RAFT: Recurrent All-Pairs Field Transforms for Optical Flow

Zachary Teed and Jia Deng

Princeton University
{zteed,jiadeng}@cs.princeton.edu

Abstract. We introduce Recurrent All-Pairs Field Transforms (RAFT), a new deep network architecture for optical flow. RAFT extracts per-pixel features, builds multi-scale 4D correlation volumes for all pairs of pixels, and iteratively updates a flow field through a recurrent unit that performs lookups on the correlation volumes. RAFT achieves state-of-the-art performance. On KITTI, RAFT achieves an F1-all error of 5.10%, a 16% error reduction from the best published result (6.10%). On Sintel (final pass), RAFT obtains an end-point-error of 2.855 pixels, a 30% error reduction from the best published result (4.098 pixels). In addition, RAFT has strong cross-dataset generalization as well as high efficiency in inference time, training speed, and parameter count. Code is available at https://github.com/princeton-vl/RAFT.

1 Introduction

Optical flow is the task of estimating per-pixel motion between video frames. It is a long-standing vision problem that remains unsolved. The best systems are limited by difficulties including fast-moving objects, occlusions, motion blur, and textureless surfaces.

Optical flow has traditionally been approached as a hand-crafted optimization problem over the space of dense displacement fields between a pair of images [21, 50, 13]. Generally, the optimization objective defines a trade-off between a *data* term which encourages the alignment of visually similar image regions and a *regularization* term which imposes priors on the plausibility of motion. Such an approach has achieved considerable success, but further progress has appeared challenging, due to the difficulties in hand-designing an optimization objective that is robust to a variety of corner cases.

Recently, deep learning has been shown as a promising alternative to traditional methods. Deep learning can side-step formulating an optimization problem and train a network to directly predict flow. Current deep learning methods [25, 42, 22, 48, 20] have achieved performance comparable to the best traditional methods while being significantly faster at inference time. A key question for further research is designing effective architectures that perform better, train more easily and generalize well to novel scenes.

We introduce Recurrent All-Pairs Field Transforms (RAFT), a new deep network architecture for optical flow. RAFT enjoys the following strengths:

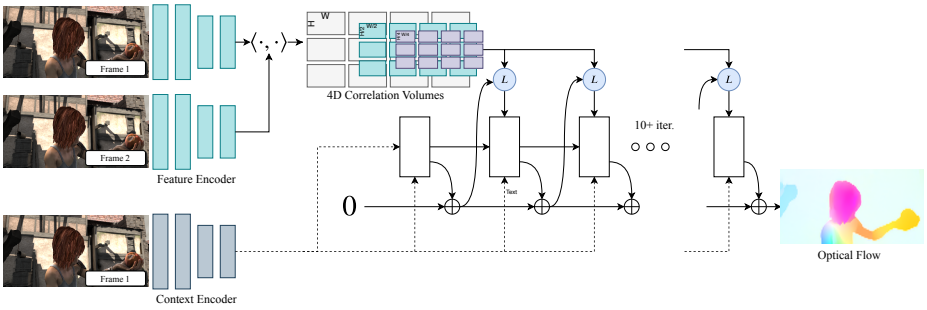

Fig. 1: RAFT consists of 3 main components: (1) A feature encoder that extracts per-pixel features from both input images, along with a context encoder that extracts features from only I_1. (2) A correlation layer which constructs a 4D $W \times H \times W \times H$ correlation volume by taking the inner product of all pairs of feature vectors. The last 2-dimensions of the 4D volume are pooled at multiple scales to construct a set of multi-scale volumes. (3) An *update operator* which recurrently updates optical flow by using the current estimate to look up values from the set of correlation volumes.

- *State-of-the-art accuracy*: On KITTI [18], RAFT achieves an F1-all error of 5.10%, a 16% error reduction from the best published result (6.10%). On Sintel [11] (final pass), RAFT obtains an end-point-error of 2.855 pixels, a 30% error reduction from the best published result (4.098 pixels).
- *Strong generalization*: When trained only on synthetic data, RAFT achieves an end-point-error of 5.04 pixels on KITTI [18], a 40% error reduction from the best prior deep network trained on the same data (8.36 pixels).
- *High efficiency*: RAFT processes 1088×436 videos at 10 frames per second on a 1080Ti GPU. It trains with 10X fewer iterations than other architectures. A smaller version of RAFT with 1/5 of the parameters runs at 20 frames per second while still outperforming all prior methods on Sintel.

RAFT consists of three main components: (1) a feature encoder that extracts a feature vector for each pixel; (2) a correlation layer that produces a 4D correlation volume for all pairs of pixels, with subsequent pooling to produce lower resolution volumes; (3) a recurrent GRU-based *update operator* that retrieves values from the correlation volumes and iteratively updates a flow field initialized at zero. Fig. 1 illustrates the design of RAFT.

The RAFT architecture is motivated by traditional optimization-based approaches. The feature encoder extracts per-pixel features. The correlation layer computes visual similarity between pixels. The update operator mimics the steps of an iterative optimization algorithm. But unlike traditional approaches, features and motion priors are not handcrafted but learned—learned by the feature encoder and the update operator respectively.

The design of RAFT draws inspiration from many existing works but is substantially novel. First, RAFT maintains and updates a single fixed flow field at high resolution. This is different from the prevailing coarse-to-fine design in prior work [42, 48, 22, 23, 49], where flow is first estimated at low resolution and upsampled and refined at high resolution. By operating on a single high-resolution flow field, RAFT overcomes several limitations of a coarse-to-fine cascade: the difficulty of recovering from errors at coarse resolutions, the tendency to miss small fast-moving objects, and the many training iterations (often over 1M) typically required for training a multi-stage cascade.

