# LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation

Tak-Wai Hui, Xiaoou Tang, Chen Change Loy
CUHK-SenseTime Joint Lab, The Chinese University of Hong Kong
{twhui,xtang,ccloy}@ie.cuhk.edu.hk

## Abstract

*FlowNet2 [14], the state-of-the-art convolutional neural network (CNN) for optical flow estimation, requires over 160M parameters to achieve accurate flow estimation. In this paper we present an alternative network that attains performance on par with FlowNet2 on the challenging Sintel final pass and KITTI benchmarks, while being 30 times smaller in the model size and 1.36 times faster in the running speed. This is made possible by drilling down to architectural details that might have been missed in the current frameworks: (1) We present a more effective flow inference approach at each pyramid level through a lightweight cascaded network. It not only improves flow estimation accuracy through early correction, but also permits seamless incorporation of descriptor matching in our network. (2) We present a novel flow regularization layer to ameliorate the issue of outliers and vague flow boundaries by using a feature-driven local convolution. (3) Our network owns an effective structure for pyramidal feature extraction and embraces feature warping rather than image warping as practiced in FlowNet2. Our code and trained models are available at github.com/twhui/LiteFlowNet.*

## 1. Introduction

Optical flow estimation is a long-standing problem in computer vision. Due to the well-known aperture problem, optical flow is not directly measurable [12, 13]. Hence, the estimation is typically solved by energy minimization in a coarse-to-fine framework [6, 20, 7, 36, 27, 22]. This class of techniques, however, involves complex energy optimization and thus it is not scalable for applications that demand real-time estimation.

FlowNet [9] and its successor FlowNet2 [14], have marked a milestone by using CNN for optical flow estimation. Their accuracies especially the successor are approaching that of 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 increment between the first image and the warped second image. The model, as a result, comprises over 160M parameters, which could be formidable in many applications. A recent network termed SPyNet [21] attempts a network with smaller size of 1.2M parameters by adopting image warping in each pyramid level. Nonetheless, the accuracy can only match that of FlowNet but not FlowNet2. The objective of this study is to explore alternative CNN architectures for accurate flow estimation yet with high efficiency. Our work is inspired by the successes of FlowNet2 and SPyNet, but we further drill down the key elements to fully unleash the potential of deep convolutional network combined with classical principles.

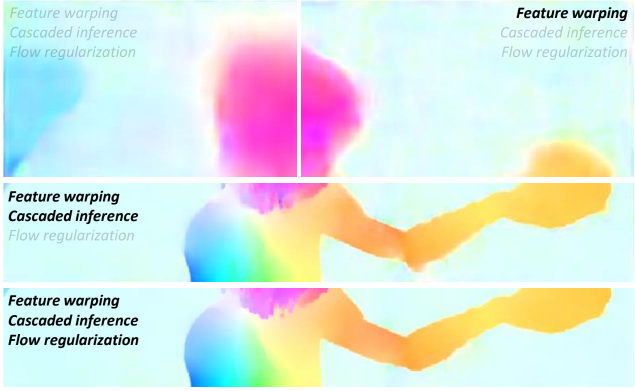

Figure 1: Examples demonstrate the effectiveness of the proposed components in LiteFlowNet for i) feature warping, ii) cascaded flow inference, and iii) flow regularization. Enabled components are indicated with bold black fonts.

There are two general principles to improve the design of FlowNet2 and SPyNet. The first principle is *pyramidal feature extraction*. 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 flow field 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 FlowNet2 that adopts U-Net architecture [23] for flow inference. In comparison to SPyNet, our network separates the process of feature extraction and flow estimation. This helps us to better pinpoint the bottleneck of accuracy and model size.

The second general principle is *feature warping*. FlowNet2 and SPyNet warp the second image towards the first image in the pair using the previous flow estimate, and then refine the estimate using the feature maps generated by the warped and the first images. Warping an image and then generating the feature maps of the warped image 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 computed by the encoder. This one-step feature warping 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 flow problem.

We now highlight the more specific differences between our network and existing CNN-based optical flow estimation frameworks:

1) *Cascaded flow inference* – At each pyramid level, we introduce a novel cascade of two lightweight networks. Each of them has a feature warping (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 feature warping, we can use a rather short displacement than [9, 14] to establish the cost volume. Besides, matching is performed only at sampled positions and thus a sparse cost-volume is aggregated. This effectively reduces the computational burden raised by the explicit matching.

2) *Flow regularization* – The cascaded flow inference resembles the role of data fidelity in energy minimization methods. Using data term alone, vague flow boundaries and undesired artifacts exist in flow fields. To tackle this problem, local flow consistency and co-occurrence between flow boundaries and intensity edges are commonly used as the cues to regularize flow field. Some of the representative methods include anisotropic image-driven [32], image- and flow-driven [28], and complementary [36] regularizations. After cascaded flow inference, we allow the flow field to be further regularized by our novel feature-driven local convolution (f-lconv) layer[1] 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. To our best knowledge, state-of-the-art CNNs do not explore such a flow regularization.

