A Lightweight Optical Flow CNN – Revisiting Data Fidelity and Regularization

Tak-Wai Hui, Xiaoou Tang, *Fellow, IEEE,* and Chen Change Loy, *Senior Member, IEEE*

Abstract—Over four decades, the majority addresses the problem of optical flow estimation using variational methods. With the advance of machine learning, some recent works have attempted to address the problem using convolutional neural network (CNN) and have showed promising results. FlowNet2 [1], the state-of-the-art CNN, requires over 160M parameters to achieve accurate flow estimation. Our LiteFlowNet2 outperforms FlowNet2 on Sintel and KITTI benchmarks, while being 25.3 times smaller in the footprint and 3.1 times faster in the running speed. LiteFlowNet2 which is built on the foundation laid by conventional methods has marked a milestone to achieve the corresponding roles as data fidelity and regularization in variational methods. We present an effective flow inference approach at each pyramid level through a novel lightweight cascaded network. It provides high flow estimation accuracy through early correction with seamless incorporation of descriptor matching. A novel flow regularization layer is used to ameliorate the issue of outliers and vague flow boundaries through a novel feature-driven local convolution. Our network also owns an effective structure for pyramidal feature extraction and embraces feature warping rather than image warping as practiced in FlowNet2. Comparing to our earlier work, LiteFlowNet2 improves the optical flow accuracy on Sintel clean pass by 24%, Sintel final pass by 8.9%, KITTI 2012 by 16.8%, and KITTI 2015 by 17.5%. Our network protocol and trained models will be made publicly available on https://github.com/twhui/LiteFlowNet2.

Index Terms—Convolutional neural network, cost volume, deep learning, optical flow, regularization, and warping.

1 Introduction

Optical flow which refers to the point correspondence across a pair of images is induced by the spatial motion at any image position. Due to the well-known aperture problem, optical flow cannot be directly measured. The partial observability of optical flow is the major reason that makes it a challenging problem. The optical flow problem has attracted many attentions since the seminal works by Horn and Schunck [2], and Lucas and Kanade [3] about four decades ago. Most of the approaches estimate optical flow relying on an energy minimization method in a coarse-to-fine framework [4], [5], [6]. Optical flow is refined iteratively using a numerical approach from the coarsest level towards the finest level by warping one of the images in the image pair towards the other using the flow estimate from the coarser level. The warping technique is theoretically justified to minimize the energy functional [4], [5]. On the other hand, normal flow which is directly measurable is more ready for motion estimation [7], [8], [9].

FlowNet [10] and its successor FlowNet2 [1], are pioneers to use convolutional neural network (CNN) for optical flow estimation. Their accuracies especially the successor are approaching to the state-of-the-art energy minimization approaches, while the speed is several orders of magnitude faster. To push the envelop of accuracy, FlowNet2 is designed as a cascade of variants of FlowNet that each network in the cascade refines the preceding flow field by contributing on the flow adjustment between the first image and the warped second image. The model, as a result, comprises over 160M parameters and has a slow runtime, which could be formidable in many applications. A recent CNN termed SPyNet [11] attempts a smaller network with 1.2M parameters by adopting image warping in each pyramid level. Nonetheless, the accuracy can only match that of FlowNet but not FlowNet2.

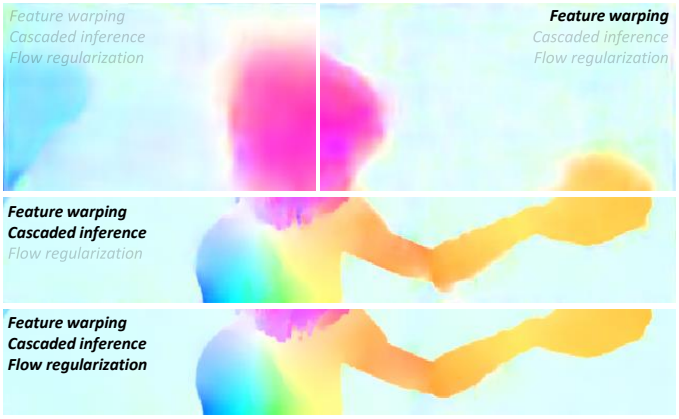

Fig. 1: Examples demonstrate the effectiveness of the proposed components in LiteFlowNet for i) feature warping, ii) cascaded flow inference, and iii) flow regularization. Enabled components are indicated with bold black fonts.

Our earlier work, LiteFlowNet [12], is inspired by their successes of addressing the classical flow problem using CNN. We have drilled down some of the important principles of solving the flow problem which are used by conventional methods are however not fully explored by FlowNet2 and SPyNet. An overview of the working principles of the state-of-the-art optical flow CNNs is summarized in Table 1. We fully unleash the potential of deep convolutional network for optical flow estimation by adopting *data*

- *T.-W. Hui and X. Tang are with the Department of Information Engineering, The Chinese University of Hong Kong, Sha Tin, Hong Kong. E-mails: {twhui, xtang}@ie.cuhk.edu.hk.*
- *C. C. Loy is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: ccloy@ntu.edu.sg.*

TABLE 1: A comparison of different optical flow CNNs.

	FlowNetS [10]	FlowNetC [10]	FlowNet2 [1]	SPyNet [11]	LiteFlowNet
No. of stacked networks	1	1	5	1	1
Feature pyramid	7 levels	7 levels	7 levels	✗	6 levels
Upsampling features	✓	✓	✓	✗	✗
Cost volume	✗	single-level long range	single-level long range	✗	multi-level short range
Warping	✗	✗	image per stack	image per level	feature per level
Flow inference	direct	direct	direct	residual	cascaded & residual
Regularization	✗	✗	✗	✗	one per level

fidelity and *regularization* in classical variational methods to CNN.

In this work, we provide more details on the correspondences between conventional methods and our optical flow CNNs. We also present LiteFlowNet2 that has a better flow accuracy and runtime by optimizing the network architecture and training procedures of LiteFlowNet. More details are provided in Section 6.2. We first discuss the motivations from classical variational methods on the design of our optical flow networks.

Data Term. Point correspondence across two images is generally constrained by the classical brightness constancy [2]. Gradient [4] and higher-order brightness constancy [5] assumptions are also widely used in the literature. The above constancy assumptions are collectively known as data fidelity and are often combined to form a hybrid data term [4], [13]. Although different matching quantities are proved to be useful in solving the optical flow problem, finding a correct proportion of their contributions in the hybrid data term is non-trivial and requires a highly engineered data fusion model [14]. An improper mixture of the brightness and gradient terms can severely affect the performance [13]. To avoid the aforementioned difficulties, we do not explicitly define the matching quantities. We use a CNN to train a *pyramidal feature descriptor* which resembles data fidelity in variational methods and is prepared for establishing robust point correspondence later. An image pair is transformed from the spatial domain to the learned feature space in forms of two pyramids of multi-scale high-dimensional feature maps.

Image Warping. It is proved that warping effectively minimizes an energy functional by using a numerical method in a coarse-to-fine framework [4], [5]. Intuitively, at each iteration the numerical solver displaces every pixel value of the second image in the image pair according to the constraints imposed in the functional so that the warped image has a visual appearance close to the first image. Image warping is practiced in FlowNet2 [1] and SPyNet [11] between network cascades and pyramid levels respectively. However, warping an image and then generating the feature maps of the warped image as the above CNN-based methods are two ordered steps. We find that the two steps can be reduced to a single one by directly warping the feature maps of the second image, which have been provided by the feature descriptor. This one-step *feature warping* (f-warp) process reduces the more discriminative feature-space distance instead of the RGB-space distance between the two images. This makes our network more powerful and efficient in addressing the optical flow problem.

Smoothness Term. Merely using data fidelity for flow estimation is an ill-posed problem. One example is the one-to-many point correspondences in homogeneous regions of an image pair. With the co-occurrence between motion boundaries and intensity edges, flow estimate is often smoothed by anisotropic image-driven regularization [13], [15]. However, image-driven strategies are prone to over-segmentation artifacts in the textured image regions since image edges do not necessarily correspond to flow edges. More advanced methods overcome the previous shortcomings through the use of an anisotropic image- and flow-driven regularization [16] and a complementary regularizer [17]. With the motivation to establish robust point correspondence in the learned feature space, we generalize the use of regularization from the spatial space to the feature space. We allow the flow field to be regularized by our novel *feature-driven local convolution* (f-lconv) layer at each pyramid level. The kernels of such a local convolution are adaptive to the pyramidal features from the encoder, flow estimate, and occlusion probability map. This makes the flow regularization to be both flow- and image-aware. We name it as the feature-driven local convolution layer in order to distinguish it from the local convolution (lconv) layer of which filter weights are locally fixed in conventional CNNs [18]. To the best of our knowledge, we are the first to explore such a flow regularization layer.

The proposed network, dubbed LiteFlowNet, consists of an encoder and a decoder. The encoder maps the given image pair, respectively, into two pyramids of multi-scale high-dimensional features. The decoder then estimates the multi-scale flow fields in a coarse-to-fine framework. At each pyramid level, the decoder infers the flow field by selecting and using the features of the same resolution from the feature pyramids. This design leads to a lighter network compared to FlowNet and FlowNet2 that adopt U-Net architecture [19] for flow inference. In comparison to SPyNet, our network separates the process of feature extraction and flow estimation into encoder and decoder respectively. This helps us to better pinpoint the bottleneck of accuracy and model size.

At each pyramid level, we introduce a novel cascaded flow inference. Each of them has a f-warp layer to displace the feature maps of the second image towards the first image using the flow estimate from the previous level. Flow residue is computed to further reduce the feature-space distance between the images. This design is advantageous to the conventional design of using a single network for flow inference. First, the cascade progressively improves flow accuracy thus allowing an early correction of the estimate without passing more errors to the next level. Second, this design allows seamless integration with descriptor matching. We assign a matching network to the first inference. Consequently, pixel-accuracy flow field can be generated first and then refined to sub-pixel accuracy in the subsequent inference network. Since at each pyramid level the feature-space distance between the images has been reduced by f-warp, a short searching range rather than a long searching range [1], [10] is used to establish a cost volume. Besides, matching can be performed at sampled positions to aggregate a sparse cost volume. This effectively reduces the computational burden raised by the explicit matching. After cascaded flow inference, the flow field is further regularized by our novel f-lconv layer.

The effectiveness of the aforementioned contributions are depicted in Figure 1. In summary, the contributions of this work

are in three aspects:

1) We present a study to bridge the correspondences between the well-established principles in conventional methods for optical flow estimation and our lightweight optical flow CNNs.
2) LiteFlowNet2, a lightweight convolutional network, is evolved from our earlier work LiteFlowNet [12] to better address the problem of optical flow estimation by improving flow accuracy and computation time. More design analysis and experimental results are also provided.
3) LiteFlowNet2 outperforms the state-of-the-art FlowNet2 [1] on Sintel and KITTI benchmarks, while being 25.3 times smaller in the model size and 3.1 times faster in the runtime. The optical flow processing frequency of LiteFlowNet2 reaches up to 25 flow fields per second for an image pair with size 1024×436 on a NVIDIA GTX 1080. Our network protocol and trained models will be made publicly available on https://github.com/twhui/LiteFlowNet2.