Second, the update operator of RAFT is recurrent and lightweight. Many recent works [24, 42, 48, 22, 25] have included some form of iterative refinement, but do not tie the weights across iterations [42, 48, 22] and are therefore limited to a fixed number of iterations. To our knowledge, IRR [24] is the only deep learning approach [24] that is recurrent. It uses FlowNetS [15] or PWC-Net [42] as its recurrent unit. When using FlowNetS, it is limited by the size of the network (38M parameters) and is only applied up to 5 iterations. When using PWC-Net, iterations are limited by the number of pyramid levels. In contrast, our update operator has only 2.7M parameters and can be applied 100+ times during inference without divergence.

Third, the update operator has a novel design, which consists of a convolutional GRU that performs lookups on 4D multi-scale correlation volumes; in contrast, refinement modules in prior work typically use only plain convolution or correlation layers.

We conduct experiments on Sintel[11] and KITTI[18]. Results show that RAFT achieves state-of-the-art performance on both datasets. In addition, we validate various design choices of RAFT through extensive ablation studies.

2 Related Work

Optical Flow as Energy Minimization Optical flow has traditionally been treated as an energy minimization problem which imposes a tradeoff between a *data* term and a *regularization* term. Horn and Schnuck [21] formulated optical flow as a continuous optimization problem using a variational framework, and were able to estimate a dense flow field by performing gradient steps. Black and Anandan[9] addressed problems with oversmoothing and noise sensitivity by introducing a robust estimation framework. TV-L1 [50] replaced the quadratic penalties with an L1 data term and total variation regularization, which allowed for motion discontinuities and was better equipped to handle outliers. Improvements have been made by defining better matching costs [45, 10] and regularization terms [38].

Such continuous formulations maintain a single estimate of optical flow which is refined at each iteration. To ensure a smooth objective function, a first order Taylor approximation is used to model the data term. As a result, they only work well for small displacements. To handle large displacements, the coarse-to-fine strategy is used, where an image pyramid is used to estimate large displacements

at low resolution, then small displacements refined at high resolution. But this coarse-to-fine strategy may miss small fast-moving objects and have difficulty recovering from early mistakes. Like continuous methods, we maintain a single estimate of optical flow which is refined with each iteration. However, since we build correlation volumes for all pairs at both high resolution and low resolution, each local update uses information about both small and large displacements. In addition, instead of using a subpixel Taylor approximation of the data term, our update operator learns to propose the descent direction.

More recently, optical flow has also been approached as a discrete optimization problem [35, 13, 47] using a global objective. One challenge of this approach is the massive size of the search space, as each pixel can be reasonably paired with thousands of points in the other frame. Menez et al[35] pruned the search space using feature descriptors and approximated the global MAP estimate using message passing. Chen et al. [13] showed that by using the distance transform, solving the global optimization problem over the full space of flow fields is tractable. DCFlow [47] showed further improvements by using a neural network as a feature descriptor, and constructed a 4D cost volume over all pairs of features. The 4D cost volume was then processed using the Semi-Global Matching (SGM) algorithm [19]. Like DCFlow, we also constructed 4D cost volumes over learned features. However, instead of processing the cost volumes using SGM, we use a neural network to estimate flow. Our approach is end-to-end differentiable, meaning the feature encoder can be trained with the rest of the network to directly minimize the error of the final flow estimate. In contrast, DCFlow requires their network to be trained using an embedding loss between pixels; it cannot be trained directly on optical flow because their cost volume processing is not differentiable.

Direct Flow Prediction Neural networks have been trained to directly predict optical flow between a pair of frames, side-stepping the optimization problem completely. Coarse-to-fine processing has emerged as a popular ingredient in many recent works [42, 49, 22–24, 48, 20, 8, 51]. In contrast, our method maintains and updates a single high-resolution flow field.

Iterative Refinement for Optical Flow Many recent works have used iterative refinement to improve results on optical flow [25, 39, 42, 22, 48] and related tasks [29, 52, 44, 28]. Ilg et al. [25] applied iterative refinement to optical flow by stacking multiple FlowNetS and FlowNetC modules in series. SpyNet[39], PWC-Net[42], LiteFlowNet[22], and VCN [48] apply iterative refinement using coarse-to-fine pyramids. The main difference of these approaches from ours is that they do not share weights between iterations.

More closely related to our approach is IRR[24], which builds off of the FlownetS and PWC-Net architecture but shares weights between refinement networks. When using FlowNetS, it is limited by the size of the network (38M parameters) and is only applied up to 5 iterations. When using PWC-Net, iterations are limited by the number of pyramid levels. In contrast, we use a much simpler refinement module (2.7M parameters) which can be applied for 100+ iterations during inference without divergence. Our method also shares similarites

with Devon [31], namely the construction of the cost volume without warping and fixed resolution updates. However, Devon does not have any recurrent unit. It also differs from ours regarding large displacements. Devon handles large displacements using a dilated cost volume while our approach pools the correlation volume at multiple resolutions.

Our method also has ties to TrellisNet [5] and Deep Equilibrium Models (DEQ) [6]. Trellis net uses depth tied weights over a large number of layers, DEQ simulates an infinite number of layers by solving for the fixed point directly. TrellisNet and DEQ were designed for sequence modeling tasks, but we adopt the core idea of using a large number of weight-tied units. Our update operator uses a modified GRU block[14], which is similar to the LSTM block used in TrellisNet. We found that this structure allows our update operator to more easily converge to a fixed flow field.

Learning to Optimize Many problems in vision can be formulated as an optimization problem. This has motivated several works to embed optimization problems into network architectures [4, 3, 43, 32, 44]. These works typically use a network to predict the inputs or parameters of the optimization problem, and then train the network weights by backpropogating the gradient through the solver, either implicitly[4, 3] or unrolling each step [32, 43]. However, this technique is limited to problems with an objective that can be easily defined.

