Assisting Unsupervised Optical Flow Estimation with External Information

by

Shuai Yuan

Department of Computer Science
Duke University

Defense Date: Nov 15, 2023

Approved:

Carlo Tomasi, Supervisor

Ronald E. Parr

Cynthia D. Rudin

Boyuan Chen

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2023

ABSTRACT

Assisting Unsupervised Optical Flow Estimation with External Information

by

Shuai Yuan

Department of Computer Science
Duke University

Defense Date: Nov 15, 2023

Approved:

Carlo Tomasi, Supervisor

Ronald E. Parr

Cynthia D. Rudin

Boyuan Chen

An abstract of a dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2023

Abstract

Optical flow estimation is a long-standing problem in computer vision with broad applications in autonomous driving, robotics, *etc.*. Due to the scarcity of ground-truth labels, the unsupervised estimation of optical flow is especially important. However, it is a poorly constrained problem and presents challenges in the presence of occlusions, motion boundaries, non-Lambertian surfaces, lack of texture, and illumination changes. Therefore, we explore using external information, namely partial labels, semantics, and stereo views, to assist unsupervised optical flow estimation.

Supervised training of optical flow predictors generally yields better accuracy than unsupervised training. However, the improved performance comes at an often high annotation cost. Semi-supervised training trades off accuracy against annotation cost. We use a simple yet effective semi-supervised training method to show that even a small fraction of labels can improve flow accuracy by a significant margin over unsupervised training. In addition, we propose active learning methods based on simple heuristics to further reduce the number of labels required to achieve the same target accuracy. Our experiments on both synthetic and real optical flow datasets show that our semi-supervised networks generally need around 50% of the labels to achieve close to full-label accuracy, and only around 20% with active learning on Sintel. We also analyze and show insights on the factors that may influence active learning performance. Code is available at https://github.com/duke-vision/optical-flow-active-learning-release.

Unsupervised optical flow estimation is especially hard near occlusions and motion boundaries and in low-texture regions. We show that additional information such as semantics and domain knowledge can help better constrain this problem. We introduce SemARFlow, an unsupervised optical flow network designed for autonomous driving data that takes estimated semantic segmentation masks as additional inputs. This additional information is injected into the encoder and into a learned upsampler that refines the flow output. In addition, a simple yet effective semantic augmentation module provides self-supervision when learning flow and its boundaries for vehicles, poles, and sky. Together, these injections of semantic information improve the KITTI-2015 optical flow test error rate from 11.80% to 8.38%. We also show visible improvements around object boundaries as well as a greater ability to generalize across datasets. Code is available at https://github.com/duke-vision/semantic-unsup-flow-release.

Both optical flow and stereo disparities are image matches and can therefore benefit from joint training. Depth and 3D motion provide geometric rather than photometric information and can further improve optical flow. Accordingly, we design a first network that estimates flow and disparity jointly and is trained without supervision. A second network, trained with optical flow from the first as pseudo-labels, takes disparities from the first network, estimates 3D rigid motion at every pixel, and reconstructs optical flow again. A final stage fuses the outputs from the two networks. In contrast with previous methods that only consider camera motion, our method also estimates the rigid motions of dynamic objects, which are of key interest in applications. This leads to better optical flow with visibly more detailed occlusions and object boundaries as a result. Our unsupervised pipeline achieves 7.36% optical flow error on the KITTI-2015 benchmark and outperforms the previous state-of-the-art 9.38% by a wide margin. It also achieves slightly better or comparable stereo depth results. Code will be made available.

Dedication

To my family.

Contents

List of Tables

List of Figures

Acknowledgements

There are countless people I would like to thank for their guidance, support, encouragement, and companionship throughout my PhD journey. Among them, I would like to express my special gratitude to the following people.

First and foremost, I want to thank my PhD advisor Dr. Carlo Tomasi for his years of guidance and support. He taught me to find and think about valuable research problems that really matter to the field, to better organize and write clear and concise research papers, and to maintain good academic reputation and connections in the research community. His passion and rich experience has inspired me and guided me to success in my research projects.

I also would like to thank my committee members, Dr. Ronald Parr, Dr. Cynthia Rudin, Dr. Boyuan Chen, and Dr. Marry Cummings, for their teachings and valuable discussions on my research. Thank you to Marilyn Butler, Elizabeth Labriola, Jennifer Schmidt, and Pamela Spencer for always being so supportive and helpful with logistics and administrative tasks. Thank you to our Duke Computer Science Lab staff, especially Joe Shamblin, for assisting me with all my experiment-related requests in a very timely manner. Lastly, thank you to Duke Computer Science Department for giving me this precious opportunity to pursue my PhD and being my new home as an international student.

Thank you to my wonderful computer vision lab mates, Shuzhi Yu, Hannah Kim, Xian Sun, Kelsey Liebermsan, Swarna Kamlam Ravindran, Lesia Semenova, Maria Izzi, Kahlia Hogg, Song Yuan, Xiaohan Wang, and Runyu He, for valuable research discussions and daily companionship during my PhD. I also would like to thank my other research collaborators at Duke, including Xiang Wang, Chenwei Wu, Fanjie Kong, Weituo Hao, and Dr. Rong Ge, for your inspiration and cooperation. Thank

you to my roommates and friends at Duke, Bide Chen, Junsuke Nohara, Weilun Guo, Ziwei Zhu, Wentao Wang, Boyang Pan, Kaixiang Wang, and Xin Lin, for your help and care during my PhD.

I am also very grateful to my family. Thank you to my parents for your unconditional love and support. Despite the 12-hour time difference and the vast 7,600 miles that separate us, your unwavering support and uplifting encouragement have been a constant source of strength during the most challenging moments of my PhD journey.

I also would like to appreciate the three summer internships at Meta Reality Labs. Thank you to my mentors, Denis Demandolx, Rakesh Ranjan, Bingxiong Lin, and Ergys Ristani, as well as my project peers, Lei Luo, Harry Hui, Can Pu, Fan Zhang, Shangyi Cheng, Rohan Chabra, and Po-Chen Wu, for introducing me to a new cutting-edge research area in virtual/augmented/mixed reality and showing me how a good team of researchers operates.

Lastly, I would like to thank the following agencies for financially supporting my PhD research: Duke University, National Science Foundation (NSF), Intelligence Advanced Research Projects Agency (IARPA), and Amazon AWS cloud computing awards.

Chapter 1

Introduction

1.1 Problem Definition

Optical flow estimation is a long-standing problem in computer vision and has broad applications in autonomous driving [3], virtual/augmented reality [4], and robotics [5]. As a pixel-level video motion representation, optical flow has been applied to many down-stream tasks such as action recognition [6], object tracking [7], video decomposition [8], video interpolation [9], and video editing [10].

Optical flow is defined as the field of relative displacements of each pixel between two consecutive video frames. Given two consecutive frames in the same video, for each pixel in the first frame, we find its correspondence in the second frame. Each displacement vector pointing from one pixel to its correspondence is called an *optical flow* vector. We estimate the optical flow vector for every pixel in the first frame to constitute a *dense* optical flow estimation. Although the trajectory of each moving point across multiple frames can also be analyzed, we focus on two-frame motions, *i.e.,* optical flow, for simplicity.

1.2 Current Methods and Issues

Traditional Methods Variational methods for computing optical flow were first proposed in the 1980s, including the Lucas-Kanade method [11] and the Horn-Schunk method [12]. These methods tried to solve the optical flow prediction problem by designing algorithms by hand to find displacement vectors that match similar patches in the two frames [13, 14, 15]. These algorithms were based on modeling the geometry of motion by hand rather than based on data.

Since traditional methods rely on patch matching, the textures of the patches play a very important role. If a point has poor texture, it is easily confused with neighboring points. Mathematically, this leads to ill-posed degenerate equations for traditional methods. This degeneracy is called the *aperture problem* and is pervasive in real-life videos [16]. Therefore, most traditional methods can only output a sparse optical flow field, where only points with good features are matched [17].

Supervised Methods Inspired by the success of deep Convolutional Neural Network (CNN) models for image classification and object recognition [18], much recent work has modeled the optical flow prediction problem under the supervised learning framework [19, 20, 21, 22, 23, 24, 25, 26]. Under this setting, deep neural networks are used to learn the mapping from image pairs to the corresponding optical flow by minimizing the supervised loss, namely, some distance measure between computed and true flow. This supervised learning setting is strongly data-driven, in that the geometry of motion is not modeled explicitly but learned through examples, thanks to the strong expressive power of CNNs.

One of the main obstacles to the supervised learning of motion is the difficulty in obtaining a large number of ground-truth labels. Since manual labeling is almost impossible for ground-truth dense optical flow, given both the huge amount of pixels in a video sequence and the difficulty in knowing true flow in the first place, many methods have used synthetic data in training. However, it is still an open question whether synthetic data are an adequate proxy for real data. Most supervised networks are still hard to generalize across data domains.

Unsupervised Methods Due to lack of ground-truth labels, many methods have tried to predict optical flow in an unsupervised or self-supervised way, where they use surrogate loss terms based on the assumptions of constant brightness and smooth flow [27, 28, 1, 2, 29]. Thanks to unsupervised training, it is now possible to train flow networks directly on large real datasets from the target domain.

Still, unsupervised optical flow estimation is very challenging. First, the mapping between pixel positions in two consecutive frames is not one-to-one along motion discontinuities: A background pixel visible in the first frame may no longer be visible in the second, because the corresponding world point has become hidden behind the foreground object. This is called an *occlusion*, and the same phenomenon viewed in reverse time is called a *disocclusion*.

Second, most optical flow estimation methods assume that flow varies smoothly over the image, in order to provide a regularizer for an otherwise ill-posed problem, and implement this assumption by charging a *smoothness loss* for abrupt spatial changes of optical flow. This assumption is not valid along motion boundaries, and the resulting flow estimates are consequently poor there.

1.3 Assistance from External Information

To help better constrain unsupervised optical flow estimation, we consider injecting various kinds of additional information as long as they are practical and feasible.

1.3.1 Partial Labeling

Many methods have achieved remarkable performances on either the supervised or the unsupervised learning of optical flow. However, the semi-supervised learning of optical flow has not been well explored.

Since optical flow labels are especially difficult to obtain, many research papers have been focusing on unsupervised models. However, though labelling the whole dataset is expensive, it may be still feasible to label a part of the data, and using those partial labels may help boost the performance of unsupervised models. Therefore, our goal for semi-supervised training is to obtain the best possible performance under certain constraints of label ratios.

Another angle to view this problem is through *label efficiency*. Under the same setting (the same data, network architecture, etc.), supervised model generally works better than the unsupervised model. However, there should be a whole spectrum between "supervised" (100% label) and "unsupervised" (0% label). Intuitively, as we inject a small fraction of labels (such as 10%) into training, the performance should improve abruptly instead of linearly. This suggests that partial labeling may be a practical solution to improve unsupervised optical flow.

When exploring semi-supervised learning with partial labels, another interesting question is: how to pick the samples to label? Or in other words, compared with random sampling, is there a smarter way of choosing which samples to label? This brings us to the concept of active learning, where the model is allowed to actively select samples to query labels with human-in-the-loop training.

Details of this project are discussed in Chapter 2.

1.3.2 Semantics

Since optical flow resolves motion at the pixel level, no object-level information is typically utilized explicitly in its estimation. Therefore, it is interesting to see how incorporating the knowledge of object semantics can help refine optical flow.

Smantics has been extensively studied as an additional, important source of visual information, and much progress has been made in recent decades on how to incorporate it. Unlike optical flow labels, semantic labels are relatively more feasible and easier to obtain. Many semantic models have been pretrained on large datasets, so they can be used off-the-shelf. For applications like autonomous driving, semantic segmentation is usually performed as part of the first steps to perceive the environment, so it is fair to assume that semantic segmentation is available with little extra cost in real applications.

Moreover, semantics can help the reasoning of optical flow. Different types of objects may have different motion distributions, which can add a strong prior for flow estimation. It may also help better tackle occlusions, which always happen between semantic objects. It is also possible to borrow semantic boundaries from the segmentation to sharpen flow boundaries. Therefore, we propose a simple yet effective model to inject semantic segmentation as an external input to improve unsupervised optical flow estimation.

Details of this project are discussed in Chapter 3.

1.3.3 Stereo or Multiple Views

Images are 2D projections of the 3D real world. When we infer optical flow from 2D image inputs, the lack of the third dimension, depth, has created confusions in various ways such as occlusions. Therefore, 3D scene geometry can be helpful to compensate those lost motion information.

For example, if a background pixel is occluded by a foreground object, that pixel is not seen in the second frame, so its correspondence cannot be found, and its optical flow will not be observable. However, if we know the 3D position of that pixel and its 3D rigid motion, we can reconstruct its motion in the 3D world, which can be later projected to the 2D image frame as optical flow. This reconstruction should be much more reliable than “guessing” the occluded flow based on 2D smoothness.

Fig 1.1 illustrates how 2D videos are captured and generated from 3D scenes. Changes between the two captured frames are a result of the motion of image points (optical flow), which in turn depends on the scene structure (3D coordinates of world points) and on the motion of both camera and objects in the scene. This simple generative model does not account for other factors that cause image change, such as changes of lighting. The term "3D scene geometry" encompasses both structure and motion.

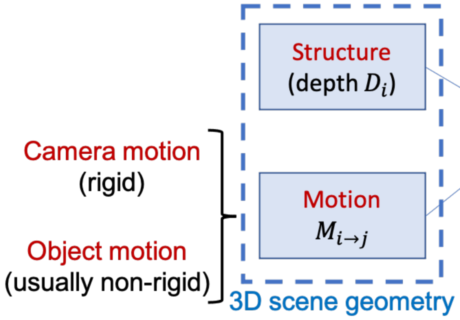


Figure 1.1: How 2D videos are captured and generated from 3D scenes

Specifically, the scene structure is usually represented by a depth map, which tells for each pixel the distance from its corresponding real-world point to the camera center of projection and is measured along the optical axis of the camera. The depth can be reliably estimated given stereo camera inputs. The motion between frames consists of camera motion, which is strictly rigid, and independent object motion, which can be non-rigid. Even if the object motion is non-rigid, it can usually be decomposed and approximated as motions of many rigid bodies. The motion of a set of points is *rigid* when it leaves the distance between any two points in the set unchanged, and a rigid motion has six degrees of freedom.

Given the camera calibration information, the structure (depth map) can be used to generate a 3D point cloud that reflects the 3D world coordinates of each visible pixel in the frame. We then apply the rigid motions to obtain the new point coordinates in the next frame, which are in turn projected on the image. Thus, the motion of each pixel, which is called the optical flow, is determined, and the next video frame changes accordingly. In this generation process, the video frames are our observed data. Therefore, the estimation of structure and motion is conceptually an inference problem in the reverse direction of this generation process.

We consider autonomous driving as a major application, where stereo cameras are placed on top of a vehicle facing straight to the front. Most parts of the scene, including the road, buildings, trees, and so on, are static, so the motion of corresponding regions in the frame is caused completely by the camera motion. There are indeed non-static regions such as moving cars, pedestrians, and cyclers, but these objects often take up a very small area of the frame, and their motions can usually be approximated by some other rigid motions. Once we have computed reliable depths (from stereo matching) and 3D rigid motions (from traditional methods or neural networks), we can reconstruct optical flow from those external stereo information.

Details of this project are discussed in Chapter 4.

Chapter 2

Optical Flow Training under Limited Label Budget via Active Learning

This chapter has been published at the European Conference on Computer Vision (ECCV) in 2022 [30].

2.1 Introduction

The estimation of optical flow is a very important but challenging task in computer vision with broad applications including video understanding [31], video editing [10], object tracking [32], and autonomous driving [33].

Inspired by the successes of deep CNNs in various computer vision tasks [18, 34], much recent work has modeled optical flow estimation in the framework of supervised learning, and has proposed several networks of increasingly high performance on benchmark datasets [19, 35, 20, 36, 21, 22, 23]. Ground-truth labels provide a strong supervision signal when training these networks. However, ground-truth optical flow annotations are especially hard and expensive to obtain. Thus, many methods use synthetic data in training, since ground-truth labels can be generated as part of data synthesis. Nevertheless, it is still an open question whether synthetic data are an adequate proxy for real data.

Another way to circumvent label scarcity is unsupervised training, which does not require any labels at all. Instead, it relies on unsupervised loss measures that enforce exact or approximate constraints that correct outputs should satisfy. Common losses used in unsupervised optical flow estimation are the photometric loss, which penalizes large color differences between corresponding points, and the smoothness loss, which

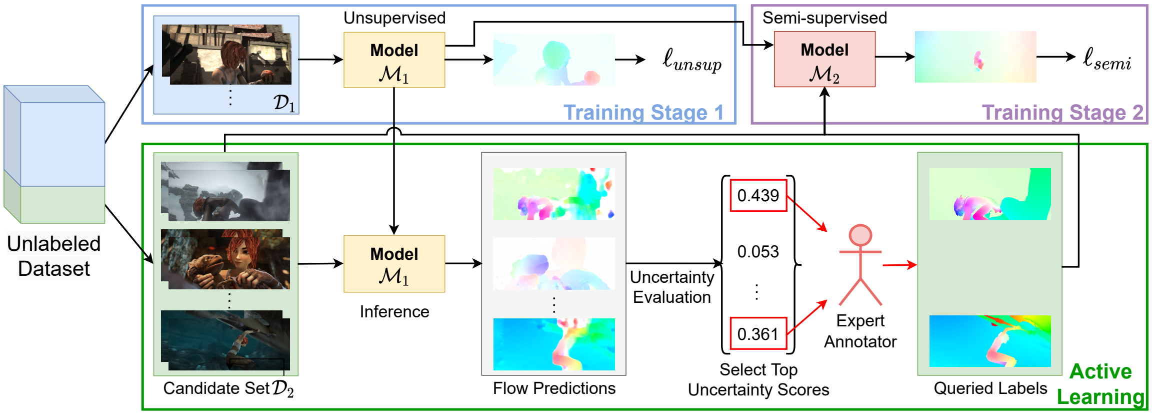


Figure 2.1: Overview of our active learning framework for the semi-supervised training.

penalizes abrupt spatial changes in the flow field [37, 38, 27, 39, 28, 1]. While unsupervised methods allow training on large datasets from the application domain, their performance is still far from ideal because the assumed constraints do not always hold. For instance, the photometric loss works poorly with non-Lambertian surfaces or in occlusion regions [40], while the smoothness loss fails near motion discontinuities [41].

Semi-supervised training can be a way to combine the advantages of both supervised and unsupervised training for optical flow models. The idea is simple, and amounts to training the network with a mix of labeled and unlabeled data. This is possible because we can charge different losses (supervised or unsupervised) to different samples depending on whether they are labeled or not.

The trade-off between performance and labeling cost is of interest in real practice, since it describes the marginal benefit that can be accrued at the price of a unit of labeling effort. However, little work has focused on the semi-supervised training of optical flow. Existing methods have tried to improve flow estimates given an available, partially labeled dataset [42, 43, 44]. Other work uses semi-supervised training to address specific problem conditions, *e.g.*, foggy scenes [45].

In contrast, we are particularly interested in label efficiency, that is, in the performance improvement gained as the fraction of labeled samples increases from 0 ("unsupervised") to 1 ("supervised"). Specifically, we use a simple yet effective semi-supervised algorithm and show that the model error drops significantly as soon as a small fraction of the samples are labeled. This suggests that even a modest labeling budget can lead to a significant performance boost.

Given a specific labeling budget, an important related question is how to determine which part of the dataset to label. A simple method is random sampling, but it is possible to do better. Specifically, we propose and evaluate criteria that suggest whose labels bring larger benefits in training. This brings us to the concept of active learning.

Active Learning (AL) has been shown to be effective in reducing annotation costs while maintaining good performance in many vision tasks including image classification [46, 47], object detection [48, 49], semantic segmentation [50, 51], and instance segmentation [52]. The general idea is to allow the training algorithm to select valuable unlabeled samples for which to query labels for further training. This selection is especially important for optical flow estimation, since generating labels for additional samples incurs high costs in terms of computation, curation, and sometimes even hand annotations.

While annotating individual flow vectors by hand is effectively impossible in practice, annotation can be and often is done by hand at a higher level and, even so, is costly. For instance, in KITTI 2015 [33], correspondences between points on CAD models of moving cars are annotated by hand so that dense optical flow can be inferred for these cars. In addition, nonrigid objects such as pedestrians or bicyclists are manually masked out, and so are errors in the flow and disparity masks inferred from LiDAR and GPS/IMU measurements and from stereo depth estimation. This is still manual annotation and curation, painstaking and expensive. Some amount of curation, at the very least, is necessary for most high-quality training sets with real imagery, and the methods we propose aim to reduce the need for this type of work, and to make the products of whatever manual work is left more effective. To the best of our knowledge, we are the first to study active learning as a way to moderate the high annotation costs for optical flow estimation.

As illustrated in Fig. 2.1, our training pipeline (top part of the diagram) includes an unsupervised first stage and a semi-supervised second stage. We split our unlabeled dataset to two sets, one ($\mathcal{D}_1$) used to pre-train an unsupervised model $\mathcal{M}_1$ and the other ($\mathcal{D}_2$) used as the *candidate* set, from which samples are selected to query labels from expert annotators. After training model $\mathcal{M}_1$ on $\mathcal{D}_1$ in Stage 1, we estimate flow for all the samples in $\mathcal{D}_2$ and score each of them based on our active learning criteria. We query for labels for top-scoring samples and add these to $\mathcal{D}_2$ for further semi-supervised training in Stage 2. In this paper, we show that using active learning to query labels can help further reduce the number of labels required to achieve a given performance target in semi-supervised training.

In summary, our contributions are as follows.

- We show on several synthetic and real-life datasets that the performance from unsupervised training of optical flow estimators can be improved significantly as soon as a relatively small fraction of labels are added for semi-supervised training.
- To the best of our knowledge, we are the first to explore active learning as a way to save annotation cost for optical flow estimation, and our novel pipeline can be used directly in real practice.
- We set up the new problem of semi-supervised training of optical flow under certain label ratio constraints. We anticipate follow-up research to propose

better methods for this problem.

2.2 Related Work

Supervised Optical Flow Supervised methods use deep networks to learn the mapping from image pairs to the corresponding optical flow by minimizing the supervised loss, namely, some distance measure between computed and true flow. FlowNet [19] used a multi-scale encoder-decoder structure with skip connections between same-scale layers. Following this framework, many networks have been proposed to decrease both model size and error. Traditional ideas or heuristics have been introduced into the network, including image pyramid in SPyNet [35], feature pyramid, warping, and cost volume in PWC-Net [20] and LiteFlowNet [36]. Iterative decoder modules have also been explored as a way to reduce model size while retaining accuracy in IRR-PWC [21] and RAFT [22]. The latter built the network based on full-pair correlations and has led to many follow-up models that have achieved the state-of-the-art performance [23].

Unsupervised Optical Flow Recent research has focused on the unsupervised learning of optical flow as a compromise between label availability and model performance. Initial work on this topic proposed to train FlowNet-like networks using surrogate loss terms, namely photometric loss and smoothness loss [37, 38]. As found by many papers, flow at occlusion region is especially challenging for unsupervised networks [40]. Thus, much research focused on solving the occlusion problem via occlusion masks [40], bi-directional consistency [27], multi-frame consistency [39, 53], and self-supervised teacher-student models [28, 54]. ARFlow [1] integrated a second forward pass using transformed inputs for augmentation and has achieved the state-of-the-art unsupervised performance. Multi-frame unsupervised models have also been investigated [39, 29].

Semi-supervised Training in Vision Semi-supervised training targets applications where partial labels are available. Early approaches in image classification [55, 56, 57, 58] utilize label propagation with regularization and augmentation based on the belief that nearby data points tend to have similar class labels. A more recent class of methods train on unlabeled samples with pseudo-labels [59, 60] predicted by a supervised trained network trained with labeled samples. Similar teacher-student models have also been explored [61, 62].

Although widely explored in many other vision tasks, there is little work on semi-supervised optical flow. Some early work utilized semi-supervised learning to achieve comparable flow accuracy to the supervised methods [43, 44, 42, 43]. Others applied semi-supervised methods to tackle specific cases of optical flow, such as dense foggy scenes[45] and ultrasound elastography[63]. In contrast, we focus on label efficiency for optical flow estimation: Instead of proposing semi-supervised networks that focus on improving benchmark performances by adding external unlabeled data, we are more focused on the trade-off between performance and label ratio given a fixed dataset.

Active Learning in Vision Active Learning (AL) aims to maximize model performance with the least amount of labeled data by keeping a human in the training loop. The general idea is to make the model actively select the most valuable unlabeled samples and query the human for labels which are used in the next stage of training. There are two main categories, namely, uncertainty-based (select samples based on some pre-defined uncertainty metric) [64, 65, 66, 67], and distribution-based (query representative samples of sufficient diversity) [68, 69].

Active learning has achieved extensive success in various fields in computer vision, including image classification [46, 47], object detection [48, 49], semantic segmentation [50, 51], and instance segmentation [52]. However, the concept has received little attention in optical flow estimation where acquiring labels is especially difficult. To the best of our knowledge, we are the first to apply active learning to optical flow estimation to reduce annotation cost.

2.3 Method

As we are among the first to explore active learning as a way to tackle the high annotation costs in optical flow training, we start from simple yet effective methods to implement our ideas. This section describes our semi-supervised training method (Sec. 2.3.1), active learning heuristics (Sec. 2.3.2), and network structure and loss functions (Sec. 2.3.3).

2.3.1 Semi-supervised Training

Given a partially labeled data set, we implement the semi-supervised training by charging a supervised loss to the labeled samples and an unsupervised loss to the unlabeled ones. Specifically, the semi-supervised loss for each sample $\boldsymbol{x}$ is

$$\ell_{\text{semi}}(\boldsymbol{x}) = \begin{cases} \ell_{\text{unsup}}(\boldsymbol{x}), & \text{if } \boldsymbol{x} \text{ is unlabeled,} \\ \alpha \ell_{\text{sup}}(\boldsymbol{x}), & \text{otherwise} \end{cases} \tag{2.1}$$

where $\alpha > 0$ is a balancing weight. We do not include the unsupervised loss for labeled samples (although in principle this is also an option) to avoid any conflict between the two losses, especially on occlusion and motion boundary regions.

Thus, the final loss of the data set $\mathcal{D} = \mathcal{D}^u \cup \mathcal{D}^l$ is

$$\mathcal{L}_{\text{semi}} = \sum_{\boldsymbol{x} \in \mathcal{D}} \ell_{\text{semi}}(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \mathcal{D}^u} \ell_{\text{unsup}}(\boldsymbol{x}) + \alpha \sum_{\boldsymbol{x} \in \mathcal{D}^l} \ell_{\text{sup}}(\boldsymbol{x}), \tag{2.2}$$

where $\mathcal{D}^u$ and $\mathcal{D}^l$ are the unlabeled and labeled sets. We define the *label ratio* as $r = |\mathcal{D}^l|/|\mathcal{D}|$. During training, we randomly shuffle the training set $\mathcal{D}$, so that each batch of data has a mix of labeled and unlabeled samples.

2.3.2 Active Learning Heuristics

Figure 2.1 shows a general overview of our active learning framework. After pre-training our model on unlabeled data (Stage 1), we invoke an active learning algorithm to determine samples to be labeled for further training. Specifically, we first use the pre-trained model to infer flow on the samples of another disjoint unlabeled data set (the *candidate* set) and select a fraction of the samples to be labeled, based on some criterion. After obtaining those labels, we continue to train the model on the partially labeled candidate set using the semi-supervised loss (Stage 2). Note that in this second stage, we do not include the unlabeled data used in pre-training (see ablation study in Sec. 2.4.5). By allowing the model to actively select samples to query labels, we expect the model to achieve the best possible performance under a fixed ratio of label queries (the "label budget").

So, what criteria should be used for selecting samples to be labeled? Many so-called uncertainty-based methods for active learning algorithms for image classification or segmentation use the soft-max scores to compute how confident the model is about a particular output. However, optical flow estimation is a regression problem, not a classification problem, so soft-max scores are typically not available, and would be in any case difficult to calibrate.

Instead, we select samples for labeling based on heuristics specific to the optical flow problem. For example, the photometric loss is low for good predictions. In addition, unsupervised flow estimation performs poorly at occlusion regions and motion discontinuities. These considerations suggest the following heuristic metrics to flag points for which unsupervised estimates of flow are poor:

- *Photo loss*: the photometric loss used in training.
- *Occ ratio*: the ratio of occlusion pixels in the frame, with occlusion estimated

by consistency check of forward and backward flows [27].

- *Flow grad norm*: the magnitude of gradients of the estimated flow field as in [70] averaged across the frame, used to indicate the presence of motion boundaries.

We experiment with three active learning methods, each using one of the metrics above. When querying labels for a given label ratio r, we first compute the metric for each sample in the candidate set, and then sort and pick the samples with largest uncertainties as our queries.

2.3.3 Network Structure and Loss Functions

Network Structure We adopt the unsupervised state-of-the-art, ARFlow [1], as our base network, which is basically a lightweight variant of PWC-Net [20]. PWC-Net-based structures have been shown to be successful in both supervised and unsupervised settings, so it is a good fit for our hybrid semi-supervised training. We do not choose RAFT because it has been mostly proven to work well in the supervised setting, while our setting (Sec. 2.4.4) is much closer to the unsupervised one (see appendix for details).

Each sample is a triple $\boldsymbol{x} = (I_1, I_2, U_{12})$ where $I_1, I_2 \in \mathbb{R}^{h \times w \times 3}$ are the two input frames and U_{12} is the true optical flow (set as "None" for unlabeled samples). The network estimates a multi-scale forward flow field $f(I_1, I_2) = \{\hat{U}_{12}^{(2)}, \hat{U}_{12}^{(3)}, \cdots, \hat{U}_{12}^{(6)}\}$, where the output $\hat{U}_{12}^{(l)}$ at scale l has dimension $\frac{h}{2^l} \times \frac{w}{2^l} \times 2$. The finest estimated scale is $\hat{U}_{12}^{(2)}$, which is up-sampled to yield the final output.

Unsupervised Loss For unsupervised loss $\ell_{\text{unsup}}(\boldsymbol{x})$ we follow ARFlow[1], which includes a photometric loss $\ell_{\text{ph}}(\boldsymbol{x})$, a smoothness loss $\ell_{\text{sm}}(\boldsymbol{x})$, and an augmentation loss $\ell_{\text{aug}}(\boldsymbol{x})$:

$$\ell_{\text{unsup}}(\boldsymbol{x}) = \ell_{\text{ph}}(\boldsymbol{x}) + \lambda_{\text{sm}}\ell_{\text{sm}}(\boldsymbol{x}) + \lambda_{\text{aug}}\ell_{\text{aug}}(\boldsymbol{x}). \tag{2.3}$$

Specifically, given the sample $\boldsymbol{x}$, we first estimate both forward and backward flow, $\hat{U}_{12}^{(l)}$ and $\hat{U}_{21}^{(l)}$, and then apply forward-backward consistency check [27] to estimate their corresponding occlusion masks, $\hat{O}_{12}^{(l)}$ and $\hat{O}_{21}^{(l)}$.

To compute the photometric loss, we first warp the frames by $\hat{I}_1^{(l)}(\boldsymbol{p}) = I_2^{(l)}(\boldsymbol{p} + \hat{U}_{12}^{(l)}(\boldsymbol{p}))$, where $I_2^{(l)}$ is I_2 down-sampled to the l-th scale and $\boldsymbol{p}$ denotes pixel coordinates at that scale. The occlusion-aware photometric loss at each scale can be then defined as

$$\ell_{\mathrm{ph}}^{(l)}(\boldsymbol{x}) = \sum_{i=1}^{3} c_i \ \rho_i(\hat{I}_1^{(l)}, I_1^{(l)}, \hat{O}_{12}^{(l)}) \tag{2.4}$$

where ρ_1, ρ_2, ρ_3 are three distance measures with the estimated occlusion region filtered out in computation. As proposed in [1], these three measures are the L_1-norm, structural similarity (SSIM) [71], and the ternary census loss [27], respectively, weighted by c_i.

The edge-aware smoothness loss of each scale l is computed using the second-order derivatives:

$$\ell_{\mathrm{sm}}^{(l)}(\boldsymbol{x}) = \frac{1}{2|\Omega^{(l)}|} \sum_{z \in \{x,y\}} \sum_{\boldsymbol{p} \in \Omega^{(l)}} \left\| \frac{\partial^2 \hat{U}_{12}^{(l)}(\boldsymbol{p})}{\partial z^2} \right\|_1 e^{-\delta \left\| \frac{\partial I_1(\boldsymbol{p})}{\partial z} \right\|_1}, \tag{2.5}$$

where $\delta = 10$ is a scaling parameter, and $\Omega^{(l)}$ denotes the set of pixel coordinates on the l-th scale.

We combine the losses of each scale linearly using weights $w_{\mathrm{ph}}^{(l)}$ and $w_{\mathrm{sm}}^{(l)}$ by

$$\ell_{\mathrm{ph}}(\boldsymbol{x}) = \sum_{l=2}^{6} w_{\mathrm{ph}}^{(l)} \ell_{\mathrm{ph}}^{(l)}(\boldsymbol{x}), \quad \ell_{\mathrm{sm}}(\boldsymbol{x}) = \sum_{l=2}^{6} w_{\mathrm{sm}}^{(l)} \ell_{\mathrm{sm}}^{(l)}(\boldsymbol{x}). \tag{2.6}$$

We also include the photometric and smoothness loss for the backward temporal direction, which is not shown here for conciseness.

After the first forward pass of the network, ARFlow also conducts an additional forward pass on input images transformed with random spatial, appearance, and occlusion transformations to mimic online augmentation. The augmentation loss $\ell_{\text{aug}}(\boldsymbol{x})$ is then computed based on the consistency between outputs before and after the transformation. See [1] for details.

Supervised Loss For supervised loss $\ell_{\text{sup}}(\boldsymbol{x})$, we apply the multi-scale robust L_1-norm

$$\ell_{\text{sup}}(\boldsymbol{x}) = \sum_{l=2}^{6} \frac{w_{\text{sup}}^{(l)}}{|\Omega^{(l)}|} \sum_{\boldsymbol{p} \in \Omega^{(l)}} (\|\hat{U}_{12}^{(l)}(\boldsymbol{p}) - U_{12}^{(l)}(\boldsymbol{p})\|_1 + \epsilon)^q, \tag{2.7}$$

where $U_{12}^{(l)}$ is the down-sampled true flow to the l-th scale. A small ϵ and $q < 1$ is included to penalize less on outliers. We set $\epsilon = 0.01$ and $q = 0.4$ as in [20].

Semi-supervised Loss The semi-supervised loss is computed by Eq. (2.1).

2.4 Experimental Results

2.4.1 Datasets

As most optical flow methods, we train and evaluate our method on FlyingChairs [19], FlyingThings3D [72], Sintel [73], and KITTI [74, 33] datasets. Apart from the labeled datasets, raw Sintel and KITTI frames with no labels are also available and often used in recent unsupervised work [28, 1, 75, 76]. As common practice, we have excluded the labeled samples from the raw Sintel and KITTI datasets.

In our experiments, we also split our own train and validation set on Sintel and KITTI. We split Sintel clean and final passes by scenes to 1,082 training samples and 1,000 validation samples. For KITTI, we put the first 150 samples in each of

2015 and 2012 set as our training set, yielding 300 training samples and 94 validation samples. A summary of our data splits is in the appendix.

2.4.2 Implementation Details

We implement the model in PyTorch [77], and all experiments share the same hyperparameters as follows. Training uses the Adam optimizer [78] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and batch size 8. The balancing weight α in Eq. (2.1) is set as 1. The weights of each unsupervised loss term in Eq. (2.3) are $\lambda_{\text{sm}} = 50$ for Sintel and $\lambda_{\text{sm}} = 75$ otherwise; and $\lambda_{\text{aug}} = 0.2$ unless otherwise stated. The weights of different distance measures in Eq. (2.4) are set as $(c_1, c_2, c_3) = (0.15, 0.85, 0)$ in the first 50k iterations and $(c_1, c_2, c_3) = (0, 0, 1)$ in the rest as in ARFlow [1].

The supervised weights $w_{\text{sup}}^{(l)}$ for scales $l = 2, 3, \cdots, 6$ in Eq. (2.7) are 0.32, 0.08, 0.02, 0.01, 0.005 as in PWC-Net[20]. The photometric weights $w_{\text{ph}}^{(l)}$ in Eq. (2.6) are 1, 1, 1, 1, 0, and the smoothness weights $w_{\text{sm}}^{(l)}$ in Eq. (2.6) are 1, 0, 0, 0, 0.

For data augmentation, we include random cropping, random rescaling, horizontal flipping, and appearance transformations (brightness, contrast, saturation, hue, Gaussian blur). Please refer to the appendix for more details.

2.4.3 Semi-supervised Training Settings

The goal of this first experiment is to see how the validation error changes as we gradually increase the label ratio r from 0 (unsupervised) to 1 (supervised). We are specifically interested in the changing error rate, which reflects the marginal gain of a unit of labeling effort.

We ensure that all experiments on the same dataset have exactly the same setting except the label ratio r for fair comparison. For each experiment, the labeled set is sampled uniformly. We experiment on all four datasets independently using label ratio $r \in \{0, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1\}$ with settings below.

FlyingChairs and FlyingThings3D As a simple toy experiment, we split the labeled and unlabeled sets randomly and train using the semi-supervised loss. We train for 1,000k iterations with a fixed learning rate $\eta = 0.0001$.

Sintel Unlike the two large datasets above, Sintel only has ground-truth labels for 2,082 clean and final samples, which is too small to train a flow model effectively on its own. Thus, the single-stage schedule above may not apply well.

Instead, as is common practice in many unsupervised methods, we first pre-train the network using the large Sintel raw movie set in an unsupervised way. Subsequently, as the second stage, we apply semi-supervised training with different label ratios on our training split of clean and final samples. Note that we compute the label ratio r as the ratio of labeled samples only in our second-stage train split, which does not include the unlabeled raw data samples in the first stage. This is because the label ratio would otherwise become too small (thus less informative) since the number of raw data far exceeds clean and final data.

We train the first stage using learning rate $\eta = 0.0001$ for 500k iterations, while the second stage starts with $\eta = 0.0001$, which is cut by half at 400, 600, and 800 epochs, and ends at 1,000 epochs. Following ARFlow [1], we turn off the augmentation loss by assigning $\lambda_{\text{aug}} = 0$ in the first stage.

KITTI We apply a similar two-stage schedule to KITTI. We first pre-train the network using KITTI raw sequences with unsupervised loss. Subsequently, we assign labels to our train split of the KITTI 2015/2012 set with a given label ratio by random sampling and then run the semi-supervised training. The learning rate schedule is the same as that for Sintel above.

2.4.4 Active Learning Settings

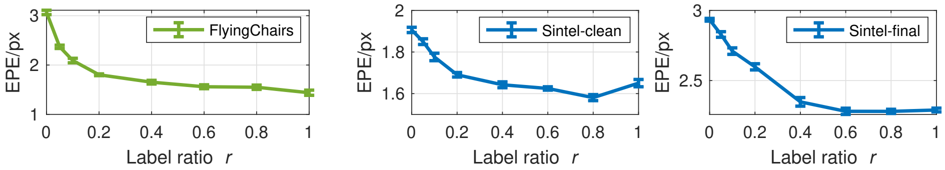
The second part of experiments is on active learning, where we show that allowing the model to select which samples to label can help reduce the error.

We mainly experiment on Sintel and KITTI since they are close to real data. Since active learning is a multi-stage process (which needs a pre-trained model to query labels for the next stages), it fits well with the two-stage semi-supervised settings described in Sec. 2.4.3. Thus, we use those settings with labels queried totally at random as our baseline. In comparison, we show that using the three active learning heuristics described in Sec. 2.3.2 to query labels can yield better results than random sampling. We try small label ratios $r \in \{0.05, 0.1, 0.2\}$ since the semi-supervised training performance starts to saturate at larger label ratios.

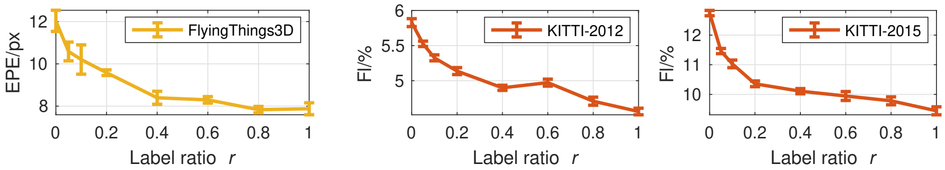
2.4.5 Main Results

Semi-supervised Training We first experiment with the semi-supervised training with different label ratios across four commonly used flow datasets. As shown in Fig. 2.2, the model validation error drops significantly at low label ratios and tends to saturate once an adequate amount of labels are used. This supports our hypothesis that even a few labels can help improve performance significantly.

Another observation is that the errors for FlyingChairs, FlyingThings3D, and Sintel saturate at around 50% labeling, whereas KITTI keeps improving slowly at high label ratios. One explanation for this discrepancy may involve the amount of repetitive information in the dataset: Sintel consists of video sequences with 20-50 frames that are very similar to each other, while KITTI consists of individually-selected frame pairs independent from the other pairs.



(a) FlyingChairs (b) Sintel



(c) FlyingThings (d) KITTI

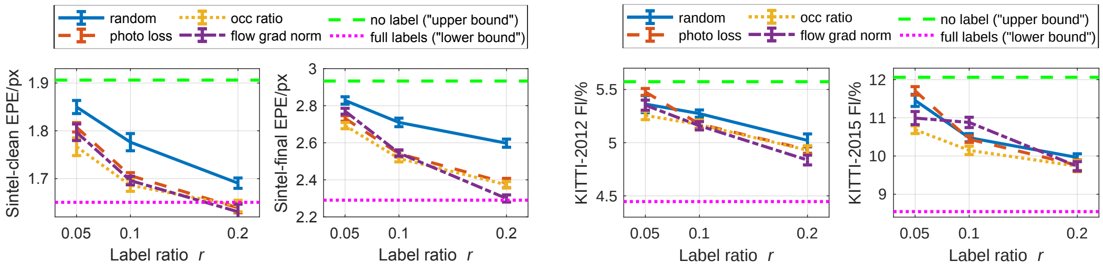
Figure 2.2: Model validation errors of the semi-supervised training with different label ratios. 'EPE': End-Point Error, 'Fl': Flow error percentage.