2 Related Work

The problem of optical flow estimation has been widely studied in the literature since 1980s. A detailed review is beyond the scope of this work. Here, we briefly review some of the major approaches, namely variational, machine learning, and CNN-based methods.

Variational Methods. Since the pioneering work by Horn and Schunck [2], variational methods have dominated optical flow estimation. Brox *et al.* address illumination changes by combining the brightness and gradient constancy assumptions [4]. Brox *et al.* integrate rich descriptors into a variational formulation [6]. In DeepFlow [20], Weinzaepfel *et al.* propose to correlate multi-scale patches and incorporate this as the matching term in a functional. In PatchMatch Filter [21], Lu *et al.* establish dense correspondence using the superpixel-based PatchMatch [22]. Revaud *et al.* propose a method EpicFlow that uses externally matched flows as the initialization and then performs interpolation [23]. Zimmer *et al.* design the complementary regularization that exploits directional information from the constraints imposed in data term [17]. Our network that infers optical flow and performs flow regularization is inspired by the use of data fidelity and regularization in variational methods.

Machine Learning Methods. Black *et al.* propose to represent complex image motion as a linear combination of the learned basis vectors [24]. Roth *et al.* formulates the prior probability of flow field as Field-of-Experts model [25] that captures higher order spatial statistics [26]. Sun *et al.* study the probabilistic model of brightness inconstancy in a high-order random field framework [16]. Nir *et al.* represent image motion using the over-parameterization model [27]. Rosenbaum *et al.* model the local statistics of optical flow using Gaussian mixtures [28]. Given a set of sparse matches, Wulff *et al.* propose to regress them to a dense flow field using a set of basis flow fields (PCA-Flow) [29]. It can be shown that the parameterized model [24], [27], [29] can be efficiently implemented in a CNN.

CNN-Based Methods. In the work of Dosovitskiy *et al.* termed FlowNet [10], a post-processing step that involves energy minimization is required to reduce smoothing effect across flow boundaries. This process is not end-to-end trainable. On the contrary, we present an end-to-end approach that performs in-network flow regularization using our novel f-lcon layer, which plays a similar role as the regularization term in variational methods. In FlowNet2 [1], Ilg *et al.* introduce a huge network cascade (over 160M parameters) that consists of variants of FlowNet. The cascade improves flow accuracy with an expense of model size and computational complexity. A compact network termed SPyNet [11] from Ranjan *et al.* is inspired from spatial pyramid. Nevertheless, the accuracy is far below FlowNet2. LiteFlowNetX, a small-sized variant of our network, outperforms SPyNet while being 1.33 times smaller in the model size. Zweig *et al.* present a network to interpolate third-party sparse flows but requiring off-the-shelf edge detector [30]. DeepFlow [20] that involves convolution and pooling operations is however not a CNN, since the "filter weights" are non-trainable image patches. According to the terminology used in FlowNet, DeepFlow uses correlation. Our LiteFlowNet [12] from Hui *et al.* and PWC-Net [31] from Sun *et al.* both are pioneers for the next generation of lightweight optical flow networks. PWC-Net and LiteFlowNet reduce the number of model parameters by 17 and 30 times smaller than FlowNet2 respectively while the performance is on par with it. Both of them use feature matching and warping as inspired by conventional methods. LiteFlowNet further incorporates a cascaded flow inference to refine coarse estimates resulted from feature matching. Furthermore, a novel feature-driven local convolution is used to regularize flow fields. These differences make LiteFlowNet to attain a smaller model size than PWC-Net.

An alternative approach for establishing point correspondence is to match image patches. Zagoruyko *et al.* first introduce to CNN-feature matching [32]. Güney *et al.* find feature representation and formulate optical flow estimation in MRF [33]. Bailer *et al.* [34] use multi-scale features and then perform feature matching as Flow Fields [35]. Although pixel-wise matching can establish accurate point correspondence, the computational demand is too high for practical use (it takes several seconds even a GPU is used). As a tradeoff, Dosovitskiy *et al.* [10] and Ilg *et al.* [1] perform feature matching only at a reduced spatial resolution. We reduce the computational burden of feature matching by using a short-ranged matching of warped CNN features and a sub-pixel refinement at every pyramid level. We further reduce the computation cost by constructing a sparse cost volume at high-resolution pyramid levels.

We are inspired by the feature transformation used in Spatial Transformer [36]. Our network uses the proposed f-warp layer to displace each channel[1] of the given vector-valued feature according to the provided flow field. Unlike Spatial Transformer, f-warp layer is not fully constrained and is a relaxed version of it as the flow field is not parameterized. While transformation in FlowNet2 and SPyNet is limited to images, LiteFlowNet is a more generic warping network that warps high-level CNN features.

3 LiteFlowNet

Two lightweight sub-networks that are specialized in *pyramidal feature extraction* and *optical flow estimation* constitute LiteFlowNet as shown in Figure 2. Since the spatial dimension of feature maps is contracting in feature extraction and that of flow fields is expanding in flow estimation, we name the two

1. We can also use f-warp layer to displace each channel *differently* when multiple flow fields are supplied. The usage, however, is beyond the scope of this work.

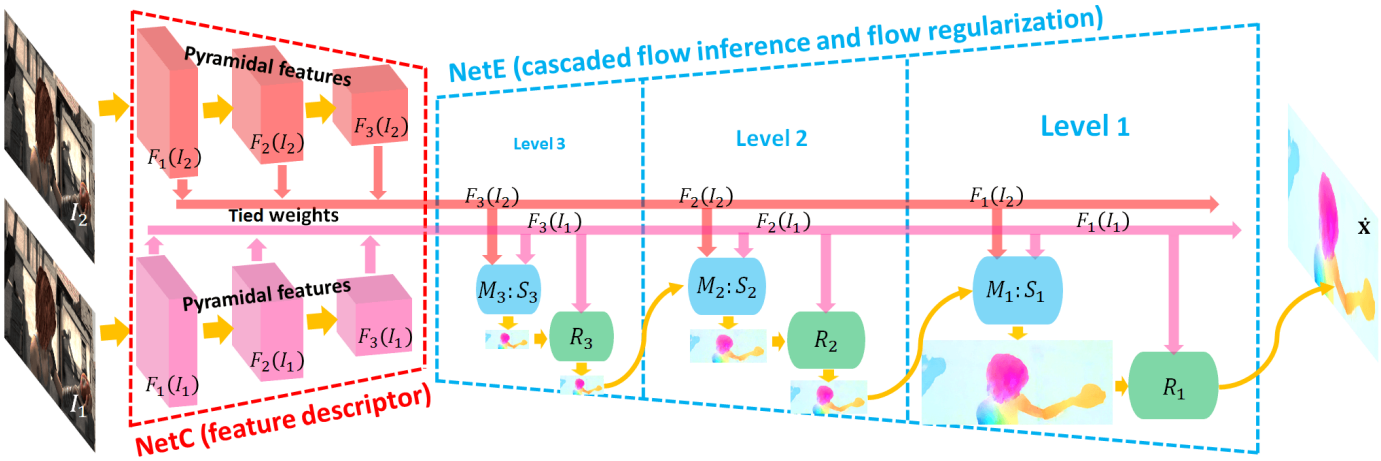

Fig. 2: The network structure of LiteFlowNet. For the ease of representation, only a design of 3-level pyramid is shown. Given an image pair (I_1 and I_2), NetC generates two pyramids of high-level features ($\mathcal{F}_k(I_1)$ in pink and $\mathcal{F}_k(I_2)$ in red, $k \in [1, 3]$). NetE yields multi-scale flow fields such that each of them is generated by a cascaded flow inference module M:S (in blue color, including a descriptor matching unit M and a sub-pixel refinement unit S) and a regularization module R (in green color). Flow inference and regularization modules correspond to data fidelity and regularization terms in conventional energy minimization methods respectively.

sub-networks as NetC and NetE respectively. NetC transforms any given image pair respectively into two pyramids of multi-scale high-dimensional features. NetE consists of cascaded flow inference and regularization modules that estimate optical flow fields from low to high spatial resolutions.

Pyramidal Feature Extraction. As shown in Figure 2, NetC is a two-stream sub-network in which the filter weights are shared across the two streams. Each of them functions as a *pyramidal feature descriptor* that transforms an image I to a pyramid of multi-scale high-dimensional features $\{\mathcal{F}_k(I)\}$ from the highest spatial resolution ($k = 1$) to the lowest spatial resolution ($k = L$). The pyramidal features are generated by stride-1 and stride-s convolutions with the reduction of spatial resolution by a factor s down the inverted pyramid. In the following, we omit the subscript k that indicates the level of pyramid for brevity. We use $\mathcal{F}_i$ to represent CNN features for I_i. When we discuss the operations in a pyramid level, the same operations are applicable to other levels.

We use the design principle that high-resolution feature maps require a large receptive field for convolutional processing. For every decrement of two pyramid levels (down the inverted pyramid), we assign a smaller receptive field than the previous level. Suppose we use a 6-level pyramidal feature extraction, the receptive field sizes are set to 7, 7, 5, 5, 3, and 3. Since receptive field size across convolution layers can be accumulated, we improve the computational efficiency by replacing a large-kernel convolution layer with multiple small-kernel convolution layers. Except a 7×7 kernel is used at the first convolution layer in NetC, 3×3 kernels are used for the subsequent layers and the number of convolution layers are set to 3, 2, 2, 1, and 1. More details about the network architecture can be found in the Appendix.

