Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

RAFT meets DICL: A Recurrent All-Pair Transform for Optical Flow Estimation based on Displacement Invariant Cost Volume Learning

Maximilian Luz

Course of Study:	Informatik
Examiner:	Prof. Dr.-Ing. Andrés Bruhn
Supervisor:	M.Sc. Azin Jahedi
Commenced:	September 1, 2021
Completed:	May 2, 2022

Abstract

Motion is a fundamental aspect of visual perception, both in animals but also machines. In particular, the extraction of motion information from monocular image sequences as vector field of displacements, the so-called optical flow, is one of the core problems in computer vision. Its applications are widespread and, with the recent advancements in autonomous driving and other autonomous machines interacting with our world, ever-growing. Additionally, systems for action recognition, object tracking, navigation and localization, video processing, as well as other visual analysis tasks make use of such motion information, creating a considerable demand for accurate optical flow estimation methods, especially ones working well across various challenging scenarios.

To this end, we propose the combination of two recent approaches: The *recurrent all-pairs field transform for optical flow* (RAFT) by Teed and Deng (ECCV 2020), and *displacement-invariant matching cost learning* (DICL) by Wang et al. (NeurIPS 2020). RAFT introduces a novel recurrent neural network architecture, estimating and refining optical flow iteratively on a single spatial level by sampling from and decoding a hierarchical 4D cost volume. This has made it one of the current state-of-the-art methods in terms of accuracy. In RAFT, this cost volume, consisting of matching costs describing the pairwise similarity between feature embeddings of two subsequent frames, is computed via the standard dot product. DICL, on the other hand, employs a convolutional neural network as a learned cost function to provide more accurate cost values, but does not estimate flow recurrently, instead following a more classical coarse-to-fine warping scheme.

Within this thesis, we combine the best of both techniques, specifically, the dynamic cost learning approach proposed within DICL and the recurrent estimation and refinement approach proposed within RAFT. We do so by deriving a generic RAFT-based framework that allows for the use of arbitrary and potentially learned cost functions, such as the one employed by DICL, only requiring differentiability thereof. We furthermore provide an in-depth analysis of said combination, for both a RAFT-like hierarchical cost volume method and a more feasible coarse-to-fine method, and discuss several difficulties encountered with it. While we are yet unable to show significant improvements in accuracy compared to RAFT, we believe that our overall approach displays the potential for such and, moreover, may enable new cost learning strategies.

Zusammenfassung

Bewegung ist ein grundlegender Aspekt der visuellen Wahrnehmung, sowohl bei Tieren als auch bei Maschinen. Die Extraktion von Bewegungsinformationen aus monokularen Bildsequenzen als Vektorfeld von Verschiebungen, dem sogenannten optischen Fluss, ist eines der Kernprobleme des Maschinensehens. Dessen Anwendungen sind weitverbreitet und werden, unter anderem durch Fortschritte beim autonomen Fahren und anderen autonomen Maschinen, welche mit unserer Welt interagieren, stets mehr und wichtiger. Überdies nutzen Systeme zur Handlungserkennung (Action Recognition), Objektverfolgung, Navigation und Lokalisierung, Videoverarbeitung, sowie für diverse andere visuelle Analyseaufgaben solche Bewegungsinformationen, wodurch ein erheblicher Bedarf an genauen Methoden zur Schätzung des optischen Flusses entsteht, insbesondere an solchen, die in verschiedenen anspruchsvollen Szenarien gut und zuverlässig funktionieren.

Zu diesem Zweck schlagen wir die Kombination zweier moderner Ansätzen vor: der *recurrent all-pairs field transform for optical flow* (RAFT) von Teed und Deng (ECCV 2020) und dem *displacement-invariant matching cost learning* (DICL) von Wang et al. (NeurIPS 2020). RAFT führt eine neuartige rekurrente neuronale Architektur ein, die den optischen Fluss iterativ auf einer einzigen räumlichen Ebene durch Abtasten und Dekodieren eines hierarchischen 4D-Kostenvolumens ermittelt und verfeinert. Hierdurch wurde es zu einer der derzeit genauesten Methoden. Bei RAFT wird dieses Kostenvolumen, das aus Werten besteht, welche die paarweise Ähnlichkeit zwischen den merkmalsbeschreibenden Vektoren zweier aufeinander folgender Bilder einer Sequenz beschreiben, über das Skalarprodukt berechnet. DICL hingegen verwendet ein neuronales Netzwerk als erlernbare Kostenfunktion, um genauere Kostenwerte zu generieren, ermittelt aber den Fluss nicht rekurrent, sondern folgt einem eher klassischen grob-zu-fein Warping Schema.

In dieser Masterarbeit kombinieren wir das Beste aus beiden Techniken, im Speziellen den dynamischen Kostenlernansatz von DICL und den rekurrenten Ermittlungs- und Verfeinerungsansatz von RAFT. Wir tun dies, indem wir einen generischen RAFT-basierten Ansatz herleiten, welcher die Verwendung beliebiger und potenziell erlernbarer Kostenfunktionen, insbesondere jener von DICL, ermöglicht, wobei nur deren Differenzierbarkeit erforderlich ist. Des Weiteren liefern wir eine detaillierte Analyse dieser Kombination, sowohl für eine RAFT-ähnliche hierarchische Kostenvolumenmethode als auch für eine praktikablere grob-zu-fein Methode, und erörtern mehrere Schwierigkeiten, die dabei auftreten. Obwohl wir noch nicht in der Lage sind, signifikante Verbesserungen in der Genauigkeit im Vergleich zu RAFT aufzuzeigen, glauben wir, dass unser Gesamtansatz das Potenzial für diese zeigt und darüber hinaus neue Kostenlernstrategien ermöglichen kann.

Acknowledgements

I would, foremost, like to thank my supervisor Azin Jahedi for her continued support throughout the creation of this thesis. Our discussions kept me focused, and I am grateful for the valuable feedback and insightful advice she provided. It was a great pleasure working together.

I would also like to thank Prof. Dr. Andrés Bruhn for the opportunity of writing this thesis at the Computer Vision Group, and for his general support.

This thesis would not have been possible without the hardware resources provided through this group as well as by the BwUniCluster 2.0. Therefore, I also acknowledge support by the state of Baden-Württemberg through bwHPC. My gratitude further extends to Anton Malina for his excellent technical support, something all too often overlooked.

Finally, I want to express my profound gratitude to my family for their unconditional support and encouragement throughout all of my life and my studies, especially to my parents, Eva and Gerhard, and my sister, Katharina. I am blessed to have you. Further thanks extends to my friends, accompanying me on this journey.

I would like to let all of you know that your support is not something I take for granted.

Contents

List of Figures

List of Tables

Acronyms

AAE Average Angular Error.

AE Angular Error.

AEE Average Endpoint Error.

API Application Programming Interface.

BP Bad Pixel Error.

CNN Convolutional Neural Network.

DAP Displacement-aware Projection.

DCNN Deep Convolutional Neural Network.

DICL Displacement-invariant Matching Cost Learning.

DNN Deep Neural Network.

EE Endpoint Error.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

MNet Matching Cost Network.

RAFT Recurrent All-pairs Field Transform For Optical Flow.

RNN Recurrent Neural Network.

ViT Vision Transformer.

VRAM Video Random Access Memory.

Part I

Methodology

1 Introduction

Even before the nascence of present-day computing as we know it, humans have envisioned automata with manlike abilities. For example, Homer, in his Iliad, described "servants made of gold, fashioned like living girls", constructed by the god Hephaestus to aid him [Hom16, Book XVIII:368-467].[1] In another example from Greek mythology, Talos, the protector of Europa, is usually said to have been created by Hephaestus as a giant bronze intelligent automaton. We argue that, while there is no explicit mention of visual reasoning capabilities, visual perception must be a key component of such machines: Being able to perceive and understand the visual world is a major evolutionary perk, shared universally — although in varying forms — by all larger animals. It is therefore only natural that we would want to enable computers, as the fundamental instruments empowering our modern life, to, equally, reason about and understand visual input. The field of computer vision is the result of this desire, in which we aim to make sense of the visual world via images and videos by extracting computer interpretable and structured information from these largely unstructured sources. Considering the quite ageless imaginations of intelligent machines interacting with our world, it comes to no surprise that this field has a long and rich history.

One particular aspect of this domain is motion estimation and the incorporation of motion cues. The importance of motion for visual reasoning has been studied extensively in the field of ecological psychology and specifically by the psychologist Gibson, whose work on this topic (see, for example, Gibson [Gib50; Gib86]) had considerable influence on later developments. The studies conducted in this field have shown that motion plays a significant role in visual perception: Gibson [Gib50], for example, lays out the theory on how humans are able to recover a sense of distance and 3D information from monocular 2D image sequences, solely through the motion present in the scene. In fact, this principle has also been applied in computer vision via the so-called structure from motion techniques. However, direct reconstruction of 3D models is not the only application for motion therein. It can, for example, be used for object tracking, localization, analysis tasks, and more.

The focus of this thesis lies not in the applications of motion cues, but in estimating the motion itself from 2D image sequences as a vector field in the image plane. This field of motion is referred to as *optical flow*. In particular, we investigate a specific class of methods for this problem, combining two concepts that have both individually lead to significant performance improvements over previous approaches: Recurrent refinement, as proposed in the "recurrent all-pairs field transform for optical flow" (RAFT) by Teed and Deng [TD20], and dynamic cost volume construction, specifically as proposed by Wang et al. [WZD+20] in their "displacement-invariant matching cost learning" (DICL) method. The former is the basis for many of the current state-of-the-art techniques and has shown great results by itself, whereas the latter suggests performance benefits by moving away from fixed cost measures.

[1] See Kalligeropoulos and Vasileiadou [KV08] for more examples of automata in Homeric epics.

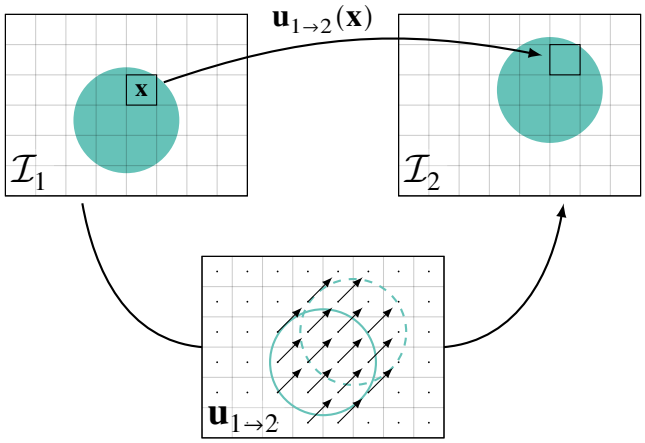

Figure 1.1: Illustration of optical flow in the two-frame case. The green circle moves from its position in the first image ($\mathcal{I}_1$, left) upwards and to the right to the position depicted in the second image ($\mathcal{I}_2$, right). The flow vector for the specific pixel $\mathbf{x}$ is depicted by the arrow $\mathbf{u}_{1\to2}(\mathbf{x})$. The full vector field of the forward flow $\mathbf{u}_{1\to2}$ is shown below, with arrows indicating the (instantaneous) flow at the respective source pixel.