Another approach is to learn iterative updates directly from data [1, 2]. These approaches are motivated by the fact that first order optimizers such as Primal Dual Hybrid Gradient (PDHG)[12] can be expressed as a sequence of iterative update steps. Instead of using an optimizer directly, Adler et al. [1] proposed building a network which mimics the updates of a first order algorithm. This approach has been applied to inverse problems such as image denoising [26], tomographic reconstruction [2], and novel view synthesis[17]. TVNet [16] implemented the TV-L1 algorithm as a computation graph, which enabled the training the TV-L1 parameters. However, TVNet operates directly based on intensity gradients instead of learned features, which limits the achievable accuracy on challenging datasets such as Sintel.

Our approach can be viewed as learning to optimize: our network uses a large number of update blocks to emulate the steps of a first-order optimization algorithm. However, unlike prior work, we never explicitly define a gradient with respect to some optimization objective. Instead, our network retrieves features from correlation volumes to propose the descent direction.

3 Approach

Given a pair of consecutive RGB images, I_1, I_2, we estimate a dense displacement field $(\mathbf{f}^1, \mathbf{f}^2)$ which maps each pixel (u, v) in I_2 to its corresponding coordinates $(u', v') = (u + f^1(u), v + f^2(v))$ in I_2. An overview of our approach is given in Figure 1. Our method can be distilled down to three stages: (1) feature extraction, (2) computing visual similarity, and (3) iterative updates, where all stages are differentiable and composed into an end-to-end trainable architecture.

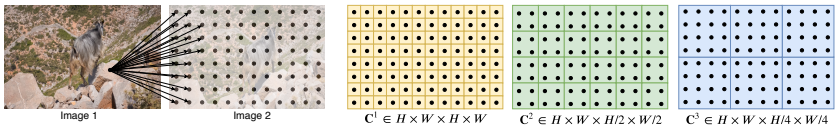

Fig. 2: Building correlation volumes. Here we depict 2D slices of a full 4D volume. For a feature vector in I_1, we take take the inner product with all pairs in I_2, generating a 4D $W \times H \times W \times H$ volume (each pixel in I_2 produces a 2D response map). The volume is pooled using average pooling with kernel sizes $\{1, 2, 4, 8\}$.

3.1 Feature Extraction

Features are extracted from the input images using a convolutional network. The feature encoder network is applied to both I_1 and I_2 and maps the input images to dense feature maps at a lower resolution. Our encoder, g_θ outputs features at $1/8$ resolution $g_\theta : \mathbb{R}^{H \times W \times 3} \mapsto \mathbb{R}^{H/8 \times W/8 \times D}$ where we set $D = 256$. The feature encoder consists of 6 residual blocks, 2 at $1/2$ resolution, 2 at $1/4$ resolution, and 2 at $1/8$ resolution (more details in the supplemental material).

We additionally use a context network. The context network extracts features only from the first input image I_1. The architecture of the context network, h_θ is identical to the feature extraction network. Together, the feature network g_θ and the context network h_θ form the first stage of our approach, which only need to be performed once.

3.2 Computing Visual Similarity

We compute visual similarity by constructing a full correlation volume between all pairs. Given image features $g_\theta(I_1) \in \mathbb{R}^{H \times W \times D}$ and $g_\theta(I_2) \in \mathbb{R}^{H \times W \times D}$, the correlation volume is formed by taking the dot product between all pairs of feature vectors. The correlation volume, $\mathbf{C}$, can be efficiently computed as a single matrix multiplication.

$$\mathbf{C}(g_\theta(I_1), g_\theta(I_2)) \in \mathbb{R}^{H \times W \times H \times W}, \qquad C_{ijkl} = \sum_h g_\theta(I_1)_{ijh} \cdot g_\theta(I_2)_{klh} \tag{1}$$

Correlation Pyramid: We construct a 4-layer pyramid $\{\mathbf{C}^1, \mathbf{C}^2, \mathbf{C}^3, \mathbf{C}^4\}$ by pooling the last two dimensions of the correlation volume with kernel sizes 1, 2, 4, and 8 and equivalent stride (Figure 2). Thus, volume $\mathbf{C}^k$ has dimensions $H \times W \times H/2^k \times W/2^k$. The set of volumes gives information about both large and small displacements; however, by maintaining the first 2 dimensions (the I_1 dimensions) we maintain high resolution information, allowing our method to recover the motions of small fast-moving objects.

Correlation Lookup: We define a lookup operator $L_\mathbf{C}$ which generates a feature map by indexing from the correlation pyramid. Given a current estimate of optical flow $(\mathbf{f}^1, \mathbf{f}^2)$, we map each pixel $\mathbf{x} = (u, v)$ in I_1 to its estimated

correspondence in I_2: $\mathbf{x}' = (u + f^1(u), v + f^2(v))$. We then define a local grid around $\mathbf{x}'$

$$\mathcal{N}(\mathbf{x}')_r = \{\mathbf{x}' + \mathbf{dx} \mid \mathbf{dx} \in \mathbb{Z}^2, ||\mathbf{dx}||_1 \leq r\} \tag{2}$$

as the set of integer offsets which are within a radius of r units of $\mathbf{x}'$ using the L1 distance. We use the local neighborhood $\mathcal{N}(\mathbf{x}')_r$ to index from the correlation volume. Since $\mathcal{N}(\mathbf{x}')_r$ is a grid of real numbers, we use bilinear sampling.

We perform lookups on all levels of the pyramid, such that the correlation volume at level k, $\mathbf{C}^k$, is indexed using the grid $\mathcal{N}(\mathbf{x}'/2^k)_r$. A constant radius across levels means larger context at lower levels: for the lowest level, $k = 4$ using a radius of 4 corresponds to a range of 256 pixels at the original resolution. The values from each level are then concatenated into a single feature map.

