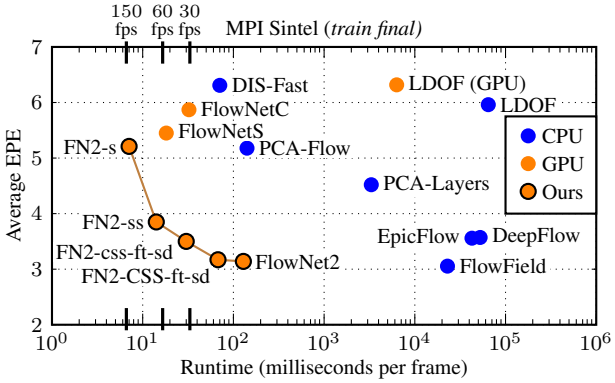



Figure 5. Runtime vs. endpoint error comparison to the fastest existing methods with available code. The FlowNet2 family outperforms other methods by a large margin. The behaviour for the KITTI dataset is the same; see supplemental material.

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. This is probably due to KITTI containing very large displacements in general. Fine-tuning on a combination of the KITTI2012 and KITTI2015 training sets reduces the error roughly by a factor of 3 (FlowNet2-ft-kitti). Among non-stereo methods we obtain the best EPE on KITTI2012 and the first rank on the KITTI2015 benchmark. This shows how well and elegantly the learning approach can integrate the prior of the driving scenario.

**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. One can observe that 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], we can see that on real world data FlowNet 2.0 clearly has advantages in terms of being robust to homogeneous regions (rows 2 and 5), image and compression artifacts (rows 3 and 4) and it yields smooth flow fields with sharp motion boundaries.

| | Method | Sintel *clean* AEE | | Sintel *final* AEE | | KITTI 2012 AEE | | KITTI 2015 | | | Middlebury AEE | | Runtime ms per frame | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *train* | *test* | *train* | *test* | *train* | *test* | AEE *train* | Fl-all *train* | Fl-all *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 | – | 70 | – |
| | 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. †*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).

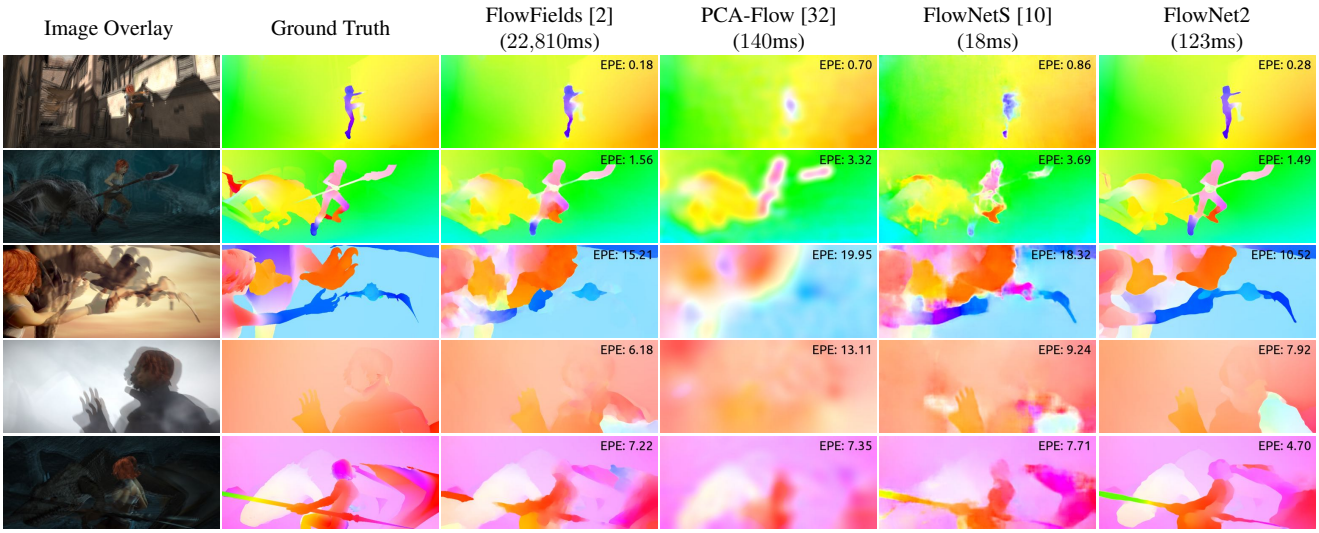



Figure 6. Examples of flow fields from different methods estimated on Sintel. FlowNet2 performs similar to FlowFields and is able to extract fine details, while methods running at comparable speeds perform much worse (PCA-Flow and FlowNetS).