Feature Warping. At each pyramid level, a flow field is inferred from high-level features $\mathcal{F}_1$ and $\mathcal{F}_2$ of images I_1 and I_2. Flow inference becomes more challenging if I_1 and I_2 are captured far away from each other because a correspondence needs to be searched in a large area. With the motivation of *image warping* used in conventional methods [4], [5] and recent CNNs [1], [11] for addressing large-displacement flow, we propose to reduce feature-space distance between $\mathcal{F}_1$ and $\mathcal{F}_2$ by *feature warping* (f-warp). Specifically, $\mathcal{F}_2$ is warped towards $\mathcal{F}_1$ by f-warp via flow estimate $\dot{\mathbf{x}}$ to $\widetilde{\mathcal{F}}_2(\mathbf{x}) \triangleq \mathcal{F}_2(\mathbf{x} + \dot{\mathbf{x}}) \sim \mathcal{F}_1(\mathbf{x})$. This allows our network to infer residual flow $\Delta\dot{\mathbf{x}}$ between $\mathcal{F}_1$ and $\widetilde{\mathcal{F}}_2$ that has smaller flow magnitude (more details in Section 3.1) but not the complete flow field $\dot{\mathbf{x}}$ that is more difficult to infer. Unlike conventional methods, f-warp is performed on high-level CNN features but not on images. This makes our network more powerful and efficient in addressing the optical flow problem. To allow end-to-end training, $\mathcal{F}$ is interpolated to $\widetilde{\mathcal{F}}$ for any sub-pixel displacement $\dot{\mathbf{x}}$ as follows:

$$\widetilde{\mathcal{F}}(\mathbf{x}) = \sum_{\mathbf{x}_s^i \in N(\mathbf{x}_s)} \mathcal{F}(\mathbf{x}_s^i) \left(1 - |x_s - x_s^i|\right) \left(1 - |y_s - y_s^i|\right), \tag{1}$$

where $\mathbf{x}_s = \mathbf{x} + \dot{\mathbf{x}} = (x_s, y_s)^\top$ denotes the source coordinates in the input feature map $\mathcal{F}$ that defines the sample point, $\mathbf{x} = (x, y)^\top$ denotes the target coordinates of the regular grid in the interpolated feature map $\widetilde{\mathcal{F}}$, and $N(\mathbf{x}_s)$ denotes the four pixel neighbors of $\mathbf{x}_s$. The above bilinear interpolation allows back-propagation during training as its gradients can be efficiently computed [36].

3.1 Cascaded Flow Inference

At each pyramid level of NetE, pixel-by-pixel matching of high-level feature vectors across a given image pair yields a coarse flow estimate. A subsequent refinement on the coarse flow further improves it to sub-pixel accuracy. The use of cascaded flow inference is novel in the literature.

First Flow Inference – Descriptor Matching. Point correspondence between I_1 and I_2 is established through computing the dot product of high-level feature vectors in individual pyramidal features $\mathcal{F}_1$ and $\mathcal{F}_2$ as follows:

$$c(\mathbf{x}, \mathbf{d}) = \mathcal{F}_1(\mathbf{x}) \cdot \mathcal{F}_2(\mathbf{x} + \mathbf{d})/N, \tag{2}$$

where c is the matching cost between point $\mathbf{x}$ in $\mathcal{F}_1$ and point $\mathbf{x} + \mathbf{d}$ in $\mathcal{F}_2$, $\mathbf{d} \in \mathbb{Z}^2$ is the displacement vector from $\mathbf{x}$, and N is the length of the feature vector. The x- and y-components of $\mathbf{d}$ are bounded by $\pm D$ and $D \in \mathbb{Z}_+$. A cost volume C is built

by aggregating all the matching costs $c(\mathbf{x}, \mathbf{d})$ into a 3D grid. At pyramid level k, the dimension of C is $\frac{H}{2^{k-1}} \times \frac{W}{2^{k-1}} \times (2D+1)$ for an image pair with size $H \times W$.

Unlike the conventional construction of cost volume [1], [10], we reduce the computational burden raised in three ways:

1) *Multi-Level Short Searching Range*: Matching of feature vectors between $\mathcal{F}_1$ and $\mathcal{F}_2$ is performed within a short searching range at every pyramid level instead of using a long searching range only at the highest spatial-resolution level.
2) *Feature Warping*: We reduce feature-space distance between $\mathcal{F}_1$ and $\mathcal{F}_2$ prior constructing the cost volume by warping $\mathcal{F}_2$ towards $\mathcal{F}_1$ using our proposed f-warp layer through flow estimate[2] $\dot{\mathbf{x}}$ from previous level.
3) *Spare Cost Volume*: We perform matching only at the sampled positions in the pyramid levels of high spatial resolution. The sparse cost volume is interpolated in the spatial dimension to fill the missed matching costs for the unsampled positions.

The first two techniques effectively reduce the searching space needed, while the third technique reduces the frequency of matching per pyramid level. This in turn causes a speed-up in constructing the cost volume.

In the descriptor matching unit M, a residual flow $\Delta\dot{\mathbf{x}}_m$ between $\mathcal{F}_1(\mathbf{x})$ and $\mathcal{F}_2(\mathbf{x} + s\dot{\mathbf{x}}^{\uparrow s})$ is inferred from the associated cost volume C as illustrated in Figure 3. A complete flow field $\dot{\mathbf{x}}_m$ is computed as follows:

$$\dot{\mathbf{x}}_m = \underbrace{M\big(C(\mathcal{F}_1, \widetilde{\mathcal{F}}_2; D)\big)}_{\Delta\dot{\mathbf{x}}_m} + s\dot{\mathbf{x}}^{\uparrow s}. \tag{3}$$

Second Flow Inference – Sub-Pixel Refinement. Since the cost volume in descriptor matching unit is aggregated by measuring pixel-by-pixel correlation, flow estimate $\dot{\mathbf{x}}_m$ from the previous inference is only up to pixel-level accuracy. We introduce the second flow inference in the wake of descriptor matching as shown in Figure 3. It aims to refine the pixel-level flow field $\dot{\mathbf{x}}_m$ from the descriptor matching unit to sub-pixel accuracy. This prevents erroneous flows being amplified by upsampling and passing to the next pyramid level. Specifically, $\mathcal{F}_2$ is warped to $\widetilde{\mathcal{F}}_2$ via flow estimate $\dot{\mathbf{x}}_m$. Sub-pixel refinement unit S yields a more accurate flow field $\dot{\mathbf{x}}_s$ by minimizing feature-space distance between $\mathcal{F}_1$ and $\widetilde{\mathcal{F}}_2$ through a computing residual flow $\Delta\dot{\mathbf{x}}_s$ for correcting $\dot{\mathbf{x}}_m$ as follows:

$$\dot{\mathbf{x}}_s = \underbrace{S\big(\mathcal{F}_1, \widetilde{\mathcal{F}}_2, \dot{\mathbf{x}}_m\big)}_{\Delta\dot{\mathbf{x}}_s} + \dot{\mathbf{x}}_m. \tag{4}$$

3.2 Flow Regularization

Cascaded flow inference resembles the role of data fidelity in conventional minimization methods. However using data term alone, vague flow boundaries and undesired artifacts commonly exist in flow field [15], [17]. To tackle this problem, we propose to use a *feature-driven local convolution* (f-lcon) to regularize the flow field from cascaded flow inference. The operation of f-lcon is well-governed by the Laplacian formulation of diffusion of pixel values [37] (see Section 4.2 for more details). In contrast to *local convolution* (lcon) used in conventional CNNs [18], f-lcon is more generalized. Not only is a distinct filter used for each position of a feature map, but the filter is adaptively constructed for individual flow patches.

2. $\dot{\mathbf{x}}$ from previous level needs to be upsampled in spatial resolution (denoted by "$\uparrow s$") and magnitude (multiplied by a scalar s) to $s\dot{\mathbf{x}}^{\uparrow s}$ for matching the spatial resolution of the pyramidal features at the current level.

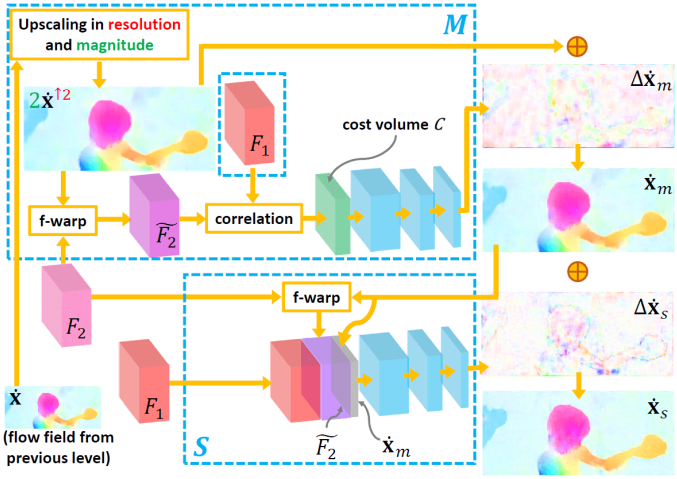

Fig. 3: A cascaded flow inference module M:S in NetE. It consists of a descriptor matching unit M and a sub-pixel refinement unit S. In M, f-warp transforms high-level feature $\mathcal{F}_2$ to $\widetilde{\mathcal{F}}_2$ via upscaled flow field $2\dot{\mathbf{x}}^{\uparrow 2}$ estimated at previous pyramid level. In S, $\mathcal{F}_2$ is warped by $\dot{\mathbf{x}}_m$ from M. Residual flow $\Delta\dot{\mathbf{x}}_m$ is inferred from the cost volume C and $\Delta\dot{\mathbf{x}}_s$ is used to correct $\dot{\mathbf{x}}_m$ due to pixel-level cost aggregation. In comparison to residual flow $\Delta\dot{\mathbf{x}}_m$, more flow adjustment exists on flow boundaries in $\Delta\dot{\mathbf{x}}_s$.

Consider a general case, a vector-valued feature F that has to be regularized has C channels and a spatial dimension $M \times N$. Define $\mathbf{G} = \{g\}$ as the set of filters used in f-lcon layer. The operation of f-lcon to F can be formulated as follow:

$$f_g(x, y, c) = g(x, y, c) * f(x, y, c), \tag{5}$$

where "$*$" denotes convolution, $f(x, y, c)$ is a $w \times w$ patch centered at position (x, y) of channel c in F, $g(x, y, c)$ is the corresponding $w \times w$ regularization filter, and $f_g(x, y, c)$ is a scalar output for $\mathbf{x} = (x, y)^\top$ and $c = 1, 2, ..., C$. To be specific for regularizing flow field $\dot{\mathbf{x}}_s$ from the cascaded flow inference, we replace F to $\dot{\mathbf{x}}_s$. Flow regularization module R is defined as follows:

$$\dot{\mathbf{x}}_r = R(\dot{\mathbf{x}}_s; \mathbf{G}). \tag{6}$$

The f-lcon filters need to be specialized for smoothing flow field. It should behave as an averaging filter if the variation of flow vectors over the patch is smooth. It should also not over-smooth flow field across flow boundary. To this end, we define a feature-driven CNN distance metric $\mathcal{D}$ that estimates local flow variation using pyramidal feature $\mathcal{F}_1$, flow field $\dot{\mathbf{x}}_s$ from the cascaded flow inference, and occlusion probability map[3] O. In summary, $\mathcal{D}$ is adaptively constructed by a CNN unit R_D as follows:

$$\mathcal{D} = R_{\mathcal{D}}(\mathcal{F}_1, \dot{\mathbf{x}}_s, O). \tag{7}$$

With the introduction of feature-driven distance metric $\mathcal{D}$, each filter g of f-lcon is constructed as follows:

$$g(x, y, c) = \frac{\exp(-\mathcal{D}(x, y, c)^2)}{\sum_{(x_i, y_i) \in N(x, y)} \exp(-\mathcal{D}(x_i, y_i, c)^2)}, \tag{8}$$

3. We use L_2 brightness error $||I_2(\mathbf{x} + \dot{\mathbf{x}}) - I_1(\mathbf{x})||_2$ between the warped second image and the first image as the occlusion probability map.

