Supplementary Material: Deep Discrete Flow

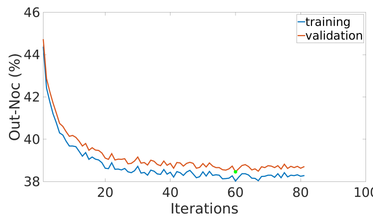
Fatma Güney[1] and Andreas Geiger[1,2]

[1]Autonomous Vision Group, MPI for Intelligent Systems, Tübingen
[2]Computer Vision and Geometry Group, ETH Zürich

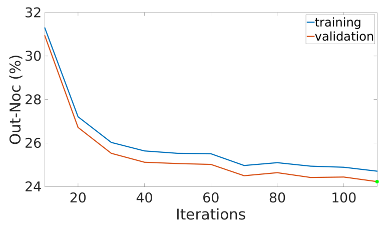
Abstract. In this **supplementary document**, we first present additional experiments to show the convergence of training in Section 1.1 and compare the performance of exact matching against a frequently used approximate nearest neighbour search library in Section 1.2. We then show the speed-ups we gain by the strided implementation of the patch-based dilated convolutions in Section 2.1. We compare the runtime of forward propagating a full resolution image pair for the different local and context architectures in Section 2.2 and then report the runtime for individual stages of our pipeline compared to Discrete Flow in Section 2.3. For reproducibility, we list the parameter settings of the different stages in Section 3. Finally, we show additional qualitative results on the MPI Sintel and KITTI validation sets in Section 4.

1 Additional Experiments

1.1 Convergence



(a) **Local Network**



(b) **Context Network**

Fig. 1: **Outlier ratio in non-occluded regions over training iterations.** This figure shows the average outlier ratio in non-occluded regions over the training iterations on KITTI.

In Fig. 1, we plot the average outlier ratio in non-occluded regions over the training iterations for both local and context networks on the KITTI training and validation sets. As concluded in the main paper, we also note that training the context network on top of the local network decreases outlier ratios significantly. Importantly, the training and validation curves behave similarly for both networks, indicating that our Siamese architectures don't suffer from overfitting. In both plots, one iteration corresponds to one hundred thousand batch updates.

1.2 Approximate vs. Exact Matching

Formulating optical flow as an approximate nearest neighbour (ANN) search on the target image is a common way of dealing with the size of the search space [1–3]. In this section, we show that exact matching can be done as efficiently utilizing the GPU and show improved results compared to ANN.

We can perform exact matching for grid points on the reference image efficiently as matrix-matrix multiplication on the GPU. However, there are still some restrictions due to the limited memory which prevent performing the entire computation at once. We show our experiments by dividing the set of grid point into chunks and changing the number of chunks to find a compensation between memory usage and run-time for different number of matches per grid point in Table 1.

Number of Chunks	Memory	Run-time
9	6.63	8.42
18	3.89	8.34
45	2.25	3.03
90	1.71	3.58
180	1.43	4.52
306	1.32	6.94

(a) **Number of Matches = 1**

Number of Chunks	Memory	Run-time
9	6.70	5.58
18	3.94	5.74
45	2.29	6.53
90	1.74	6.94
180	1.46	7.36
306	1.35	8.42

(b) **Number of Matches = 1024**

Table 1: **Exact Matching.** This table compares different number of chunks used to compute the exact matching in terms of memory and run-time for two different number of matches, 1 as in WTA and 1024 as used in BP.

For comparison, we use the popular FLANN library [4] (which has also been used in [3]) to perform approximate nearest neighbor matching. We found that the FLANN parameter "number of checks" which specifies the maximum number of leafs to visit when searching for neighbors has a clear effect on the performance. Therefore, we vary the number of checks in Table 2, fixing the number of search trees to 8. For both datasets, only using 1024 checks yields performance comparable to exact matching. We also note that run-time for FLANN is much higher, 45 to 60 seconds depending on the number of checks, when using full resolution images and a large number of feature maps, 256 as in case of context architecture 12.

Matching	Number of Checks	Out-Noc
	32	30.25%
FLANN	128	21.21 %
	1024	14.56 %
Exact	-	12.19 %

(a) **MPI Sintel**

Matching	Number of Checks	Out-Noc
	32	56.09%
FLANN	128	41.09 %
	1024	26.19 %
Exact	-	20.28 %

(b) **KITTI**

Table 2: **Comparison to FLANN.** This table compares the performance of exact matching to approximate matching using the FLANN library [4], varying the number of checks which specify the maximum number of leafs to visit.