The effectiveness of the aforementioned contributions are depicted in Figure 1. In summary, we propose a compact LiteFlowNet to estimate optical flow. Our network innovates the useful elements from conventional methods. *e.g*., brightness constraint in data fidelity to pyramidal CNN features and image warping to CNN feature warping. More specifically, we present a cascaded flow inference with feature warping and flow regularization in each pyramid level, which are new in the literature. Overall, our network outperforms FlowNet [9] and SPyNet [21] and is on par with the recent FlowNet2 [14], while having 30 times fewer parameters and being 1.36 times faster than FlowNet2.

## 2. Related Work

Here, we briefly review some of the major approaches for optical flow estimation.

**Variational methods.** Since the pioneering work by Horn and Schunck [12], variational methods have dominated optical flow estimation. Brox *et al*. address illumination changes by combining the brightness and gradient constancy assumptions [6]. Brox *et al*. integrate rich descriptors into variational formulation [7]. In DeepFlow [31], Weinzaepfel *et al*. propose to correlate multi-scale patches and incorporate this as the matching term in functional. In PatchMatch Filter [16], Lu *et al*. establish dense correspondence using the superpixel-based PatchMatch [4]. Revaud *et al*. propose a method EpicFlow that uses externally matched flows as initialization and then performs interpolation [22]. Zimmer *et al*. design the complementary regularization that exploits directional information from the constraints imposed in data term [36]. 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 [5]. Roth *et al*. formulates the prior probability of flow field as Field-of-Experts model [26] that captures higher order spatial statistics [25]. Sun *et al*. study

[1] We name it as *feature-driven local convolution* (f-lconv) layer in order to distinguish it from local convolution (lconv) layer of which filter weights are locally fixed in conventional CNNs [29].

the probabilistic model of brightness inconstancy in a high-order random field framework [28]. Nir *et al*. represent image motion using the over-parameterization model [19]. Rosenbaum *et al*. model the local statistics of optical flow using Gaussian mixtures [24]. 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) [33]. It can be shown that the parameterized model [5, 19, 33] can be efficiently implemented using CNN.

**CNN-based methods.** In the work of Fischer *et al*. termed FlowNet [9], 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. In our work, we present an end-to-end approach that performs in-network flow regularization using the proposed f-lconv layer, which plays similar role as the regularization term in variational methods. In FlowNet2 [14], 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. Our model uses a more efficient architecture containing 30 times fewer parameters than FlowNet2 while the performance is on par with it. A compact network termed SPyNet [21] from Ranjan *et al*. is inspired from spatial pyramid. Nevertheless, the accuracy is far below FlowNet2. 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 [37]. DeepFlow [31] 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.

An alternative approach for establishing point correspondence is to match image patches. Zagoruyko *et al*. first introduce to CNN-feature matching [35]. Güney *et al*. find feature representation and formulate optical flow estimation in MRF [11]. Bailer *et al*. [2] use multi-scale features and then perform feature matching as Flow Fields [1]. 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, Fischer *et al*. [9] and Ilg *et al*. [14] 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 at sampled positions and a sub-pixel refinement at every pyramid level.

We are inspired by the feature transformation used in Spatial Transformer [15]. Our network uses the proposed f-warp layer to displace each channel[2] 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, our decider network is a more generic warping network that warps high-level CNN features.

## 3. LiteFlowNet

LiteFlowNet is composed of two compact sub-networks that are specialized in pyramidal feature extraction and optical flow estimation 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 call the two sub-networks as NetC and NetE respectively. NetC transforms any given image pair into two pyramids of multi-scale high-dimensional features. NetE consists of cascaded flow inference and regularization modules that estimate coarse-to-fine flow fields.

**Pyramidal Feature Extraction.** As shown in Figure 2, NetC is a two-stream network in which the filter weights are shared across the two streams. Each of them functions as a 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-$s$ convolutions with the reduction of spatial resolution by a factor $s$ up the 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.

**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. With the motivation of image warping used in conventional methods [6, 20] and recent CNNs [14, 21] 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 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 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:

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

[2] 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.