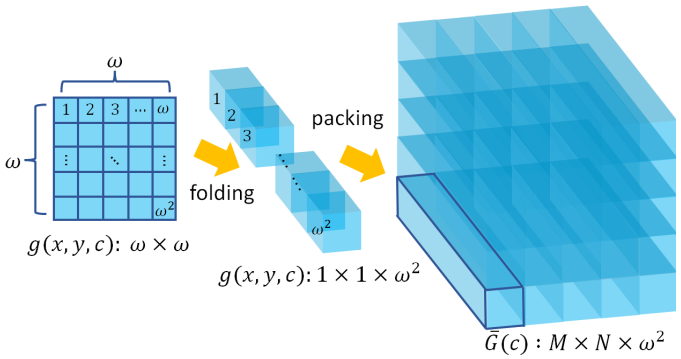

Fig. 4: Folding and packing of f-lcon filters $\{g\}$. The (x, y)-entry of a 3D tensor $\bar{G}(c)$ is a 3D column with size $1 \times 1 \times w^2$. It corresponds to the unfolded $w \times w$ f-lcon filter $g(x, y, c)$ to be applied at position (x, y) of channel c in vector-valued feature F.

where $N(x, y)$ denotes the neighborhood containing $\omega \times \omega$ pixels centered at position (x, y).

Here, we provide a mechanism to perform f-lcon efficiently. For a C-channel input F, we use C tensors $\bar{G}(1), ..., \bar{G}(C)$ to store f-lcon filter set $\mathbf{G}$. As illustrated in Figure 4, each f-lcon filter $g(x, y, c)$ is folded into a $1 \times 1 \times w^2$ 3D column and then packed into the (x, y)-entry of a $M \times N \times w^2$ 3D tensor $\bar{G}(c)$. Same folding and packing operations are also applied to each patch in each channel of F. This results C tensors $\bar{F}(1), ..., \bar{F}(C)$ for F. In this way, Equation (5) can be reformulated to:

$$F_g(c) = \bar{G}(c) \odot \bar{F}(c), \tag{9}$$

where "$\odot$" denotes element-wise dot product between the corresponding columns of the tensors. With the abuse of notation, $F_g(c)$ means the c-th xy-slice of the regularized C-channel feature F_g. Equation (9) reduces the dimension of tensors from $M \times N \times w^2$ (right-hand side in prior to the dot product) to $M \times N$ (left-hand side).

4 Correspondences between LiteFlowNet and Variational Optical Flow

We first provide a brief review for estimating optical flow using variational methods. In the next two sub-sections, we will bridge the correspondences between LiteFlowNet and classical variational methods.

Consider an image sequence $I(\mathbf{x}, t) : \mathbb{R}^3 \rightarrow \mathbb{R}$ with $\mathbf{x} = (x, y)^\top \in \Omega$ over a rectangular spatial domain $\Omega \subset \mathbb{R}^2$ and temporal dimension t. The optical flow field $\dot{\mathbf{x}} : \Omega \rightarrow \mathbb{R}^2$ that is induced by the spatial motion of the scene and/or the camera itself corresponds to the displacement vector field between images I_1 (at $t = 1$) and I_2 (at $t = 2$). The flow field can be estimated by minimizing an energy functional E of the general form [17]:

$$\begin{aligned} E(\dot{\mathbf{x}}) &= E_{dat}(\dot{\mathbf{x}}) + \lambda E_{reg}(\nabla \dot{\mathbf{x}}) \\ &= \int_\Omega \big(e_{data}(\dot{\mathbf{x}}) + \lambda e_{reg}(\nabla \dot{\mathbf{x}})\big) d\mathbf{x}, \end{aligned} \tag{10}$$

where e_{dat} and e_{reg} represent the data and regularization costs respectively, and $\lambda > 0$ is the smoothness weight.

4.1 Data Term

Point correspondences across a pair of images is imposed in the data term of Eq. (10) as a combination of several matching quantities $\{D_i\}$ as follows [14], [17]:

$$E_{dat}(\dot{\mathbf{x}}) = \int_\Omega \sum \gamma_i D_i(I_1, I_2) d\mathbf{x}, \tag{11}$$

where γ_i is the weighting factor for D_i. Two popular matching quantities are image brightness constancy assumption $\Psi\big(|I_2(\mathbf{x} + \dot{\mathbf{x}}) - I_1(\mathbf{x})|^2\big)$ [2] and gradient constancy assumption $\Psi\big(|\nabla I_2(\mathbf{x} + \dot{\mathbf{x}}) - \nabla I_1(\mathbf{x})|^2\big)$ [4], where Ψ is a robust penalty function. Other higher-order constancy data terms are also popular [5]. The contributions of different matching quantities need to be compromised by using appropriate weighting factors [13], [14]. It is also necessary to maintain differentiability of the data and regularization (Section 4.2) term because Eq. (10) needs to be solved using the Euler-Lagrange equation.

In comparison to conventional methods, we do not explicitly define those matching quantities $\{D_i\}$. We train NetC of LiteFlowNet as a CNN-based *pyramidal feature descriptor* $\mathcal{F}(I) : \mathbb{R}^2 \rightarrow \mathbb{R}^N$ that transforms a given image pair (I_1, I_2) respectively into two pyramids of multi-scale high-dimensional features. With the introduction of the feature descriptor, the *cascaded flow inference* in NetE that has been presented in Section 3.1 is trained to solve for the minimization of the difference between the high-level features $\mathcal{F}_2(\mathbf{x} + \dot{\mathbf{x}})$ of I_2 and $\mathcal{F}_1(\mathbf{x})$ of I_1 by computing the optical flow $\dot{\mathbf{x}}$ between $\mathcal{F}(I_1)$ and $\mathcal{F}(I_2)$. In other words, our cascaded flow inference resembles the role of data term in variational methods.

4.2 Regularization Term

Flow field that is merely computed by data fidelity is fragile to outliers. Energy functional is often augmented to enforce dependency between neighboring flow vectors [2]. Regularization of a vector field can be viewed as diffusion of pixel values [37]. By applying the Euler-Lagrange equation to Eq. (10), the regularization component is given by:

$$\text{div}\left(\partial_{\nabla \dot{\mathbf{x}}} E_{reg}(\nabla \dot{\mathbf{x}})\right) = \text{div}(\mathbf{D} \nabla \dot{\mathbf{x}}), \tag{12}$$

where $\mathbf{D}$ is a 2×2 diffusion tensor. The above divergence formulation can also be rewritten into an oriented Laplacian form as follows:

$$\text{div}(\mathbf{D} \nabla \dot{\mathbf{x}}) = \text{trace}(\mathbf{T} \mathbf{H}_i), i = 1, 2, \tag{13}$$

where $\mathbf{H}_i$ is the Hessian matrix of the i-th vector component of the flow field and $\mathbf{T}$ is a 2×2 tensor. The solution of Eq. (13) is given by:

$$\dot{\mathbf{x}} = K(\mathbf{T}) * \dot{\mathbf{x}}', \tag{14}$$

where K is a 2D oriented Gaussian kernel (the exact structure of K depends on $\mathbf{D}$ used in E_{reg}) and $\dot{\mathbf{x}}'$ is the intermediate flow field generated from the data term [38]. In other words, enforcing smoothness constraint on the flow field is equivalent to applying a convolution with a 2D oriented Gaussian kernel to the intermediate flow field generated by the data term.

Unlike the smoothing kernel in Eq. (14) that requires engineered regularizing structure, we design a *feature-driven local convolution* (f-lcon) filter $\mathbf{G} = \{g\}$ that regularizes each flow vector differently in the flow field by adapting f-lcon kernel to the pyramidal feature $\mathcal{F}$ generated by NetC, intermediate flow field $\dot{\mathbf{x}}'$

from the data term, and occlusion probability map O. Our feature-driven flow regularization is defined as follows:

$$\dot{\mathbf{x}} = g\left(\mathcal{F}(I_1(\mathbf{x})), \dot{\mathbf{x}}', O(\mathbf{x})\right) * \dot{\mathbf{x}}'. \tag{15}$$

The flow regularization module R in NetE that performs the above feature-driven flow smoothing operation has been presented in Section 3.2. By replacing the intermediate flow field $\dot{\mathbf{x}}'$ to flow field $\dot{\mathbf{x}}_s$ generated from the cascaded flow inference, Eq. (15) corresponds to Eq. (5). This concludes that our feature-driven regularization resembles the role of regularization term in variational methods.

5 Relationship between Flow Inference in LiteFlowNet and Basis Representation of Optical Flow

The parameterized models of image motion [24], [27], [29] use a linear combination of K basis vectors $\{\mathbf{m}_i \in \mathbb{R}^{2mn}\}$ to approximate image motion $\dot{\mathbf{x}}$ within a $m \times n$ image patch as follows:

$$\dot{\mathbf{x}}_{vec} = \textstyle\sum_{i=1}^{K} a_i \mathbf{m}_i, \tag{16}$$

where $\dot{\mathbf{x}}_{vec} \in \mathbb{R}^{2mn}$ is the vectorized flow field of $\dot{\mathbf{x}}$ by packing all the x- and y-components of $\dot{\mathbf{x}}$ into a single vector and a_is are the flow coefficients to be estimated.

The above basis representation is closely related to the cascaded flow inference in NetE. At pyramid level k, the second to the final layers of the descriptor matching and sub-pixel flow refinement units in the cascaded flow inference can be represented by the following:

$$F^{n-1} = \sigma\left(\mathbf{W}^{n-1} * F^{n-2} + \mathbf{b}^{n-1}\right), \tag{17.1}$$

$$\Delta\dot{\mathbf{x}}_k = \mathbf{W}^{N} * F^{N-1} + \mathbf{b}^{N}, \tag{17.2}$$

where "$*$" denotes a convolutional operator, $n = 2, 3, ..., N$, and N is the total number of convolutional layers used. Furthermore, F^i and $\mathbf{W}^i$ represents the set of feature maps and the set of convolution filters used at the i-th layer respectively. A trainable bias b is added to each feature map after the convolutional operation. We denote a set of bias scalars as $\mathbf{b}$. Each convolutional layer is followed by an activation function (σ) for non-linear mapping unless otherwise specified.