In the remainder of this introductory chapter, we will build the foundation of this thesis: We begin by discussing the optical flow problem, its difficulties, and its applications in Section 1.1. Thereafter, Section 1.2 presents the motivation and core idea behind our approach in more detail. Finally, we will conclude this chapter with Section 1.3 by laying out the structure of the remaining document.

1.1 Optical Flow

Optical flow describes the concept of motion in the image plane, represented as a two-dimensional time variant vector field. Given an image sequence $\{\dots, \mathcal{I}_t, \mathcal{I}_{t+1}, \dots\}$, we denote the forward optical flow field as $\mathbf{u}_{t\to t+1} : \Omega \to \mathbb{R}^2$, where $\Omega = \{\mathbf{x}_{i,j}\}_{i,j}$ represents the set of pixel positions in the image plane of the camera (i.e., Ω represents the image domain).[2] The forward flow $\mathbf{u}_{t\to t+1}(\mathbf{x})$ then describes how the pixel $\mathbf{x}$ in the image $\mathcal{I}_t$ moves to its target position in the next image, $\mathcal{I}_{t+1}$. Similarly, the backward flow is given as $\mathbf{u}_{t+1\to t} : \Omega \to \mathbb{R}^2$ and describes movement from $\mathcal{I}_{t+1}$ back to $\mathcal{I}_t$. In this thesis, we are only interested in the dense forward flow for the two-frame case. We therefore define Ω to be the sampling grid of our camera and simplify the problem to estimating $\mathbf{u}_{1\to2}$ given a pair of images $\{\mathcal{I}_1, \mathcal{I}_2\}$. We then further reduce notation and define $\mathbf{u} \coloneqq \mathbf{u}_{1\to2}$. An illustration of this is given in Figure 1.1.

Estimation of optical flow belongs to the category of correspondence problems and is therefore related to stereo disparity estimation and image registration. As the name suggests, these problems revolve around finding correspondences in images. In image registration, correspondences often

[2] Note that we somewhat simplified this definition: Some estimation methods for optical flow may produce sparse results, for example due to occlusions, which we discuss in the next paragraph. For such methods, the set Ω on which we defined $\mathbf{u}$ may therefore be sparse or even time-variant (i.e., depend on the input sequence, the respective time t of estimation, and whether we estimate forward or backward flow).

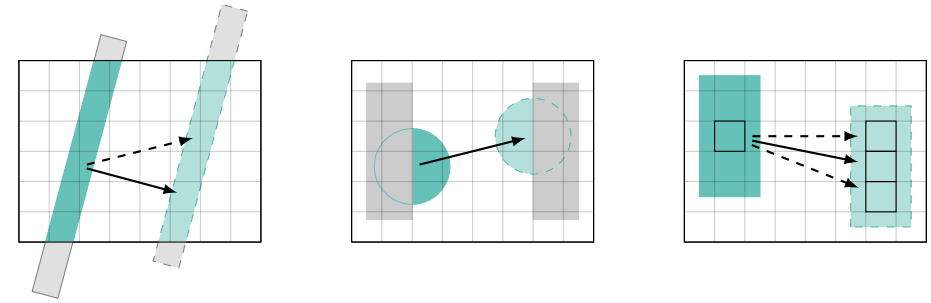

Figure 1.2: Difficulties in optical flow estimation. Moving objects are shown in green, with darker initial positions (first frame) and lighter/dashed target positions (second frame). Motion is indicated by arrows. *Aperture problem:* Only the normal component of the flow relative to an edge can be estimated (solid arrow). Estimation of the real flow (dashed arrow) is not possible due to missing information (gray, out of frame). *Occlusions:* Parts of objects may be occluded (here by gray rectangles) in either the first or the second frame, or both. *Ambiguities:* Textureless or repetitive areas can lead to ambiguous flow candidates (dashed arrows). Real motion (solid arrow) cannot be estimated if looking at local spots (black squares) only.

follow a predefined model, such as an affine or a grid transformation, whereas in stereo estimation, correspondences can be restricted to a line by the epipolar constraint. In optical flow estimation, however, the search space is unconstrained by the problem itself and, as a result, the number of potential matches is quadratic in input size. Due to this, most methods restrict the search space by considering only matches in a predefined neighborhood around the source pixel, thus essentially limiting the maximum flow magnitude that the method can estimate correctly.

Further difficulties, illustrated in Figure 1.2, arise from the *aperture problem*, denoting the restriction that only the component of the flow normal to an edge can be estimated, *occlusions*, which can lead to pixels of one frame not being visible (i.e., occluded) in the other frame, or *ambiguities*, which allow for multiple flow possibilities (as, for example, in large textureless areas). Additionally, changes in lighting and other photometric properties can make reliable estimation of optical flow more complicated. Due to these issues, approaches at solving the optical flow problem generally require some sort of regularization and can benefit from additional information, such as using more than two images of the sequence or occlusion maps.

The concept of optical flow itself is closely related to the concept of *scene flow*, which describes the three-dimensional motion of points in the world (i.e., scene). Hence, optical flow is simply the projected scene flow, making it a fundamental building block for motion in computer vision. In contrast to scene flow, optical flow is bound to a single camera, whereas estimation of scene flow requires multiple cameras, as the projection is irreversible given only a single one. Further, this relation means that scene flow can be computed from optical flow, stereo disparities, and the camera configuration. More importantly, however, it makes optical flow a key concept and integral part for many downstream tasks, including autonomous vehicles [GCT98; GLU12; MG15] and robot navigation [CGN13; YBH15], for example via localization and mapping. Further applications are object tracking [AJN+16; LXM+21], segmentation [TYB16], action recognition [SLG+19; SZ14; WS13; YLZ19], and visual surveillance [KMK15], but also biometrics [SW18], including face

recognition [HDP11] and gait recognition [MMBH18; SJAS21]. Video processing can similarly benefit from optical flow, for example via temporal consistency [LHW+18]. Video classification [BC08], indexing and retrieval [HXL+11], content analysis [Lou18], or tampering detection [SM16] as part of forensic analyses are other examples. Optical flow also finds its use in biomedical imaging [Lai10; TBS12], for example for tracking and determining cell movements [Miu05; VHW16].

1.2 Motivation and Idea

Concurrent neural network based optical flow estimation methods, in large, rely on the same core concepts: First, a feature encoder transforms the input images into a suitable feature representation. These encodings are then used to compute the matching costs of potential correspondences, creating the so-called cost volume. The cost volume can optionally be refined before it is used to, finally, estimate optical flow via a decoder. As an (again optional) last step, the produced flow can be post-processed, usually by incorporating contextual information obtained from a secondary feature encoder. Differences in methods stem mostly from the implementation and layout of these components. For example, creating, refining, and decoding the full all-pairs cost volume is generally infeasible. Due to this, strategies have to be used which increase the search space of correspondences without increasing memory or compute costs (e.g., multiscale processing or dilated convolutions). An overview of such techniques is presented in combination with related work in Section 2.1.

A method of particular interest to us has been proposed by Teed and Deng [TD20] with their "recurrent all-pairs field transform for optical flow" (RAFT). They introduced a novel recurrent flow estimation and refinement scheme, which updates flow in each step based on the previous estimate. Cost values are sampled from an all-pairs cost volume via a lookup operator. This operator samples the cost volume at a predefined set of displacement hypotheses around the currently estimated displacement location (implied by estimated flow and pixel position), therefore avoiding the quadratic dependency of memory and compute resources on the input size that would otherwise be associated with processing the full all-pairs cost volume. Further, the cost values are computed via normalized dot products between feature encodings, which can be implemented efficiently as a tensor product between the two feature tensors (each containing the feature encodings for all locations of the respective input image). Via this combination, they were able to set a new state-of-the-art. At the time of writing, techniques based on RAFT dominate the common optical flow benchmarks.[3] Section 3.1 will discuss the RAFT architecture in more detail.

We believe that one limiting factor of RAFT is the cost volume: Several works have shown that enhancing the cost volume, either by refinement (see, e.g., Hui and Loy [HL20] and Yang and Ramanan [YR19]) or by replacing the fixed correlation/cost measures (such as the dot product) with learned ones (see, e.g., Wang et al. [WZD+20] and Xiao et al. [XYS+20]), can lead to better results compared to their respective baselines. To our knowledge, however, there is only one technique for optical flow estimation that computes matching costs in a fully learned way using a convolutional neural network (CNN) as cost function: "displacement-invariant matching cost learning" (DICL) by Wang et al. [WZD+20]. In their paper, Wang et al. integrate their cost learning module into a coarse-to-fine approach which predates RAFT and, as far as we can tell, no attempts at integrating the DICL module into RAFT have been published yet. This may in part be due to the higher memory

[3] We consider only non-anonymous entries.

requirements of the CNN (compared to the dot product), making computation of an all-pairs cost volume infeasible and such an integration not straightforward. We are of the opinion that this poses a gap in research, especially as DICL shows clear benefits over other coarse-to-fine approaches, and aim to help explore said gap in this thesis. As evidenced by the recent Separable Flow [ZWPT21] approach, using a cost aggregation scheme to enhance the cost volume, it stands to reason that such improvements to the cost measure are also applicable to RAFT.

1.3 Outline

This thesis is divided into two parts. In the remainder of Part I, we will discuss the methodological foundations of our approach. Chapter 2 presents the general foundations: We will look at related work in Section 2.1, covering both classical and foundational optical flow estimation approaches, as well as modern deep learning ones. We discuss training and evaluation datasets, benchmarks, as well as further training considerations (such as data scheduling and augmentations) in Section 2.2, present common error measures in Section 2.3, and, lastly, explain visualizations used throughout this thesis in Section 2.4.

