AutoFlow: Learning a Better Training Set for Optical Flow

Deqing Sun, Daniel Vlasic, Charles Herrmann, Varun Jampani, Michael Krainin, Huiwen Chang,
Ramin Zabih, William T. Freeman, and Ce Liu
Google Research

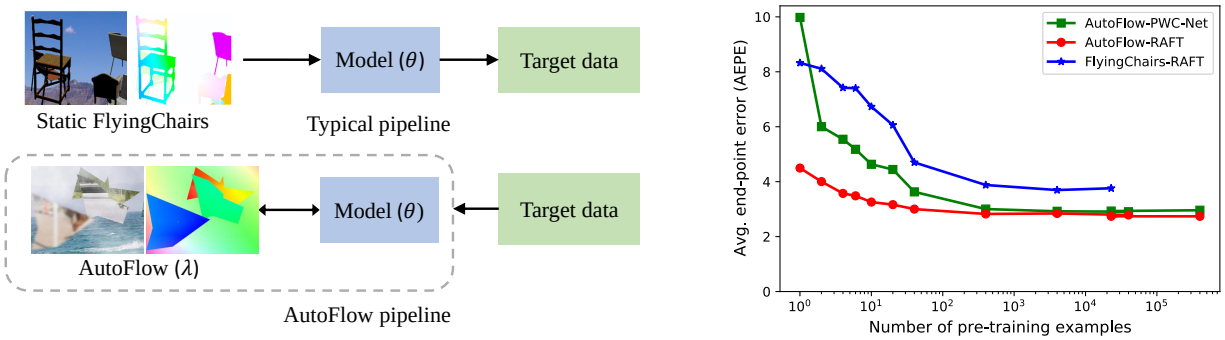

Figure 1: Left: **Pipelines for optical flow**. A typical pipeline pre-trains models on static datasets, *e.g.*, FlyingChairs, and then evaluates the performance on a target dataset, *e.g.*, Sintel. AutoFlow learns pre-training data which is optimized on a target dataset. Right: **Accuracy w.r.t. number of pre-training examples on Sintel.final.** Four AutoFlow pre-training examples with augmentation achieve lower errors than 22,872 FlyingChairs pre-training examples with augmentation. The gap between PWC-Net and RAFT becomes small when pre-trained on enough AutoFlow examples.

Abstract

Synthetic datasets play a critical role in pre-training CNN models for optical flow, but they are painstaking to generate and hard to adapt to new applications. To automate the process, we present AutoFlow, a simple and effective method to render training data for optical flow that optimizes the performance of a model on a target dataset. AutoFlow takes a layered approach to render synthetic data, where the motion, shape, and appearance of each layer are controlled by learnable hyperparameters. Experimental results show that AutoFlow achieves state-of-the-art accuracy in pre-training both PWC-Net and RAFT. Our code and data are available at `autoflow-google.github.io`.

1. Introduction

Datasets have been a driving force for the development of AI algorithms. Convolutional neural networks (CNNs) [26] were proposed in the 1990's but were not widely adopted for vision tasks until the early 2010's, with the advent of AlexNet [24]. One key ingredient for deep CNN models was the large amount of manually labeled images, *e.g.*, from ImageNet [41]. The performance gain by AlexNet over shallow models stimulated a paradigm shift in high-level vision tasks. Since then, new models have been invented in rapid succession, even achieving "superhuman" performance on image classification tasks [14, 15].

Manual labeling, however, cannot provide reliable ground truth for a variety of low-level vision tasks like optical flow and stereo. Since these labels are either difficult or impossible to obtain, synthetic data play a key role in enabling deep models to perform well on such tasks. For example, all top-performing CNN models for optical flow are pre-trained on two large synthetic datasets, FlyingChairs [8] and FlyingThings3D [32], before being fine-tuned on limited target datasets, *e.g.*, Sintel [3] and KITTI [12].

However, the success of FlyingChairs raises some interesting questions. For example, *how realistic should the rendering be?* Several new datasets have been developed to be more realistic than FlyingChairs, such as virtual KITTI [11], VIPER [38], and REFRESH [30], but none of them have proven more effective than FlyingChairs and FlyingThings3D at pre-training models. In fact, a comprehensive study has revealed that "realism is overrated" [32]. There are some hypotheses for why FlyingChairs works, *e.g.*, that it has been designed to match the motion statistics of Sintel, or that it has many thin structures and fine motion details. However, it still remains unclear what set of

principles makes an effective optical flow dataset.

To address these questions, we argue that we should make explicit the objective function for rendering training data. We formulate the generation of training data as a joint optimization problem, which couples rendering the data with training the model. This generation process depends on a set of hyperparameters being optimized. The hyperparameters are evaluated by the performance of the trained model on a target dataset, as shown in Fig. 1.

To understand what matters, we ask: *how simple can the rendering be?* Thus, we start from an even simpler rendering pipeline than FlyingChairs, a 2D layered approach that requires neither manual labeling nor 3D models. The motion and shape of each layer are randomly generated according to hyperparameters, as shown in Fig. 2. We can then learn the rendering hyperparameters to optimize the performance of a model on a target dataset.

This simple rendering pipeline is surprisingly effective at generating training datasets for optical flow. Trained on its rendered data from scratch, both the recent RAFT model and the widely-used PWC-Net model obtain consistent improvements in accuracy on Sintel and KITTI over the same models trained on FlyingChairs (Fig. 1 and Table 1). Further, using 4 AutoFlow examples with augmentation results in lower errors on Sintel.final for RAFT than using 22,872 FlyingChairs examples with augmentation. More interestingly, the gap between PWC-Net and RAFT becomes small when trained on enough AutoFlow examples.