Suppose F^{N-1} in Eq. (17.2) is a C-channel vector-valued feature map, the equation can be re-written into the expanded form as the following:

$$u_k - u_{k-1} = \textstyle\sum_{i=1}^{C}\left(\mathbf{w}_i^u * F_i + b_i^u\right), \tag{18.1}$$

$$v_k - v_{k-1} = \textstyle\sum_{i=1}^{C}\left(\mathbf{w}_i^v * F_i + b_i^v\right), \tag{18.2}$$

where $\Delta\dot{\mathbf{x}}_k = (u_k - u_{k-1}, v_k - v_{k-1})^\top$, $\mathbf{W} = \{\mathbf{w}_i^u, \mathbf{w}_i^v\}$, $F = \{F_i\}$, and $\mathbf{b} = \{b_i^u, b_i^v\}$ (superscripts N and $N-1$ are removed for brevity). If we consider the residual flow $\Delta\dot{\mathbf{x}}_k$ in Eq. (18) as the flow field that we need to estimate although it is not the full flow, then the vectorized combination of $\mathbf{w}_i^u * F_i + b_i^u$ and $\mathbf{w}_i^v * F_i + b_i^v$ is equivalent to vector $a_i\mathbf{m}_i$ in Eq. (16). In particular, filter weights $\mathbf{W}$ and feature maps F in Eq. (17.2) correspond to the basis flow fields $\{\mathbf{m}_i\}$ and flow coefficients $\{a_i\}$ in Eq. (16), respectively. Furthermore, the computation of flow coefficients in the cascaded flow inference is governed by Eq. (17.1). This concludes that the parameterized model of image motion is well-encapsulated in our cascaded flow inference.

6 Experiments

6.1 LiteFlowNet

Network Details. In LiteFlowNet, NetC generates 6-level pyramidal features and NetE predicts flow fields for levels 6 to 2. Flow field at level 2 is upsampled to the same resolution at level 1 as the given image pair. We set the maximum searching radius for constructing cost volumes to 3 pixels for levels 6 to 4 and 6 pixels for levels 3 to 2. Matching is performed at each position across two pyramidal features to form a cost volume, except for levels 3 to 2 that it is performed at a regularly sampled grid (using a stride of 2) to form a sparse cost volume. All convolution layers use 3×3 filters, except the first layer in NetC uses 7×7 filters, each last layer in descriptor matching M, sub-pixel refinement S, and flow regularization R uses 5×5 filters for levels 4 to 3 and 7×7 filters for level 2. Each convolution layer is followed by a leaky rectified linear unit layer, except f-lcon and the last layer in M, S and R networks. More network details can be found in the Appendix.

Training Details. We train our LiteFlowNet stage-wise by the following steps: 1) NetC and M_6:S_6 of NetE are trained for 300k iterations. 2) R_6 together with the trained network in step 1 are trained for 300k iterations. 3) For levels $k \in [5, 2]$, M_k:S_k followed by R_k is added into the trained network each time. The new network cascade is trained for 200k iterations, except the last cascade is trained for 300k iterations. The new filter weights in level k are initialized from the previous level $k-1$. Learning rates are initially set to 1e-4, 5e-5, and 4e-5 for levels 6 to 4, 3 and 2 respectively. We reduce it by a factor of 2 starting at 120k, 160k, 200k, and 240k iterations. We use the same batch size (Chairs: 8 and Things3D: 4), data set resolution (randomly cropped Chairs: 448×320 and Things3D: 768×384), loss weights (levels 6 to 2: 0.32, 0.08, 0.02, 0.01, 0.005), training loss (L_2 flow error), Adam optimization ($\gamma = 0.5$, weight decay = 0.0004), data augmentation (including noise injection), scaled ground-truth flow (by a factor of $\frac{1}{20}$), and training schedule[4] (Chairs [10] $\rightarrow$ Things3D [39]) as FlowNet2 [1]. We use a training loss for every inferred flow field. When fine-tuning on Things3D, it is trained for 500k iterations. Learning rate is set to 3e-6 and is reduced by half starting at 200k iterations for every increment of 100k iterations. We denote **LiteFlowNet-pre** and **LiteFlowNet** as the networks trained on Chairs and Chairs $\rightarrow$ Things3D, respectively. After fine-tuning on Things3D, we use the generalized Charbonnier function $\rho(x) = (x^2 + \epsilon^2)^q$ as the robust training loss for further fine-tuning on subsequent datasets.

6.2 LiteFlowNet2

We improve the network architecture of LiteFlowNet [12] with the following motivations and evolve our earlier model to LiteFlowNet2.

Pyramid Level. We analyze the flow accuracy of LiteFlowNet trained on Chairs $\rightarrow$ Things3D in terms of average end-point error (AEE) and computation time consumed on each pyramid level of the 6-level LiteFlowNet and present the results in Table 2. We find that the improvement of optical flow accuracy is not significant when comparing the AEE at level 3 to that at level 2. On the contrary, 60% of the total runtime spent on the flow inference at

4. We excluded a small amount of training data in Things3D undergoing extremely large flow displacement as advised by the authors (https://github.com/lmb-freiburg/flownet2/issues).

TABLE 2: AEE at different pyramid levels and Runtime spent on different components of LiteFlowNet.

	NetC	NetE				
Level	-	6	5	4	3	2
Sintel Clean AEE	-	5.41	3.85	3.03	2.65	2.48
Runtime (ms)	14.36	1.69	2.03	4.88	13.06	52.51

TABLE 3: AEE on the Chairs testing set. Models are trained on the Chairs training set.

FlowNetS	FlowNetC	SPyNet	LiteFlowNetX-pre	LiteFlowNet-pre
2.71	2.19	2.63	2.25	**1.57**

level 2. In LiteFlowNet2, we improve the computational efficiency by restricting the number of flow inferences to four pyramid levels (from level 6 to 3) instead of five pyramid levels.

Network Depth. Since LiteFlowNet2 has one fewer flow inference level than LiteFlowNet, optical flow accuracy will be reduced to certain extent. In order to compensate the loss, we add two convolution layers (with 128 and 96 output channels) between the 128- and 64-channel convolution layers to each cascaded flow inference in NetE.

Pseudo Flow Inference and Regularization. As shown in Table 2, the computation time increases exponentially with the flow accuracy across different levels of LiteFlowNet. We optionally address the problem of inefficient computation by introducing a simplified flow inference (without descriptor matching) and regularization at level 2. The pseudo network is similar to the original one, except we remove all the layers before the last layer in flow inference and f-lcon layer in regularization, and replace the removed layers by feedforwarding the upsampled features respectively from the layer prior the last layer in flow inference and regularization at level 3. The pseudo network is only used for the models fine-tuned on KITTI because we have experienced that the improvement is not significant on Sintel. This is because KITTI is a more challenging and real-world benchmark, the flow accuracy needs to be improved by using one more pyramidal flow inference.

Training Details. We train **LiteFlowNet2** using the same training procedure as LiteFlowNet except for a few differences. Learning rates are initially set to 1e-4 and 6e-5 for levels 6 to 4 and 3 (3 to 2 if pseudo network is used) respectively. At the last flow regularization, the flow field is upsampled to the same resolution as the image pair and we introduce an additional training loss with a loss weight 6.25e-4.

6.3 Results

We compare LiteFlowNet, LiteFlowNet2, and its variants to the state-of-the-art methods on public benchmarks including FlyingChairs (Chairs) [10], Sintel clean and final passes [43], KITTI 2012 [44], and KITTI 2015 [45].

FlyingChairs. We first compare the intermediate results of different well-performing networks trained on Chairs alone in Table 3. Average end-point error (AEE) is reported. LiteFlowNet-pre outperforms the compared networks. No intermediate result is available for FlowNet2 [1] as each cascade is trained on the Chairs → Things3D schedule individually. Since FlowNetC, FlowNetS (variants of FlowNet [10]), and SPyNet [11] have fewer parameters than FlowNet2 and the later two models do not perform feature matching, we construct a small-size counterpart **LiteFlowNetX-pre** for a fair comparison by removing the matching part and shrinking the model sizes of NetC and NetE by about 4 and 5 times, respectively. Despite that LiteFlowNetX-pre is 43 and 1.33 times smaller than FlowNetC and SPyNet, respectively, it still outperforms these networks and is on par with FlowNetC that uses explicit feature matching.

MPI Sintel. In Table 4, LiteFlowNetX-pre outperforms FlowNetS [10], FlowNetC [10], and SPyNet [11] that are trained on Chairs on all cases. LiteFlowNet, trained on the Chairs → Things3D schedule, performs better than LiteFlowNet-pre as expected. LiteFlowNet also outperforms SPyNet, FlowNet2-S [1], and FlowNet2-C [1]. With the improved architecture and training protocol, LiteFlowNet2 outperforms its predecessor LiteFlowNet, PWC-Net [31], and is on par with EpicFlow [23]. We also fine-tuned LiteFlowNet on a mixture of Sintel clean and final training data (**LiteFlowNet-ft**) using the generalized Charbonnier loss with the settings $\epsilon^2 = 0.01$ and $q = 0.4$. We randomly crop 768 × 384 patches and use a batch size of 4. No noise augmentation is performed but we introduce image mirroring to improve the diversity of the training set. Learning rate is set to 5e-5 and the training schedule is similar to the training of LiteFlowNet on Things3D except it is trained for 600k and is re-trained with a reduced learning rate for a reduced number of iterations. LiteFlowNet-ft outperforms FlowNet2-ft-sintel [1] and EpicFlow [23] for Sintel final testing set. Despite DC Flow [41] (a hybrid method consists of CNN and post-processing) performs better than LiteFlowNet, its GPU runtime requires several seconds that makes it formidable in many applications. For fine-tuning LiteFlowNet2, we improve the diversity of the training set by using a mixture of Sintel and KITTI (**LiteFlowNet2-ft**) for a batch size of 4 containing 2 of image pairs from each of the training sets. LiteFlowNet2-ft outperforms all the compared methods on Sintel Clean testing set. It is the second best method on Sintel Final testing set. Figure 5 shows some examples of flow fields on the training and testing sets of Sintel. End-point error (EPE) on the training sets is provided. Since LiteFlowNet(-ft) and LiteFlowNet2(-ft) have flow regularization layer, sharper flow boundaries and lesser artifacts can be observed in the generated flow fields.