### 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 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.

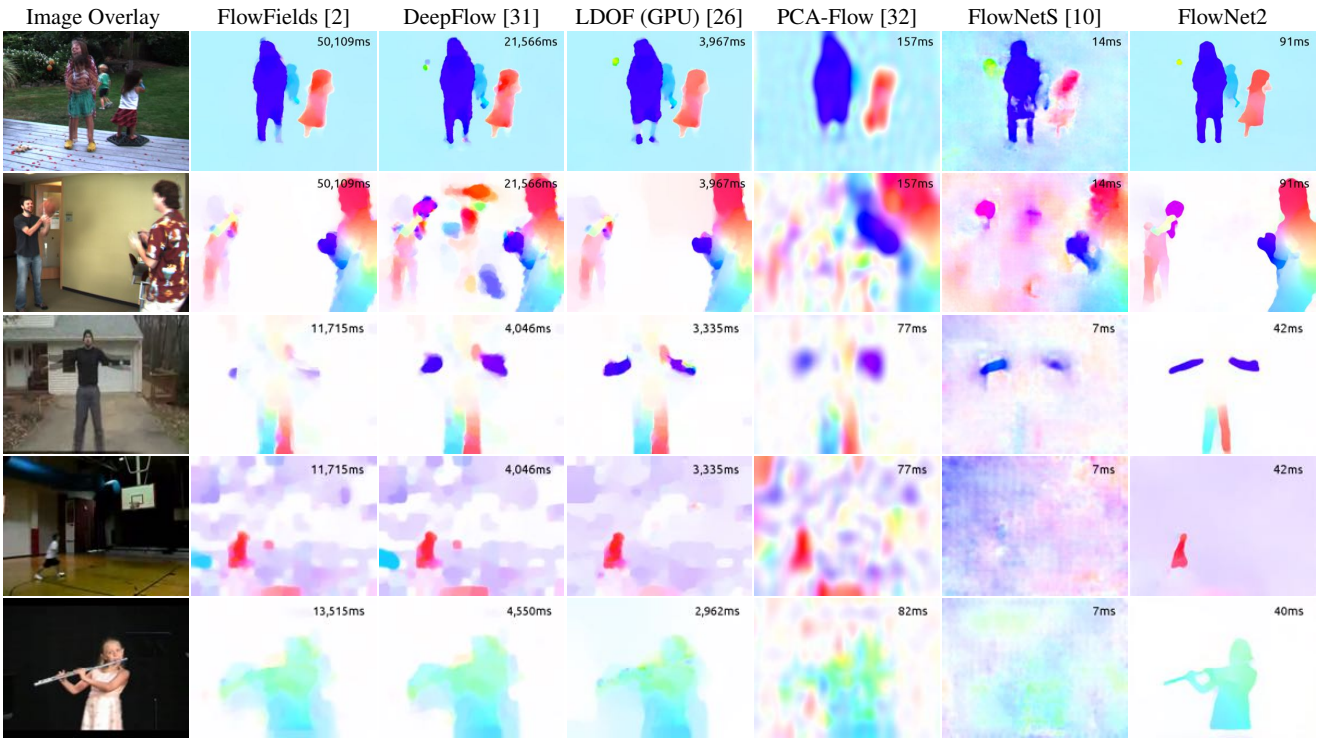



Figure 7. Examples of flow fields from different methods estimated on real-world data. The top two rows are from the Middlebury data set and the bottom three from UCF101. Note how well FlowNet2 generalizes to real-world data, i.e. it produces smooth flow fields, crisp boundaries and is robust to motion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

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.

## 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.

| | 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** | – |
| FlowNetS [10] | 56.87%‡ | 3/62‡ | 55.27% |
| FlowNet2-css-ft-sd | 77.88% | 26/65 | – |
| 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. Due to long training times only selected methods could be evaluated. †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.

## 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 International 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 International 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 *International 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 International 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 International 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 International Joint Conference on Artificial Intelligence (IJCAI)*.

[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 *International 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 International 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.