An analysis of the rendered data also suggests some interesting properties. For example, the motion statistics of the AutoFlow dataset and its augmented version do not resemble those of Sintel (Fig. 8) and underrepresent small motions. Though at first glance this distribution may seem abnormal, there may be a simple, intuitive explanation: tiny motion matters little in the overall error.

To summarize, our contributions are the following.

- We have introduced, to our knowledge, the first learning approach to render training data for optical flow.
- AutoFlow compares favorably against FlyingChairs and FlyingThings3D in pre-training RAFT.
- AutoFlow also leads to a significant performance gain for PWC-Net, even competitive against RAFT.
- We present a detailed analysis of what features are important to dataset generation for optical flow.

2. Related Work

Datasets for high-level computer vision Manually labeled datasets, such as ImageNet [41], PASCAL [10], MS-COCO [27], and CityScapes [5], have been widely adopted for high-level vision tasks. However, manual labeling is hard to scale, and quite a few synthetic datasets have been developed [39, 9, 11, 22]. Meta-Sim [22] learns to minimize the distribution gap between the rendered and target datasets and can also optimize task performance. However, Meta-Sim can model only limited scenes because it relies on obtaining valid scene structures from a grammar.

Model	Dataset	Sintel.clean	Sintel.final	KITTI
PWC-Net	FlyingChairs	3.27	4.42	11.43
	Chairs→Things	2.39	3.90	9.81
	AutoFlow	**2.17**	**2.91**	**5.76**
RAFT	FlyingChairs	2.27	3.76	7.63
	Chairs→Things	**1.68**	2.80	5.92
	AutoFlow	1.95	**2.57**	**4.23**

Table 1: **AEPE results for pre-training**. AutoFlow can better train RAFT and PWC-Net from scratch than the widely-used FlyingChairs dataset and perform competitively against the FlyingChairs→FlyingThings3D schedule.

RenderGAN [42] learns to augment the dataset for handwriting classification. Differentiable rendering [29, 37] enables gradients to be passed to rendering parameters, which, however, do not directly relate to the scene distribution hyperparameters. Yang and Deng [50] proposed a "hybrid gradient" approach to make use of analytical gradients whenever available. These methods focus on the generation of a single image and cannot directly apply to the generation of optical flow.

Datasets for optical flow Similar to other vision tasks, datasets have been the driving force behind the development of optical flow. However, unlike high-level vision tasks, it is only possible to obtain ground truth under controlled lab environments [1] or rigid scenes/objects [12, 23]. Early work relied on synthetic datasets for evaluation, such as the well-known "Yosemite" sequence [2]. MPI-Sintel [3], one of the leading benchmark datasets for optical flow, was rendered using the Blender engine. Roth and Black [40] used real depth data to render synthetic data, which is limited to static scenes. KITTI [12] was created using LIDAR for static scenes and later extended to rigidly moving cars [33] for autonomous driving applications.

Dosovitskiy *et al*. [8] created a synthetic dataset, FlyingChairs. Mayer *et al*. [32] further introduced a large dataset for optical flow and related tasks, FlyingThings3D. Ilg *et al*. [18] found that sequentially training on FlyingChairs and then on FlyingThings3D obtains the best results; this has since become standard practice in the field. Efforts to improve these two datasets include the autonomous driving scenario [11], more realistic rendering [38], realistic backgrounds from SLAM [30], and human datasets [36]. However, none have proven more effective than FlyingChairs and FlyingThings3D for pre-training.

Mayer *et al*. [31] performed a comprehensive study of synthetic datasets for optical flow and disparity estimation. They developed each synthetic dataset heuristically, with no regard for target dataset performance. Our rendering pipeline is largely inspired by their 2D rendering techniques. But instead of designing each dataset by hand, we learn these parameters via jointly solving rendering and training to optimize the performance on a target dataset.

CNN models for optical flow The seminal FlowNet paper [8] pioneered the CNN-based approach for optical flow. Its follow-up, FlowNet2 [18], significantly improved FlowNet's performance by stacking several sub-networks into one large model. Spy-Net [35], PWC-Net [45], and LiteFlowNet [16] were designed using several well-established principles for optical flow. For the first time, PWC-Net obtained more accurate results on the Sintel and KITTI benchmarks than traditional approaches. Quite a few new network architectures were proposed based on the PWC-Net framework [17, 51, 20, 52]. Recently, Teed and Deng [48] introduced the RAFT architecture, which used a recurrent architecture to obtain a significant performance gain over its predecessors on Sintel and KITTI.

The advances in these network architectures have significantly improved their performance on benchmark datasets. However, all these models follow nearly the same training procedures, *i.e*., pre-training on FlyingChairs and FlyingThings3D and then fine-tuning on limited training data on the target domain. In this paper, we focus on dataset generation and show that it is possible to achieve accuracy similar to or better than that of FlyingChairs and FlyingThings3D in pre-training by learning to render training data. We learn the rendering hyperparameters for the recent RAFT model and find that they also apply to PWC-Net.

Evaluating CNN models for optical flow The improvement in accuracy comes from innovations on both the model architecture and the training procedures. Previous work [46] shows that changes in training procedure result in significant performance boosts for FlowNetC and PWC-Net. Here we find that changing the datasets and incorporating recent practices in training significantly improves PWC-Net and narrows down its performance gap from RAFT.