Chapter 3 continues on the foundations laid in Chapter 2 and presents the specific methods upon which we build our approaches, namely, RAFT in Section 3.1 and DICL in Section 3.2. In Section 3.3, we then, finally, derive the core concept of our approaches, detailing the integration of the displacement-invariant matching cost learning (DICL) approach, proposed by Wang et al. [WZD+20], into the larger RAFT optical flow estimation framework, proposed by Teed and Deng [TD20]. Section 3.4 concludes this first part by giving insights into two design choices, sampling (as an alternative to warping) and gradient blocking, integral to not just our methods, but also both RAFT and, for the latter, DICL.

Part II focuses on the experimental aspects of this thesis, starting with Chapter 4 presenting the specific methods evaluated and the training strategy employed. We introduce a single-level RAFT+DICL method in Section 4.1, a multi-level method based on the multiscale approach of RAFT in Section 4.2, and a coarse-to-fine multiscale method in Section 4.3 aiming to address shortcomings encountered within the earlier two approaches. Section 4.4 introduces a new loss term as a generalization of the sequence loss used by RAFT, which we will use in later evaluations. Our training strategy will be presented in Section 4.5, followed by our evaluation strategy in Section 4.6. Finally, Section 4.7 details the rationale and plan behind our evaluation approach.

Chapter 5 discusses the first base evaluations and their results. We begin by analyzing our RAFT and DICL baselines in Section 5.1, performing various ablation studies to gain a better understanding thereof. Section 5.2 then investigates our single-level method to gauge general feasibility of our larger RAFT+DICL approach, followed by our multi-level method in Section 5.3 and our coarse-to-fine method in Section 5.4. Thereafter, Section 5.5 concludes this chapter with a short summary of the results and insights gained so far.

Chapter 6 presents the major difficulties we encountered while training our methods, ways in which they can be resolved, as well as additional improvements to our approaches. In particular, Section 6.1 discusses the stability issues we observed (on both a global and a per-level basis), means with which they can be avoided, and a theory as to why they occur (almost) exclusively in our methods. Following up on this, Section 6.2 explores the cost volumes created by our approaches,

ultimately introducing and evaluating an attempt at forming and, through this, improving the learned cost function. Lastly, Section 6.3 investigates variations of the specific neural network used as the core ingredient of the cost learning approach.

We conclude this thesis in Chapter 7, summarizing our findings in Section 7.1. Finally, Section 7.2 will provide a look at further ideas for improvements of the approaches discussed and potential future work building on the foundation laid here.

2 Preliminaries

There are three main components to every data-driven machine learning method, such as neural networks: The model and overall architecture, the training data and strategy, and, lastly, the evaluation methodology with which we gauge its performance. It is well known that all three are critical for success. If the validation data and the error measures used cannot correctly assess the model's true capabilities in the desired target domain or the data does not sufficiently challenge our technique (e.g., as it contains only easy examples), we cannot make informed decisions on how the model may be improved. If the training data is of bad quality, we cannot expect good results. The model may compensate for outliers during training, however, if bad samples become the norm we will train the models to fit those instead of our intended target. Similarly, if the training strategy is bad, we may never achieve the full potential of our model and, in the worst case, may even attribute this to architectural issues. Lastly, if the architecture is not suited for the task, it may not be able to capture the dynamics of the problem at hand or, at the other end of the spectrum, be susceptible to overfitting.

In the context of optical flow estimation, a considerable amount of work has been put into all three of these aspects. Models have been carefully designed to incorporate inductive biases and, in general, the capability to accurately detect motion, commonly through feature correlation. Training and evaluation data has been obtained and improved to satisfy the need for the large-scale training that neural networks require and to qualitatively and comparatively validate the produced results on open benchmarks. This chapter will lay the groundwork for the rest of this thesis by discussing these aspects: We will start in Section 2.1 by discussing the works related to our approaches, both classical and modern. Section 2.2 will present available training and evaluation datasets, including benchmarks, before we describe common error measures, in use by said benchmarks, in Section 2.3. Lastly, Section 2.4 will present visualization schemes for optical flow itself, the per-pixel endpoint error, and cost volumes.

2.1 Related Work

Determining image correspondences can be described as one of the fundamental problems of computer vision. In particular, optical flow estimation has a long-standing history, beginning in the 1980s and continuing up to today. As such, there have been many influential works, forming the current state-of-the-art methods. In the following subsections, we attempt to give a detailed overview of the techniques related to the cost-learning approaches that we investigate in this thesis. Our specific focus lies on the two-frame case, due to which we do not consider multi-frame methods. We begin in Section 2.1.1 with the foundations of optical flow and classical non-learning techniques, as we find that many of the core principles devised during this time have been adapted into concurrent deep learning approaches, often with great success. Section 2.1.2 will then focus on learning-based methods, starting with early approaches involving patch matching and descriptor learning, before

continuing to fully end-to-end differentiable models. There, we will focus again on the core concepts, but also on cost learning and iterative refinement, in this order. Finally, in Section 2.1.3, we will attempt to relate cost learning to (deep) metric learning and give an overview of the latter.

2.1.1 Foundations of Optical Flow

The true foundations of optical flow can be found not in computer vision, but in psychology and ecology: In these fields, *visual perception*, i.e., the studies of how humans and other animals perceive, view, and interpret their surroundings through their visual system, has attracted a considerable amount of research (see, e.g., Bruce et al. [BGG03] and Gibson [Gib50; Gib86]). While the connection between visual perception and motion had already been noted previously, e.g., by Helmholtz [Hel25] discussing reconstruction of 3D information from monocular vision by help of self-motion, it was Gibson and his pioneering work who formed the concept (and coined the term) of *optical flow* around the 1950s [Gib50].

Estimation of optical flow from image sequences, on the other hand, finds its roots in the early 1980s with the seminal works of both Horn and Schunck [HS81] and Lucas and Kanade [LK81]. Lucas and Kanade approach this problem from a perspective of *image registration*, assuming local constancy of optical flow in a given neighborhood and using a least squares fit. They further employ a linear approximation to the brightness constancy constraint $\mathcal{I}_1(\mathbf{x}) = \mathcal{I}_2(\mathbf{x} + \mathbf{u}(\mathbf{x}))$, where $\mathcal{I}_1$ and $\mathcal{I}_2$ denote the brightness of the first and second image (respectively), $\mathbf{x}$ denotes a pixel position, and $\mathbf{u}$ the optical flow displacement vector field [LK81]. While image registration had already been explored previously in terms of discrete block-matching techniques, their continuous model explicitly allows for sub-pixel precision, leading to significantly better results when used for optical flow estimation.

Horn and Schunck [HS81] equally base their method on the linearized brightness constancy constraint, however, frame their approach in a *variational* setting and therein lay the groundwork that inspired, in large, the following generations of optical flow estimation techniques: They propose to (globally) minimize an energy functional $E(\mathbf{u})$, to which the vector field $\mathbf{u}$ is given as an input, via the calculus of variations. The functional $E(\mathbf{u})$ consists of both a data term, derived from the linearized brightness constancy constraint, and a smoothness term, providing regularization. The data term penalizes deviation from the chosen optical flow constraint, ensuring adherence to it, whereas the smoothness term ensures well-posedness of the problem (in the sense of Hadamard [Had02]), by penalizing deviation from a smooth flow field.[1] The latter is required due to the *aperture problem*, which describes the fact that, as a result of ambiguities, only the flow component orthogonal to an edge can be inferred correctly.

Due to the linearization of the brightness constancy constraint, both the works of Horn and Schunck [HS81] and Lucas and Kanade [LK81] are inherently limited to small displacements. The latter, however, attempts to ameliorate this by use of a multiscale optimization approach. This is significantly improved upon by follow-up methods, which, apart from improvements to the data

[1] In the method of Lucas and Kanade [LK81] this kind of regularization is provided by assumption of constancy in the neighborhood.

and smoothness terms as well as other algorithmic advancements, introduce techniques such as coarse-to-fine processing (see, e.g., Anandan [Ana89], Black and Anandan [BA96], and Memin and Perez [MP98]).

The combination of *coarse-to-fine* processing and *warping*, in particular, has been proven critical for the estimation of large displacements [BBPW04; SRB10]: Optical flow is computed iteratively across multiple levels, each having a different spatial resolution. As input, each level receives the original image pair, downsampled and reduced in size respectively for that level, adhering to the sampling theorem. Computation of flow begins at the coarsest level, yielding an initial estimate, which is then upsampled to a finer level. On this finer level, the second image is, in general, warped towards the first image using said estimate. Expressed mathematically, the (backward-)warped image $\tilde{\mathcal{I}}_2$ is computed via $\tilde{\mathcal{I}}_2(\mathbf{x}) \coloneqq \mathcal{I}_2(\mathbf{x} + \mathbf{u}(\mathbf{x}))$, i.e., by looking up the pixel in the original image at the position pointed to by the current flow estimate.[2] That, in turn, leads to an estimation of the residual flow, i.e., the remaining flow between the current estimate and the target. The new estimate is then given as the sum of residual and current estimates, and the process is continued until the finest level has been reached.

This approach allows estimation of large motions on coarser levels and small motions on finer levels. It, however, has limited success when it comes to large motions of small objects, as they are generally not represented well (or outright missing) in the downsampled images. A further benefit of this coarse-to-fine warping scheme is the capability of estimating large motions without significantly increasing computational requirements when working with algorithms utilizing local neighborhoods. Increasing neighborhood size (assuming this can be done without violating any assumptions, e.g., made when linearizing the brightness constancy constraint) scales quadratically, whereas the coarse-to-fine warping approach scales logarithmically.

Surveys and comparisons of these classical methods are, for example, given by Baker et al. [BSL+11], Fortun et al. [FBK15], and Sun et al. [SRB14]. Key concepts developed therein and during this time can still be found in many of the concurrent deep learning approaches, such as coarse-to-fine warping and multiscale/multi-level processing in general, as well as more structural considerations, such as separation into estimation and refinement subnetworks, which can (somewhat loosely) be compared to the data and regularization/smoothness terms of variational approaches.

2.1.2 Learning-Based Approaches

Similar to optical flow estimation, discussed in the previous subsection, modern (deep-)learning-based computer vision techniques find their foundations around 1980, specifically with the Neocognitron model proposed by Fukushima [Fuk80]. The key contribution of this work is the introduction of a then novel concept: The convolutional neural network (CNN). Already back then, these networks were stacked to obtain a deep convolutional neural network (DCNN). It, however, took CNNs and deep neural networks (DNNs) until the conception and considerable success of DanNet [CMS12] and, shortly later, AlexNet [KSH12] around 2012 to take hold and attract wider attention. In large, this can be accredited to the increase in computational power and accessibility thereof since then,

[2] Note that, as images are discrete and the flow vectors $\mathbf{u}(\mathbf{x})$ continuous, some form of interpolation (typically bilinear interpolation) is performed to evaluate $\mathcal{I}_2(\mathbf{x} + \mathbf{u}(\mathbf{x}))$.

as well as the growing availability of training data. The (re-)discovery and popularization of the backpropagation algorithm through Rumelhart et al. [RHW86], now used for training such networks, can be seen as an additional cornerstone.

