DFLOW: LEARNING TO SYNTHESIZE BETTER OPTICAL FLOW DATASETS VIA A DIFFERENTIABLE PIPELINE

Anonymous authors
Paper under double-blind review

ABSTRACT

Comprehensive studies of synthetic optical flow datasets have attempted to reveal what properties lead to accuracy improvement. However, manually identifying and verifying all such necessary properties are intractable mainly due to the requirement of large-scale trial-and-error experiments with iteratively generating whole synthetic datasets. To tackle this challenge, we propose a differentiable optical flow data generation pipeline and a loss function to drive the pipeline, called DFlow. These enable automatic and efficient synthesis of a dataset effective to a target domain, given a snippet of target data. This distinctiveness is achieved by proposing an efficient data comparison method, where we approximately encode reference sets of data into neural networks and compare the proxy networks instead of explicitly comparing datasets in a sample-wise way. Our experiments show the competitive performance of our DFlow against the prior arts in pre-training. Moreover, the RAFT model pre-trained with DFlow achieves state-of-the-art performance on the Sintel public benchmark in fine-tuning.

1 INTRODUCTION

Optical flow is a fundamental computer vision problem to find dense pixel-wise correspondences between two subsequent frames. Optical flow is indeed a key building block in many practical applications, including 3D vision, video understanding, action analysis, video enhancement, editing, *etc*. Recently, optical flow has been significantly advanced by learning-based approaches with deep neural networks (Fischer et al., 2015; Ilg et al., 2017; Ranjan & Black, 2017; Hui et al., 2018; Sun et al., 2018; Teed & Deng, 2020) in terms of accuracy and efficiency.

A driving force of these prior arts is large-scale supervised datasets. However, since it is difficult to collect a reasonable amount of real-world optical flow labels, they have exploited large-scale synthetic datasets, *e.g.*, Fischer et al. (2015); Mayer et al. (2016), which become the standard in optical flow, *e.g.*, training on FlyingChairs (Fischer et al., 2015) followed by FlyingThings3D (Mayer et al., 2016). After the seminal studies, there have been various efforts to build different synthetic datasets (Gaidon et al., 2016; Richter et al., 2017; Lv et al., 2018; Aleotti et al., 2021). Despite the vast efforts of these studies, it remains unclear which factors are important for an effective synthetic dataset construction according to target data.

Instead of manually identifying important design criteria, AutoFlow (Sun et al., 2021) pioneers the first learning-based approach to go beyond being heuristic by posing data generation as a hyperparameter optimization problem to maximize the validation performance on a target dataset. AutoFlow generates data samples by composing simple 2D layers with non-differentiable hyperparameters, which are optimized by sampling-based evolutionary search. The use of evolutionary search requires large resources, which would be critical because it requires re-generating a dataset for each target scenario.

To address the high computational cost, we propose DFlow, which is an efficient synthetic optical flow dataset generation method. We parameterize each data sample with a learnable parameter composed of graphic operations, such as warping layer and real-world effects. This allows us to exploit efficient gradient descent methods to generate each sample, and thereby DFlow is more than an order of magnitude efficient in GPU hours than AutoFlow when constructing a training dataset.

We also introduce a new loss function that learns the data parameters by contrasting a target dataset from a base dataset, *e.g.*, FlyingChairs. Since directly using large datasets in the contrastive learning

	AutoFlow (Sun et al., 2021)	DFlow (Ours)
Learning method	Evolutionary search	Gradient descent
Computational resource	48 P100 GPUs	A single V100 GPU
GPU hours for constructing a dataset	336 days / P100	9.3 days / V100
Data resolution	1280×720	512×384

Table 1: **Comparison of AutoFlow and DFlow (ours).**

process is cumbersome, we approximate the base and target datasets by two networks trained on respective datasets as proxies. This approximation allows an end-to-end differentiable pipeline from the data parameters to the loss function.

Through comprehensive experiments, we show that DFlow is effective in pre-training and fine-tuning. The DFlow data has a size of 512×384, which is the same as FlyingChairs, but the RAFT network (Teed & Deng, 2020) pre-trained on DFlow achieves comparable performance compared to the high-resolution competing datasets (Sun et al., 2021; Mayer et al., 2016). In addition, the RAFT network initially pre-trained on DFlow achieves state-of-the-art performance on the Sintel public benchmark in fine-tuning.

We summarize our contributions as follows:

- A simple and efficient differentiable data generation pipeline for optical flow (refer to Table 1);
- A contrastive-style learning scheme and its loss function by approximating expensive dataset-to-dataset comparison to leverage proxy neural networks (refer to Sec 3);
- State-of-the-art performance on the Sintel public benchmark (refer to Table 5).

2 Related Work

Optical Flow. Dense optical flow estimation is to find pixel-wise correspondences from the brightness patterns of images (Gibson, 1950; Gibson & Carmichael, 1966; Horn & Schunck, 1981). After conventional optimization algorithms (Black & Anandan, 1993; Zach et al., 2007), deep-learning algorithms (Fischer et al., 2015; Ilg et al., 2017) become dominant due to their computational efficiency and reasonable performance. Prior arts (Xu et al., 2017; Bailer et al., 2017; Wulff et al., 2017; Sun et al., 2018) have attempted to implement explicit neural modules that are suitable for optical flow estimation. Recently, RAFT (Teed & Deng, 2020) adopts recurrent architectures and achieves a notable performance improvement, which is represented as state-of-the-art.

Those recent advances in learning-based approaches require large-scale data with ground-truth, but labeling dense optical flow is a highly undetermined task, *i.e.*, challenging (Fischer et al., 2015). The previous real-world datasets have been built under sophisticated labeling conditions, including the special sensor hardware, controlled environment, or limited objects (Scharstein & Szeliski, 2002; Scharstein & Pal, 2007; Geiger et al., 2012; Kondermann et al., 2014). It leads to the limitation of the size of datasets. To relieve this issue, synthetic datasets (Fischer et al., 2015; Mayer et al., 2016) have been proposed and achieved remarkable accuracy despite the gap between real and synthetic datasets. After that, previous arts endeavor to construct more realistic synthetic datasets (Gaidon et al., 2016; Richter et al., 2017; Lv et al., 2018; Aleotti et al., 2021), but it remains unclear what properties lead to accuracy improvement. In this work, we propose a differentiable synthetic data generation pipeline with the target knowledge so that the generated dataset could improve its performance more.

Learning Based Optical Flow Dataset. In computer vision tasks other than optical flow, there are attempts to generate data in learnable ways (Sixt et al., 2018; Yang & Deng, 2020; Kaspar et al., 2019; Zhao et al., 2021); in optical flow, AutoFlow (Sun et al., 2021) is the first learning-based data generation approach. It is our closest related work in the sense that they generate a dataset for the performance improvement on the specific target dataset. Our method is another learning-based approach but the first differentiable one; thus, ours is more efficient than AutoFlow in terms of the computation cost and corresponding GPU hours for data generation. We set the resolution of generated data to be smaller than AutoFlow, which is disadvantageous to ours in terms of the amount of information in a single sample. Despite this, ours achieves comparable performance with AutoFlow while achieving faster training by the small resolution of data. We list other differences in Table 1.

3 DIFFERENTIABLE DATA GENERATION PIPELINE

Our data generation pipeline synthesizes subsequent frames $\mathbf{I}_t$ and $\mathbf{I}_{t+1}$, and a flow (motion) map $\mathbf{F}_{GT}$ between those two frames, given a snippet of target domain data, which ends up constructing a synthetic optical flow dataset. We concisely denote a pair of subsequent frames and a flow map as a data sample. We parameterize each data sample with a learnable parameter $\boldsymbol{\theta}$ that characterizes a composition of elemental graphic operations, such as color perturbation and warping layers. We refer to the composition process as a data generator, which is designed to be differentiable.

With this differentiable data generator part, we specifically seek to synthesize optical flow data improving the accuracy on the target dataset. We define a loss function $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \mathcal{L}_{\text{reg}}$ to effectively update the learnable parameters, which drives the data generation to an improving direction in the target data domain. With this loss, we efficiently learn the data parameters $\{\boldsymbol{\theta}\}$ in a fully-differentiable way.