Self-supervised and semi-supervised learning of optical flow Significant progress has been made on self-supervised optical flow [21, 28]. However, state-of-the-art, self-supervised methods still lag behind supervised ones, e.g., models pre-trained on FlyingChairs and FlyingThings [48, 51] are more accurate on Sintel than models [21, 28] trained on Sintel image pairs using self-supervised loss.

Learning to learn A recent trend in neural network research is learning to learn, which aims at automating the manual process of network design or hyperparameter selection. Existing methods mainly focus on learning hyperparameters for the architecture [54], loss function, optimization, and augmentation [4, 6]. In contrast, we focus on learning to render synthetic training data for optical flow.

3. Generating training data

We take a layered approach [49, 44] to rendering image pairs and their optical flow, as shown in Fig. 2. For the first frame, we randomly sample K images $\mathbf{I}_1^k$ from an image dataset and order them by depth, with the first layer being the background. Next, we sample an alpha mask $\mathbf{M}_1^k$ (section 3.1) and an optical flow field $\mathbf{W}^k$ (section 3.2) for each layer according to the rendering hyperparameters (section 3.5). The optical flow field is used to warp the image and the mask into the second frame:

$$\begin{aligned} \mathbf{I}_2^k &= f(\mathbf{I}_1^k, \mathbf{W}^k) \qquad 1 \le k \le K, \\ \mathbf{M}_2^k &= f(\mathbf{M}_1^k, \mathbf{W}^k) \qquad 1 \le k \le K, \end{aligned} \tag{1}$$

where f represents the forward warping function according to the flow field.

We composite the images and the flow with back-to-front alpha blending, starting with the background layer:

$$\begin{aligned} \mathbf{I}^k &= \mathbf{M}^k \odot \mathbf{I}^k + \left(1 - \mathbf{M}^k\right) \odot \mathbf{I}^{k-1}, \\ \mathbf{W}^k &= \bar{\mathbf{M}}^k \odot \mathbf{W}^k + \left(1 - \bar{\mathbf{M}}^k\right) \odot \mathbf{W}^{k-1}, \end{aligned} \tag{2}$$

where $\bar{\mathbf{M}}$ is the alpha mask binarized around its middle value, and $\odot$ denotes the element-wise product and broadcasts to the channel dimension. We slightly abuse the notation, using the same symbols for images and masks before and after composition, as well as dropping subscripts.

Finally, we apply certain visual effects (section 3.3) to the images to cover some natural variations in videos. Figure 6 shows examples of complete images and their flows.

3.1. Object Masks

The background layer has a fully opaque mask. For each foreground layer, we test two ways of generating the mask: random polygons and manual segmentation.

Random polygons For each foreground layer, we generate a random polygon [31] to serve as its alpha mask. Each polygon has a random number of sides, with vertices randomly sampled in angle and radius around a center. Each polygon can also have a hole, which itself is a smaller random polygon. Further, we can control polygon smoothness through subdivision. Finally, the mask can be blurred with a Gaussian filter in order to feather its boundary (this is applied to both polygon and manual object masks). Examples of random polygon masks are shown in Fig. 3.

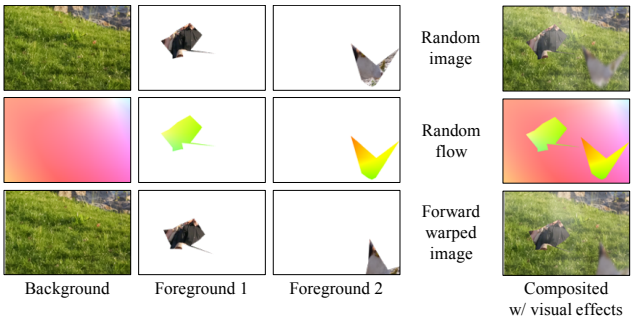

Figure 2: **Rendering pipeline** for AutoFlow uses a layered approach. Each layer is created from a random image and mask (top row), its flow field is randomly generated (middle row), and the layer is accordingly warped (bottom row). All layers are alpha-composited back-to-front and undergo visual effects such as motion blur and fog (right column).

Figure 3: **Foreground object masks** are random polygons that can have holes, smoothed edges, and blurred boundaries. Mask values have been inverted for visualization.

Manual segmentation To make foreground objects more semantically congruent, we use the images and manual labels from OpenImages [25]. The location and size of each foreground object within the image are randomly sampled.

3.2. Motion Model

The motion of each layer is a combination of rigid transformation (scale, rotation, translation), perspective distortion, and a *bilinear grid warp*. A bilinear grid warp of size n is a set of flow vectors defined on the vertices of a $n \times n$ grid, then bilinearly interpolated in the interior of the grid (the grid being uniformly distributed over the image) [47]. This allows for more complex forward flow with a fast analytic solution for forward image warping (we invert the bilinear interpolating function within each grid cell, which boils down to solving a quadratic equation). In fact, all of our base motions can be modeled with a bilinear grid warp: rigid transform can be expressed by rigidly moving the corners of a grid, and perspective distortion by independently moving the corners of a 2×2 grid.

For the foreground layers, we employ all modalities of motion (rigid + grid), while for the background we only apply moderate perspective distortion. Figure 4 demonstrates our motion modalities on a sample foreground object.

Figure 4: **Base motions** a foreground object (left) can undergo (left-to-right): rigid transformation, perspective distortion, and bilinear grid warp.