KITTI. LiteFlowNet consistently performs better than LiteFlowNet-pre especially on KITTI 2015 as shown in Table 4. It also outperforms SPyNet [11], FlowNet2-S [1], and FlowNet2-C [1]. Its successor, LiteFlowNet2, outperforms FlowNet2 [1], LiteFlowNet, and PWC-Net [31] as well. We also fine-tuned LiteFlowNet and LiteFlowNet2 on a mixture of KITTI 2012 and KITTI 2015 training data (**LiteFlowNet-ft** and **LiteFlowNet2-ft**) using the same augmentation and training schedule as the case of Sintel except that we reduced the amount of augmentation for spatial motion to fit the driving scene. The height of KITTI dataset is less than that of Sintel about 100 pixels. We randomly crop 896 × 320 patches to maintain a similar patch area as Sintel and use a batch size of 4. After fine-tuning, LiteFlowNet and LiteFlowNet2 generalize well to real-world data. LiteFlowNet-ft outperforms FlowNet2-ft-kitti [1] and PWC-Net-ft [31]. With the improved architecture and training protocol, LiteFlowNet2-ft outperforms LiteFlowNet and PWC-Net-ft [31], and is on par or outperforms PWC-Net+ [42]. It is ranked the first achieving 2.72% in Out-Noc (percentage of erroneous pixels in total) on KITTI 2012 and achieving 7.20% in Fl-fg (percentage of outliers averaged only over foreground regions) on KITTI

TABLE 4: AEE of different methods. The values in parentheses are the results of the networks on the data they were trained on, and hence are not directly comparable to the others. Out-Noc: Percentage of erroneous pixels defined as EPE >3 pixels in non-occluded areas. Fl-all: Percentage of outliers averaged over all pixels. Inliers are defined as EPE <3 pixels or <5%. The best number for each category is highlighted in bold. (Note: [1]The values are reported from [1]. [2]We re-trained the model using the code provided by the authors. [3,4,5]The values are computed using the trained models provided by the authors. [4]Large discrepancy exists as the authors mistakenly evaluated the results on the disparity dataset. [5]Up-to-date dataset is used. [6]Trained on Driving and Monkaa [39])

	Method	Sintel Clean		Sintel Final		KITTI 2012			KITTI 2015		
		train	test	train	test	train	test	test (Out-Noc)	train	train (Fl-all)	test (Fl-all)
Conventional	LDOF[1] [6]	4.64	7.56	5.96	9.12	10.94	12.4	-	18.19	38.11%	-
	DeepFlow[1] [20]	2.66	5.38	3.57	7.21	4.48	5.8	-	10.63	26.52%	29.18%
	Classic+NLP [40]	4.49	6.73	7.46	8.29	-	7.2	-	-	-	-
	PCA-Layers[1] [29]	3.22	5.73	4.52	7.89	5.99	5.2	-	12.74	27.26%	-
	EpicFlow[1] [23]	2.27	4.12	3.56	6.29	**3.09**	3.8	-	9.27	27.18%	**27.10%**
	FlowFields[1] [35]	**1.86**	**3.75**	**3.06**	**5.81**	3.33	**3.5**	-	**8.33**	**24.43%**	-
Hybrid	Deep DiscreteFlow [33]	-	3.86	-	5.73	-	3.4	-	-	-	21.17%
	Bailer *et al.* [34]	-	**3.78**	-	5.36	-	**3.0**	-	-	-	19.44%
	DC Flow [41]	-	-	-	**5.12**	-	-	-	-	-	**14.86%**
Heavyweight CNN	FlowNetS [10]	4.50	7.42	5.45	8.43	8.26	-	-	-	-	-
	FlowNetS-ft [10]	(3.66)	6.96	(4.44)	7.76	7.52	9.1	-	-	-	-
	FlowNetC [10]	4.31	7.28	5.87	8.81	9.35	-	-	-	-	-
	FlowNetC-ft [10]	(3.78)	6.85	(5.28)	8.51	8.79	-	-	-	-	-
	FlowNet2-S[3] [1]	3.79	-	4.99	-	7.26	-	-	14.28	51.06%	-
	FlowNet2-S re-trained[2]	*3.96*	-	*5.37*	-	*7.31*	-	-	*14.51*	*51.38%*	-
	FlowNet2-C[3] [1]	3.04	-	4.60	-	5.79	-	-	11.49	44.09%	-
	FlowNet2 [1]	**2.02**	**3.96**	**3.54**[4]	6.02	4.01[5]	-	-	10.08[5]	29.99%[5]	-
	FlowNet2-ft-sintel [1]	(1.45)	4.16	(2.19[4])	**5.74**	**3.54**[5]	-	-	**9.94**[5]	**28.02%**[5]	-
	FlowNet2-ft-kitti [1]	3.43	-	4.83[4]	-	(1.43[5])	**1.8**	4.82%	(2.36[5])	(8.88%[5])	**11.48%**
Lightweight CNN	SPyNet [11]	4.12	6.69	5.57	8.43	9.12	-	-	-	-	-
	SPyNet-ft [11]	(3.17)	6.64	(4.32)	8.36	*3.36*[6]	4.1	12.31%	-	-	35.07%
	PWC-Net [31]	2.55	-	3.93	-	4.14	-	-	10.35	33.67%	-
	PWC-Net-ft [31]	(2.02)	4.39	(2.08)	5.04	(1.45)	1.7	4.22%	(2.16)	(9.80%)	9.60%
	PWC-Net_ROB [42]	(1.81)	3.90	(2.29)	4.90	-	-	-	-	-	11.63%
	PWC-Net-ft+ [42]	(1.71)	**3.45**	(2.34)	**4.60**	(0.99)	1.4	3.36%	(1.47)	(7.59%)	**7.72%**
	LiteFlowNetX-pre [12]	3.70	-	4.82	-	6.81	-	-	16.64	36.64%	-
	LiteFlowNetX [12]	3.58	-	4.79	-	6.38	-	-	15.81	34.90%	-
	LiteFlowNet-pre [12]	2.78	-	4.17	-	4.56	-	-	11.58	32.59%	-
	LiteFlowNet [12]	2.48	-	4.04	-	4.00	-	-	10.39	28.50%	-
	LiteFlowNet-ft [12]	(1.35)	4.54	(1.78)	5.38	(1.05)	1.6	3.27%	(1.62)	(5.58%)	9.38%
	LiteFlowNet2	**2.24**	-	**3.78**	-	**3.42**	-	-	**8.97**	**25.88%**	-
	LiteFlowNet2-ft	(1.30)	**3.45**	(1.62)	4.90	(1.00)	**1.4**	**2.72%**	(1.47)	(4.80%)	**7.74%**

TABLE 5: Number of training parameters and runtime. The model for which the runtime is in parentheses is measured using Torch, and hence are not directly comparable to the others using Caffe. (Note: [1]The runtime is longer when comparing to the provided value [31] because it was measured by a faster GPU than ours.)

	Shallow		Deep				
Model	FlowNetC [10]	SPyNet [11]	FlowNet2 [1]	PWC-Net [31]	LiteFlowNetX	LiteFlowNet	LiteFlowNet2
No. of learnable layers	26	35	115	59	69	94	91
No. of parameters (M)	39.16	1.20	162.49	8.75	0.90	5.37	6.42
Runtime (ms)	31.51	(129.83)	121.49	39.63[1]	35.10	88.53	39.69
Frame/second (fps)	31	(8)	8	25	28	12	25

2015 benchmarks using 2-frame input at the time of writing. Without pseudo network, Out-Noc for KITTI 2012 and Fl-fg for KITTI 2015 become 3.07% and 7.73% respectively. Figure 6 shows some examples of flow fields on the training and testing sets KITTI 2012 and KITTI 2012. As in the case for Sintel, LiteFlowNet(-ft) and LiteFlowNet2(-ft) perform the best among the compared methods. Even though LiteFlowNet, LiteFlowNet2, and its variants perform pyramidal descriptor matching in a limited searching range, it yields reliable large-displacement flow fields for real-world data due to the feature warping (f-warp) layer introduced. More analysis will be presented in Section 6.5.

6.4 Runtime and Parameters

We measure runtime of each CNN on a machine equipped with an Intel Xeon E5 2.2GHz and an NVIDIA GTX 1080. Timings are averaged over 100 runs for an image pair of size 1024 × 436. For a fair comparison, we also exclude the reading and writing time as PWC-Net [31]. As summarized in Table 5, LiteFlowNet has **30.3 times** fewer parameters than FlowNet2 [1] and is **1.4 times** faster in runtime. LiteFlowNetX, a variant of LiteFlowNet having a smaller model size and without descriptor matching, has **43.5 times** fewer parameters than FlowNetC [10] and a comparable runtime. It has **1.3 times** fewer parameters than SPyNet [11]. LiteFlowNet2 requires **1.4 and 25.3 times** fewer parameters than PWC-Net (also PWC-Net+ [42]) and FlowNet2 respectively. It is **2.2 and 3.1 times** faster than LiteFlowNet and FlowNet respectively. Its processing frequency can reach up to 25 flow fields per second and is similar to the recent work PWC-Net. LiteFlowNet, LiteFlowNet2 and its variants are currently the most compact CNNs for accurate optical flow estimation.

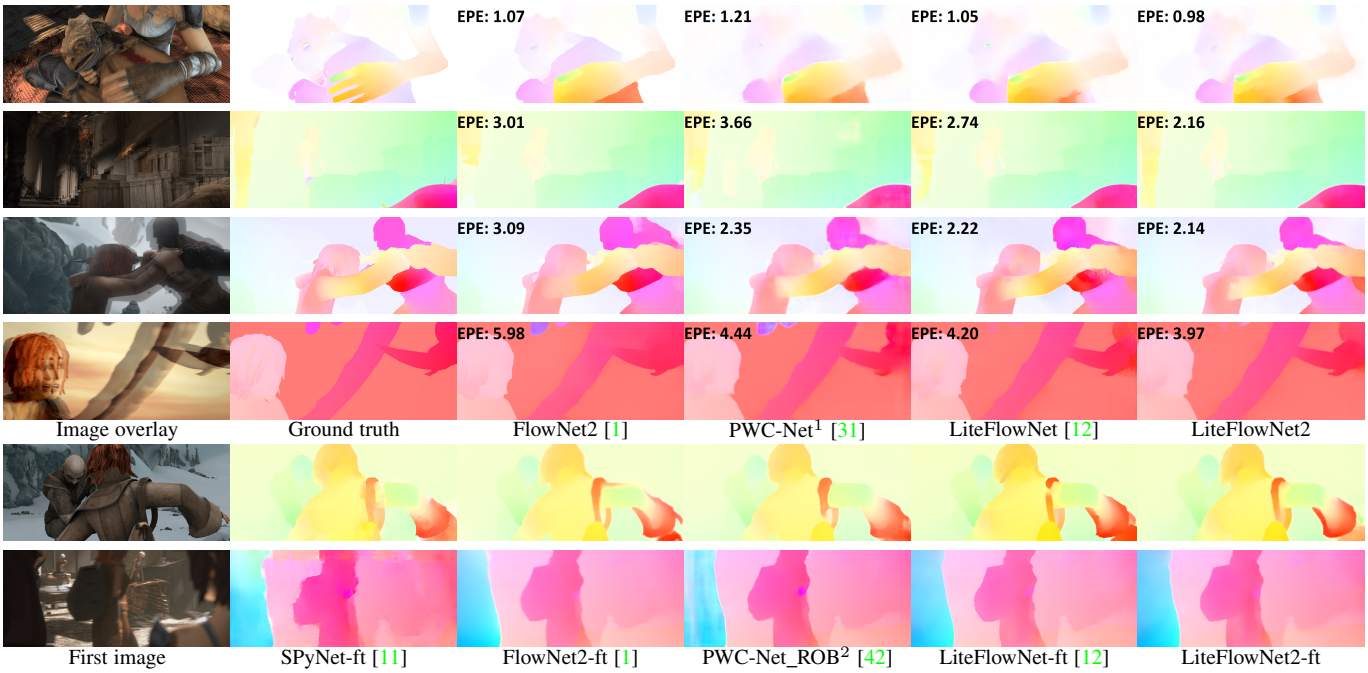

Fig. 5: Examples of flow fields from different methods on Sintel training sets (clean pass: first to second rows, final pass: third to fourth rows) and testing sets (clean pass: fifth row, final pass: last row). Fine details are well preserved and less artifacts can be observed in the flow fields of LiteFlowNet2 and LiteFlowNet2-ft. (Note: [1]Only the trained models with a larger-capacity feature pyramid extractor (footprint: 8.75M vs 9.37M and runtime: 39.63ms vs 41.12ms) of PWC-Net [31] trained on Chairs → Things3D and [2]PWC-Net_ROB [42] trained by fine-tuning on Sintel, KITTI 2015, and HD1K are provided by the authors.)