Active Learning Our active learning results are shown in Fig. 2.3. We compare the validation errors for our three active learning criteria against the baseline setting, in which the labeled samples are selected randomly. To better illustrate the scale of the differences, we add two horizontal lines to indicate totally unsupervised and supervised errors as the "upper" and "lower" bound, respectively.

The Sintel results (Fig. 2.3(a)) show that all our three active learning algorithms can improve the baseline errors by large margins. Notably, our active learning algorithms can achieve close to supervised performance with only 20% labeling. This number is around 50% without active learning.

The KITTI results (Fig. 2.3(b)) show slight improvements with active learning. Among our three algorithms, "occ ratio" works consistently better than random sampling, especially at a very small label ratio $r = 0.05$. We discuss the reason why our active learning methods help less on KITTI at the end of this chapter.

Among our three active learning heuristics, "occ ratio" has the best performance overall and is therefore selected as our final criterion. Note that the occlusion ratio is computed via a forward-backward consistency check, so it captures not only real occlusions but also inconsistent flow estimates.



(a) Sintel (b) KITTI

Figure 2.3: Validation errors of different active learning algorithms compared with random sampling (baseline)

Benchmark Testing We also show results on the official benchmark test sets. Qualitative examples are also included in the appendix. As is shown in Tab. 2.1, compared with the backbone ARFlow [1] and two other top unsupervised estimators [79, 28], our Sintel test EPEs improve significantly even when we utilize a very small fraction (5-20%) of labels in training. This holds true for both clean and final passes, as well as occluded and non-occluded pixels. To indicate the scale of improvements, our semi-supervised results are even comparable to the supervised IRR-PWC [21], which has a similar PWC-Net-based structure, even if we only use 20% of the Sintel labels. We also include the state-of-the-art RAFT [21] results to get a sense of the overall picture.

In addition, Tab. 2.1 also shows that our active learning method works favorably against the baseline ("rand"). We found that our active learning method ("occ") may overly sample the same scenes (*e.g.,* "ambush"), so we also test an alternative ("occ-2x") to balance the queried samples. Specifically, we select a double number of samples with top uncertainties and then randomly sample a half from them to query labels. This helps diversify our selected samples when the label ratio is very small. Our active learning methods perform comparably or better than the baseline, especially on the realistic final pass.

Table 2.2 shows our benchmark testing results on KITTI. Consistent with our

Table 2.1: Sintel benchmark results (EPE/px).

			Train		Test					
	Label ratio r	Method	Clean	Final	Clean			Final		
			all	all	all	noc	occ	all	noc	occ
unsup	$r = 0$	SelFlow [28]	(2.88)	(3.87)	6.56	2.67	38.30	6.57	3.12	34.72
		UFlow [79]	(2.50)	(3.39)	5.21	2.04	31.06	6.50	3.08	34.40
		ARFlow [1]	(2.79)	(3.73)	**4.78**	**1.91**	**28.26**	**5.89**	**2.73**	**31.60**
semi-sup	$r = 0.05$	Ours(rand)	(2.09)	(2.99)	4.04	1.52	24.65	5.49	2.62	28.86
		Ours(occ)	(1.95)	(2.38)	4.11	1.63	**24.39**	**5.28**	**2.49**	**28.03**
		Ours(occ-2x)	(1.94)	(2.55)	**3.96**	**1.45**	24.42	5.35	2.50	28.58
	$r = 0.1$	Ours(rand)	(2.36)	(3.18)	**3.91**	**1.47**	**23.82**	5.21	2.46	27.66
		Ours(occ)	(1.64)	(1.98)	4.28	1.68	25.49	5.31	**2.44**	28.68
		Ours(occ-2x)	(1.75)	(2.30)	4.06	1.63	23.94	**5.09**	2.49	**26.31**
	$r = 0.2$	Ours(rand)	(2.17)	(2.93)	3.89	1.56	**22.86**	5.20	2.50	27.19
		Ours(occ)	(1.35)	(1.63)	4.36	1.86	24.76	5.09	2.45	26.69
		Ours(occ-2x)	(1.57)	(2.05)	**3.79**	**1.44**	23.02	**4.62**	**2.07**	**25.38**
sup	$r = 1$	PWC-Net [20]	(2.02)	(2.08)	4.39	1.72	26.17	5.04	2.45	26.22
		IRR-PWC [21]	(1.92)	(2.51)	3.84	1.47	23.22	4.58	2.15	24.36
		RAFT [22]	(0.77)	(1.27)	**1.61**	**0.62**	**9.65**	**2.86**	**1.41**	**14.68**

findings on Sintel, our semi-supervised methods are significantly better than the compared unsupervised state-of-the-art methods, and close to the supervised IRR-PWC [21], even if we only use a very small fraction (5-20%) of labels. In addition, our active learning method also works consistently better than the baseline for all tested label ratios, especially on the harder KITTI-2015 set.

Ablation Study on Settings of Stage 2 We try different active learning schedules in Stage 2 and show our current setting works the best. We report the Sintel final EPE for different Stage 2 settings with label ratio $r = 0.1$. In Tab. 2.3, the first row is our current Stage 2 setting, *i.e.,* semi-supervised training on the partial labeled train set. The second row refers to supervised training only on the labeled part of train set, without the unsupervised samples. The third row considers also including the unlabeled raw data (used in Stage 1) in the Stage 2 semi-supervised training. We can see that our current setting works significantly better than the two alternatives.

Table 2.2: KITTI benchmark results (EPE/px and Fl/%).

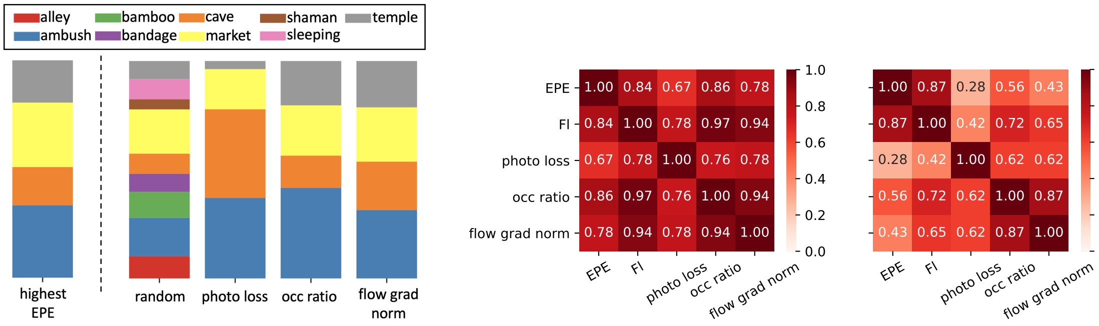
			Train		Test					
	Label ratio r	Method	2012	2015	2012		2015			
			EPE	EPE	Fl-noc	EPE	Fl-all	Fl-noc	Fl-bg	Fl-fg
unsup	$r = 0$	SelFlow [28]	(1.69)	(4.84)	4.31	2.2	14.19	9.65	12.68	21.74
		UFlow [79]	(1.68)	(2.71)	4.26	1.9	11.13	8.41	9.78	17.87
		ARFlow [1]	(1.44)	(2.85)	-	1.8	11.80	-	-	-
semi-sup	$r = 0.05$	Ours(rand)	(1.25)	(2.61)	3.90	1.6	9.77	6.99	8.33	17.02
		Ours(occ)	(1.22)	(2.29)	3.90	**1.5**	**9.65**	**6.94**	**8.20**	**16.91**
	$r = 0.1$	Ours(rand)	(1.21)	(2.56)	3.75	1.5	9.51	6.69	8.01	17.01
		Ours(occ)	(1.21)	(1.98)	**3.74**	1.5	**8.96**	**6.28**	**7.74**	**15.04**
	$r = 0.2$	Ours(rand)	(1.16)	(2.10)	3.50	1.5	8.38	**5.68**	7.37	**13.44**
		Ours(occ)	(1.13)	(1.73)	**3.49**	1.5	**8.30**	5.69	**7.25**	13.53
sup	$r = 1$	PWC-Net [20]	(1.45)	(2.16)	4.22	1.7	9.60	6.12	9.66	9.31
		IRR-PWC [21]	-	(1.63)	3.21	1.6	7.65	4.86	7.68	7.52
		RAFT [22]	-	(0.63)	-	-	**5.10**	**3.07**	**4.74**	**6.87**

Table 2.3: Ablation study: different Stage 2 settings. Sintel final validation EPE, label ratio $r = 0.1$. * denotes current setting.

Data split [Loss]	Method			
	random	photo loss	occ ratio	flow grad norm
train [semi-sup]*	**2.71(±0.02)**	**2.54(±0.02)**	**2.52(±0.02)**	**2.54(±0.02)**
train [sup]	2.82(±0.01)	2.82(±0.01)	2.59(±0.01)	2.77(±0.01)
raw+train [semi-sup]	3.13(±0.04)	3.09(±0.05)	3.15(±0.06)	3.07(±0.05)

The second setting works poorly due to overfitting on the very small labeled set, which means that the unlabeled part of the train split helps prevent overfitting. The third setting also fails due to the excessive amount of unlabeled data used in Stage 2, which overwhelms the small portion of supervised signal from queried labels.

Model Analysis and Visualization Figure 2.4(a) shows which Sintel samples are selected by different active learning methods. As shown in the left-most column, the pre-trained model after Stage 1 generally has high EPEs (top 20% shown in the figure) on four scenes, namely “ambush”, “cave”, “market”, and “temple”. The random baseline tends to select a bit of every scene, whereas all our three active learning algorithms query scenes with high EPEs for labels. This confirms that our



(a) Selected samples in Sintel (b) Sintel (c) KITTI

Figure 2.4: (a) Sintel samples selected by different methods ($r = 0.2$); correlation matrices with errors for Sintel (b) and KITTI (c).

active learning criteria capture samples that are especially challenging to the current model, which explains the success of active learning.

We also analyze the relationships between our criteria and model errors through correlation matrices visualized by heat maps in Figs. 2.4(b) and 2.4(c). We can see that the sample errors in Sintel generally have high correlations with all three score values, whereas in KITTI the correlations are much smaller. Also, the "occ ratio" score generally has the highest correlation with sample errors among the three proposed methods. All these observations are consistent with our active learning validation results. Thus, we posit that the correlation between uncertainty values and sample errors can be a good indicator in designing effective active learning criteria.

Discussion on Factors That May Influence Active Learning

- **Pattern Homogeneity**: Based on our validation results in Fig. 2.3, active learning seems more effective on Sintel than on KITTI. This may be because KITTI samples are relatively more homogeneous in terms of motion patterns. Unlike the Sintel movie sequences, which contain arbitrary motions of various scales, driving scenes in KITTI exhibit a clear *looming motion* caused by the dominant forward motion of the vehicle that carries the camera. Specifically,

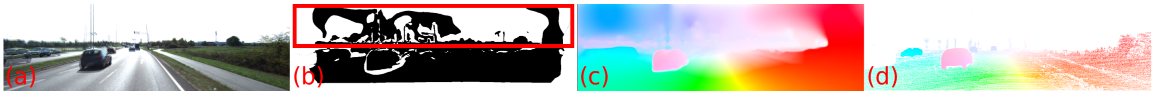


Figure 2.5: Information mismatch example: (a) input images; (b) estimated occlusion map; (c) flow prediction; (d) flow ground truth.

Sintel has extremely hard scenes like "ambush" as well as extremely easy scenes like "sleeping". This large variation of difficulty makes it possible to select outstandingly helpful samples and labels. In contrast, since KITTI motions are more patterned and homogeneous, any selection tends to make little difference with respect to random sampling.

- **Label Region Mismatch**: KITTI only has sparse labels, *i.e.,* only a part of the image pixels have labels. This is crucial because our active learning criteria are computed over the whole frame, so there is a mismatch between the support of our criteria and the KITTI labels. Specifically, the sparse labels may not cover the problematic regions found by our criteria. One example is shown in Fig. 2.5. The sky region has bad predictions due to lack of texture, and the "occ ratio" method captures the inconsistent flow there by highlighting the sky region. However, the ground-truth labels do not cover the sky region, so having this sample labeled does not help much in training.

2.5 Conclusion

In this paper, we first analyzed the trade-off between model performance and label ratio using a simple yet effective semi-supervised optical flow network and found that the unsupervised performance can be significantly improved even with a small fraction of labels. We then explored active learning as a way to further improve the performance and reduce annotation costs. Our active learning method works consistently better than baseline on Sintel and KITTI datasets.

For potential future work, it may be interesting to explore how to deal with sparse labels in the active learning framework or how to query labels by region rather than full frame.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant No. 1909821 and by the Intelligence Advanced Research Projects Agency under contract number 2021-21040700001.

Chapter 3

SemARFlow: Injecting Semantics into Unsupervised Optical Flow Estimation for Autonomous Driving

This chapter has been published at the International Conference on Computer Vision (ICCV) in 2023 [80].

3.1 Introduction

Optical flow estimation, *i.e.,* , pixel-level motion tracking across video frames, has broad applications in many computer vision tasks that include object tracking [81], video editing [10, 9], and autonomous driving [82, 83].

Thanks to the success of deep convolutional neural networks [84, 34, 85] and transformer networks [86, 87, 88, 89] in computer vision, many top-performing supervised optical flow networks have been proposed in recent years [19, 20, 21, 22, 23], in which ground-truth labels supervise training. However, real optical flow is hard to label, so most supervised methods train (or at least pre-train) on synthetic datasets [19, 90, 91, 92] which makes them hard to adapt to real applications due to the significant gap between synthetic and real data [93, 94].

Due to label scarcity, *unsupervised* training of optical flow estimators [95] instead uses loss terms based on the assumptions of constant brightness and smooth flow [27, 79, 2]. Some self-supervision techniques have also been studied to enhance model performance [96, 28, 1]. Unsupervised training makes it possible to train flow networks directly on large real datasets from the target domain.

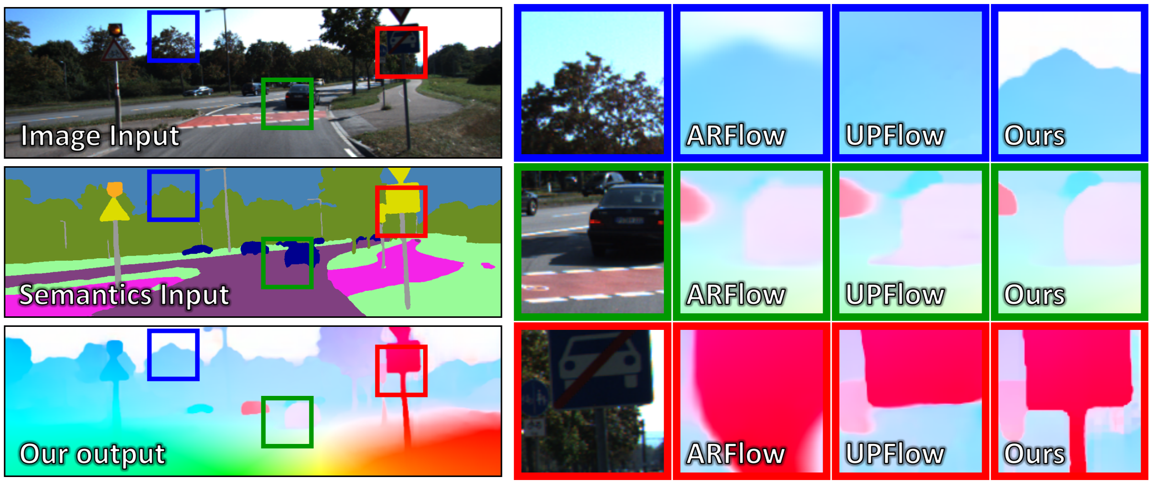


Figure 3.1: An example on KITTI-2015 test set (sample #63).

Even so, unsupervised optical flow estimation is a poorly constrained problem. Brightness is not constant in regions with occlusions [40] or on shiny surfaces [97], nor is it smooth across motion boundaries [98, 99]. Moreover, points in regions with poor texture content [17] or in dark shadows [83] are difficult to track as they are easily confused with neighboring points (the so-called aperture problem). This dearth of reliable constraints makes the unsupervised training of flow networks especially challenging.

A natural way to address this issue is to inject additional constraints in the form of semantics and domain knowledge. For example, in autonomous driving applications [100, 82], we have clear expectations about object types and layouts of the scene, as well as prior knowledge of how each type of object typically moves. We focus on this application domain and show that this additional information helps the network achieve better flow results.

To make an autonomous driving system work well in reality, it is best to train it on real data. However, annotating real driving video with optical flow labels is expensive [30], as it requires careful synchronization and calibration of diverse sensors including cameras, LiDAR, and GPS/IMU [3], aided by some manual annotation and curation based on CAD models of moving objects [33].

In contrast, annotating semantic labels seems much more feasible, and indeed semantic labels are available in most (if not all) existing driving datasets. We consider semantic segmentation because it provides semantics at the pixel level, the same level as optical flow. As one of the most popular and well-studied tasks in modern computer vision, semantic segmentation [101, 102, 103] has been extensively adopted for autonomous driving systems. In this paper, we show that adding semantic segmentation inputs helps improve unsupervised optical flow performance significantly.

Specifically, we first infer semantic segmentation maps using an off-the-shelf model [104], which of course is trained with semantic labels. An encoder with semantic map input is used to aggregate image and semantic features (Sec. 3.3.1), and a learned upsampler is added into the iterative decoder to refine flow around object boundaries given semantic inputs (Sec. 3.3.2). We also propose a simple yet effective semantic augmentation module for self-supervision, which provides realistic augmentations specific to the vehicles, poles, and sky classes based on domain knowledge and segmentation maps (Sec. 3.3.3). An occluder cache is implemented to improve efficiency (Sec. 3.3.4). Semantic augmentation provides challenging samples for self-supervision, which help train flow better in occluded regions and around foreground object boundaries.

Overall, by injecting semantic segmentation inputs, our SemARFlow network achieves significantly better unsupervised flow results both quantitatively (Sec. 3.4.3) and qualitatively (Sec. 3.4.4). Adapted from ARFlow [1], SemARFlow reduces KITTI-2015 [33] test error from 11.80% to 8.38% and out-performs the current unsupervised state-of-the-art UPFlow [2] (9.38%) by a clear margin. The example in Fig. 3.1 also demonstrates visible improvements around object boundaries. The effectiveness of each module is justified through an extensive ablation study (Sec. 3.4.5) and improvement analysis (Sec. 3.4.6). In addition to performance boost, unsupervised flow networks with additional semantic inputs generalize better across different datasets (Sec. 3.4.7).

Our research is essentially novel compared to previous approaches. Some early work incorporates semantics in traditional energy-minimization methods [11, 12] for optical flow through geometric constraints such as piece-wise rigid motion and planar surface motion [105, 106, 107, 108]. In comparison, to the best of our knowledge, we are the first to inject semantics into the unsupervised training of recent optical flow networks. Some research also trains a network for segmentation and optical flow jointly [109]. In contrast, we leverage existing, separately trained segmentation systems both because they are available and because modularity—separating segmentation from flow estimation—is important for the development of large real-application systems such as autonomous driving.

In summary, our contributions are as follows.

- To the best of our knowledge, we are the first to explore adding semantic inputs to assist the unsupervised training of deep optical flow networks.
- We propose a simple yet effective network called SemARFlow that achieves state-of-the-art results both quantitatively and qualitatively. Our model works well on real-life occlusions and yields sharp motion boundaries around objects.
- We provide full training and inference code as well as trained models to encourage follow-up research.

3.2 Related Work

Supervised optical flow Since the introduction of the pioneering deep optical flow network FlowNet [19], more and more top-performing CNN-based flow estimators have been proposed over the years [110, 111, 20, 36, 21, 22]. Recently, vision transformer networks and attention mechanism have also been applied to this problem, and these have achieved state-of-the-art performance on benchmark datasets [112, 24, 23, 113, 114]. The supervised methods are often pre-trained on large synthetic datasets such as FlyingChairs [19] and FlyingThings3D [90] before fine-tuning on the target dataset. However, there is a clear gap between the artificially generated data and real scenarios.

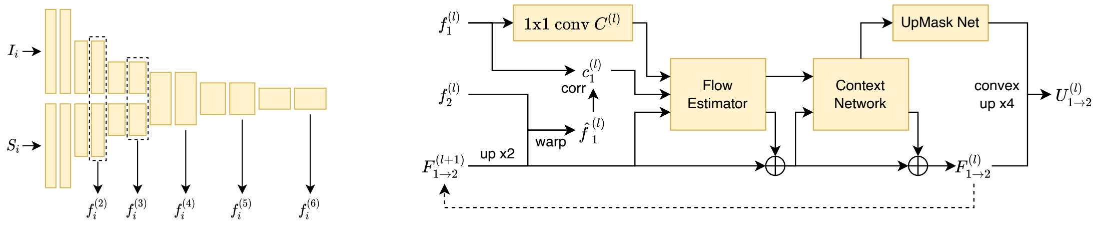
Unsupervised optical flow Due to the lack of ground-truth labels, unsupervised optical flow estimation uses surrogate losses such as photometric loss and smoothness loss to supervise training [95, 38]. To tackle the issues around occlusion regions, various methods have been proposed including estimated occlusion masks [40], forward-backward consistency [27], and multi-frame fusion [39, 53]. To better upsample the flow in the decoder, UPFlow [2] additionally predicts a confidence map and an interpolation flow to guide flow refinement and has become the state-of-the-art unsupervised flow method.

Some latest research has also explored the use of self-supervision to enhance flow prediction. Early methods apply the knowledge distillation technique to train a two-stage teacher-student network [96, 28]. ARFlow [1] further improves this idea by generating reliable self-supervision signals from data transformations, while merging the two training stages into single-stage training with one added loss term. SimFlow [115] replaces the handcrafted features with deep self-supervised features to measure similarity in the unsupervised losses. SMURF [29] utilizes a RAFT-like structure [22] and applies multi-frame self-supervised training with many technical improvements. Our SemARFlow also uses self-supervision but with the guidance of semantic segmentation, which is much more realistic than guidance without semantics.

Semantic segmentation Semantic segmentation classifies each pixel of the given image into semantic objects. Fully Convolutional Network (FCN) [101] is one of the early CNN-based segmentation methods. It takes inputs of arbitrary sizes and outputs dense pixel-level predictions, becoming one of the popular backbone architectures for follow-up work. Deconvolution network [102] is also proposed to better recover low-level details of the prediction. One main challenge for these systems is that they lack global scene information. Subsequent work addresses this issue by enlarging the receptive field of the network with global pyramid pooling layers as in PSPNet [116], hybrid dilated convolutions [117], and a fast-down-sampling strategy [118, 119]. Attention modules have also shown to help in semantic segmentation by capturing full-image dependencies of all pixels [120, 103] or the semantic interdependencies across spatial and channel dimensions [121].

There also exists extensive work on semantic segmentation in the context of autonomous driving [122, 123, 124, 125, 104], thanks to the publication of large-scale driving datasets [126, 33]. To better train the network on a coarsely labeled dataset like KITTI [33], Zhu *et al.*use a video prediction network SDCNet [127] to synthesize new training samples with relaxed label propagation [104]. Due to their good performance on KITTI [33], we utilize their network models to infer semantic inputs for all our experiments.

Combining semantics and optical flow Though there has been much progress on both semantic segmentation and optical flow estimation, the semantic optical flow problem (how to exploit semantics to help optical flow estimation) has received limited attention in recent years, and the current best results are thus much outdated. Some early methods incorporate semantics through geometric constraints to refine flow on various semantic regions, such as planar regions (using homographies) [105], static regions (using rigid camera motion and epipolar constraints) [106, 108], and



(a) shared pyramidal encoder ($i \in \{1, 2\}$)

(b) one iteration (at the lth-level) of the iterative decoder ($l \in \{6, 5, 4, 3, 2\}$)

Figure 3.2: Network structure. See text in Sec. 3.3.1 and Sec. 3.3.2 for explanations. More detailed diagrams are in appendix.

rigid objects (estimating rigid motion for each object instance) [107]. However, most methods are traditional flow methods based on energy minimization, where an initial flow estimate is usually needed and semantics is mostly used for refinement. In comparison, we explore adapting latest unsupervised optical flow networks to leverage semantic inputs in one single stage of estimation.

Apart from using semantics to help flow, some research has also explored using optical flow to help semantic segmentation [128, 129], or to train both tasks jointly [109]. There are also studies on exploiting semantics on some other correspondence matching tasks such as stereo matching [130, 131] and 3D scene flow estimation, where some additional depth cues such as stereo camera inputs [132, 133] or point clouds [134] are needed.

3.3 Method

Our network is adapted from the two-frame version of ARFlow [1], which uses a lightweight PWCNet [20] as its backbone. The inputs are two consecutive frames $I_1, I_2 \in \mathbb{R}^{H\times W\times 3}$ as well as their semantic segmentation maps $S_1, S_2 \in \{0, 1, \cdots, c\}^{H\times W}$, where the number of classes c is 19 as we use the Cityscapes format [126]. A detailed diagram of the structure of our network can be found in the Appendix.

3.3.1 Semantic Encoder

We first inject semantic information into the encoder. As shown in Fig. 3.2(a), shared by each frame $i \in \{1, 2\}$, separate convolutional layers extract features from image I_i and semantics S_i (one-hot encoded). Features from the two pipelines are concatenated and fed to additional layers. Features at different resolutions $(H/2^l, W/2^l)$ form a pyramid $\{f_i^{(l)} \mid 2 \leq l \leq 6\}$. The semantic information in these features helps delineate objects in dark shadows, where appearance is more homogeneous.

3.3.2 Iterative Decoder with a Learned Upsampler

Following [21, 1], an iterative residual refinement decoder starts from zero estimate $F_{1\to 2}^{(7)} = 0$. For iteration $l \in \{6, 5, 4, 3, 2\})$, the decoder refines feature map $F_{1\to 2}^{(l+1)}$ into $F_{1\to 2}^{(l)}$ based on features $f_1^{(l)}, f_2^{(l)}$ (Fig. 3.2(b)).

More specifically, $F_{1\to 2}^{(l+1)}$ is upsampled to match the resolution and used to warp $f_2^{(l)}$ to yield warped feature $\hat{f}_1^{(l)}$. Correlation volumes are computed between $f_1^{(l)}$ and $\hat{f}_1^{(l)}$. A one-by-one convolutional layer $C^{(l)}$ compresses the number of channels to a fixed number so that the same layer can be reused across all iterations as proposed by [21]. A flow estimator network predicts a flow residual to be added to the current estimate, and a context network then aggregates flow information spatially and refines the current flow again.

A *learned* upsampler network upsamples the final output $F_{1\to 2}^{(2)}$ in our system. This is different from ARFlow [1], which simply uses four-fold bilinear interpolation, making the final flow boundaries blurry. In contrast, our model learns to sharpen flow boundaries based on the semantic inputs, which have clear boundaries around moving objects. To this end, we add a convex upsampler network similar to the one in RAFT [22]. Different from UPFlow [2], our learned upsampler is only used

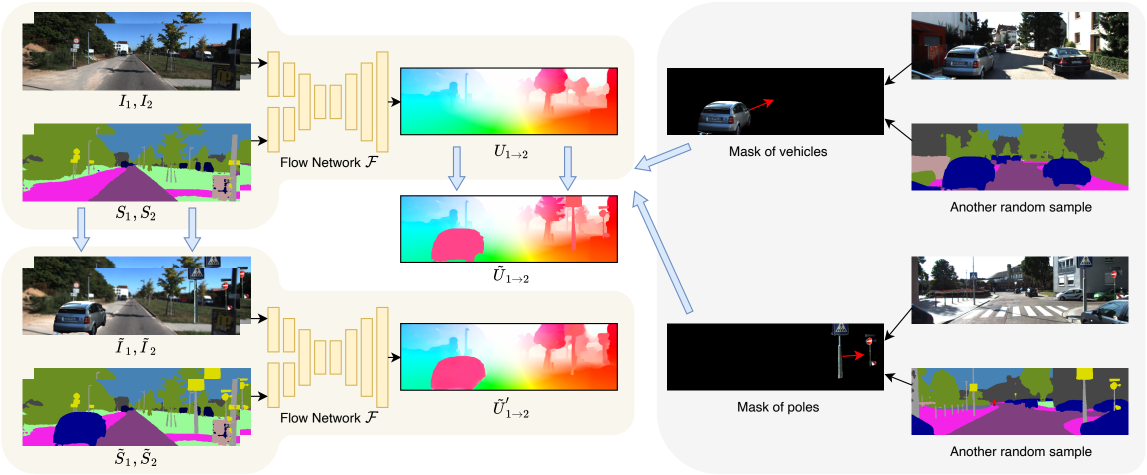


Figure 3.3: Illustration of semantic augmentation as self-supervision. See text in Sec. 3.3.3 for details.

to upsample from internal estimate $F^{(l)}_{1\to 2}$ to output flow $U^{(l)}_{1\to 2}$, but not when we upsample $F^{(l+1)}_{1\to 2}$ at the first step in the decoder iteration. The upsampled $U^{(l)}_{1\to 2}$ has resolution $(H/2^{l-2}, W/2^{l-2})$, so $U_{1\to 2} = U^{(2)}_{1\to 2}$ (with the original resolution) is the final flow prediction of the network.

3.3.3 Semantic Augmentation as Self-supervision

ARFlow has a very effective in-network augmentation module that samples random transformations $\mathcal{T}_{\theta_1}, \mathcal{T}_{\theta_2}$ of the flow prediction $U_{1\to 2}$ in the first pass of the network and then uses the transformed images $\hat{I}_1 = \mathcal{T}_{\theta_1}(I_1), \hat{I}_2 = \mathcal{T}_{\theta_2}(I_2)$ in a second pass. The prediction $U_{1\to 2}$ is also transformed accordingly and used to self-supervise the output of the second pass. See [1] for details.

We retain the augmentation module but make a third pass of the network using semantics-transformed inputs for self-supervision in addition to the ARFlow augmentations of appearance (*e.g.*, , color jitter, random noise) and spatial transformations (*e.g.*, , random rotation, random rescaling). The idea behind semantic augmentation is to blend in real object motions across samples.

To this end (see Fig. 3.3), we carve objects from other samples based on their semantic maps and paste them as moving foreground objects into the current sample I_1, I_2, thereby producing augmented images $\tilde{I}_1, \tilde{I}_2$. The segmentation maps S_1, S_2 and the first pass output flow $U_{1\to2}$ are transformed accordingly, and the transformed $\tilde{U}_{1\to2}$ self-supervises the third-pass network outputs $\tilde{U}'_{1\to2} = \mathcal{F}(\tilde{I}_1, \tilde{I}_2, \tilde{S}_1, \tilde{S}_2)$, where $\mathcal{F}$ is the flow network.

Similar to occlusion hallucination in SelFlow [28], our semantic augmentation also creates new occlusions and uses the reliable non-occluded flow from the first pass to self-supervise the flow on those newly occluded pixels. However, rather than superpixels filled with noise, we create occlusions with real objects, which are much more realistic.

Moreover, since we determine the motion of each occluder, we can use its known optical flow for motion self-supervision. As illustrated in Fig. 3.3, the occluder motion can be very different from that of the background. This trains the model to assume less smoothness around highly-dynamic objects and to output sharper motion estimates near motion boundaries. These hallucinated but realistic occluders also provide more examples of actively moving objects, which are relatively scarce in the original datasets.

Vehicles We augment vehicle classes (car, truck, bus, train) as they are major sources of errors in previous models (see Tab. 3.5 for statistics). We group these four classes because they are easily confused with each other in the semantic input. To carve out a single car instance with high probability, we find connected components of vehicle segmentation regions using OpenCV [135] and accept masks whose bounding boxes are 50-300 pixels wide and 50-150 pixels high. We also drop masks that occupy less than 60% area of their bounding boxes to remove largely occluded vehicles.

Poles We augment pole classes (pole, traffic light, traffic sign) not only because they are common foreground objects that cause occlusions, but also because their motions are usually poorly estimated due to their thin shapes. We use a 200-pixel wide, image-height sliding window to scan the segmentation of these pole classes and find the window with the most pole pixels. If more than 10% pole pixels occur in it, that object mask is stored. This produces groups of nearby poles rather than just individual poles, which may not provide enough augmentation by themselves.

Sky We also augment the sky through prior knowledge that flow in the sky should be very small. We shrink the sky flow estimate by half before using it for self-supervision. As shown in Fig. 3.3, the middle tree part in $U_{1\to 2}$ is very blurry, with a large part of sky mixed in there. After shrinking sky flow by half, the tree in $\tilde{U}_{1\to 2}$ now has sharper flow around the boundary between tree and sky. Using this as self-supervision helps the model sharpen the motion boundaries between sky and other regions.

Another way to apply this prior knowledge is to post-process, *i.e.,* , shrink the final *output* flow in the sky region, instead of using the shrunk flow to self-supervise. However, the latter approach is preferable because it allows the network to balance prior knowledge (self-supervision) with the current sample observation (photometric loss). Our experiments do show cases where some small objects (such as some overhead electric power lines) are misclassified as part of the sky in the semantic input. In these cases, the network learns to rely less on prior knowledge.

3.3.4 Occluder Cache

During training, we maintain a cache of occluding objects for semantic augmentation. We sample objects from the cache to augment the current batch and then push any new objects found in the batch into the cache, with random replacement if the cache is full.

Specifically, after we load a sample for training, we first use the current model to infer its optical flow and then search within the sample for any cars (or poles) that meet the standards defined in Sec. 3.3.3 as “cutouts”. For each cutout, we first find the flow vectors on all pixels that belong to that cutout and then compute the mean flow for later reference. We store the current sample, the cutout mask, and the mean flow together in the cache.

Later on, when we randomly select a cutout to augment another sample, we also retrieve the stored mean flow of this cutout, which is then augmented by random rescaling (by 0.8-1.5 times) and reversing (with 50% probability). We use the augmented mean flow to translate the entire cutout object as an occluder to generate a new sample. For realism, occluders are pasted to the same location they occupied in their image of origin. Holding occluders in a cache provides a random mixture of samples across multiple batches and makes semantic augmentation very efficient.

3.3.5 Loss Functions

Photometric loss In the first pass, forward and backward flow $U_{1\to 2}^{(l)}, U_{2\to 1}^{(l)}$ at each level are predicted and occlusion masks $O_{1\to 2}^{(l)}, O_{2\to 1}^{(l)}$ are computed by forward-backward consistency [27]. Frames are warped by

$$I_i'^{(l)}(\boldsymbol{p}) = I_j^{(l)}(\boldsymbol{p} + U_{i\to j}^{(l)}(\boldsymbol{p})), \quad (i, j \in \{1, 2\})$$

where $I_i^{(l)}$ is I_i down-sampled to the l-th scale, and $\boldsymbol{p}$ denotes pixel coordinates at that scale. Following [1], three measures, namely L_1-distance (ρ_1), structural similarity (SSIM) [71] (ρ_2), and census loss [27] (ρ_3) are linearly combined to measure photometric differences between $I_i^{(l)}$ and $I_i'^{(l)}$. Occlusion regions are masked out and both forward and backward directions are taken into account at each level. The final photometric loss is

$$\ell_{\text{ph}} = \frac{1}{2} \sum_{(i,j)\in\{(1,2),(2,1)\}} \sum_{l=2}^{6} a_l \sum_{k=1}^{3} c_k \rho_k (I_i^{(l)}, I_i'^{(l)}, O_{i\to j}^{(l)}). \tag{3.1}$$

Smoothness loss Unlike most previous work, we do not include a smoothness loss because we find it to conflict with our learned upsampler. See ablation study in Sec. 3.4.5.

Augmentation loss As in ARFlow [1], our second forward pass computes the L_1-distance between the transformed flow $\hat{U}_{1\to 2}$ and the second pass output $\hat{U}'_{1\to 2} = \mathcal{F}(\hat{I}_1, \hat{I}_2, \hat{S}_1, \hat{S}_2)$ for each pixel $\boldsymbol{p}$ as

$$\hat{D}(\boldsymbol{p}) = \|\hat{U}_{1\to 2}(\boldsymbol{p}) - \hat{U}'_{1\to 2}(\boldsymbol{p})\|_1.$$

This loss is then averaged over the transformed non-occluded region where self-supervision $\hat{U}_{1\to 2}$ is accurate

$$\ell_{\text{ar}} = \frac{\sum_{\boldsymbol{p}} (1 - \hat{O}_{1\to 2}(\boldsymbol{p})) \hat{D}(\boldsymbol{p})}{\sum_{\boldsymbol{p}} (1 - \hat{O}_{1\to 2}(\boldsymbol{p}))}. \tag{3.2}$$

For semantic augmentation (third pass), since we are also doing self-supervision with a new generated sample, we use a similar loss definition

$$\ell_{\text{aug}} = \frac{\sum_{\boldsymbol{p}} (1 - \tilde{O}_{1\to 2}(\boldsymbol{p})) \tilde{D}(\boldsymbol{p})}{\sum_{\boldsymbol{p}} (1 - \tilde{O}_{1\to 2}(\boldsymbol{p}))}, \tag{3.3}$$

where $\tilde{D}(\boldsymbol{p})$ is the distance between the semantic-augmented flow $\tilde{U}_{1\to 2}$ and the third-pass output $\tilde{U}'_{1\to 2}$

$$\tilde{D}(\boldsymbol{p}) = \|\tilde{U}_{1\to 2}(\boldsymbol{p}) - \tilde{U}'_{1\to 2}(\boldsymbol{p})\|_1,$$

and the augmented mask $\tilde{O}_{1\to 2}$ is now computed by

$$\tilde{O}_{1\to 2} = 1 - \max(1 - O_{1\to 2}, M)$$

because we penalize on both the originally non-occluded region $1 - O_{1\to 2}$ and the pasted foreground object mask M (of which we know the true motion).

Final loss (with $\lambda = 0.02$ as in ARFlow [1]):

$$\ell = \ell_{\text{ph}} + \lambda(\ell_{\text{ar}} + \ell_{\text{aug}}) \ . \tag{3.4}$$

3.4 Experiments

3.4.1 Datasets

We mainly use KITTI [74, 33] and Cityscapes [126] datasests for experiments. Following [1], we train our model first on KITTI raw sequences [33] (55.7k samples) and then fine-tune on KITTI-2015 multi-view extension [33] (11.8k samples). We validate our model using KITTI-2015 train [33] (200 samples) and KITTI-2012 train set [74] (194 samples) since they are the only sets that have optical flow labels. We also train on Cityscapes [126] sequences (83.3k samples) to test model generalization ability, and we sample every other frame to match its frame rate with KITTI. All semantic segmentation inputs are estimated using an off-the-shelf model [104] with a DeepLabV3Plus [136] backbone, which achieves 83.45% mean IoU on Cityscapes and 72.82% mean IoU on KITTI.

3.4.2 Implementation Details

We implement the model in PyTorch [77] [1]. We use the Adam optimizer [78] with $\beta_1 = 0.9, \beta_2 = 0.999$ and batch size 4. We first train on KITTI raw sequences [33] for

---

[1] Code and instructions are available at https://github.com/duke-vision/semantic-unsup-flow-release.

Table 3.1: KITTI benchmark results (EPE/px and Fl/%).

	Method	Train		Test					
		2012	2015	2012		2015			
		EPE	EPE	Fl-noc	EPE	Fl-all	Fl-noc	Fl-bg	Fl-fg
supervised	PWC-Net+ [138]	-	(1.50)	3.36	1.4	7.72	4.91	7.69	7.88
	IRR-PWC [21]	-	(1.63)	3.21	1.6	7.65	4.86	7.68	7.52
	RAFT [22]	-	(0.63)	-	-	5.10	3.07	4.74	6.87
	Separable Flow [23]	-	(0.69)	-	-	4.53	2.78	4.25	5.92
unsupervised	SelFlow [28]	1.69	4.84	4.31	2.2	14.19	9.65	12.68	21.74
	SimFlow [115]	-	5.19	-	-	13.38	8.21	12.60	17.27
	ARFlow [1]	1.44	2.85	-	1.8	11.80	-	-	-
	UFlow [79]	1.68	2.71	4.26	1.9	11.13	8.41	9.78	17.87
	UPFlow [2]	**1.27**	2.45	-	**1.4**	9.38	-	-	-
	Ours (baseline)	1.39	2.61	4.30	1.7	9.89	6.98	8.82	15.21
	Ours (+enc)†	1.29	2.42	3.97	1.5	8.99	3.97	8.19	13.01
	Ours (+enc +aug)†	1.28	**2.18**	**3.90**	1.5	**8.38**	**3.90**	**7.48**	**12.91**

100k iterations with a fixed learning rate 0.0002 and then train on KITTI multi-view extension set [74, 33] for another 100k iterations using OneCycleLR schedule [137] with maximum learning rate 0.0004 and linear annealing.

For data augmentation, we include random horizontal flipping and swapping of the input frames. We resize the inputs to 256 × 832 before feeding into the network. The photometric loss weight for each scale a_l $(2 \leq l \leq 6)$ in Eq. (3.1) are set as 1, 1, 1, 1, 0. The weights for three photometric distance measures ρ_k $(1 \leq k \leq 3)$ in Eq. (3.1) are set as 0.15, 0.85, 0 for the first 50k iterations and 0, 0, 1 afterwards. We start the appearance and spatial augmentation (second pass as in ARFlow [1]) at 50k iterations, and semantic augmentation (third pass) after 150k iterations.

3.4.3 Benchmark Testing Results

As common practice, we evaluate optical flow predictions based on two error measurements, Fl (error rate) and EPE (mean L_2 distance). When computing Fl, the

Table 3.2: KITTI-2015 test results (Fl/%) compared with other semantic optical flow methods.