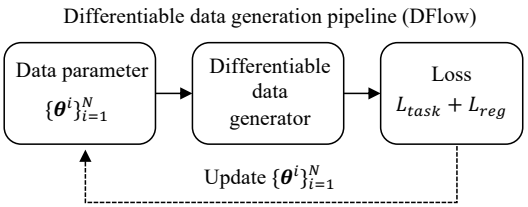

Figure 1: **An overview of DFlow.** We parameterize data samples to be synthesized. The learnable data parameters are updated by our proposed loss functions consisting of the task and regularization losses, $\mathcal{L}_{\text{task}}$ and $\mathcal{L}_{\text{reg}}$.

The overview of our data generation pipeline is illustrated in Fig. 1, called DFlow. In Sec. 3.1, we first describe the design of the loss function. In Secs. 3.2 and 3.3, we describe the data generator method from the parameters $\{\boldsymbol{\theta}\}$ and its regularization methods, respectively. Further detailed descriptions and implementation details can be found in the appendix.

3.1 TASK LOSS FUNCTION

Given our differentiable data generator, a sample parameter $\boldsymbol{\theta}$ is rendered to a data sample, *i.e.*, a pair of subsequent frames $\mathbf{I}_t(\boldsymbol{\theta})$ and $\mathbf{I}_{t+1}(\boldsymbol{\theta})$ and those corresponding flow map $\mathbf{F}_{GT}(\boldsymbol{\theta})$. By rendering a collection of data samples with the data generator and a set of $\{\boldsymbol{\theta}\}$, we can construct a dataset. To find an updated parameter $\boldsymbol{\theta}^*$ that improves the accuracy of the optical flow model, we need an effective criterion to drive the parameter update.

As a tractable criterion to efficiently find such a dataset, we are motivated by learning-to-augment and dataset condensation approaches (Sixt et al., 2018; Kaspar et al., 2019; Zhao et al., 2021). They synthesize data to have characteristics similar to target data under the motivation that learning with two similar datasets is likely to yield accuracy improvement (Ben-David et al., 2010; Kaspar et al., 2019). However, directly comparing combinatorial pairs of elements in the target dataset and synthesized one is computationally cumbersome, and even intractable during iteratively updating a synthesized dataset. To efficiently deal with this issue, we propose to encode the target dataset into a proxy network, called target network.

With an optical flow neural network $f_{\text{target}}(\mathbf{I}_t, \mathbf{I}_{t+1}) \rightarrow \mathbf{F}$, we pre-train the network with the given target optical flow dataset, which yields the trained target network. We deem the target network as a differentiable proxy of the target dataset that encodes the knowledge of target data. Compared to exploiting the target data as a set, leveraging this differentiable proxy would make the whole procedure tractable. With the target network, we define a target loss $\mathcal{L}_{\text{target}}\left(f_{\text{target}}\left(\mathbf{I}_t(\boldsymbol{\theta}), \mathbf{I}_{t+1}(\boldsymbol{\theta})\right), \mathbf{F}_{GT}(\boldsymbol{\theta})\right)$ that measures flow errors for a data sample in the perspective of target domain, where the target loss $\mathcal{L}_{\text{target}}(\cdot)$ is the same loss used for pre-training the target network, *e.g.*, the sequence loss suggested by RAFT (Teed & Deng, 2020).

With the target network and loss, we first attempt to update the parameters $\{\boldsymbol{\theta}\}$ by

$$\{\boldsymbol{\theta}^*\} = \arg\min_{\{\boldsymbol{\theta}\}} \sum_i \mathcal{L}_{\text{target}}\left(f_{\text{target}}\left(\mathbf{I}_t(\boldsymbol{\theta}_i), \mathbf{I}_{t+1}(\boldsymbol{\theta}_i)\right), \mathbf{F}_{GT}(\boldsymbol{\theta}_i)\right). \tag{1}$$

If the target loss is small for a data sample, it means that the target network is familiar with the data sample and the sample is close to one of target data, *i.e.*, similar characteristics to target data. Thereby, we distill the target knowledge into generated data. However, we observe instability and local optima when using the target network solely, where optimizing the loss is stuck at a high

value. We hypothesize that, since a snippet of the target dataset is typically on small-scale in general scenarios,[1] such a weakly trained target network may not produce sufficiently rich training signals.

To complement more training signals, we employ the contrastive-style learning scheme by using another comparator network, called base network, which is trained with a common base dataset. Motivation of this is to more strongly drive the data generation to an improving direction in a target domain by additionally constrasting with a common data domain. This may facilitate the learning parameters to capture distinctive features of the target domain. Similar to the target network, we pre-train the base network but on a large-scale dataset, FlyingChairs (Fischer et al., 2015), as a base dataset, which is randomly generated. The usage is similar to the case of the target network except for maximizing flow errors $\mathcal{L}_{\text{base}}$ to implement a contrasting behavior. To implement the contrasting behavior, we define a wrapping loss function, $\mathcal{L}_{\text{task}}(\mathcal{L}_{\text{target}}, \mathcal{L}_{\text{base}}) \colon \mathbb{R}^2 \rightarrow \mathbb{R}$, called the task loss.

What Function Type is Suitable for $\mathcal{L}_{\text{task}}$. There are many candidates for $\mathcal{L}_{\text{task}}$ satisfying the aforementioned criteria. As a function form, we are first motivated by sample-wise weighting distillation (Zhou et al., 2021), where the loss function is dynamically weighted according to each data sample. The loss function is weighted by the target and base losses in a multiplication form. We additionally examine addition forms to select the best one. We consider tanh, sigmoid, and exponential functions to implement bounded contrasting behaviors.[2] Table 2 shows the accuracy of different forms of loss functions using 1000 data samples generated from the target dataset, *i.e.*, Sintel (Butler et al., 2012). The addition form with the exponential function shows promising results in terms of performance, which corresponds to

Form	$\mathcal{L}_{\text{task}}$	Dataset	
		Sintel clean	Sintel final
Multiplication	Exponential	4.26	4.49
	Sigmoid	4.12	4.35
	Tanh	2.02	3.35
Addition	Exponential	**1.85**	**3.09**
	Sigmoid	2.00	3.13
	Tanh	1.95	3.21

*Bold denotes the best.

Table 2: **AEPE of different forms of $\mathcal{L}_{\text{task}}$.** We train the RAFT network with different loss forms. The addition form with exponential shows promising results. Refer to the appendix for details.

$$\mathcal{L}_{\text{task}}(\mathcal{L}_{\text{target}}, \mathcal{L}_{\text{base}}) = \mathcal{L}_{\text{target}} + \alpha \exp(-\beta \tfrac{\mathcal{L}_{\text{base}}}{\mathcal{L}_{\text{target}}+\epsilon} + \gamma), \tag{2}$$

where we set the balance parameters $\{\beta, \gamma\}$ to $\{1, 0\}$, and especially set α to 20 for the experiments in Table 2. We found that, depending on α, generated data characteristics are distinctive. When generating our final dataset, we first generate subsets of data with different α values, and ensemble the subsets into a single dataset by taking union, denoted as $\{\alpha\}$ *combination*. This is found to be notably effective.

3.2 Data Generator

In this section, we describe the differentiable data generator that synthesizes a data sample from a parameter $\boldsymbol{\theta}$. We start from sampling N pairs of layer images and masks, $\{\mathbf{L}_0^l\}_{l=1}^N$ and $\{\mathbf{M}_0^l\}_{l=1}^N$, which are used as layers for frame composition. Specifically, we randomly sample N images $\{\mathbf{L}_0^l\}_{l=1}^N$ from the public image datasets (Kuznetsova et al., 2020; Perazzi et al., 2016; Mayer et al., 2016; Lin et al., 2014), and N masks $\{\mathbf{M}_0^l\}_{l=1}^N$ from the segmentation dataset (Everingham et al., 2010). With these, we synthesize layers and composite them to generate subsequent frames, $\mathbf{I}_t$ and $\mathbf{I}_{t+1}$, with those corresponding ground-truth optical flow $\mathbf{F}_{GT}$,

$$\{\mathbf{L}_0^l\}_{l=1}^N, \{\mathbf{M}_0^l\}_{l=1}^N \rightarrow \{\mathbf{I}_t, \mathbf{I}_{t+1}, \mathbf{F}_{GT}\}\,. \tag{3}$$

Figure 2 illustrates the data generator consisting of a pipeline of color perturbation, geometric warping, flow field translation, layer composition, and real-world effects in a sequential manner, where we denote all the controllable parameters of them as a learnable parameter $\boldsymbol{\theta}$ for each data sample. In the following paragraphs, we present each part of the data generator in detail.

Geometric Warping and Flow Field Translation. We propose a data generator that has two steps of geometric warping with warping fields $\mathbf{W}_{0\rightarrow t+1}$ and $\mathbf{W}_{t+1\rightarrow t}$ which are computed from

[1] For example, KITTI 2015 consists of just 200 number of training samples.