2 Run-time Analysis

2.1 Run-time of Patch-Based Dilated Convolutional Networks

Context Arch.	Naïve	Proposed
1	543.37 ms	15.17 ms
2	146.69 ms	4.80 ms
3	58.37 ms	2.23 ms
4	360.41 ms	6.40 ms
5	195.22 ms	3.60 ms
6	63.47 ms	2.27 ms
7	170.60 ms	7.30 ms
8	646.11 ms	38.77 ms
9	1384.48 ms	11.00 ms
11	823.37 ms	42.46 ms
12	184.88 ms	9.28 ms
13	69.17 ms	2.79 ms
14	417.39 ms	11.42 ms
15	207.55 ms	4.21 ms
16	95.34 ms	4.41 ms
17	201.37 ms	9.39 ms
18	867.846 ms	55.78 ms
19	2335.74 ms	26.96 ms

Table 3: **Patch-based Dilated Convolutions.** This table compares the run-time of one batch update in milliseconds for the naïve and the proposed implementations using different context architectures with dilated convolutions.

As mentioned in Section 3.1 of the paper, a naïve implementation of patch based dilated convolutions is computationally very inefficient. In Table 3, we show the speed-ups we gain by the proposed strided convolution implementation. We measure the average run-time of forward and backward propagation of a batch of 128 patch pairs followed by normalization and similarity score computation. As evidenced by our experiments, the proposed implementation leads to run-time gains up to two orders of magnitude compared to the naïve implementation. In other words, using the naïve implementation would require 200 days for training a model which we train in 2 days using the proposed implementation for some architectures.

2.2 Run-time Comparison of Different Architectures

In Table 4, we compare the run-time of different local and context architectures for one forward pass of two full-resolution images. For context, we only report the additional time after the local architecture 1. Note that this number would be different depending on the number of feature maps in the last layer of the local architecture. Using the context architecture 12 on top of local architecture 1, run-time is approximately 2 seconds in total.

2.3 Run-time of Full Deep Discrete Flow Model

In Section 2.2, we compared the run-time of different architectures. In this section, we fix the local and context architectures to 1 and 12, respectively, and perform a run-time analysis of the following stages in our pipeline. In Table 5, we list the run-time for different parts of the discrete inference together with

Local Arch.	Run-time
1	1.27
2	1.89
3	2.08
4	2.52
5	2.62

Context Arch.	Run-time
1	1.06
2	0.76
3	0.45
4	0.76
5	0.46
6	0.45
7	1.06
8	1.67
9	1.06

Context Arch.	Run-time
11	1.63
12	0.87
13	0.46
14	0.88
15	0.46
16	0.51
17	1.13
18	2.10
19	1.46

Table 4: **Run-time (in sec) of Different Local and Context Architectures** This table compares the run-time of forward propagating two full-resolution Sintel images using different architectures.

the post-processing and the interpolation. Since these measurements are image dependent, we take the average run-time per stage over a set of images in the validation set of each dataset.

We pre-compute pairwise term using parallelism and hashing as in [3], and use an efficient implementation of Belief Propagation (BP) based on parallel checkerboard update scheme. The parallel BP has a comparable run-time to the Block Coordinate Descent used in [3] which takes 20 to 25 seconds in total for inference including the pairwise computation. As shown in Table 5, the slowest part is refining the data term using non-maxima suppression and adding proposals from the neighbouring pixels. The post-processing and interpolation run-times are negligible compared to the discrete inference.

	Refining Data Term	Pre-computing Pairwise	Belief Propagation	Post-processing	Interpolation
MPI Sintel	49.97	10.13	5.24	0.002	2.20
KITTI	35.42	14.17	10.54		1.66

Table 5: **Run-time (in sec) Analysis of Individual Stages** This table lists the run-time for different parts of the discrete inference as average over a set of validation images.

3 Parameter Settings

In this section, we provide the details of the parameter settings used to produce the results in the paper. Specifically, we list the parameters used in the feature matching baseline, discrete optimization, and the post-processing.