Method	Fl-all	Fl-noc	Fl-bg	Fl-fg
JFS [106]	17.07	9.81	15.9	22.92
SOF [105]	16.81	10.86	14.63	27.73
MRFlow [108]	12.19	8.86	10.13	22.52
Bai *et al.* [107]	11.62	8.75	8.61	26.69
Ours (final)	**8.38**	**3.90**	**7.48**	**12.91**

estimate of each pixel is considered correct if the error is smaller than 3 pixels or 5% of the magnitude of ground-truth flow [33].

As shown in Sec. 3.4.2, our semantic modules (both semantic encoder and semantic augmentation) improve performance on both KITTI-2012 and KITTI-2015 sets on all metrics. Our final model achieves 8.38% Fl-all error rate on KITTI-2015 test set, which is significantly better than ARFlow [1] (11.80%), from which we adapt. We also outperform the current state-of-the-art UPFlow [2] (9.38%) by a clear margin. All these results strongly suggest that adding semantic inputs can help improve unsupervised flow estimation significantly.

In addition, Sec. 3.4.3 shows that our test results significantly outperform all previous semantic optical flow methods, most of which are based on traditional energy minimization. We are the first to apply semantic inputs to recent unsupervised flow networks, so we are able to push the state-of-the-art by a clear margin.

3.4.4 Qualitative Results

Some qualitative results are shown in Fig. 3.4. We can see that our flow outputs have very sharp boundaries and very clean object motion, especially around cars and poles. Our model successfully learns to adapt flow estimation to the semantic map input. Moreover, our model is able to handle very challenging samples where the foreground car motion is drastically different from the background due to the

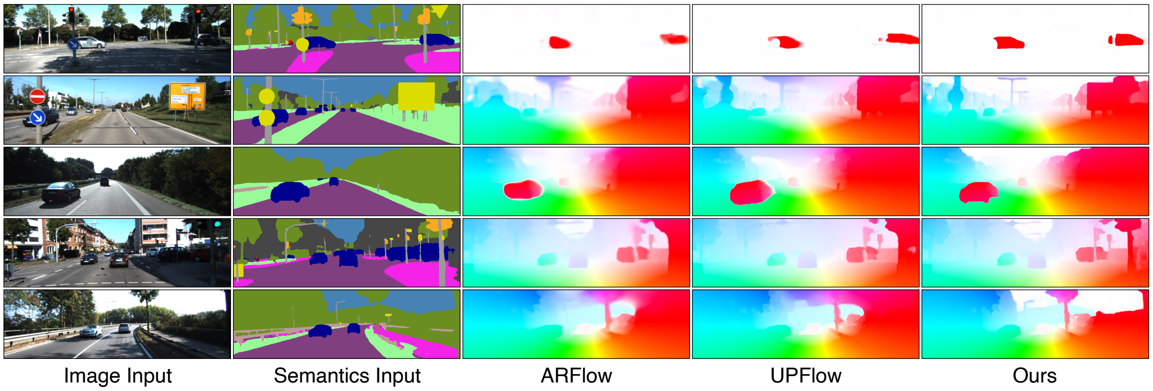


Figure 3.4: Qualitative results on KITTI test set (sample #7, 20, 38, 112, 183) compared with ARFlow [1] and UPFlow [2].

Table 3.3: Ablation study on KITTI-2015 train set (EPE/px and Fl/%).

up	no sm	enc	aug	Fl-all	EPE-all	EPE-noc	EPE-occ
				10.36	2.90	2.07	6.97
		1		9.92	2.69	1.89	6.57
		2		9.83	2.65	1.86	6.40
		3		9.75	2.61	1.85	6.54
		4		9.75	2.64	1.85	6.55
✓				10.22	2.83	1.97	6.75
✓	✓			8.87	2.61	1.85	6.04
✓	✓	3		8.26	2.42	1.73	5.79
✓	✓		✓	8.80	2.48	1.60	6.64
✓	✓	3	✓	**7.79**	**2.18**	**1.40**	**5.64**

challenging samples we create in semantic augmentation.

3.4.5 Ablation Study

We also do ablation study to show the effectiveness of each of our proposed modules. We can see in Tab. 3.3 that adding a semantic encoder to the vanilla ARFlow does help, and the optimal number of encoder layer groups added is 3. Also, our learned upsampler improves results significantly once we turn off the smoothness loss. This makes sense because the upsampler learns to adapt to boundaries, which

Table 3.4: Ablation study of different semantic augmentation options on KITTI-2015 train (EPE/px and Fl/%). See text for explanations.

Options of aug	Fl-all	EPE-all	EPE-noc	EPE-occ
Ours (final)	**7.79**	**2.18**	**1.40**	5.64
start from 100k	7.92	2.19	**1.40**	**5.41**
vehicles only	7.94	2.21	1.44	5.78
loss on new occ	8.15	2.27	1.42	5.62

can be distracted by the smoothness loss. Moreover, both our semantic encoder and semantic augmentation module help improve the results further, which suggests the effectiveness of adding semantic inputs.

Another ablation study is on different options for the semantic augmentation (Tab. 3.4). We try starting augmentation earlier from 100k iterations, which works slightly worse. There are mainly two reasons: (1) we use the first forward output to self-supervise our augmented output, so if we start early, the model will use poor output to self-supervise the augmented pass; and (2) our semantic augmentation creates very challenging samples with objects moving very differently from the background, so exposing these hard samples to the model too early may make it hard to train. Apart from this, we also try only using vehicles to augment as well as focusing loss on the newly occluded region, which are both inferior to our final version.

3.4.6 Improvement Analysis

It may be interesting to understand where our improvements come from in terms of semantic classes, so we compute the KITTI-2015 train set error for each class. As shown in Tab. 3.5, our model improves the flow on every class. After reweighing the absolute improvement of each class based on their proportion in the evaluated pixels, we find that car flow accounts for nearly half of our improvement on Fl-all, indicating the effectiveness of augmenting vehicle objects.

Table 3.5: KITTI-2015 flow error (Fl-all/%) for each semantic class. The first row shows the proportion of each class among evaluated pixels.

	road	car	terrain	vegetation	sidewalk	building	wall	pole
Proportion	42.4%	17.6%	14.0%	11.6%	6.2%	4.1%	1.3%	1.1%
ARFlow [1]	4.57	15.79	9.42	15.34	4.61	6.00	16.34	10.75
Ours (final)	3.67	10.17	8.74	13.47	3.09	4.27	11.18	9.44
Rel. improvement	19.6%	35.6%	7.3%	12.2%	33.0%	28.7%	31.6%	12.2%
Rew. contribution	19.0%	49.4%	4.8%	10.9%	4.7%	3.5%	3.4%	0.7%

Table 3.6: Generalization results (train on Cityscapes, and test on KITTI-2015 train).

Method	<u>Fl-all</u>	EPE-all	EPE-noc	EPE-occ
ARFlow (our impl.)	13.21	4.08	2.88	9.40
Ours (baseline)	12.27	3.81	2.43	9.91
Ours (+enc)†	11.28	3.33	2.12	8.75
Ours (+enc +aug)†	**10.32**	**2.64**	**1.56**	**7.12**

3.4.7 Generalization Ability

We also investigate the generalization ability of our semantic-aided flow models. We train flow models on Cityscapes and directly test them on KITTI-2015 train set without fine-tuning. Following [75], we crop the bottom 25% of the frame to remove the car logo and resize the frame to size 256 × 704. All other experiment settings are exactly the same as for KITTI. As we can see from Tab. 3.6, adding semantic inputs significantly helps our unsupervised flow model generalize and adapt better across datasets.

3.5 Conclusion and Future Work

In this paper, we show that adding semantic segmentation inputs can help significantly improve the performance of unsupervised optical flow networks on autonomous

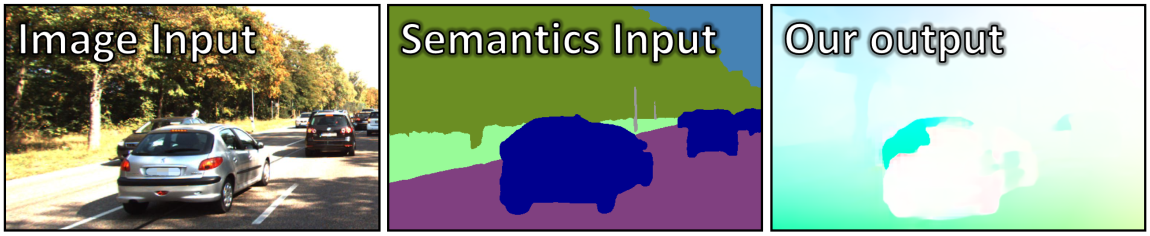


Figure 3.5: An example on KITTI-2015 train set (sample #164). See the discussion in Sec. 3.5 for details.

driving datasets. We propose a novel network model called SemARFlow, a semantic adaptation of ARFlow [1], with a semantic encoder, a learned upsampler, and a semantic augmentation module, where some domain-specific motion prior has been taken into account. Our network better predicts flow at occlusion regions and effectively sharpens flow estimates around object boundaries, even for very challenging samples. The additional semantic inputs also make our network generalize better across datasets.

One direction for future work is motivated by the example illustrated in Fig. 3.5. Our model does output sharp flow boundaries between the cars and the background, thanks to good boundary information from the semantic maps. However, the boundary between these two fast-moving cars is inferred from image information only, because their semantic maps merge into one. As a result, that part of the flow boundary is less accurate. An interesting direction for future work is to use instance-level semantics as input for more detailed object masks.

Acknowledgments. This research is based upon work supported by the National Science Foundation under Grant No. 1909821.

Chapter 4

UFD-PRiME: Unsupervised Joint Learning of Optical Flow and Stereo Depth through Pixel-Level Rigid Motion Estimation

This chapter has been published as an arXiv preprint [139] and is under review for a conference.

4.1 Introduction

The estimation of optical flow and stereo depth are long-lasting problems in computer vision. They help intelligent systems understand 3D structure and motion in applications such as autonomous driving [3], virtual/augmented reality [4], and robotics [5].

Since deep neural networks have revolutionized many traditional computer vision tasks [18, 140, 141, 9, 142], various supervised networks have been proposed to learn optical flow [19, 20, 21, 22, 23, 24, 25, 26, 143, 144, 145] and stereo depth [146, 147, 148, 149, 150, 151] end-to-end. However, these systems, and especially the most

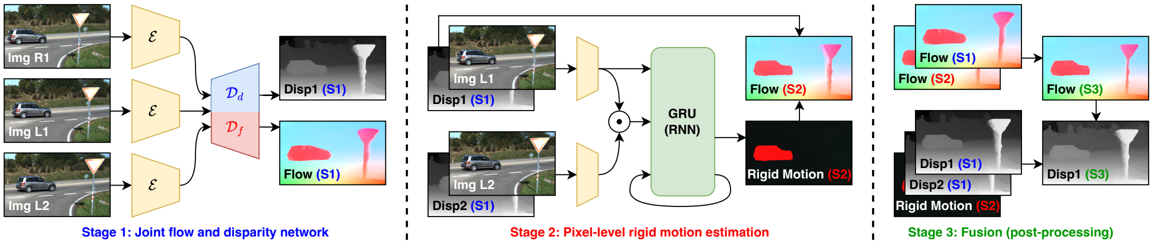


Figure 4.1: Pipeline overview.

recent ones [86, 152, 153], demand high-quality ground truth. Since annotating real data with optical flow or depth can be very expensive [30], much recent interest has turned to unsupervised training [95, 38].

Inspired by traditional methods [11, 12, 15], recent unsupervised flow and stereo matching networks rely on the constant brightness and smoothness assumptions to design loss functions [27, 28, 1, 2, 29, 154, 155, 156]. These methods share some of the issues of traditional methods with degraded estimates due to occlusions [40], motion boundaries [99], non-Lambertian surfaces [157], lack of texture [158], and illumination changes [159].

To better learn optical flow and stereo depth, a natural idea from multi-task learning [160] is to combine information from both tasks so they can benefit from each other. For example, given a stereo video sequence, stereo disparity can be estimated from left frame I_{L1} synchronized to right frame I_{R1}, and optical flow can be found from I_{L1} and its temporal successor I_{L2}. Scene depth can be computed from stereo disparity given camera calibration. Flow and disparity should be consistent in terms of structural layouts and motion patterns. They both result from image matching and can therefore be estimated jointly by a unified network that reuses features and parameters across tasks [154].

Stereo depth can also be used to reconstruct optical flow [76, 75]. Specifically, if objects do not deform, a number of 6-degree-of-freedom rigid motions occur in the field of view. Given camera calibration, the positions and 3D motions of all scene points can be computed, and the resulting 3D trajectories can be projected to the image plane to obtain optical flow. This reconstructed flow is available even if the point is occluded or out-of-sight in the next frame, so it can potentially complement the flow computed by 2D photometric matching.

Motivated by these close relationships between optical flow and stereo depth, many methods have explored training flow and depth together with rigid motions [76, 161, 75, 162, 163, 164, 165, 166, 167]. However, most methods assume stationary scenes and only estimate the camera motion (also known as "egomotion") either by traditional methods such as SfM [168, 169], PnP [170, 171], and ICP [172], or by neural networks [173, 174]. This may cause problems for dynamic objects such as moving vehicles, which are usually of key interest in autonomous driving.

Some methods compute a binary segmentation mask to indicate which pixels may be problematic during flow reconstruction [75, 165, 163, 164]. However, these masks are hard to learn *per se* [175]. Even when they are correct, the rigid motions of dynamic objects are typically discarded rather than corrected. We argue that a more detailed rigid motion representation can handle dynamic objects better.

To this end, we propose UFD-PRiME, an unsupervised joint model for optical flow and stereo depth inference with a specific focus on rigid motion estimation for every pixel. As shown in Fig. 4.1, our system has three stages (Sec. 4.3.2). We first train a light-weight joint network adapted from ARFlow [1] to obtain good initial estimates of flow and disparity (Sec. 4.3.3). Subsequently, we adapt RAFT-3D [176] to generate a dense rigid motion map for flow reconstruction (Sec. 4.3.4). Finally, we fuse and refine our results from previous stages (Sec. 4.3.5). To the best of our knowledge, we are the first to introduce dense rigid motion maps to the unsupervised training of flow and stereo depth.

Our system outperforms all previous methods both quantitatively (Sec. 4.4.3) and qualitatively (Sec. 4.4.4). Our final stage achieves 7.36% optical flow error on the KITTI-2015 benchmark [33], which is significantly better than previous state-of-the-art systems UPFlow [2] (9.38%) and FLC [164] (9.70%), while maintaining marginally better stereo depths at the same time. Surprisingly, even our simple first-stage network achieves 9.01% optical flow error, already outshining the state-of-the-art methods. Moreover, scene flow evaluations suggest that our system indeed captures accurate 3D motion information (Sec. 4.4.5). Extensive ablation studies justify the effectiveness of our current network settings (Sec. 4.4.6). Despite having three stages, our system runs efficiently thanks to the small network sizes (Sec. 4.4.7).

In summary, our contributions are as follows.

- We propose a simple yet effective unsupervised joint network for optical flow and stereo depth that achieves state-of-the-art performance.
- We show the effectiveness of pixel-level rigid motion in improving optical flow results, especially on dynamic objects and occlusion regions. To the best of our knowledge, we are the first to adopt dense rigid motion maps in the unsupervised training of flow and stereo depth.
- We provide complete training and testing code together with our trained models at all stages (see supplementary material) for the sake of reproducibility.

4.2 Related Work

Optical Flow Estimation Although many successful supervised optical flow methods have been proposed in recent years [20, 21, 22, 23, 24, 25, 26], the unsupervised estimation of optical flow remains a challenging problem. Early unsupervised methods adopt photometric and smoothness losses as surrogates of ground truth [95, 38, 27]. Follow-up techniques have been proposed to better train flow, including occlusion masking [40, 27], iterative refinement [21, 1], learned upsampling [80, 2], and multi-frame fusion [39, 53, 29]. Self-supervised training has also shown to be effective in boosting model performance through teacher-student models [96, 28], augmentation loss [1], and synthetic dataset learning [177, 29, 178]. We adopt ARFlow [1] as our backbone flow network due to its simplicity and good performance.

Stereo Depth Estimation Stereo depth estimation computes the disparity between rectified stereo images. Early traditional methods develop hand-crafted matching costs [179, 180] and matching algorithms [181, 182, 183]. Supervised CNNs have also been introduced to learn disparity using 3D cost volumes [146, 147, 148, 149, 150] and recurrent field transforms [151]. Due to the scarcity of ground-truth labels, many unsupervised networks have also been proposed, which aim at learning deep matching features from confident matches [184, 185] and disparity map smoothness [186]. Depth has also been predicted from monocular images in an unsupervised manner [187, 173]. We rely on stereo depth because it is generally more stable and can generalize better.

3D Rigid Motion Estimation 3D rigid motion is a compact motion representation with six degrees of freedom [16] and has been extensively studied in traditional computer vision [188, 189, 190, 191, 192, 193]. Traditional methods estimate dense 3D rigid motions from frame correspondences and geometric constraints, which are sensitive to noise, especially for degenerate cases with co-plane/co-linear motions [194] or degenerate camera motions [189]. Many neural network methods have also been proposed to learn 3D rigid motion [173, 174, 176]. RAFT-3D [176] mimics an optimization process with Special Euclidean Lee algebra [195, 196] to recurrently refine the rigid motion map, supervised by scene flow labels. We adapt from RAFT-3D due to its top supervised performance.

Joint Learning of Flow and Depth Early methods have combined flow and disparity networks using spatial-temporal consistency [197, 154, 198, 199, 200]. Other work estimates camera motion from the input frames, flow, or features to reconstruct 3D flow, which is then constrained to be consistent with 2D optical flow estimates [76, 161, 166, 199]. Binary segmentation masks that separate camera motion from other dynamic motions have been either computed [162, 163, 167, 164] or learned [75, 165]. The results of these methods underscore the benefits or training flow and stereo depth jointly.

4.3 Method

4.3.1 Problem Definition

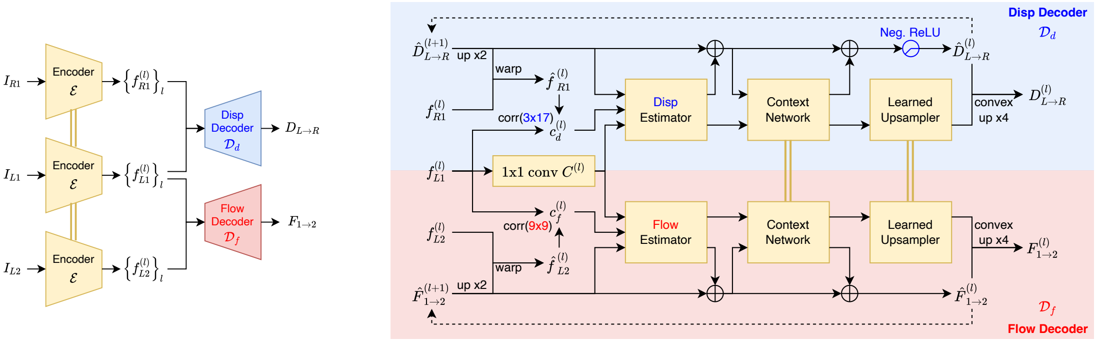
The input of our system is a set of two consecutive stereo RGB frame pairs $I_{L1}, I_{R1}, I_{L2}, I_{R2} \in \mathbb{R}^{H \times W \times 3}$, where the subscript "L/R" refers to left/right view and "1/2" refers to the first/second in temporal order. Without loss of generality, our goal is to estimate the left-view optical flow $F_{L1 \to L2} \in \mathbb{R}^{H \times W \times 2}$ and the first-time disparity $D_{L1 \to R1} \in \mathbb{R}^{H \times W \times 1}$. We assume the camera parameters are known, including camera intrinsics K and baseline distance b.

4.3.2 Overview of Three Stages

As shown in Fig. 4.1, our system contains three stages. We first train an unsupervised network to estimate optical flow and disparity jointly (Sec. 4.3.3). Images and Stage-1 disparities yield RGBD inputs for a network that estimates pixel-level 3D rigid motion and is trained with Stage-1 flow as pseudo-labels. This rigid motion map is used to reconstruct dense 3D motion, which is then projected onto the image plane to yield Stage-2 optical flow (Sec. 4.3.4). Stage 3 fuses results from previous stages into the final outputs (Sec. 4.3.5).

4.3.3 Stage 1: Joint Flow and Disparity Network

Overview The overall network structure is shown in Fig. 4.2(a). We first apply the same encoder $\mathcal{E}$ to extract multi-scale feature pyramids $f_{L1}^{(l)}, f_{L2}^{(l)}, f_{R1}^{(l)}$ on the l-th



(a) Network structure overview. (b) One iteration of the iterative decoder at the lth-level ($l \in \{6, 5, 4, 3, 2\}$).

Figure 4.2: Our Stage-1 network structure adapted from ARFlow [1]. Double lines stand for weight sharing.

level ($2 \leq l \leq 6$). Then, we use two different decoders $\mathcal{D}_f, \mathcal{D}_d$ to compute optical flow $F_{1\to 2}$ and disparity $D_{L\to R}$.

Some previous methods such as FLC [164] use one unified decoder to solve both tasks together. In contrast, we insist to use two separate decoders for flexibility. For instance, we can compute the backward flow and disparity $F_{2\to 1}$, $D_{R\to L}$ in the same pass of the network by simply swapping the feature inputs to each decoder. This property is especially important for unsupervised training because the backward flow and disparity are required when estimating occlusion masks (based on forward-backward consistency [27]) used in the unsupervised photometric loss [40].

Shared Decoders Although the flow and disparity decoders can run separately, they can share weights in some modules for joint learning. Optical flow and disparity estimation are essentially both correspondence matching problems, so the decoders should work in similar ways. However, disparity estimation is a 1D search problem rather than a 2D search like optical flow. Thus, we also make changes to the disparity decoder to utilize that special property.

We adopt the flow decoder from ARFlow [1] due to its simplicity and light weight.

As shown in the lower part of Fig. 4.2(b), our flow decoder contains a warping and correlation module, a flow estimator network, a context network, as well as a learned upsampler network suggested by SemARFlow [80]. The decoder is applied recurrently to refine flow estimate $\hat{F}^{(l)}_{1\to2}$ starting from zero flow $\hat{F}^{(7)}_{1\to2} = \mathbf{0}$ until 1/4 resolution $\hat{F}^{(2)}_{1\to2}$. The upsampled $F^{(2)}_{1\to2}$ is then used as the final output of the flow decoder. We refer readers to the original papers [1, 80] for more details.

The disparity decoder structure is shown in the upper part of Fig. 4.2(b). We copy the same structure from the flow decoder except for the following small changes.

- We change the window size of the pairwise correlation module from 9×9 to 3×17 to make the disparity decoder focus more along the horizontal direction.
- We use a new disparity estimator to replace the flow estimator network since the number of input channels has changed as we change the correlation module.
- We apply negative ReLU, which filters only negative values, before upsampling to make sure our left-to-right disparity output is negative.
- We add a redundant y-channel of all zeros to the estimated 1D disparity $\hat{D}^{(l)}_{L\to R}$ to make it 2D so that we can reuse the same context network and learned upsampler.

Loss We apply the same photometric and augmentation loss as in [80] for both flow and disparity, which are then linearly combined with weights $w_f = 0.7, w_d = 0.3$.

4.3.4 Stage 2: Pixel-Level Rigid Motion Estimation

Many previous methods have shown successful examples of reconstructing optical flow using depth maps and the 6-DoF rigid motion [165, 162, 163, 167, 164]. Nevertheless, most methods only estimate the rigid motion of the camera, assuming that the whole scene is stationary. This is problematic for dynamic objects such as moving vehicles.

Pixel-Level Rigid Motion Map In contrast, we explore estimating rigid motion for every rigid body in the view to enhance flow reconstruction. Since the number of objects in each frame is variable, inspired by RAFT-3D [176], we represent rigid motion at the pixel-level as a dense map $T \in SE(3)^{H\times W}$, where $SE(3)$ is the 3D Special Euclidean group for rigid motions [201]. The rigid motion of each pixel $T(\boldsymbol{x})$ is defined as the rigid motion of the object to which that pixel belongs. This map not only contains full rigid motion information of the scene, but also implies a segmentation of rigid bodies.

Reconstructing Flow from Disparity and Rigid Motion Suppose (x, y) is a point from the reference frame I_{L1}, and its disparity has been estimated as D. We can compute its depth $Z \in \mathbb{R}^+$ and its 3D coordinates $\boldsymbol{X} \in \mathbb{R}^3$ by

$$Z = \frac{f_x b}{|D|}, \quad \boldsymbol{X} = ZK^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{4.1}$$

where $K \in \mathbb{R}^{3\times 3}$ is the camera intrinsics matrix, f_x is the horizontal focal distance of the camera, and b is the baseline distance between left and right cameras [16].

Suppose the point belongs to a rigid body that has rigid motion $(R, \boldsymbol{t})$. The coordinates of that point in the next camera coordinates system will be $\boldsymbol{X'} = R\boldsymbol{X} + \boldsymbol{t}$, which can be projected back to the frame by

$$Z' \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = K(R\boldsymbol{X} + \boldsymbol{t}), \tag{4.2}$$

where (x', y') is the corresponding position and Z' is its new depth in the new frame.

The reconstructed optical flow is then $(x' - x, y' - y)$. Note that we only update the optical flow in Stage 2 with no refinement on disparities.

Network We borrow RAFT-3D [176] as our network structure. RAFT-3D is a supervised scene flow network that takes RGBD inputs and estimates rigid motion maps using a Dense-SE3 layer [176]. The rigid motion maps are recurrently refined and used to reconstruct scene flow, which is then supervised by labels in an end-to-end manner.

Different from RAFT-3D, our system only takes RGB inputs without depths, as defined in Sec. 4.3.1. We also do not have the scene flow labels for supervision. Therefore, we use our Stage-1 model to generate depth inputs and flow pseudo-labels to train the model.

Since our Stage-1 outputs may not be reliable in occlusion regions, we also estimate their occlusion masks through forward-backward consistency check [27]. The occlusion regions for both disparities and flows are masked out in the loss, so we only penalize at places where both our predictions and pseudo-labels are reliable. More details are included in Appendix A.2.

Loss RAFT-3D [176] is originally trained with a supervised scene flow loss, which can be computed using our pseudo-labels from Stage-1 model inference. In addition, we add a smoothness loss to better constrain our rigid motion map. Specifically, we compute the first-order gradient of the 6D rigid motion map and take its L1 norm as our smoothness loss, which is applied to estimates at all decoder iterations in accordance with the original RAFT-3D loss. We expect that the rigid motion for occlusion regions can be imputed based on local smoothness, which allows us to reconstruct occluded flow accurately.

4.3.5 Stage 3: Fusion and Post-Processing

We now have two different optical flow estimates based on photometric (Stage 1) and geometric (Stage 2) constraints, so we find a way to fuse them. Also, our disparity is not refined in Stage 2, so we refine it here based on the dense rigid motion maps.

Flow Fusion The previous stages estimate flow in two different ways, namely 2D photometric matching (Stage 1) and 3D motion reconstruction (Stage 2). These flows are reliable in different regions, so we fuse them in light of that.

We reuse the occlusion masks computed in Stage 2 to define reliable pixels for both flows. For Stage-1 flow , we define flow on non-occluded pixels as reliable. For Stage-2 flow, since it is reconstructed from the disparity map, we adopt the disparity occlusion mask and define its non-occluded pixels as reliable. Note that the flow occlusion regions are usually larger than disparity occlusions, unless the motion is very small. This indicates that our Stage-2 flow is generally more reliable than Stage 1 in typical scenarios.

To fuse the two flows, we examine their reliabilities at each pixel. If both flows are reliable, we take the average. If exactly one of them is reliable, we copy that reliable estimate. If none of them is reliable, we stick to Stage-2 flow due to its better overall performance.

Disparity Refinement We aim to refine disparities given the rigid motion map and flow. For a pixel $(x_1, y_1) \in I_{L1}$, its correspondence $(x_2, y_2) \in I_{L2}$ can be found using the fused flow above. Similar as shown in Eqs. (4.1) and (4.2), we can unproject the pixels to 3D camera coordinates and find their relationships from rigid motion $(R, \boldsymbol{t})$ as follows,

$$\frac{f_x b}{|D_2|} K^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = R \left(\frac{f_x b}{|D_1|} K^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \right) + \boldsymbol{t}, \tag{4.3}$$

where D_1, D_2 are the disparities of the same point in I_{L1}, I_{L2}. Our current estimates $\hat{D}_1, \hat{D}_2$ can be retrieved from Stage-1 model outputs (flow warping needed to compute $\hat{D}_2$), and we optimize $\delta_1 = D_1 - \hat{D}_1, \delta_2 = D_2 - \hat{D}_2$ so that Eq. (4.3) holds. We can rewrite Eq. (4.3) as

$$\frac{1}{|\hat{D}_1 + \delta_1|} \boldsymbol{\alpha}_1 + \frac{1}{|\hat{D}_2 + \delta_2|} \boldsymbol{\alpha}_2 = \boldsymbol{t}, \tag{4.4}$$

where we denote constant vectors $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in \mathbb{R}^3$ as

$$\boldsymbol{\alpha}_1 = -f_x b R K^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}, \boldsymbol{\alpha}_2 = f_x b K^{-1} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}. \tag{4.5}$$

To linearize Eq. (4.4), we do first-order Taylor's expansions for δ_1, δ_2 around zero and since $\hat{D}_1, \hat{D}_2 \leq 0$, we get

$$\left(-\frac{1}{\hat{D}_1} + \frac{1}{\hat{D}_1^2} \delta_1 \right) \boldsymbol{\alpha}_1 + \left(-\frac{1}{\hat{D}_2} + \frac{1}{\hat{D}_2^2} \delta_2 \right) \boldsymbol{\alpha}_2 = \boldsymbol{t}, \tag{4.6}$$

which is an over-determined linear system in δ_1, δ_2 and can be solved in the least-squares sense in closed form [202].

4.4 Experiments

4.4.1 Datasets

We experiment on KITTI datasets [74, 33]. For all stages, our models are first trained on KITTI raw frames (25.2k samples) and then fine-tuned on KITTI multi-view extensions (6.7k samples), as suggested by ARFlow [1]. We validate our model using KITTI-2015 [33] and 2012 [74] training sets (around 200 samples for each).

4.4.2 Implementation Details

We train our Stage 1 and 2 PyTorch [77] models (see code in the supplementary material) with the Adam optimizer [78] with batch size 4. Stage 3 requires no training.

In Stage 1, we adopt the refined schedule from SemARFlow [80] and train on raw frames for 100k iterations with a fixed learning rate 2e-4 and then on the multi-view extension set for another 100k iterations with the OneCycleLR scheduler [137] (max learning rate 4e-4). In Stage 2, we apply the same scheduler but adjust the iterations as in RAFT-3D [176]. We first train on raw for 200k iterations and then on multi-view for 50k iterations. All other implementation details are the same as in the originals (ARFlow [1] for Stage 1 and RAFT-3D [176] for Stage 2).

We augment training data with appearance transformations (brightness, contrast, saturation, hue, gaussian blur, *etc.*) and randomly swap the input images both in time (I_{L1} and I_{L2}) and space (I_{L1} and I_{R1}). With spatial swaps we also flip the images horizontally to ensure negative disparity. The inputs are resized to 384×1280 in Stage 1 but cropped to 256×832 in Stage 2 to save memory.

4.4.3 Benchmark Test Results

Optical Flow As shown in Tab. 4.1, even our Stage 1 results outperform all previous state-of-the-art unsupervised methods including FLC[164] and UPFlow [2] on all metrics on KITTI-2015 [33] and 2012 [74]. Through rigid motion estimation in Stage 2, our reconstructed flow results improve even further. After leveraging and fusing flows in the first two stages, our Stage 3 finally achieves the best test errors (7.36%) on KITTI-2015, a 20%+ decrease compared with the state-of-the-art UPFlow [2]

Table 4.1: Optical flow test errors on KITTI benchmarks (EPE/px and Fl/%).

Methods	Jt?	St?	KITTI-2012		KITTI-2015				
			train	test	train				test
			EPE	EPE	EPE-all	EPE-noc	EPE-occ	Fl-all	Fl-all
SelFlow [28]			1.69	2.2	4.84	-	-	-	14.19
ARFlow [1]			1.44	1.8	2.85	-	-	-	11.80
UPFlow [2]			1.27	1.4	2.45	-	-	-	9.38
DF-Net [161]	✓		3.54	4.4	8.98	-	-	26.01	25.70
CC-uft [75]	✓		-	-	5.66	-	-	20.93	25.27
EPC++(m) [162]	✓		2.30	2.6	5.84	-	-	-	21.56
EPC++(s) [162]	✓	✓	1.91	2.2	5.43	-	-	-	20.52
UnOS [163]	✓	✓	1.92	-	5.58	-	-	-	18.00
UnRigidFlow [167]	✓	✓	1.64	1.8	5.19	-	-	-	11.66
Flow2Stereo [154]	✓	✓	1.45	1.7	3.54	2.12	-	-	11.10
EffiScene [165]	✓	✓	1.68	-	4.20	-	-	14.31	13.08
FLC [164]	✓	✓	1.25	1.5	2.35	1.57	6.68	9.09	9.70
Ours (Stage 1)	✓	✓	1.26	1.5	2.31	1.70	5.31	7.93	9.01
Ours (Stage 2)	✓	✓	1.04	**1.2**	2.04	1.49	**4.78**	6.87	7.63
Ours (Stage 3)	✓	✓	**1.02**	**1.2**	**1.99**	**1.42**	4.87	**6.65**	**7.36**

(9.38%). These results strongly indicate that all our stages contribute significantly to improving flow results.

Stereo Depth Since we do not refine disparity in Stage 2, disparities from Stage 1 and 2 are the same. Tab. 4.2 shows that our Stage 1 and 2 results outperform the state of the art on most evaluation metrics (still comparable if not better). The Stage-3 refinement also improves the squared relative error and RMSE, indicating better depths on far-away objects. Stereo matching is known to be an easier problem than optical flow, so it has less margin for improvement.

4.4.4 Qualitative Examples

The examples in Fig. 4.3 illustrate how each stage works compared with UPFlow [2] (state-of-the-art) and ARFlow [1] (the backbone of our Stage-1 model).

Table 4.2: Stereo depth evaluation on KITTI-2015 train set compared with other unsupervised joint training of flow and stereo depth methods.

Methods	Error (Lower is Better)				Accuracy (Higher is Better)		
	Abs Rel	Sq Rel	RMSE	RMSE-log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Godard *et al.* [187]	0.068	0.835	4.392	0.146	0.942	0.978	0.989
UnOS [163]	0.049	0.515	3.404	0.121	0.965	0.984	**0.992**
UnRigidFlow [167]	0.051	0.532	3.780	0.126	0.957	0.982	0.991
EffiScene [165]	0.049	0.522	3.461	0.120	0.961	0.984	**0.992**
FLC [164]	**0.047**	**0.394**	**3.358**	0.119	-	-	-
Ours (Stage 1&2)	0.048	0.574	3.616	**0.118**	0.970	**0.986**	**0.992**
Ours (Stage 3)	**0.047**	0.565	3.588	**0.118**	**0.971**	**0.986**	**0.992**

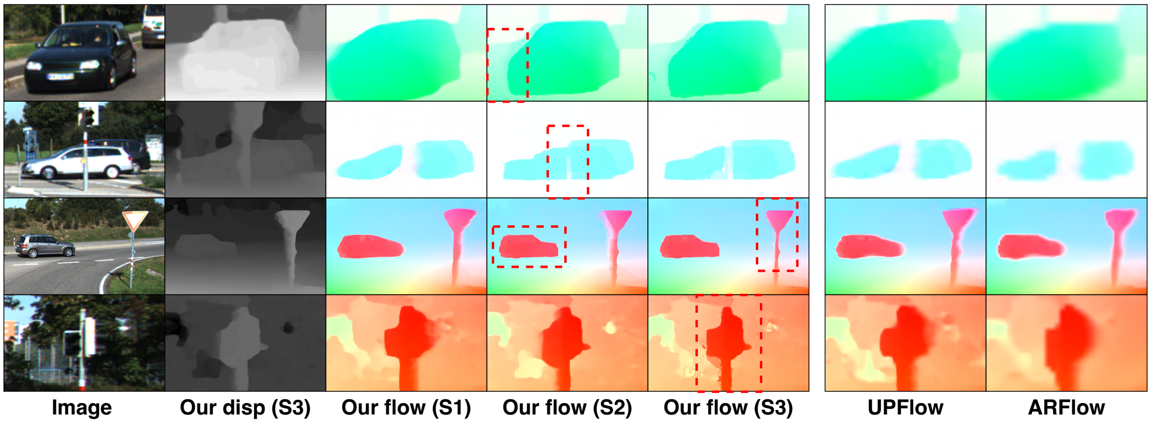


Figure 4.3: Qualitative results on KITTI test set (cropped from sample #103, 94, 78, 10) compared with UPFlow [2] and ARFlow [1].

Our Stage-1 results are visually comparable with the state-of-the-art methods. After applying pixel-level rigid motion estimation in Stage 2, our method handles occlusions around moving cars better, whether they are in the foreground (Row 1 and 3) or occluded by other objects like poles (Row 2). This is consistent with our claim that the occluded flow can be reconstructed based on stereo depth, which is generally more reliable.

Rows 3 and 4 in Fig. 4.3 show that our Stage-3 refinement can help sharpen thin foreground objects like traffic lights and poles. In this example, the traffic light is moving to the right. The Stage-1 flow is only sharp on the left side due to occlusions

Table 4.3: Scene flow evaluation of our method before and after pixel-level rigid motion estimation (Stage 2) on KITTI-2015 test benchmark.

Methods	Disparity 1			Disparity 2			Optical Flow			Scene Flow		
	bg	fg	all	bg	fg	all	bg	fg	all	bg	fg	all
UnOS	5.10	14.55	6.67	9.61	24.28	12.05	16.93	23.34	18.00	19.70	35.43	22.32
Ours (S1)	**3.65**	**15.04**	**5.55**	14.59	**20.09**	15.50	7.91	**14.52**	9.01	18.13	28.26	19.82
Ours (S2)	**3.65**	**15.04**	**5.55**	**5.30**	20.50	**7.83**	**5.96**	15.96	**7.63**	**7.64**	**26.25**	**10.74**

on the right, whereas the Stage-2 flow is only sharp on the right due to disparity occlusions on the left. By analyzing the reliable regions of each output, our fused Stage-3 output is sharp on both sides of the object. This illustrates a typical way in which Stage 3 improves, *i.e.,* by combining the flow results from photometric (Stage 1) and geometric (Stage 2) constraints that are usually reliable in different regions.

4.4.5 Scene Flow Evaluations

As in RAFT-3D [176], we evaluate our Stage-2 rigid motion estimates via scene flow due to lack of dense rigid-motion ground truth. Our Stage 1 does not output "Disparity 2" (the changed disparity of points from the first frame), so we approximate it by warping our first-pair disparity with flow. Tab. 4.3 shows that our Stage-2 rigid motion estimation very much improves scene flow results, suggesting that it captures dense 3D rigid motions well.

4.4.6 Ablation Studies

Balancing Optical Flow and Disparity in Stage 1 Tab. 4.4 shows that $w_f = 0.7, w_d = 0.3$ are the best balancing weights for flow and disparity losses in Stage 1. We also compare with settings where we train on either flow or disparity alone using the same network and show that our joint network benefits both tasks.

Table 4.4: Ablation study (Stage 1): KITTI-2015 validation errors on optical flow and stereo depth using different balancing weights w_f, w_d.

w_f	w_d	Optical Flow Error				Depth Error			
		EPE-all	EPE-noc	EPE-occ	Fl-all	Abs Rel	Sq Rel	RMSE	RMSE-log
1	0	2.38	1.73	5.56	8.02	-	-	-	-
0.9	0.1	2.38	1.74	5.52	8.07	0.049	0.618	3.692	0.120
0.7*	0.3*	**2.31**	**1.71**	5.31	**7.93**	**0.048**	**0.574**	**3.616**	**0.118**
0.5	0.5	2.34	1.73	**5.30**	7.94	**0.048**	0.593	3.628	**0.118**
0.3	0.7	2.38	1.77	5.32	8.10	0.050	0.695	3.672	0.119
0.1	0.9	36.92	27.95	73.69	83.96	0.050	0.596	3.658	0.120
0	1	-	-	-	-	0.050	0.586	3.646	0.120

Table 4.5: Ablation study (Stage 1): KITTI-2015 train set results when sharing different modules in the decoder.

Shared?			Disp corr	Optical Flow Error				Depth Error			
Fl	Cxt	Up		EPE	EPE-noc	EPE-occ	Fl	Abs Rel	Sq Rel	RMSE	RMSE-log
✓	✓	✓	9 × 9	2.37	1.71	5.51	8.06	0.051	0.754	3.785	0.122
	✓	✓	9 × 9	2.34	1.71	5.32	7.96	0.048	0.588	3.613	0.118
	✓*	✓*	3 × 17*	**2.31**	**1.70**	**5.31**	**7.93**	0.048	0.574	3.616	0.118
		✓	3 × 17	2.34	1.71	5.32	8.03	0.048	0.593	3.625	0.118
			3 × 17	2.36	1.72	5.39	8.02	**0.047**	**0.555**	**3.565**	**0.117**

Different Options of Shared Decoders in Stage 1 We examine different options for how the two decoders in Stage 1 share model weights, from the most shared (Row 1) to the least shared setting (Row 5). Tab. 4.5 shows that our final setting has the best flow results while maintaining comparable depth evaluations.

Table 4.6: Ablation study (Stage 2): KITTI-2015 train and test results using different options. * marks our final setting.