[2] We observe that constrasting without bounds could lead to instability and divergence. We list the specific equations of the used loss functions in the appendix.

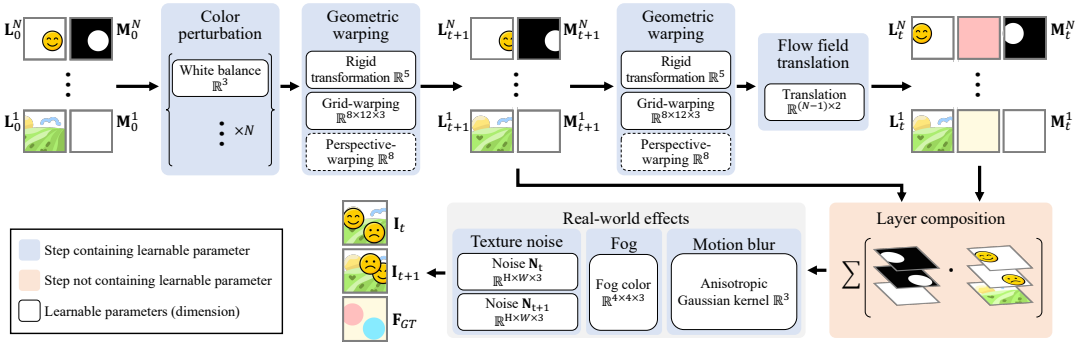

Figure 2: **Schematic of data generator.** The data generator is composed of color perturbation, two steps of geometric warpings, flow field translation, layer composition, and real-world effects. The whole pipeline is differentiably parameterized.

combinations of rigid transformation, perspective warping, and grid warping. First, we warp the layer images and masks, $\{\mathbf{L}_0^l\}_{l=1}^N$ and $\{\mathbf{M}_0^l\}_{l=1}^N$, to be the layer images $\{\mathbf{L}_{t+1}^l\}_{l=1}^N$ and masks $\{\mathbf{M}_{t+1}^l\}_{l=1}^N$ at the next frame $t+1$ by applying the same warping field $\mathbf{W}_{0\to t+1}$ to $\{\mathbf{L}_0^l\}_{l=1}^N$ and $\{\mathbf{M}_0^l\}_{l=1}^N$.

$$\mathbf{L}_{t+1}^l = \mathbf{W}_{0\to t+1} \circ \mathbf{L}_0^l, \quad \mathbf{M}_{t+1}^l = \mathbf{W}_{0\to t+1} \circ \mathbf{M}_0^l, \tag{4}$$

where $\circ$ denotes the geometric warping operation according to a given warping field, and the same warping field $\mathbf{W}_{0\to t+1}$ is shared across all the layers in this step.

In the next step, each layer image $\mathbf{L}_t^l$ and mask $\mathbf{M}_t^l$ at the current frame t is generated from the ones at the next frame, *i.e.*, $\mathbf{L}_{t+1}^l$ and $\mathbf{M}_{t+1}^l$. The previous arts (Fischer et al., 2015; Sun et al., 2021) apply independent warping fields to each layer for constructing diverse multi-layer flow fields, but we observe that introducing too many parameters increases the optimization difficulty in our framework, leading to failure of optimization, *e.g.*, the loss value is not decreased. Thus, we adopt a different strategy to simulate multi-layer flow fields with minimal parameters. We also use a single and shared warping field $\mathbf{W}_{t+1\to t}$ across all the layers, but apply different translation factors to each layer. We introduce the flow field translation $\{\mathbf{D}^l\}_{l=1}^N$ to the geometric warping $\mathbf{W}_{t+1\to t}$ to translate the warping field $\mathbf{W}_{t+1\to t}$, before warping the layer images and masks as:

$$\mathbf{L}_t^l = (\mathbf{D}^l \circ \mathbf{W}_{t+1\to t}) \circ \mathbf{L}_{t+1}^l, \quad \mathbf{M}_t^l = (\mathbf{D}^l \circ \mathbf{W}_{t+1\to t}) \circ \mathbf{M}_{t+1}^l. \tag{5}$$

The above parameterization is efficient for approximating multi-layer geometric warpings because it only introduces 2 additional parameters per layer to represent global translation while sharing a single complex warping field $\mathbf{W}_{t+1\to t}$. For data generation, this geometric parameterization is sufficient to generate diverse motion fields for layers. Through this two steps of warping, we generate the respective layer images for subsequent frames.

As aforementioned, we propose to use the two steps of warping with $\mathbf{W}_{0\to t+1}$ and $\mathbf{W}_{t+1\to t}$. This is distinctive from the existing works, including Fischer et al. (2015); Sun et al. (2021), where they only use a single step of warping from a randomly generated image to a synthetic next frame. This only parameterizes the next frames, not the reference frames randomly generated. This could not update texture and mask shapes of the reference frames, whereas our method parameterizes both frames with two warping fields, $\mathbf{W}_{0\to t+1}$ and $\mathbf{W}_{t+1\to t}$. It allows us to jointly update subsequent frames.

Layer Composition. From the geometric warping, we have N pairs of layer images, masks, and optical flow fields in order of depth. We superimpose each stack of layers to generate subsequent frames and those ground truth flow maps. A naïve strategy for composing them would be the alpha blending. However, we observe that the alpha blending does not provide gradients when there is a fully opaque mask that occludes rear layers. It probably makes optimization inefficient.

To mitigate this inefficiency, we leverage the softmax splatting (Niklaus & Liu, 2020) that makes the layers fully differentiable during composition. The composition follows:

$$\left\{\sum_{l=1}^N \boldsymbol{K}_{t+1}^l \odot \mathbf{L}_{t+1}^l,\ \sum_{l=1}^N \boldsymbol{K}_t^l \odot \mathbf{L}_t^l,\ \sum_{l=1}^N \boldsymbol{K}_t^l \odot \left(\mathbf{D}^l \circ \mathbf{W}_{t+1\to t}\right)\right\} \to \{\mathbf{I}_{t+1}, \mathbf{I}_t, \mathbf{F}_{GT}\}, \tag{6}$$

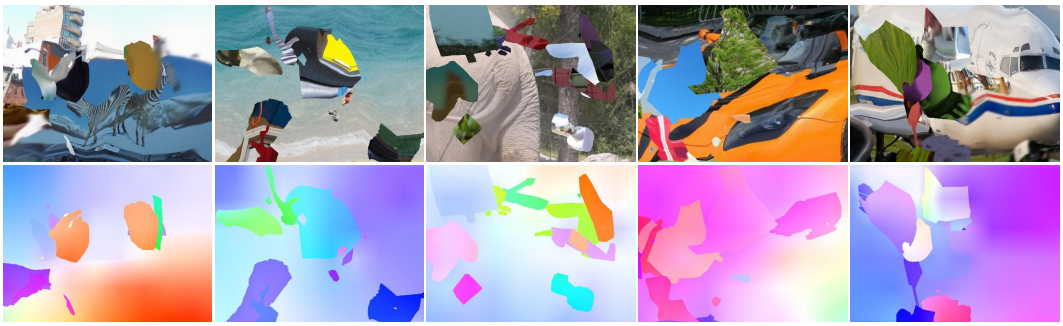

Figure 3: **Generated data samples.** Top: current frames; Bottom: optical flow visualizations.

where $\odot$ is element-wise product, and $\boldsymbol{K}$ denotes weight images for each layer. The weight images $\boldsymbol{K}$ are computed by pixel-wisely applying the softmax function across the stacked mask layers.

Color Perturbation and Real-world Effects. Prior studies (Mayer et al., 2018; Sun et al., 2021) have shown that synthetic data containing real-world effects often brings the performance improvement of optical flow networks in generalization. Inspired by the observations, we introduce the color perturbation and real-world effects into our data generator as well. Color perturbation adjusts the white balance of each data sample and real-world effects consist of texture noises, fog, and motion blur, which are all parameterized to be controlled and updated.

3.3 Regularization

The grid warping and texture noise may take shortcuts only to minimize the task loss. These behaviors are undesirable, and the generated data might not properly train the optical flow network. To handle this potential issue, we propose a regularization term $\mathcal{L}_{\text{reg}}$ consisting of grid and noise regularization losses, *i.e.*, $\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{grid}} + \mathcal{L}_{\text{noise}}$. The grid regularization $\mathcal{L}_{\text{grid}}$ prevents from producing infeasible 2D motion. The noise regularization $\mathcal{L}_{\text{noise}}$ prevents from producing too noisy images. Implementation details can be found in the appendix.