TABLE 6: AEE of different variants of LiteFlowNet-pre trained on Chairs dataset with some of the components disabled.

Variants	M	MS	WM	WSR	WMS	ALL
Feature **W**arping	✗	✗	✓	✓	✓	✓
Descriptor **M**atching	✓	✓	✓	✗	✓	✓
Sub-pix. Refinement	✗	✓	✗	✓	✓	✓
Regularization	✗	✗	✗	✓	✗	✓
FlyingChairs (train)	3.75	2.70	2.98	1.63	1.82	**1.57**
Sintel clean (train)	4.70	4.17	3.54	3.19	2.90	**2.78**
Sintel final (train)	5.69	5.30	4.81	4.63	4.45	**4.17**
KITTI 2012 (train)	9.22	8.01	6.17	5.03	4.83	**4.56**
KITTI 2015 (train)	18.24	16.19	14.52	13.20	12.32	**11.58**

6.5 Ablation Study

We investigate the role of each component in LiteFlowNet-pre trained on Chairs by evaluating the performance of different variants with some of the components disabled unless otherwise stated. The AEE results are summarized in Table 6 and examples of flow fields are illustrated in Figure 7.

Feature Warping. We consider two variants LiteFlowNet-pre (WM and WMS) and compare them to the counterparts with feature warping disabled (M and MS). Flow fields from M and MS are more vague. Large degradation in AEE is noticed especially for KITTI 2012 (↓ 33%) and KITTI 2015 (↓ 25%). With feature warping (f-warp), pyramidal features that input to flow inference are closer in appearance to each other. This facilitates flow estimation in subsequent pyramid levels by computing residual flows.

Descriptor Matching. We compare the variant WSR without descriptor matching for which the flow inference part is made as deep as that in the unamended LiteFlowNet-pre (ALL). No noticeable difference between the flow fields from WSR and ALL. Since the maximum displacement of the example flow field is not very large (only 14.7 pixels), accurate flow field can still be yielded from WSR. For evaluation covering a wide range of flow displacement (especially large-displacement benchmark, KITTI), degradation in AEE is noticed for WSR. This suggests that descriptor matching is useful in addressing large-displacement flow. We also study the design of combing the cost volume together with the features (`conv_redir`) feedforwarding from the encoder as FlowNetC [10] and FlowNet2 [1] in LiteFlowNet2-pre. However, the AEE of Sintel Final (4.14 vs 4.34) and KITTI 2015 are increased (11.31 vs 11.46).

Sub-Pixel Refinement. The flow field generated from WMS is more crisp and contains more fine details than that generated from WM with sub-pixel refinement disabled. Less small-magnitude flow artifacts (represented by light color on the background) are also observed. Besides, WMS achieves smaller AEE. Since descriptor matching establishes pixel-by-pixel correspondence, sub-pixel refinement is necessary to yield detail-preserving flow fields.

Regularization. In comparison WMS with regularization disabled to ALL, undesired artifacts exist in homogeneous regions (represented by very dim color on the background) of the flow field generated from WMS. Flow bleeding and vague flow boundaries are observed. Degradation in AEE is also noticed. This suggests that the proposed feature-driven local convolution (f-lcon) plays the vital role to smooth flow field and maintain crisp flow boundaries as regularization term in conventional variational methods.

Searching Range. We compare three variants of LiteFlowNet2-pre with different cost-volume settings as shown in Table 7. On

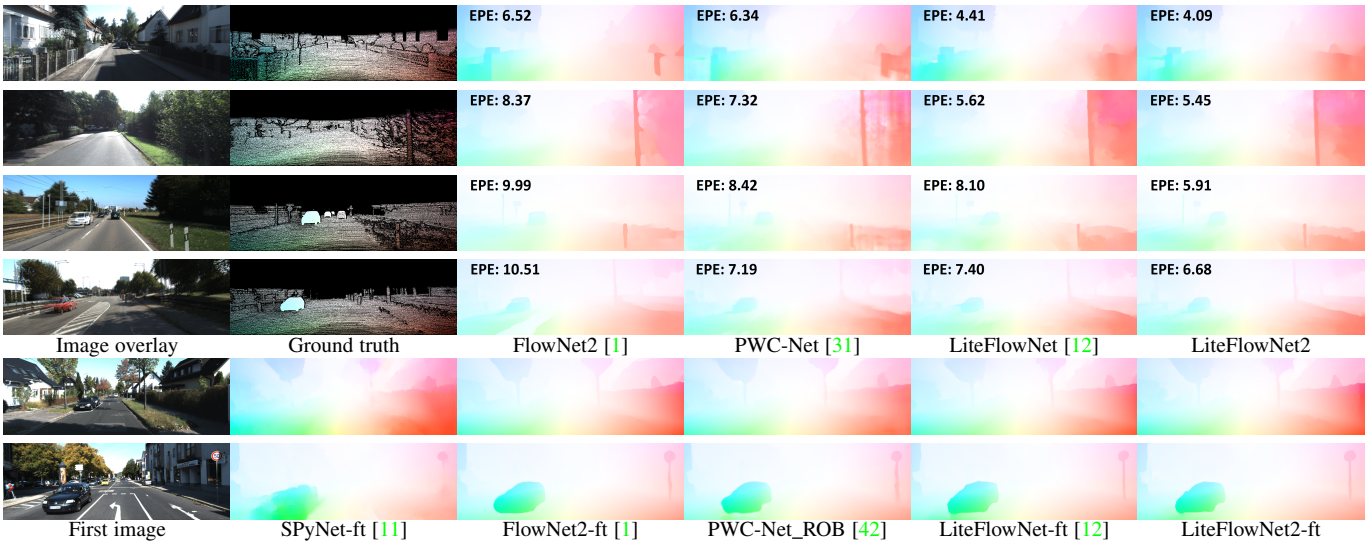

Fig. 6: Examples of flow fields from different methods on the KITTI training sets (2012: first to second rows, 2015: third to fourth row) and testing sets (2012: fifth row, 2015: last row).

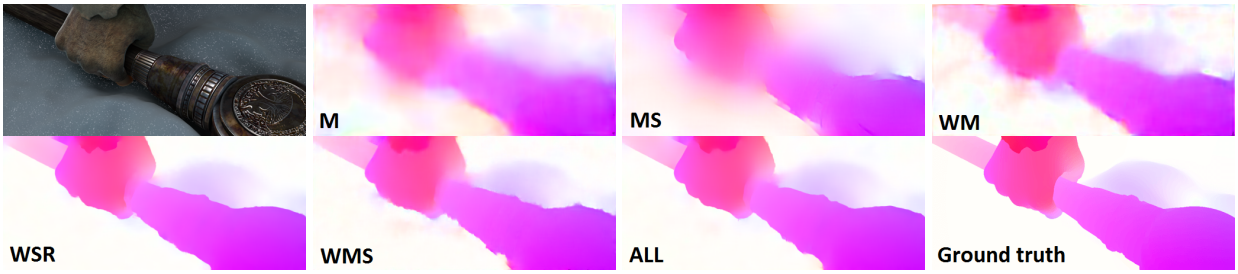

Fig. 7: Examples of flow fields from different variants of LiteFlowNet-pre trained on Chairs with some of the components disabled. LiteFlowNet-pre is denoted as "All". W = Feature **W**arping, M = Descriptor **M**atching, S = **S**ub-Pixel Refinement, R = **R**egularization.

TABLE 7: AEE and runtime of LiteFlowNet2-pre trained on Chairs under different cost-volume settings. The value in parentheses represents another setting.

Searching Range (pixels)	3	3 (6)	4
Stride	1	1 (2)	1
Levels	6 to 3	6 to 4 (3)	6 to 3
Sintel Clean	2.73	2.78	2.71
Sintel Final	4.14	4.14	4.14
KITTI 2012 (train)	4.26	4.11	4.20
KITTI 2015 (train)	11.72	11.31	11.12
Runtime (ms)	41.33	**39.69**	44.33

the whole, a larger searching range leads to a lower AEE. The improvement is more significant on large-displacement benchmark, KITTI. Our design that uses a larger searching range together with a sparse cost volume in a high-resolution pyramid level not only improves flow accuracy but also promotes efficient computation. We choose the second cost-volume setting for our final models due to the shortest computation time.

7 Conclusion

We have reached a milestone in CNN optical flow estimation by developing a lightweight and effective convolutional network through adopting data fidelity and regularization from variational methods. LiteFlowNet uses pyramidal feature extraction, feature warping (f-warp), and multi-level flow inference to break the de facto rule of accurate flow network requiring large model size. To address large-displacement and detail-preserving flows, it exploits short-range matching to generate pixel-level flow field and further improves the estimate to sub-pixel accuracy in each cascaded flow inference. To result crisp flow boundaries, flow field is adaptively regularized through feature-driven local convolution (f-lcon). The evolution of LiteFlowNet creates LiteFlowNet2, that runs 2.2 times faster and attains a better flow accuracy. LiteFlowNet2 outperforms the state-of-the-art FlowNet2 [10] on the public benchmarks while being 3.1 times faster in runtime and 25.3 times smaller in model size. It outperforms or on par with PWC-Net+ [42] while being 1.4 times smaller in model size. With its lightweight, accurate, and fast flow computation, we expect that LiteFlowNet2 can be deployed to many applications such as video processing, motion segmentation, action recognition, SLAM, 3D reconstruction and more.