Options		Train	Test			
Mask	Smooth	Fl-all	Fl-all	Fl-noc	Fl-bg	Fl-fg
		8.47	9.53	6.93	8.38	15.25
✓		7.19	7.87	6.42	6.56	**14.42**
✓*	✓*	**6.87**	**7.63**	**6.29**	**5.96**	15.96

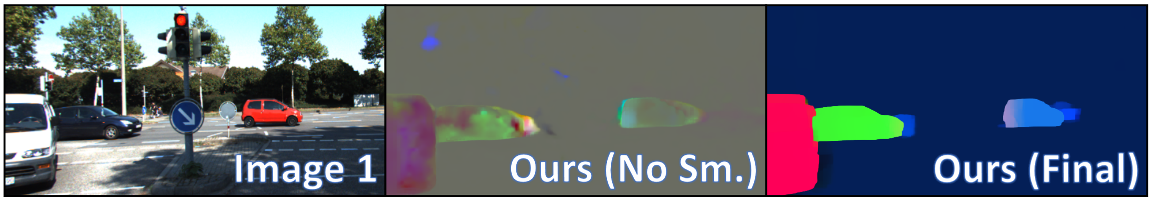


Figure 4.4: Visual comparisons of the estimated dense rigid motion fields with and without smoothness loss (KITTI test sample #8).

Changes to RAFT-3D in Stage 2 Tab. 4.6 shows that our proposed changes in Stage 2 are necessary to optimize RAFT-3D [176] in the unsupervised setting. Moreover, Fig. 4.4 visualizes the 6 DoF rigid motion field in RGB colors after reducing dimensionality to 3 with PCA [203]. The added smoothness loss clearly helps separate all rigid bodies in the frame. This also implies that our dense rigid motion map can potentially be used to generate rigid body segmentation masks, which we discuss in Appendix B.3.

4.4.7 Time Efficiency

We time our stages and show that they run efficiently. We infer input samples of dimension 376 × 1242 on one NVIDIA GeForce RTX 2080 Ti GPU. Our Stage-1 model contains 3.2 million parameters and can infer both flow and disparity together in 0.077(±0.001) seconds. Our Stage-2 model contains 6.0 million parameters and can perform dense rigid motion estimation and flow reconstruction in 0.471(±0.052) seconds. Our Stage 3 is a simple post-processing stage and runs instantly on CPUs.

4.5 Conclusion

We propose UFD-PRiME, a system for the joint unsupervised training of optical flow and stereo depth using pixel-level rigid motion estimation. We first design a simple yet effective unsupervised network based on ARFlow [1] to train flow and disparity jointly, where we use separate decoders with partial weight-sharing modules to handle the two tasks. Then, we train a pixel-level rigid motion estimation network adapted from RAFT-3D [176] to estimate dense 3D rigid motion maps, which are then used to reconstruct flow once again given depth. Finally, we fuse and refine flow and disparities by reliability analysis and stereo geometry. Our method outperforms all previous methods significantly on optical flow errors on the KITTI benchmarks [74, 33], especially on occlusion regions and around dynamic objects, while maintaining marginally better stereo depth evaluations.

Limitations Estimating rigid motion for each pixel alone can be sensitive. For each 2D motion vector observed in the image plane, there are many different 3D rigid motions that can generate the same 2D projected motion. In our system, we use a smoothness loss on the rigid motion map to alleviate this issue. A possibly better solution is to aggregate object and instance level information so that we can assign the same rigid motion to every pixel on the same rigid body. We leave this study for future work.

Chapter 5

Other Collaborative Research

5.1 Guarantees for Tuning the Step Size Using a Learning-to-Learn Approach

This paper has been published at the International Conference on Machine Learning (ICML) in 2021 [204]. I am the second author of this paper.

Choosing the right parameters for optimization algorithms is often the key to their success in practice. Solving this problem using a learning-to- learn approach—using meta-gradient descent on a meta-objective based on the trajectory that the optimizer generates—was recently shown to be effective. However, the meta-optimization problem is difficult. In particular, the meta-gradient can often explode/vanish, and the learned optimizer may not have good generalization performance if the meta-objective is not chosen carefully. In this paper we give meta-optimization guarantees for the learning-to-learn approach on a simple problem of tuning the step size for quadratic loss. Our results show that the naïve objective suffers from meta-gradient explosion/vanishing problem. Although there is a way to design the meta-objective so that the meta-gradient remains polynomially bounded, computing the meta-gradient directly using backpropagation leads to numerical issues. We also characterize when it is necessary to compute the meta-objective on a separate validation set to ensure the generalization performance of the learned optimizer. Finally, we verify our results empirically and show that a similar phenomenon appears even for more complicated learned optimizers parametrized by neural networks.

5.2 Unsupervised Flow Refinement near Motion Boundaries

This paper has been published at the British Machine Vision Conference (BMVC) in 2022 [99]. I am the third author of this paper.

Unsupervised optical flow estimators based on deep learning have attracted increasing attention due to the cost and difficulty of annotating for ground truth. Although performance measured by average End-Point Error (EPE) has improved over the years, flow estimates are still poorer along motion boundaries (MBs), where the flow is not smooth, as is typically assumed, and where features computed by neural networks are contaminated by multiple motions. To improve flow in the unsupervised settings, we design a framework that detects MBs by analyzing visual changes along boundary candidates and replaces motions close to detections with motions farther away. Our proposed algorithm detects boundaries more accurately than a baseline method with the same inputs and can improve estimates from any flow predictor without additional training.

5.3 Cross-Attention Transformer for Video Interpolation

This paper has been published at the Asian Conference on Computer Vision Workshops (ACCVW) in 2022 [9] and also received the best paper award at the Workshop on Vision Transformers: Theory and Application. I am the third author of this paper.

We propose TAIN (Transformers and Attention for video INterpolation), a residual neural network for video interpolation, which aims to interpolate an intermediate frame given two consecutive image frames around it. We first present a novel vision transformer module, named Cross-Similarity (CS), to globally aggregate input image features with similar appearance as those of the predicted interpolated frame. These CS features are then used to refine the interpolated prediction. To account for occlusions in the CS features, we propose an Image Attention (IA) module to allow the network to focus on CS features from one frame over those of the other. TAIN outperforms existing methods that do not require flow estimation and performs comparably to flow-based methods while being computationally efficient in terms of inference time on Vimeo90k, UCF101, and SNU-FILM benchmarks.

5.4 Mitigating Test-Time Bias for Fair Image Retrieval

This paper has been published at the Conference on Neural Information Processing Systems (NeurIPS) in 2023 [142]. I am the second author of this paper.

We address the challenge of generating fair and unbiased image retrieval results given neutral textual queries (with no explicit gender or race connotations), while maintaining the utility (performance) of the underlying vision-language (VL) model. Previous methods aim to disentangle learned representations of images and text queries from gender and racial characteristics. However, we show these are inadequate at alleviating bias for the desired equal representation result, as there usually exists test-time bias in the target retrieval set. So motivated, we introduce a straightforward technique, Post-hoc Bias Mitigation (PBM), that post-processes the outputs from the pre-trained vision-language model. We evaluate our algorithm on real-world image search datasets, Occupation 1 and 2, as well as two large-scale image-text datasets, MS-COCO and Flickr30k. Our approach achieves the lowest bias, compared with various existing bias-mitigation methods, in text-based image retrieval result while maintaining satisfactory retrieval performance. The source code is publicly available at https://anonymous.4open.science/r/Fair_ Text_based_Image_Retrieval-D8B2.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We conclude that unsupervised optical flow estimation can benefit greatly from the injection of external information including partial labels, semantics, and stereo views.

By adding partial labeling to unsupervised optical flow, we first analyze the trade-off between model performance and label ratio using a simple yet effective semi-supervised optical flow network and find that the unsupervised performance can be significantly improved even with a small fraction of labels. We then explore active learning as a way to further improve the performance and reduce annotation costs. Our active learning method works consistently better than baseline on Sintel and KITTI datasets.

In SemARFlow, we show that adding semantic segmentation inputs can help significantly improve the performance of unsupervised optical flow networks on autonomous driving datasets. We propose a novel network model called SemARFlow, a semantic adaptation of ARFlow [1], with a semantic encoder, a learned upsampler, and a semantic augmentation module, where some domain-specific motion prior has been taken into account. Our network better predicts flow at occlusion regions and effectively sharpens flow estimates around object boundaries, even for very challenging samples. The additional semantic inputs also make our network generalize better across datasets.

In UFD-PRiME, we propose a system for the joint unsupervised training of optical flow and stereo depth using pixel-level rigid motion estimation. We first design a simple yet effective unsupervised network based on ARFlow [1] to train flow and disparity jointly, where we use separate decoders with partial weight-sharing modules to handle the two tasks. Then, we train a pixel-level rigid motion estimation network adapted from RAFT-3D [176] to estimate dense 3D rigid motion maps, which are then used to reconstruct flow once again given depth. Finally, we fuse and refine flow and disparities by reliability analysis and stereo geometry. Our method outperforms all previous methods significantly on optical flow errors on the KITTI benchmarks [74, 33], especially on occlusion regions and around dynamic objects, while maintaining marginally better stereo depth evaluations.

6.2 Future Work

6.2.1 Multi-Frame Optical Flow/Tracking

One way to extend the two-frame optical flow is to track across multiple frames. Some early work starts from tracking across three frames, where they estimate the optical flow from a reference frame to its previous and next frames and then apply the linear motion assumption for refinement [39]. It is also possible to estimate the pairwise optical flow among a snippet of 5-10 frames, where the consistency of these optical flow can be utilized as constraints [144]. Alternatively, recent research has also proposed to analyze the motion in the full video simultaneously using global motion representations [205].

6.2.2 Integrating Pixel-Level and Object-Level Tracking

Object tracking from videos is also a very important task in real applications [206]. Unlike pixel-level tracking (*i.e.,* optical flow), object tracking relies more on reasoning about the semantics of the scene rather than geometry. In object tracking, semantic objects of interest (such as cars and humans) are usually first detected in the form of bounding boxes or segmentation masks. Subsequently, the boxes or masks are matched to associate objects in different frames.

One interesting research topic is to investigate how to integrate pixel-level and object-level tracking. As they are both matching problems, they could benefit from each other. For example, the pixel-level optical flow can be used as a strong prior when matching objects. In return, object tracking can add rich semantic information into optical flow analysis. Moreover, some feature-level integration is also possible to assist both tasks.

6.2.3 Unified Correspondence Matching with Large Models

As a simple yet essential task in computer vision, correspondence matching is used in various applications, though it may be termed differently in different contexts. In motion and tracking, it is called "optical flow estimation". In stereo vision, it is termed "multi-view stereo matching", and if two stereo cameras both face forward, perpendicular to their image planes, the problem is now called "disparity matching". In past research, these problems are solved separately with very different methods due to their different data distributions and practical requirements. For example, if the matched points are used to solve the 6-degree-of-freedom camera pose, only sparse correspondence is needed, so the methods usually prioritize finding the few most reliable matches instead of dense correspondence.

Since these tasks are all correspondence matching in nature, we could borrow ideas across methods for improvements. In addition, inspired by the latest trend of building general-purpose large models using large datasets such as Segment Anything Model (SAM) [207], it may be interesting to explore the potential of a "Correspond Anything Model" that is pre-trained on large diverse datasets and can be used as a base model for all correpondence matching problems regardless of contexts.

Appendix A

Literature Review

A.1 Traditional SfM

Structure-from-Motion (SfM) is one of the most long-standing and well-studied problems in stereo vision. Its goal is to reconstruct 3D structure, that is, the three-dimensional shape of objects in the scene, based on video sequences or image collections that reflect projections of the same structure from different viewpoints [168]. During this process, the camera motion, which can be considered as the coordinate transformation between camera viewpoints, is also estimated when formalizing the geometric relationships between images.

Traditional SfM methods are complicated engineering systems that contain a pipeline of multiple steps including feature extraction [208], correspondence matching [209], geometric verification [210], and incremental reconstruction [169]. The incremental reconstruction step combines multiple two-frame geometries to synthesize approximate global structure using image registration and triangulation [211], which is followed by a bundle adjustment algorithm that mitigates accumulated errors and performs global refinement [212].

Though well-studied, traditional SfM methods are hard to use directly in our problem setting of 3D scene geometry estimation because of the following key differences.

- **Input data requirements**: SfM requires that the nearby images have relatively large camera baselines (far away enough) so that the fundamental matrices can be estimated reliably [108], whereas the input of 3D scene geometry estimation is usually consecutive frames that are captured from really close distances. Also, the inputs of SfM are usually large unordered image collections that have up to dozens or hundreds of images. In contrast, the input of 3D scene geometry estimation is typically a short image sequence with 2-5 sequential frames.

- **Output emphasis**: SfM methods are mainly focused on reconstructing accurate global 3D structure while creating motion estimates as a by-product, while in our setting the structure and motion are both valued parts of the 3D scene geometry as they both help to explain how the video is generated.
- **Output representations**: SfM output structures in the form of 3D point coordinates in a unified world reference system, whereas we use the depth map of each frame to represent structure. Point clouds are sparse and in world coordinates, while depth maps are dense and in camera coordinates. The motion component in SfM is usually represented by the camera extrinsics (the relative rotation matrix and translation vector between cameras) or the fundamental matrices (used to describe stereo geometric constraints) for the uncalibrated settings; in our problem, the camera motion is represented by a 6-degree-of-freedom rigid transformation ($SE(3)$), while some methods also consider independent object motion to model the real motion more precisely [173].

A.2 Depth and Disparity

Depth is typically computed by geometric triangulation on pairs of images or from learning the association between individual images and depth from large numbers of examples, in either supervised or unsupervised fashion. These methods are reviewed next.

A.2.1 Stereo (Binocular) Depth/Disparity

The stereo depth estimation problem has been extensively studied in traditional computer vision, where the stereo correspondences among two or more frames of the same scene have been used to infer the depth of each point.

In the most general setting, the relative positions between multi-view cameras also need to be estimated at the same time, making it essentially an SfM problem that we mentioned before. However, using a calibrated left/right camera rig can greatly simplify this problem, which is thus a common setting in real applications.

Specifically, two cameras are placed side to side on a horizontal support and have the same viewing direction. The camera intrinsic and extrinsic parameters are known through calibration. As a result, since the same world point appears on the same horizontal line in both the left and right frame, the stereo matching problem is reduced to a 1D search problem in the image frame. This problem is called disparity estimation, where the disparity of a point is the difference of x coordinates between its projections in the left and right frame.

The estimated disparity d is then used to compute depth D, that is, the distance of each world point along the optical axis of the camera, as $D = fT/d$ where f is the focal distance and T is the baseline (inter-camera distance). Focal distance and disparity are usually in pixels, and baseline and depth are usually in meters or millimeters. Therefore, given the camera calibration parameters, estimating disparity is equivalent to determining depth.

Traditional stereo depth estimation methods are commonly composed of four steps: matching cost computation, cost aggregation, disparity computation or optimization, and disparity refinement [213]. Specifically, a matching cost is first defined to represent the likelihood that two pixels are a match, and then the matching costs between different pixel pairs are aggregated before optimizing and refining the disparity. Some recent work has also estimated disparity using an encoder-decoder convolutional neural network [90].

A.2.2 Supervised Monocular Depth

Monocular depth estimation has also been studied in computer vision, where only one image of the scene is given as input to the system. This topic is attractive because stereo image pairs may not be available as input in some applications. Also, the stereo depth estimation suffers from data noise such as occlusions, texture-less areas, and appearance changes (non-Lambertian surfaces) when finding the point correspondences; however, this is not a problem for monocular estimation since the correspondences between frames are not needed.

Unlike stereo depth estimation, where the depth is resolved using stereo geometry, the monocular depth is usually inferred based on structural prior and semantic information, which can be learned by the deep neural networks. As a result, the method works well as long as it is deployed on images that conform to the statistics of the image-depth association patterns in the training set. When used on other images, on the other hand, depth estimates can be inaccurate or entirely wrong. The first supervised depth estimation network was proposed by Eigen *et al.* [214], where a multi-scale network and a scale-invariant loss were used to train the network. Many other record-breaking deep network architectures have also been proposed [215, 216], but all of these networks require ground-truth annotations, which are sometimes difficult to acquire.

A.2.3 Unsupervised Monocular Depth

Unsupervised monocular depth estimation, which was also called self-supervised monocular depth in some literature [217], aims at training a depth network without ground-truth supervision.

Typically, unsupervised methods train a monocular depth network by attempting to synthesize a given image from other nearby views. This view synthesis requires an estimate of the depth of the scene. If this estimate is accurate, the synthesized view is ideally the same as the given image. The difference between the synthetic view and the image can then be used to define a reconstruction loss on the depth estimate. This loss is then used as supervisory signal to train the network. Noted that the nearby image views are only needed while computing the training loss and are not needed at inference time.

Two main choices for the nearby images mentioned above are (a) stereo image pairs with known calibration and (b) monocular image sequences (videos) from unknown positions. View synthesis based on stereo image pairs was first explored in many network architectures using both discretized depth [218] and continuous disparity [219]. Based on these networks, Godard *et al.* has further improved depth performance using a left-right consistency loss that enforces the consistency between left and right frame predictions [187].

View synthesis based on monocular videos has also been investigated, where consecutive frames (usually two to five frames) are used as the nearby images to perform the reconstruction. This setting is more broadly applicable because a synchronized stereo pair is no longer needed. However, the problem is more challenging because the motion between frames also needs to be estimated concurrently. As the first work under this setting, Zhou *et al.* combined a depth net and a pose net to estimate both monocular depth and camera motion at the same time [174]. A concurrent work also used a similar structure but with object masks in the network to tackle independently moving objects, whose image motion is not consistent with that induced by the motion of the camera [173]. Other constraints such as depth-normal consistency [220] and edge consistency [221] have also been explored to improve the results.

Though view synthesis based on monocular videos has achieved some success, one common problem of those methods is scale ambiguity. The scale of the scene generally cannot be inferred from images alone: we can always scale the depth map and the translation between cameras by any common, non-zero factor without changing the images. This is not a problem for methods based on calibrated stereo image pairs because the scale is inferred from the known distance between cameras. However, using monocular videos generally suffers from this problem since the camera motion is unknown and needs to be estimated as well. As a consequence, the depth outputs of these methods are usually only accurate up to some scale. Therefore, when evaluating the depth results, these methods need to first normalize the output to the correct scale using the ground-truth median.

A.3 Optical Flow

As one of the most important and well-studied tasks in video motion analysis, optical flow prediction has been receiving attention in the research community for decades. In the early ages, many methods were proposed to solve optical flow prediction alone, while in recent years, much more progress has been made by combining related tasks or vision features, including occlusions, motion boundaries, and depth, when predicting optical flow.

In this section, we will briefly review some (single-task) optical flow prediction methods. Based on the differences in problem settings, the methods are clustered into three groups: variational tracking, supervised learning, and unsupervised learning.

A.3.1 Variational Flow

Variational methods for computing optical flow were first proposed in the 1980s. These methods tried to solve the optical flow prediction problem by designing algorithms by hand to find displacement vectors that match similar patches in the two frames. These algorithms were based on modeling the geometry of motion by hand rather than based on data.

Specifically, in 1981, Lucas and Kanade developed a method for tracking a sparse set of small image windows, under the assumption that the motion within each window is constant in space [11]. The authors formalized tracking as a nonlinear optimization problem and solved it using Newton-Raphson iterations. In the same year, Horn and Schunk [12] proposed solving the optical flow estimation problem at every pixel by minimizing a functional and solving the corresponding Euler-Lagrange partial differential equations. Their model defines the objective function as a combination of two loss terms: The photometric loss computed from two input images $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ is $\|g(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x}))-f(\boldsymbol{x})\|^2$, where $\boldsymbol{u}(\boldsymbol{x})$ is the unknown motion field. This term penalizes changes of appearance for the same point before and after motion. The smoothness loss $\left\|\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}^T}\right\|^2$ is the squared Frobenius norm of the spatial Jacobian of the motion field $\boldsymbol{u}(\boldsymbol{x})$. This combination of loss terms has been adopted in most subsequent work.

Extensive research has been conducted in the years since to improve the results of these variational methods. For example, image pyramids allow estimating both pixel trajectories [222] and optical flow [14] coarse-to-fine, in order to accommodate displacements of widely varying magnitudes. The condition number of a Gramian of the image intensity gradient in each window of the first frame was also used as a way to measure the extent to which that window can be tracked reliably [223]. Recently, in 2015, an optical flow model was introduced that is aware of both motion boundaries and occlusions. This *EpicFlow* (Edge-Preserving Interpolation of Correspondences Flow) [15] model uses a sparse-to-dense interpolation scheme and applies variational energy minimization to estimate optical flow. EpicFlow was the state-of-the-art method until recently.

A.3.2 Supervised Learning

Inspired by the successes of deep Convolutional Neural Network (CNN) models for image classification and object recognition [18], much recent work has modeled the optical flow prediction problem under the supervised learning framework. Under this setting, deep neural networks are used to learn the mapping from image pairs to the corresponding optical flow by minimizing the supervised loss, namely, some distance measure between computed and true flow. This supervised learning setting is strongly data-driven, in that the geometry of motion is not modeled explicitly but learned through data sets of examples, thanks to the strong expressive power of CNNs.

The first major paper to do so introduced the FlowNet [19] architecture in 2015. This network uses a multi-scale encoder-decoder structure with skip connections between same-scale layers. The encoder part is designed to extract features of different scales (patch size) and levels (low-level shape features and high-level semantic features) and the decoder part predicts flow by refinement. The network was trained end-to-end with a synthetic data set called Flying Chairs, introduced in the same paper.

Most of the subsequent models followed the same general framework of FlowNet and only changed the specific network structure to improve prediction accuracy. Specifically, two significantly different directions were explored in the literature:

1. To make the network structure more complex in order to increase the learning

capacity of the model, so as to enable it to capture more complicated motion patterns. The data set used to train the model is also expanded and refined at the same time to prevent overfitting.

2. To simplify the network structure by introducing network modules based on domain-specific knowledge (traditional computer vision heuristics) to make the network more flow-specific.

An example of the first direction mentioned above is FlowNet2 [224], which was introduced as a follow-up system to FlowNet by the same research group in 2017. This new version implemented several enhancements in both network architecture and training set for better estimation quality. Specifically, the network was made more complex by (a) stacking multiple FlowNet models, and (b) adding a fusion network to combine multiple branches of stacked FlowNets. The data set was expanded by aggregating two data sets, Flying Chairs [19] and Flying Things 3D [90]. Some empirical tricks were also used, including data set ordering and network module preferences. As a result of all these changes, the model decreased the error by half from FlowNet but the inference was marginally slow due to the large size of the model. Because of this, few follow-up approaches adopted this complex structure.

In contrast, the second direction strove to simplify networks to decrease both the model size and the inference time. As simplifying the network architecture may hurt the prediction accuracy, field-specific knowledge was incorporated into the design of network modules. This knowledge amounted mainly to traditional ideas or heuristics in computer vision, including the use of image or feature pyramids and cost volumes. These ideas achieved significant progress in both prediction quality and speed.

For example, SPyNet (Spatial Pyramid Network) [35] introduced image pyramids explicitly in the network. An image pyramid can decompose large motions as a combination of small displacements at different scales, and the prediction becomes a coarse-to-fine process in multiple steps. The image pyramid substitutes the encoder part of the network, and the decoder is more carefully structured. As a result, the model is significantly smaller while still being effective.

Following the spirit of SPyNet, PWC-Net (Pyramid, Warping, Cost Volume Network) [20] also added field-specific knowledge in its network design. They first changed the pyramidal structure from image pyramids to feature pyramids by adding a frame encoder (an encoder that takes each frame as input, not the frame pair) so that image-level features can be extracted. They also adopted warping and cost volume computation, a feature matching score that was originally proposed in a traditional method called DCFlow[225]. Similar ideas were also explored in LiteFlowNet [36]. Later, an iterative residual refinement version of the network, IRR-PWC [21], was proposed to make the model size even smaller, while keeping the accuracy on par. With the help of these traditional heuristics, the PWC-Net managed to achieve a win-win in the accuracy/model size trade-off and is, therefore, the most adopted backbone model in subsequent work.

The current state-of-the-art supervised flow model is RAFT [22], which maintains an all-pairs correlation volume instead of the local cost volumes. They keep the features at a high resolution (only 8 times smaller than original) and compute a full-field correlation volume (four dimensions) between every two pixels. The multi-scale correlations are generated by an average pooling on the full-field correlation, and they only average-pool on the last two dimensions to keep the features at high resolutions. Since the features are always high-resolution, the decoder is a recurrent network that refines the optical estimation recurrently, which mimics an optimization process, instead of the regular coarse-to-fine estimation networks. This method has improved the performances with a large margin on all evaluation data sets.

A.3.3 Unsupervised Learning

One of the main obstacles to the supervised learning of motion is the difficulty in obtaining a large number of ground-truth labels. Since manual labeling is almost impossible for ground-truth dense optical flow, given both the huge amount of pixels in a video sequence and the difficulty in knowing true flow in the first place, many methods have used synthetic data in training. It is still an open question whether synthetic data are an adequate proxy for real data. Accordingly, recent research has been focused on the unsupervised learning of video motion as a compromise between labeled data availability and model performance.

Basic models Unsupervised learning of optical flow minimizes at training time more or less the same loss functions that variational methods minimize at inference time, namely, the photometric loss. The first piece of work that explored this technique was DSTFlow (Dense Spatial Transform Flow) [38] published in 2017, which used a similar framework as did STN [226]. They first adopted a trained FlowNet [19] model as the localization layer, and then they performed backward warping and measured the discrepancy of appearance between the warped image and the real image. This discrepancy was then used as the loss term in gradient backpropagation. This work started a new path of research, the unsupervised learning of video motion.

Similar frameworks have been adopted by much subsequent work on unsupervised prediction. The key problem for unsupervised prediction is: how can we “tell” the network about what optical flow looks like? With the absence of ground-truth labels, the only way to do so is to define constraints that optical flow should satisfy, and train the network to satisfy those constraints as much as possible. Therefore, a very natural way to improve the results is by adding more accurate and powerful constraints. Examples of this include UnFlow [27], which used a bi-directional structure to implement forward-backward constraints, and MultiFrameFlow [39], which explored the three-frame analysis of one past, one present, and one future frame. Many of these methods implicitly include occlusion estimation by the forward-backward consistency check, as we discuss in greater detail in the next section.

A Typical Problem: Occlusions The optical flow of occluded pixels cannot be measured, because they have no correspondence in the second frame. If a flow value is assigned to these pixels nonetheless, then the photometric loss tends to be large because the source pixel is in the background and the destination pixel is often in the foreground. These large loss values tend in turn to skew flow estimates around occlusion regions.

Occlusion is especially a problem for unsupervised training because the photometric cue is the main source of information that evaluates whether the current estimation is plausible, whereas in supervised training the ground-truth supervision can push the network to be robust to occlusions.

The problem with occlusions in the unsupervised learning of flow was first revealed by Wang *et al.* in their work OccAwareFlow [40] in 2018. They showed that the backward warping process, which is unavoidable in almost all unsupervised methods, may create incorrect warped patterns specifically at the occlusion regions, and the model will tend to discredit the correct flow vector, as the local pattern does not match. Therefore, they estimated the occlusion by conducting a forward warping of the predicted backward flow and used that information to reweigh different pixels in the loss function. This may not be a perfect solution, but noticing the occlusion problem is itself an important contribution.

Two interesting ideas introduced in recent literature to address occlusion-related problems are discussed next. These methods are called occlusion-aware optical flow estimations.

Occlusion-Aware Through Bi-directional Flow Estimation The optical flow was estimated simultaneously in both temporal directions in the UnFlow architecture [27], with two important benefits. First, if frame pair (f, g) is used for training, then pair (g, f) can be used as well, thereby effectively doubling the size of the training set. Second, the bidirectional loss makes the forward-backward check possible, and occlusions can be estimated through it. UnFlow then modifies the loss function by masking out occluded regions in the computation of the photometric loss, and by adding explicit regularization terms that discourage large numbers of occluded pixels.

Since then, the forward-backward consistency check has been used in many occlusion-aware methods to generate occlusion estimates.

A different way to use flows in the two temporal directions was proposed in the MultiFrameFlow system [39], which uses three frames (past, present, and future) as input. Backward flow and occlusions are computed from the present frame to the past frame, and forward flow and occlusions are computed from the present frame to the future frame. Both computations use off-the-shelf predictors from the literature, trained in an unsupervised manner. The case in which a point is visible in the present but occluded in both past and future (a transient, one-frame occlusion) is ruled out as unlikely. Furthermore, velocity is assumed to be constant over the three-frame temporal window at every pixel. This formulation allows including additional occlusion and flow priors to the loss function, to reflect the continuity of occlusion and optical flow both in space and in time.

This shows that using multiple frames can produce more constraints based on flow and occlusion information, leading to a possible new research direction for unsupervised optical flow estimation.

Occlusion-Aware Through Data Distillation (Occlusion Hallucination) The DDFlow system [96] incorporates an idea that, for reasons left unclear, is called *data* *distillation*. Even without manual annotation, it is easy to produce artificial occlusions with known flow even for occluded pixels: First, let a *teacher network* estimate flow for all unoccluded pixels in a training image pair. Then, feed cropped versions of the same images in the pair to a *student network*. Pixels that are within the cropped area in the first frame but fall outside it in the second are occluded from the point of view of the student network. However, the teacher network can provide supervision to the student by telling (i) where the artificial occlusions occur and (ii) what the correct flow is at those locations. The student network can therefore use this supervision to learn to both detect occlusions and hallucinate the flow for occluded pixels. At test time, only the student network is used for prediction.

This distillation/hallucination approach is entirely data-driven, in that the student learns to detect and estimate flow for occluded pixels from supervision, rather than through hand-crafted terms added to the loss function. SelFlow [28], an updated version of DDFlow, achieved state-of-the-art results with a clear margin in 2019.

Some recent unsupervised networks have also adopted this type of architecture with two network copies ("student and teacher"), including ARFlow [1], which applies a random transformation to the teacher's prediction and use it as augmented pseudo-labels in the student network. CoT-AMFlow [227] has also attempted to aggregate two copies of the same unsupervised network structure in training.

A.4 Motion Boundaries and Segmentation

Optical flow estimation methods typically address the aperture problem by including a term in the training risk that penalizes deviations from flow smoothness. However, flow is typically only piecewise smooth, and the smoothness penalty must therefore be turned off along motion boundaries. These are image curves across which the optical flow is discontinuous.

In addition, it is also often useful to identify objects in the scene that move by different motions, a problem that is called *motion segmentation* in the literature. This Section surveys approaches for estimating motion boundaries and segment different options apart.

A.4.1 Motion Boundaries

Traditional methods compute motion boundaries based on statistical tests and aggregations of some candidate curves or segments found by filtering hand-crafted features [228, 229]. Weinzaepfel *et al.* [230] feed optical flow estimates and warping errors to a structural random forest [231] to learn motion boundaries in a supervised manner. Some work has also applied deep learning networks to the estimation of motion boundaries [232].

A.4.2 Motion Segmentation

The goal of motion segmentation is to partition an image into object that move differently from each other. One way to make the definition of this problem rigorous is through the notion of *rigid motion*. A set of points in three dimensions is said to be moving rigidly if the distances between any two points in the set is left unchanged. The motion of the set can then be described by a coordinate transformation, which has six degrees of freedom. A clean way to define motion segmentation is then as the problem of computing the coarsest partition of the image into (the projections of) rigidly moving point sets.

When objects deform as they move (that is, when the distances between points on the object change over time), this definition no longer applies. Image motion is often discontinuous across curves, the motion boundaries, but these boundaries are not necessarily closed. Thus, rigid-motion segmentation and motion boundary detection are distinct problems. When they are confused, results reported by different researchers can be subjective and inconsistent with each other.

Early motion segmentation papers assume that all objects belong to a number of layers with different depths, similarly to what is done in animated movies [233, 234]. Under this assumption, the motion within the frame can be decomposed into independent motions within each depth layer, and the motion segmentation problem then becomes the problem of assigning pixels to depth layers. Some methods assume that one rigid motion, or perhaps the motioning one layer, dominates the image, in the sense that a majority of pixels belong to it. This is often the case when the camera moves. If the scene is in large part static, then the relative motion between the camera and the static part of the scene is rigid. One can then determine the dominant motion by some robust clustering method, remove the pixels in this motion from consideration, and repeat [235, 236].

While the assumption of the existence of a dominant motion is often warranted, it is not clear that all the motions in the image can always be sorted into a sequence of dominant motions. At any rate, the dominant-motion approach (with a single dominant motion) is often relevant in traffic scenes when the camera itself is mounted on a vehicle: In that case, much of the motion in the image is frequently generated by camera motion alone, and is therefore both rigid and dominant. Additional motions often come from other vehicles that move in the scene, and these motions, while not dominant, are still rigid. Pedestrians are more difficult to incorporate into this framework, as their bodies often deform while they move. A detailed treatment of a walking person would be to decompose it into rigidly moving limbs, but this treatment is both computationally expensive and difficult to implement when the person is small in the field of view. A simpler approach is to approximate the whole person as a single, rigidly moving object, thereby ignoring fine motions.

Motion estimation and the computation of optical flow go hand in hand. If the image is segmented into the dominant motion induced by motion of the camera and a set of additional motions for independently moving objects, rigid or not, then the flow in the dominant component can be regularized by making it consistent with the six degrees of freedom of a single rigid motion. Specifically, the two components of flow at every pixel can be written as functions of a single depth value, unique to that pixel, and six additional degrees of freedom that are shared among all of the pixels of the dominant component. The flow for the remaining motions is then estimated by standard flow estimation methods, that is, by computing correspondences between frames. Many scene geometry estimation methods already include this camera motion segmentation called "explainability mask" [187, 161, 75].

A.5 Combining Depth, Motion, and Optical Flow

3D structure, optical flow, and camera motion are strongly related, because for static parts of the scene any one of these three quantities can be computed from the other two. Methods that estimate these quantities jointly can therefore take advantage of the constraints that these relationships entail. This observation has been exploited in both traditional methods and methods based on deep learning, as discussed next.

A.5.1 Traditional Methods

Classical methods estimate the camera motion between two frames by statistical estimation and optimization [237]. By using stereo videos, more accurate motion estimates can be established. For example, the stereo and motion correspondences can be computed simultaneously based on their consistency in the scene structure [238]. Apart from camera motion, the motion of an individual object and surface can also be estimated [239, 240]. The estimates can also be improved with analysis based on multiple frames [241].

Traditional methods usually combine SfM and optical flow by placing them in a unified 3D geometry model and derive the mathematical relationships between these terms. Much work has tried to keep explicitly the epipolar geometry in the model and solve the optical flow along the epipolar lines [242, 243], which is a 1D searching problem. MR-Flow [108] adopted the Plane+Parallax approach [193], which first eliminates the rotation and scaling components in the motion using homography transformations (planar motions) and then solves for the residual flow (parallax flow). The MR-Flow model achieved state-of-the-art result on the MPI-Sintel clean pass until deep learning-based methods were proposed.

To better model the independent flow on the moving object regions, there has also been a trend to add a segmentation step and to treat the independent object part separately [244, 108].

A.5.2 Deep Learning Methods

One advantage of applying deep neural networks is that the tasks can be easily combined by stacking network layers step-by-step or aggregating separate sub-networks to minimize a joint loss function. End-to-end training has also made training multiple tasks at the same time neat and simple.

Most of the networks that estimate depth, motion, and optical flow jointly are trained in an unsupervised way. This is by part because a data set with full annotations of all three tasks is hard to acquire. In addition, it is these cross-task relationships that have made the unsupervised joint training possible; if we can do supervised training, the separate single-task supervised models may just be good enough without these cross-task relationships.

The first method that utilized depth and motion predictions to refine optical flow was a stacked network model called GeoNet [76]. It first predicts depth and camera motion using two separate networks and then uses those predictions to reconstruct the camera motion flow, which is then fed to a flow residual net to refine a final optical flow prediction. Following the structure in the SfMLearner [174], the authors first use the unsupervised flow loss of the reconstructed camera motion flow to train the depth net and the camera motion net jointly. Subsequently, they freeze the depth and camera motion net and train the flow residual network using the unsupervised flow loss of the final optical flow. This stacked network uses the reconstructed camera motion flow as a good initial flow estimate, which makes the training of flow easier, but the training of the flow does not help the depth and motion net in return since they are frozen at that stage.

To enforce cross-task constraints more strongly, the DF-Net [161] was proposed, which added a separate optical flow network and defined a cross-task consistency loss to help the joint training. Since the reconstructed camera motion flow, which is computed using the depth and camera motion estimation, should be consistent with the 2D matching-based optical flow estimation, the cross-task consistency loss is defined as the difference between these two flows. Note that these two flows are only consistent at the camera motion region, which can be estimated using a forward-backward consistency check. This architecture treats depth, camera motion, and optical flow with three sub-networks in parallel. This can determine the optical flow from two sources of information: 3D scene geometry and 2D matching. Combining these two sources help make optical flow predictions more robust.

Since the camera motion flow and the estimated flow based on 2D matching are only consistent at camera motion regions, it is natural to use the camera motion flow only at camera motion regions, while switching to the 2D matching flow for the other independent moving regions. This requires the segmentation of camera motion and independent object motion. The CC (Competitive Collaboration) network [75] has attempted to add a separate mask net to estimate such segmentation (the so-called ”explainability mask”). The downside of this method is that training four parallel sub-networks is very complicated. The training schedule alternates between two stages, a competition stage and a collaboration stage. During the competition stage, the depth and camera motion nets are trained using the reconstructed flow loss; the flow net is also trained subsequently using the flow loss. In the collaboration stage, the mask net is trained to refine the estimate of the part where the reconstructed camera motion flow is consistent with the estimated flow. The training process iterates between these two stages until a validation criterion is reached. This joint network models more precisely how the optical flows of different regions are generated, but both model and training schedule are quite complex.

Following the methods above, many new models have been explored for better performances. The EPC++ model [162] added an epipolar constraint loss and explored adding stereo left-right pairs to train the joint network, which has alleviated the motion confusion of the moving regions, where the object stays in the same position within the frame between consecutive images. As another potential improvement, GLNet [245] attempted to implement online optimization, where they use the test samples to train the network for one more step or optimize the output using the unsupervised loss. This has achieved a large boost in evaluation but is one order of magnitude slower at inference time.

Following these previous literature, we will also focus on the unsupervised joint training of depth, motion, and optical flow. We will use similar sub-network architectures and task relationships, but the difference is that we will try to build the whole system with stereo video input. We will explore using stereo depth computed by the disparity (also similar to optical flow) to see whether it can benefit the joint training process.

A.6 Combining Semantics and Optical Flow

Though there have been many recent improvements on either semantic segmentation or optical flow estimation, the problem of semantic optical flow (how to exploit semantic information to help optical flow training) has received very limited attention, and the current best results are thus much out-of-dated. Some early methods incorporate semantics by refining specific semantic regions, such as the planar regions (using homographies) [105], independently moving objects (using affine motions) [105], and static scene (using camera motion and epipolar constraints) [106]. There are also methods that take advantage of instance-level semantics and compute rigid motion for each object instance [107]. However, most methods are traditional flow methods based on energy minimization, and they all need some initial flow before refinement, which means the semantic part is in some sense a post-processing of existing methods. In comparison, we explore adapting the most up-to-dated unsupervised optical flow networks, which are trained from scratch with semantic information.

One work that applies semantics to optical flow networks is [109], which jointly trains video segmentation and optical flow. Nevertheless, we are still very different in that (1) their motivation is to use the learned flow to help the feature/label propagation part of video segmentation, so their emphasis is on segmentation, (2) we treat semantic segmentation and semantic-aided optical flow as two separate modules so that both problems can be improved independently, and we show significantly better results than them.

Some other works also explored how optical flow can in return help semantic segmentation [128, 129]. There are also studies on how to exploit semantics to stereo matching [130, 131], and to 3D scene flow estimation given extra depth cues such as stereo camera inputs [132, 133] and point clouds [134].

One typical work that started to combine semantics and optical flow was Semantic Optical Flow (SOF) [105], which utilizes semantic information for flow estimation with a layered representation. They divide all the semantic categories into three big categories, namely Things, Plane, and Stuff. Category Things refer to objects that can move independently, either rigid or non-rigid. The category includes bicycle, bird, car, etc. Their motion is assumed to be affine plus a smooth deformation from affine. Category Plane refers to regions that have broad spatial extent like roads. Particularly, water and sky are considered as planes. Their motion is modeled by homographies. Category Stuff corresponds to classes that have complicated 3D textures such as buildings and vegetation. Regions of unknown class are also modeled as Stuff. The motion of these objects are hard to model, and thus, they use a classical spatially varying dense optical flow to model their motion.

Subsequently, the semantic segmentation provides object boundaries. They avoid using the initial flow estimates near these object boundaries to compute the affine motion in both the foreground and background. The identities of the segmented objects are constant over time, and they use this consistency to encourage temporal consistency for optical flow estimation. For each object belonging to the Things category, they find its corresponding regions in the subsequent frames. These corresponding regions set the spatial extent of the flow estimation. They use the spatial relationship of the segmented objects to infer depth ordering. They always assume Things are foreground objects.

Another concurrent work was SDF [107]. They assume that there are only background and rigid moving foreground objects. For each of objects, they estimate its motion by estimating the fundamental matrices. They use the instance-level segmentation to detect objects since different objects, connected due to occlusion, of the same semantic category may not have the same motion. The proposed CNN-based flow estimator also predicts uncertainties of the flow estimates, which are used in rejecting outliers in computing the fundamental metrics. They model the optical flow estimation as a classification problem. The model predicts the probability of the location of the matching point in the second frame, and is trained with ground-truth flow. The number of moving objects are set beforehand, that is, it is assumed that only a certain number of objects can move.

Apart from semantic segmentation, some work has also investigated using optical flow to improve video segmentation [109]. One challenge for video segmentation is that only one frame of the video clip is labeled, so most methods require some feature/label propagation based on pre-computed optical flow. Since optical flow can be learned in a self-supervised way, this paper propose to jointly train video segmentation and optical flow at the same time. They use one shared feature encoder and two separate decoders. The two decoders are interwoven so that (1) optical flow is used to check semantic feature consistency for the semantic task; (2) use semantic mask to estimate occlusion and refine the learned occlusion mask for the flow task.