Finally, we obtain our total loss by adding the regularization loss and the task loss as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}}(\mathcal{L}_{\text{target}}, \mathcal{L}_{\text{base}}) + \mathcal{L}_{\text{reg}}. \tag{7}$$

Figure 3 shows samples generated by our proposed differentiable data generation pipeline.

4 Results

In this section, we analyze the effects of our method in pre-training and fine-tuning perspectives with different optical flow models. We report the average end-point error (AEPE) for Sintel (Butler et al., 2012) and AEPE & F1 for KITTI 2015 (Menze & Geiger, 2015). From this section, we refer to our dataset generated by targeting Sintel with RAFT proxy models as DFlow, unless separately mentioned. Other details of experiment setups and implementation can be found in the appendix.

Pre-training RAFT Results. Pre-training performance is a key factor in evaluating the applicability of an optical flow dataset (Fischer et al., 2015; Mayer et al., 2016; 2018; Aleotti et al., 2021; Sun et al., 2021). We train the state-of-the-art RAFT networks (Teed & Deng, 2020) from scratch on respective competing datasets, including our DFlow. Table 3-(a) shows the pre-training results of each dataset. Compared to FlyingChairs that has the same data resolution with DFlow, the model trained on DFlow outperforms the one trained on FlyingChairs in both Sintel and KITTI 2015 datasets. DFlow also shows favorable AEPEs against AutoFlow, despite the fact that DFlow uses around $\frac{1}{4}$ resolution of AutoFlow. Figure 4 shows the qualitative results obtained from FlyingChairs and DFlow. These results show that DFlow is effective for learning an accurate model in challenging scenes, such as shaded, foggy, and motion-blurred scenes.

Pre-training FlowNet Results. Since DFlow is generated with RAFT proxy networks, a question naturally arises about the effectiveness of DFlow in training other model architectures. We further investigate the effectiveness of training another architecture, FlowNet (Fischer et al., 2015), in

Table 3: **Pre-training results.** We train the (a) RAFT (Teed & Deng, 2020) and (b) FlowNet (Fischer et al., 2015) networks on respective pre-training datasets, and evaluate on the evaluation datasets, *i.e.*, Sintel and KITTI 2015. Chairs→Things denotes the heterogeneous dataset experiment pre-training on FlyingChairs followed by FlyingThings3D.

Model	Pre-training dataset	Evaluation dataset		
		Sintel clean	Sintel fianl	KITTI 2015
		AEPE	AEPE	AEPE / F1
(a) RAFT	FlyingChairs	2.28	4.51	9.85 / 37.56
	AutoFlow	2.08	**2.75**	4.66 / -
	DFlow (Ours)	**1.81**	2.93	**4.59 / 15.03**
	Chairs→Things	1.43	2.71	5.04 / 17.40
(b) FlowNet	FlyingChairs	4.71	6.22	20.36 / 62.20
	AutoFlow	5.43	6.03	19.64 / 43.95
	DFlow (Ours)	**3.45**	**4.73**	**12.94 / 38.95**

*Bold denotes the best.

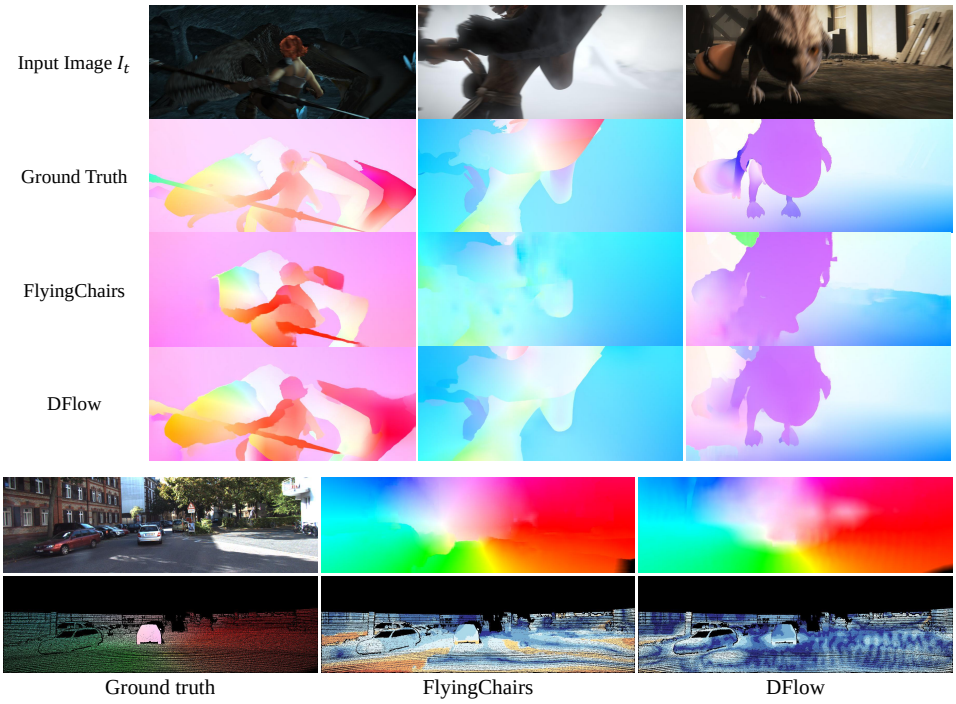

Figure 4: **Qualitative results.** Top: pre-training results on the Sintel final pass. Bottom: pre-training results on KITTI 2015. The RAFT network pre-trained on DFlow shows robust results in challenging shaded, foggy, and motion-blurred scenes.

Table 3-(b). FlowNet pre-trained on DFlow shows noticeable improvement over the ones pre-trained on the competing datasets, which implies that our DFlow would generalize well to other architectures despite using RAFT proxy networks.

Pre-training Performance according to Motion Magnitudes. Table 4 summarizes AEPEs of RAFT pre-trained on respective FlyingChairs (Fischer et al., 2015) and our DFlow according to different motion ranges. The model trained on DFlow shows higher accuracy than that of FlyingChairs except for the minimal motion range. Performance gaps between the models tend to be enlarged in larger motion. These results may be explained by human intuition that the small motion hardly contributes to the error of optical flow networks. For the same reason, most motion magnitudes of DFlow are in the middle and high motion ranges, as shown in Fig. 5.

Table 4: **Motion accuracy of the RAFT networks in different magnitude ranges.** We evaluate the AEPE score of models pre-trained on FlyingChairs and DFlow in different motion ranges. As the motion magnitude increases, the model trained on DFlow achieves more considerable performance improvements than that trained on FlyingChairs.

Pre-training dataset	Evalutaion dataset	Motion magnitude ranges				
		< 1	$[1, 10]$	$(10, 20]$	$(20, 30]$	> 30
FlyingChairs	Sintel Final	**0.57**	1.15	3.31	6.47	24.08
DFlow (Ours)		0.69	**0.80**	**2.04**	**3.59**	**15.18**
FlyingChairs	KITTI 2015	**0.74**	1.67	2.65	3.86	21.32
DFlow (Ours)		1.16	**0.81**	**1.31**	**2.10**	**9.72**

*Bold denotes the best.

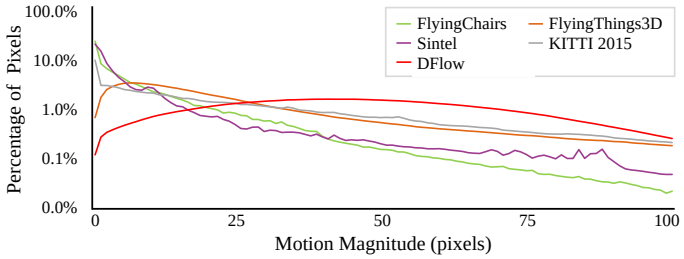

Figure 5: **Motion magnitude histograms.** DFlow focuses more on the mid-high ranges of motion than the other datasets, because small motion hardly contributes to errors of networks.

Fine-tuning Results on Public Benchmarks. Table 5 shows the fine-tuning results on the public benchmark test sets with corresponding dataset schedules suggested by Teed & Deng (2020). DFlow improves the original RAFT recipe (C→T→TSKH/K) on both benchmarks by replacing the conventional initial dataset schedule, *i.e.*, FlyingChairs followed by FlyingThings3D (C→T), with our DFlow. In particular, DFlow achieves state-of-the-art performance on the Sintel public benchmark, which validates that DFlow as a pre-training dataset effectively affects fine-tuning results.