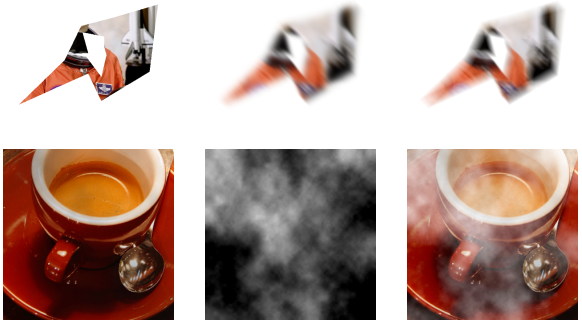

Figure 5: **Visual effects** improve performance on realistic datasets. Top row shows an object motion-blurred due to diagonal movement using Gaussian (middle) or box (right) filters. Bottom row shows a random semi-transparent fog overlaid on top of an image.

3.3. Visual Effects

To generalize better to more realistic video data, we simulate common visual effects including motion blur and fog (Fig. 5). These effects only modify the image data and have no influence over the ground truth flow.

Motion blur We approximate the motion blur of each layer by applying a filter to both the image and the mask. Standard deviations of the filter are computed by taking a proportion of the average absolute flow in each dimension over all the pixels within the mask. We apply the same motion blur filter to both the first and second images.

Fog To simulate fog, we generate a white image with a random semi-transparent alpha mask and overlay it on top of the composited initial and final images. The fog does not move between the images, nor does it affect the ground truth flow. To compute the alpha mask, we generate several random normal images of various resolutions, with their standard deviations being inversely proportional to their resolutions. We then bicubically resample each to the desired fog resolution and sum them up. Finally, we adjust the resulting image so that its mean and standard deviation match controllable hyperparameters.

3.4. Data Augmentation

To increase the diversity of the training data, we apply data augmentation [8, 45] to the rendered data. Inspired by RandAugment [7], we randomly select several transformations among rotation, scale, squeeze, translation, and additive noise at each iteration. The number of transformations and their strength levels are hyperparameters to learn.

3.5. Hyperparameters

During training, we tune a number of hyperparameters that dictate data generation and augmentation, including the shape, size, and position of masks, the complexity and magnitude of motion, and the visual effects. Respective values are uniformly sampled from the specified ranges, and the ranges are hyperparameters to learn. Please refer to the supplementary material for the detailed list of hyperparameters.

4. Learning to Render Training Data

Given a target dataset for optical flow, *e.g.*, Sintel or KITTI, we want to learn the hyperparameters to render training data so that a CNN model trained on the rendered data has optimal performance on the target dataset. Every set of hyperparameters corresponds to a rendered training dataset. In this section, we will first present the learning objective and then the search algorithm.

4.1. Problem Formulation

Given the rendering pipeline for generating training data and the range for the rendering hyperparameters Λ, our goal is to search for the set of optimal hyperparameters λ that optimizes a metric Ω on a model θ,

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \Omega\left(\theta(\lambda)\right). \tag{3}$$

The model θ minimizes a loss function $\mathcal{L}$ on the rendered datasets according to the set of hyperparameters λ

$$\theta(\lambda) = \arg\min_{\theta} \mathcal{L}\left(\mathbf{W}(\lambda), \phi_\theta(\mathbf{I}_1(\lambda), \mathbf{I}_2(\lambda))\right), \tag{4}$$

where the model θ includes the parameters of a network ϕ_θ that maps two input images to their optical flow. Be default, we use the sequence loss function proposed by RAFT as the loss function and the average end-point error (AEPE) as the metric on the target datasets unless stated otherwise.

4.2. Hyperparameter Search Algorithm

To learn the hyperparameters for rendering the dataset, we develop a hybrid algorithm based on the population-based training (PBT) algorithm [19, 4] and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm [13]. Specifically, we classify the hyperparameters into subgroups and use the CMA-ES algorithm to search the selected subgroups of hyperparameters to optimize the learning metric. CMA-ES maintains a sampling distribution over the search space. It samples a few points, evaluates them, and updates the distribution based on the ranking of the points w.r.t. to the learning metric. The sampling distribution is a multivariate Gaussian whose covariance matrix is adapted over time. Our algorithm takes $\mathcal{N}$ iterations, with each iteration training $\mathcal{M}$ in parallel. The time complexity grows linearly w.r.t. the number of search iterations and the number of training steps per search.

5. Experimental Results

Implementation details We randomly sample images from different sequences of the Davis dataset [34] as appearances for each layer. Our baseline is a TensorFlow implementation of RAFT [43], the performance of which is similar to that of the official PyTorch implementation. Throughout this section, we refer to our method or the data it generates as AutoFlow. By default, we use the average end-point error (AEPE) on the final pass of Sintel training dataset (Sintel.final) as the learning metric because it is currently the most challenging dataset.

Empirically, it takes about 7 days to finish 8 searching iterations using 48 NVIDIA P100 GPUs, with each iteration training 8 models in parallel. Hyperparameters about 5% less accurate are often found within 2 days. Alternatively, the time can also be reduced to less than 2 days by using fewer training steps (40k) and then reusing the searched hyperparameters for the full 200k steps. That has roughly a 3% drop in accuracy on Sintel.

5.1. AutoFlow Versus the State of the Art

Pre-training results We pre-trained RAFT and PWC-Net from scratch using different datasets. The hyperparameters for AutoFlow have been learned for RAFT. As summarized in Table 1, models trained from scratch using AutoFlow are comparable to or more accurate than models trained on FlyingChairs or FlyingChairs→FlyingThings3D. As shown in Fig. 7, RAFT trained on AutoFlow can successfully recover blurry objects under large motion in the final pass of Sintel. Table 2 summarizes the errors for regions with different motion magnitude. RAFT trained on AutoFlow performs better than RAFT trained on FlyingChairs, especially in regions with large motion. Using end-point error (epe) as the learning metric results in better accuracy in regions with large motion than using angular error (ae).