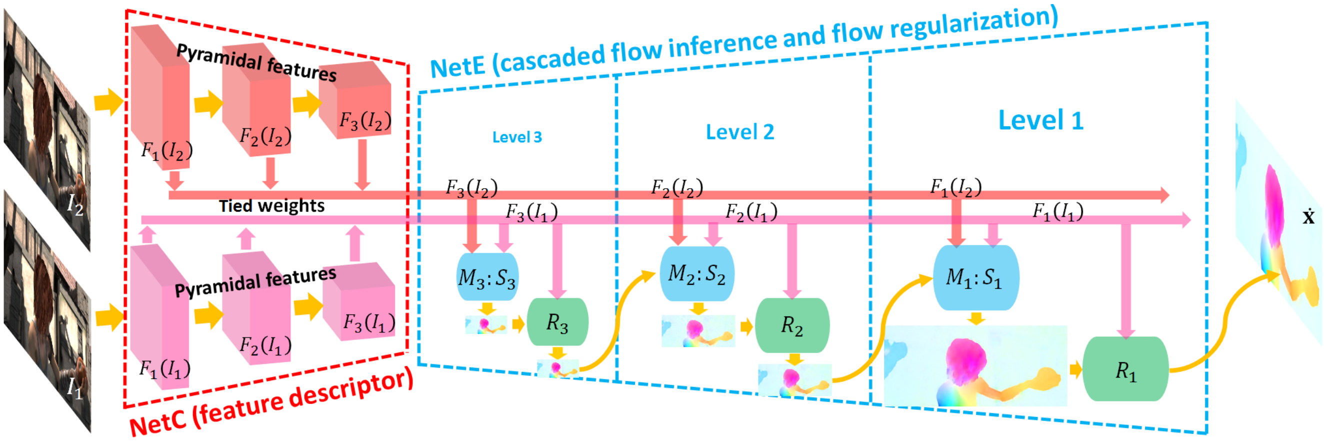

Figure 2: The network structure of LiteFlowNet. For the ease of representation, only a 3-level design is shown. Given an image pair ($I_1$ and $I_2$), NetC generates two pyramids of high-level features ($\mathcal{F}_k(I_1)$ in pink and $\mathcal{F}_k(I_2)$ in red, $k \in [1, 3]$). NetE yields multi-scale flow fields 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.

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 [15].

### 3.1. Cascaded Flow Inference

At each pyramid level of NetE, pixel-by-pixel matching of high-level features yields coarse flow estimate. A subsequent refinement on the coarse flow further improves it to sub-pixel accuracy.

**First Flow Inference (descriptor matching).** Point correspondence between $I_1$ and $I_2$ is established through computing correlation 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 $\mathcal{F}_2$, $\mathbf{d} \in \mathbb{Z}$ is the displacement vector from $\mathbf{x}$, and $N$ is the length of the feature vector. A cost volume $C$ is built by aggregating all the matching costs into a 3D grid.

We reduce the computational burden raised by cost-volume processing [9, 14] in three ways: 1) We perform short-range matching at every pyramid level instead of long-range matching at a single level. 2) We reduce feature-space distance between $\mathcal{F}_1$ and $\mathcal{F}_2$ by warping $\mathcal{F}_2$ towards $\mathcal{F}_1$ using our proposed f-warp through flow estimate[3] $\dot{\mathbf{x}}$ from previous level. 3) 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 last technique reduces the frequency of matching per pyramid level.

In the descriptor matching unit $M$, residual flow $\Delta\dot{\mathbf{x}}_m$ is inferred by filtering the cost volume $C$ as illustrated in Figure 3. A complete flow field $\dot{\mathbf{x}}_m$ is computed as follows:

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

**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 $\dot{\mathbf{x}}_m$ 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 computing residual flow $\Delta\dot{\mathbf{x}}_s$ as the following:

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

### 3.2. Flow Regularization

Cascaded flow inference resembles the role of data fidelity in conventional minimization methods. Using data

[3] $\dot{\mathbf{x}}$ 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.

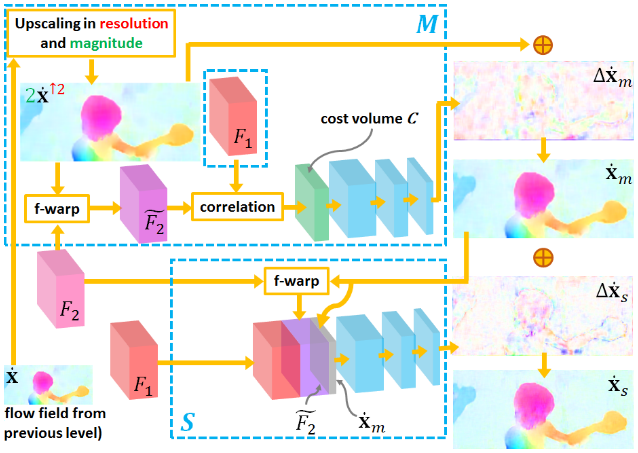

Figure 3: A cascaded flow inference module $M$:$S$ in NetE. It consists of a descriptor matching unit $M$ and a sub-pixel refinement unit $S$. In $M$, f-warp transforms high-level feature $\mathcal{F}_2$ to $\widetilde{\mathcal{F}}_2$ via upscaled flow field $2\dot{\mathbf{x}}^{\uparrow 2}$ estimated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

term alone, vague flow boundaries and undesired artifacts commonly exist in flow field [32, 36]. To tackle this problem, we propose to use a feature-driven local convolution (f-lcon) to regularize flow field from the cascaded flow inference. The operation of f-lcon is well-governed by the Laplacian formulation of diffusion of pixel values [30]. In contrast to local convolution (lcon) used in conventional CNNs [29], f-lcon is more generalized. Not only is a distinct filter used for each position of feature map, but the filter is adaptively constructed for individual flow patches.