Nowadays, CNNs pose as fundamental building blocks for the vast majority of contemporary computer vision architectures, with the notable (and rather recent) exception being vision transformers (ViTs) [DBK+21]. We refer the reader to the book by Goodfellow et al. [GBC16] for an introduction to deep learning, the article by Schmidhuber [Sch15] for a comprehensive history thereof, the survey by Srinivas et al. [SSM+16] for an overview of CNNs in computer vision, and lastly the surveys by Khan et al. [KNH+21] and Xu et al. [XWL+22] for ViTs.

In the following two subsections, we will provide a detailed overview of the learning-based methods for optical flow estimation related to our work. We will begin with early approaches, focusing on feature encoding and patch matching. Thereafter, we will discuss contemporary and end-to-end trained methods, starting first with the general network structure by briefly going over the evolution thereof. Following this, we will transition to cost volumes and their refinement, leading up to cost volume learning, the DICL method, and its relatives. Finally, we will discuss iterative and recurrent approaches, focusing on RAFT and the concepts that enable its success. For more details on optical flow estimation using deep learning, we refer the reader to the surveys by Hur and Roth [HR20], Savian et al. [SET20], and Sun et al. [SYLK20].

Early Approaches

In contrast to the classical techniques discussed in the previous section, which only contain a comparatively small number of parameters that often have to be fine-tuned manually or via automated parameter sweeps, deep learning models are significantly larger in parameter count and learn their parameters, including convolutional filter banks, from data. Early approaches of integrating deep learning into optical flow estimation methods focussed largely on descriptor and image patch matching. For refining these (often sparse and filtered) matches into dense flow fields, they used techniques based on variational methods, such as DeepFlow [WRHS13], EpicFlow [RWHS15], or later also the learned approach by Zweig and Wolf [ZW17].

Notable works in this category are PatchBatch by Gadot and Wolf [GW16], the approach by Bailer et al. [BVS17], and DC-Flow by Xu et al. [XRK17]. All three use techniques similar to deep metric learning (see Section 2.1.3): In particular, they employ an encoder structure with tied weights, i.e., a *Siamese network* [BGL+94], to map image patches into a (lower-dimensional) feature space. In this space, similarity is then computed via a fixed measure (such as the dot product or euclidean distance). Lastly, a hinge or triplet loss is used for optimization. The earlier stereo matching approaches by Žbontar and LeCun [ŽL15; ŽL16] stand in contrast to this: They forego any fixed measure[3] and, instead, directly compute matching scores via fully connected layers, taking the concatenated feature vectors of the respective two patches to be compared as input.

[3] At least in their "accurate" architecture. The "fast" architecture of [ŽL16] also uses a cosine similarity measure.

Contemporary Approaches

Inspired by the success of DCNN-based architectures in other computer vision tasks, Dosovitskiy et al. [DFI+15] proposed FlowNet, the first fully differentiable and therefore end-to-end trainable approach for optical flow estimation using neural networks. By introduction of the synthetically generated FlyingChairs dataset, they were able to show that — given a sufficient amount of training data — such architectures could pose as viable alternatives to the then common multistep techniques, consisting of descriptor generation, matching, interpolation, and refinement. While FlowNet was not yet able to beat state-of-the-art algorithms (such as EpicFlow [RWHS15]) in qualitative performance, applicability to GPUs and significantly reduced runtime costs made it an interesting approach for real-time use.

The architecture proposed by Dosovitskiy et al. [DFI+15] can be split into two parts: An encoder, which downsamples and processes the input into a hidden representation of motion, and a decoder, which upsamples and refines that representation to produce optical flow. Specifically, they suggested two encoder variants: One, FlowNetS, stacking input images and the other, FlowNetC, computing dense patch correlations between image features using the dot product for a limited set of displacement hypotheses per pixel, essentially generating and processing a *correlation volume*.[4] In the latter, features are obtained from a Siamese network. This model was improved upon by Ilg et al. [IMS+17] with FlowNet 2.0, stacking multiple FlowNet instances, and later Sun et al. [SYLK20] with a revised training strategy that allowed FlowNetC to out-perform the significantly larger FlowNet 2.0. Further modifications to this architecture were proposed by Harley et al. [HDK17] for incorporating segmentation information, Ilg et al. [IÇG+18] for uncertainty estimation, and Ilg et al. [ISKB18] for jointly estimating flow and occlusions.

Follow-up work focussed on both reducing parameter count (from 39 million for FlowNetC and 162 million for FlowNet 2.0) and improving qualitative performance. Ranjan and Black [RB17] proposed SPyNet, borrowing from classical approaches by using a spatial image pyramid and estimating residual flow updates in a coarse-to-fine manner via image warping. Going back to the coarse-to-fine warping strategy proved successful, however, was quickly improved upon by both Sun et al. [SYLK18; SYLK20] with PWC-Net and Hui et al. [HTL18; HTL21] with LiteFlowNet, advocating for the use of feature pyramids and warping of features instead of image pyramids and warping of images. Both works use a Siamese encoder network to generate feature vectors and, further, a cost volume over a restricted set of displacement hypotheses to encode motion. On each pyramid level, starting from the coarsest, the cost volume is computed via a scaled dot product from the feature vectors of the first and (warped) second frame, and then refined and decoded into flow via a DCNN. In addition to this, Sun et al. [SYLK18] as well as Hui et al. [HTL18] propose refining the produced flow via a context or regularization network before upscaling it to the next finer level.

Both PWC-Net and LiteFlowNet have shown that careful integration of domain knowledge and inductive biases can improve performance and reduce model size at the same time. In a way, these networks can be seen as reintroducing the well-established principles of non-learning and classical optical flow estimation methods into a modern deep learning framework. Succeeding work has

[4] Note that *cost* and *correlation* are inversely related. Therefore, the terms *cost volume* and *correlation volume* can be seen as referring to the same broader concept (sometimes also referred to as *correspondence volume*), and we may use them interchangeably in this context. We will, however, explicitly use the respective term when needed.

largely centered around the same principles, i.e., the use of coarse-to-fine warping, encoder networks with shared weights, a cost volume, refinement and decoding thereof, as well as regularization/post-processing of the generated flow.

We are particularly interested in works focusing on cost volumes: Hui and Loy [HL20] improve over LiteFlowNet [HTL18] and LiteFlowNet2 [HTL21] with LiteFlowNet3 by applying a confidence-/uncertainty-based refinement network to the cost volume and enhancing flow refinement with a similar confidence-based approach. Neoral et al. [NŠM19] improve over PWC-Net with ContinualFlow by, among other things, estimating an occlusion map directly from the cost volume and then using this in combination with the cost volume to produce the residual flow update. Their rationale is that costs for an occluded pixel are equally high over all displacements. For training, however, they require ground-truth occlusion data.

Other notable methods are HD^3 by Yin et al. [YDY19], attempting to learn match probability densities instead of costs or correlation-based scores by using the Kullback-Leibler divergence as loss, and VCN by Yang and Ramanan [YR19], proposing refinement of a multichannel cost volume via separable 4D convolutions. VCN achieves this by using multiple feature representations, computing the cost volume entries via a cosine similarity measure, one channel per representation. Cost volume channels are refined independently, transformed into flow hypotheses via a (truncated) soft-argmin, again one per channel, and then fused into a single estimate via a hypothesis selection network. Recently, a similar multi-hypotheses approach under the name of DCVNet has been proposed by Jiang and Learned-Miller [JL21], focusing on real-time use, i.e., speed and lower memory consumption. Their approach employs multiple cost volumes with different dilation and stride combinations, each combination again using multiple hypotheses (as proposed by VCN). Instead of a sequential coarse-to-fine warping scheme, they generate the cost volumes in parallel, relying on dilations to extend the field of displacement candidates.

Cost Learning. Less work has, however, been spent on exploring learned alternatives to the fixed correlation measures used when computing the (initial) cost volume and, on a broader scale, attempts at learning to generate it. We base the methods presented in this thesis on the "displacement-invariant matching cost learning" (DICL) approach by Wang et al. [WZD+20]. They propose stacking the feature vectors of the first and second frame for each displacement hypothesis, where the latter have been offset by the respective displacement. These stacks are then processed by a 2D CNN, handing each displacement independently and reducing the stacked feature tensors to a (preliminary) cost volume. As a final step, a displacement-aware projection (DAP) layer is applied to reweigh the displacement costs individually for each pixel, allowing for inclusion of directional priors and to achieve a more uni-modal distribution, before producing flow via a soft-argmin regression. We refer to Section 3.2 for more details.

In essence, DICL learns a function to generate the cost volume, which can, due to the 2D convolutions, incorporate spatial information from neighboring pixels. It is, however, limited in two ways: First, in that it does not include information from other displacements until the DAP layer and, more importantly, second, in the number of displacement hypotheses it is able to consider due to the memory required through stacking and processing the feature vectors. At the time of writing, the closest contender to DICL, and by extension our methods, is the Separable Flow approach by Zhang et al. [ZWPT21]. Instead of processing the full cost volume, they propose a separation in u and v components onto which they apply a semi-global guided aggregation mechanism, previously

introduced for stereo disparity estimation in GANet [ZPYT19]. This allows the authors to avoid the aforementioned issues of DICL and enables processing of a full all-pairs cost volume by approximation, but again limits them to a simple fixed inner product for generating the initial cost volume before decomposition. Separable Flow [ZWPT21] can be compared with LiteFlowNet3 [HL20] in that both use data-driven operations to modify the cost volume. The latter, however, only uses a single modulation operation that can be interpreted as replacing low-confidence cost values with better approximations, whereas the former iteratively builds-up and actively changes the cost volume over multiple aggregation steps from not only a local neighborhood, but also (semi-)global information.

The stereo estimation methods by Žbontar and LeCun [ŽL15; ŽL16], discussed earlier in this section, may also be seen as closely related as they, in contrast to other patch matching techniques, do not use any fixed correlation measure. However, they do not incorporate any spatial information of neighboring patches after the feature encoding stage. They instead process each patch independently and only output one cost or correlation value per input pair. Additional complications of applying such approaches to optical flow estimation are discussed in Section 2.1.3, when we relate cost learning to (deep) metric learning.

