CCMR: High Resolution Optical Flow Estimation via Coarse-to-Fine Context-Guided Motion Reasoning Supplementary Material

Azin Jahedi[1] Maximilian Luz[2] Marc Rivinius[3] Andrés Bruhn[1]
[1]VIS, University of Stuttgart [2]RLL, University of Freiburg [3]SEC, University of Stuttgart
{azin.jahedi,andres.bruhn}@vis.uni-stuttgart.de
luz@cs.uni-freiburg.de, marc.rivinius@sec.uni-stuttgart.de

In the following, we provide additional details on our approach. Regarding the architecture, we further detail on the improved feature consolidation unit and how it is embedded in the whole feature extractor. We then show additional pre-training generalization results, comparing our approach to its baselines. After that we give additional details on the memory advantages of our approach compared to a simple coarse-to-fine GMA variant. We then describe the validation set on KITTI that we used for delineate the architectural from the fine-tuning benefits. Afterwards, we provide the inference settings of our approach and finally comment on practical aspects such as memory requirements and runtime.

I. Improved Feature Extraction

In Section 3 of the main paper, we gave a brief description of the U-Net style feature extractor[1], and how the feature consolidation unit has been modified to increase its expressiveness and decrease its size. Let us now elaborate on the whole multi-scale feature extractor with the improved feature consolidation unit in more detail.

Fig. I shows the architecture of the modified encoder for computing the image features. First, intermediate features are computed in a top-down manner which are denoted by $\hat{F}_2, \hat{F}_3$ and $\hat{F}_4$. Then to obtain multi-scale features, the better-structured finer features are semantically enhanced by consolidating them with the deeper coarser-scale features. To this end, bilinearly upsampled coarser features F_{s-1} and intermediate finer features $\hat{F}_s$ are stacked and passed through a Res-Conv unit for consolidation. As the discussed in the paper, the Conv layer after the residual unit is activation-free. Note that two different encoders with the shown architecture are used for computing image and context features. While the one for the image features is shared among the two input images, the one for the context features has independent weights. Fig. 3 in the main paper

[1] which is based on MS-RAFT(+) [2, 3]

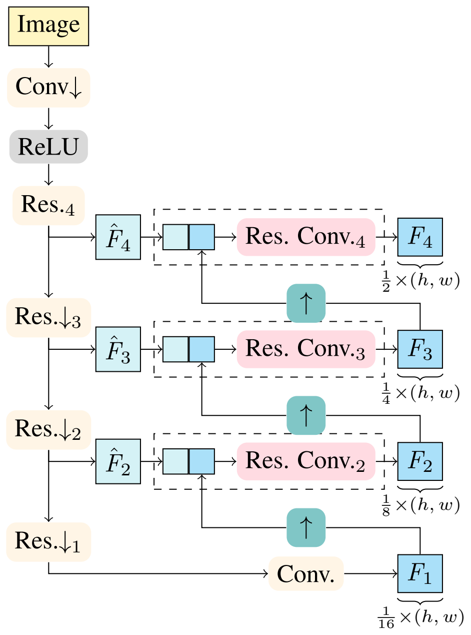

Figure I. Our modified U-Net style feature extractor. Note that all convs shown in the figure are activation free. The red highlighted modules is our small but effective modification.

shows the feature consolidation unit for each of the features $F_{s,1}, F_{s,2}, C_s$ but only for one scale (scale s), unlike what is illustrated in Fig. I which shows the feature computation for different scales. Note that the each of the dashed boxes in Fig. I corresponds to feature consolidation in Fig. 3 of the main paper at a fixed scale s.

In the multi-scale estimation framework, to share the recurrent unit among all coarse-to-fine scales, the context features must have a fixed number of channels across scales (as they serve as input). In [2, 3], this was realized by a large residual unit for consolidation to keep the output channels 256 at all scales. Using a leaner residual unit and converting the number of channels to 256 using the subsequent conv in the consolidation unit, our optical flow model became

Table I. Pre-training results. Results on *Sintel (train)* and *KITTI (train)* after pre-training with *Chairs* and *Things*.