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. 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[4] $O$. In summary, $\mathcal{D}$ is adaptively constructed by a CNN unit $R_D$ as follows:

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

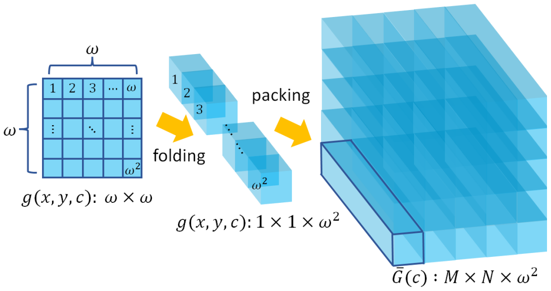
$\omega$
1
2
3
…
$\omega$
$\omega$
⋮
⋱
⋮
$\omega^2$
$g(x, y, c)$: $\omega \times \omega$
1
2
3
$\omega^2$
folding
packing
$g(x, y, c)$: $1 \times 1 \times \omega^2$
$\bar{G}(c)$ : $M \times N \times \omega^2$

Figure 4: Folding and packing of f-lcon filters $\{g\}$. The $(x, y)$-entry of 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 $F$.

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

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

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. Experiments

**Network Details.** In LiteFlowNet, NetC generates 6-level pyramidal features and NetE predicts flow fields for levels 6 to 2. Flow field in level 2 is upsampled to yield flow field in level 1. We set the maximum searching radius in cost-volume to 3 pixels (levels 6 to 4) or 6 pixels (levels 3 to 2).

[4] We use the 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.

Table 1: 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** |

Matching is performed at each position in pyramidal features, except for levels 3 to 2 that it is performed at a regularly sampled grid (a stride of 2). All convolution layers use $3 \times 3$ filters, except each last layer in descriptor matching $M$, sub-pixel refinement $S$, and flow regularization $R$ units uses $5 \times 5$ (levels 4 to 3) or $7 \times 7$ (level 2) filters. Each convolution layer is followed by a leaky rectified linear unit layer, except f-lcon and the last layer in $M$, $S$ and $R$ CNN units. More details can be found in the supplementary material.

**Training Details.** We train our network stage-wise by the following steps: 1) NetC and $M_6$:$S_6$ of NetE is trained for 300k iterations. 2) $R_6$ together with the trained network in step 1 is 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 (level 2: 300k) iterations. Filter weights are initialized from previous level. 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 L2 training loss, Adam optimization, data augmentation, and training schedule (Chairs [9] → Things3D [17]) as FlowNet2 [14]. We denote **LiteFlowNet-pre** and **LiteFlowNet** as the networks trained on Chairs and Chairs → Things3D, respectively.

## 4.1. Results

We compare several variants of LiteFlowNet to state-of-the-art methods on public benchmarks including FlyingChairs (Chairs) [9], Sintel clean and final [8], KITTI12 [10], KITTI15 [18], and Middlebury [3].

**FlyingChairs.** We first compare the intermediate results of different well-performing networks trained on Chairs alone in Table 1. Average end-point error (AEE) is reported. LiteFlowNet-pre outperforms the compared networks. No intermediate result is available for FlowNet2 [14] as each cascade is trained on the Chairs → Things3D schedule individually. Since FlowNetC, FlowNetS (variants of FlowNet [9]), and SPyNet [21] have fewer parameters than FlowNet2 and the later two models do not perform feature matching, we also construct a small-size counterpart **LiteFlowNetX-pre** 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 matching.