Appendix B

Math Derivations for Stereo Vision

B.1 Camera Calibration

One key step when we deal with 3D videos is finding the projection relationship from 3D world coordinates to the image coordinates, *i. e.*, locating where a 3D world point is shown on the image frame and vice versa. This process requires the camera parameters saved in the calibration file.

There are generally three coordinate systems involved, the image coordinates, the camera coordinates, and the world coordinates. The image coordinates η, ξ refer to the row and column values of the point in pixels. The camera coordinates have the origin at the center of projection with its X, Y, and Z axis pointing right, downward, and forward, respectively. The world coordinates, on the other hand, can be any 3D coordinates in the world, which is usually a unified coordinate system in multi-camera settings. Camera and world coordinates are typically in millimeters or meters.

The transformation between camera coordinates and image coordinates depends on the internal parameters of the camera, which are called the camera intrinsics. On the other hand, the transformation between world coordinates and camera coordinates depends on the relative position and orientation of the two coordinates, which are called the camera extrinsics. When we project a 3D world point to its corresponding image coordinates, we first need to transform the world coordinates $(X^{(w)}, Y^{(w)}, Z^{(w)})^T$ to the camera coordinates $(X^{(c)}, Y^{(c)}, Z^{(c)})^T$ using the extrinsics and then transform from $(X^{(c)}, Y^{(c)}, Z^{(c)})^T$ to its image coordinates $(x, y)^T$ using the intrinsics.

Camera extrinsics The camera extrinsics involve the relative rotation R (a 3×3 matrix) and translation $\boldsymbol{t}$ (a 3-dim vector) between the world coordinates and the camera coordinates, so we have

$$\begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix} = R \begin{bmatrix} X^{(w)} \\ Y^{(w)} \\ Z^{(w)} \end{bmatrix} + \boldsymbol{t}, \text{ and in reverse } \begin{bmatrix} X^{(w)} \\ Y^{(w)} \\ Z^{(w)} \end{bmatrix} = R^T \left(\begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix} - \boldsymbol{t} \right).$$

We sometimes add a redundant dimension of 1 to the 3D coordinates and make it the 4D homogeneous world/camera coordinates to simplify the equations

$$\begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix} = [R \mid \boldsymbol{t}] \begin{bmatrix} X^{(w)} \\ Y^{(w)} \\ Z^{(w)} \\ 1 \end{bmatrix}, \text{ and in reverse } \begin{bmatrix} X^{(w)} \\ Y^{(w)} \\ Z^{(w)} \end{bmatrix} = \left[R^T \mid -R^T\boldsymbol{t}\right] \begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \\ 1 \end{bmatrix},$$

where $E = [R \mid \boldsymbol{t}]$ is called the camera extrinsics matrix. Note that the extrinsics E is a 3×4 matrix.

Camera Intrinsics When we project the point from its camera coordinates to its image coordinates, the required internal camera parameters are included in the camera intrinsics matrix with the following form (assuming no skewness, that is, that the sensor plane is exactly perpendicular to the optical axis)

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where f_x and f_y are the focal distances in pixels, and c_x and c_y are the image coordinates of the principal point, defined as the intersection of the optical axis with the image plane. There are two focal distance parameters f_x and f_y because the physical camera pixel may not be squares. Instead, if the pixels are rectangular, the lengths of the pixel in the x and y dimension are different, so when we measure the same focal distance in pixels, the result will be different in the x and y dimension, which gives us f_x and f_y, respectively.

The projection from the camera coordinates to the image coordinates can be computed from similar triangles in geometry. In addition, we need to scale the unit of the world coordinates from meters to pixels and also shift the origin to the top-left corner of the image, which can be done by simply applying the camera intrinsics matrix K.

$$\begin{bmatrix} X^{(h)} \\ Y^{(h)} \\ Z^{(h)} \end{bmatrix} = K \begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix} = \begin{bmatrix} f_x X^{(c)} + c_x Z^{(c)} \\ f_y Y^{(c)} + c_y Z^{(c)} \\ Z^{(c)} \end{bmatrix}$$

$$\begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix} = K^{-1} \begin{bmatrix} X^{(h)} \\ Y^{(h)} \\ Z^{(h)} \end{bmatrix} = \begin{bmatrix} (X^{(h)} - c_x Z^{(h)})/f_x \\ (Y^{(h)} - c_y Z^{(h)})/f_y \\ Z^{(h)} \end{bmatrix},$$

where $(X^{(h)}, Y^{(h)}, Z^{(h)})^T$ is the vector of homogeneous image coordinates. To obtain the 2D image coordinates, we still need to divide by the last dimension to project it on the image plane

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X^{(h)}/Z^{(h)} \\ Y^{(h)}/Z^{(h)} \end{bmatrix} = \begin{bmatrix} f_x \frac{X^{(c)}}{Z^{(c)}} + c_x \\ f_y \frac{Y^{(c)}}{Z^{(c)}} + c_y \end{bmatrix}.$$

Note that this projection is not uniquely invertible because any 2D point in the image actually corresponds to all the 3D world points on the same projection ray, and the true correspondence depends on the depth of the world point. If we know the actual depth of this point $Z^{(c)}$ (Note $Z^{(h)} = Z^{(c)}$ by definition), we can project the image point to its homogeneous image coordinates by

$$\begin{bmatrix} X^{(h)} \\ Y^{(h)} \\ Z^{(h)} \end{bmatrix} = \begin{bmatrix} xZ^{(c)} \\ yZ^{(c)} \\ Z^{(c)} \end{bmatrix}.$$

The two steps mentioned above are commutative, *i. e.*, we can also first divide by the last dimension and then apply the camera intrinsics K to get the same result.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z^{(c)}} \begin{bmatrix} X^{(h)} \\ Y^{(h)} \\ Z^{(h)} \end{bmatrix} = \frac{1}{Z^{(c)}} K \begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix} = K \frac{1}{Z^{(c)}} \begin{bmatrix} X^{(c)} \\ Y^{(c)} \\ Z^{(c)} \end{bmatrix}$$

B.2 Stereo Depth from Disparity

We assume that the left and right cameras are identical (same focal distance, image plane size, etc.) and are aligned horizontally with the same orientation (the 3D displacement vector between the two cameras is parallel to the x-axes of both cameras). The camera intrinsics and the displacement of the two cameras can be found in the calibration file.

Suppose Ω denotes the set of all pixel coordinates in the frame: $\Omega = \{(x, y)^T \mid 0 \le x < w, 0 \le y < h; x, y \in \mathbb{Z}\}$. Given the left and right camera view of the same scene $I^{(l)}, I^{(r)} \in \mathbb{R}^{h \times w \times c}$, we can compute the disparity for each point in the left view (*i.e.,* finding point correspondences from left to right) $d : \Omega \to \mathbb{R}$, such that under the photometric constancy assumption and assuming no occlusions we have $\forall (x, y)^T \in \Omega$ (x and y being the column and row index, different from the matrix index order)

$$I^{(l)}(y, x) = I^{(r)}(y, x + d(x, y)).$$

Ideally, the disparity should be negative in this case since the object always appear on the right in the left view and left in the right view, *i. e.,* $d(x, y) < 0, \forall (x, y)^T \in \Omega$.

Suppose that the displacement between the two cameras is T (a scalar because the displacement is along the x-axis only) and the focal distance in horizontal pixels is f_x (which equals the focal distance multiplied by the number of pixels per millimeter). The depth of each point $(x, y)^T \in \Omega$ can be computed as

$$D(x, y) = \frac{f_x T}{|d(x, y)|}. \tag{B.1}$$

Eq B.1 shows that the depth of the point is proportional to its inverse disparity, where the constant factor is determined by the focal distance and the distance between cameras. This makes sense because the objects closer to the camera usually have larger disparities (larger offsets between their positions in the left and right frames).

Technical Issues The required parameter T can be found in the extrinsics matrix E, and the parameter f_x can be found in the intrinsics K. However, some data set (including the KITTI data set that we are using) does not provide each of them separately but the aggregated projection matrix $P = KE$ instead. Therefore, we need to find T and f_x from P.

As introduced in the last section, the extrinsics matrix E is composed of rotation R and translation $\boldsymbol{t}$ and is used to project a point from world reference coordinates (a unified reference coordinates for all cameras) to camera coordinates. The intrinsics matrix projects from camera coordinates to homogeneous image coordinates, and we can divide all coordinates by the last one to obtain the actual image coordinates. Thus, for a point in the world reference coordinates $(X^{(w)}, Y^{(w)}, Z^{(w)})^T$, the P matrix directly projects it to the homogeneous image coordinates

$$\begin{bmatrix} X^{(h)} \\ Y^{(h)} \\ Z^{(h)} \end{bmatrix} = P \begin{bmatrix} X^{(w)} \\ Y^{(w)} \\ Z^{(w)} \\ 1 \end{bmatrix},$$

and the image coordinates of that point can be found by $(x, y)^T = (x^{(h)}/z^{(h)}, y^{(h)}/z^{(h)})^T$.

Finding the Focal Distance f_x We can estimate f_x by simply taking $P_{1,1}$ (the first value in P). The camera set has four cameras that are placed on the same horizontal line in parallel and are numbered #0, #2, #1, #3 from left to right (#0 and #1 are grey-scale cameras, while #2 and #3 are color cameras). The world reference coordinates is camera #0, which is aligned the same way as the left camera (camera #2) and the right camera (camera #3), and the rotation R should be identity up to some measurement noise. Also, we assume that the translation $\boldsymbol{t} = (t_x, t_y, t_z)^T$ is only along the x-axis, *i. e.* $t_y \approx 0, t_z \approx 0$. Thus, we have

$$P = K \left[I_{3\times3} \mid \begin{matrix} t_x \\ 0 \\ 0 \end{matrix} \right] = \begin{bmatrix} f_x & 0 & c_x & f_x t_x \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Since we have the P matrices for both the left and right camera (which are the same up to measurement noise), we take the average for robustness.

Finding the Camera Distance T Since we only have $P = K[R \mid \boldsymbol{t}]$, where $R = I_{3\times3}$, we have $P = [K \mid K\boldsymbol{t}]$, so we can estimate $\boldsymbol{t}$ for each of the left and right camera by

$$\boldsymbol{t} = P^{-1}_{1:3,1:3} P_{1:3,4},$$

where $P^{-1}_{1:3,1:3}$ denotes the left 3×3 matrix in P and $P_{1:3,4}$ denotes the last column in P. Note that the translation $\boldsymbol{t}$ here means how the coordinates are transformed. If we want to find how the camera origins are represented in the reference coordinates system, we need to reverse the sign as $-\boldsymbol{t}$.

We compute the translation factor $\boldsymbol{t}$ for both the left camera and the right camera and then subtract them to obtain the displacement between the left and right camera.

The displacement is along the x-axis by assumption, so we only take the x-dimension and use it as the T value in Eq B.1.

B.3 Camera Motion Representation

The camera motion can be represented in two forms, a matrix form that represents the linear transformation of coordinates, and a vector form that lists the degrees of freedom.

The matrix form $C = [R|\boldsymbol{t}]$ (a 3×4 matrix) consists of a rotation R and a translation $\boldsymbol{t}$. For a 3D point x in the first frame, we can find its coordinates y in the next frame after applying the camera motion C

$$\boldsymbol{y} = R\boldsymbol{x} + \boldsymbol{t}.$$

The vector form $\boldsymbol{c} = (t_x, t_y, t_z, r_x, r_y, r_z)^T$ contains the six degrees of freedom. t_x, t_y, t_z are the translation components along the x, y, and z-axis, whereas r_x, r_y, r_z are the degrees of rotation (in radian) around the three axes.

These two forms can be transformed to each other by

$$\begin{cases} R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos r_x & -\sin r_x \\ 0 & \sin r_x & \cos r_x \end{bmatrix} \begin{bmatrix} \cos r_y & 0 & \sin r_y \\ 0 & 1 & 0 \\ -\sin r_y & 0 & \cos r_y \end{bmatrix} \begin{bmatrix} \cos r_z & -\sin r_z & 0 \\ \sin r_z & \cos r_z & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ \boldsymbol{t} = (t_x, t_y, t_z)^T, \end{cases}$$

$$\begin{cases} r_x = \arctan\left(\frac{-R_{23}}{R_{33}}\right), \quad r_y = \arctan\left(\frac{R_{13}}{\sqrt{R_{23}^2 + R_{33}^2}}\right), \quad r_z = \arctan\left(\frac{-R_{12}}{R_{11}}\right), \\ (t_x, t_y, t_z)^T = \boldsymbol{t}, \end{cases}$$

where R_{ij} denotes the element on the ith row and jth column in the matrix R.

Camera Motion from Pose Sometimes the data set gives camera pose labels instead of the camera motion labels between consecutive frames. The camera pose matrix for the ith frame is a 3×4 matrix that projects a point from the ith frame camera coordinate system to the first (=0th, reference) frame coordinate system. Thus, for two consecutive frames, we need to compute the camera motion label based on the two camera poses of those frames.

Suppose the camera pose for the first frame is E_1 and the second being E_2, where

$$E_1 = [R_1 \mid \boldsymbol{t}_1], \quad E_2 = [R_2 \mid \boldsymbol{t}_2].$$

We have for the point $\boldsymbol{x} = (x, y, z)^T$ in the first frame coordinates, its reference coordinates is $R_1\boldsymbol{x} + \boldsymbol{t}_1$, and thus the second frame coordinates

$$R_2^{-1}(R_1\boldsymbol{x} + \boldsymbol{t}_1 - \boldsymbol{t}_2) = R_2^{-1}R_1\boldsymbol{x} + R_2^{-1}(\boldsymbol{t}_1 - \boldsymbol{t}_2).$$

Thus, the camera motion matrix is

$$C = [R_2^{-1}R_1 \mid R_2^{-1}(\boldsymbol{t}_1 - \boldsymbol{t}_2)].$$

B.4 Camera Motion Transformation

Given the camera calibration parameters, the camera motion, and the depth map of the first frame, we can find for each pixel in the first frame where that pixel is in the next frame. This involves three steps as follows. The optical flow can be reconstructed by subtracting the first and second frame image coordinates.

1. Use depth to map from the first frame image coordinates to the world camera coordinates (camera calibration)
2. Transform from the first frame world coordinates to the second frame world coordinates (apply camera motion)

3. Project from the second frame world coordinates to the second frame image coordinates (camera calibration)

Specifically, the camera calibration matrix is given in the form of $P = KE$. For each pixel with image coordinates (x_i, y_i), its estimated physical depth is $Z_i^{(c)}$. Now, we have a point cloud of pixels $\{(x_i, y_i, Z_i^{(c)}) \mid i = 0, 1, \cdots w \times h\}$.

Step 1. Fixed time 1; transform from image coordinates to the left camera coordinates (camera calibration):

Since the KITTI data set keeps the left camera coordinates as its world reference system in the calibration process, we can simply complete this step using the left camera calibration. For each point, we first multiply the first two dimensions by $Z_i^{(c)}$ to obtain its homogeneous image coordinates $(x_i Z_i^{(c)}, y_i Z_i^{(c)}, Z_i^{(c)})$ and then apply the inverse left camera calibration transformation P by

$$\begin{bmatrix} X_i^{(w)} \\ Y_i^{(w)} \\ Z_i^{(w)} \end{bmatrix} = {P_{1:3,1:3}}^{-1} \left(\begin{bmatrix} x_i Z_i^{(c)} \\ y_i Z_i^{(c)} \\ Z_i^{(c)} \end{bmatrix} - P_{1:3,4} \right),$$

where $P_{1:3,1:3}$ refers to the left three columns of P, and $P_{1:3,4}$ is the last column of P.

Step 2. Fixed world coordinates, time 1 to time 2 (camera motion):

Now, we apply the computed camera motion matrix C (a 3×4 matrix with a rotation and a translation component) to transform each point in the first-frame coordinates to its location in the next frame.

$$\begin{bmatrix} \hat{X}_i^{(w)} \\ \hat{Y}_i^{(w)} \\ \hat{Z}_i^{(w)} \end{bmatrix} = C \begin{bmatrix} X_i^{(w)} \\ Y_i^{(w)} \\ Z_i^{(w)} \\ 1 \end{bmatrix}$$

Step 3. Fixed time 2; world coordinates to the image coordinates (camera calibration):

This is simply the inverse of step 1. We first compute the homogeneous image coordinates by

$$\begin{bmatrix} \hat{X}_i^{(h)} \\ \hat{Y}_i^{(h)} \\ \hat{Z}_i^{(h)} \end{bmatrix} = P \begin{bmatrix} \hat{X}_i^{(w)} \\ \hat{Y}_i^{(w)} \\ \hat{Z}_i^{(w)} \\ 1 \end{bmatrix}.$$

Then, we project the homogeneous image coordinates to the image plane and obtain the final image coordinates as $(\hat{x}_i, \hat{y}_i) = (\hat{X}_i^{(h)}/\hat{Z}_i^{(h)}, \hat{Y}_i^{(h)}/\hat{Z}_i^{(h)})$.

After the three steps mentioned above, we have transformed every pixel (x_i, y_i) to where it is in the next frame $(\hat{x}_i, \hat{y}_i)$ according to the camera motion. We can then subtract one image point from the other to compute the camera motion flow (u, v) as $u(x_i, y_i) = \hat{x}_i - x_i$ and $v(x_i, y_i) = \hat{y}_i - y_i$.

B.5 Transform from Left Camera Motion to Right Camera Motion

Change the Calibration Files from Using the cam0 Coordinates to the Left Camera Coordinates There are three camera coordinates that involve our system: cam0, cam2, and cam3. The raw data set used four cameras in total to collect the images, with cam0 and cam2 on the left, and cam1 and cam3 on the right. Among these cameras, cam0 and cam1 collect gray-scale images, whereas cam2 and cam3 collect color images. Since we only use the color images as input, our left and right cameras refer to cam2 and cam3, respectively. However, the raw data set used the camera coordinates of cam0 as the world coordinates, so the projection matrices provided in the calibration files are actually projecting from the world cam0 coordinates to the image coordinates of each camera. Since all four cameras are in parallel on the same horizontal line, we know that their projection matrices should have the following relationships. Suppose P_0, P_l, and P_r are the projection matrices for cam0, cam2, and cam3, respectively.

$$P_0 = K[I_{3\times3} \mid \mathbf{0}], \quad P_l = K[I_{3\times3} \mid \boldsymbol{t}_l], \quad P_r = K[I_{3\times3} \mid \boldsymbol{t}_r],$$

where K is the camera intrinsics, $\boldsymbol{t}_l$ and $\boldsymbol{t}_r$ are the translation vectors from cam0 to cam2 and cam3. Note that $\boldsymbol{t}_l$ and $\boldsymbol{t}_r$ are exactly horizontal.

To simplify the system, we switch the world reference to the cam2 coordinates (the left view input) so that we can totally forget about cam0 in our system. This can be done by simply changing the translation component as follows.

$$P_l' = K[I_{3\times3} \mid \mathbf{0}], \quad P_r' = K[I_{3\times3} \mid \boldsymbol{t}_r - \boldsymbol{t}_l]$$

These two projection matrices now use the left camera coordinates as the world reference, so there is no cam0 involved now in the rest of the process.

The Relationship Between the Left Camera Motion to the Right Camera Motion Suppose the projection matrices of left and right view are $P_l = [K \mid \boldsymbol{v}_l]$ and $P_r = [K \mid \boldsymbol{v}_r]$, which transforms from any parallel world reference to each image coordinates. Given the left and right camera motion $C_l = [R_l \mid \boldsymbol{t}_l]$ and $C_r = [R_r \mid \boldsymbol{t}_r]$.

Note that the camera motion matrices transform from each point's previous coordinates to its camera coordinates in the next frame. For example, for a point $\boldsymbol{p}_1$ in the left view, the coordinates of the same point in the next frame (in the moved camera reference) can be found by $\boldsymbol{p}_2 = R_l\boldsymbol{p}_1 + \boldsymbol{t}_l$. Thus, in the first-frame reference, plugging in $\boldsymbol{p}_2 = \mathbf{0}$, we can find the coordinates of the moved origin as $\boldsymbol{\alpha} = -R_l^{-1}\boldsymbol{t}_l$. The new axes in the second-frame reference can also be found in R_l^{-1}, where each column corresponds to an axis of the second-frame camera reference represented in the first-frame reference system.

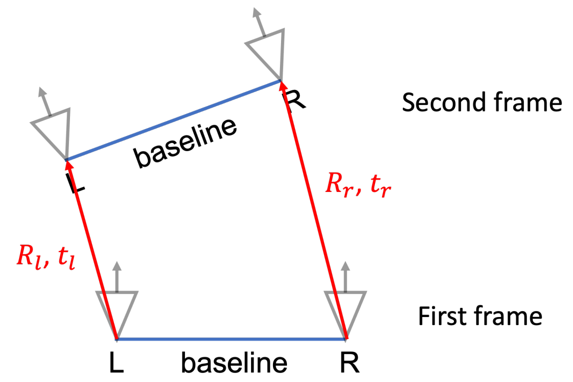


Figure B.1: The camera motions for left and right cameras

Now, if we consider the motion of the right view. The baseline vector can be found based on the camera calibration matrices. Since $P_l = [K \mid \boldsymbol{v}_l] = K[I_{3\times 3} \mid \boldsymbol{b}_l]$ and $P_r = [K \mid \boldsymbol{v}_r] = K[I_{3\times 3} \mid \boldsymbol{b}_r]$, the baseline vector from left to right is $-(\boldsymbol{b}_r - \boldsymbol{b}_l) = K^{-1}(\boldsymbol{v}_l - \boldsymbol{v}_r)$. Since the right camera is always in parallel with the left camera, the rotation component of the right camera should be the same as the left camera, so the second-frame right reference should also have axis vectors as shown in the columns of R_l^{-1}. Suppose the first column of R_l^{-1} is $\boldsymbol{r}_x$. the vector that points from the first-frame right origin to the second-frame right origin can be found as

$$\boldsymbol{\beta} = -K^{-1}(\boldsymbol{v}_l - \boldsymbol{v}_r) - R_l^{-1}\boldsymbol{t}_l + \|K^{-1}(\boldsymbol{v}_l - \boldsymbol{v}_r)\|\boldsymbol{r}_x.$$

If we plug in the camera parameters as in

$$P_l = \begin{bmatrix} f_x & 0 & c_x & f_x b_l \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad P_r = \begin{bmatrix} f_x & 0 & c_x & f_x b_r \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

we then have

$$K^{-1} = \begin{bmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$\boldsymbol{\beta} = (b_l - b_r)\left(\boldsymbol{r}_x - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right) - R_l^{-1}\boldsymbol{t}_l \; = \underbrace{(R_l^{-1} - I_{3\times 3})(\boldsymbol{b}_l - \boldsymbol{b}_r)}_{\text{motion of the baseline}} + \underbrace{(-R_l^{-1}\boldsymbol{t}_l)}_{\text{motion of the left camera}} .$$

Finally, the right camera motion matrix can be represented as

$$\begin{cases} R_r = R_l, \\ \boldsymbol{t}_r = -R_r\boldsymbol{\beta} = (R_l - I_{3\times 3})(\boldsymbol{b}_l - \boldsymbol{b}_r) + \boldsymbol{t}_l. \end{cases}$$

B.6 Data Augmentation for Images with Camera Calibration

It is common to apply data augmentation to the images in the training stage to improve model robustness. However, the data augmentation process is not straightforward in our case, where we have both images and their corresponding camera parameters as the input. When we augment the images, we may also need to transform the camera parameters accordingly.

For example, many algorithms first resize the input to a certain dimension before feeding into the network. This resizing process defines new image pixel grids and hence changes the mapping from the world coordinates to the image coordinates. Therefore, we need to modify the camera calibration matrices at the same time, with the projection matrix P as an example.

Appearance Change Appearance change includes all changes with respect to photometric parameters such as mean value, contrast, and gaussian blur. These changes do not modify the image grids, so there is no need to change the calibration matrix.

Resizing Suppose we resize the original image from size (w, h) to $(\alpha w, \beta h)$, and the original projection matrix is $P = K[I_{3\times3} \mid \boldsymbol{t}] = [K|\boldsymbol{v}]$. For any world point $\boldsymbol{p}$, its homogeneous image coordinates is

$$\boldsymbol{p}_h = \begin{bmatrix} x_h \\ y_h \\ z_h \end{bmatrix} = K\boldsymbol{p} + \boldsymbol{v},$$

and the 2D image coordinates will be $\boldsymbol{p}_i = (x_h/z_h, y_h/z_h)^T$.

After we resize the image, the x and y dimensions of $\boldsymbol{p}_i$ have also been scaled by α and β times, respectively, *i. e.*, $\boldsymbol{p}'_i = (\alpha x_h/z_h, \beta y_h/z_h)^T$. Thus, we have the new homogeneous coordinates

$$\boldsymbol{p}'_h = \begin{bmatrix} \alpha x_h \\ \beta y_h \\ z_h \end{bmatrix} = K'\boldsymbol{p} + \boldsymbol{v}',$$

where $P' = [K'|\boldsymbol{v}']$ is the new projection matrix. Thus, we obtain

$$K'\boldsymbol{p} + \boldsymbol{v}' = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} (K\boldsymbol{p} + \boldsymbol{v}),$$

so finally,

$$K' = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} K, \quad \boldsymbol{v}' = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{v}.$$

This shows that we need to scale the first row of P by α and second row by β.

Cropping Cropping does not change the scale of the axes but shift the origin of image coordinates. Suppose the top left corner of the cropping window is at (a, b) in the original image coordinates. Now, the old point (a, b) is at $(0, 0)$ in the new image coordinates, so we only need to subtract the (c_x, c_y) parameters in P by (a, b), *i. e.*,

$$P = \begin{bmatrix} f_x & 0 & c_x & f_x t \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \Longrightarrow \quad P' = \begin{bmatrix} f_x & 0 & c_x - a & f_x t \\ 0 & f_y & c_y - b & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Horizontal Flip Horizontal flip means we apply a 2D reflection $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ with translation $\begin{pmatrix} w \\ 0 \end{pmatrix}$ to the 2D image coordinates.

Using the same annotations as previous sections, we have the new image coordinates $\boldsymbol{p}_i' = (w - x_h/z_h, y_h/z_h)^T$ and its corresponding homogeneous coordinates

$$\boldsymbol{p}_h' = \begin{bmatrix} wz_h - x_h \\ \beta y_h \\ z_h \end{bmatrix} = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{p}_h,$$

so finally,

$$K' = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} K, \quad \boldsymbol{v}' = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{v}.$$

However, the first element of K' (where we have"f_x" originally) is now negative. This does not affect the transformation from the camera coordinates to the image coordinates but is somewhat confusing since we now have a negative focal distance parameter.

One way to address this issue mentioned above is to also flip the camera reference system horizontally so that the image points are mapped to a mirroed 3D world instead of the actual world. Since $P = K[I_{3\times3} \mid \boldsymbol{t}] = [K|\boldsymbol{v}]$, where $\boldsymbol{t} = (t, 0, 0)^T$, if we do a horizontal reflection of the 3D world points with respect to the y-z plane of the world reference, we have the new point coordinates

$$\boldsymbol{p}' = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{p}.$$

Note that we need to reflect the world points with respect to the y-z plane of the world reference instead of the camera references. This is because we need to make sure we yield consistently the same reflected world coordinates for both views, whose camera references are different.

For $P' = [K'|\boldsymbol{v}']$, we have $\boldsymbol{p}'_h = K'\boldsymbol{p}' + \boldsymbol{v}'$. Thus,

$$\boldsymbol{p}'_h = K' \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{p} + \boldsymbol{v}' = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} (K\boldsymbol{p} + \boldsymbol{v}), \quad \forall \boldsymbol{p} \in \mathbb{R}^3$$

and therefore

$$K' = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} K \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & w - c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

$$\boldsymbol{v}' = \begin{bmatrix} -1 & 0 & w \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{v} = -\boldsymbol{v}$$

since $\boldsymbol{v} = (f_x t, 0, 0)^T$. Note that the baseline has been reverted (the negative sign in $\boldsymbol{v}'$) because we reflect the world reference. Thus, when we later compute the baseline length using the differences between the $\boldsymbol{v}'$ parameters of the two views, we need to take absolute values to make sure the length is positive.

Appendix C

Summary of Datasets and Evaluation

C.1 Datasets

FlyingChairs The synthetic FlyingChairs data set [19, 70] is one of the most widely used data sets in optical flow prediction. It also has a second version [70] that provides labels for occlusion and motion boundaries, whereas the first version only has optical flow labels. This second version data set consists of 22k training samples and 640 validation samples and provides labels for optical flow, occlusion, and motion boundary in both the forward and the backward direction. Each sample is a pair of images (dimension: 384×512) that are generated by moving some chair patterns over a background scenery picture. It is a large data set that many state-of-the-art models are trained on. However, since the data are synthetic, the images and the motions are not very realistic, and performance may suffer on real data.

FlyingThings3D FlyingThings3D [90] is another very useful data set for stereo and motion analysis. The data set consists of left-right image pair sequences of length around 10 with image size 540×960, and if we split the sequences to consecutive frame pairs, there are around 22k training samples and 4k validation samples. The data set provides clean pass and final pass of the images, as well as the optical flow and disparity labels. It is also a synthetic data set very similar to FlyingChairs. The differences are (a) it uses multiple shapes of objects instead of only chairs, (b) it uses more complicated affine transformations, including rotation and scaling, (c) it provides stereo views and disparity labels. It may also worth noticing that the average optical flow magnitude is significantly larger than in the FlyingChairs data set, meaning that large motions are generally involved.

MPI-Sintel MPI-Sintel [91] is also a synthetic data set, but it is generated from an open source 3D animated short movie, and it looks more realistic than FlyingChairs and FlyingThings3D. The data set has been split into different types of scenes and has around 1k training samples and 552 test samples of dimension 436×1024. The official data set provides ground truth for optical flow and occlusion. This data set is very close to the real daily video domain, which includes various scales of body motions, and it has two data passes: clean and final. The clean pass renders pure geometry, color, shading, and shadows, whereas the final pass also includes multiple realistic renderings like smoke and motion blur. This data set is often used to evaluate how certain models can perform on almost-real data. It cannot be used as the only training set because of its small size, but fine-tuning on it is possible.

KITTI The KITTI data sets (KITTI-2012 [3], KITTI-2015 [246]) have real data captured from cameras mounted on a vehicle and provides ground-truth labels for optical flow and disparity computation. The ground-truth motion is collected from radar sensors on the car, and the labels are sparse (not all the pixels have labels). Since it is hard and expensive to collect the optical flow labels using this technique, this data set (the 2012 subset and 2015 subset combined) only has around 400 training samples and 400 test samples in total with dimension around 375×1242. Researchers specifically care about the performance on this data set because it is currently the only real labeled data available and because the data are relevant to autonomous driving applications.

KITTI-Odometry The KITTI Odometry data set [3] is an extension of KITTI used in SLAM tasks. The data set contains 22 left-right image sequences of similar image dimensions as in KITTI, where 11 of them are the training data with camera pose ground truth of each frame. The camera pose labels are captured by the IMU sensors on the vehicle, which can be used to compute the relative camera motion between every two frame.

KITTI-raw The KITTI data set also provides raw data, including multiple video sequences of each camera and the corresponding velodyne lidar points. Both the KITTI-2012/2015 and the KITTI-Odometry data set are generated from these raw data. In our experiments, we use the raw data to generate our training samples including the stereo frame pairs and the ground-truth depth labels. This is because the KITTI-2012/2015 and the KITTI Odometry data set are too small to train an unsupervised model, so we need to generate more from raw data. The data set also provides the camera calibration and odometry information so that we can pick meaningful samples to use. For example, we eliminate stationary samples so that the data are not redundant. We use the same test split as in [214] for comparison.

C.2 Evaluation Metrics

Optical Flow The error for flow prediction is evaluated by the average End Point Error (EPE). This error measures the norm of the differences between the predicted flow vectors and the ground-truth vectors, averaged over the whole frame. Specifically, suppose for pixel $(x, y)^T \in \Omega$ in the frame, the ground-truth and predicted flow are $\boldsymbol{u}^*(x, y)$ and $\boldsymbol{u}(x, y)$ respectively. The EPE can be expressed as

$$e_f(x, y) = \|\boldsymbol{u}(x, y) - \boldsymbol{u}^*(x, y)\|_2$$

$$\mathrm{EPE}_f = \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} e_f(x, y).$$

Another error measure, usually used in KITTI data sets, is the outlier percentage Fl. An optical flow vector is considered outlier if the norm of its error is larger than 3 pixels *and* larger than 5% of the ground-truth flow magnitude.

$$\mathrm{Fl} = 100 \cdot \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} \mathbb{I}\left(e_f(x,y) > 3 \ \wedge \ e_f(x,y) > 0.05 \cdot \|\boldsymbol{u}^*(x,y)\|_2\right),$$

where $\mathbb{I}(\cdot)$ denotes the indicator variable.

Disparity Since disparity is also finding point correspondences between frames, its evaluation metrics are similar to those of flow. Specifically, for disparity prediction $d(x,y)$ and ground truth $d^*(x,y)$ the EPE and the outlier rate D1 are

$$e_d(x,y) = |d(x,y) - d^*(x,y)|$$

$$\mathrm{EPE}_d = \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} e_d(x,y),$$

$$\mathrm{D1} = 100 \cdot \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} \mathbb{I}\left(e_d(x,y) > 3 \ \wedge \ e(x,y) > 0.05 \cdot |d^*(x,y)|\right).$$

Depth Given the depth estimation $D(x,y)$ and its ground truth $D^*(x,y)$, there are many ways to evaluate, including four error measures and three accuracy measures. The error measures (the smaller, the better) are the absolute relative error (AbsRel), the squared relative error (SqRel), the rooted mean squares (RMS), and the rooted mean squares in logarithmic scale (RMSlog)

$$\mathrm{AbsRel} = \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} \frac{|D(x,y) - D^*(x,y)|}{D^*(x,y)},$$

$$\mathrm{SqRel} = \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} \frac{(D(x,y) - D^*(x,y)^2}{D^*(x,y)},$$

$$\mathrm{RMS} = \sqrt{\frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} (D(x,y) - D^*(x,y))^2},$$

$$\mathrm{RMSlog} = \sqrt{\frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} (\log D(x,y) - \log D^*(x,y))^2}.$$

The accuracy measures (the larger, the better) are the percentages of accurate predictions δ_t under different thresholds t

$$\delta_t = \frac{1}{|\Omega|} \sum_{(x,y)^T \in \Omega} \mathbb{I}\left(\max\left(\frac{D(x,y)}{D^*(x,y)}, \frac{D^*(x,y)}{D(x,y)} \right) < t \right).$$

In experiments, the thresholds $t = 1.25$, $t = 1.25^2$, and $t = 1.25^3$ are commonly used.

Camera motion Given the matrix-form ground-truth camera motion $\boldsymbol{C}^* = [\boldsymbol{R}^* | \boldsymbol{t}^*]$ and our estimation $\boldsymbol{C} = [\boldsymbol{R} | \boldsymbol{t}]$. The error is defined based on whether the inverse of our estimated transformation can cancel with the ground-truth transformation, or in other words, if we first apply the ground-truth transformation and then the inverse of our estimated transformation, whether that results to be an identity mapping.

Specifically, given any point $\boldsymbol{x}$, the ground truth maps it to $\boldsymbol{y} = \boldsymbol{R}^* \boldsymbol{x} + \boldsymbol{t}^*$. Then, we apply the inverse of our estimation, which gives us

$$\hat{\boldsymbol{x}} = \boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{t}) = \boldsymbol{R}^{-1}\boldsymbol{R}^* \boldsymbol{x} + \boldsymbol{R}^{-1}(\boldsymbol{t}^* - \boldsymbol{t}).$$

This equation shows us that the error actually consists of two parts, the rotation error (indicated by $\boldsymbol{R}^{-1}\boldsymbol{R}^*$) and the translation error (indicated by $\boldsymbol{R}^{-1}(\boldsymbol{t}^* - \boldsymbol{t})$).

For the rotation error, since it is a 3D rotation matrix, we evaluate it using the angle of the rotation θ, which can be found by

$$|\theta| = \arccos\left(\frac{\mathbf{Tr}(\boldsymbol{R}^{-1}\boldsymbol{R}^*) - 1}{2} \right).$$

For the translation error, we can simply use the L2 distance to measure the error $\|\boldsymbol{R}^{-1}(\boldsymbol{t}^{*} - \boldsymbol{t})\|_2$. Note that since $\boldsymbol{R}$ is orthogonal, we can use $\boldsymbol{R}^{T}$ instead of $\boldsymbol{R}^{-1}$ in implementations.

Appendix D

Appendix for Optical Flow Training under Limited Label Budget via Active Learning

D.1 Methodology Details

D.1.1 Semi-supervised Training

In this work, we explore the spectrum between totally supervised (100% labeled) and totally unsupervised (0% labeled) training. The question is: what is the intermediate state of semi-supervised learning if we have exactly a fraction r of training samples labeled ($0 < r < 1$)? Intuitively, the performance should be monotonically increasing when we increase the label ratio r.

Specifically, the most ideal setting requires us to find a semi-supervised learning scheme that

- trains using the information of all labeled and unlabeled samples at the same time, and
- is continuous at $r = 0$ (unsupervised) and $r = 1$ (supervised), *i.e.,* , if we set $r = 0$, it should be equivalent as the current unsupervised learning pipeline, and if we set $r = 1$, it should be equivalent as the fully supervised setting.

We want to define our semi-supervised setting as a smooth transition between the supervised and unsupervised settings. The naive solution is to train a fixed neural network architecture with a semi-supervised loss that works differently for the labeled and unlabeled samples. For example,

$$\ell_{\text{semi}}(\boldsymbol{x}) = \begin{cases} \ell_{\text{unsup}}(\boldsymbol{x}), & \text{if } \boldsymbol{x} \text{ is unlabeled,} \\ \alpha\ell_{\text{sup}}(\boldsymbol{x}), & \text{otherwise,} \end{cases} \tag{D.1}$$

where $\alpha > 0$ is the coefficient to balance the two different losses. We then have the final loss term

$$\mathcal{L}_{\text{semi}} = \sum_{\boldsymbol{x} \in \mathcal{D}} \ell_{\text{semi}}(\boldsymbol{x}) = \sum_{\boldsymbol{x} \in \mathcal{D}^u} \ell_{\text{unsup}}(\boldsymbol{x}) + \alpha \sum_{\boldsymbol{x} \in \mathcal{D}^l} \ell_{\text{sup}}(\boldsymbol{x}),$$

where $\mathcal{D}^u$ and $\mathcal{D}^l$ are the unlabeled and labeled sample set, and $\mathcal{D} = \mathcal{D}^u \cup \mathcal{D}^l$. Under this setting, the label ratio is $r = |\mathcal{D}^l|/(|\mathcal{D}^u| + |\mathcal{D}^l|)$, and changing r from 0 to 1 will change the setting smoothly from unsupervised to supervised.

However, one concern with this setting is that we need to fix the same dataset $\mathcal{D}$ for all experiments with varying $0 \leq r \leq 1$, but the datasets used in current state-of-the-art supervised and unsupervised methods are usually different. For example, to get the best results on the Sintel dataset [73], unsupervised methods first train on the Sintel raw movie dataset and then fine-tune on Sintel. However, the latest supervised methods usually first train on the FlyingChairs [19] and FlyingThings3D [72] datasets before training on the small Sintel set. This difference in dataset is important to notice because it is one of the advantages of unsupervised learning that it can use much more data (probably from the same data distribution as the test data) than supervised training.

In light of the problem mentioned above, we decide to use the unsupervised datasets as our data in the semi-supervised training. There are mainly two reasons. First of all, the label ratio can be very low in daily practice, so defining our setting closer to the unsupervised setting may be more practical. Second, the unsupervised training is harder to converge than the supervised training because of the lack of supervisory signals, so using a framework that is closer to the unsupervised training may be better for convergence in both scenarios.

Another concern is in the training schedule. In the setting defined above, we only have one stage of training that use all labeled and unlabeled samples in the same stage. Another option is to split to two stages, one unsupervised stage using all unlabeled samples (similar as a pre-training stage) and a supervised stage using the labeled samples. We discuss the pros and cons of both schedules in Appendix D.2.3.

D.1.2 Network Choice: Why ARFlow? Why Not RAFT?

We found that RAFT is not appropriate to be tested at this stage because it has been mostly proven to work in the supervised setting, but our semi-supervised flow is actually much closer to the unsupervised setting in the following two ways.

- Our label ratio is very low (5-10%), which is almost unsupervised. The supervision signal is extremely sparse.
- Our first training stage is unsupervised, so the model is initialized in an unsupervised way.

Therefore, a reliable unsupervised base model is preferred in our setting. This is why we choose ARFlow [1] (unsupervised SOTA) instead of RAFT [22] (supervised SOTA).

Admittedly, there is recent work [29] on unsupervised versions of RAFT. However, this work is based on multi-frame inputs, and it also adds too much complexity (such as self-supervision) into the model, so we do not think it is the right time to move towards RAFT now. However, we do agree that it is worth trying in the future once a simple and reliable unsupervised appraoch for RAFT is available.

D.2 Experiment Details

D.2.1 Summary of Available Datasets

Official Datasets We train and evaluate our method on two large synthetic datasets, FlyingChairs [19] and FlyingThings3D [72], as well as two more realistic datasets, Sintel [73] and KITTI [74, 33].

FlyingChairs [19] and FlyingThings3D [72] consist of image pairs generated by moving chairs or everyday objects across the background images along randomized 3D trajectories. These two datasets are large but unrealistic, so they are usually only used to pre-train supervised networks.