Generalization across datasets We further compared with the recent DSMNet [53] method that aims at narrowing down domain gaps. DSMNet reported an F-all score of 11.2% in *non-occlusion* regions on the KITTI 2015 *training* set for a modified PWC-Net trained on FlyingThings3D and Sintel. The F-all scores by RAFT and PWC-Net trained on

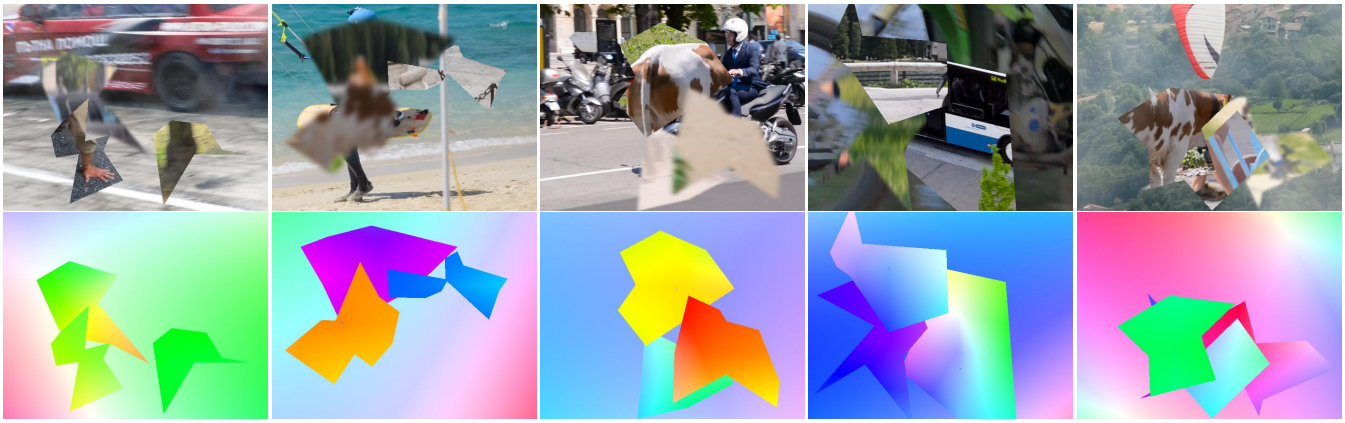

Figure 6: **Samples of AutoFlow**. Top: first images; bottom: visualized flow field.

	GT range	< 1	[1,10]	(10,20]	(20,30]	> 30
AEPE	Chairs	0.43	0.89	3.13	5.63	19.61
	AutoFlow-epe	0.35	0.65	1.87	3.36	**15.08**
	AutoFlow-ae	**0.31**	**0.63**	**1.86**	**3.24**	16.00
AAE	Chairs	10.86	6.94	6.63	9.61	13.37
	AutoFlow-epe	10.88	5.41	4.95	6.19	**10.35**
	AutoFlow-ae	**9.77**	**5.11**	**4.85**	**5.96**	10.68

Table 2: **Results in different motion ranges**. RAFT trained by AutoFlow tends to perform better for medium to large motion than RAFT trained by FlyingChairs. Using end-point error (epe) as the learning metric results in more accurate large motion than using angular error (ae).

AutoFlow that has been optimized for Sintel.final are 8.7% and 11.0%, respectively, suggesting that AutoFlow generalizes well across datasets.

Improving PWC-Net We modified the pre-training procedure of PWC-Net [45] using the one-cycle learning rate schedule and gradient clipping from RAFT [48], as summarized in Table 3. Applying gradient clipping not only improves accuracy but also makes training more stable: two out of eight runs diverged without gradient clipping.

Learning rate	Gradient clipping	Sintel clean	Sintel final	KITTI
Piecewise	✗	2.64	3.44	7.26
Piecewise	✓	2.40	3.11	6.26
One-cycle	✓	**2.17**	**2.91**	**5.76**

Table 3: **Improvements on pre-training PWC-Net**. Both one-cycle learning rate schedule and gradient clipping help.

Fine-tuning results We followed the TF-RAFT procedure to fine-tune the model pre-trained by AutoFlow and denoted the method as RAFT-A. We applied the same fine-tuned model to Sintel and KITTI, as summarized in Table 4. RAFT-A is more accurate than TF-RAFT on the more challenging Sintel.final and KITTI benchmarks, demonstrating the benefits of pre-training on AutoFlow.

Method	Dataset schedule	S.clean	S.final	KITTI
FlowNet2	C→T→S	3.96	6.02	11.48%*
PWC-Net	C→T→S	3.86	5.13	9.60%*
VCN	C→T→SKHTC	2.81	4.40	6.30%*
RAFT [48]	C→T→SKHT/K	1.94	3.18	5.10%*
TF-RAFT [43]	C→T→SKHTV	**1.84**	3.32	5.56%
RAFT-A	A→SKHTV	2.01	**3.14**	**4.78%**

Table 4: **Results on public benchmarks** (AEPE for Sintel and Fl-all for KITTI). A, C, H, K, S, and T stand for AutoFlow, FlyingChairs, HD1K, KITTI, Sintel, and FlyingThings3D, respectively. *indicates where weights for KITTI differ from those for Sintel.