Table 3: 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. Abbreviation LFlowNet refers to LiteFlowNet.

| | Shallow | | Deep | Very Deep | |
|---|---|---|---|---|---|
| Model | FlowNetC | SPyNet | LFlowNetX | LFlowNet | FlowNet2 |
| # layers | 26 | 35 | 74 | 99 | 115 |
| # param. (M) | 39.16 | **1.20** | **0.90** | **5.37** | 162.49 |
| Runtime (ms) | **32.28** | (129.83) | **35.83** | **90.25** | 122.39 |

**MPI Sintel.** In Table 2, LiteFlowNetX-pre outperforms FlowNetS (and C) [9] and SPyNet [21] that are trained on Chairs on all cases except the Middlebury benchmark. LiteFlowNet, trained on the Chairs → Things3D schedule, performs better than LiteFlowNet-pre as expected. LiteFlowNet also outperforms SPyNet, FlowNet2-S (and -C) [14]. We also fine-tuned LiteFlowNet on a mixture of Sintel clean and final training data (**LiteFlowNet-ft**) using the generalized Charbonnier loss [27]. LiteFlowNet-ft performs better than EpicFlow [22] for Sintel final testing set. Despite hybrid methods (CNN + post-processing) [11, 2, 34] perform better than FlowNet2, its GPU runtime requires several seconds that makes it formidable in many applications. Figure 5 shows some examples of flow fields on Sintel dataset. LiteFlowNet-ft and FlowNet2-ft-sintel perform the best among the compared methods. As LiteFlowNet has flow regularization module, sharper flow boundaries and lesser artifacts can be observed in the generated flow fields.

**KITTI.** LiteFlowNet consistently performs better than LiteFlowNet-pre especially on KITTI15 as shown in Table 2. It also outperforms SPyNet [21] and FlowNet2-S (and C) [14]. By fine-tuning on a mixture of KITTI12 and KITTI15 training data (**LiteFlowNet-ft**), LiteFlowNet generalizes well to real-world data. LiteFlowNet-ft is on par with FlowNet2-ft-kitti. Figure 6 shows some examples of flow fields on KITTI. As in the case for Sintel, LiteFlowNet-ft and FlowNet2-ft-kitti performs the best among the compared methods. Even though LiteFlowNet 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 4.3.

**Middlebury.** LiteFlowNet has comparable performance with conventional methods. It outperforms FlowNetS (and C) [9] and FlowNet2-S (and C) [14] and is on par with SPyNet [21] and FlowNet2. On the benchmark, LiteFlowNet-ft refers to the one fine-tuned on Sintel.

## 4.2. Runtime and Parameters

We measure runtime of a CNN using a machine equipped with an Intel Xeon E5 2.2GHz and an NVIDIA GTX 1080. Timings are averaged over 100 runs for Sintel image pairs of size $1024 \times 436$. As summarized in Table 3, LiteFlowNet has about **30 times** fewer parameters than FlowNet2 [14] and is **1.36 times** faster in runtime. LiteFlowNetX, a variant of LiteFlowNet having a smaller

Table 2: 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. 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 [14]. $^2$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 [17])