Sintel [73] is a challenging benchmark dataset obtained from a computer-animated movie. This dataset is closer to real-life scenes as it contains fast motions, large occlusions, and many realistic artifacts like illumination change and fog or blur. It provides both clean and final passes with corresponding dense optical flow labels. Apart from that, the unlabeled raw movie frames have also been used in many recent unsupervised work [28, 1].

KITTI dataset was first released in 2012 [74] and extended in 2015[33]. The dataset contains frame pairs of road scenes from a camera mounted on a car. Sparse optical flow labels are provided using 3D laser scanner and egomotion information. KITTI raw frames with no labels are also available and used in unsupervised training [75, 1, 76].

Tab. D.1 summarizes the dataset information. We have excluded the labeled samples from the raw Sintel and KITTI dataset, so all splits in the table are disjoint.

Our Data Splits We cannot test our model using the official KITTI and Sintel test set for all the experiments since the website restricts the number of submissions.

Table D.1: Available official datasets for optical flow estimation.

Dataset	Split	# of samples	Labeled?
FlyingChairs	train	22,232	✓
	val	640	✓
FlyingThings3D	train	19,621	✓
	val	3,823	✓
Sintel	raw	12,466	✗
	clean	1,041	✓
	final	1,041	✓
KITTI	raw	27,858	✗
	2012	194	✓
	2015	200	✓

Table D.2: Our train/val split of Sintel and KITTI

Dataset	Our train split	# train	# val
Sintel clean+final	alley_1, ambush_4, ambush_6, ambush_7, bamboo_2, bandage_2, cave_2, market_2, market_5, shaman_2, sleeping_2, and temple_3	1082	1000
KITTI 2015+2012	first 150 samples for each	300	94

We test on the official test set only for the major final models in our paper. Thus, we need to split our own validation set from Sintel and KITTI.

Our own train/val split is shown in Tab D.2. For Sintel, as suggested in the official implementation of ARFlow [1], we split the following folders of both clean and final passes as our *train* split of Sintel: alley_1, ambush_4, ambush_6, ambush_7, bamboo_2, bandage_2, cave_2, market_2, market_5, shaman_2, sleeping_2, and temple_3. For KITTI, we take the first 150 samples in each of the 2015 set and 2012 set as our *train* split and the rest as our *val* split.

D.2.2 Data Augmentation Parameters

The data augmentation parameters in our experiments are summarized in Tab. D.3. Our data augmentation implementations are borrowed from the official code base of ARFlow [1] and RAFT [22]. We use ColorJitter from the torchvision.transforms

Table D.3: Data augmentation parameters

	FlyingChairs	FlyingThings3D	Sintel	KITTI
Cropping	384 × 448	384 × 768	384 × 768	320 × 960
Rescaling	✗	✗	scale $\in [2^{-0.2}, 2^{0.6}]$ with prob. 0.8	✗
Horizontal flip	with prob. 0.5	with prob. 0.5	with prob. 0.5	with prob. 0.5

Appearance	brightness = 0.5
	contrast = 0.5
	saturation = 0.5
	hue = 0
	gamma = `True`
	gblur = `True`
		brightness = 0.5
		contrast = 0.5
		saturation = 0.5
		hue = 0
		gamma = `True`
		gblur = `True`
			brightness = 0.4
			contrast = 0.4
			saturation = 0.4
			hue = 0.16
			gamma = `True`
			gblur = `True`
				brightness = 0.3
				contrast = 0.3
				saturation = 0.3
				hue = 0.1
				gamma = `False`
				gblur = `True`

package to implement the appearance transformations. In addition, "gamma" means raising the normalized image color value (between 0 and 1) to a power sampled between 0.7 and 1.5 uniformly, and "gblur" means applying gaussian blur with radius 3 with probability 0.5.

D.2.3 Training Schedule Design

Three options of the training pipelines are listed below. We now explain and discuss them one by one.

A. train on *all* data (semi-sup)

B. train on *all* data (unsup) → query partial labels from *all* data → train on *all* data (semi-sup)

C. train on *non-candidate* set (unsup) → query partial labels from *candidate* set → train on *candidate* set (semi-sup)

A one-stage training schedule (option A) can be used if our goal is only to visualize the change of performance when we gradually increase the label ratio from 0 to 1. We can simply train on the full dataset with partial labels using the semi-supervised loss in one stage. Thus, we use this setting in our first experiment to draw the label ratio-validation error curves. We randomly shuffle and mix labeled and unlabeled samples in mini-batches to stabilize our training. However, this assumes that the partial labels have to be assigned *before* training independent of the model and thus may be naive compared with the other two options, which use active learning.

Now, we want to explore a semi-supervised training pipeline where the labels are assigned *during* the training. This has to be a pipeline of at least two stages because we need to query labels at some point in the process. Specifically, as shown in option B, we first have a totally unlabeled dataset, so we train our first model using the unsupervised loss. Then, based on the current trained model, we pick a part of the dataset that can help the current model most to query labels. Subsequently, we continue training using the semi-supervised loss. This reflects a workflow that can be applied in real practice so that the researchers only need to pay for the partial labels that can help the most. Note that we can easily change the pipeline to query labels multiple times by stacking more stages in the end.

One problem for option B is that it assumes every sample can be labeled. However, in real life, it is possible that only a subset of the original dataset can be labeled. For instance, labeling the ground-truth flow of an autonomous driving dataset (like KITTI) requires lidar sensors deployed when the videos are collected. If a raw video does not have the corresponding lidar information, it cannot be labeled but can still be used in the unsupervised part of training. Therefore, a more general setting is to define a candidate set to indicate those samples that can be labeled. Note that we can always split the full dataset manually to a candidate and a non-candidate splits even if every sample is eligible to get the label, which actually brings benefits in generalization as we will discuss next.

After splitting the dataset to a candidate and a non-candidate set, we can define the pipeline as in option C above. We first do unsupervised training on the non-candidate set and then use the current model to select samples out of the candidate set to get labels. This is beneficial because the model has not seen the candidate set in its first stage of unsupervised training. This can help add generalization ability because when we select samples to label, we are actually validating the current model on the new unseen candidate set. The selected samples are thus the ones that can help the current model generalize the most. This is why we stick to option C as our experiment settings in all the experiments on KITTI and Sintel.

Experiments on semi-supervised training (drawing the label ratio-validation error curves) Since our interest in this experiment is to see how the error changes when we assign different ratios of labels, we first use the simplest training schedule (option A) on two toy datasets, FlyingChairs and FlyingThings3D, to plot the whole figure. The experiment settings are as follows.

- FlyingChairs: train on the *train* split (semi-sup), evaluate on the *val* split
- FlyingThings3D: train on the *train* split (semi-sup), evaluate on the *val* split

Subsequently, we also would like to see the curve on two regular datasets, Sintel and KITTI, but since a large part of the data (raw dataset) is not labeled, we have to pre-train on those data in an unsupervised manner. This fits into the reason for specifying a candidate set, where only part of the data we have in hand are eligible to query labels. Moreover, to better fit the state-of-the-art unsupervised training schedule, we adopt option C as our training schedule. For Sintel and KITTI, we assign our *train* split as the candidate set, and the large unlabeled data (*raw* sets) as the non-candidate set[1], yielding the following training schedules.

---

[1] We are not using the KITTI multi-view extension set for simplicity.

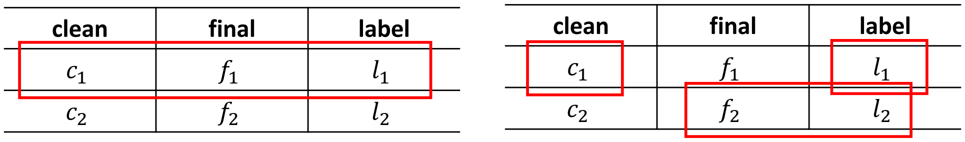
- Sintel: train on *raw* Sintel videos (unsup) → *randomly* select and assign labels for our *train* split → train on our *train* split (semi-sup) → evaluate on our *val* split.
- KITTI: train on *raw* KITTI videos (unsup) → *randomly* select and assign labels from our *train* split → train on our *train* split (semi-sup) → evaluate on our *val* split.

Experiments on our active learning algorithms We consider many heuristics as the algorithms to select samples to label. We use Sintel and KITTI datasets and apply the same training schedule as in the previous experiments. The only difference is that we now use our algorithms to select samples to label instead of random selection.

- Sintel: train on *raw* Sintel videos (unsup) → apply our *active learning algorithms* to select and assign labels for our *train* split → train on our *train* split (semi-sup) → evaluate on our *val* split.
- KITTI: train on *raw* KITTI videos (unsup) → apply our *active learning algorithms* to select and assign labels from our *train* split → train on our *train* split (semi-sup) → evaluate on our *val* split.

D.2.4 A Special Note on Sintel Label Queries in Pairs

When we run experiments on Sintel, the same set of labels are provided for both clean and final pass input frames, since the final pass is simply another rendering of the same content with more realistic artifacts like motion blur. In other words, we always ensure that the corresponding clean and final samples are either both labeled or both unlabeled. The reasons are as follows.



(a) Sampling in pairs (b) Sampling separately

Figure D.1: Examples of two different sampling methods on Sintel

In our project, we want to investigate the trade-off between model performance and annotation costs. We use label ratio r to represent annotation cost, so we need to make sure that the total fraction of labels needed in our experiment is consistent with the label ratio r. A simple example is shown in Fig. D.1. Suppose we have a tiny training set of only two clean-final pairs (c_1, f_1) and (c_2, f_2), and we set the label ratio $r = 0.5$. We have a total of four samples, so we need to select two of them to be labeled. If we sample clean and final images in pairs (as it is done in our experiments), the results may be like in Fig. D.1(a), where only l_1 (a half of the label set, consistent with $r = 0.5$) is needed in the experiment.

However, if we select clean and final samples separately, the selection may be like Fig. D.1(b). In this case, c_1 and f_2 are selected to be labeled, so both l_1 and l_2 are needed. This is inconsistent with $r = 0.5$ because 100% of the label set is needed, meaning that the annotation cost here is 100%. Therefore, the label ratio r here does not represent the actual annotation cost needed in the experiment. Even though the labels are not used during 100% of the training time (*e.g.,* , l_1 is not used when we train with f_1), we still need to pay full cost for annotating l_1 and l_2. Thus, this alternative sampling method does not support our investigation since r does not reflect the annotation cost accurately.

Another option may be to use only one split (clean or final) for the experiments, *i.e.,* , training one semi-supervised model only on the clean split and another model only on the final split. This also solves the problem above that the label ratio r does not reflect the true annotation cost. Nevertheless, this setting is largely different from most of the previous work, where both clean and final images are used to train one model that works on both passes at the same time. In this case, we are not able to compare with previous results. Such comparisons are crucial because we want to show that our semi-supervised models are significantly better than the state-of-the-art unsupervised models and also close to the supervised results.

D.3 More Data and Results

D.3.1 Raw Validation Data

Our raw data values are shown in Tabs. D.4 and D.5. All pseudo error bars are obtained by taking the standard deviations in the last 50 epochs or 50k iterations. Semi-supervised training validation errors (Fig. 2 in the paper) are shown in Tab. D.4, and active learning validation errors (Fig. 3 in the paper) are shown in Tab. D.5.

D.3.2 Benchmark Qualitative Results

Some qualitative results are shown in Fig. D.2. We can see that our active learning method is especially effective at hard sequences like “ambush”, “cave”, “market” and “temple”, and less effective at easy sequences where errors are already very small even for the unsupervised model. KITTI qualitative results are also shown in Fig. D.3, where the differences are less visible with the naked eye.

D.3.3 Analysis on More Uncertainty Scores

We have also tried more metrics with heuristics defined as below.

- *flow norm*: the 2-norm of the estimated flow vectors averaged across the frame, used to reflect large motions.

Table D.4: Validation error for semi-supervised training on different datasets

Label ratio r	FlyingChairs EPE/px	Sintel clean EPE/px	Sintel final EPE/px
0	3.066(±0.044)	1.906(±0.013)	2.933(±0.010)
0.05	2.369(±0.033)	1.850(±0.014)	2.828(±0.020)
0.1	2.091(±0.046)	1.776(±0.018)	2.710(±0.023)
0.2	1.803(±0.018)	1.691(±0.011)	2.598(±0.022)
0.4	1.653(±0.037)	1.643(±0.014)	2.349(±0.031)
0.6	1.560(±0.039)	1.625(±0.008)	2.281(±0.022)
0.8	1.550(±0.043)	1.581(±0.015)	2.281(±0.015)
1	1.439(±0.052)	1.651(±0.018)	2.290(±0.013)

Label ratio r	FlyingThings3D EPE/px	KITTI 2012 Fl/%	KITTI 2015 Fl/%
0	12.037(±0.500)	5.827(±0.057)	12.742(±0.090)
0.05	10.588(±0.444)	5.525(±0.038)	11.462(±0.088)
0.1	10.205(±0.694)	5.325(±0.042)	11.030(±0.128)
0.2	9.584(±0.132)	5.137(±0.050)	10.357(±0.096)
0.4	8.395(±0.307)	4.899(±0.036)	10.109(±0.087)
0.6	8.296(±0.154)	4.973(±0.049)	9.947(±0.150)
0.8	7.833(±0.152)	4.709(±0.057)	9.784(±0.134)
1	7.876(±0.283)	4.562(±0.047)	9.448(±0.134)

- *img grad norm*: the magnitude of gradients of the input images, used to reflect edges in the scene

- *texture score*: used to evaluate whether the input images have good textures (high scores for good textures); computed based on Good Features to track [17]. We select 16*16 windows with stride=8. For each window, we compute the Z matrix for each pixel from image gradients and add them up to get a summed 2-by-2 positive semi-definite symmetric matrix. We compute the smaller eigenvalues (must positive) of the matrix for each window and take the average.

- *color change*: used to indicate illumination change. We compute the color histogram for each RGB channel as well as its cumulative distribution. We compute the distances between the cumulative distributions of the first and second frames and take average across the RGB channels.

- *param grad norm*: the norm of the loss gradients with respect to the network parameters. Intuitively, if a sample contributes large gradients to the network, it is likely that this sample does not fit well with the current network, so it may need labels.

- *max corr vol*: the maximum value of the correlation volume at each pixel averaged across the whole frame. We use the correlation volume at the second-level decoder here. Intuitively, a large maximum correlation volume means that the pixel has a good match within the window, so the error may be small.

We plot similar correlation matrices (as the ones in the last part of the main paper) with all our metrics in Fig. D.4. We can see that all metrics are more or less consistent with our intuitions. Note that the "img grad norm" and "texture score" are negatively correlated with the errors because larger values indicate better textures and thus smaller estimation errors. Also, "max corr vol" is negatively correlated because larger values indicate better matches found for the first image pixels. From Fig. D.4, we can see that the metrics that are more correlated with the errors are "occ ratio", "flow grad norm", "photo loss", which are then used in our experiments.

Comparing the Sintel correlation matrix (Fig. D.4(a)) from that of KITTI (Fig. D.4(b)), we can see that the Sintel metrics are generally more correlated with the errors, whereas KITTI metrics are generally less effective in detecting the samples of large errors. Especially for the texture related scores like "img grad norm" and "texture score", Sintel errors have correlations around 0.5, but KITTI errors are almost independent of the sample errors. We guess it may be because KITTI can already achieve pretty decent results by merely learning the flow distribution patterns (the looming motion), so it does not have to track every patch closely.

Table D.5: Active learning validation errors, mean and std

Label ratio r	Method	Sintel	
		clean EPE/px	final EPE/px
0	-	1.906 (±0.013)	2.933 (±0.010)
0.05	random	1.850 (±0.014)	2.828 (±0.020)
	photo loss	1.807 (±0.010)	2.731 (±0.015)
	occ ratio	**1.767 (±0.019)**	**2.693 (±0.017)**
	flow grad norm	1.797 (±0.017)	2.770 (±0.016)
0.1	random	1.776 (±0.018)	2.710 (±0.023)
	photo loss	1.706 (±0.006)	2.541 (±0.018)
	occ ratio	**1.686 (±0.013)**	**2.515 (±0.018)**
	flow grad norm	1.696 (±0.009)	2.545 (±0.017)
0.2	random	1.691 (±0.011)	2.598 (±0.022)
	photo loss	1.639 (±0.010)	2.383 (±0.025)
	occ ratio	1.643 (±0.013)	2.373 (±0.018)
	flow grad norm	**1.631 (±0.016)**	**2.299 (±0.019)**
1	-	1.651 (±0.018)	2.290 (±0.013)

Label ratio r	Method	KITTI	
		2012 Fl/%	2015 Fl/%
0	-	5.573 (±0.056)	12.062 (±0.153)
0.05	random	5.363 (±0.080)	11.456 (±0.158)
	photo loss	5.477 (±0.032)	11.705 (±0.112)
	occ ratio	**5.256 (±0.040)**	**10.689 (±0.101)**
	flow grad norm	5.353 (±0.047)	10.994 (±0.171)
0.1	random	5.273 (±0.034)	10.480 (±0.108)
	photo loss	5.175 (±0.040)	10.441 (±0.087)
	occ ratio	5.170 (±0.043)	**10.148 (±0.110)**
	flow grad norm	**5.159 (±0.039)**	10.880 (±0.135)
0.2	random	5.021 (±0.061)	9.962 (±0.096)
	photo loss	4.934 (±0.033)	9.759 (±0.140)
	occ ratio	4.929 (±0.043)	9.736 (±0.147)
	flow grad norm	**4.837 (±0.046)**	**9.731 (±0.122)**
1	-	4.446 (±0.034)	8.545 (±0.086)

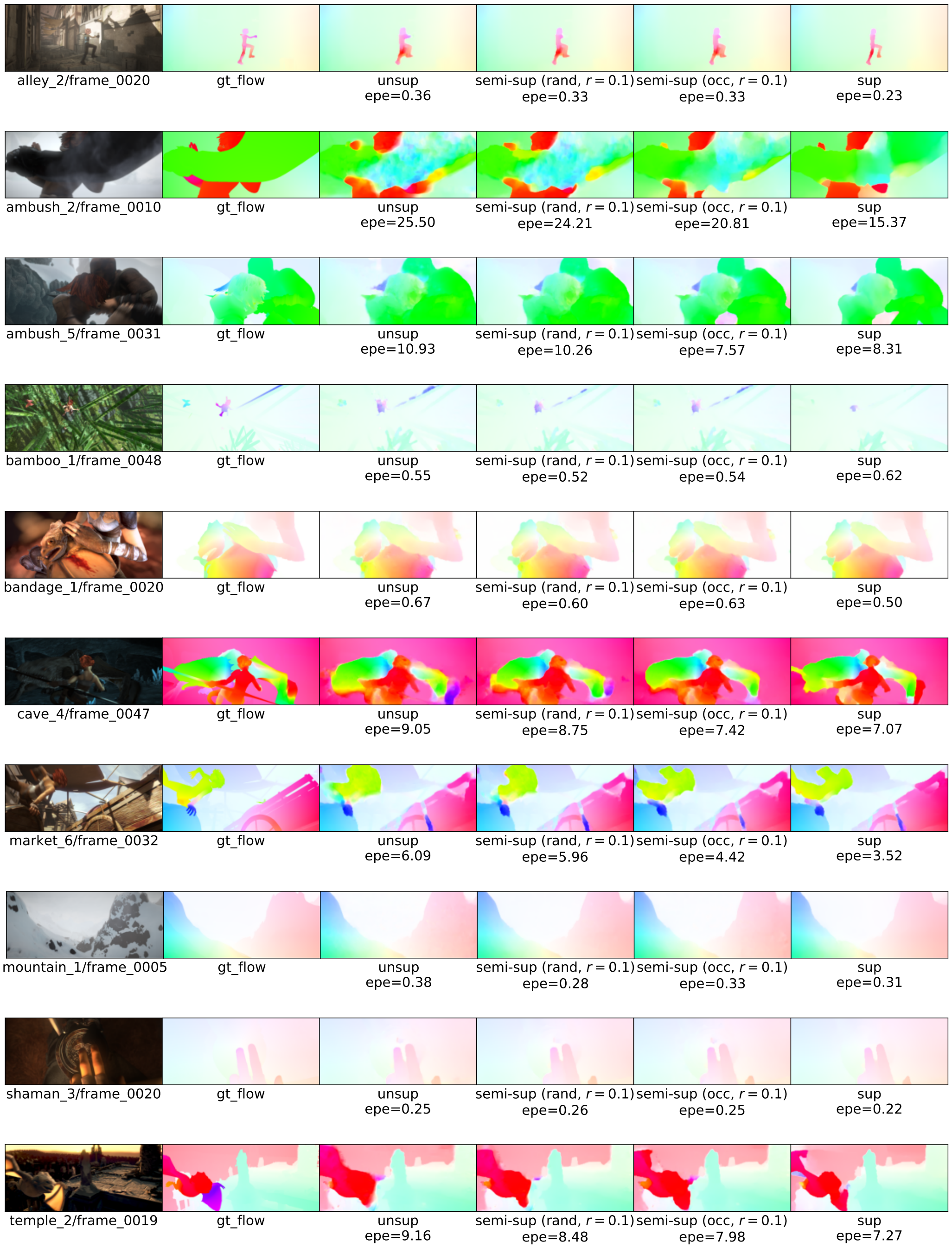


Figure D.2: Qualitative results on Sintel. Examples selected form our final pass validation split.

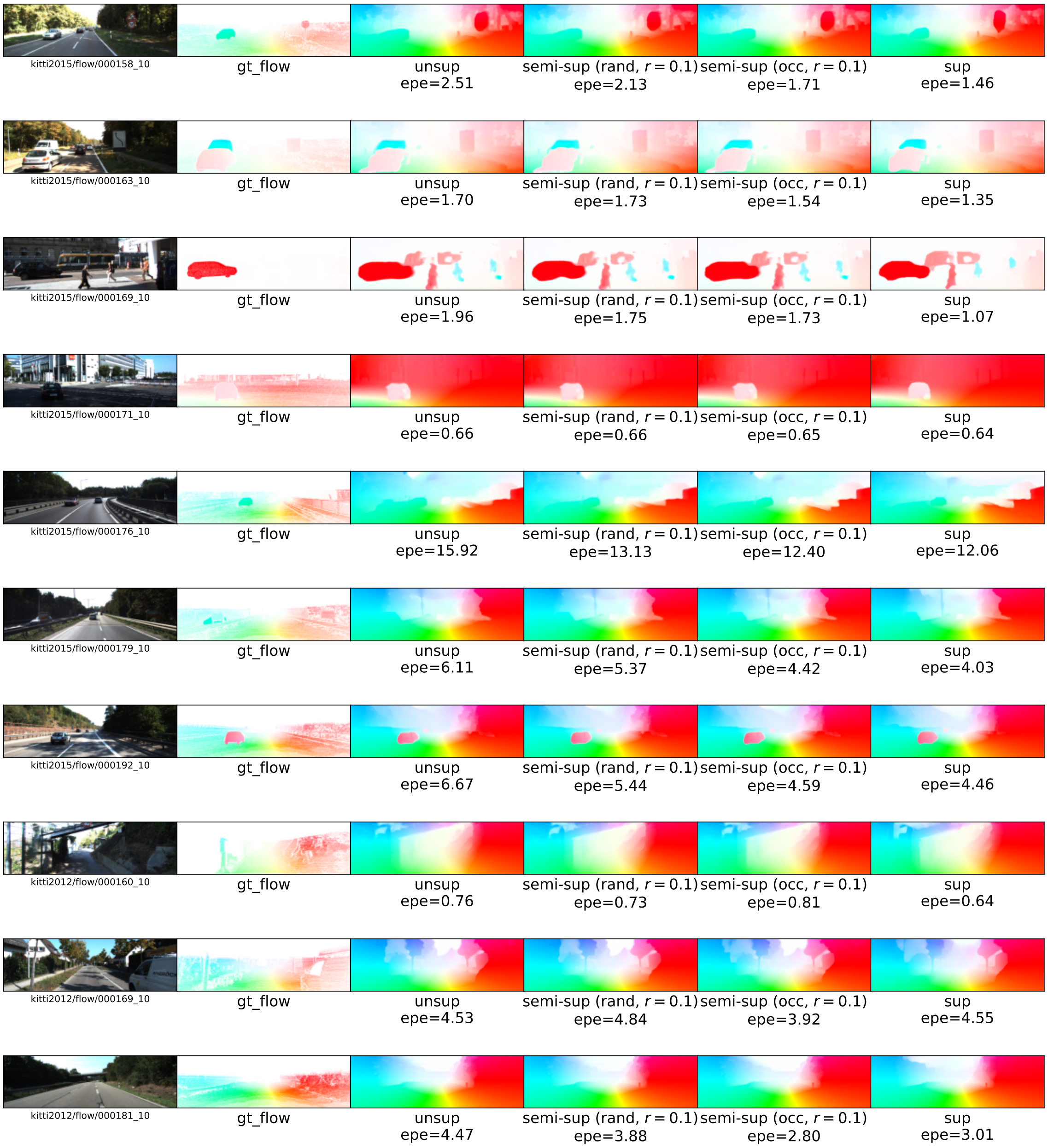
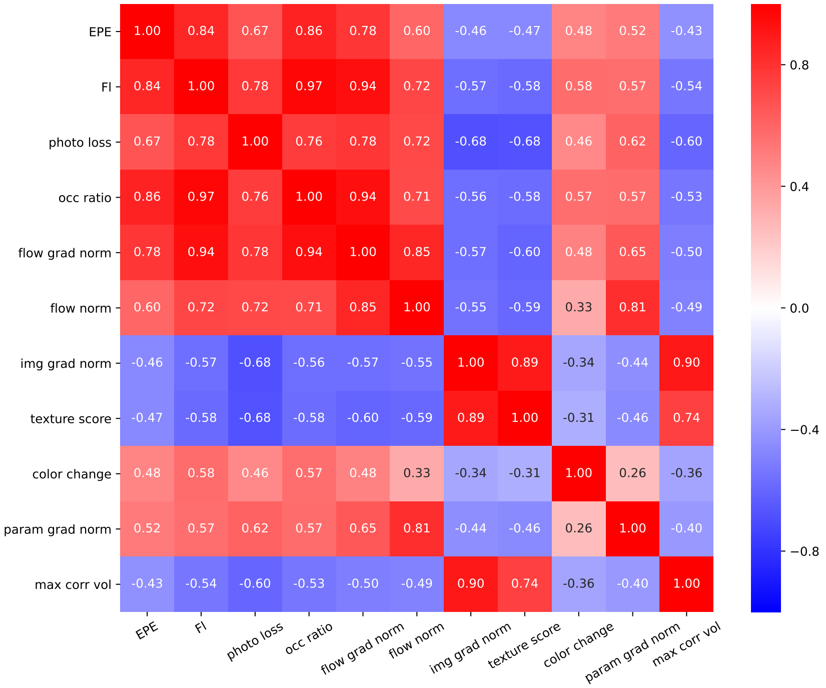
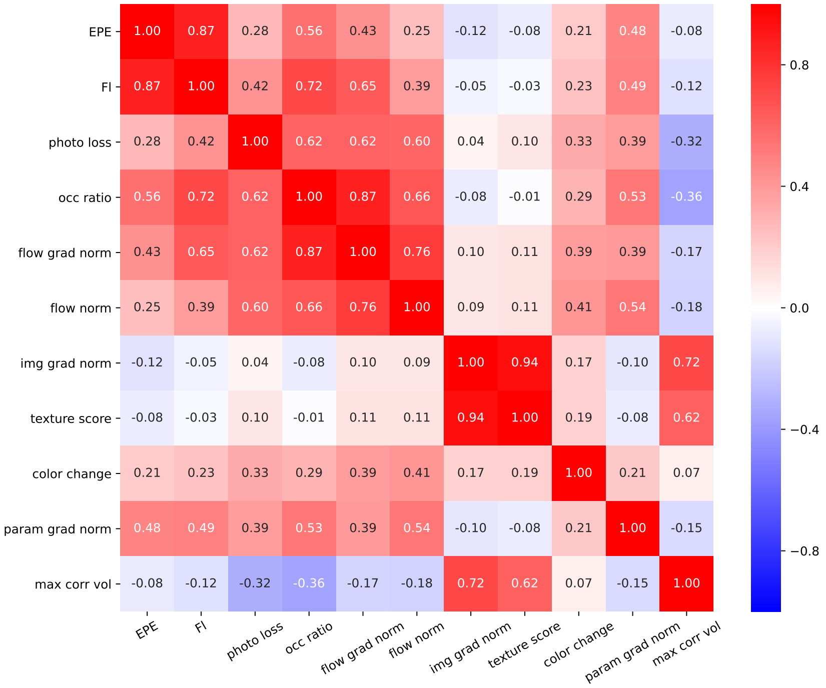


Figure D.3: Qualitative results on KITTI. Examples are selected from our validation split.



	EPE	Fl	photo loss	occ ratio	flow grad norm	flow norm	img grad norm	texture score	color change	param grad norm	max corr vol
EPE	1.00	0.84	0.67	0.86	0.78	0.60	-0.46	-0.47	0.48	0.52	-0.43
Fl	0.84	1.00	0.78	0.97	0.94	0.72	-0.57	-0.58	0.58	0.57	-0.54
photo loss	0.67	0.78	1.00	0.76	0.78	0.72	-0.68	-0.68	0.46	0.62	-0.60
occ ratio	0.86	0.97	0.76	1.00	0.94	0.71	-0.56	-0.58	0.57	0.57	-0.53
flow grad norm	0.78	0.94	0.78	0.94	1.00	0.85	-0.57	-0.60	0.48	0.65	-0.50
flow norm	0.60	0.72	0.72	0.71	0.85	1.00	-0.55	-0.59	0.33	0.81	-0.49
img grad norm	-0.46	-0.57	-0.68	-0.56	-0.57	-0.55	1.00	0.89	-0.34	-0.44	0.90
texture score	-0.47	-0.58	-0.68	-0.58	-0.60	-0.59	0.89	1.00	-0.31	-0.46	0.74
color change	0.48	0.58	0.46	0.57	0.48	0.33	-0.34	-0.31	1.00	0.26	-0.36
param grad norm	0.52	0.57	0.62	0.57	0.65	0.81	-0.44	-0.46	0.26	1.00	-0.40
max corr vol	-0.43	-0.54	-0.60	-0.53	-0.50	-0.49	0.90	0.74	-0.36	-0.40	1.00

(a) Sintel



	EPE	Fl	photo loss	occ ratio	flow grad norm	flow norm	img grad norm	texture score	color change	param grad norm	max corr vol
EPE	1.00	0.87	0.28	0.56	0.43	0.25	-0.12	-0.08	0.21	0.48	-0.08
Fl	0.87	1.00	0.42	0.72	0.65	0.39	-0.05	-0.03	0.23	0.49	-0.12
photo loss	0.28	0.42	1.00	0.62	0.62	0.60	0.04	0.10	0.33	0.39	-0.32
occ ratio	0.56	0.72	0.62	1.00	0.87	0.66	-0.08	-0.01	0.29	0.53	-0.36
flow grad norm	0.43	0.65	0.62	0.87	1.00	0.76	0.10	0.11	0.39	0.39	-0.17
flow norm	0.25	0.39	0.60	0.66	0.76	1.00	0.09	0.11	0.41	0.54	-0.18
img grad norm	-0.12	-0.05	0.04	-0.08	0.10	0.09	1.00	0.94	0.17	-0.10	0.72
texture score	-0.08	-0.03	0.10	-0.01	0.11	0.11	0.94	1.00	0.19	-0.08	0.62
color change	0.21	0.23	0.33	0.29	0.39	0.41	0.17	0.19	1.00	0.21	0.07
param grad norm	0.48	0.49	0.39	0.53	0.39	0.54	-0.10	-0.08	0.21	1.00	-0.15
max corr vol	-0.08	-0.12	-0.32	-0.36	-0.17	-0.18	0.72	0.62	0.07	-0.15	1.00

(b) KITTI

Figure D.4: The correlation matrices of more active learning criteria with sample errors

Appendix E

Appendix for SemARFlow

E.1 Network Structure

Our detailed network structure is shown in Fig. E.1 and Fig. E.2. The dimension of each convolutional layer and each tensor in the network is marked in the figure.

In the semantic encoder, we have six levels of layer groups, and each layer group consists of two convolutional layers with leaky ReLU non-linearity. The layers in the same group have the same number of output channels marked in the figure. We use 3×3 convolutional kernels everywhere. The dilation value is set as 2 whenever we need to reduce the dimension by half.

The iterative decoder has a flow estimator, a context network, and a upmask net, which are shared across all levels. Some 1×1 convolutional layers are applied to transform the input features of different levels to features with the same number of channels so that they can be fed into the same shared flow estimator.

Our upmask net outputs a 144-channel mask for upsampling. We first unfold the 144 channels to 16 groups, each of which has 9 values. Since we are upsampling four times, each original value needs to correspond to 16 values in the output, and each output value is computed as a convex linear combinition of the 3×3 input window, so each group of 9 values in the mask are used as the coefficients here. We apply a softmax transform to make sure these 9 coefficients sum up to be one.

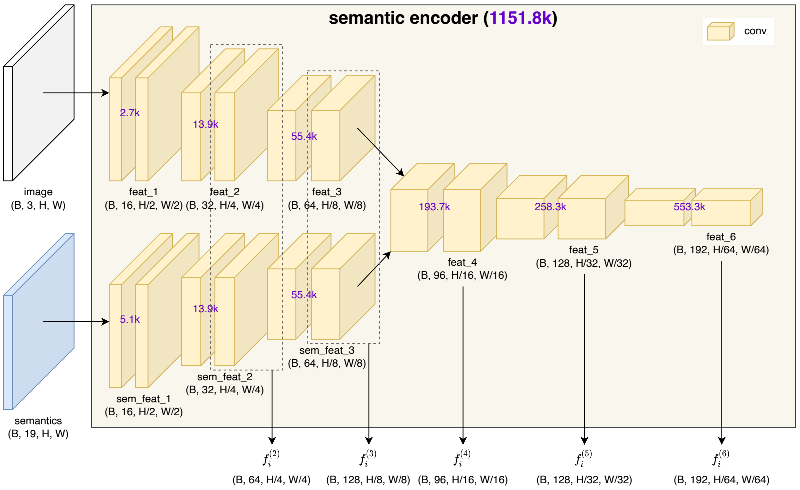


Figure E.1: Semantic encoder network structure. Purple numbers show the number of parameters.

E.2 Supplementary Results

E.2.1 Full Data Tables

We provide the full data table of all our experiments on the validation set in Tabs. E.1 to E.4. For test sets, we have submitted test results to the benchmark website, so please refer to the website for full evaluations.

E.2.2 More Qualitative Results

More qualitative results on the KITTI-2015 test set are shown in Fig. E.3.

E.2.3 Time Efficiency

Our network runs very efficiently thanks to its small size. Our network with semantic encoder and learned upsampler only has 2.6M parameters in total, so the model parameter size is only around 10MB.

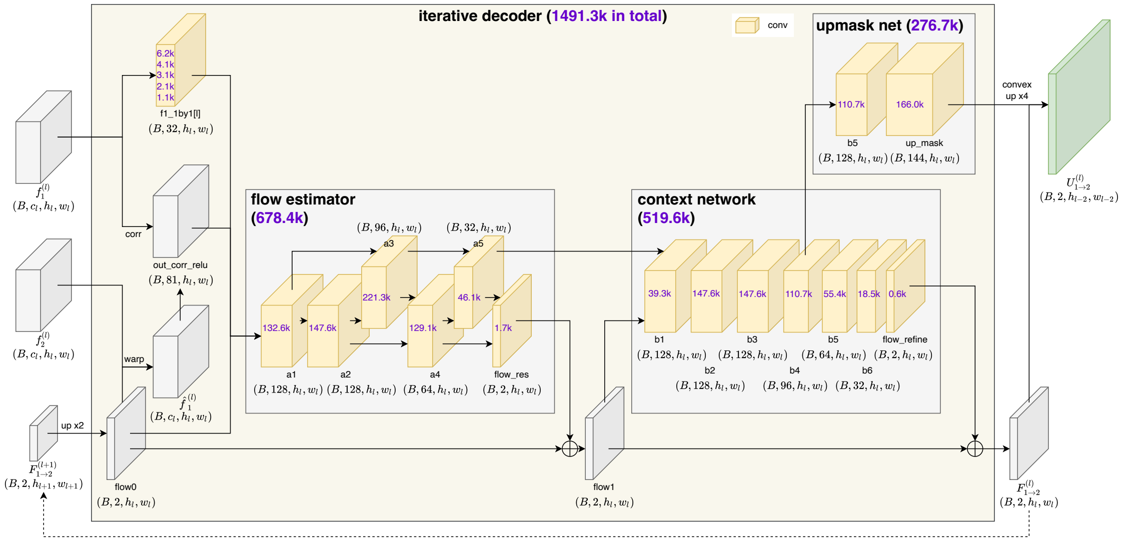


Figure E.2: Iterative decoder network structure. Purple numbers show the number of parameters.

Training We run 200k iterations in total. For our basic network with a semantic encoder and a learned upsampler, it takes around 44-48 hours on 2 NVIDIA GeForce RTX 2080 Ti GPUs. After adding semantic augmentation, it takes longer because we add a third forward pass of the network, but since we only use semantic augmentation for the last 50k iterations, the running time is still feasible: 54-58 hours on 2 NVIDIA GeForce RTX 2080 Ti GPUs.

Inference For inputs of size 256 × 832, inferring the forward flow of each sample takes 0.0168(±0.0005) second, *i.e.*, 60 frames per second, on a single NVIDIA GeForce RTX 2080 Ti GPU.

Tips on how to process semantic inputs efficiently We need to one-hot encode the semantic map input before feeding into the network. We do one-hot transformation *after* data augmentation to save time because otherwise, doing horizontal flip or bilinear interpolation for a 19-channel map is very time-consuming. We also avoid

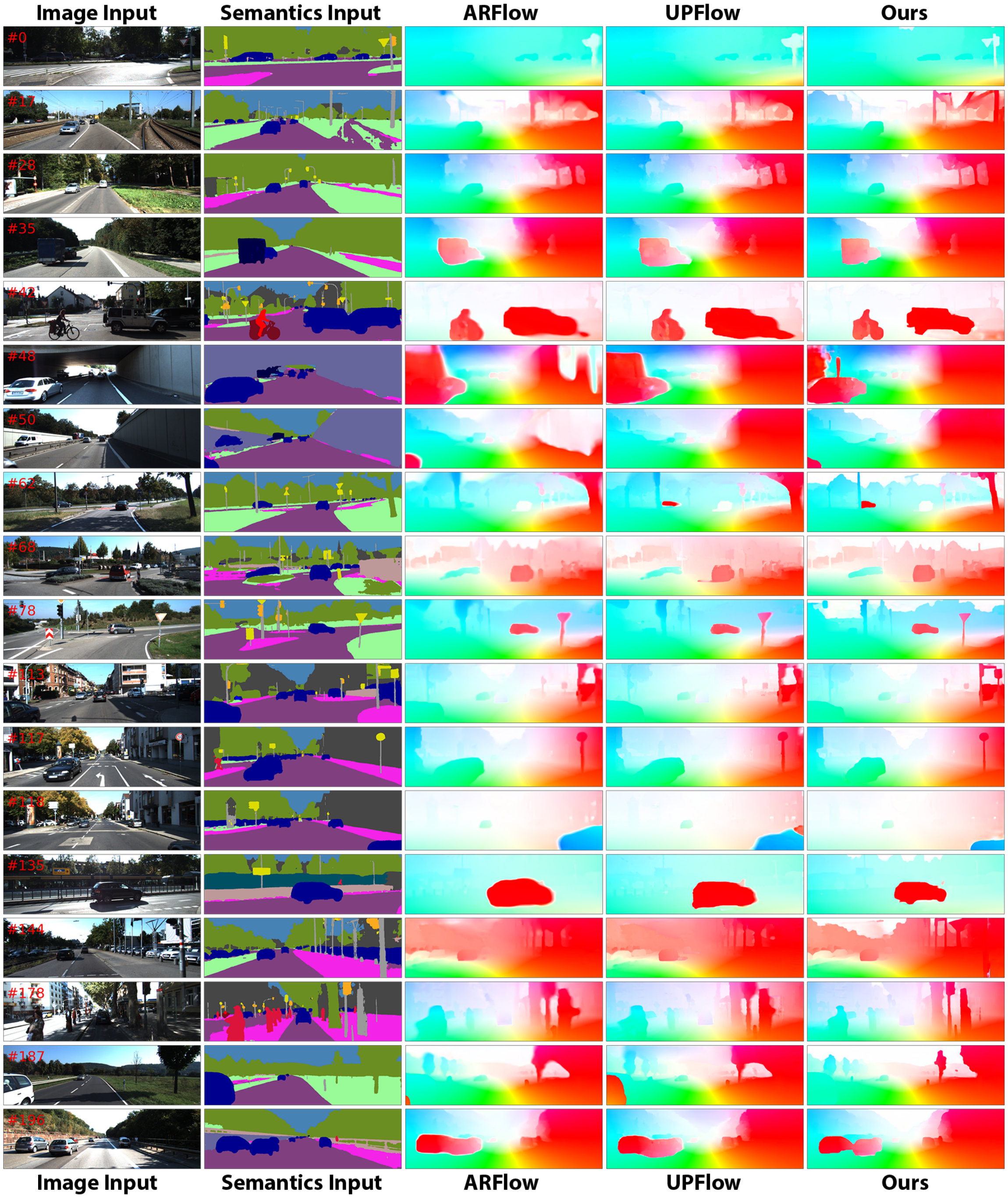


Figure E.3: More qualitative results from the KITTI-2015 test set. Sample IDs are shown on the top left corners of the images.