5.2. Ablation Study

To further analyze AutoFlow, we performed a series of ablation studies designed to determine how different design choices affect performance. Since it is computationally expensive to learn all the hyperparameters for each setup, we fixed the learned hyperparameters unless explicitly specified. For each experiment, we ran 8 independent trials, and Table 5 summarizes the most accurate one for each setup.

Motion blur and fog Removing the motion blur effect leads to a significant drop in performance on the final pass of Sintel and KITTI, despite the rough approximation used for simulating the motion blur. Gaussian and box filters have similar results. Removing the fog effect also results in a moderate performance drop in the final pass of Sintel and KITTI. Neither motion blur nor fog effects have a significant effect on the clean pass of Sintel.

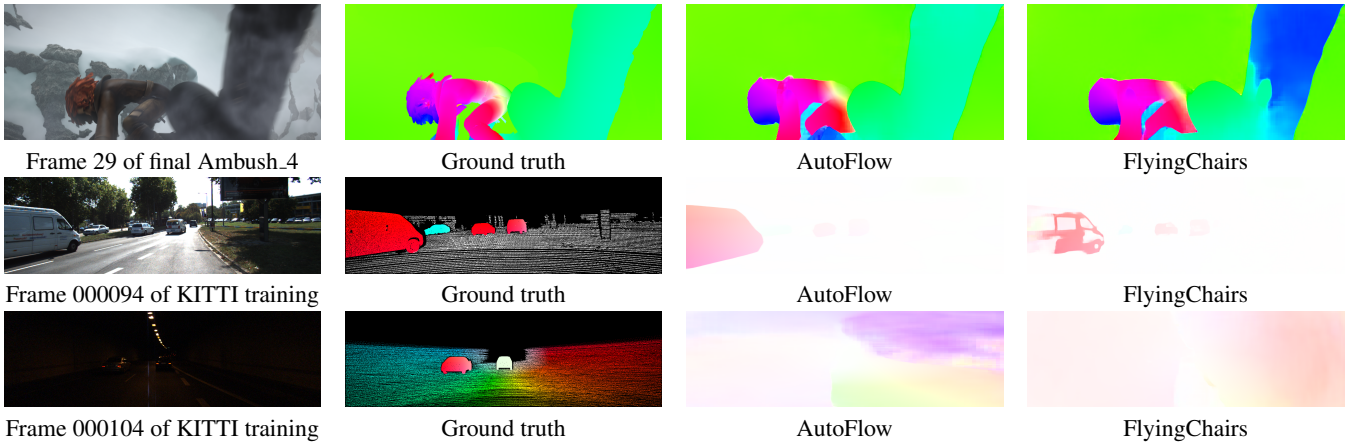

Figure 7: **Visual comparison.** Row 1: RAFT trained on AutoFlow works better for frames with strong motion blur. Row 2: RAFT trained on AutoFlow can capture the car structure. Row 3: low levels of light cause both methods to struggle.

Appearance We tested three different image sources for the appearance image of each layer: Davis, OpenImages [25], and Sintel (*c.f*. Table 5). Neither OpenImages nor Sintel achieves better results than Davis. By default, we downsample Davis images to 1280×720 (720p) resolution as appearance images for each layer. Downsampling to 960×540 (540p) has similar results while 1920×1080 (1080p) has degraded performance, likely because the hyperparameters have been learned for the 720p resolution.

Foreground object masks We tested three versions of masks for the foreground objects: random polygons with sharp edges, random polygons with smooth edges (default), and instance segmentation from the OpenImage [25] dataset. Polygons with smooth edges perform consistently better than those with sharp edges. We also experimented with instance segmentation from the OpenImage dataset due to its diverse set of segmentation masks, but there we only observed a small improvement on Sintel.clean.

Number of foreground objects The number of foreground objects determines the complexity of a scene. Using only a background layer, *i.e*., 0 foreground object, results in large errors. Adding one foreground object significantly improves the performance. Using three or four foreground objects tends to work best, while more than four foreground objects bring no further gain.

Motion model Removing the bilinear grid warping results in a performance degradation on both Sintel and KITTI, suggesting that more complex and flexible motion than parametric motion is critical.

Number of training steps We learned the hyperparameters using 200k training steps for RAFT. With the same hyperparameters, running more iterations to train RAFT, such as 800k, results in moderate gains on both Sintel and KITTI.

Target datasets AutoFlow directly optimizes the performance on a target dataset. To test how well AutoFlow generalizes, we learned hyperparameters for Sintel.final and KITTI separately and found that the generalization gap is small. It is likely that the rendering pipeline and the small number of hyperparameters act as a form of regularization, which helps generalization.

Data augmentation RandAugment leads to moderate improvement over applying the same augmentation at every training step, likely because RandAugment increases the diversity of training data. Turning off spatial augmentation results in a moderate drop in accuracy on both KITTI and Sintel. Turning off color augmentation results in severe performance degradation on KITTI, likely because KITTI data includes more lighting changes.

Motion statistics We compared the statistics of motion magnitude for different datasets in Figure 8. The motion statistics of AutoFlow differ from those of Sintel and FlyingChairs. AutoFlow has little small motion, concentrates mainly in the middle-range motion, and does not exhibit an exponential falloff. We further analyzed the augmented data, as it is used to train models. The augmented AutoFlow also has little small motion and concentrates in the middle to high-range motion, probably because tiny motion matters little in the overall learning metric.