| | Method | Sintel clean | | Sintel final | | KITTI12 | | KITTI15 | | | Middlebury | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | train | test | train | test | train | test | train | train (Fl-all) | test (Fl-all) | train | test |
| Conventional | LDOF$^1$ [7] | 4.64 | 7.56 | 5.96 | 9.12 | 10.94 | 12.4 | 18.19 | 38.11% | - | 0.44 | 0.56 |
| | DeepFlow$^1$ [31] | 2.66 | 5.38 | 3.57 | 7.21 | 4.48 | 5.8 | 10.63 | 26.52% | 29.18% | 0.25 | 0.42 |
| | Classic+NLP [27] | 4.49 | 6.73 | 7.46 | 8.29 | - | 7.2 | - | - | - | **0.22** | **0.32** |
| | PCA-Layers$^1$ [33] | 3.22 | 5.73 | 4.52 | 7.89 | 5.99 | 5.2 | 12.74 | 27.26% | - | 0.66 | - |
| | EpicFlow$^1$ [22] | 2.27 | 4.12 | 3.56 | 6.29 | **3.09** | 3.8 | 9.27 | 27.18% | **27.10%** | 0.31 | 0.39 |
| | FlowFields$^1$ [1] | **1.86** | **3.75** | **3.06** | **5.81** | 3.33 | **3.5** | **8.33** | **24.43%** | - | 0.27 | 0.33 |
| Hybrid | Deep DiscreteFlow [11] | - | 3.86 | - | 5.73 | - | 3.4 | - | - | 21.17% | - | - |
| | Bailer *et al.* [2] | - | **3.78** | - | 5.36 | - | **3.0** | - | - | 19.44% | - | - |
| | DC Flow [34] | - | - | - | **5.12** | - | - | - | - | **14.86%** | - | - |
| Heavyweight CNN | FlowNetS [9] | 4.50 | 7.42 | 5.45 | 8.43 | 8.26 | - | - | - | - | 1.09 | - |
| | FlowNetS-ft [9] | (3.66) | 6.96 | (4.44) | 7.76 | 7.52 | 9.1 | - | - | - | 0.98 | - |
| | FlowNetC [9] | 4.31 | 7.28 | 5.87 | 8.81 | 9.35 | - | - | - | - | 1.15 | - |
| | FlowNetC-ft [9] | (3.78) | 6.85 | (5.28) | 8.51 | 8.79 | - | - | - | - | 0.93 | - |
| | FlowNet2-S$^3$ [14] | 3.79 | - | 4.99 | - | 7.26 | - | 14.28 | 51.06% | - | 1.04 | - |
| | *FlowNet2-S re-trained*$^2$ | *3.96* | - | *5.37* | - | *7.31* | - | *14.51* | *51.38%* | - | *1.13* | - |
| | FlowNet2-C$^3$ [14] | 3.04 | - | 4.60 | - | 5.79 | - | 11.49 | 44.09% | - | 0.98 | - |
| | FlowNet2 [14] | **2.02** | **3.96** | **3.54**$^4$ | 6.02 | 4.01$^5$ | - | 10.08$^5$ | 29.99%$^5$ | - | **0.35** | **0.52** |
| | FlowNet2-ft-sintel [14] | (1.45) | 4.16 | (2.19$^4$) | **5.74** | **3.54**$^5$ | - | **9.94**$^5$ | **28.02%**$^5$ | - | **0.35** | - |
| | FlowNet2-ft-kitti [14] | 3.43 | - | 4.83$^4$ | - | (1.43$^5$) | **1.8** | (2.36$^5$) | (8.88%$^5$) | **11.48%** | 0.56 | - |
| Lightweight CNN | SPyNet [21] | 4.12 | 6.69 | 5.57 | 8.43 | 9.12 | - | - | - | - | **0.33** | 0.58 |
| | SPyNet-ft [21] | (3.17) | 6.64 | (4.32) | 8.36 | *3.36*$^6$ | 4.1 | - | - | 35.07% | **0.33** | 0.58 |
| | LiteFlowNetX-pre | 3.70 | - | 4.82 | - | 6.81 | - | 16.64 | 36.64% | - | 0.45 | - |
| | LiteFlowNetX | 3.58 | - | 4.79 | - | 6.38 | - | 15.81 | 34.90% | - | 0.46 | - |
| | LiteFlowNet-pre | 2.78 | - | 4.17 | - | 4.56 | - | 11.58 | 32.59% | - | 0.45 | - |
| | LiteFlowNet | **2.52** | - | **4.05** | - | **4.25** | - | **10.46** | **29.30%** | - | 0.39 | - |
| | LiteFlowNet-ft | (1.64) | **4.86** | (2.23) | **6.09** | (1.26) | **1.7** | (2.16) | (8.16%) | **10.24%** | **0.33** | **0.50** |

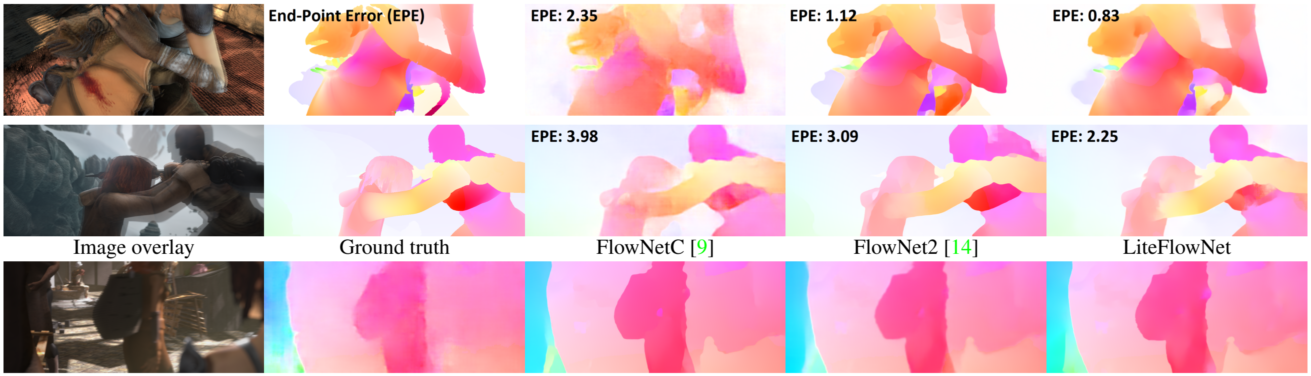

Image overlay Ground truth FlowNetC [9] FlowNet2 [14] LiteFlowNet

First image FlowNetC [9] FlowNet2 [14] FlowNet2-ft-sintel [14] LiteFlowNet-ft

Figure 5: Examples of flow fields from different methods on Sintel training sets for clean (top row), final (middle row) passes, and the testing set for final pass (last row). Fine details are well preserved and less artifacts can be observed in the flow fields of LiteFlowNet.

model size and without descriptor matching, has about **43 times** fewer parameters than FlowNetC [9] and a comparable runtime. LiteFlowNetX also has **1.33 times** fewer parameters than SPyNet [21]. LiteFlowNet and its variants are currently the most compact CNNs for flow estimation.