Table 6 lists the parameters of the Daisy feature descriptor used as baseline in Section 4.1 in the paper. The first column corresponds to the parameters used in Discrete Flow [3] and the second column to the optimized parameters on MPI Sintel.

Table 7 shows the parameters of the discrete optimization and the post-processing. We optimized over these parameters using BCD on the validation

Parameter	DF [3]	Optimized
Radius	5	10
Radius Quantization No.	4	4
Angular Quantization No.	4	8
Histogram Quantization No.	4	8
Normalization Type	2	2

Table 6: **Feature Matching Baseline Parameters.** Here, we show the values of the parameters from Discrete Flow [3] and optimized for the feature matching baseline used in the paper.

sets as defined in the paper, to minimize EPE in case of MPI Sintel and the average outlier ratio in case of KITTI as evaluated by the benchmarks. We empirically also tested larger values for the size of the label set by keeping the ratio of number of proposals from neighbors fixed, but did not observe a significant change in the performance, therefore we fixed the size of the label set to 300 during the optimization. In addition, we kept the NMS radius and the stride fixed. We observed that more iterations or a larger truncation threshold of the pairwise term give slightly better results, but at the cost of higher run-time. For interpolation, we used the default values as defined in Epic Flow for each dataset.

Parameter	Value
Number of Initial Matches	1024
Size of the Label Set	300
Proposals from Neighbours	210
NMS Radius in Pixels	2
Stride for Discrete CRF in Pixels	4
Truncation Threshold of the Data Term	-0.25
Truncation Threshold of the Pairwise Term	15
Relative Weight of Pairwise Term	0.009
Number of Iterations	10

(a) **Discrete Optimization Parameters**

Parameter	MPI Sintel	KITTI
Similarity Threshold (fw-/bw-check)	7.63	2.83
Minimum Segment Size (outlier removal)	100	277.45
Consistency Threshold (outlier removal)	5	13.01
Epic Flow	dataset	

(b) **Post-processing Parameters**

Table 7: **Model Parameters.** Here, we list the parameter settings used for discrete optimization in (a), and for post-processing and interpolation in (b).

4 Additional Qualitative Results

In this section, we show additional qualitative results which we **randomly picked** from the MPI Sintel and the KITTI validation sets. For each case, we show the input image, ground-truth flow in the first row, results for Discrete Flow with Daisy Proposals in the second row, local network in the third row, and the context network in the last row. The first double column shows the flow result and corresponding error image for the WTA solution, the second double column shows the results after discrete optimization and the last double column shows the final results after post-processing and interpolation.