Analysis of Components. We analyze the effects of each generation pipeline component: $\{\alpha\}$ combination, the number of foregrounds, color perturbation, real-world effects, regularizations, and target dataset. For the fair and quick experiments, we generate 2000 training data and train RAFT networks with the same training details. As shown in Table 6, we add or remove each of the components from a default setting to measure their effects with the pre-training experiment. The default setting indicates the dataset with all components and the number of foregrounds parameter in the range of 8 to 12 except the $\{\alpha\}$ combination. We observe that applying each component brings consistent performance improvement on the target dataset, *i.e.*, Sintel, which is used for generating DFlow; whereas the results on KITTI 2015, which is not used for generating DFlow, are not consistent with the results of Sintel.

- *Data update.* This analysis compares DFlow with randomly initialized data without any optimization, *i.e.*, randomly generated by our pipeline. We train and compare the RAFT networks with those datasets, which show the effectiveness gap between the two datasets.
- $\{\alpha\}$ *Combination.* We collect the same amount of data by combining subsets of data, which are obtained from diverse α values. This balances the performance between evaluation datasets because of the regularization effect.
- *Number of Foregrounds.* Without any foreground, we observe the significant performance drop on Sintel and KITTI 2015. Adding only 2 foregrounds brings notable performance improvement.
- *Color Perturbation and Real-world Effects.* Without the color perturbation, the performance on Sintel is dropped moderately. Removing the motion blur significantly affects the performance on Sintel, especially in the Sintel final pass. Without fog and texture noise effects, the performance is dropped moderately.
- *Regularizations.* Removing the regularizations shows a slight performance drop in Sintel.

Table 5: **Fine-tuning results on public benchmarks.** We report F1 scores for KITTI 2015 and AEPE for Sintel public benchmarks. C, K, S, T, and H denote FlyingChairs, KITTI 2015, Sintel, FlyingThings3D, and HD1K (Kondermann et al., 2014).

Dataset schedule	Sintel clean	Sintel final	KITTI 2015
C → T → TSKH/K (Teed & Deng, 2020)	1.94	3.18	5.10
AutoFlow → TSKHV (Sun et al., 2021)	2.01	3.14	**4.78**
DFlow (Ours) → TSKH/K	**1.62**	**3.07**	5.03

*Bold denotes the best.

Table 6: **Analysis of data generation components.** We analyze the effects of each generation component. The experiments are conducted by adding or removing one of the generation components. The default setting is with all components and the number of foregrounds parameter in the range of 8 to 12 except the $\{\alpha\}$ combination, which are annotated with underlines.

		Evaluation dataset		
		Sintel clean	Sintel final	KITTI 2015
Experiments	State	AEPE	AEPE	AEPE / F1
Data update	On	1.86	3.04	5.21 / 16.27
	Off	9.70	9.67	17.31 / 46.84
$\{\alpha\}$ combination	Off	1.86	3.04	5.21 / 16.27
	On	1.90	3.02	4.90 / 15.78
Number of foregrounds	0	4.10	4.89	7.56 / 23.76
	2	2.15	3.32	5.73 / 17.60
	4	2.00	3.15	5.41 / 16.33
	8-12	1.86	3.04	5.21 / 16.27
Color perturbation	On	1.86	3.04	5.21 / 16.27
	Off	2.07	3.20	4.87 / 16.09
Motion blur	On	1.86	3.04	5.21 / 16.27
	Off	1.95	3.57	5.06 / 16.24
Fog	On	1.86	3.04	5.21 / 16.27
	Off	1.98	3.20	5.04 / 15.68
Texture noise	On	1.86	3.04	5.21 / 16.27
	Off	2.02	3.15	5.47 / 16.56
Noise regularization	On	1.86	3.04	5.21 / 16.27
	Off	1.91	3.09	4.97 / 15.87
Grid regularization	On	1.86	3.04	5.21 / 16.27
	Off	1.88	3.10	5.09 / 16.35
Target dataset	Sintel	1.86	3.04	5.21 / 16.27
	KITTI 2015	2.14	3.79	4.31 / 14.29

- *Target Dataset.* The generated dataset optimized to KITTI 2015 shows a notable performance gain on KITTI 2015, while the performance on Sintel significantly drops. This may be caused by the distribution gap between the two datasets.

5 Conclusion

We propose a new data generation pipeline for training optical flow networks. Our pipeline consists of geometric warping, flow field translation, color perturbation, and real-world effects, which are all parameterized differentiably. We propose a new objective function that drives our data optimization by leveraging the compressed knowledge of the proxy networks pre-trained on target and base datasets, respectively. Optical flow models trained on our datasets achieve favorable or superior performance against the competing dataset on pre-training and fine-tuning experiments. Our data generation pipeline including the loss function appears to be general and may be extended to other tasks, which would be an interesting future direction.

ETHICS STATEMENT

This paper develops an optical flow data generation method and uses the public datasets (Kuznetsova et al., 2020; Perazzi et al., 2016; Mayer et al., 2016; Lin et al., 2014; Everingham et al., 2010) for data generation. Therefore, the author does not expect any potential ethical issues related to sensitive information of the dataset.

REPRODUCIBILITY STATEMENT

We introduce the whole data generation pipeline in Sec. 3. In Sec. 3.1, we present the loss function for data optimization. In Secs. 3.2 and 3.3, we describe the data generator method from the parameters and its regularization methods, respectively. We list more details of the data generation pipeline in the appendix.

REFERENCES

Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. Learning optical flow from still images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Christian Bailer, Kiran Varanasi, and Didier Stricker. Cnn-based patch matching for optical flow with thresholded hinge embedding loss. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

Michael J Black and Padmanabhan Anandan. A framework for the robust estimation of optical flow. In *IEEE International Conference on Computer Vision (ICCV)*, 1993.

D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision (ECCV)*, pp. 611–625, 2012.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2): 303–338, 2010.

Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual worlds as proxy for multi-object tracking analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

James J Gibson. The perception of the visual world. 1950.

James Jerome Gibson and Leonard Carmichael. *The senses considered as perceptual systems*. Houghton Mifflin Boston, 1966.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. volume 13, pp. 723–773, 2012.

Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3): 185–203, 1981.

Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Alexandre Kaspar, Tae-Hyun Oh, Liane Makatura, Petr Kellnhofer, and Wojciech Matusik. Neural inverse knitting: From images to manufacturing instructions. In *International Conference on Machine Learning (ICML)*, 2019.

Daniel Kondermann, Rahul Nair, Stephan Meister, Wolfgang Mischler, Burkhard Güssefeld, Sabine Hofmann, Claus Brenner, and Bernd Jähne. Stereo ground truth with error bars. In *Asia Conference on Computer Vision (ACCV)*, 2014.

Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. In *International Journal of Computer Vision (IJCV)*, 2020.

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 *European Conference on Computer Vision (ECCV)*, 2014.

Zhaoyang Lv, Kihwan Kim, Alejandro Troccoli, Deqing Sun, James M Rehg, and Jan Kautz. Learning rigidity in dynamic scenes with a moving camera for 3d motion field estimation. In *European Conference on Computer Vision (ECCV)*, pp. 468–484, 2018.

Nikolaus Mayer, Eddy Ilg, Philip Hausser, 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

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? *International Journal of Computer Vision*, 126(9):942–960, 2018.

Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3061–3070, 2015.

Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5437–5446, 2020.

Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Stephan R Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 2213–2222, 2017.

Daniel Scharstein and Chris Pal. Learning conditional random fields for stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 2002.

Leon Sixt, Benjamin Wild, and Tim Landgraf. RenderGAN: Generating realistic labeled data. *Frontiers in Robotics and AI*, 5:66, 2018.

Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Deqing Sun, Daniel Vlasic, Charles Herrmann, Varun Jampani, Michael Krainin, Huiwen Chang, Ramin Zabih, William T Freeman, and Ce Liu. Autoflow: Learning a better training set for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision (ECCV)*, 2020.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Jonas Wulff, Laura Sevilla-Lara, and Michael J Black. Optical flow in mostly rigid scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Jia Xu, René Ranftl, and Vladlen Koltun. Accurate optical flow via direct cost volume processing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5807–5815, 2017.

Dawei Yang and Jia Deng. Learning to generate 3d training data through hybrid gradient. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–789, 2020.

Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint pattern recognition symposium*, 2007.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with distribution matching. *International Conference on Learning Representations (ICLR)*, 2021.