### 4.3. 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. The AEE results are summarized in Table 4 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 warping disabled (M and MS). Flow fields from M and MS are more vague. Large degradation in AEE is noticed especially for KITTI12 (33%) and KITTI15 (25%). With feature warping, pyramidal features that input to flow inference are closer to each other. This facilitates flow estimation in subsequent pyramid level by computing residual flow.

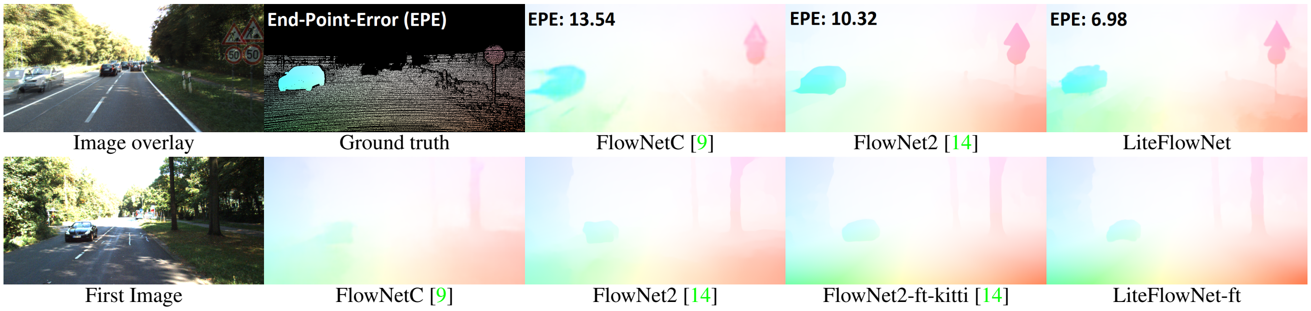

Image overlay · Ground truth · FlowNetC [9] · FlowNet2 [14] · LiteFlowNet

First Image · FlowNetC [9] · FlowNet2 [14] · FlowNet2-ft-kitti [14] · LiteFlowNet-ft

Figure 6: Examples of flow fields from different methods on the training set (top) and the testing set (bottom) of KITTI15.

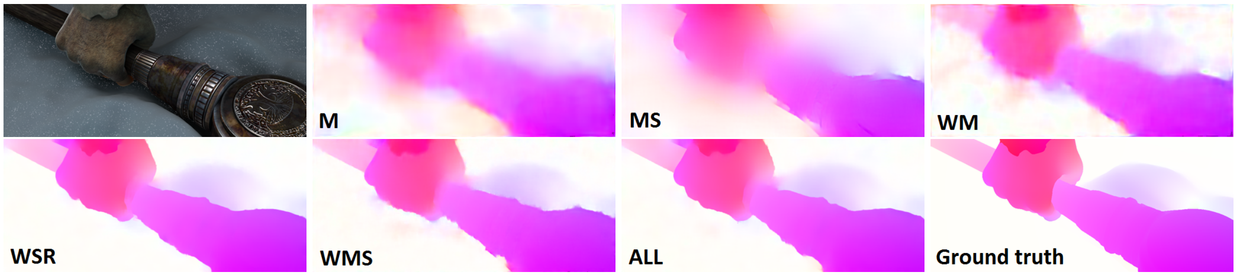

Figure 7: Examples of flow fields from different variants of LiteFlowNet-pre trained on Chairs with some of the components disabled. LiteFlowNet-pre is denoted as "All". W = Feature **W**arping, M = Descriptor **M**atching, S = **S**ub-Pixel Refinement, R = **R**egularization.

Table 4: 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 | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| **S**ub-pix. Refinement | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| **R**egularization | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| 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** |
| KITTI12 (train) | 9.22 | 8.01 | 6.17 | 5.03 | 4.83 | **4.56** |
| KITTI15 (train) | 18.24 | 16.19 | 14.52 | 13.20 | 12.32 | **11.58** |

**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.

**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 field.

**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.

## 5. Conclusion

We have presented a compact network for accurate flow estimation. LiteFlowNet outperforms FlowNet [9] and is on par with the state-of-the-art FlowNet2 [14] on public benchmarks while being faster in runtime and 30 times smaller in model size. Pyramidal feature extraction and feature warping (f-warp) help us to break the de facto rule of accurate flow network requiring large model size. To address large-displacement and detail-preserving flows, LiteFlowNet exploits short-range matching to generate pixel-level flow field and further improves the estimate to sub-pixel accuracy in the cascaded flow inference. To result crisp flow boundaries, LiteFlowNet regularizes flow field through feature-driven local convolution (f-lcon). With its lightweight, accurate, and fast flow computation, we expect that LiteFlowNet can be deployed to many applications such as motion segmentation, action recognition, and more.