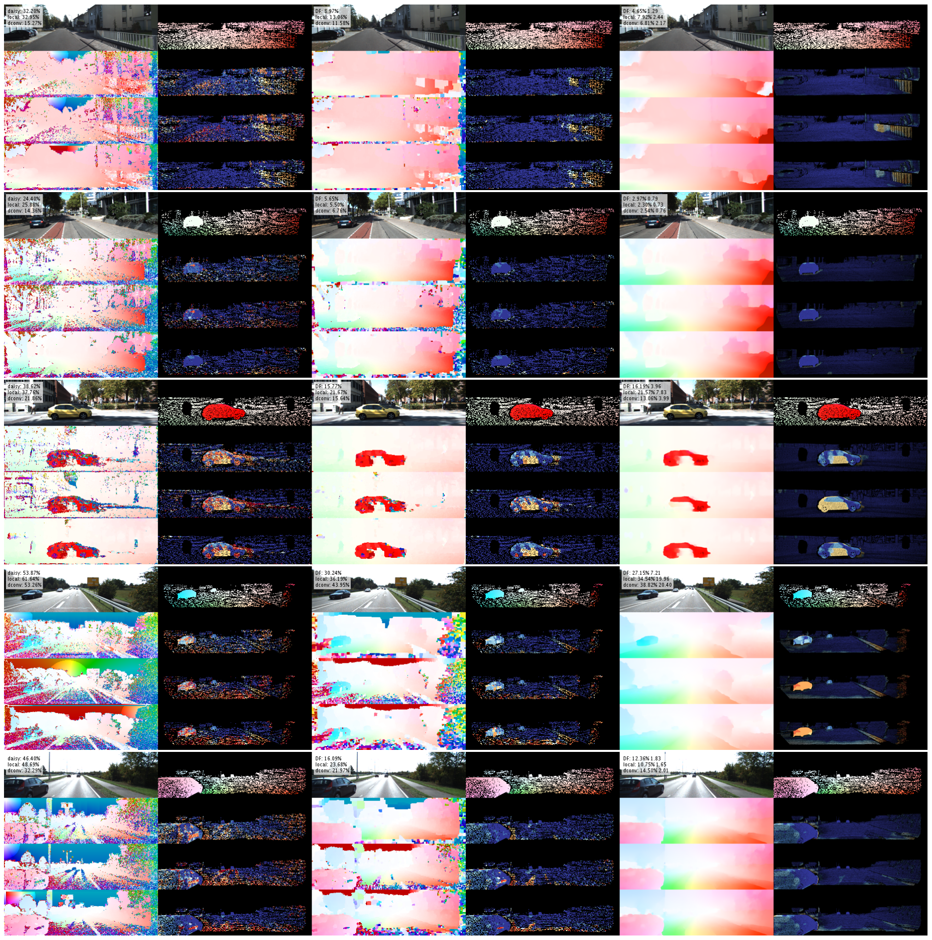


Fig. 2: **Qualitiative Results.** See text for details.

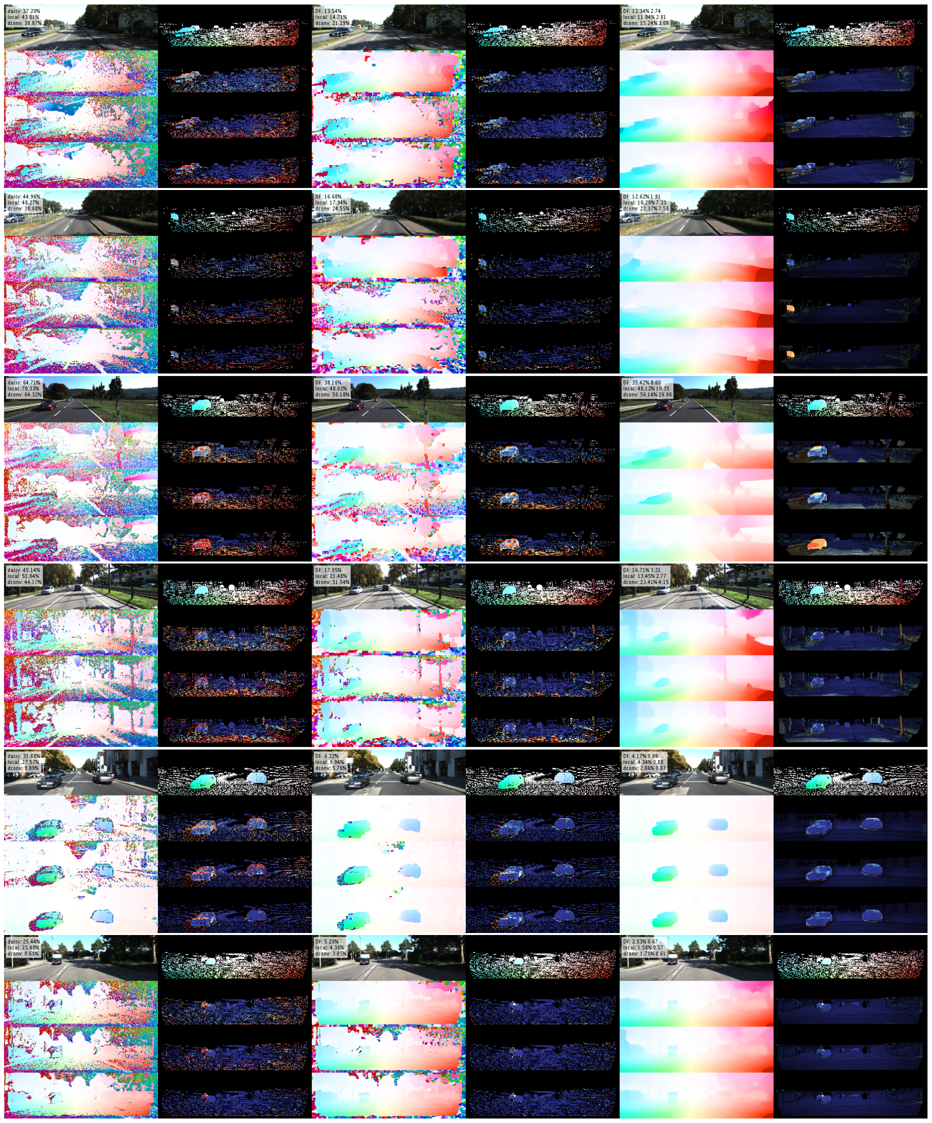


Fig. 3: **Qualitiative Results.** See text for details.