Appendix

LiteFlowNet2 consists of two compact sub-networks, namely NetC and NetE. NetC is a two-steam sub-network in which the two streams share the same set of filters. The input to NetC is an image pair (I_1, I_2). The network architectures of the 6-level NetC and NetE at pyramid level 5 are provided in Table 8 and Tables 9 to 11, respectively. We use suffixes "M", "S" and "R" to highlight the layers that are used in descriptor matching, sub-pixel refinement,

and flow regularization modules in NetE, respectively. The name of convolution layer is replaced from "`conv`" to "`flow`" to highlight when the output is a flow field.

REFERENCES

[1] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet2.0: Evolution of optical flow estimation with deep networks," *CVPR*, pp. 2462–2470, 2017.

[2] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Arifical Intelligence*, vol. 17, pp. 185–203, 1981.

[3] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," *IJCAI*, pp. 674–679, 1981.

[4] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," *ECCV*, pp. 25–36, 2004.

[5] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert, "Highly accurate optic flow computation with theoretically justified warping," *IJCV*, vol. 67, no. 2, pp. 141–158, 2006.

[6] T. Brox and J. Mailk, "Large displacement optical flow: Descriptor matching in variational motion estimation," *PAMI*, vol. 33, no. 3, pp. 500–513, 2011.

[7] T.-W. Hui and R. Chung, "Determining motion directly from normal flows upon the use of a spherical eye platform," *CVPR*, pp. 2267–2274, 2013.

[8] ——, "Determining shape and motion from non-overlapping multi-camera rig: A direct approach using normal flows," *CVIU*, vol. 117, no. 8, pp. 947–964, 2013.

[9] ——, "Determining shape and motion from monocular camera: A direct approach using normal flows," *PR*, vol. 48, no. 2, pp. 422–437, 2015.

[10] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "FlowNet: Learning optical flow with convolutional networks," *ICCV*, pp. 2758–2766, 2015.

[11] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," *CVPR*, pp. 4161–4170, 2017.

[12] T.-W. Hui, X. Tang, and C. C. Loy, "LiteFlowNet: A lightweight convolutional neural network for optical flow estimation," *CVPR*, pp. 8981–8989, 2018.

[13] L. Xu, J. Jia, and Y. Matsushita, "Motion detail preserving optical flow estimation," *PAMI*, vol. 34, no. 9, pp. 1744–1757, 2012.

[14] T. H. Kim, H. S. Lee, and K. M. Lee, "Optical flow via locally adaptive fusion of complementary data costs," *ICCV*, pp. 2373–2381, 2013.

[15] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof, "Anisotropic Huber-L^1 optical flow," *BMVC*, 2009.

[16] D. Sun, S. Roth, J. Lewis, and M. J. Black, "Learning optical flow," *ECCV*, pp. 83–97, 2008.

[17] H. Zimmer, A. Bruhn, and J. Weickert, "Optic flow in harmony," *IJCV*, vol. 93, no. 3, pp. 368–388, 2011.

[18] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," *CVPR*, pp. 1701–1708, 2014.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," *MICCAI*, pp. 234–241, 2015.

[20] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "DeepFlow: Large displacement optical flow with deep matching," *ICCV*, pp. 500–513, 2013.

[21] J. Lu, H. Yang, D. Min, and M. N. Do, "PatchMatch Filter: Efficient edge-aware filtering meets randomized search," *CVPR*, pp. 1854–1861, 2013.

[22] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *SIGGRAGH*, pp. 83–97, 2009.

[23] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "EpicFlow: Edge-preserving interpolation of correspondences for optical flow," *CVPR*, pp. 1164–1172, 2015.

[24] M. J. Black, Y. Yacoobt, A. D. Jepsont, and D. J. Fleets, "Learning parameterized models of image motion," *CVPR*, pp. 674–679, 1997.

[25] S. Roth and M. J. Black, "Fields of experts: A framework for learning image priors," *CVPR*, pp. 860–867, 2005.

[26] S. Roth and M. Black, "On the spatial statistics of optical flow," *ICCV*, pp. 42–49, 2005.

[27] T. Nir, A. M. Bruckstein, and R. Kimmel, "Over-parameterized variational optical flow," *IJCV*, vol. 76, no. 2, pp. 205–216, 2008.

[28] D. Rosenbaum, D. Zoran, and Y. Weiss, "Learning the local statistics of optical flow," *NIPS*, pp. 2373–2381, 2013.

[29] J. Wulff and M. J. Black, "Efficient sparse-to-dense optical flow estimation using a learned basis and layers," *CVPR*, pp. 120–130, 2015.

[30] S. Zweig and L. Wolf, "Interponet, a brain inspired neural network for optical flow dense interpolation," *arXiv preprint arXiv:1611.09803v3*, 2017.

[31] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Models matter, so does training: An empirical study of CNNs for optical flow estimation," *arXiv preprint arXiv:1809.05571*, 2018.

[32] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," *CVPR*, pp. 4353–4361, 2015.

[33] F. Gney and A. Geiger, "Deep discrete flow," *ACCV*, 2016.

[34] C. Bailer, K. Varanasi, and D. Stricker, "CNN-based patch matching for optical flow with thresholded hinge embedding loss," *CVPR*, pp. 3250–3259, 2017.

[35] C. Bailer, B. Taetz, and D. Stricker, "Flow Fields: Dense correspondence fields for highly accurate large displacement optical flow estimation," *ICCV*, pp. 4015–4023, 2015.

[36] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *NIPS*, pp. 2017–2025, 2015.

[37] D. Tschumperlé and R. Deriche, "Vector-valued image regularization with PDEs: A common framework for different applications," *PAMI*, vol. 27, no. 4, pp. 506–517, 2005.

[38] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi, "Bilateral filtering-based optical flow estimation with occlusion detection," *ECCV*, pp. 211–224, 2006.

[39] N. Mayer, E. Ilg, P. Husser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," *CVPR*, pp. 4040–4048, 2016.

[40] D. Sun, S. Roth, and M. J. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *IJCV*, vol. 106, no. 2, pp. 115–137, 2014.

[41] J. Xu, R. Ranftl, and V. Koltun, "Accurate optical flow via direct cost volume processings," *CVPR*, pp. 1289–1297, 2017.

[42] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume," *CVPR*, pp. 8934–8943, 2018.

[43] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," *ECCV*, pp. 611–625, 2012.

[44] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving?" *CVPR*, pp. 3354–3361, 2012.

[45] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," *CVPR*, pp. 3061–3070, 2015.

TABLE 8: The network details of NetC. "# Ch. In / Out" means the number of input or output channels of the feature maps. "conv" denotes convolution.

Layer name	Kernel	Stride	# Ch. In / Out	Input
conv1	7 × 7	1	3 / 32	I_1 or I_2
conv2_1	3 × 3	2	32 / 32	conv1
conv2_2	3 × 3	1	32 / 32	conv2_1
conv2_3	3 × 3	1	32 / 32	conv2_2
conv3_1	3 × 3	2	32 / 64	conv2_3
conv3_2	3 × 3	1	64 / 64	conv3_1
conv4_1	3 × 3	2	64 / 96	conv3_2
conv4_2	3 × 3	1	96 / 96	conv4_1
conv5	3 × 3	2	96 / 128	conv4_2
conv6	3 × 3	2	128 / 192	conv5

TABLE 9: The network details of the descriptor matching unit (M) in NetE at pyramid level 5. "upconv", "f-warp", "corr", and "loss" denote the fractionally strided convolution (so-called deconvolution), feature warping, correlation, and the layer where training loss is applied, respectively. Furthermore, "conv5a' and "conv5b" denote the high-dimensional features of images I_1 and I_2 generated from NetC at pyramid level 5.

Layer name	Kernel	Stride	# Ch. In / Out	Input
upconv5_M	4 × 4	$\frac{1}{2}$	2 / 2	flow6_R
f-warp5_M	-	-	128, 2 / 128	conv5b, upconv5_M
corr5_M	1 × 1	1	128, 128 / 49	conv5a, f-warp5_M
conv5_1_M	3 × 3	1	49 / 128	corr5_M
conv5_2_M	3 × 3	1	128 / 128	conv5_1_M
conv5_3_M	3 × 3	1	128 / 96	conv5_2_M
conv5_4_M	3 × 3	1	96 / 64	conv5_3_M
conv5_5_M	3 × 3	1	64 / 32	conv5_4_M
conv5_6_M	3 × 3	1	32 / 2	conv5_5_M
flow5_M, loss5_M	element-wise sum		2, 2 / 2	upconv5_M, conv5_4_M

TABLE 10: Network details of sub-pixel refinement module (S) in NetE at pyramid level 5.

Layer name	Kernel	Stride	# Ch. In / Out	Input
f-warp5_S	-	-	128, 2 / 128	conv5b, flow5_C
conv5_1_S	3×3	1	258 / 128	concat(conv5a, f-warp5_S, flow5_C)
conv5_2_S	3 × 3	1	128 / 128	conv5_1_S
conv5_3_S	3 × 3	1	128 / 96	conv5_2_S
conv5_4_S	3 × 3	1	96 / 64	conv5_3_S
conv5_5_S	3 × 3	1	64 / 32	conv5_4_S
conv5_6_S	3 × 3	1	32 / 2	conv5_5_S
flow5_S, loss5_S	element-wise sum		2, 2 / 2	flow5_C, conv5_4_S

TABLE 11: Network details of flow regularization module (R) in NetE at pyramid level 5. "warp", "norm", "softmax", and "f-lcon" denote the image warping, L2 norm of the RGB brightness difference between the two input images, normalized exponential operation over each 1 × 1 × (# Ch. In) column in the 3-D tensor, and feature-driven local convolution, respectively. Furthermore, "conv_dist" highlight the output of the convolution layer is used as the feature-driven distance metric $\mathcal{D}$. "im5a" and "im5b" denote the down-sized images of I_1 and I_2 at pyramidal level 5.

Layer name	Kernel	Stride	# Ch. In / Out	Input
m-flow5_R	remove mean		2 / 2	flow5_S
warp5_R	-	-	3, 2 / 3	im5b, flow5_S
norm5_R	L2 norm		3, 3 / 1	im5a, warp5_R
conv5_1_R	3 × 3	1	131 / 128	concat(conv5a, m-flow5_R, norm5_R)
conv5_2_R	3 × 3	1	128 / 128	conv5_1_R
conv5_3_R	3 × 3	1	128 / 64	conv5_2_R
conv5_4_R	3 × 3	1	64 / 64	conv5_3_R
conv5_5_R	3 × 3	1	64 / 32	conv5_4_R
conv5_6_R	3 × 3	1	32 / 32	conv5_5_R
conv5_dist_R	3 × 3	1	32 / 9	conv5_6_R
softmax5_R	-	1	9 / 9	conv5_dist_R
f-lcon5_R, loss5_R	3 × 3	1	9, 2 / 2	softmax5_R, flow5_R