**Acknowledgement.** This work is supported by SenseTime Group Limited and the General Research Fund sponsored by the Research Grants Council of the Hong Kong SAR (CUHK 14241716, 14224316, 14209217).

## References

[1] C. Bailer, B. Taetz, and D. Stricker. Flow Fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. *ICCV*, pages 4015–4023, 2015. 3, 7

[2] C. Bailer, K. Varanasi, and D. Stricker. CNN-based patch matching for optical flow with thresholded hinge embedding loss. *CVPR*, pages 3250–3259, 2017. 3, 6, 7

[3] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 92(1):1–31, 2011. 6

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *SIGGRAGH*, pages 83–97, 2009. 2

[5] M. J. Black, Y. Yacoobt, A. D. Jepsont, and D. J. Fleets. Learning parameterized models of image motion. *CVPR*, pages 674–679, 1997. 2, 3

[6] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. *ECCV*, pages 25–36, 2004. 1, 2, 3

[7] T. Brox and J. Mailk. Large displacement optical flow: Descriptor matching in variational motion estimation. *PAMI*, 33(3):500–513, 2011. 1, 2, 7

[8] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. *ECCV*, pages 611–625, 2012. 6

[9] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. *ICCV*, pages 2758–2766, 2015. 1, 2, 3, 4, 6, 7, 8

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? *CVPR*, pages 3354–3361, 2012. 6

[11] F. Gney and A. Geiger. Deep discrete flow. *ACCV*, pages 207–224, 2016. 3, 6, 7

[12] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Arifical Intelligence*, 17:185–203, 1981. 1, 2

[13] T.-W. Hui and R. Chung. Determining motion directly from normal flows upon the use of a spherical eye platform. *CVPR*, pages 2267–2274, 2013. 1

[14] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. FlowNet2.0: Evolution of optical flow estimation with deep networks. *CVPR*, pages 2462–2470, 2017. 1, 2, 3, 4, 6, 7, 8

[15] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *NIPS*, pages 2017–2025, 2015. 3, 4

[16] J. Lu, H. Yang, D. Min, and M. N. Do. PatchMatch Filter: Efficient edge-aware filtering meets randomized search. *CVPR*, pages 1854–1861, 2013. 2

[17] 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*, pages 4040–4048, 2016. 6, 7

[18] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. *CVPR*, pages 3061–3070, 2015. 6

[19] T. Nir, A. M. Bruckstein, and R. Kimmel. Overparameterized variational optical flow. *IJCV*, 76(2):205–216, 2008. 3

[20] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert. Highly accurate optic flow computation with theoretically justified warping. *IJCV*, 67(2):141–158, 2006. 1, 3

[21] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. *CVPR*, pages 4161–4170, 2017. 1, 2, 3, 6, 7

[22] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. EpicFlow: Edge-preserving interpolation of correspondences for optical flow. *CVPR*, pages 1164–1172, 2015. 1, 2, 6, 7

[23] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. *MICCAI*, pages 234–241, 2015. 2

[24] D. Rosenbaum, D. Zoran, and Y. Weiss. Learning the local statistics of optical flow. *NIPS*, pages 2373–2381, 2013. 3

[25] S. Roth and M. Black. On the spatial statistics of optical flow. *ICCV*, pages 42–49, 2005. 2

[26] S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. *CVPR*, pages 860–867, 2005. 2

[27] D. Sun, S. Roth, and M. J. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *IJCV*, 106(2):115–137, 2014. 1, 6, 7

[28] D. Sun, S. Roth, J. Lewis, and M. J. Black. Learning optical flow. *ECCV*, pages 83–97, 2008. 2, 3

[29] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the gap to human-level performance in face verification. *CVPR*, pages 1701–1708, 2014. 2, 5

[30] D. Tschumperlé and R. Deriche. Vector-valued image regularization with PDEs: A common framework for different applications. *PAMI*, 27(4):506–517, 2005. 5

[31] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. DeepFlow: Large displacement optical flow with deep matching. *ICCV*, pages 500–513, 2013. 2, 3, 7

[32] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L$^1$ optical flow. *BMVC*, 2009. 2, 5

[33] J. Wulff and M. J. Black. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. *CVPR*, pages 120–130, 2015. 3, 7

[34] J. Xu, R. Ranftl, and V. Koltun. Accurate optical flow via direct cost volume processings. *CVPR*, pages 1289–1297, 2017. 6, 7

[35] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. *CVPR*, pages 4353–4361, 2015. 3

[36] H. Zimmer, A. Bruhn, and J. Weickert. Optic flow in harmony. *IJCV*, 93(3):368–388, 2011. 1, 2, 5

[37] S. Zweig and L. Wolf. InterpoNet, A brain inspired neural network for optical flow dense interpolation. *CVPR*, pages 4563–4572, 2017. 3