Further methods improving upon the fixed correlation measures have been suggested by Truong et al. [TDGT20] and Xiao et al. [XYS+20]: Xiao et al. [XYS+20] propose learning a positive definite metric tensor M via the Cayley representation, generalizing the inner product $\mathbf{x}^\top \mathbf{y}$ to the elliptic inner product $\mathbf{x}^\top M \mathbf{y}$. Truong et al. [TDGT20], on the other hand, propose an optimization-based meta-learning approach to, asymmetrically and dynamically, enhance the first frame features by actively optimizing their representation in each forward pass. They therefore replace the cost computation step $c(\mathbf{f}_1, \mathbf{f}_2)$, where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the feature tensors for the first and second frame (respectively) and c is a fixed measure (e.g., the standard inner product), by $c(\mathbf{w}, \mathbf{f}_2)$, where $\mathbf{w}$ is generated on-the-fly for each sample based on a loss function $\mathcal{L}(\mathbf{w}; \mathbf{f}_1, \mathbf{f}_2)$.

Notably different approaches have been proposed by Jaegle et al. [JGB+21] and Teney and Hebert [TH17]. Teney and Hebert [TH17] rely heavily on classical signal processing principles to reduce the number of model parameters and facilitate training without large amounts of data, which was not yet available at the time of conception. They carefully designed their architecture to obtain rotational invariance by tying weights, essentially creating one network path per displacement hypothesis of the produced cost volume. By stacking input frames, they forego the use of any fixed correlation measure. Jaegle et al. [JBA+22], in contrast, extend the Perceiver [JGB+21] model, based largely on cross-attentional vision transformers (ViTs) [DBK+21], for dense prediction tasks, including optical flow. Their model does not (explicitly) rely on any of the established concepts commonly used for optical flow estimation, such as cost volumes or multiscale processing, however, still manages to perform on-par with current state-of-the-art methods.

Iterative Refinement and Sampling. Next to advancements regarding the cost volume, another focus of recent methods has been on iterative estimation and refinement of the optical flow. The current state-of-the-art in optical flow estimation is given by Teed and Deng [TD20] with their "recurrent all-pairs field transforms" (RAFT) method, as well as derivatives thereof, such as Separable Flow [ZWPT21] and GMA [JCL+21]. The success of RAFT can be ascribed to the clever combination of (in part) independently verified principles that, together, form a powerful optical flow estimation method. In particular, it employs a Siamese feature encoder similar to other methods,

but only uses a single-level feature representation and only produces flow output on a single level. More importantly, however, RAFT relies on an all-pairs cost volume, which is pooled to allow for a multiscale cost representation. Optical flow is generated iteratively by a recurrent neural network (RNN) using a sampling approach: First, the current flow estimate is used to sample the cost volume via a lookup operator. This lookup operator uses the location pointed to by the current flow estimate and samples the cost volume at said position, as well as offsets around it. Essentially, this can be understood as creating a cost volume slice with costs for different displacement hypotheses, similar to other approaches, however, on multiple scales at once (as opposed to the single-level representation commonly found in coarse-to-fine methods). The cost volume slice is then, together with the current flow estimate and a context feature encoding of the first frame, fed to a recurrent refinement network, which produces a residual flow update to be added to the estimate. RAFT will serve as the framework for our cost learning approach. A full description of the method can be found in Section 3.1.

Prior to RAFT, Hur and Roth [HR19] have shown that iterative refinement combined with weight sharing in these refinement steps can both improve performance and reduce the number of parameters at the same time. Similarly, Hofinger et al. [HBP+20] have shown that replacing warping with sampling can improve performance in coarse-to-fine approaches by, first, avoiding the ghosting effect introduced through warping and, second, being able to better preserve smaller objects. Ghosting happens, for example, when a faster object moves across a slower background, which then causes warping to produce a "ghost" of that object at the previously occluded spot. Other methods, such as MaskFlowNet [ZSD+20] and LiteFlowNet3 [HL20], have attempted to repair the damage caused by this effect in a post-processing step applied to the warped features. Sampling, on the other hand, bypasses this problem as it does not modify the original feature map. It can be seen as an extension of spatial transformer networks [JSZK15] to optical flow. We will discuss the respective benefits and drawbacks of sampling and warping in Section 3.4.1 in more detail.

Based on the same principles of iterative refinement, sampling, and estimation on a single level, Lu et al. [LVW+20] proposed Devon. In contrast to RAFT, however, their approach relies on dilated sampling stencils to incorporate motion across multiple scales, as opposed to pooling the cost volume. Further, they neither make use of a recurrent network, nor weight sharing for iterative estimation. Instead, they use multiple instances of the same architecture with different weights in succession for refinement. Devon is outperformed by both the aforementioned DCVNet [JL21], also making use of dilated cost volumes and single-level flow estimation, and RAFT.

2.1.3 Cost Learning in Relation to Metric Learning

In general, the similarity measures used for computing matching costs or correlations in optical flow estimation are not metrics.[5] For example, the commonly used dot product does neither fulfill positivity nor the triangle inequality. The cosine distance function fulfills positivity, but does neither fulfill the triangle inequality nor the identity of indiscernibles. However, we argue that a good cost function and feature representation should still convey some semblance of distance. In particular, when we have given some cost function $c : X \times X \to \mathbb{R}$ and some elements $x, y, z \in X$,

[5] A metric d is defined as a function $d : X \times X \to \mathbb{R}_0^+$ on some set X, satisfying the following three axioms (for $x, y, z \in X$): Identity of indiscernibles $d(x, y) = 0 \Leftrightarrow x = y$, symmetry $d(x, y) = d(y, x)$, and the triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$.

the relation $c(x, y) = c(x, z)$ should indicate that both y and z are equally likely matches for x. Further, definiteness (be it positive or negative) is helpful to guarantee some internal consistency of the measure. Matching cost functions and similarity measures can therefore be seen as a severely relaxed category of metrics and, in reverse, we can use metrics as our cost measures. It is beyond the scope of this thesis to investigate the influence and necessity of different axioms. However, we will briefly look at techniques for (deep) metric learning in this section, as they show some similarities to cost learning for optical flow estimation. We will, first, formulate the metric learning problem as it is commonly used and then discuss techniques and their applicability. Overviews of classical metric learning approaches are given by Kulis [Kul13] and Suárez et al. [SGH21], whereas deep metric learning is discussed by Kaya and Bilge [KB19] and Lu et al. [LHZ17] in more detail. In computer vision, deep metric learning has been successfully applied to face recognition (e.g., [CHL05; DGXZ19; LWY+17; WWZ+18]), person re-identification (e.g., [CCZH17; HBL17]), and other visual similarity tasks.

Metric learning is defined as the problem of learning a metric (sometimes also relaxed to finding a metric-like function) $d : X \times X \to \mathbb{R}_0^+$ that best represents our desired notion of distance (or inverse similarity) on the set X. The majority of approaches, however, do not learn the function d directly, but decompose it into $d(x, y) = \|f(x) - f(y)\|_2$ using the Euclidean distance (or a similar measure) and reformulate the problem as learning a feature encoder $f : X \to Y$ [Kul13; LHZ17; Yan06]. This can be directly compared to patch matching approaches (e.g., [BVS17; GW16]) or, if we view the input samples as overlapping image patches, many of the contemporary optical flow estimation architectures (e.g., [SYLK18; TD20; YR19]), which also learn a (Siamese) feature encoder and use a fixed measure for computing the initial costs. Differences lie in the measure used and the (optional) application of refinement steps on top of the generated result.

Further, end-to-end optical flow estimation methods are generally trained as regression problem, whereas metric learning is often treated as a multi-class clustering problem. The latter is problematic, as many concurrent approaches, such as the center loss [WZLQ16], SphereFace [LWY+17], CosFace [WWZ+18], and ArcFace [DGXZ19] assume a fixed set of clusters with ground-truth labels. They are therefore not (directly) applicable to our problem. Another category of approaches is given by discriminative techniques, such as the contrastive loss [CHL05], triplet loss [SKP15], and structured loss [HBL17; SXJS16]. Instead of assuming a fixed set of classes, they rely solely on the relationship between multiple samples. The general idea behind these techniques is that matching samples (i.e., samples of the same class) are pulled together in the embedding space, whereas non-matching samples are pushed apart. Patch matching techniques can therefore be seen as directly related to discriminative metric learning methods.

Many contrastive and discriminative approaches rely on sample mining to prevent the network from collapsing: The contrastive loss, for example, uses one positive (i.e., matching) sample pair in combination with multiple negative (i.e., non-matching) sample pairs for optimization [CHL05]. While such strategies can be directly applied to patch matching (see, e.g., the fast architecture of Žbontar and LeCun [ŽL16]), application to cost volume learning is not straight-forward. Specifically, in our DICL-based approach, we want to generate costs by incorporating spatial information from the neighboring feature vectors (for example to provide means for regularization). Such approaches may produce cost values for multiple (neighboring) locations in one single step. Therefore, a (natural) sample pair that is positive in regard to one location may not be positive in regard to a neighboring location, making retrieving and optimizing on a balanced set of samples infeasible.

Loss functions that do not depend on sample mining have been proposed, e.g., by Hadsell et al. [HCL06] and Žbontar and LeCun [ŽL16] (accurate architecture). The former use a spring-based model, whereas the latter simply reformulate the patch matching problem as a binary classification problem. A similar approach could be applied as a (possibly additional) loss term on the cost volume, but care may need to be taken as there exists a considerable class imbalance of matching versus non-matching samples. Ground-truth labels (matching/non-matching) can be derived from the ground-truth flow.

2.2 Training and Evaluation Data

It is well understood that training data is a key ingredient for the success of modern deep learning methods. Unfortunately, ground-truth optical flow data is notoriously difficult to obtain for real-world image sequences [BSL+11; GLSU13]. Synthetic 3D imagery containing natural motion, which could be used for high-quality synthetic datasets, is equally hard to come by [WBSB12]. Further, classical techniques did not require large amounts of ground-truth data and solely relied on it for validation and parameter tuning. Combined, this ensures that the benchmarks which best reflect the performance of optical flow estimation methods in a real-world application have, in general, a comparatively little amount of training data.[6] For this reason, as well as performance benefits through curriculum learning [BLCW09], concurrent deep learning methods are generally trained with a multi-dataset strategy, only fine-tuning on the final benchmark datasets (see, e.g., [IMS+17; SYLK20; TD20; WZD+20]).