Efficient Computation for High Resolution Images: The all pairs correlation scales $O(N^2)$ where N is the number of pixels, but only needs to be computed once and is constant in the number of iterations M. However, there exists an equivalent implementation of our approach which scales $O(NM)$ exploiting the linearity of the inner product and average pooling. Consider the cost volume at level m, $\mathbf{C}^m_{ijkl}$, and feature maps $g^{(1)} = g_\theta(I_1)$, $g^{(2)} = g_\theta(I_2)$:

$$\mathbf{C}^m_{ijkl} = \frac{1}{2^{2m}} \sum_p^{2^m} \sum_q^{2^m} \langle g^{(1)}_{i,j}, g^{(2)}_{2^m k+p, 2^m l+q} \rangle = \langle g^{(1)}_{i,j}, \frac{1}{2^{2m}} (\sum_p^{2^m} \sum_q^{2^m} g^{(2)}_{2^m k+p, 2^m l+q}) \rangle$$

which is the average over the correlation response in the $2^m \times 2^m$ grid. This means that the value at $\mathbf{C}^m_{ijkl}$ can be computed as the inner product between the feature vector $g_\theta(I_1)_{ij}$ and $g_\theta(I_2)$ pooled with kernel size $2^m \times 2^m$.

In this alternative implementation, we do not precompute the correlations, but instead precompute the pooled image feature maps. In each iteration, we compute each correlation value on demand—only when it is looked up. This gives a complexity of $O(NM)$.

We found empirically that precomputing all pairs is easy to implement and not a bottleneck, due to highly optimized matrix routines on GPUs—even for 1088x1920 videos it takes only 17% of total inference time. Note that we can always switch to the alternative implementation should it become a bottleneck.

3.3 Iterative Updates

Our update operator estimates a sequence of flow estimates $\{\mathbf{f}_1, ..., \mathbf{f}_N\}$ from an initial starting point $\mathbf{f}_0 = \mathbf{0}$. With each iteration, it produces an update direction $\Delta\mathbf{f}$ which is applied to the current estimate: $\mathbf{f}_{k+1} = \Delta\mathbf{f} + \mathbf{f}_{k+1}$.

The update operator takes flow, correlation, and a latent hidden state as input, and outputs the update $\Delta\mathbf{f}$ and an updated hidden state. The architecture of our update operator is designed to mimic the steps of an optimization algorithm. As such, we used tied weights across depth and use bounded activations to encourage convergence to a fixed point. The update operator is trained to perform updates such that the sequence converges to a fixed point $\mathbf{f}_k \rightarrow \mathbf{f}^*$.

Initialization: By default, we initialize the flow field to 0 everywhere, but our iterative approach gives us the flexibility to experiment with alternatives. When applied to video, we test *warm-start* initialization, where optical flow from the previous pair of frames is forward projected to the next pair of frames with occlusion gaps filled in using nearest neighbor interpolation.

Inputs: Given the current flow estimate $\mathbf{f}^k$, we use it to retrieve correlation features from the correlation pyramid as described in Sec. 3.2. The correlation features are then processed by 2 convolutional layers. Additionally, we apply 2 convolutional layers to the flow estimate itself to generate flow features. Finally, we directly inject the input from the context network. The input feature map is then taken as the concatenation of the correlation, flow, and context features.

Update: A core component of the update operator is a gated activation unit based on the GRU cell, with fully connected layers replaced with convolutions:

$$z_t = \sigma(\text{Conv}_{3\text{x}3}([h_{t-1}, x_t], W_z)) \tag{3}$$

$$r_t = \sigma(\text{Conv}_{3\text{x}3}([h_{t-1}, x_t], W_r)) \tag{4}$$

$$\tilde{h_t} = \tanh(\text{Conv}_{3\text{x}3}([r_t \odot h_{t-1}, x_t], W_h)) \tag{5}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t} \tag{6}$$

where x_t is the concatenation of flow, correlation, and context features previously defined. We also experiment with a separable ConvGRU unit, where we replace the 3×3 convolution with two GRUs: one with a 1×5 convolution and one with a 5×1 convolution to increase the receptive field without significantly increasing the size of the model.

Flow Prediction: The hidden state outputted by the GRU is passed through two convolutional layers to predict the flow update $\Delta\mathbf{f}$. The output flow is at 1/8 resolution of the input image. During training and evaluation, we upsample the predicted flow fields to match the resolution of the ground truth.

Upsampling: The network outputs optical flow at 1/8 resolution. We upsample the optical flow to full resolution by taking the full resolution flow at each pixel to be the convex combination of a 3x3 grid of its coarse resolution neighbors. We use two convolutional layers to predict a $H/8 \times W/8 \times (8 \times 8 \times 9)$ mask and perform softmax over the weights of the 9 neighbors. The final high resolution flow field is found by using the mask to take a weighted combination over the neighborhood, then permuting and reshaping to a $H \times W \times 2$ dimensional flow field. This layer can be directly implemented in PyTorch using the `unfold` function.

3.4 Supervision

We supervised our network on the l_1 distance between the predicted and ground truth flow over the full sequence of predictions, $\{\mathbf{f}_1, ..., \mathbf{f}_N\}$, with exponentially increasing weights. Given ground truth flow $\mathbf{f}_{gt}$, the loss is defined as

$$\mathcal{L} = \sum_{i=1}^{N} \gamma^{i-N} ||\mathbf{f}_{gt} - \mathbf{f}_i||_1 \tag{7}$$

Fig. 3: Flow predictions on the Sintel test set.

where we set $\gamma = 0.8$ in our experiments.

4 Experiments

We evaluate RAFT on Sintel[11] and KITTI[18]. Following previous works, we pretrain our network on FlyingChairs[15] and FlyingThings[33], followed by dataset specific finetuning. Our method achieves state-of-the-art performance on both Sintel (both clean and final passes) and KITTI. Additionally, we test our method on 1080p video from the DAVIS dataset[37] to demonstrate that our method scales to videos of very high resolutions.

Implementation Details: RAFT is implemented in PyTorch[36]. All modules are initialized from scratch with random weights. During training, we use the AdamW[30] optimizer and clip gradients to the range $[-1, 1]$. Unless otherwise noted, we evaluate after 32 flow updates on Sintel and 24 on KITTI. For every update, $\Delta\mathbf{f} + \mathbf{f}_k$, we only backpropgate the gradient through the $\Delta\mathbf{f}$ branch, and zero the gradient through the $\mathbf{f}_k$ branch as suggested by [20].