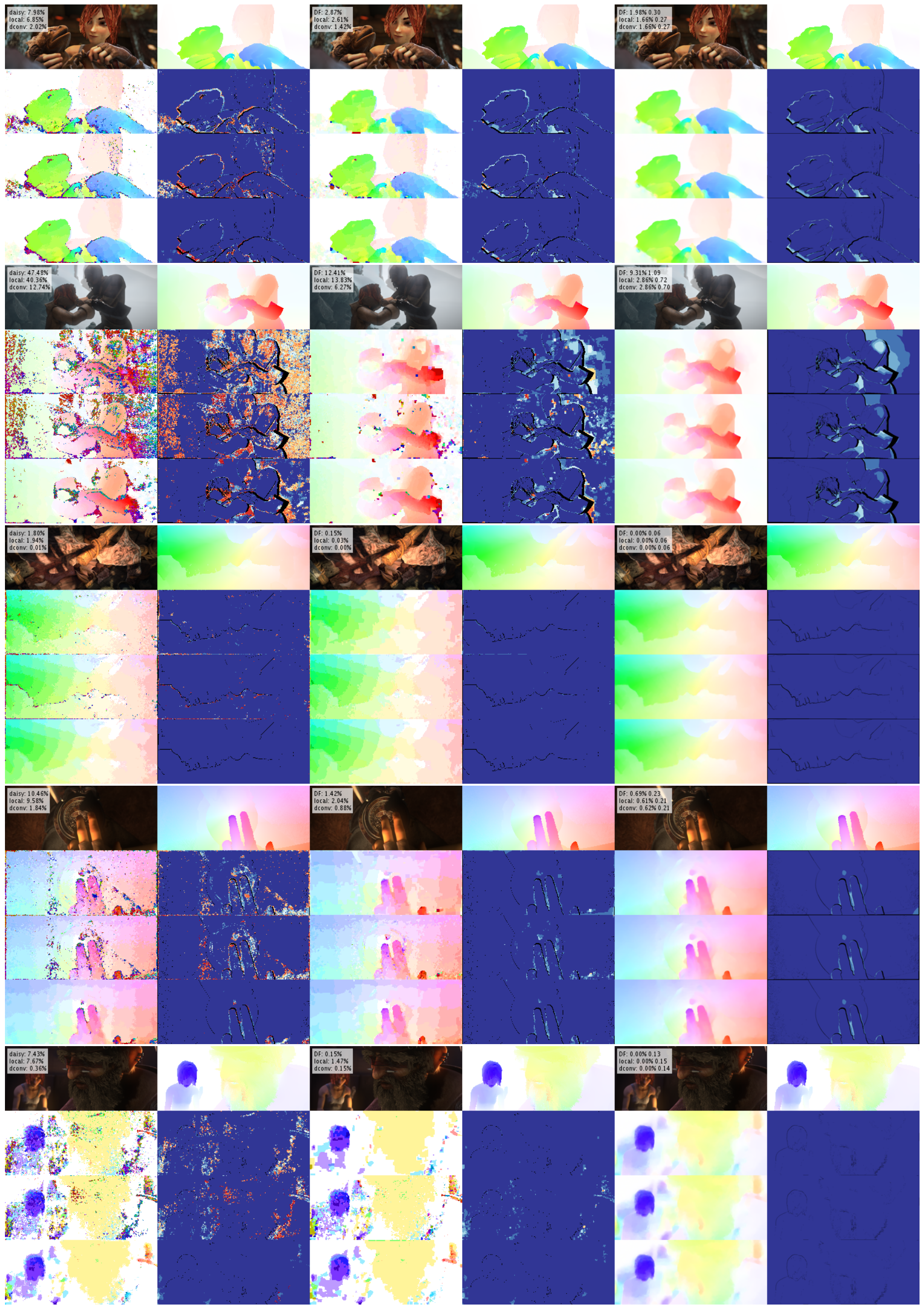


Fig. 4: **Qualitiative Results.** See text for details.