In this section, we will briefly discuss available datasets, as well as augmentations and other advancements regarding training data and strategies. We will begin with Section 2.2.1, presenting the commonly used benchmark and evaluation datasets, which contain a mixture of real-world and high quality synthetic data. Section 2.2.2 goes over many of the datasets that have been devised specifically for training, which are exclusively of synthetic nature. Therein, we will discuss the most commonly used and fairly general datasets, as well as touch on some more task-specific datasets, designed, e.g., for autonomous driving or (human) action recognition. At last, Section 2.2.3 will discuss improvement in training via both the prevalent training schedule structure and data augmentations.

2.2.1 Benchmarks and Common Evaluation Sets

Comparative evaluation is critical for the advancement of any group of technical or scientific methods. The earliest attempt at a standardized and publicly available benchmark was presented by Baker et al. [BSL+11] with their Middlebury dataset, split into a public training set and private test samples for evaluation.[7] Prior to this, research relied on surveys and reviews, such as the one by Barron et al. [BFB94], and independently published results on commonly used sequences, such as the Yosemite sequence by Quam (see Barron et al. [BFB94] for details thereon). The Middlebury dataset consists of a, in today's standard, tiny number of samples depicting both synthetic scenes

[6] A notable exception being the fairly recent VIPER benchmark [RHK17], which we will discuss below.

[7] In general, all *benchmarks* that we mention (explicitly by that term) here and below do not publish their test/evaluation ground-truth flow data to prevent cheating. Only input images are provided.

with independent motion and rigid scenes from the real world. To obtain ground-truth data for real-world images, Baker et al. relied on a specialized lab setup, tracking a hidden fluorescent texture only visible in UV light, which, however, limited its applicability to indoor scenes. A notable addition to the synthetic part of the training dataset was later provided by Mac Aodha et al. [MBP10; MHPB13] with the (extended) UCL dataset. Due to the considerably small number of samples, the somewhat limited realism, as well as the saturation of accuracy on it by top-performing methods, the Middlebury benchmark is of little relevance today.

Nowadays, evaluation of deep learning optical flow estimation methods focuses on three main datasets and their corresponding benchmarks: The synthetic MPI Sintel dataset [BWSB12], as well as the real-world KITTI 2012 [GLU12] and KITTI 2015 [MHG15; MHG18] datasets. While all three allow fine-tuning, they, however, also require pre-training as neither contains enough samples to fully train concurrent approaches. The Sintel dataset by Butler et al. [BWSB12] was generated from the 2010 computer-animated open-source short film of the same name, which, in turn, was created by the Blender Foundation as a showcase of their open-source 3D software Blender. As such, this benchmark contains high-quality and naturalistic motion, making it a challenging task even for concurrent methods. Two render passes are available: “clean” and “final”. The “clean” pass contains natural lighting and illumination, whereas the “final” pass contains additional effects, such as motion blur and depth of field. Wulff et al. [WBSB12] describe in more details the steps required to ensure their level of quality and, in general, what needs to be considered when distilling a 3D movie into a dataset for use in computer vision.

The KITTI 2012 [GLU12] and KITTI 2015 [MHG15; MHG18] datasets, on the other hand, are based on real-world automotive data. Ground-truth data for both optical flow and stereo estimation has been obtained via LiDAR. An overview of the process is given by Geiger et al. [GLSU13]. Due to this, however, ground-truth data is sparse and, in the 2012 version, restricted to rigid scenes only, without any independently moving objects. In the 2015 version, ground truth for independently moving vehicles has been added by fitting 3D CAD models through a semi-automated annotation process relying on manual input [MG15].

Other notable benchmarks include the HCI HD1K [KNH+16] and the VIPER benchmark [RHK17]. The HCI HD1K benchmark is comparatively similar to the KITTI benchmarks in that it uses real-world automotive data obtained using LiDAR. In contrast to KITTI, however, this benchmark provides higher resolution images, denser ground-truth flow, and focuses on uncertainty in the ground truth estimation process, which it provides as uncertainty maps. A more in-depth comparison as well as an overview of the data and its acquisition is given by Kondermann et al. [KNH+16; KNM+15].

A radically different approach is taken by Richter et al. [RHK17] with their VIPER project: They note that realism in virtual worlds is a content creation problem, which is currently best addressed by commercial productions. To make use thereof, they propose obtaining ground-truth data for various computer vision tasks by hooking into the graphics processing APIs used by computer games, a technique commonly used by the game modding community. Via this, they are able to obtain a significantly larger amount of training, validation, and test data compared to the previously discussed benchmarks, while still retaining a high measure of both quality and variety regarding motion. Among optical flow, the VIPER dataset and benchmark provides ground-truth data for

semantic instance segmentation, object detection and tracking, object-level 3D scene layout, and visual odometry [RHK17]. Most recently, the 2020 Robust Vision Challenge[8] combined the Middlebury, KITTI, MPI Sintel, and VIPER benchmarks for a complete evaluation.

2.2.2 Synthetic Training Sets

Due to the difficulties in obtaining ground-truth data for optical flow estimation, a number of different synthetic datasets have been proposed specifically for (pre-)training of deep-neural-network-based methods. To train their FlowNet model, Dosovitskiy et al. [DFI+15] created the FlyingChairs dataset using rendered images of 3D chair models superimposed onto background images containing real-world urban and landscape scenes. Randomized affine transformations were applied to the chair and background image to generate both the first and second frame. Transformations in between frames were sampled such that the displacement histogram of the FlyingChairs dataset resembles the one from the MPI Sintel set. Camera motion was simulated by making chair transformations relative to the background transformation [DFI+15].

While this approach generates fairly simplistic motion that cannot convey information about depth, it has been proven effective and useful as a first training stage for large neural networks (see, e.g., Ilg et al. [IMS+17] and Sun et al. [SYLK20]), and is commonly used as such (e.g., in RAFT [TD20] and DICL [WZD+20]). Several extensions to this dataset have been suggested: FlyingChairs2 [ISKB18] has been created using the same settings as the original FlyingChairs dataset, however additionally provides backward flow, motion boundaries, and occlusion maps. Similarly, FlyingChairsOcc [HR19] provides both backward flow and occlusion maps. ChairsSDHom [IMS+17] and its extended version [ISKB18] have been designed to obtain a motion histogram closer to the UCF101 dataset [SZS12] for action recognition, which has overall smaller displacements. The extended version provides additional ground-truth modalities, equivalent to what FlyingChairs2 provides over FlyingChairs.

To incorporate more realistic motions, Mayer et al. [MIH+16] created the FlyingThings3D dataset using 3D models from ShapeNet [CFG+15]. Simple geometric shapes were used as background objects and assembled randomly on a ground plane, foreground objects were sampled from ShapeNet. Both the smaller number of foreground objects and the camera follow a randomly generated smooth 3D trajectory, whereas the larger number of background objects remain static. Textures were chosen randomly from procedurally generated images, as well as landscape, cityscape, and texture-style photographs [MIH+16]. In addition to ground-truth data for forward and backward flow, FlyingThings3D also provides stereo disparities and disparity changes for full scene flow, motion boundaries, as well as object and material segmentation maps. Similar to Sintel [BWSB12], it also provides both a "clean" and a "final" render pass.

Due to its considerable size and 3D motion, it is commonly used as a second stage when training (see, e.g., Sun et al. [SYLK20], FlowNet 2.0 [IMS+17], or RAFT [TD20]). Mayer et al. [MIH+16] provide two more datasets, containing the same modalities as FlyingThings3D: Monkaa and Driving. The Monkaa dataset is based on the animated open-source Monkaa movie, which, similar to the Sintel movie, was made by the Blender Foundation. Its unique properties are the use of softly articulated non-rigid motion in animals as well as visually challenging fur. In contrast to the Sintel

[8] Found online at http://www.robustvision.net/rvc2020.php. Accessed on April 19, 2022.

dataset, however, the Monkaa dataset contains additional custom scenes, created using 3D assets of the movie, and the authors explicitly state that they do not strive for the same amount of naturalism [MIH+16]. The Driving dataset contains synthetic automotive scenes and was specifically modeled to (loosely) resemble the KITTI datasets [MIH+16].

Mayer et al. were not the only ones to recreate the dynamics of the real-world KITTI datasets. In particular, Gaidon et al. [GWCV16] cloned sequences thereof in their Virtual KITTI dataset, using the Unity game engine. This allowed them to create a larger dataset by performing small changes to obtain multiple variations of each scene. These variations are not only limited to (among other things) the number, colors, shapes, and paths of cars or the camera trajectory, but also include different lighting and weather conditions. Re-creation as a digital world further enabled them to include accurate high-resolution ground truth for optical flow, as well as for object detection, tracking, and scene and instance segmentation [GWCV16]. Being able to adjust weather and imaging conditions allowed them to investigate and compare algorithms in challenging visual scenarios. The Virtual KITTI dataset was improved upon by Cabon et al. [CMH20] with the Virtual KITTI 2 dataset, using advancements in the Unity game engine to achieve better photo-realism. Virtual KITTI 2 retains the same sequences and geometry as Virtual KITTI, but additionally contains backward optical flow and stereo imagery for scene flow.

Somewhat similar to FlyingThings3D, McCormac et al. [MHLD17] sampled random objects and textures to produce a large-scale synthetic 3D dataset with 5 million samples: SceneNet RGB-D. Notably different, however, they focused on physically accurate and (mostly) realistic indoor scenes by sampling objects according to the room or scene type and a desired predefined density. Objects were sampled from ShapeNet [CFG+15], scene layouts from SceneNet [HPB+16]. Additionally, they used a physics simulator to obtain a physically plausible and stable configuration of objects after initial random placement, obeying gravity and collisions. Scenes were rendered via ray-tracing to obtain photorealistic lighting and visual effects, including transparency and reflections. While rendering, the physics simulation was disabled and objects kept static. Only the camera was moved, following a randomly generated trajectory [MHLD17].

Inspired by the lack of realistic human motion on previously available datasets, Ranjan et al. [RHT+20] created the multi-human optical flow dataset. For this, they followed a randomized but realism-focused approach, conceptually similar to the one of SceneNet RGB-D. In particular, they employed a parametrized human model that allowed for randomization of body shapes and relied on a motion capture database to generate plausible and realistic human motion.