Training Schedule: We train RAFT using two 2080Ti GPUs. We pretrain on FlyingThings for 100k iterations with a batch size of 12, then train for 100k iterations on FlyingThings3D with a batch size of 6. We finetune on Sintel for another 100k by combining data from Sintel[11], KITTI-2015 [34], and HD1K[27] similar to MaskFlowNet [51] and PWC-Net+ [41]. Finally, we finetune on KITTI-2015 for an additionally 50k iterations using the weights from the model finetuned on Sintel. Details on training and data augmentation are provided in the supplemental material. For comparison with prior work, we also include results from our model when finetuning only on Sintel and only on KITTI.

4.1 Sintel

We train our model using the FlyingChairs→FlyingThings schedule and then evaluate on the Sintel dataset using the *train* split for validation. Results are shown in Table 1 and Figure 3, and we split results based on the data used for training. C + T means that the models are trained on FlyingChairs(C) and FlyingThings(T), while +ft indicates the model is finetuned on Sintel data. Like PWC-Net+[41] and MaskFlowNet [51] we include data from KITTI and HD1K

Fig. 4: Flow predictions on the KITTI test set.

when finetuning. We train 3 times with different seeds, and report results using the model with the median accuracy on the clean pass of Sintel (train).

When using C+T for training, our method outperforms all existing approaches, despite using a significantly shorter training schedule. Our method achieves an average EPE (end-point-error) of 1.43 on the Sintel(train) clean pass, which is a 29% lower error than FlowNet2. These results demonstrates good cross dataset generalization. One of the reasons for better generalization is the structure of our network. By constraining optical flow to be the product of a series of identical update steps, we force the network to learn an update operator which mimics the updates of a first-order descent algorithm. This constrains the search space, reduces the risk of over-fitting, and leads to faster training and better generalization.

When evaluating on the Sintel(test) set, we finetune on the combined clean and final passes of the training set along with KITTI and HD1K data. Our method ranks 1st on both the Sintel clean and final passes, and outperforms all prior work by 0.9 pixels (36%) on the clean pass and 1.2 pixels (30%) on the final pass. We evaluate two versions of our model, Ours (two-frame) uses zero initialization, while Ours (warp-start) initializes flow by forward projecting the flow estimate from the previous frame. Since our method operates at a single resolution, we can initialize the flow estimate to utilize motion smoothness from past frames, which cannot be easily done using the coarse-to-fine model.

4.2 KITTI

We also evaluate RAFT on KITTI and provide results in Table 1 and Figure 4. We first evaluate cross-dataset generalization by evaluating on the KITTI-15 (train) split after training on Chairs(C) and FlyingThings(T). Our method outperforms prior works by a large margin, improving EPE (end-point-error) from 8.36 to 5.04, which shows that the underlying structure of our network facilitates generalization. Our method ranks 1st on the KITTI leaderboard among all optical flow methods.

4.3 Ablations

We perform a set of ablation experiments to show the relative importance of each component. All ablated versions are trained on FlyingChairs(C) + FlyingThings(T). Results of the ablations are shown in Table 2. In each section of

Training Data	Method	Sintel (train)		KITTI-15 (train)		Sintel (test)		KITTI-15 (test)
		Clean	Final	F1-epe	F1-all	Clean	Final	F1-all
-	FlowFields[7]	-	-	-	-	3.75	5.81	15.31
-	FlowFields++[40]	-	-	-	-	2.94	5.49	14.82
S	DCFlow[47]	-	-	-	-	3.54	5.12	14.86
S	MRFlow[46]	-	-	-	-	2.53	5.38	12.19
C + T	HD3[49]	3.84	8.77	13.17	24.0	-	-	-
	LiteFlowNet[22]	2.48	4.04	10.39	28.5	-	-	-
	PWC-Net[42]	2.55	3.93	10.35	33.7	-	-	-
	LiteFlowNet2[23]	2.24	3.78	8.97	25.9	-	-	-
	VCN[48]	2.21	3.68	8.36	25.1	-	-	-
	MaskFlowNet[51]	2.25	3.61	-	23.1	-	-	-
	FlowNet2[25]	2.02	3.54[1]	10.08	30.0	3.96	6.02	-
	Ours (small)	2.21	3.35	7.51	26.9	-	-	-
	Ours (2-view)	**1.43**	**2.71**	**5.04**	**17.4**	-	-	-
C+T+S/K	FlowNet2 [25]	(1.45)	(2.01)	(2.30)	(6.8)	4.16	5.74	11.48
	HD3 [49]	(1.87)	(1.17)	(1.31)	(4.1)	4.79	4.67	6.55
	IRR-PWC [24]	(1.92)	(2.51)	(1.63)	(5.3)	3.84	4.58	7.65
	VCN [48]	(1.66)	(2.24)	(1.16)	(4.1)	2.81	4.40	**6.30**
	ScopeFlow[8]	-	-	-	-	3.59	4.10	6.82
	Ours (2-view, bilinear)	(1.09)	(1.53)	(1.07)	(3.9)	2.77	3.61	**6.30**
	Ours (warm-start, bilinear)	(1.10)	(1.61)	-	-	**2.42**	**3.39**	-
C+T+S+K+H	LiteFlowNet2[2] [23]	(1.30)	(1.62)	(1.47)	(4.8)	3.45	4.90	7.74
	PWC-Net+[41]	(1.71)	(2.34)	(1.50)	(5.3)	3.45	4.60	7.72
	MaskFlowNet[51]	-	-	-	-	2.52	4.17	6.10
	Ours (2-view)	(0.76)	(1.22)	(0.63)	(1.5)	1.94	3.18	**5.10**
	Ours (warm-start)	(0.77)	(1.27)	-	-	**1.61**	**2.86**	-