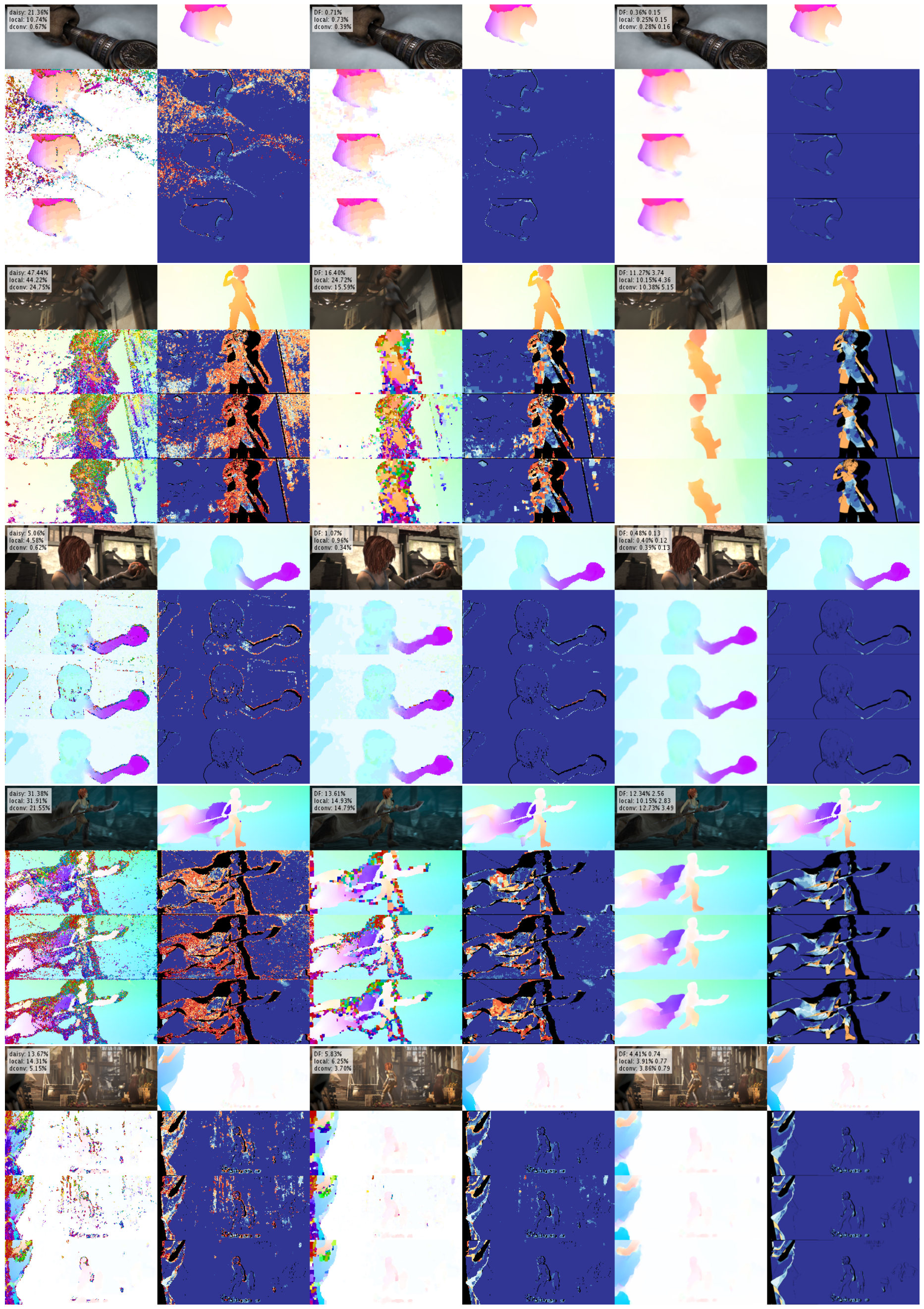


Fig. 5: **Qualitiative Results.** See text for details.