Number of pre-training examples With the hyperparameters learned, we can render different numbers of pre-training examples, as shown in Fig. 1. The training of both RAFT and PWC-Net converge using one pre-training example with data augmentation, and more examples lead to better results. Four AutoFlow examples result in lower errors on Sintel.final for RAFT than 22,872 FlyingChairs examples. In this low-data regime, data augmentation plays a key role. Without spatial augmentation, the AEPE by

Experiment		Sintel clean	Sintel final	KITTI
Fog	On	2.08	2.75	4.66
	Off	2.07	3.11	4.92
Motion blur	Box	2.08	2.75	4.66
	Gaussian	2.17	2.75	4.71
	Off	2.10	3.77	5.68
Appearance	Davis	2.08	2.75	4.66
	OpenImages	2.20	2.85	4.83
	Sintel-540p	2.17	2.88	4.75
Resolution	540p	2.06	2.88	4.87
	720p	2.08	2.75	4.66
	1080p	2.33	2.85	5.09
Object mask	Sharp	2.09	2.80	4.94
	Smooth	2.08	2.75	4.66
	Instance	1.99	2.78	4.85
Number of foreground objects	0	5.32	5.74	9.02
	1	2.38	3.07	5.22
	2	2.17	2.93	4.99
	3	2.11	2.76	4.64
	4	2.08	2.75	4.66
	5	2.02	2.87	4.66
	6	2.05	2.84	4.57
Grid warp	On	2.08	2.75	4.66
	Off	2.30	2.92	5.26
Training steps	50k	2.42	3.27	5.81
	200k	2.08	2.75	4.66
	800k	1.95	2.57	4.23
Target dataset	Sintel.final	2.08	2.75	4.66
	KITTI	2.09	2.82	4.33
Augmentation	All	2.22	2.87	4.87
	RandAugment	2.08	2.75	4.66
	No spatial	2.78	3.37	5.22
	No color	2.24	2.92	14.06

Table 5: **Ablation study**. Baseline options are underlined.

RAFT trained on 4 AutoFlow examples drops from 3.57 to 7.66, more severe than the drop from 2.75 to 3.37 in Table 5. Further, as shown in Fig. 9, although the statistics of 4 AutoFlow examples differ significantly from those of the full AutoFlow, they are similar for augmented data.

Discussions While AutoFlow empirically works better than FlyingChairs/FlyingThings3D, we should note that the comparisons are not strictly fair because of differences in implementations and hyperparameters. Although comparing motion statistics reveals some interesting properties, learning hyperparameters for a 3D rendering pipeline in the same setup would help identify key design choices.

6. Conclusions

We have introduced AutoFlow, a simple and effective method to learn pre-training data for optical flow. AutoFlow uses 2D rendering but achieves results comparable to or better than those obtained by FlyingChairs and FlyingThings3D that have been generated using 3D models. In particular, using as few as 4 AutoFlow examples with augmentation results in more accurate results on Sintel.final for RAFT than 22,872 FlyingChairs examples with augmentation. AutoFlow also significantly improves PWC-Net, even on par with RAFT. We hope that our approach will provide another option for pre-training optical flow and enable further progress and innovation in this direction.

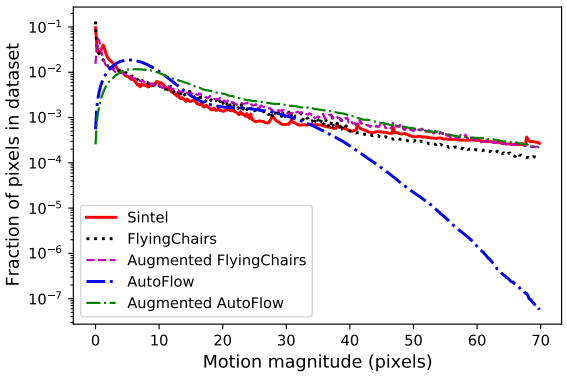

Figure 8: **Histogram of motion magnitude for different datasets**. The augmented AutoFlow concentrates more on middle to large-range motion than Sintel, likely because the small motion contributes little to the overall error.

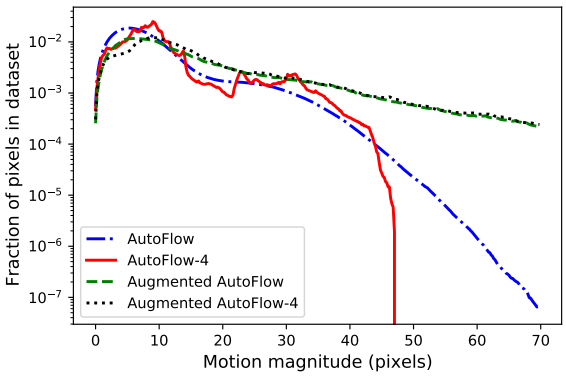

Figure 9: **Histogram of motion magnitude for AutoFlow**. While statistics differ between the 4-example AutoFlow and full AutoFlow, they are close for the augmented data.

Acknowledgements We would like to thank Shuyang Cheng, Ekin Dogus Cubuk, Alex Dosovitskiy, Rico Jonschkowski, David Kao, Ang Li, Aaron Sarna, Austin Stone, and Barret Zoph for helpful discussions and support.