Table 1: Results on Sintel and KITTI datasets. We test the generalization performance on Sintel(train) after training on FlyingChairs(C) and FlyingThing(T), and outperform all existing methods on both the clean and final pass. The bottom two sections show the performance of our model on public leaderboards after dataset specific finetuning. S/K includes methods which use only Sintel data for finetuning on Sintel and only KITTI data when finetuning on KITTI. +S+K+H includes methods which combine KITTI, HD1K, and Sintel data when finetuning on Sintel. Ours (warm-start) ranks 1st on both the Sintel clean and final passes, and 1st among all flow approaches on KITTI. ([1]FlowNet2 originally reported results on the disparity split of Sintel, 3.54 is the EPE when their model is evaluated on the standard data [22]. [2] [23] finds that HD1K data does not help significantly during Sintel finetuning and reports results without it.)

the table, we test a specific component of our approach in isolation, the settings which are used in our final model is underlined. Below we describe each of the experiments in more detail.

Architecture of Update Operator: We use a gated activation unit based on the GRU cell. We experiment with replacing the convolutional GRU with a set of 3 convolutional layers with ReLU activation. We achieve better performance by using the GRU block, likely because the gated activation makes it easier for the sequence of flow estimates to converge.

Weight Tying: By default, we tied the weights across all instances of the update operator. Here, we test a version of our approach where each update operator learns a separate set of weights. Accuracy is better when weights are tied and the parameter count is significantly lower.

Experiment	Method	Sintel (train)		KITTI-15 (train)		Parameters
		Clean	Final	F1-epe	F1-all	
Reference Model (bilinear upsampling), Training: 100k(C) → 60k(T)						
Update Op.	ConvGRU	1.63	2.83	5.54	19.8	4.8M
	Conv	2.04	3.21	7.66	26.1	4.1M
Tying	Tied Weights	1.63	2.83	5.54	19.8	4.8M
	Untied Weights	1.96	3.20	7.64	24.1	32.5M
Context	Context	1.63	2.83	5.54	19.8	4.8M
	No Context	1.93	3.06	6.25	23.1	3.3M
Feature Scale	Single-Scale	1.63	2.83	5.54	19.8	4.8M
	Multi-Scale	2.08	3.12	6.91	23.2	6.6M
Lookup Radius	0	3.41	4.53	23.6	44.8	4.7M
	1	1.80	2.99	6.27	21.5	4.7M
	2	1.78	2.82	5.84	21.1	4.8M
	4	1.63	2.83	5.54	19.8	4.8M
Correlation Pooling	No	1.95	3.02	6.07	23.2	4.7M
	Yes	1.63	2.83	5.54	19.8	4.8M
Correlation Range	32px	2.91	4.48	10.4	28.8	4.8M
	64px	2.06	3.16	6.24	20.9	4.8M
	128px	1.64	2.81	6.00	19.9	4.8M
	All-Pairs	1.63	2.83	5.54	19.8	4.8M
Features for Refinement	Correlation	1.63	2.83	5.54	19.8	4.8M
	Warping	2.27	3.73	11.83	32.1	2.8M
Reference Model (convex upsampling), Training: 100k(C) → 100k(T)						
Upsampling	Convex	1.43	2.71	5.04	17.4	5.3M
	Bilinear	1.60	2.79	5.17	19.2	4.8M
Inference Updates	1	4.04	5.45	15.30	44.5	5.3M
	3	2.14	3.52	8.98	29.9	5.3M
	8	1.61	2.88	5.99	19.6	5.3M
	32	1.43	2.71	5.00	17.4	5.3M
	100	1.41	2.72	4.95	17.4	5.3M
	200	1.40	2.73	4.94	17.4	5.3M

Table 2: Ablation experiments. Settings used in our final model are underlined. See Sec. 4.3 for details.

Context: We test the importance of context by training a model with the context network removed. Without context, we still achieve good results, outperforming all existing works on both Sintel and KITTI. But context is helpful. Directly injecting image features into the update operator likely allows spatial information to be better aggregated within motion boundaries.

Feature Scale: By default, we extract features at a single resolution. We also try extracting features at multiple resolutions by building a correlation volume at each scale separately. Single resolution features simplifies the network architecture and allows fine-grained matching even at large displacements.

Lookup Radius: The lookup radius specifies the dimensions of the grid used in the lookup operation. When a radius of 0 is used, the correlation volume is retrieved at a single point. Surprisingly, we can still get a rough estimate of flow when the radius is 0, which means the network is learning to use 0'th order information. However, we see better results as the radius is increased.

Correlation Pooling: We output features at a single resolution and then perform pooling to generate multiscale volumes. Here we test the impact when

this pooling is removed. Results are better with pooling, because large and small displacements are both captured.

Correlation Range: Instead of all-pairs correlation, we also try constructing the correlation volume only for a local neighborhood around each pixel. We try a range of 32 pixels, 64 pixels, and 128 pixels. Overall we get the best results when the all-pairs are used, although a 128px range is sufficient to perform well on Sintel because most displacements fall within this range. That said, all-pairs is still preferable because it eliminates the need to specify a range. It is also more convenient to implement: it can be computed using matrix multiplication allowing our approach to be implemented entirely in PyTorch.

Features for Refinement: We compute visual similarity by building a correlation volume between all pairs of pixels. In this experiment, we try replacing the correlation volume with a warping layer, which uses the current estimate of optical flow to warp features from I_2 onto I_1 and then estimates the residual displacement. While warping is still competitive with prior work on Sintel, correlation performs significantly better, especially on KITTI.

Features for Refinement: We compute visual similarity by building a correlation volume between all pairs of pixels. In this experiment, we try replacing the correlation volume with a warping layer, which uses the current estimate of optical flow to warp features from I_2 onto I_1 and then estimates the residual displacement. While warping is still competitive with prior work on Sintel, correlation performs significantly better, especially on KITTI.