	Sintel (train)		KITTI (train)
Method	Clean↓	Final↓	Fl↓
RAFT	1.43	2.71	17.4
GMA	1.30	2.74	17.1
DIP	1.30	2.82	13.73
KPA-flow	1.28	2.68	15.9
AnyFlow	1.17	2.58	<u>13.01</u>
MS-RAFT	1.13	2.60	-
MatchFlow_GMA	1.03	2.45	15.6
Flowformer++	**0.90**	**2.30**	14.13
GMFlow+	<u>0.91</u>	-	-
GMA	1.36	2.74	17.51
MS-RAFT	1.18	2.58	13.84
MS-RAFT+	1.03	2.52	13.83
CCMR (ours)	1.07	2.40	13.30
CCMR+ (ours)	0.98	<u>2.36</u>	**12.84**

28% smaller. Our leaner residual unit has two residual layers with less intermediate and output channels. Following our ablation strategy from Sec. 4 of the main paper which considers the 3-scale variant, our model with smaller residual unit yields pre-training generalization results of 1.07 for Sintel Clean and 2.40 for Final, which is a bit worse than 1.05, obtained by the larger model for Clean and better than its Final results: 2.49, respectively. Therefore the results of the smaller model were favorable on average.

II. Pre-Training Generalization Results

In Section 4.2 of the main paper, we compared the pre-training generalization results of our method to those of the baselines and of recent methods; see main paper Tab. 2. The results reported in the respective publications, based on those methods using their *own* training schedules, are listed again in the upper part of Tab. I. Additionally, in the lower part of Tab. I, we now compare our results to the MS-RAFT(+) and GMA baselines for which, this time, we applied the training schedule of *our* method, i.e. all methods have exactly the same training scheme. Also in this case, our CCMR(+) method outperforms the corresponding baselines, indicating that its improved pre-training generalization performance is not due to the different training setting.

III. CCMR vs. Coarse-to-Fine GMA

In Section 3.4 of the main paper, we outlined the infeasibility of a simple coarse-to-fine version of GMA. We demonstrated that even inference in such a network needs an enormous amounts of VRAM. In the following we add upon this by elaborating on the memory requirements to train such a network. Considering patch sizes used by RAFT's [5] training setting, the minimum and maximum

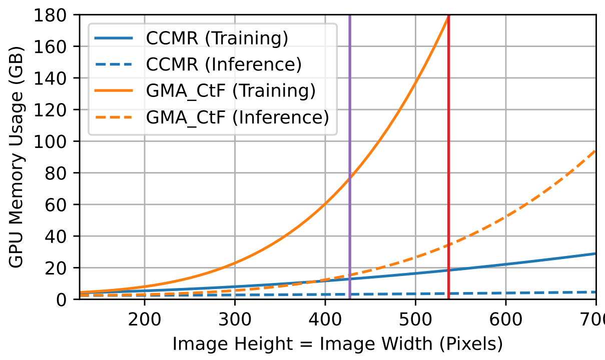

Figure II. Comparison of memory consumption for our approach and the direct extension of GMA to multiple scales[3]. X-axis shows height or width of squared input images.

size are $368 \times 496 \approx 427^2$ and $400 \times 720 \approx 536^2$, used in the first and second training stages, respectively. These patch sizes were also used for training GMA [4] and our CCMR approach. Now let us take a look at the memory consumption for training **a single sample (batch of one)** shown in Fig. II. Considering the intersections of the solid orange line with the purple (patch size $\approx 427^2$) and red line (patch size of $\approx 536^2$) shows that the coarse-to-fine version of GMA requires from 75 to about 180 GBs of VRAM. This means: training this network using GMA's training setting[2] requires $8 \times 75 = 600$ GBs and about $6 \times 180 = 1080$ GBs for training the first stage on Chairs and second stage on Things, respectively, while our approach requires more than an order of magnitude less VRAM for training the second phase using the same patch size and batch size. In fact our approach needs at most 94 GBs which we distribute over three Nvidia A100 GPUs, each having 40 GBs of VRAM. Note that in the measurements shown in Fig. II, the matching costs were computed on demand, without the computation of the all-pairs cost volume, therefore the consumed memory is not due to computing or keeping the 4D all-pairs cost volume in the VRAM, but for different partially attention-based coarse-to-fine calculations.

Of course, when the memory requirements become critical, using smaller patches for training can be taken into account. However, since using larger patches for training provides more content for the network, reducing the patch size drastically during training is not advisable [1].

[2]GMA uses a batch of 8 samples for the first training phase and a batch size of 6 for the other training phases.