References

[1] S Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 2011. 2

[2] J.L. Barron, D.J. Fleet, and S.S. Beauchemin. Performance of optical flow techniques. *IJCV*, 1994. 2

[3] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *Proc. ECCV*, 2012. 1, 2

[4] Shuyang Cheng, Zhaoqi Leng, Ekin Dogus Cubuk, Barret Zoph, Chunyan Bai, Jiquan Ngiam, Yang Song, Benjamin Caine, Vijay Vasudevan, Congcong Li, et al. Improving 3d object detection through progressive population based augmentation. In *Proc. ECCV*, 2020. 3, 5

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. CVPR*, 2016. 2

[6] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 3

[7] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020. 5

[8] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, Thomas Brox, et al. FlowNet: Learning optical flow with convolutional networks. In *Proc. ICCV*, 2015. 1, 2, 3, 5

[9] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017. 2

[10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010. 2

[11] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proc. CVPR*, pages 4340–4349, 2016. 1, 2

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proc. CVPR*, pages 3354–3361. IEEE, 2012. 1, 2

[13] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016. 5

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 1

[15] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proc. CVPR*, 2017. 1

[16] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proc. CVPR*, 2018. 3

[17] Junhwa Hur and Stefan Roth. Iterative residual refinement for joint optical flow and occlusion estimation. In *Proc. CVPR*, pages 5754–5763, 2019. 3

[18] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *Proc. CVPR*, 2017. 2, 3

[19] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017. 5

[20] Huaizu Jiang, Deqing Sun, Varun Jampani, Zhaoyang Lv, Erik Learned-Miller, and Jan Kautz. Sense: A shared encoder network for scene-flow estimation. In *Proc. ICCV*, 2019. 3

[21] Rico Jonschkowski, Austin Stone, Jonathan T Barron, Ariel Gordon, Kurt Konolige, and Anelia Angelova. What matters in unsupervised optical flow. In *Proc. ECCV*, 2020. 3

[22] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *Proc. ICCV*, pages 4551–4560, 2019. 2

[23] Daniel Kondermann, Rahul Nair, Katrin Honauer, Karsten Krispin, Jonas Andrulis, Alexander Brock, Burkhard Gussefeld, Mohsen Rahimimoghaddam, Sabine Hofmann, Claus Brenner, et al. The hci benchmark suite: Stereo and flow ground truth with uncertainties for urban autonomous driving. In *CVPR Workshops*, pages 19–28, 2016. 2

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Proc. NeurIPS*, 2012. 1

[25] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV*, 2020. 4, 7

[26] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, pages 740–755. Springer, 2014. 2

[28] Pengpeng Liu, Michael Lyu, Irwin King, and Jia Xu. Selflow: Self-supervised learning of optical flow. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 3

[29] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Proc. ECCV*. Springer, 2014. 2

[30] Zhaoyang Lv, Kihwan Kim, Alejandro Troccoli, Deqing Sun, James Rehg, and Jan Kautz. Learning rigidity in dynamic scenes with a moving camera for 3d motion field estimation. In *Proc. ECCV*, 2018. 1, 2

[31] Nikolaus Mayer, Eddy Ilg, Philipp Fischer, Caner Hazirbas, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. What makes good synthetic training data for learning disparity and optical flow estimation? *IJCV*, 126(9):942–960, 2018. 3

[32] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proc. CVPR*, 2016. 1, 2

[33] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3d estimation of vehicles and scene flow. In *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015. 2

[34] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Computer Vision and Pattern Recognition*, 2016. 5

[35] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Proc. CVPR*, 2017. 3

[36] Anurag Ranjan, David T Hoffmann, Dimitrios Tzionas, Siyu Tang, Javier Romero, and Michael J Black. Learning multi-human optical flow. *IJCV*, pages 1–18, 2020. 2

[37] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 2

[38] Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2232–2241, 2017. 1, 2

[39] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proc. CVPR*, pages 3234–3243, 2016. 2

[40] Stefan Roth and Michael J Black. On the spatial statistics of optical flow. *IJCV*, 74(1):33–50, 2007. 2

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 1, 2

[42] Leon Sixt, Benjamin Wild, and Tim Landgraf. Rendergan: Generating realistic labeled data. *Frontiers in Robotics and AI*, 5:66, 2018. 2

[43] Deqing Sun, Charles Herrmann, Varun Jampani, Michael Krainin, Forrester Cole, Austin Stone, Rico Jonschkowski, Ramin Zabih, William T. Freeman, and Ce Liu. TF-RAFT: A tensorflow implementation of raft. In *ECCV Robust Vision Challenge Workshop*, 2020. 5, 6

[44] Deqing Sun, Erik B Sudderth, and Michael J Black. Layered image motion with explicit occlusions, temporal consistency, and depth ordering. In *Proc. NeurIPS*, pages 2226–2234, 2010. 3

[45] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, June 2018. 3, 5, 6

[46] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Models matter, so does training: An empirical study of cnns for optical flow estimation. *IEEE TPAMI*, 2019. 3

[47] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010. 4

[48] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *Proc. ECCV*, 2020. 3, 6

[49] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5):625–638, Sept. 1994. 3

[50] Dawei Yang and Jia Deng. Learning to generate 3d training data through hybrid gradient. In *Proc. CVPR*, 2020. 2

[51] Gengshan Yang and Deva Ramanan. Volumetric correspondence networks for optical flow. In *Advances in neural information processing systems*, pages 794–805, 2019. 3

[52] Zhichao Yin, Trevor Darrell, and Fisher Yu. Hierarchical discrete distribution decomposition for match density estimation. In *Proc. CVPR*, pages 6044–6053, 2019. 3

[53] Feihu Zhang, Xiaojuan Qi, Ruigang Yang, Victor Prisacariu, Benjamin Wah, and Philip Torr. Domain-invariant stereo matching networks. In *Proc. ECCV*, 2020. 5

[54] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 3