Helong Zhou, Liangchen Song, Jiajie Chen, Ye Zhou, Guoli Wang, Junsong Yuan, and Qian Zhang. Rethinking soft labels for knowledge distillation: A bias-variance tradeoff perspective. In *International Conference on Learning Representations (ICLR)*, 2021.

APPENDIX

In this appendix, we present implementation details and additional experimental results. The contents are listed as follows:

CONTENTS

A Details of Differentiable Data Generation Pipeline

A.1 Base and Target Networks

A.2 The Form of Task Loss Functions

A.3 Layer Composition

A.4 Color Perturbation and Real-world Effects

A.5 Regularization Loss Function

B Details of Experiment Setup

B.1 Generation Details of the DFlow Dataset

B.2 Details of Pre-training Results

B.3 Details of Fine-tuning Results

C Additional Experiments

C.1 Contrasting Effect of Base and Target Networks

C.2 Validation Results

C.3 Additional Qualitative Results

D Additional Generated Data Samples

A DETAILS OF DIFFERENTIABLE DATA GENERATION PIPELINE

In this section, we describe the implementation details of differentiable data generation pipeline.

A.1 BASE AND TARGET NETWORKS

Base Network. For the architecture, we use the same architecture for both base and target networks. We use the published weight of RAFT (Teed & Deng, 2020) as base network[3], which is pre-trained on FlyingChairs (Fischer et al., 2015) for 100k iterations with a batch size of 12, 496×368 image size, learning rate 4×10^{-4}.

[3]https://github.com/princeton-vl/RAFT

Table 7: **The forms of task loss function.** We list the used forms of task loss function in Sec. 3.1. The form is the combination of {Multiplication, Addition} and {Exponential, Sigmoid, Tanh}. The task loss functions have the hyperparameters, α, β, γ.

Task loss function $\mathcal{L}_{\text{task}}(\mathcal{L}_{\text{target}}, \mathcal{L}_{\text{base}})$	
Multiplication & Exponential	$(1 - \alpha \exp(\beta \mathcal{L}_{\text{base}}/(\mathcal{L}_{\text{target}} + \epsilon) + \gamma))\mathcal{L}_{\text{target}}$
Multiplication & Sigmoid	$(1 + \alpha \texttt{sigmoid}(\beta \mathcal{L}_{\text{target}}/(\mathcal{L}_{\text{base}} + \epsilon) + \gamma))\mathcal{L}_{\text{target}}$
Multiplication & Tanh	$(1 + \alpha \texttt{tanh}(\beta \mathcal{L}_{\text{target}}/(\mathcal{L}_{\text{base}} + \epsilon) + \gamma))\mathcal{L}_{\text{target}}$
Addition & Exponential	$\mathcal{L}_{\text{target}} + \alpha \exp(-\beta \mathcal{L}_{\text{base}}/(\mathcal{L}_{\text{target}} + \epsilon) + \gamma)$
Addition & Sigmoid	$\mathcal{L}_{\text{target}} + \alpha \texttt{sigmoid}(\beta \mathcal{L}_{\text{target}}/(\mathcal{L}_{\text{base}} + \epsilon) + \gamma)$
Addition & Tanh	$\mathcal{L}_{\text{target}} + \alpha \texttt{tanh}(\beta \mathcal{L}_{\text{target}}/(\mathcal{L}_{\text{base}} + \epsilon) + \gamma)$

Target Network. We fine-tune the base network on the Sintel datasets (Butler et al., 2012) for the target network. During fine-tuning, we only use the image crop as data augmentation and fine-tune the base network for 20k iterations with a batch size of 6, 768×368 image size, learning rate 1.25×10^{-4}.

A.2 The Form of Task Loss Functions

We introduce our task loss function $\mathcal{L}_{\text{task}}(\mathcal{L}_{\text{target}}, \mathcal{L}_{\text{base}})$ for contrastive-style learning and evaluate the candidate of the loss function in Sec. 3.1. Table 7 shows the used task loss functions in experiments of Table 2. We use the sequence loss of flow estimates of RAFT (Teed & Deng, 2020) for $\mathcal{L}_{\{target,base\}}$.

A.3 Layer Composition

We adopt the softmax splatting (Niklaus & Liu, 2020) to compose the layer images at subsequent frames and translated warping field, *i.e.*, $\mathbf{L}_t^l$, $\mathbf{L}_{t+1}^l$, and $\mathbf{D}^l \circ \mathbf{W}_{t+1\to t}$. We get the importance image $\boldsymbol{Z}$ and weight image $\boldsymbol{K}$ of each layer and frame,

$$\boldsymbol{Z}_t^l = \mathbf{M}_t^l a^l - \max_l(\mathbf{M}_t^l a^l), \tag{8}$$

$$\boldsymbol{K}_t^l = \frac{\exp(\boldsymbol{Z}_t^l)}{\sum_{l=1}^N \exp(\boldsymbol{Z}_t^l)}. \tag{9}$$

where $a^l = cl$ and c is a constant value which we set 6 for layer composition. As the value of c increases, sharper boundaries can be obtained. The subsequent frames and ground-truth optical flow are as follows:

$$\left\{ \sum_{l=1}^N \boldsymbol{K}_{t+1}^l \odot \mathbf{L}_{t+1}^l,\ \sum_{l=1}^N \boldsymbol{K}_t^l \odot \mathbf{L}_t^l,\ \sum_{l=1}^N \boldsymbol{K}_t^l \odot \left(\mathbf{D}^l \circ \mathbf{W}_{t+1\to t}\right) \right\} \quad \to \{\mathbf{I}_{t+1}, \mathbf{I}_t, \mathbf{F}_{GT}\}, \tag{10}$$

A.4 Color perturbation and Real-world effects

Color Perturbation. Color perturbation consists of N white balances to adjusts the color intensity of each layer. Each white balance has three-channel values from zero to one and has half of the chance to be applied. Color perturbation is the same for subsequent frames and optimized.

Texture Noises. We approximate the shot noise to mimic the real-world photos taken at different times. We apply three-channel texture noises to all of the pixels while masking the area where the noises are actually applied. Two different texture noises, $\mathbf{N}_t$ and $\mathbf{N}_{t+1}$, are applied to subsequent frames and optimized.

Fog and Motion Blur. Inspired by AutoFlow (Sun et al., 2021), we introduce two real-world effects; fog and motion blur. To generate a random fog, we follow the implementation of AutoFlow and superimpose the fog on $\mathbf{I}_t$ and $\mathbf{I}_{t+1}$. However, one difference is that we adjust the three-channel values of fog, which yields the colored fog. The fog is the same for the subsequent frames and is optimized as well. For the motion blur, we randomly sample object masks from PASCAL VOC (Everingham et al., 2010) and combine them to generate a motion blur mask. We use the 2D gaussian blur to

(a) (b) (c) (d)

Figure 6: **Color perturbation and real-world effects.** Real-world effects are composed of texture noises, fog, and motion blur. From left to right: (a) color perturbation, (b) texture noises, (c) fog, (d) motion blur. (a) is an overlaid image before (upper right) and after color perturbation (lower left).

Table 8: **Generation details of DFlow.** Data augmentation applied when generating DFlow.

Data augmentation	Color jitter				Random resize and crop		
	brightness	contrast	saturation	hue	min scale	max scale	image crop
	0.1	0.1	0.1	0.04	0.93	2.30	496×368

approximate the motion blur kernel, and the motion blurred frames are alpha-blended to $\mathbf{I}_t$ and $\mathbf{I}_{t+1}$. The standard deviation of each axis and angle of rotation are parameters to be updated.

Fig. 6 shows the color perturbation and real-world effects including texture noise, fog, and motion blur.

A.5 Regularization Loss Function

To prevent from generating infeasible 2D motion and too noisy images, we apply grid and noise regularization losses to grid warpings and texture noises, respectively. We define the grid regularization loss $\mathcal{L}_{\text{grid}}$ as follows:

$$\mathcal{L}_{\text{grid}} = \max(0, \sum_{k=1}^{w}\sum_{j=1}^{h-1}[\mathbf{C}_t(k,j) - \mathbf{C}_t(k,j+1)] + \sum_{k=1}^{w-1}\sum_{j=1}^{h}[\mathbf{C}_t(k,j) - \mathbf{C}_t(k+1,j)]), \quad (11)$$

where $\mathbf{C}_t$ is the warped coordinate by the geometric warping of next step, *i.e.*, $\mathbf{W}_{t+1\to t}$. $\mathcal{L}_{\text{grid}}$ gives a penalty when the warped coordinates of the previous grid exceed those of the next grid. We also define the noise regularization loss $\mathcal{L}_{\text{noise}}$ as follows:

$$\mathcal{L}_{\text{noise}} = (\|\mathbf{N}_t\|_1 + \|\mathbf{N}_{t+1}\|_1), \quad (12)$$

where $\mathbf{N}_t$ and $\mathbf{N}_{t+1}$ are texture noises of current and next frames, respectively.