[3]Note that, for GMA_CtF, we ran samples for inference and training for patch sizes which required memory up to the memory limit of our GPU—about 40 GBs ($\approx 352^2$ for training and 544^2 for inference). In this case, the memory consumption for larger patches was extrapolated using a quadratic function.

IV. Validation Set for KITTI 2015

In Section 4.2 of the main paper, we delineated the architectural impact from the fine-tuning impact on KITTI. To this end we split KITTI 2015 (train) into a training and a validation set. The validation set contained the following 50 sequences: #0-#29, #60-#69 and #110-#119 and the train-split consists of the remaining 150 samples. Using a fine-tuned version on such a split did not only allow us to compare the different baseline in a fair manner, but also to find effective number of iterations during inference.

V. Inference Settings

Similar to RAFT [5] and and many of its successors, such as GMA [4], we used different **inference** settings for the different benchmarks, however, as mentioned in the paper, we used one unified architecture[4] for each 3-scale and 4-scale variant for different pre-training and fine-tuning stages.

To compute the flow on the Sintel benchmark, we applied a cold warm-strategy as in MS-RAFT+ [2], using 8 GRU iterations at the coarsest scale and 10 at other scales. For KITTI, 35 iterations at the two coarsest scales, 5 and 15 iterations at the next finer scales are performed, respectively, for the 4-scale model. These iteration numbers were found empirically on the training set of Sintel for models pre-trained on Chairs and Things, and for KITTI on the KITTI validation split after fine-tuning the model on the training-split previously described in Sec. IV. In the case of our 3-scale model, we used 10, 15 and 20 GRU iterations for Sintel and 6, 18 and 30 iterations for KITTI from the coarsest to the finest scale, respectively.

Importantly, the optimal number of iterations for the best performance highly depends on the model. Therefore, we investigated different combinations of GRU iterations per scale for both our method and our baselines and reported the best results in the lower part of Tab. I. In the same way, we performed our comparison to the baselines for the fine-tuned models on the KITTI validation split in Tab. 3 of the main paper. Therefore our clear advantage over the baselines is independent of the training or inference settings. In our investigation, we found that in the case of KITTI, our model benefits from more GRU iterations than MS-RAFT(+), where MS-RAFT+ results degrades in accuracy using more than 10 GRU iterations per-scale while the accuracy of our method kept increasing. In the case of GMA, the accuracy barely changed by increasing the GRU iterations.

VI. Memory Requirements and Runtime

Estimating the flow using our 4-scale model on Sintel-size images (436×1024) takes about 0.9 seconds using 8, 10, 10 and 10 GRU iterations from the coarsest to the finest scale, respectively. This estimation requires 4.5 GBs of VRAM. Note that, in this setting, the matching costs are computed on-demand for each iteration. Analogously, estimating the flow using our 3-scale model takes 0.55 seconds and requires 3.1 GBs of VRAM when performing 10, 15 and 20 GRU iterations from the coarsest to finest scale, respectively. Note that, in this case, as the finest matching scale of the 3-scale model is at $\frac{1}{4}(h,w)$ instead of at $\frac{1}{2}(h,w)$ in the 4-scale model, the computation of the all-pair cost volume is rather affordable. When doing so, the runtime reduces to 0.43 seconds, while the amount of required VRAM increases to 12.5 GB.

[4] which was suggested by our ablations: last two rows in Tab. 5 in the main paper.

References

[1] A. Bar-Haim and L. Wolf. ScopeFlow: dynamic scene scoping for optical flow. In *CVPR*, pages 7995–8004, 2020. 2

[2] A. Jahedi, M. Luz, L. Mehl, M. Rivinius, and A. Bruhn. High resolution multi-scale RAFT (ECCV Robust Vision Challenge 2022). *arXiv:2210.16900*, 2022. 1, 3

[3] A. Jahedi, L. Mehl, M. Rivinius, and A. Bruhn. Multi-Scale RAFT: Combining hierarchical concepts for learning-based optical flow estimation. In *ICIP*, pages 1236–1240, 2022. 1

[4] S. Jiang, D. Campbell, Y. Lu, H. Li, and R. Hartley. Learning to estimate hidden motions with global motion aggregation. In *ICCV*, pages 9772–9781, 2021. 2, 3

[5] Z. Teed and J. Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *ECCV*, pages 402–419, 2020. 2, 3