Recently, Sun et al. [SVH+21] proposed a different and, in this space, novel methodology for automated generation of synthetic optical flow training data. Their AutoFlow approach learns to jointly optimize both training data and flow estimation model in regard to a target dataset, such as Sintel or KITTI. To achieve this, they designed a parametric model for dataset generation as well as data augmentations. Hyperparameters of this model dictate shape, structure, and complexity of the generated data, specific instances are sampled in accordance thereof. Sun et al. employed a population-based training approach (see Jaderberg et al. [JDO+17]) for said joint optimization. Using both RAFT [TD20] and PWC-Net [SYLK18; SYLK20] as models, this technique not only allowed them to obtain equal or better estimation results over the commonly used strategy (training on FlyingChairs followed by FlyingThings3D and fine-tuning), but also helped them gain better insights into what distribution of data works best. For example, they note that the motion statistics

differ from those of Sintel and FlyingChairs (which has been modeled after the Sintel motion histogram), in that AutoFlow focuses on middle-range motion and has little small motion [SVH+21]. The samples generated from training RAFT were published as the AutoFlow Static dataset.

2.2.3 Data Scheduling and Augmentations

Not only models and datasets are important, but also the strategy in which data is presented. Sun et al. [SYLK20], for example, were able to improve the performance of FlowNetC [DFI+15] significantly, simply by use of additional training data and an adapted training schedule. The strategies prevalent today, while slightly different for each model, follow the same larger structure (see, e.g., RAFT [TD20], DICL [WZD+20], or PWC-Net [SYLK18; SYLK20]): First, the model is pre-trained on FlyingChairs. It is assumed that the simplistic motion and the detailed structure of chairs helps the model learn in the early stages [MIF+18]. Second, the model is trained with FlyingThings3D, which provides more realistic motion, as discussed in the previous section. Lastly, it is fine-tuned on the target dataset, for example Sintel or KITTI. Some approaches, notably RAFT [TD20], use a mixture of data during this step. This sequence has been proven effective, for example, by Ilg et al. [IMS+17] and Sun et al. [SYLK20].

The overall concept of employing multiple training stages with varying difficulty is known as curriculum learning and has been shown to improve performance in other machine learning areas as well [BLCW09]. In particular, Bengio et al. [BLCW09] were able to show improvements in generalization by help of curriculum learning and hypothesize that it can improve the speed of convergence and the quality of the local minima found. An overview of the impact of training schedules is given by Sun et al. [SYLK20], while Mayer et al. [MIF+18] additionally investigate the composition of different synthetic datasets and explore their use for different training stages.

In addition to multi-dataset training schedules, randomized data augmentation has been proven useful as a means to improve generalization and prevent overfitting, both for optical flow estimation [DFI+15; SVH+21] and for neural networks in computer vision in general [KSH12]. To prevent overfitting, augmentations aim to reduce the number of duplicate input samples that the model receives during training by modifying them in a randomized fashion, whereas to improve generalization, specific properties (such as brightness and color) can be changed to make samples more challenging. In the two-frame optical flow estimation case, changes can be applied symmetrically to both images or, asymmetrically, to each image individually (e.g., to simulate a change in lighting or to add occlusion). Commonly applied augmentations can be divided in photometric and geometric: Geometric augmentations include translation, rotation, scaling, cropping, and flipping axes, whereas photometric augmentations include color changes (be it multiplicative or additive changes, brightness changes, or gamma changes), as well as additive noise. The specific probabilities and ranges with which to apply these changes are hyperparameters.

Recently, Bar-Haim and Wolf [BW20] investigate the effects of scoping, i.e., the combination of an affine transformation and cropping. Their results suggest that larger scopes may achieve better performance and that augmentations should be relaxed over the training stages. Sun et al. [SVH+21] suggest automatic tuning thereof (in a time-varying manner), in combination with their AutoFlow dataset generation approach. Similar strategies for automated learning of augmentations were proposed in the general computer-vision domain, for example, by Cubuk et al. [CZM+19] and Cubuk et al. [CZSL20].

2.3 Error Measures

We have covered data in the previous section, but are still missing a key ingredient for comparative evaluation and benchmarks: Error measures. Several ways of measuring errors have been proposed over the years by different benchmarks and surveys, focusing on different aspects, such as the error in endpoint or angle of the flow, or the percentage of outliers under a respective definition thereof. In this section, we will briefly discuss the commonly used measures and give their definitions. We first state the nomenclature and prerequisites required, before continuing on with the actual measures, beginning with the endpoint error. Thereafter, we discuss the angular error and, finally, the outlier-based measures, including the bad pixel error and the Fl-all score.

We define measures over a domain of pixels Ω given by their position $\mathbf{x} \in \Omega$. This set is generally restricted to pixels with ground truth available, however, may be confined further: For example, next to overall scores, the KITTI benchmarks [GLU12; MHG18] publish separate scores for foreground and background pixels. Similarly, the MPI Sintel benchmark [BWSB12] publishes additional scores for matched and unmatched pixels, i.e., pixels that are visible in both frames and pixels that are not. We let $\mathbf{u}_{\text{gt}}$ and $\mathbf{u}_{\text{est}}$ denote the ground-truth and estimated flow fields, respectively, with $\mathbf{u} = (u, v)$ denoting the horizontal and vertical components of the flow. Lastly, we define the L^p norm as

$$\|\mathbf{v}\|_p \coloneqq \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}, \tag{2.1}$$

where $\mathbf{v} \in \mathbb{R}^n$ and $p \in [1, \infty)$. Specifically, we make use of the L^2 norm, commonly referred to as the Euclidean norm, which implies the Euclidean distance $\|\mathbf{v} - \mathbf{s}\|_2$ between positions $\mathbf{v}, \mathbf{s} \in \mathbb{R}^n$.

2.3.1 Endpoint Error and Average Endpoint Error

The most straight-forward error measure is given by the *endpoint error (EE)*:

$$\text{EE}\left(\mathbf{u}_{\text{gt}}(\mathbf{x}), \mathbf{u}_{\text{est}}(\mathbf{x})\right) \coloneqq \left\|\mathbf{u}_{\text{gt}}(\mathbf{x}) - \mathbf{u}_{\text{est}}(\mathbf{x})\right\|_2. \tag{2.2}$$

It is defined as the Euclidean distance between the endpoints, derived from the ground-truth flow $\mathbf{u}_{\text{gt}}$ and estimated flow $\mathbf{u}_{\text{est}}$ at a specific pixel $\mathbf{x} \in \Omega$. As a result, it considers both direction and magnitude of the flow. Following Equation (2.2), we define the *average endpoint error (AEE)* as

$$\text{AEE}\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right) \coloneqq \frac{1}{|\Omega|} \sum_{\mathbf{x} \in \Omega} \text{EE}\left(\mathbf{u}_{\text{gt}}(\mathbf{x}), \mathbf{u}_{\text{est}}(\mathbf{x})\right), \tag{2.3}$$

i.e., the endpoint error averaged over all pixels $\mathbf{x} \in \Omega$ in the respective domain. This measure often poses as the basis for loss functions, however, the average L^1 distance may be used as well. Both, endpoint error and average endpoint error, are generally given in pixels (px).

2.3.2 Angular Error and Average Angular Error

The *angular error (AE)* was introduced by Fleet and Jepson [FJ90] as the angle between spatio-temporal flows $\mathbf{w} = (u, v, 1)^\top$:

$$\text{AE}\left(\mathbf{w}_{\text{gt}}(\mathbf{x}), \mathbf{w}_{\text{est}}(\mathbf{x})\right) \coloneqq \arccos \frac{\mathbf{w}_{\text{gt}}(\mathbf{x})^\top \mathbf{w}_{\text{est}}(\mathbf{x})}{\|\mathbf{w}_{\text{gt}}(\mathbf{x})\|_2 \cdot \|\mathbf{w}_{\text{est}}(\mathbf{x})\|_2}. \tag{2.4}$$

Similar to the above, $\mathbf{w}_{\text{gt}}$ refers to the ground-truth spatio-temporal flow field, $\mathbf{w}_{\text{est}}$ to the estimated spatio-temporal flow field, and $\mathbf{x} \in \Omega$ to a pixel in the respective domain. Note that the spatio-temporal flow $\mathbf{w} = (u, v, 1)^\top$ can never be zero due to its time component being fixed to one. Hence, using the spatio-temporal flow $\mathbf{w}$ instead of the instantaneous flow $\mathbf{u}$ ensures that the angular error remains well-defined, even if instantaneous ground-truth or estimated flow equal zero.

A particularly useful attribute of the angular error is its capability to handle areas with vastly different flow magnitudes in a largely comparable way. In contrast to the endpoint error, its dependence on the magnitude is significantly reduced. Though, due to considering both time and space, this dependence is not eliminated completely: A larger flow magnitude still leads to higher error values, but errors will instead run into a limit, imposed (in part) by the instantaneous flows. See Barron et al. [BFB94] and Fleet and Jepson [FJ90] for details and illustrations thereon.

Barron et al. [BFB94] popularized the *average angular error (AAE)* for the evaluation of optical flow estimation methods. Building up on Equation (2.4), we define it as the angular error averaged over our domain Ω, i.e.,

$$\operatorname{AAE}\left(\mathbf{w}_{\text{gt}}, \mathbf{w}_{\text{est}}\right) \coloneqq \frac{1}{|\Omega|} \sum_{\mathbf{x} \in \Omega} \operatorname{AE}\left(\mathbf{w}_{\text{gt}}(\mathbf{x}), \mathbf{w}_{\text{est}}(\mathbf{x})\right), \tag{2.5}$$

where $\mathbf{w} = (u, v, 1)^\top$ is, again, the spatio-temporal flow. Both, angular error and average angular error, are normally given in degrees.

2.3.3 Bad Pixel Error

The *bad pixel error (BP)* describes the proportion of flow outliers, i.e., the number of bad pixels in relation to the number of overall pixels of our domain Ω. In this case, outliers are defined via an absolute threshold t on the endpoint error. To define the bad pixel error, we first define the set of *absolute bad pixels* $B^{\Omega}_{\text{abs},t}$, using the endpoint error (EE) given in Equation (2.2), as

$$B^{\Omega}_{\text{abs},t}\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right) \coloneqq \left\{\mathbf{x} \in \Omega : \operatorname{EE}\left(\mathbf{u}_{\text{gt}}(\mathbf{x}), \mathbf{u}_{\text{est}}(\mathbf{x})\right) > t\right\}, \tag{2.6}$$

where $t \in \mathbb{R}^+$ is the aforementioned outlier threshold. Any pixel with an endpoint error larger than t is considered as being an outlier. With this, we can now define the bad pixel error as

$$\operatorname{BP}_t\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right) \coloneqq \frac{1}{|\Omega|}\left|B^{\Omega}_{\text{abs},t}\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right)\right|. \tag{2.7}$$

Note that, in literature, the bad-pixel error measure itself is commonly scaled by a factor of 100. We, however, reject this notion, as the measure is inherently of proportional nature. Therefore, we will follow the original definition employed by Scharstein and Szeliski [SS02], without said scaling, and give its values in percent.