B Details of Experiment Setup

In this section, we present the generation details of the DFlow dataset. Then, we present the details of pre-training and fine-tuning results in Sec. 4 of the main paper.

B.1 Generation details of the DFlow Dataset

To generate the DFlow dataset, the data augmentation is included in the proposed differentiable data generation pipeline. We randomly augment each data sample before inputting them to the base and target networks. For stability of data generation, we stack 6 randomly augmented data samples and input them into networks. Each data sample of DFlow is updated up to 80 times and saved depending on the threshold 25 of the target network loss. The DFlow data has a size of 512×384, which is the same as FlyingChairs. Table 8 summarizes the data augmentation applied to each data sample.

B.2 Details of Pre-training Results

Pre-training RAFT. Except for the data augmentation, we set the training parameter settings by following the training details of RAFT (Teed & Deng, 2020) on FlyingChairs (Fischer et al.,

Table 9: **Contrasting analysis in pre-training.** The dataset generated with the contrasting effect achieves comparable results at the target dataset, KITTI 2015. The contrasting effect leads to a notable performance improvement on Sintel, which is not used for data generation.

Contrasting effect	Evaluation dataset		
	Sintel clean	Sintel final	KITTI 2015
	AEPE	AEPE	AEPE / F1
✗	4.38	5.03	4.61 / **13.86**
✓	**2.14**	**3.79**	**4.31** / 14.29

*Bold denotes the best.

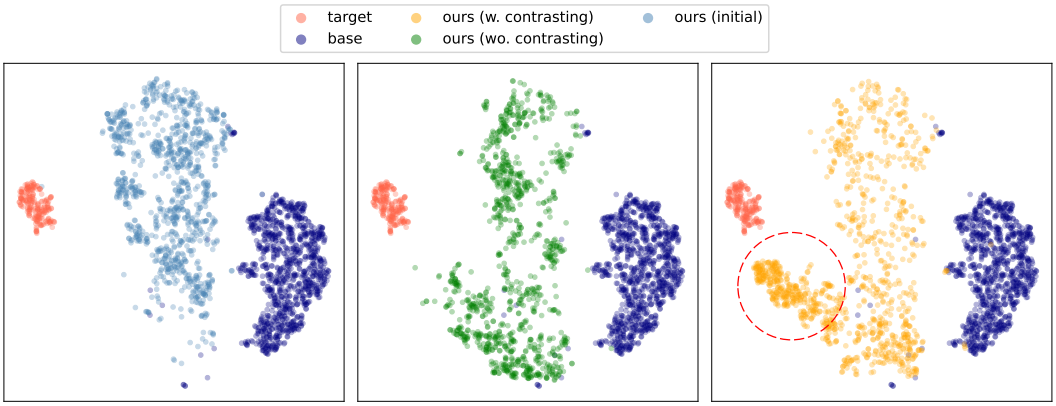

Figure 7: **t-SNE plot (van der Maaten & Hinton, 2008) of different datasets.** We use the context encoder of the base network trained on FlyingChairs as a feature extractor and extract the features of several datasets. (Red): target datasets, *e.g.*, KITTI 2015, (Blue): base dataset, *e.g.*, FlyingChairs, (Sky): our initial dataset before optimization, (green): updated our dataset without the contrasting effect, (Yellow): updated our dataset with the contrasting effect. The features of our dataset with the contrasting effect are getting closer to the features of the target dataset through the optimization process than the features of other datasets.

2015), and pre-train the RAFT network on DFlow with a data size of 15k. We use the same data augmentation applied for generating the DFlow dataset. We early stop the training at 95k iteration.

Pre-training FlowNet. We use the Pytorch implementation of FlowNet[4]. We pre-train the FlowNet model for 2700 epochs with batch size 8. Following the Pytorch Implementation of FlowNet, we use the proposed data augmentation including translation, rotation, image crop, vertical flip, and horizontal flip. For the AutoFlow result, we use the published AutoFlow dataset (Sun et al., 2021).

B.3 Details of Fine-tuning Results

To achieve fine-tuning results on the public benchmarks, we fine-tune the RAFT model pre-trained on DFlow. Except for the initial data schdule, *i.e.*, FlyingChairs followed by FlyingThings3D, we follow the same dataset schedule and training details of the original implementation (Teed & Deng, 2020).

C Additional Experiments

We analyze the contrasting effect of base and target networks in Section C.1, and evaluate the validation performance with DFlow in Section C.2.

[4]https://github.com/ClementPinard/FlowNetPytorch

Table 11: **Comparison of the validation performance of networks fine-tuned on different dataset combinations.** We fine-tune the base network on the combination of KITTI-50 and another dataset. $\text{Ours}_{\text{KITTI-50}}$ notably improves the validation accuracy on the unseen KITTI dataset, KITTI-150.

Fine-tune dataset	Data size	Validation dataset
		KITTI-150
		AEPE / F1
KITTI-50	50	3.41 / 11.47
KITTI-50+Sintel	50+2082	3.39 / 11.21
KITTI-50+$\text{Ours}_{\text{KITTI-50}}$	50+2082	<u>3.07</u> / <u>10.99</u>
KITTI-50+$\text{Ours}_{\text{KITTI-50}}$	50+12003	**2.98 / 10.70**

*__Bold__ denotes the best, and <u>underline</u> denotes the second best.

C.1 Contrasting Effect of Base and Target Networks

To evaluate the contrasting effect of base and target networks, we measure the pre-training performance depending on the contrasting effect. The details will be presented in the following paragraphs.

Contrasting Analysis in Pre-training. For the contrasting analysis, we generate two types of dataset with a sample size of 2000. For the first dataset, we use KITTI 2015 (Menze & Geiger, 2015) as the target dataset, *i.e.*, the target network is obtained by fine-tuning the base network on the KITTI 2015 dataset. To synthesize the first dataset with contrasting effect, we minimize and maximizes the loss of the target and base networks, respectively. For the other dataset, we generate a new target network that is pre-trained on the KITTI 2015 dataset. With the new target network, we generate a new pre-training dataset that is synthesized to minimize the loss of the new target data network only, *i.e.*, the contrasting effect is not applied.

Table 9 shows the pre-training results depending on the contrasting effect. With the contrasting effect, the RAFT network achieves comparable results on the target dataset, *i.e.*, KITTI 2015, compared to the network trained on the dataset generated with the new target network only. However, the contrasting effect leads to a notable performance improvement on Sintel (Butler et al., 2012), which is not used for data generation.

We also visualize data features to identify the effect of the contrasting effect. We embed the features of several datasets, such as FlyingChairs, KITTI 2015, our datasets with and without the contrasting effect, and our dataset before optimization using the context encoder of the base network as a feature extractor. As shown in Fig. 7, ours with the contrasting effect is closer to the target than one without the contrasting effect. We additionally compute distances of feature mean between ours and KITTI 2015. As shown in Table 10, the maximum mean discrepancy (MMD) (Gretton et al., 2012) of ours with the contrasting effect is lower than ours without the contrasting effect. These results show that the contrasting effect drives the data generation toward an improving direction in a target domain.

Table 10: **Distance of feature mean between ours and KITTI 2015.**

Datasets	MMD
Ours wo. contrasting	6.6892
Ours w. contrasting	6.1454

C.2 Validation Results

The advantage of the proposed pipeline is that the generated data can be used to improve the validation performance on the unseen target dataset. We split the KITTI 2015 dataset into 50 training samples and 150 validation samples; KITTI-50 and KITTI-150. Using the target network fine-tuned on KITTI-50, we synthesize a dataset; $\text{Ours}_{\text{KITTI-50}}$. Therefore, any data or information of KITTI-150 is not included in the data generation process. We fine-tune the base network on several combinations of datasets with 20k iterations and evaluate the validation performance on KITTI-150. As shown in Table 11, fine-tuning on the combination of Sintel and KITTI-50 shows comparable performance with the one of fine-tuning on KITTI-50 alone. In contrast, we observe notable improvement in validation performance when adding the same number of $\text{Ours}_{\text{KITTI-50}}$ to the training dataset.