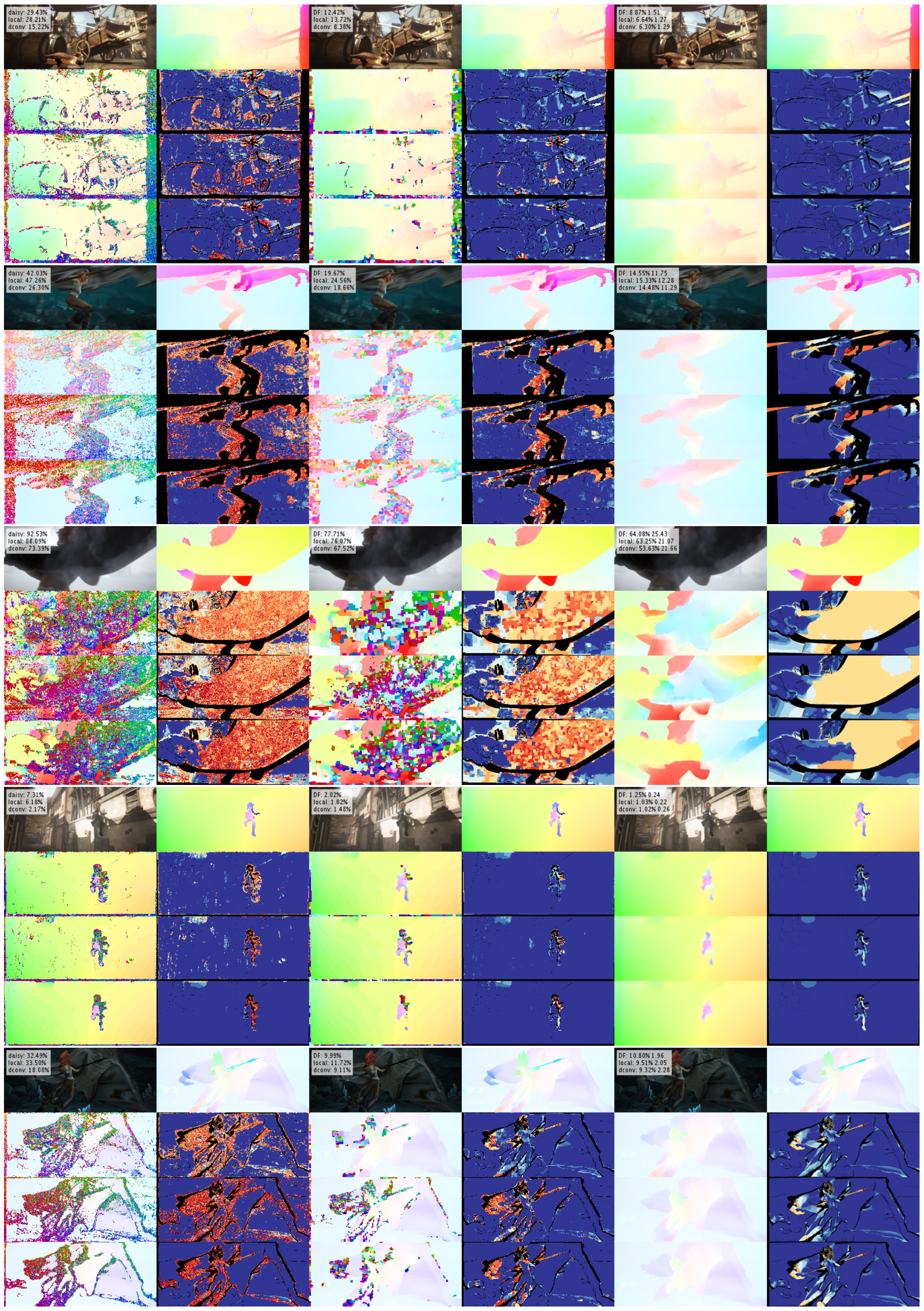


Fig. 6: **Qualitiative Results.** See text for details.

References

1. Bao, L., Yang, Q., Jin, H.: Fast edge-preserving PatchMatch for large displacement optical flow. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2014)
2. Besse, F., Rother, C., Fitzgibbon, A., Kautz, J.: PMBP: PatchMatch Belief Propagation for correspondence field estimation. International Journal of Computer Vision (IJCV) **110** (2014) 2–13
3. Menze, M., Heipke, C., Geiger, A.: Discrete optimization for optical flow. In: Proc. of the German Conference on Pattern Recognition (GCPR). (2015)
4. Muja, M., Lowe, D.G.: Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI) **36** (2014) 2227–2240