Table E.1: Full validation results for Table 1, 2, & 3 in the main paper (EPE/px and Fl/%).

up	no sm	enc	aug	KITTI-2015				
				Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
				10.360	8.528	2.901	2.068	6.967
		1		9.920	8.087	2.685	1.885	6.568
		2		9.830	7.951	2.646	1.857	6.396
		3		9.745	7.977	2.608	1.853	6.543
		4		9.745	7.951	2.644	1.852	6.553
✓				10.220	8.221	2.825	1.970	6.748
✓	✓			8.871	7.142	2.605	1.849	6.037
✓	✓	3		8.260	6.577	2.415	1.729	5.794
✓	✓		✓	8.801	6.748	2.484	1.595	6.642
✓	✓	3	✓	**7.788**	**5.963**	**2.179**	**1.399**	**5.635**

up	no sm	enc	aug	KITTI-2012				
				Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
				5.707	3.741	1.406	0.886	4.417
		1		5.484	3.606	1.354	0.861	4.205
		2		5.450	3.592	1.339	0.860	4.115
		3		5.359	3.511	1.328	0.850	4.098
		4		5.437	3.564	1.343	0.852	4.183
✓				5.728	3.725	1.420	0.888	4.495
✓	✓			5.314	3.374	1.386	0.876	4.347
✓	✓	3		4.964	3.111	1.291	0.825	**4.007**
✓	✓		✓	5.421	3.281	1.411	0.852	4.653
✓	✓	3	✓	**4.872**	**2.932**	**1.284**	**0.788**	4.175

flipping or rescaling these 19-channel semantic maps when we copy and paste occluder objects across samples for semantic augmentation.

Table E.2: Full validation results for Table 4 in the main paper (EPE/px and Fl/%).

Options of aug	KITTI-2015				
	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
Ours (final)	**7.788**	**5.963**	**2.179**	1.399	5.635
start from 100k	7.916	6.013	2.186	**1.396**	**5.406**
vehicles only	7.940	6.110	2.212	1.435	5.781
loss on new occ	8.149	6.105	2.272	1.418	5.620

Options of aug	KITTI-2012				
	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
Ours (final)	**4.872**	**2.932**	**1.284**	**0.788**	**4.175**
start from 100k	4.984	2.998	1.302	0.803	4.984
vehicles only	4.950	3.041	1.304	0.804	5.571
loss on new occ	5.167	3.083	1.361	0.810	5.892

Table E.3: Full data for Table 5 in the main paper. We show the relative improvements and reweighted contributions of all 19 classes.

	road	**car**	**terrain**	**vege.**	**sidewalk**	**bdg.**	**wall**	**pole**	**fence**	
Proportion	42.4%	17.6%	14.0%	11.6%	6.2%	4.1%	1.3%	1.1%	0.9%	
[1]	4.573	15.791	9.420	15.339	4.606	6.000	16.336	10.745	46.297	
Ours (final)	3.674	10.171	8.737	13.470	3.085	4.275	11.180	9.437	39.120	
Rel. impr.	19.7%	35.6%	7.3%	12.2%	33.0%	28.8%	31.6%	12.2%	15.5%	
Rew. contri.	19.0%	49.4%	4.8%	10.9%	4.7%	3.5%	3.4%	0.7%	3.3%	

	traf. sign	**truck**	**bicycle**	**traf. light**	**person**	**rider**	**motor.**	**sky**	**bus**	**train**
Proportion	0.4%	0.2%	0.1%	0.1%	0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%
[1]	5.254	10.461	5.421	2.290	3.852	20.811	13.385	28.547	17.179	12.564
Ours (final)	4.681	9.526	4.945	2.060	2.627	19.581	3.620	38.521	17.747	9.615
Rel. impr.	10.9%	8.9%	8.8%	10.0%	31.8%	5.9%	73.0%	-34.9%	-3.3%	23.5%
Rew. contri.	0.1%	0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%

Table E.4: Full validation results for Table 6 in the main paper (EPE/px and Fl/%). † denotes models with semantic inputs.

Method	KITTI-2015					KITTI-2012				
	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ	Fl_all	Fl_noc	EPE_all	EPE_noc	EPE_occ
ARFlow (our impl.)	13.210	10.190	4.081	2.878	9.398	7.360	4.434	1.713	0.994	5.806
Ours (baseline)	12.270	8.642	3.809	2.433	9.907	7.165	4.104	1.730	0.997	5.915
Ours (+enc)†	11.280	7.749	3.327	2.121	8.749	6.583	3.596	1.561	0.912	5.267
Ours (+enc +aug)†	**10.320**	**6.950**	**2.640**	**1.558**	**7.121**	**6.204**	**3.488**	**1.489**	**0.855**	**5.150**

E.3 Other Explorations

We have also explored many other methods to apply semantics in the unsupervised optical flow network. Although these trials are not very successful and thus are not proposed in our final model, we briefly discuss our findings for the readers' reference.

E.3.1 Adding Semantic Inputs to the Decoder

We also tried adding semantic inputs to the decoder. This is because the current semantic maps are used as encoder inputs, which are somewhat distant from our final output, so we were wondering whether adding semantic cues to the decoder directly could help it decode a better flow.

We used two shallow convolutional layers to extract a feature map from semantic input and downsample that feature to different resolutions for different levels of the iterative decoder. We found that such direct injection of semantic input into the decoder did improve the vanilla ARFlow without semantic encoder. However, it helped little for our adapted ARFlow model *with* semantic encoder.

E.3.2 Adding a Semantic Consistency Loss

As applied in some previous work [107], the semantic consistency loss enforces the output correspondence to have consistent semantic classes. This is similar to the photometric loss, but we use the output flow to warp the semantic inputs instead of the image inputs.

After experimenting this semantic consistency loss, unfortunately, we did not find it very helpful to our network. There are mainly two reasons.

Firstly, unlike the image input, which consists of roughly continuous RGB values, the semantic input is categorical, and it is common to have large areas of the same

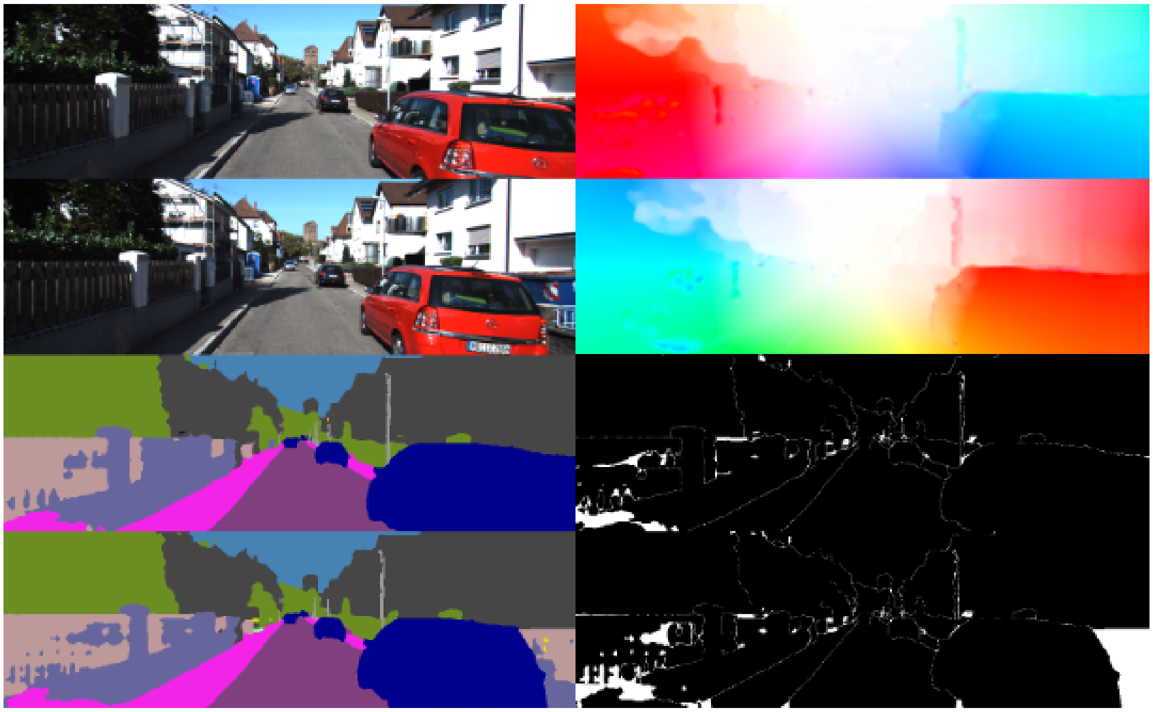


Figure E.4: Left: inputs; top right: forward and backward flow; bottom right: forward and backward semantic consistency loss

semantic class. Thus, one may understand the semantic map as a highly textureless channel of input, which cannot help differentiate different regions of the same semantic class. Also, since a large area has the same semantic class, tuning flow in that region makes no difference to the semantic consistency loss, which means the gradient will be zero for most pixels except for those near the semantic boundaries.

One solution to the aforementioned problem is to use a continuous semantic class distribution as input. For example, we can use the softmax values from the semantic segmentation network as our semantic input. However, as we mentioned earlier in Appendix E.2.3, doing augmentations on a 19-channel semantic input is very time-consuming.

Secondly, semantic consistency loss also does not work on occlusion regions, where photometric loss has issues, so we have to mask out the occlusion regions for both losses. For non-occluded regions, the current photometric loss is already good enough to find semantically consistent output by itself, so semantic consistency does not add much here. As illustrated in Fig. E.4, we trained a model with only photometric loss for only 50k iterations, and most part of the frame is already very consistent on semantics. The inconsistent parts are mostly either on semantic boundaries or in the occlusion region.

E.3.3 Using Semantic Boundaries for the Boundary-Aware Smoothness Loss

Most previous methods use smoothness loss to constrain a smooth flow output. However, motion is not smooth across motion boundaries, where motion changes abruptly. Motion boundaries usually coincide with object boundaries, so object boundaries can be a good approximation to indicate where smoothness loss should not be imposed.

Due to lack of semantic information, current methods use image edges instead to generate a weight map to reweigh smoothness loss at different pixels. In our case, since semantic maps are available, we use this information to create much clearer object boundaries.

We start from the same smoothness loss in ARFlow [1]. As visualized in Fig. E.5, the weights based on image edges are computed as the sum of the second-order image derivatives on both x and y-axis, *i.e.,* , the Laplacian of the 2D optical flow field. In comparison, our semantic boundaries are much cleaner.

However, both image edges and semantic boundaries have issues. Image edges usually provide boundaries within the same object, such as the edge of the shadow on the road, and they also have fewer boundaries in the dark region where image values are similar. Meanwhile, semantic boundaries are computed from semantic segmentation, where different instances of the same semantic class are not differentiated. This causes big issues when, for example, the semantic map of multiple cars merge into one.

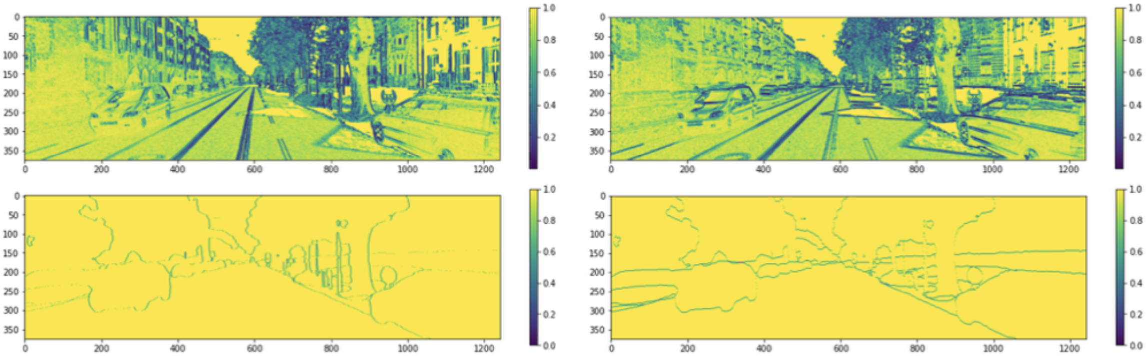


Figure E.5: Top: weights based on image edges; bottom: weights based on semantic boundaries

To fix these issues, we attempted to use a combination of both image edges and semantic boundaries. We find image edges in the vehicle (car, truck, bus, train), people (person, rider), and small vehicle (motorcycle, bicycle) regions, combined with semantic boundaries else where, as shown in Fig. E.6.

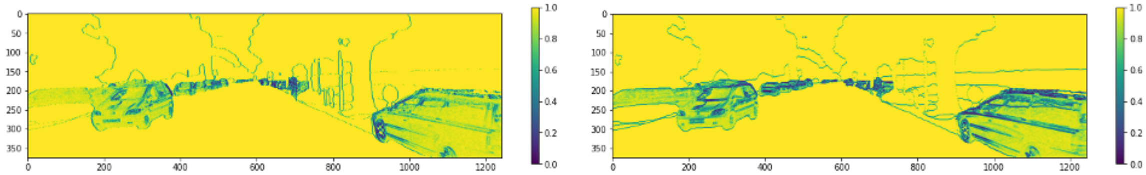


Figure E.6: The combined boundary weight

We tried all these boundaries and tuned the weight of the smoothness loss term in the total loss, and found little difference in the evaluated results. Moreover, after we add the learned upampler module in the network, applying smoothness loss is doing more harm than help. This is understandable because the goal of the learned upsampler is to make the network decide which part of the flow output should be smooth and where should not be smooth. Smoothness loss imposes a preference that the motion should be smooth in the form of zero second-order derivatives, which is not data-driven and may not be precise for real-world motion fields. Therefore, we ended up turning off the smoothness loss in our final model.

E.3.4 Learning the Initial Flow in the Decoder

Driving scenes usually have similar scene layouts (sky is on the top, and road is on the bottom, *etc.*). In addition, the motion of the camera is also mostly moving forward with some occasional slight rotations. These two effects together create a *looming* motion, where most objects are moving closer to the camera. To explain in 2D image frames, the left part of the image tends to move left, and the right part tends to move right. The lower part (mostly roads and sidewalks) also tends to move downwards until they are out of sight. These observations indicate a strong motion prior knowledge that can be used to better initialize our flow estimate.

The current iterative decoder as in ARFlow [1] initialize the flow estimate as zero motion, which gets refined later iteration by iteration. However, we can apply the looming motion prior instead by parameterizing the initial flow using some learnable parameters. Specifically, for a 256×832 input, the highest (6th) level feature has dimension 4×13, so we use a learnable $19 \times 4 \times 13 \times 2$ tensor as the prior. We condition the motion prior on the 19 semantic classes, so that we can refer to the semantic map input to generate its initial flow prior based on semantics. Note that we define the prior specifically forward flow only, and we need to flip the prior when computing backward flow in our network.

We trained the network with learned initial flow conditioned on semantics, and the learned prior is visualized in Fig. E.7. Overall, we can see that the looming motion pattern is learned by our network. However, for each semantic class, the network can only learn prior for places where that class frequently appears. For instance, the

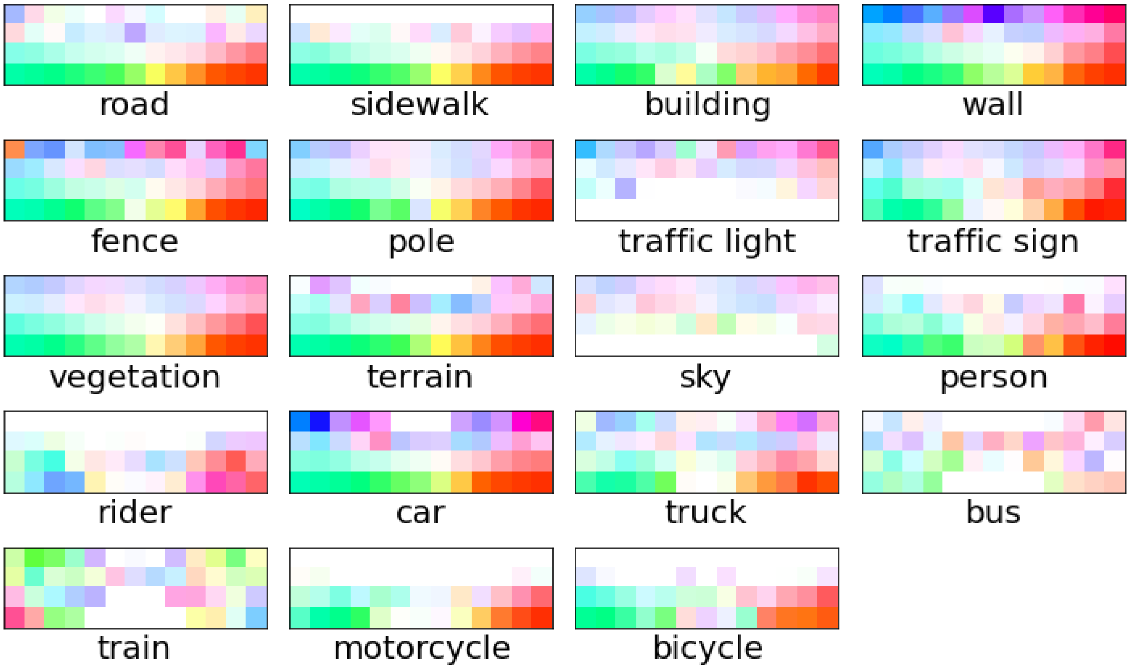


Figure E.7: The learned init flow prior for each semantic class

upper part of the road prior is not learned well because there are few road pixels on the top of the frame.

To better fix this issue, we try to parameterize the motion prior by its four corners. Specifically, we learn a 2×2 prior for each class, and bilinear interpolate this 2×2 prior to 4×13 before using it to construct initial flow based on semantics. The results are then visualized in Fig. E.8. The network has learned a very smooth and more or less similar pattern prior for each class.

In terms of evaluation metrics, both methods improve slightly by themselves, but we found the improvements became negligible after we apply the semantic segmentation module. One question here is whether good initialization matters a lot for our unsupervised flow networks. Since we refine the flow by many iterations in the decoder, the initial estimate may not change the results significantly, if the following refinement units are effective enough.

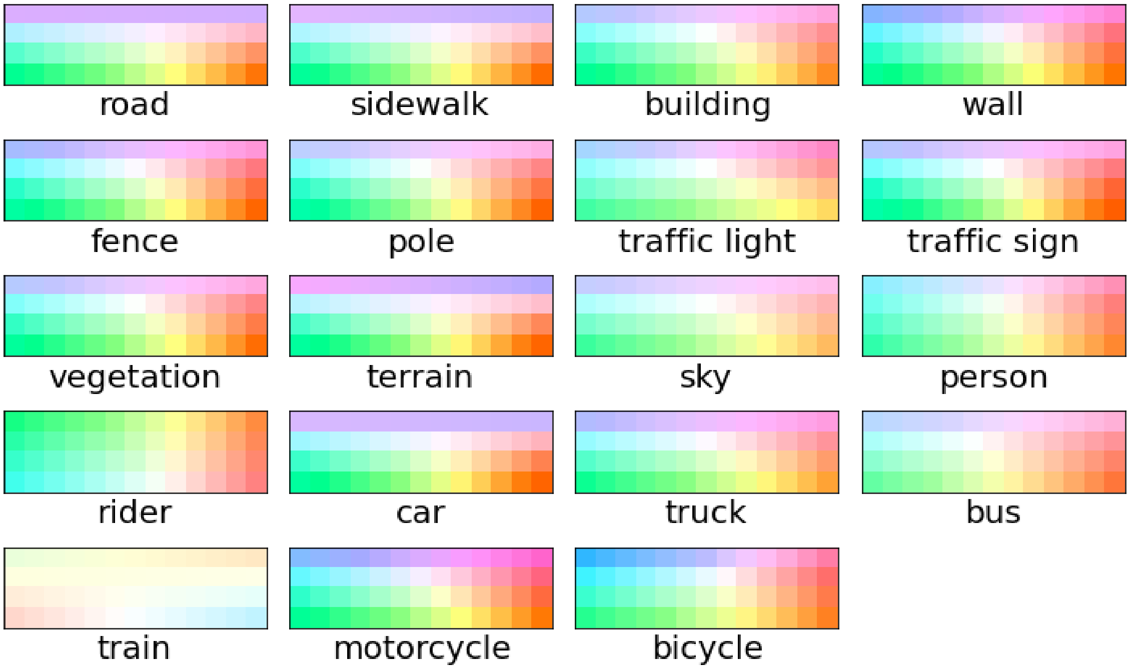


Figure E.8: The learned init flow prior for each semantic class if we parameterize by coners

E.3.5 Reweighing Losses at Different Semantic Regions

Current models usually have different scales of error at different semantic regions. Semantic classes such as car are harder to track and thus incur larger errors than other classes. Also, some classes, such as car and person, are practically more important for autonomous driving applications. Therefore, we tried to reweigh the photometric loss by their semantic class. We give higher weights to classes like car and person so that the network can focus on improving those classes more.

However, the results are hard to evaluate numerically. The current ground-truth labels are mostly concentrated on the lower part of the frame, so many practically important objects such as traffic lights, traffic signs and poles only account for a very small amount of the evaluation. Also, classes like person, rider, and bicycles do not have flow labels because reliable CAD models are not available for these dynamic objects. Based on these issues, finding a better evaluation method may be more important.

E.3.6 Using the Epipolar Constraint to Post-process the Static Region Flow

Following earlier traditional methods [106, 108], we also explored the possibility of using the static scene epipolar constraint to post-process static optical flow. Since most part of the frame is static, we can use our correspondences found to estimate the fundamental matrix between two frames, and then use epipolar constraints to refine our flow.

This method is mostly targeted on refining optical flow for those occluded static pixels. For example, a part of the road or background may be occluded by the moving cars, or they may simply move out of the frame, so their correspondence is not visible in the other frame. However, we still want the network to have a best "guess" on where the correspondence is. Most current methods rely on smoothness to generate those "guesses". However, given the epipolar constraint, we limit the searching range of correspondence to one epipolar line, which may help us "guess" more informatively. Our semantic map inputs tell us where those static region is, so we can get a more reliable fundamental matrix estimate.

Estimating the fundamental matrix only requires eight pairs of corresponding points, so we can select only the most reliable correspondences for this computation. Specifically, we define "reliable correspondence" based on three criteria: (1) not in the occlusion region, (2) not on the semantic boundary or image borders, and (3) not from any (possibly) dynamic semantic class or any texture-less semantic class.

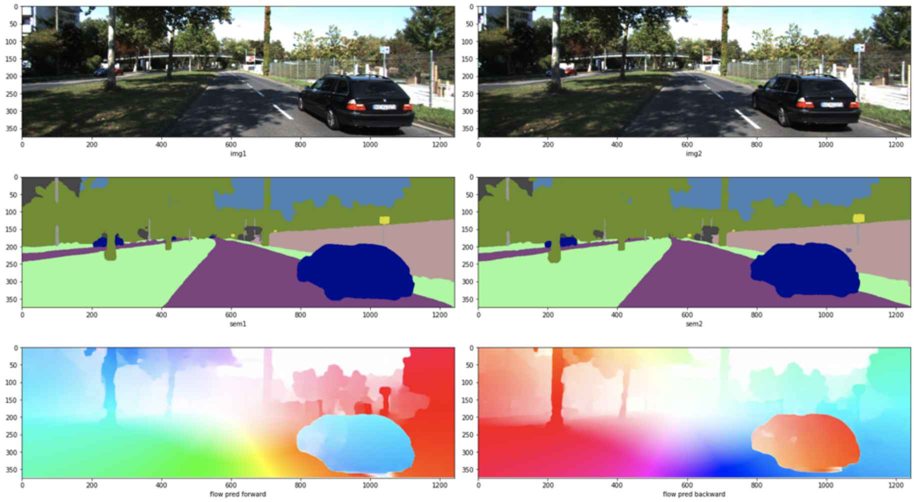
We first compute the occlusion mask through forward-backward consistency check. Then, we find semantic boundaries based on our semantic map input and define all pixels that are ≤ 5 pixels away from any boundary as the boundary pixels. For reliable static semantic classes, we include all classes except vehicles (car, truck, bus, train), people (person, rider), small vehicles (bicycle, motorcycle), and sky (poor texture). One example is shown in Figs. E.9(a) and E.9(b).

After computing the reliable static regions, we use both forward and backward flow in those regions to create a set of correspondences between two input frames. We then estimate the fundamental matrix $\hat{F}$ using RANSAC. For a typical sample in the KITTI-2015 train set, RANSAC based on correspondeces in our reliable regions mostly produces >98% inlier rate.

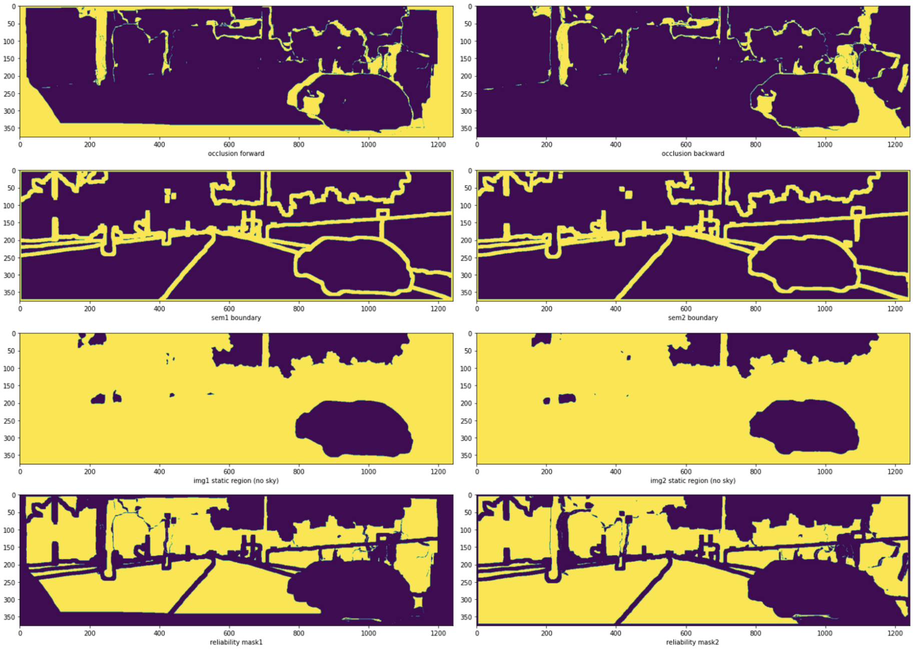
We then find the correspondences in static regions and check whether they conform to epipolar constraint. For each point $\boldsymbol{p}$ in the static region of the first frame, we compute the epipolar error

$$\epsilon(\boldsymbol{p}) = \left((\boldsymbol{p} + U_{1\to2}(\boldsymbol{p}))^T F \boldsymbol{p}\right)^2,$$

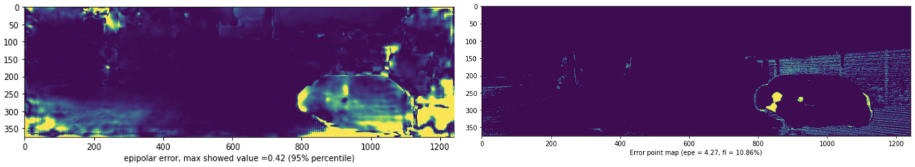
where $U_{1\to2}(\boldsymbol{p})$ is the input flow estimate. We find the pixels with top 5% epipolar error and refine those by projecting their current estimated correspondences onto their epipolar lines. As shown in Fig. E.9(c), our epipolar error successfully detects the part of the frame that has high EPE errors. However, we tested this method on the KITTI-2015 train set but did not see much improvements on the evaluation results. The reason is still under investigation.



(a) Inputs for post-processing



(b) Computing reliable static region masks



(c) Comparing epipolar errors and true errors

Figure E.9: An example for semantic post-processing

Appendix F

Appendix for UFD-PRiME

F.1 Method Details

F.1.1 Stage 1

Detailed Structures The detailed network structure of our encoder, flow decoder, and disparity decoder are shown in Figs. F.1(a) to F.1(c). The dimension of each intermediate tensor and the number of parameters are noted in the figures.

Redundant Dimension in Disparity To reuse modules across flow and disparity decoders, we need to make sure their input sizes to the same module are equal. Therefore, in our implementation, our disparity is estimated as a 2D vector. In the disparity estimator module, which is not shared, the output dimension is set as 1, so we only change the x-dimension of the disparity estimate. For other shared modules, we use 2D disparities as both input and output. In the end, we drop the y-dimension of our 2D disparity estimate to make sure our final disparity is strictly 1D.

Augmentation as Self-Supervision Our network is adapted from ARFlow [1], which has an "augmentation as self-supervision" module. In ARFlow, after each forward pass of the current sample, they apply some random transformations, including appearance transformations (brightness, contrast, hue, saturation, gaussian blur, *etc.*), spatial transformations (random affine), and occlusion transformations (random cropping), to generate an augmented sample as well as its pseudo-label. Then, they do a second forward pass using the augmented sample and self-supervise the output using the generated pseudo-label. This self-supervision module helps learn flow at occlusion regions and enhance the consistency of optical flow prediction.

We keep their augmentation as self-supervision module in training, and we apply the same transformations to all three input images, I_{L1}, I_{R1}, I_{L2}, at the same time to simulate a realistic transformation to the moving stereo camera rig. However, we have to turn off the random rotation part in the random affine transformation to ensure the left and right views are on the same horizontal line. We assume 1D horizontal search for disparities, and if rotations are applied, the true disparity will no longer be 1D, which affects the effectiveness of our disparity decoder.

F.1.2 Stage 2

Computing Occlusion Masks We first use the Stage-1 model to generate all the flow and disparities needed for Stage-2 training. In addition to the regular outputs $F_{L1\to L2}, D_{L1\to R1}$ shown in the Stage-1 network, we also compute all other flows and disparities (including backward ones) among the four input images $I_{L1}, I_{R1}, I_{L2}, I_{R2}$, as enumerated in Fig. F.2. This can be done by switching different inputs to the Stage-1 network. For example, we can compute $F_{L2\to L1}, D_{L2\to R2}$ by feeding I_{L2}, I_{R2}, I_{L1} to the network. Note that if we swap the left and right view, we also need to do a horizontal flip to all input images to ensure the disparity is negative. We can then horizontally flip the disparity output again to recover the original disparity.

With all flows and disparities estimated, we compute all occlusion masks through forward-backward consistency check [27] as follows. Suppose F_f and F_b are the forward and backward pair of flows (or disparities) and $\boldsymbol{x}$ is a point in the frame. We flag the forward occlusion mask as 1 on $\boldsymbol{x}$ whenever the following constraint holds.

$$\begin{aligned}&\|F_f(\boldsymbol{x}) + F_b(\boldsymbol{x} + F_f(\boldsymbol{x}))\|_2^2 \\ &< \alpha_1 \left(\|F_f(\boldsymbol{x})\|_2^2 + \|F_b(\boldsymbol{x} + F_f(\boldsymbol{x}))\|_2^2\right) + \alpha_2,\end{aligned}$$

where we use hyper-parameters $\alpha_1 = 0.01$, $\alpha_2 = 0.5$ as in [27]. We compute occlusion masks for every forward and backward flow and disparities computed and store them on the disk for later use.

Generating Scene Flow Pseudo-labels for RAFT-3D The RAFT-3D [176] network also requires a disparity channel input to become RGBD inputs. We can simply use the disparities estimated by the Stage-1 model for that.

In loss computation, RAFT-3D requires scene flow pseudo-labels, which include the regular 2D optical flow, as well as a disparity change value to indicate the motion on the depth dimension. For the former, we can simply use the previous flow estimate. For the latter, we need to warp previous disparities. For example, to compute scene flow pseudo-label between I_{L1} and I_{L2}, for each point $\boldsymbol{x}$, we warp $D_{L2\to R2}$ and compute the disparity change pseudo-label as follows.

$$\zeta_{L1\to L2}(\boldsymbol{x}) = D_{L2\to R2}\left(\boldsymbol{x} + F_{L1\to L2}(\boldsymbol{x})\right) - D_{L1\to R1}(\boldsymbol{x})$$

We mask out the unreliable regions of both scene flow estimates and pseudo-labels when we compute loss. Since our flow in Stage 2 is reconstructed from disparity $D_{L1\to R1}$, the unreliable region of the flow estimate is simply the estimated occlusion region of $D_{L1\to R1}$. The unreliable region of pseudo-label is generated as a union of the occlusion regions of $F_{L1\to L2}$, $D_{L1\to R1}$, and the warped $D_{L2\to R2}$.

A Smaller Context Network Our Stage-2 network is exactly the same as RAFT-3D [176] except for one change on the context network. The original RAFT-3D uses a very large context network copied from FPN [247] so that they can use semantically pre-trained weights to help better distinguish objects. In our experiment, we found empirically that using a much smaller context network such as the same structure of the feature encoder network still gives similar results. Therefore, we stick to the smaller context network to save memory.

F.1.3 Stage 3

Depth Refinement Continuing the derivation in the main paper, we need to solve the following over-determined linear system

$$\left(-\frac{1}{\hat{D}_1}+\frac{1}{\hat{D}_1^2}\delta_1\right)\boldsymbol{\alpha}_1+\left(-\frac{1}{\hat{D}_2}+\frac{1}{\hat{D}_2^2}\delta_2\right)\boldsymbol{\alpha}_2=\boldsymbol{t},$$

which can be rewritten as

$$\delta_1\frac{\boldsymbol{\alpha}_1}{\hat{D}_1^2}+\delta_2\frac{\boldsymbol{\alpha}_2}{\hat{D}_2^2}=\frac{\boldsymbol{t}}{f_x b}+\frac{\boldsymbol{\alpha}_1}{\hat{D}_1}+\frac{\boldsymbol{\alpha}_2}{\hat{D}_2}.$$

Denoting $\boldsymbol{\beta}_1=\frac{\boldsymbol{\alpha}_1}{\hat{D}_1^2}$, $\boldsymbol{\beta}_2=\frac{\boldsymbol{\alpha}_2}{\hat{D}_2^2}$, $\boldsymbol{\gamma}=\frac{\boldsymbol{t}}{f_x b}+\frac{\boldsymbol{\alpha}_1}{\hat{D}_1}+\frac{\boldsymbol{\alpha}_2}{\hat{D}_2}$, we then have

$$[\boldsymbol{\beta}_1 \mid \boldsymbol{\beta}_2]\begin{bmatrix}\delta_1\\ \delta_2\end{bmatrix}=\boldsymbol{\gamma}.$$

This over-determined linear system can be approximately solved based on least squared error criterion as

$$\begin{bmatrix}\delta_1^*\\ \delta_2^*\end{bmatrix}=(B^TB)^{-1}B^T\boldsymbol{\gamma},$$

where $B=[\boldsymbol{\beta}_1 \mid \boldsymbol{\beta}_2]\in\mathbb{R}^{3\times 2}$.

We linearize this equation here instead of solving it in the depth domain (which could be more straightforward) because solving this type of linear systems could be very sensitive in the depth domain. For example, when the 3D motion is along the moving direction of the camera, we have the projection rays from the two consecutive frames almost in parallel, so the B matrix here will be close to rank 1. To avoid this issue, we discard the refinement if B^TB is close to singular (decided by the opencv package automatically).

F.2 Experiment Details

F.2.1 Screenshots of Benchmark Results

We show the benchmark test screenshots for all our three stages from the KITTI [33] website in Figs. F.3 to F.5 to prove that our results are official. Our final-stage result entry (currently anonymous) is also shown on the official website now for your reference.

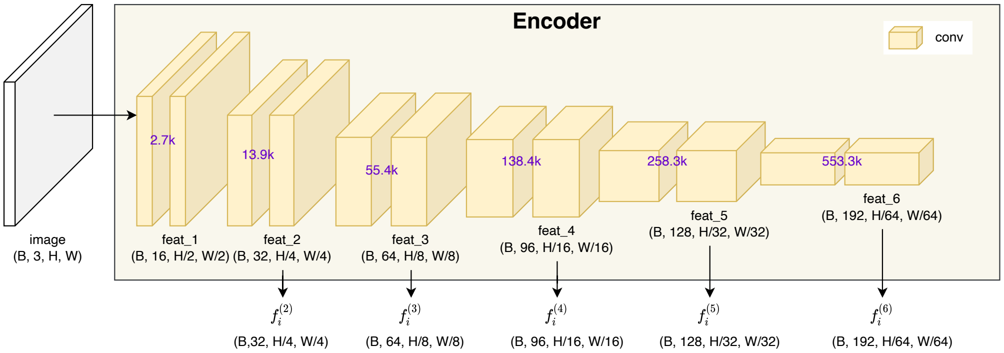
F.2.2 More Qualitative Examples

Some more qualitative examples from the KITTI-2015 [33] test set are shown in Fig. F.6.

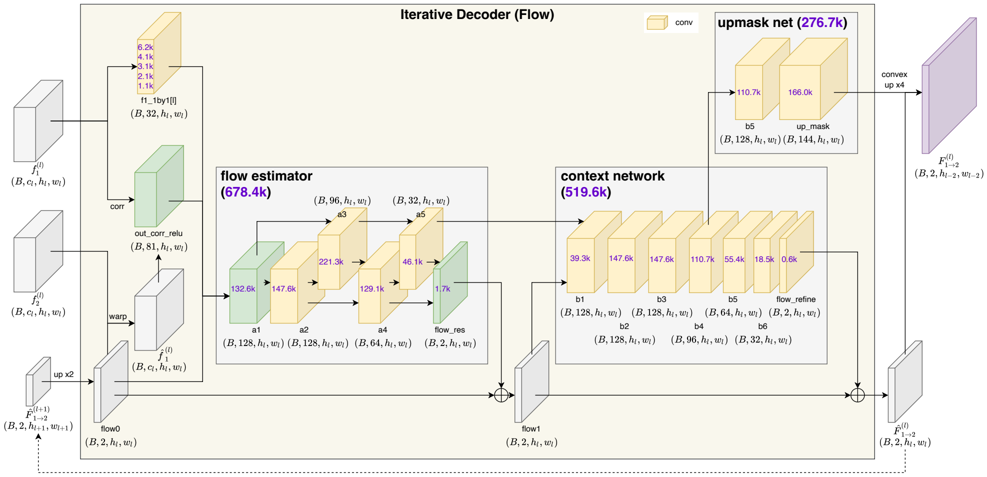
F.2.3 Visualizing Rigid Motions Segmentation

Our rigid motion also implies a soft segmentation of rigid bodies that we can refer to. This can be computed using traditional clustering algorithms based on our 6-DoF map. However, the selection of clustering method also makes a difference here. It may also be helpful if we could have object-level information to regularize our map segmentation. Thus, we leave this for future research.

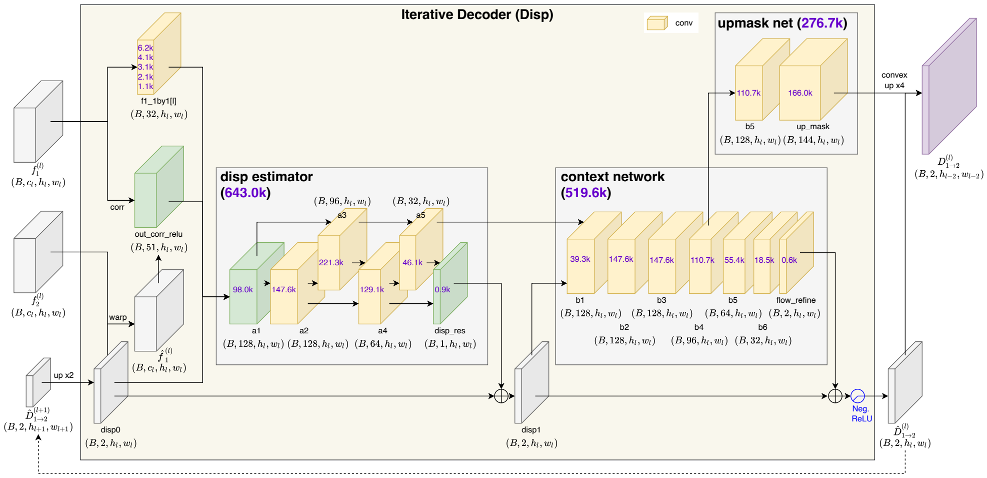
Alternatively, we can also visualize the map in RGB through PCA. For cases with exactly 3 distinct rigid motions, we can see the segmentation very clearly after normalizing the first three principal component channel-wise to between 0 and 1, as shown in the last figure in the main paper. For most cases that do not have at least three distinct rigid motions, normalizing the top 3 principal components channel-wise may magnify the small changes represented by the second or third color channel. In that case, we suggest normalizing all three channels together. One example of that is shown in our Fig. 1 in the main paper.