As our data generation pipeline is not limited to the number of samples that can generate, we analyze the effect of our data size and observe the improvement of validation accuracy.

C.3 Additional Qualitative Results

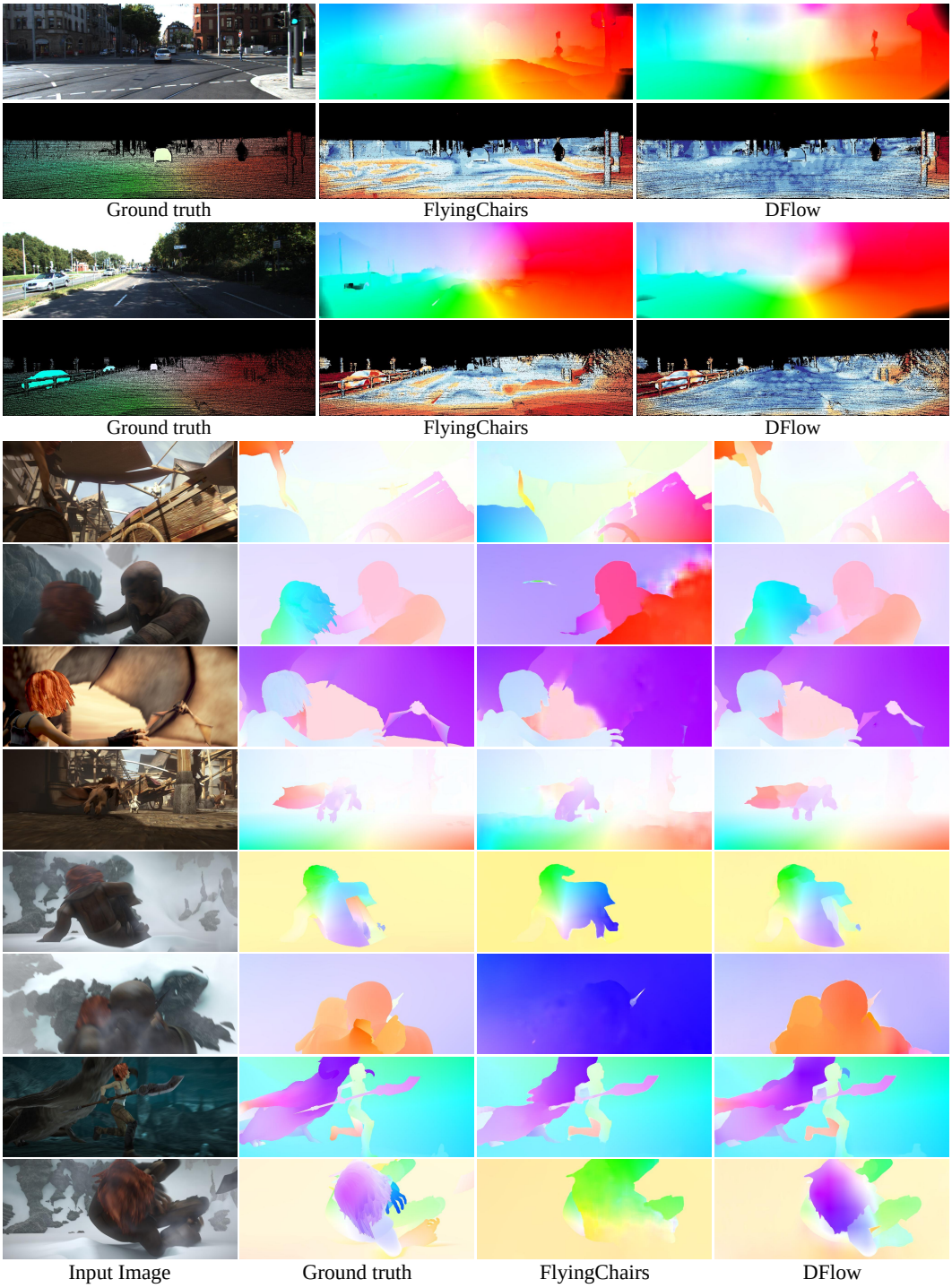

Figure 8: **Additional qualitative results.** Top: pre-training results on the KITTI 2015 dataset. Bottom: pre-training results on Sintel Final pass.

D ADDITIONAL GENERATED DATA SAMPLES

Figure 9 shows additional data samples. Color perturbation, texture noise, fog, and motion blur can be found in the data samples.

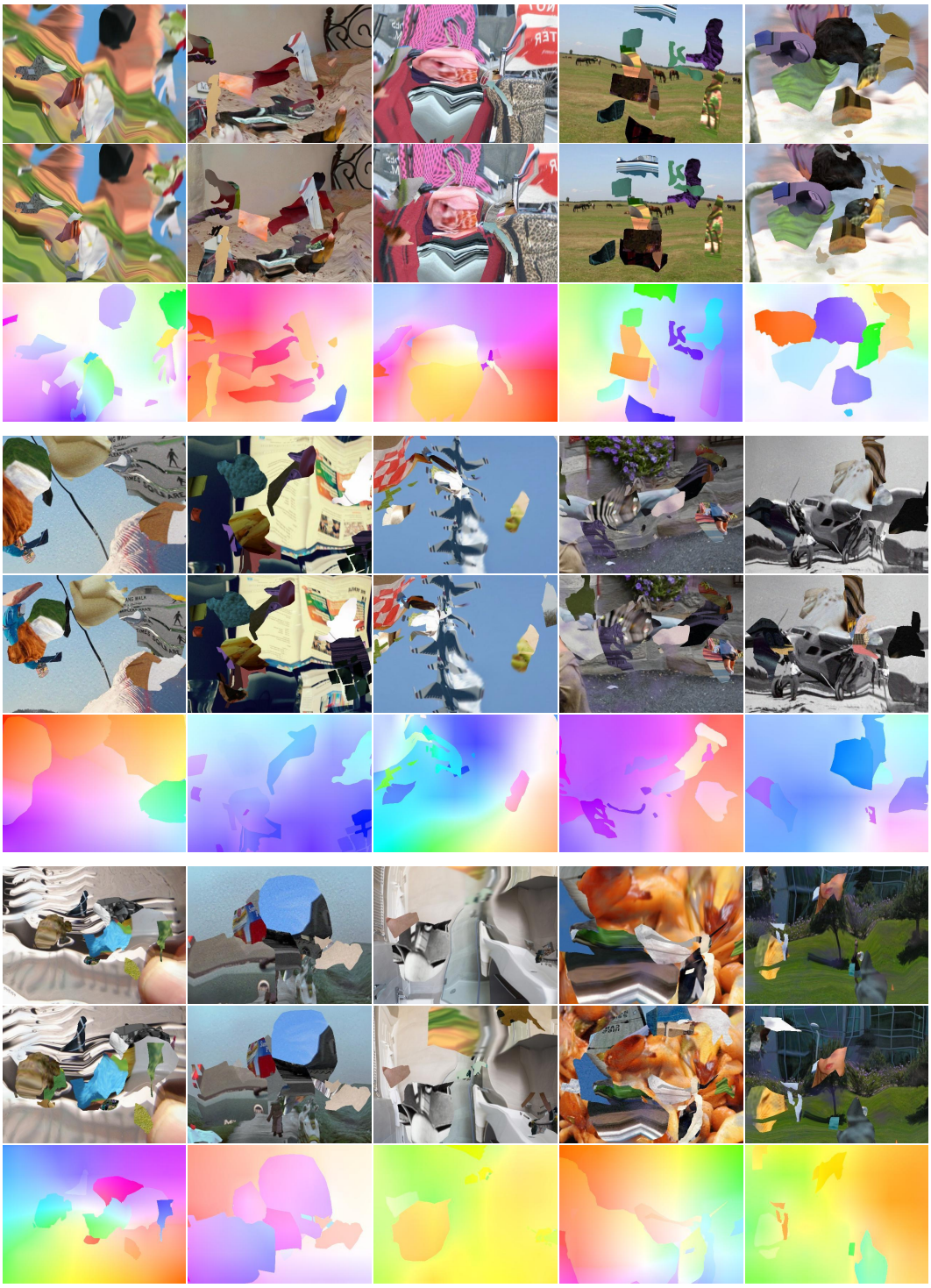

Figure 9: **Additional generated data samples.** Top: current frames; Middle: next frames; Bottom: visualizations of ground truth optical flow.