Upsampling: RAFT outputs flow fields at 1/8 resolution. We compare bilinear upsampling to our learned upsampling module. The upsampling module produces better results, particularly near motion boundaries.

Inference Updates: Although we unroll 12 updates during training, we can apply an arbitrary number of updates during inference. In Table 2 we provide numerical results for selected number of updates, and test an extreme case of 200 to show that our method doesn't diverge. Our method quickly converges, surpassing PWC-Net after 3 updates and FlowNet2 after 6 updates, but continues to improve with more updates.

4.4 Timing and Parameter Counts

Inference time and parameter counts are shown in Figure 5. Accuracy is determined by performance on the Sintel(train) final pass after training on FlyingChairs and FlyingThings (C+T). In these plots, we report accuracy and timing after 10 iterations, and we time our method using a GTX 1080Ti GPU. Parameters counts for other methods are taken as reported in their papers, and we report times when run on our hardware. RAFT is more efficient in terms of parameter count, inference time, and training iterations. Ours-S uses only 1M parameters, but outperforms PWC-Net and VCN which are more than 6x larger. We provide an additional table with numerical values for parameters, timing, and training iterations in the supplemental material.

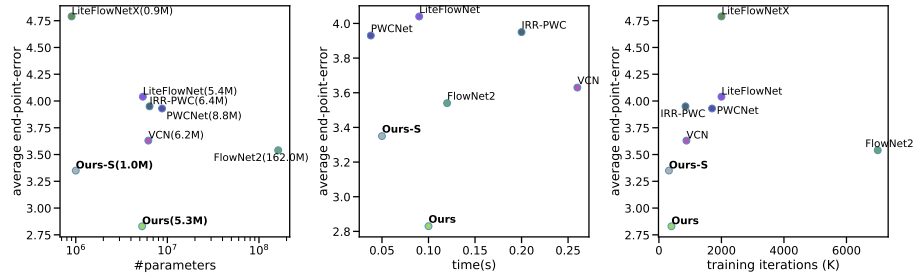

Fig. 5: Plots comparing parameter counts, inference time, and training iterations vs. accuracy. Accuracy is measured by the EPE on the Sintel(train) final pass after training on C+T. *Left*: Parameter count vs. accuracy compared to other methods. RAFT is more parameter efficient while achieving lower EPE. *Middle*: Inference time vs. accuracy timed using our hardware *Right*: Training iterations vs. accuracy (taken as product of iterations and GPUs used).

Fig. 6: Results on 1080p (1088x1920) video from DAVIS (550 ms per frame).

4.5 Video of Very High Resolution

To demonstrate that our method scales well to videos of very high resolution we apply our network to HD video from the DAVIS[37] dataset. We use 1080p (1088x1920) resolution video and apply 12 iterations of our approach. Inference takes 550ms for 12 iterations on 1080p video, with all-pairs correlation taking 95ms. Fig. 6 visualizes example results on DAVIS.

5 Conclusions

We have proposed RAFT—Recurrent All-Pairs Field Transforms—a new end-to-end trainable model for optical flow. RAFT is unique in that it operates at a single resolution using a large number of lightweight, recurrent update operators. Our method achieves state-of-the-art accuracy across a diverse range of datasets, strong cross dataset generalization, and is efficient in terms of inference time, parameter count, and training iterations.

Acknowledgments: This work was partially funded by the National Science Foundation under Grant No. 1617767.

References

1. Adler, J., Öktem, O.: Solving ill-posed inverse problems using iterative deep neural networks. Inverse Problems **33**(12), 124007 (2017)
2. Adler, J., Öktem, O.: Learned primal-dual reconstruction. IEEE transactions on medical imaging **37**(6), 1322–1332 (2018)
3. Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., Kolter, J.Z.: Differentiable convex optimization layers. In: Advances in Neural Information Processing Systems. pp. 9558–9570 (2019)
4. Amos, B., Kolter, J.Z.: Optnet: Differentiable optimization as a layer in neural networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 136–145. JMLR. org (2017)
5. Bai, S., Kolter, J.Z., Koltun, V.: Trellis networks for sequence modeling. arXiv preprint arXiv:1810.06682 (2018)
6. Bai, S., Kolter, J.Z., Koltun, V.: Deep equilibrium models. In: Advances in Neural Information Processing Systems. pp. 688–699 (2019)
7. Bailer, C., Taetz, B., Stricker, D.: Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In: Proceedings of the IEEE international conference on computer vision. pp. 4015–4023 (2015)
8. Bar-Haim, A., Wolf, L.: Scopeflow: Dynamic scene scoping for optical flow. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7998–8007 (2020)
9. Black, M.J., Anandan, P.: A framework for the robust estimation of optical flow. In: 1993 (4th) International Conference on Computer Vision. pp. 231–236. IEEE (1993)
10. Brox, T., Bregler, C., Malik, J.: Large displacement optical flow. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 41–48. IEEE (2009)
11. Butler, D.J., Wulff, J., Stanley, G.B., Black, M.J.: A naturalistic open source movie for optical flow evaluation. In: European conference on computer vision. pp. 611–625. Springer (2012)
12. Chambolle, A., Pock, T.: A first-order primal-dual algorithm for convex problems with applications to imaging. Journal of mathematical imaging and vision **40**(1), 120–145 (2011)
13. Chen, Q., Koltun, V.: Full flow: Optical flow estimation by global optimization over regular grids. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4706–4714 (2016)
14. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259 (2014)
15. Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., Brox, T.: Flownet: Learning optical flow with convolutional networks. In: Proceedings of the IEEE international conference on computer vision. pp. 2758–2766 (2015)
16. Fan, L., Huang, W., Gan, C., Ermon, S., Gong, B., Huang, J.: End-to-end learning of motion representation for video understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6016–6025 (2018)
17. Flynn, J., Broxton, M., Debevec, P., DuVall, M., Fyffe, G., Overbeck, R.S., Snavely, N., Tucker, R.: Deepview: High-quality view synthesis by learned gradient descent (2019)

18. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The kitti dataset. The International Journal of Robotics Research **32**(11), 1231–1237 (2013)
19. Hirschmuller, H.: Stereo processing by semiglobal matching and mutual information. IEEE Transactions on pattern analysis and machine intelligence **30**(2), 328–341 (2007)
20. Hofinger, M., Bulò, S.R., Porzi, L., Knapitsch, A., Kontschieder, P.: Improving optical flow on a pyramidal level. In: ECCV (2020)
21. Horn, B.K., Schunck, B.G.: Determining optical flow. In: Techniques and Applications of Image Understanding. vol. 281, pp. 319–331. International Society for Optics and Photonics (1981)
22. Hui, T.W., Tang, X., Change Loy, C.: Liteflownet: A lightweight convolutional neural network for optical flow estimation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8981–8989 (2018)
23. Hui, T.W., Tang, X., Loy, C.C.: A lightweight optical flow cnn–revisiting data fidelity and regularization. arXiv preprint arXiv:1903.07414 (2019)
24. Hur, J., Roth, S.: Iterative residual refinement for joint optical flow and occlusion estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5754–5763 (2019)
25. Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T.: Flownet 2.0: Evolution of optical flow estimation with deep networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2462–2470 (2017)
26. Kobler, E., Klatzer, T., Hammernik, K., Pock, T.: Variational networks: connecting variational methods and deep learning. In: German conference on pattern recognition. pp. 281–293. Springer (2017)
27. Kondermann, D., Nair, R., Honauer, K., Krispin, K., Andrulis, J., Brock, A., Gussefeld, B., Rahimimoghaddam, M., Hofmann, S., Brenner, C., et al.: The hci benchmark suite: Stereo and flow ground truth with uncertainties for urban autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 19–28 (2016)
28. Li, X., Wu, J., Lin, Z., Liu, H., Zha, H.: Recurrent squeeze-and-excitation context aggregation net for single image deraining. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 254–269 (2018)
29. Liang, Z., Feng, Y., Guo, Y., Liu, H., Chen, W., Qiao, L., Zhou, L., Zhang, J.: Learning for disparity estimation through feature constancy. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2811–2820 (2018)
30. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)
31. Lu, Y., Valmadre, J., Wang, H., Kannala, J., Harandi, M., Torr, P.: Devon: Deformable volume network for learning optical flow. In: The IEEE Winter Conference on Applications of Computer Vision. pp. 2705–2713 (2020)
32. Lv, Z., Dellaert, F., Rehg, J.M., Geiger, A.: Taking a deeper look at the inverse compositional algorithm. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4581–4590 (2019)
33. Mayer, N., Ilg, E., Hausser, P., Fischer, P., Cremers, D., Dosovitskiy, A., Brox, T.: A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4040–4048 (2016)

34. Menze, M., Geiger, A.: Object scene flow for autonomous vehicles. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3061–3070 (2015)
35. Menze, M., Heipke, C., Geiger, A.: Discrete optimization for optical flow. In: German Conference on Pattern Recognition. pp. 16–28. Springer (2015)
36. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
37. Pont-Tuset, J., Perazzi, F., Caelles, S., Arbeláez, P., Sorkine-Hornung, A., Van Gool, L.: The 2017 davis challenge on video object segmentation. arXiv preprint arXiv:1704.00675 (2017)
38. Ranftl, R., Bredies, K., Pock, T.: Non-local total generalized variation for optical flow estimation. In: European Conference on Computer Vision. pp. 439–454. Springer (2014)
39. Ranjan, A., Black, M.J.: Optical flow estimation using a spatial pyramid network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4161–4170 (2017)
40. Schuster, R., Bailer, C., Wasenmüller, O., Stricker, D.: Flowfields++: Accurate optical flow correspondences meet robust interpolation. In: 2018 25th IEEE International Conference on Image Processing (ICIP). pp. 1463–1467. IEEE (2018)
41. Sun, D., Yang, X., Liu, M.Y., Kautz, J.: Models matter, so does training: An empirical study of cnns for optical flow estimation. arXiv preprint arXiv:1809.05571 (2018)
42. Sun, D., Yang, X., Liu, M.Y., Kautz, J.: Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8934–8943 (2018)
43. Tang, C., Tan, P.: Ba-net: Dense bundle adjustment network. arXiv preprint arXiv:1806.04807 (2018)
44. Teed, Z., Deng, J.: Deepv2d: Video to depth with differentiable structure from motion. arXiv preprint arXiv:1812.04605 (2018)
45. Weinzaepfel, P., Revaud, J., Harchaoui, Z., Schmid, C.: Deepflow: Large displacement optical flow with deep matching. In: Proceedings of the IEEE international conference on computer vision. pp. 1385–1392 (2013)
46. Wulff, J., Sevilla-Lara, L., Black, M.J.: Optical flow in mostly rigid scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4671–4680 (2017)
47. Xu, J., Ranftl, R., Koltun, V.: Accurate optical flow via direct cost volume processing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1289–1297 (2017)
48. Yang, G., Ramanan, D.: Volumetric correspondence networks for optical flow. In: Advances in Neural Information Processing Systems. pp. 793–803 (2019)
49. Yin, Z., Darrell, T., Yu, F.: Hierarchical discrete distribution decomposition for match density estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6044–6053 (2019)
50. Zach, C., Pock, T., Bischof, H.: A duality based approach for realtime tv-l 1 optical flow. In: Joint pattern recognition symposium. pp. 214–223. Springer (2007)
51. Zhao, S., Sheng, Y., Dong, Y., Chang, E.I., Xu, Y., et al.: Maskflownet: Asymmetric feature matching with learnable occlusion mask. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6278–6287 (2020)
52. Zhou, H., Ummenhofer, B., Brox, T.: Deeptam: Deep tracking and mapping. In: Proceedings of the European conference on computer vision (ECCV). pp. 822–838 (2018)