(a) Encoder structure (Stage 1)



(b) Flow decoder structure (Stage 1)

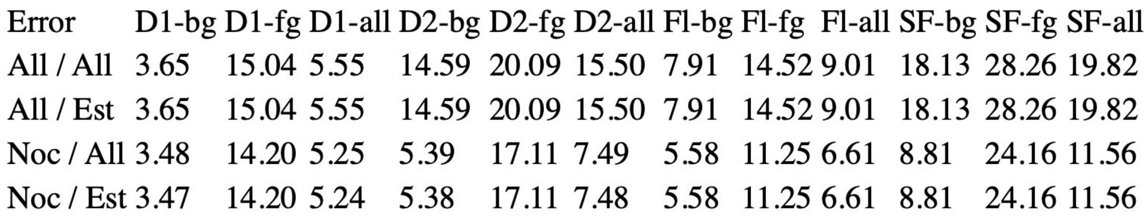


(c) Disp decoder structure (Stage 1)

Figure F.1: Stage-1 model detailed structures

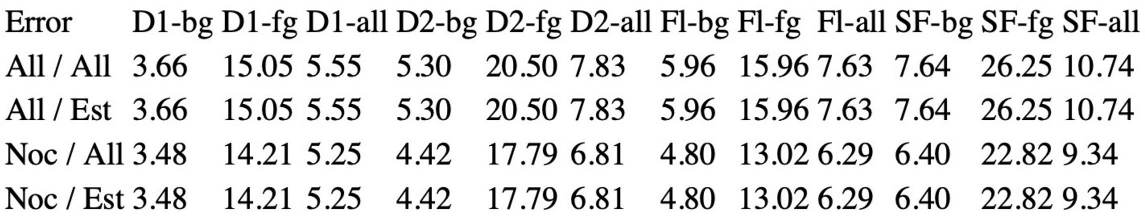
$$\begin{tikzcd}[column sep=huge, row sep=huge]
I_{L1} \arrow[r, shift left, "D_{L1\to R1}"] \arrow[d, shift left, "F_{L1\to L2}"] & I_{R1} \arrow[l, shift left, "D_{R1\to L1}"] \arrow[d, shift left, "F_{R1\to R2}"] \\
I_{L2} \arrow[u, shift left, "F_{L2\to L1}"] \arrow[r, shift left, "D_{L2\to R2}"] & I_{R2} \arrow[u, shift left, "F_{R2\to R1}"] \arrow[l, shift left, "D_{R2\to L2}"]
\end{tikzcd}$$

Figure F.2: All flows and disparities computed



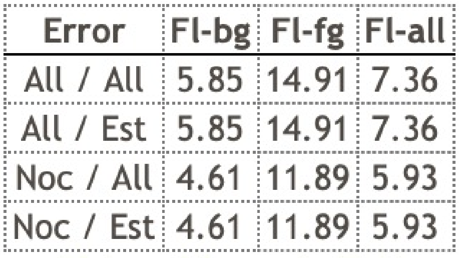
Error	D1-bg	D1-fg	D1-all	D2-bg	D2-fg	D2-all	Fl-bg	Fl-fg	Fl-all	SF-bg	SF-fg	SF-all
All / All	3.65	15.04	5.55	14.59	20.09	15.50	7.91	14.52	9.01	18.13	28.26	19.82
All / Est	3.65	15.04	5.55	14.59	20.09	15.50	7.91	14.52	9.01	18.13	28.26	19.82
Noc / All	3.48	14.20	5.25	5.39	17.11	7.49	5.58	11.25	6.61	8.81	24.16	11.56
Noc / Est	3.47	14.20	5.24	5.38	17.11	7.48	5.58	11.25	6.61	8.81	24.16	11.56

Figure F.3: Stage-1 benchmark test result screenshot



Error	D1-bg	D1-fg	D1-all	D2-bg	D2-fg	D2-all	Fl-bg	Fl-fg	Fl-all	SF-bg	SF-fg	SF-all
All / All	3.66	15.05	5.55	5.30	20.50	7.83	5.96	15.96	7.63	7.64	26.25	10.74
All / Est	3.66	15.05	5.55	5.30	20.50	7.83	5.96	15.96	7.63	7.64	26.25	10.74
Noc / All	3.48	14.21	5.25	4.42	17.79	6.81	4.80	13.02	6.29	6.40	22.82	9.34
Noc / Est	3.48	14.21	5.25	4.42	17.79	6.81	4.80	13.02	6.29	6.40	22.82	9.34

Figure F.4: Stage-2 benchmark test result screenshot



Error	Fl-bg	Fl-fg	Fl-all
All / All	5.85	14.91	7.36
All / Est	5.85	14.91	7.36
Noc / All	4.61	11.89	5.93
Noc / Est	4.61	11.89	5.93

Figure F.5: Stage-3 benchmark test result screenshot

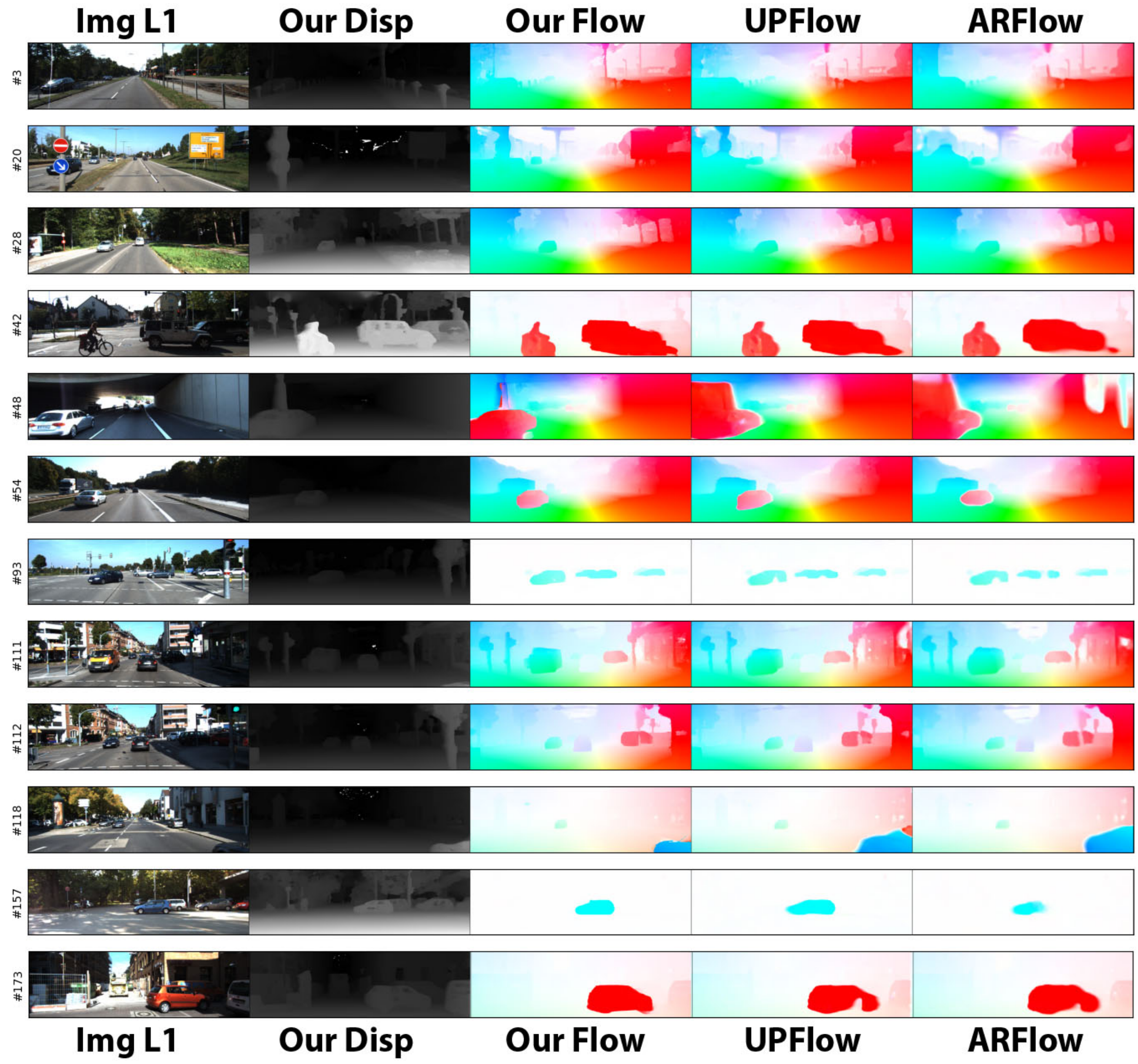


Figure F.6: More qualitative examples

Bibliography

[1] L. Liu, J. Zhang, R. He, Y. Liu, Y. Wang, Y. Tai, D. Luo, C. Wang, J. Li, and F. Huang, "Learning by analogy: Reliable supervision from transformations for unsupervised optical flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6489–6498, 2020.

[2] K. Luo, C. Wang, S. Liu, H. Fan, J. Wang, and J. Sun, "Upflow: Upsampling pyramid for unsupervised optical flow learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1045–1054, 2021.

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, IEEE, 2012.

[4] E. Bastug, M. Bennis, M. Médard, and M. Debbah, "Toward interconnected virtual reality: Opportunities, challenges, and enablers," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 110–117, 2017.

[5] X. Dong, M. A. Garratt, S. G. Anavatti, and H. A. Abbass, "Towards real-time monocular depth estimation for robotics: A survey," *IEEE Trans. Intell. Transport. Sys.*, vol. 23, no. 10, pp. 16940–16961, 2022.

[6] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems*, pp. 568–576, 2014.

[7] S. Yu, G. Wu, C. Gu, and M. E. Fathy, "Tdt: Teaching detectors to track without fully annotated videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3940–3950, 2022.

[8] V. Ye, Z. Li, R. Tucker, A. Kanazawa, and N. Snavely, "Deformable sprites for unsupervised video decomposition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2657–2666, 2022.

[9] H. H. Kim, S. Yu, S. Yuan, and C. Tomasi, "Cross-attention transformer for video interpolation," in *Proceedings of the Asian Conference on Computer Vision Workshops*, pp. 320–337, 2022.

[10] C. Gao, A. Saraf, J.-B. Huang, and J. Kopf, "Flow-edge guided video completion," in *European Conference on Computer Vision*, pp. 713–729, Springer, 2020.

[11] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *International Joint Conference on Artificial Intelligence*, vol. 2, pp. 674–679, 1981.

[12] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[13] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: a factorization method," *Proceedings of the National Academy of Sciences*, vol. 90, no. 21, pp. 9795–9802, 1993.

[14] T. Brox, C. Bregler, and J. Malik, "Large displacement optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 41–48, IEEE, 2009.

[15] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Epicflow: Edge-preserving interpolation of correspondences for optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1164–1172, 2015.

[16] D. A. Forsyth and J. Ponce, *Computer vision: a modern approach.* prentice hall professional technical reference, 2002.

[17] S. Jianbo and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[19] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2758–2766, 2015.

[20] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8934–8943, 2018.

[21] J. Hur and S. Roth, "Iterative residual refinement for joint optical flow and occlusion estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5754–5763, 2019.

[22] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*, pp. 402–419, Springer, 2020.

[23] F. Zhang, O. J. Woodford, V. A. Prisacariu, and P. H. Torr, "Separable flow: Learning motion cost volumes for optical flow estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10807–10817, 2021.

[24] S. Jiang, D. Campbell, Y. Lu, H. Li, and R. Hartley, "Learning to estimate hidden motions with global motion aggregation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9772–9781, 2021.

[25] Z. Huang, X. Shi, C. Zhang, Q. Wang, K. C. Cheung, H. Qin, J. Dai, and H. Li, "Flowformer: A transformer architecture for optical flow," in *European Conference on Computer Vision*, pp. 668–685, Springer, 2022.

[26] X. Shi, Z. Huang, D. Li, M. Zhang, K. C. Cheung, S. See, H. Qin, J. Dai, and H. Li, "Flowformer++: Masked cost volume autoencoding for pretraining optical flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1599–1610, 2023.

[27] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[28] P. Liu, M. Lyu, I. King, and J. Xu, "Selflow: Self-supervised learning of optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4571–4580, 2019.

[29] A. Stone, D. Maurer, A. Ayvaci, A. Angelova, and R. Jonschkowski, "Smurf: Self-teaching multi-frame unsupervised raft with full-image warping," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3887–3896, 2021.

[30] S. Yuan, X. Sun, H. Kim, S. Yu, and C. Tomasi, "Optical flow training under limited label budget via active learning," in *European Conference on Computer Vision*, pp. 410–427, Springer Nature Switzerland, 2022.

[31] L. Fan, W. Huang, C. Gan, S. Ermon, B. Gong, and J. Huang, "End-to-end learning of motion representation for video understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6016–6025, 2018.

[32] S. Aslani and H. Mahdavi-Nasab, "Optical flow based moving object detection and tracking for traffic surveillance," *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, vol. 7, no. 9, pp. 1252–1256, 2013.

[33] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[35] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4161–4170, 2017.

[36] T.-W. Hui, X. Tang, and C. Change Loy, "Liteflownet: A lightweight convolutional neural network for optical flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8981–8989, 2018.

[37] J. Y. Jason, A. W. Harley, and K. G. Derpanis, "Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness," in *European Conference on Computer Vision*, pp. 3–10, Springer, 2016.

[38] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, and H. Zha, "Unsupervised deep learning for optical flow estimation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[39] J. Janai, F. Guney, A. Ranjan, M. Black, and A. Geiger, "Unsupervised learning of multi-frame optical flow with occlusions," in *European Conference on Computer Vision*, pp. 690–706, 2018.

[40] Y. Wang, Y. Yang, Z. Yang, L. Zhao, P. Wang, and W. Xu, "Occlusion aware unsupervised learning of optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4884–4893, 2018.

[41] H. H. Kim, S. Yu, and C. Tomasi, "Joint detection of motion boundaries and occlusions," in *British Machine Vision Conference*, 2021.

[42] Y. Yang and S. Soatto, "Conditional prior networks for optical flow," in *European Conference on Computer Vision*, pp. 271–287, 2018.

[43] W.-S. Lai, J.-B. Huang, and M.-H. Yang, "Semi-supervised learning for optical flow with generative adversarial networks," in *Advances in Neural Information Processing Systems*, pp. 353–363, 2017.

[44] X. Song, Y. Zhao, J. Yang, C. Lan, and W. Zeng, "Fpcr-net: Feature pyramidal correlation and residual reconstruction for semi-supervised optical flow estimation," *arXiv preprint arXiv:2001.06171*, 2020.

[45] W. Yan, A. Sharma, and R. T. Tan, "Optical flow in dense foggy scenes using semi-supervised learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 13259–13268, 2020.

[46] X. Li and Y. Guo, "Adaptive active learning for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 859–866, 2013.

[47] W. H. Beluch, T. Genewein, A. Nürnberger, and J. M. Köhler, "The power of ensembles for active learning in image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9368–9377, 2018.

[48] S. Roy, A. Unmesh, and V. P. Namboodiri, "Deep active learning for object detection.," in *British Machine Vision Conference*, vol. 362, p. 91, 2018.

[49] J. Choi, I. Elezi, H.-J. Lee, C. Farabet, and J. M. Alvarez, "Active learning for deep object detection via probabilistic modeling," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[50] R. Mackowiak, P. Lenz, O. Ghori, F. Diego, O. Lange, and C. Rother, "Cereals-cost-effective region-based active learning for semantic segmentation," in *British Machine Vision Conference*, 2018.

[51] Y. Siddiqui, J. Valentin, and M. Nießner, "Viewal: Active learning with viewpoint entropy for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9433–9443, 2020.

[52] R. Wang, X.-Z. Wang, S. Kwong, and C. Xu, "Incorporating diversity and informativeness in multiple-instance active learning," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 6, pp. 1460–1475, 2017.

[53] Z. Ren, O. Gallo, D. Sun, M.-H. Yang, E. B. Sudderth, and J. Kautz, "A fusion approach for multi-frame optical flow estimation," in *Winter Conference on Applications of Computer Vision*, pp. 2077–2086, IEEE, 2019.

[54] H. Yu, X. Chen, H. Shi, T. Chen, T. S. Huang, and S. Sun, "Motion pyramid networks for accurate and efficient cardiac motion estimation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 436–446, Springer, 2020.

[55] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *Proceedings of the International Conference on Learning Representations*, 2017.

[56] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[57] T. Miyato, S.-I. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 08, pp. 1979–1993, 2019.

[58] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," in *Advances in Neural Information Processing Systems*, vol. 17, MIT Press, 2005.

[59] D.-H. Lee *et al.*, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on challenges in representation learning, ICML*, vol. 3, p. 896, 2013.

[60] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10687–10698, 2020.

[61] Y. Liu, K. Chen, C. Liu, Z. Qin, Z. Luo, and J. Wang, "Structured knowledge distillation for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2604–2613, 2019.

[62] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[63] A. K. Tehrani, M. Mirzaei, and H. Rivaz, "Semi-supervised training of optical flow convolutional neural networks in ultrasound elastography," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 504–513, Springer, 2020.

[64] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell, "Active learning with gaussian processes for object categorization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1–8, IEEE, 2007.

[65] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel, "Bayesian active learning for classification and preference learning," *arXiv preprint arXiv:1112.5745*, 2011.

[66] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," in *Proceedings of the International Conference on Machine Learning*, pp. 1183–1192, PMLR, 2017.

[67] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach, "Uncertainty-guided continual learning with bayesian neural networks," *Proceedings of the International Conference on Learning Representations*, 2020.

[68] S. Paul, J. H. Bappy, and A. K. Roy-Chowdhury, "Non-uniform subset selection for active learning in structured data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6846–6855, 2017.

[69] K. Wei, R. Iyer, and J. Bilmes, "Submodularity in data subset selection and active learning," in *Proceedings of the International Conference on Machine Learning*, pp. 1954–1963, PMLR, 2015.

[70] E. Ilg, T. Saikia, M. Keuper, and T. Brox, "Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation," in *European Conference on Computer Vision*, pp. 614–630, 2018.

[71] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[72] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2016. arXiv:1512.02134.

[73] 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*, Part IV, LNCS 7577, pp. 611–625, Springer-Verlag, 2012.

[74] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[75] A. Ranjan, V. Jampani, L. Balles, K. Kim, D. Sun, J. Wulff, and M. J. Black, "Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12240–12249, 2019.

[76] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1983–1992, 2018.

[77] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, pp. 8024–8035, Curran Associates, Inc., 2019.

[78] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Proceedings of the International Conference on Learning Representations*, 2014.

[79] R. Jonschkowski, A. Stone, J. T. Barron, A. Gordon, K. Konolige, and A. Angelova, "What matters in unsupervised optical flow," in *European Conference on Computer Vision*, pp. 557–572, Springer, 2020.

[80] S. Yuan, S. Yu, H. Kim, and C. Tomasi, "Semarflow: Injecting semantics into unsupervised optical flow estimation for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9566–9577, October 2023.

[81] J. Shin, S. Kim, S. Kang, S.-W. Lee, J. Paik, B. Abidi, and M. Abidi, "Optical flow-based real-time object tracking using non-prior training active feature model," *Real-time imaging*, vol. 11, no. 3, pp. 204–218, 2005.

[82] L. Capito, U. Ozguner, and K. Redmill, "Optical flow based visual potential field for autonomous driving," in *IEEE Intelligent Vehicles Symposium*, pp. 885–891, IEEE, 2020.

[83] S. Shen, L. Kerofsky, and S. Yogamani, "Optical flow for autonomous driving: Applications, challenges and improvements," *arXiv preprint arXiv:2301.04422*, 2023.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[85] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[86] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proceedings of the International Conference on Learning Representations*, 2021.

[87] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proceedings of the International Conference on Machine Learning*, pp. 10347–10357, PMLR, 2021.

[88] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," in *Proceedings of the International Conference on Learning Representations*, 2021.

[89] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys*, vol. 54, no. 10s, pp. 1–41, 2022.

[90] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical

flow, and scene flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4040–4048, 2016.

[91] 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*, pp. 611–625, Springer, 2012.

[92] S. R. Richter, Z. Hayder, and V. Koltun, "Playing for benchmarks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2232–2241, 2017.

[93] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 969–977, 2018.

[94] H. H. Kim, C. Cintas, G. A. Tadesse, and S. Speakman, "Spatially constrained adversarial attack detection and localization in the representation space of optical flow networks," in *International Joint Conference on Artificial Intelligence*, 2023.

[95] J. J. Yu, A. W. Harley, and K. G. Derpanis, "Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness," in *Proceedings of the European Conference on Computer Vision Workshops* (G. Hua and H. Jégou, eds.), (Cham), pp. 3–10, Springer International Publishing, 2016.

[96] P. Liu, I. King, M. R. Lyu, and J. Xu, "Ddflow: Learning optical flow with unlabeled data distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 8770–8777, 2019.

[97] R. Marsal, F. Chabot, A. Loesch, and H. Sahbi, "Brightflow: Brightness-change-aware unsupervised learning of optical flow," in *Proceedings or the IEEE winter conference on applications of computer vision*, pp. 2061–2070, 2023.

[98] H. H. Kim, S. Yu, and C. Tomasi, "Joint detection of motion boundaries and occlusions," in *British Machine Vision Conference*, 2021.

[99] S. Yu, H. H. Kim, S. Yuan, and C. Tomasi, "Unsupervised flow refinement near motion boundaries," in *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*, British Machine Vision Conference, 2022.

[100] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58443–58469, 2020.

[101] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[102] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1520–1528, 2015.

[103] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. Torr, *et al.*, “Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6881–6890, 2021.

[104] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, “Improving semantic segmentation via video propagation and label relaxation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8856–8865, 2019.

[105] L. Sevilla-Lara, D. Sun, V. Jampani, and M. J. Black, “Optical flow with semantic segmentation and localized layers,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3889–3898, 2016.

[106] J. Hur and S. Roth, “Joint optical flow and temporally consistent semantic segmentation,” in *European Conference on Computer Vision*, pp. 163–177, Springer, 2016.

[107] M. Bai, W. Luo, K. Kundu, and R. Urtasun, “Exploiting semantic information and deep matching for optical flow,” in *European Conference on Computer Vision*, pp. 154–170, Springer, 2016.

[108] J. Wulff, L. Sevilla-Lara, and M. J. Black, “Optical flow in mostly rigid scenes,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4671–4680, 2017.

[109] M. Ding, Z. Wang, B. Zhou, J. Shi, Z. Lu, and P. Luo, “Every frame counts: Joint learning of video segmentation and optical flow,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 10713–10720, 2020.

[110] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “Flownet 2.0: Evolution of optical flow estimation with deep networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1647–1655, 2017.

[111] A. Ranjan and M. J. Black, “Optical flow estimation using a spatial pyramid network,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2720–2729, 2017.

[112] H. Xu, J. Yang, J. Cai, J. Zhang, and X. Tong, "High-resolution optical flow from 1d attention and correlation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10498–10507, 2021.

[113] X. Sui, S. Li, X. Geng, Y. Wu, X. Xu, Y. Liu, R. Goh, and H. Zhu, "Craft: Cross-attentional flow transformer for robust optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 17602–17611, 2022.

[114] H. Xu, J. Zhang, J. Cai, H. Rezatofighi, and D. Tao, "Gmflow: Learning optical flow via global matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8121–8130, 2022.

[115] W. Im, T.-K. Kim, and S.-E. Yoon, "Unsupervised learning of optical flow with deep feature similarity," in *European Conference on Computer Vision*, pp. 172–188, Springer, 2020.

[116] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2881–2890, 2017.

[117] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *Proceedings or the IEEE winter conference on applications of computer vision*, pp. 1451–1460, Ieee, 2018.

[118] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," in *European Conference on Computer Vision*, pp. 325–341, 2018.

[119] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation," *International Journal of Computer Vision*, vol. 129, pp. 3051–3068, 2021.

[120] Z. Huang, X. Wang, L. Huang, C. Huang, Y. Wei, and W. Liu, "Ccnet: Criss-cross attention for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 603–612, 2019.

[121] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, "Dual attention network for scene segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3146–3154, 2019.

[122] S. Borse, Y. Wang, Y. Zhang, and F. Porikli, "Inverseform: A loss function for structured boundary-aware segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5901–5911, 2021.

[123] A. Tao, K. Sapra, and B. Catanzaro, "Hierarchical multi-scale attention for semantic segmentation," *arXiv preprint arXiv:2005.10821*, 2020.

[124] A. Ganeshan, A. Vallet, Y. Kudo, S.-i. Maeda, T. Kerola, R. Ambrus, D. Park, and A. Gaidon, "Warp-refine propagation: Semi-supervised auto-labeling via cycle-consistency," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15499–15509, 2021.

[125] Y. Cai, L. Dai, H. Wang, and Z. Li, "Multi-target pan-class intrinsic relevance driven model for improving semantic segmentation in autonomous driving," *IEEE Transactions on Image Processing*, vol. 30, pp. 9069–9084, 2021.

[126] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3213–3223, 2016.

[127] F. A. Reda, G. Liu, K. J. Shih, R. Kirby, J. Barker, D. Tarjan, A. Tao, and B. Catanzaro, "Sdc-net: Video prediction using spatially-displaced convolution," in *European Conference on Computer Vision*, pp. 718–733, 2018.

[128] H. Rashed, S. Yogamani, A. El-Sallab, P. Krizek, and M. El-Helw, "Optical flow augmented semantic segmentation networks for automated driving," in *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pp. 165–172, 2019.

[129] X. Li, J. Bai, K. Yang, and Y. Tong, "Flow2seg: Motion-aided semantic segmentation," in *Proceedings of the International Conference on Artificial Neural Networks*, pp. 225–237, Springer, 2019.

[130] Z. Wu, X. Wu, X. Zhang, S. Wang, and L. Ju, "Semantic stereo matching with pyramid cost volumes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7484–7493, 2019.

[131] G. Yang, H. Zhao, J. Shi, Z. Deng, and J. Jia, "Segstereo: Exploiting semantic information for disparity estimation," in *European Conference on Computer Vision*, pp. 636–651, 2018.

[132] Z. Ren, D. Sun, J. Kautz, and E. Sudderth, "Cascaded scene flow prediction using semantic segmentation," in *Proceedings of the International Conference on 3D Vision*, pp. 225–233, IEEE, 2017.

[133] C. Feng, L. Ma, C. Zhang, Z. Chen, L. Ge, and S. Jiang, "Ss-sf: Piecewise 3d scene flow estimation with semantic segmentation," *IEEE Access*, vol. 9, pp. 22745–22759, 2021.

[134] Y. Shi and K. Ma, "Safit: Segmentation-aware scene flow with improved transformer," in *Proceedings of the International Conference on Robotics and Automation*, pp. 10648–10655, IEEE, 2022.

[135] G. Bradski, "The opencv library.," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[136] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *European Conference on Computer Vision*, pp. 801–818, 2018.

[137] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial intelligence and machine learning for multi-domain operations applications*, vol. 11006, pp. 369–386, SPIE, 2019.

[138] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Models matter, so does training: An empirical study of cnns for optical flow estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 6, pp. 1408–1423, 2019.

[139] S. Yuan and C. Tomasi, "Ufd-prime: Unsupervised joint learning of optical flow and stereo depth through pixel-level rigid motion estimation," *arXiv preprint arXiv:2310.04712*, 2023.

[140] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1440–1448, 2015.

[141] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, pp. 630–645, Springer, 2016.

[142] F. Kong, S. Yuan, W. Hao, and R. Henao, "Mitigating test-time bias for fair image retrieval," *arXiv preprint arXiv:2305.19329*, 2023.

[143] H. Jung, Z. Hui, L. Luo, H. Yang, F. Liu, S. Yoo, R. Ranjan, and D. Demandolx, "Anyflow: Arbitrary scale optical flow with implicit neural representation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5455–5465, 2023.

[144] G. Wu, X. Liu, K. Luo, X. Liu, Q. Zheng, S. Liu, X. Jiang, G. Zhai, and W. Wang, "Accflow: Backward accumulation for long-range optical flow," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12119–12128, 2023.

[145] A. Luo, F. Yang, K. Luo, X. Li, H. Fan, and S. Liu, "Learning optical flow with adaptive graph reasoning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 1890–1898, 2022.

[146] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 66–75, 2017.

[147] X. Guo, K. Yang, W. Yang, X. Wang, and H. Li, "Group-wise correlation stereo network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3273–3282, 2019.

[148] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5410–5418, 2018.

[149] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr, "Ga-net: Guided aggregation net for end-to-end stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 185–194, 2019.

[150] X. Cheng, P. Wang, and R. Yang, "Learning depth with convolutional spatial propagation network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2361–2379, 2019.

[151] L. Lipson, Z. Teed, and J. Deng, "Raft-stereo: Multilevel recurrent field transforms for stereo matching," in *3DV*, pp. 218–227, IEEE, 2021.

[152] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, pp. 213–229, Springer, 2020.

[153] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[154] P. Liu, I. King, M. R. Lyu, and J. Xu, "Flow2stereo: Effective self-supervised learning of optical flow and stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6648–6657, 2020.

[155] S. Liu, K. Luo, N. Ye, C. Wang, J. Wang, and B. Zeng, "Oiflow: Occlusion-inpainting optical flow estimation by unsupervised learning," *IEEE Transactions on Image Processing*, vol. 30, pp. 6420–6433, 2021.

[156] S. Liu, K. Luo, A. Luo, C. Wang, F. Meng, and B. Zeng, "Asflow: Unsupervised optical flow learning with adaptive pyramid sampling," *IEEE Transactions on Circuit System and Video Technology*, vol. 32, no. 7, pp. 4282–4295, 2021.

[157] G. Chen, K. Han, B. Shi, Y. Matsushita, and K.-Y. K. Wong, "Deep photometric stereo for non-lambertian surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 129–142, 2020.

[158] J. Shi *et al.*, "Good features to track," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, IEEE, 1994.

[159] M. A. Mohamed, H. A. Rashwan, B. Mertsching, M. A. García, and D. Puig, "Illumination-robust optical flow using a local directional pattern," *IEEE Transactions on Circuit System and Video Technology*, vol. 24, no. 9, pp. 1499–1508, 2014.

[160] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[161] Y. Zou, Z. Luo, and J.-B. Huang, "Df-net: Unsupervised joint learning of depth and flow using cross-task consistency," in *European Conference on Computer Vision*, pp. 36–53, 2018.

[162] C. Luo, Z. Yang, P. Wang, Y. Wang, W. Xu, R. Nevatia, and A. Yuille, "Every pixel counts++: Joint learning of geometry and motion with 3d holistic understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2624–2641, 2019.

[163] Y. Wang, P. Wang, Z. Yang, C. Luo, Y. Yang, and W. Xu, "Unos: Unified unsupervised optical-flow and stereo-depth estimation by watching videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8071–8081, 2019.

[164] C. Chi, Q. Wang, T. Hao, P. Guo, and X. Yang, "Feature-level collaboration: Joint unsupervised learning of optical flow, stereo depth and camera motion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2463–2473, 2021.

[165] Y. Jiao, T. D. Tran, and G. Shi, "Effiscene: Efficient per-pixel rigidity inference for unsupervised joint learning of optical flow, depth, camera pose and motion segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5538–5547, 2021.

[166] G. Wang, C. Zhang, H. Wang, J. Wang, Y. Wang, and X. Wang, "Unsupervised learning of depth, optical flow and pose with occlusion from 3d geometry," *IEEE Trans. Intell. Transport. Sys.*, vol. 23, no. 1, pp. 308–320, 2020.

[167] L. Liu, G. Zhai, W. Ye, and Y. Liu, "Unsupervised learning of scene flow estimation fusing with local rigidity.," in *International Joint Conference on Artificial Intelligence*, pp. 876–882, 2019.

[168] S. Ullman, "The interpretation of structure from motion," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 203, no. 1153, pp. 405–426, 1979.

[169] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4104–4113, 2016.

[170] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *ACM Communications*, vol. 24, no. 6, pp. 381–395, 1981.

[171] V. Lepetit, F. Moreno-Noguer, and P. Fua, “Epnp: An accurate o(n) solution to the pnp problem,” *International Journal of Computer Vision*, vol. 81, pp. 155–166, 2009.

[172] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611, pp. 586–606, Spie, 1992.

[173] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki, “Sfm-net: Learning of structure and motion from video,” *arXiv preprint arXiv:1704.07804*, 2017.

[174] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, “Unsupervised learning of depth and ego-motion from video,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1851–1858, 2017.

[175] G. Yang and D. Ramanan, “Learning to segment rigid motions from two frames,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1266–1275, 2021.

[176] Z. Teed and J. Deng, “Raft-3d: Scene flow using rigid-motion embeddings,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8375–8384, 2021.

[177] H.-P. Huang, C. Herrmann, J. Hur, E. Lu, K. Sargent, A. Stone, M.-H. Yang, and D. Sun, “Self-supervised autoflow,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11412–11421, 2023.

[178] Y. Han, K. Luo, A. Luo, J. Liu, H. Fan, G. Luo, and S. Liu, “Realflow: Em-based realistic optical flow dataset generation from videos,” in *European Conference on Computer Vision*, pp. 288–305, Springer, 2022.

[179] M. J. Hannah, *Computer matching of areas in stereo images.* Stanford University, 1974.

[180] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence,” in *European Conference on Computer Vision*, pp. 151–158, Springer, 1994.

[181] V. Kolmogorov, “Convergent tree-reweighted message passing for energy minimization,” in *International Workshop on Artificial Intelligence and Statistics*, pp. 182–189, PMLR, 2005.

[182] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graphics*, vol. 28, no. 3, p. 24, 2009.

[183] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2007.

[184] C. Zhou, H. Zhang, X. Shen, and J. Jia, "Unsupervised learning of stereo matching," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1567–1575, 2017.

[185] S. Joung, S. Kim, K. Park, and K. Sohn, "Unsupervised stereo matching using confidential correspondence consistency," *IEEE Trans. Intell. Transport. Sys.*, vol. 21, no. 5, pp. 2190–2203, 2019.

[186] A. Li, Z. Yuan, Y. Ling, W. Chi, S. Zhang, and C. Zhang, "Unsupervised occlusion-aware stereo matching with directed disparity smoothing," *IEEE Trans. Intell. Transport. Sys.*, vol. 23, no. 7, pp. 7457–7468, 2021.

[187] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 270–279, 2017.

[188] P. H. Torr, "Geometric motion segmentation and model selection," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 356, no. 1740, pp. 1321–1340, 1998.

[189] P. H. Torr, A. W. Fitzgibbon, and A. Zisserman, "The problem of degeneracy in structure and motion recovery from uncalibrated image sequences," *International Journal of Computer Vision*, vol. 32, pp. 27–44, 1999.

[190] R. Vidal and S. Sastry, "Optimal segmentation of dynamic scenes from two perspective views," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. II–II, IEEE, 2003.

[191] R. Vidal, Y. Ma, S. Soatto, and S. Sastry, "Two-view multibody structure from motion," *International Journal of Computer Vision*, vol. 68, no. 1, pp. 7–25, 2006.

[192] X. Xu, L.-F. Cheong, and Z. Li, "3d rigid motion segmentation with mixed and unknown number of models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 1–16, 2019.

[193] Sawhney, "3d geometry from planar parallax," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 929–934, IEEE, 1994.

[194] C. Yuan, G. Medioni, J. Kang, and I. Cohen, "Detecting motion regions in the presence of a strong parallax from a moving camera by multiview geometric constraints," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 9, pp. 1627–1641, 2007.

[195] W. P. H. Thurston, *Three-Dimensional Geometry and Topology, Volume 1: Volume 1.* Princeton university press, 1997.

[196] Z. Teed and J. Deng, "Tangent space backpropagation for 3d transformation groups," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[197] H.-Y. Lai, Y.-H. Tsai, and W.-C. Chiu, "Bridging stereo matching and optical flow via spatiotemporal correspondence," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1890–1899, 2019.

[198] H. Xu, J. Zhang, J. Cai, H. Rezatofighi, F. Yu, D. Tao, and A. Geiger, "Unifying flow, stereo and depth estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[199] J. Hur and S. Roth, "Self-supervised monocular scene flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7396–7405, 2020.

[200] B. Bayramli, J. Hur, and H. Lu, "Raft-msf: Self-supervised monocular scene flow using recurrent optimizer," *International Journal of Computer Vision*, pp. 1–13, 2023.

[201] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[202] H. Anton and C. Rorres, *Elementary linear algebra: applications version.* John Wiley & Sons, 2013.

[203] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[204] X. Wang, S. Yuan, C. Wu, and R. Ge, "Guarantees for tuning the step size using a learning-to-learn approach," in *Proceedings of the International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 10981–10990, PMLR, 18–24 Jul 2021.

[205] Q. Wang, Y.-Y. Chang, R. Cai, Z. Li, B. Hariharan, A. Holynski, and N. Snavely, "Tracking everything everywhere all at once," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[206] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, “Trackformer: Multi-object tracking with transformers,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8844–8854, 2022.

[207] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, “Segment anything,” *arXiv preprint arXiv:2304.02643*, 2023.

[208] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[209] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, “Building rome in a day,” *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[210] A. M. Andrew, “Multiple view geometry in computer vision,” *Kybernetes*, 2001.

[211] R. I. Hartley and P. Sturm, “Triangulation,” *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[212] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle adjustment—a modern synthesis,” in *International workshop on vision algorithms*, pp. 298–372, Springer, 1999.

[213] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *International Journal of Computer Vision*, vol. 47, no. 1, pp. 7–42, 2002.

[214] D. Eigen, C. Puhrsch, and R. Fergus, “Depth map prediction from a single image using a multi-scale deep network,” *arXiv preprint arXiv:1406.2283*, 2014.

[215] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, “Deeper depth prediction with fully convolutional residual networks,” in *International Conference on 3D Vision*, pp. 239–248, IEEE, 2016.

[216] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, “Deep ordinal regression network for monocular depth estimation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2002–2011, 2018.

[217] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, “Digging into self-supervised monocular depth estimation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3828–3838, 2019.

[218] J. Xie, R. Girshick, and A. Farhadi, “Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks,” in *European Conference on Computer Vision*, pp. 842–857, Springer, 2016.

[219] R. Garg, V. K. Bg, G. Carneiro, and I. Reid, “Unsupervised cnn for single view depth estimation: Geometry to the rescue,” in *European Conference on Computer Vision*, pp. 740–756, Springer, 2016.

[220] Z. Yang, P. Wang, W. Xu, L. Zhao, and R. Nevatia, “Unsupervised learning of geometry with edge-aware depth-normal consistency,” *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[221] Z. Yang, P. Wang, Y. Wang, W. Xu, and R. Nevatia, “Lego: Learning edge with geometry all at once by watching videos,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 225–234, 2018.

[222] C. Tomasi and T. Kanade, “Detection and tracking of point,” tech. rep., features. Technical Report CMU-CS-91-132, Carnegie, Mellon University, 1991.

[223] J. Shi and C. Tomasi, “Good features to track,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[224] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “Flownet 2.0: Evolution of optical flow estimation with deep networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2462–2470, 2017.

[225] J. Xu, R. Ranftl, and V. Koltun, “Accurate optical flow via direct cost volume processing,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1289–1297, 2017.

[226] M. Jaderberg, K. Simonyan, and A. Zisserman, “Spatial transformer networks,” in *Advances in Neural Information Processing Systems*, pp. 2017–2025, 2015.

[227] H. Wang, R. Fan, and M. Liu, “Cot-amflow: Adaptive modulation network with co-teaching strategy for unsupervised optical flow estimation,” *arXiv preprint arXiv:2011.02156*, 2020.

[228] A. Spoerri, “The early detection of motion boundaries,” tech. rep., Massachusetts Institute of Technology, 1990.

[229] C. Liu, W. Freeman, and E. Adelson, “Analysis of contour motions,” *Advances in Neural Information Processing Systems*, vol. 19, pp. 913–920, 2006.

[230] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, “Learning to detect motion boundaries,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2578–2586, 2015.

[231] P. Dollár and C. L. Zitnick, “Fast edge detection using structured forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 8, pp. 1558–1570, 2014.

[232] X. Yin, X. Dai, X. Wang, M. Zhang, D. Tao, and L. Davis, "Deep motion boundary detection," *arXiv preprint arXiv:1804.04785*, 2018.

[233] J. Y. Wang and E. H. Adelson, "Layered representation for motion analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 361–366, IEEE, 1993.

[234] T. Darrell and A. P. Pentland, "Cooperative robust estimation using layers of support," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 5, pp. 474–487, 1995.

[235] M. Irani, B. Rousso, and S. Peleg, "Computing occluding and transparent motions," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 5–16, 1994.

[236] J. Odobez and P. Bouthemy, "Separation of moving regions from background in an image sequence acquired with a mobil camera," in *Video Data Compression for Multimedia Computing*, pp. 283–311, Springer, 1997.

[237] T. Zhang and C. Tomasi, "On the consistency of instantaneous rigid motion estimation," *International Journal of Computer Vision*, vol. 46, no. 1, pp. 51–79, 2002.

[238] K. J. Hanna and N. E. Okamoto, "Combining stereo and motion analysis for direct estimation of scene structure," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 357–365, IEEE, 1993.

[239] Y. Kim and J. Aggarwal, "Determining object motion in a sequence of stereo images," *IEEE Journal on Robotics and Automation*, vol. 3, no. 6, pp. 599–614, 1987.

[240] R. Koch, "3-d surface reconstruction from stereoscopic image sequences," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 109–114, IEEE, 1995.

[241] Z. Zhang and O. D. Faugeras, "Three-dimensional motion computation and object segmentation in a long sequence of stereo frames," *International Journal of Computer Vision*, vol. 7, no. 3, pp. 211–241, 1992.

[242] A. Wedel, D. Cremers, T. Pock, and H. Bischof, "Structure-and motion-adaptive regularization for high accuracy optic flow," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1663–1668, IEEE, 2009.

[243] K. Yamaguchi, D. McAllester, and R. Urtasun, "Robust monocular epipolar flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1862–1869, 2013.

[244] A. Roussos, C. Russell, R. Garg, and L. Agapito, “Dense multibody motion estimation and reconstruction from a handheld camera,” in *2012 IEEE International Symposium on Mixed and Augmented Reality*, pp. 31–40, IEEE, 2012.

[245] Y. Chen, C. Schmid, and C. Sminchisescu, “Self-supervised learning with geometric constraints in monocular video: Connecting flow, depth, and camera,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7063–7072, 2019.

[246] M. Menze and A. Geiger, “Object scene flow for autonomous vehicles,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[247] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2117–2125, 2017.

Biography

Shuai Yuan was born in China in 1996. He grew up in Nanjing and obtained his Bachelor's of Science degree in Computer Science and Technology from Nanjing University in June 2018. He continued to pursue his PhD in Computer Science at Duke University and is expected to graduate in December 2023. He has also completed three full-time research internships at Meta Reality Labs in 2020, 2022, and 2023.

As a PhD student at Duke, he published many research papers on top-tier conferences including ICCV, ECCV, ICML, NeurIPS, *etc.*. His research mainly focuses on optical flow estimation, especially for cases with limited or no ground-truth labels. His work also combines optical flow estimation with other tasks and techniques such as active learning [30], semantic segmentation [80], and stereo vision [139]. During his internships at Meta Reality Labs, he conducted cutting-edge research on AI techniques that support the next-generation virtual/augmented/mixed reality technology. Yuan also has research experiences on flow boundary refinement [99], video interpolation [9], image retrieval [142], and meta-learning [204]. He has won the best paper award from the ACCV Workshop on Vision Transformers: Theory and Application as a third-author. He has also actively reviewed for ACCV and NeurIPS conferences.