2.3.4 Fl-all Measure

Introduced by Menze and Geiger [MG15], the *Fl-all* measure describes a similar outlier-based proportion as the bad pixel error, however, in contrast to that, it employs both absolute and relative terms for defining outliers. Hence, we will first define the set of *relative bad pixels* $B^{\Omega}_{\text{rel},p}$, again using the endpoint error (EE) given in Equation (2.2), as

$$B^{\Omega}_{\text{rel},p}\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right) \coloneqq \left\{\mathbf{x} \in \Omega : \operatorname{EE}\left(\mathbf{u}_{\text{gt}}(\mathbf{x}), \mathbf{u}_{\text{est}}(\mathbf{x})\right) > p \cdot \left\|\mathbf{u}_{\text{gt}}(\mathbf{x})\right\|_2\right\}, \tag{2.8}$$

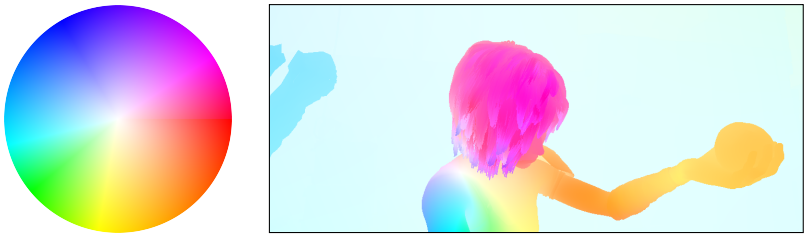

Figure 2.1: Middlebury color map [BSL+11] for visualization of optical flow vector fields (left) and example visualization of Sintel ground-truth flow using this color map (right). Hue indicates direction, saturation indicates length of vectors.

where $p \in \mathbb{R}^+$ is the relative outlier threshold. Following this definition, pixels are considered outliers if they have an endpoint error larger than a specified percentage of the magnitude of their ground-truth flow. Using Equations (2.6) and (2.8), we can now give the (generic) Fl-Ω measure as

$$\text{Fl-}\Omega\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right) \coloneqq \frac{1}{|\Omega|}\left|B^{\Omega}_{\text{abs},3}\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right) \cap B^{\Omega}_{\text{rel},0.05}\left(\mathbf{u}_{\text{gt}}, \mathbf{u}_{\text{est}}\right)\right|. \tag{2.9}$$

Therefore, the Fl-Ω measure describes the proportion of pixels that have an endpoint error larger than both 3 px and 5% of the ground-truth flow. From this, we obtain the *Fl-all* measure by defining Ω as all pixels with available ground-truth flow. This measure is primarily used in the KITTI benchmarks, in combination with the *Fl-fg* and *Fl-bg* measures, confining Ω to foreground and background pixels, respectively.

Note that the impact of the relative term is limited to pixels with a ground-truth flow magnitude larger than or equal to 60 px, as $3\,\text{px} > 0.05 \cdot \|\mathbf{u}_{\text{gt}}(\mathbf{x})\|_2$ for $\|\mathbf{u}_{\text{gt}}(\mathbf{x})\|_2 < 60\,\text{px}$. Meaning, if the absolute endpoint error threshold of 3 px is fulfilled, the relative one will be as well, as long as the ground-truth flow magnitude is smaller than 60 px.

2.4 Visualization

While we can already compare methods using the measures defined in the previous section, inspection through visualization often reveals details that cannot be conveyed by these measures and, further, allows for viewing localized errors. In this section, we briefly present the visualization schemes used throughout this thesis. Specifically, we will discuss, in this order, the color map used to visualize flow fields, an outlier-centric endpoint error visualization, and the method we use to visualize cost volume slices.

Optical Flow To visualize optical flow vector fields, we use the coloring strategy proposed by Baker et al. [BSL+11], illustrated in Figure 2.1. The direction of flow vectors is represented by the corresponding hue, whereas length is depicted by saturation, with more muted colors representing shorter flow vectors. We use black to indicate that no flow data is available for the respective pixels. This is, for instance, the case in the KITTI ground-truth data due to limitations of the LiDAR technology used to obtain it [GLU12]. An alternative color map is, for example, given by Bruhn [Bru06].

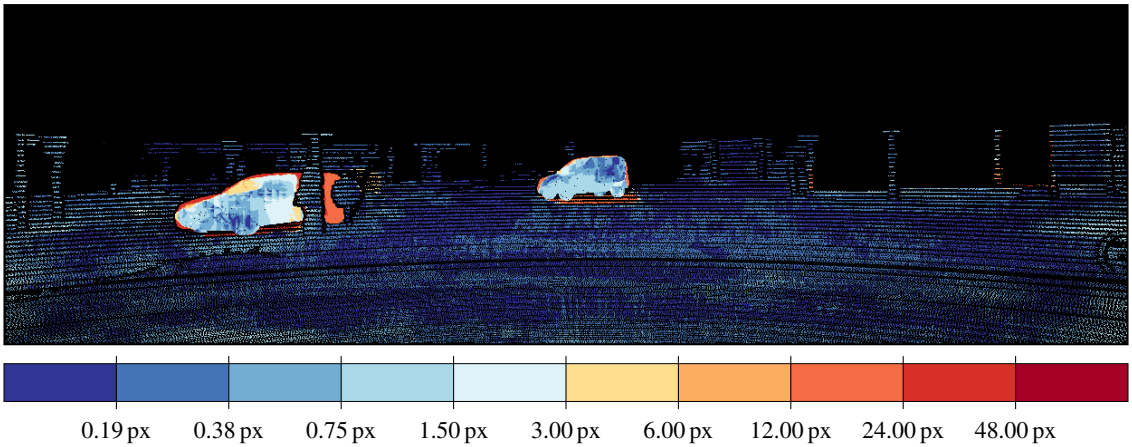

Figure 2.2: Endpoint error visualization following Menze and Geiger [MG15]. Example visualization (top) and logarithmic color map (bottom). Blue shades indicate inliers (i.e., pixels with endpoint error smaller than 3 px), red shades indicate outliers. Black indicates that no ground-truth data is available.

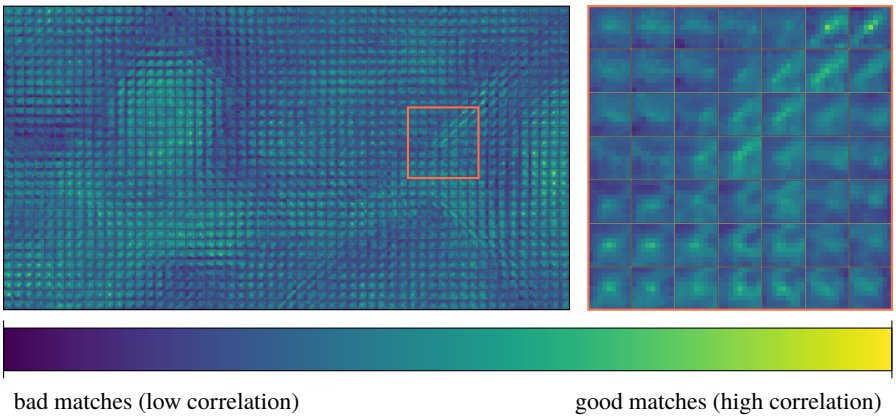

Figure 2.3: Example visualization for a cost volume slice (left) with an excerpt thereof (marked in orange) to the right. Individual source pixels are depicted as square blocks, separated by gray lines, containing the colored values for their respective displacements. Good feature matches (high correlation) are indicated by bright yellow colors, whereas bad matches (low correlation) are indicated by dark purple colors.

Endpoint Error For visualization of the endpoint error, Menze and Geiger [MG15] propose a logarithmic color map, illustrated in Figure 2.2. This map shows pixels considered as inliers in blue shades and pixels considered as outliers in red shades. The definition of outliers follows the Fl-all measure, ignoring the relative error part: Pixels are considered outliers if their endpoint error equals or exceeds 3 px (cf. Section 2.3.4 and Equation (2.9)). The logarithmic scale is centered around this threshold. We again use black when no data is available for the specific pixels.

Cost Volume To inspect cost volumes, we visualize four dimensional slices thereof. Specifically, we unfold a slice of one cost volume level into a two-dimensional plane by collecting costs for all displacement hypotheses of a source pixel, grouping them in a larger block (one per source-pixel), and placing them correspondingly to their direction inside that block. Values are then visualized via a standard color map. This, in essence, creates one meta-pixel for each source pixel, showing all displacement costs thereof. An example visualization is provided in Figure 2.3. Note that we actually visualize correlation scores instead of costs, meaning that higher values mean a better feature match and lower values a worse one (i.e., we visualize inverse matching costs).

3 Base Methods and Core Idea

With the prerequisites discussed in the previous two sections, we can now move our attention to the central aspects of this thesis and derive its core concept: The combination of RAFT and DICL. We begin with Sections 3.1 and 3.2, presenting the architectures of RAFT and DICL, respectively, focusing on the aspects relevant to our approach. In particular, we will elaborate on the recurrent flow estimation technique employed by RAFT, including its cost volume and the sampling-based access thereof, as well as the displacement invariant cost learning module around which DICL is built. Once we have familiarized ourselves with both architectures, we will, in Section 3.3, revisit the motivation and goal of this thesis and derive the fundament of our integration of DICL into RAFT, upon which all of our later experiments rely. Finally, Section 3.4 discusses further concepts, equally common to all of our proposed models. Specifically, we will compare warping- and sampling-based strategies for residual flow estimation and discuss gradient blocking.

3.1 RAFT: Recurrent All-Pairs Field Transform for Optical Flow

The "recurrent all-pairs field transform for optical flow" (RAFT) by Teed and Deng [TD20] is the baseline for many of the current state-of-the art methods, such as Separable Flow [ZWPT21] and GMA [JCL+21]. As already elaborated in Section 2.1, RAFT combines multiple principles, in particular iterative refinement, cost volume sampling, and estimation at a single level, that all factor into its success. It is, to our knowledge, the first approach using a true recurrent neural network (RNN) for estimating optical flow by residual refinement.[1] This allows it to trade quality of the solution against the speed with which it is produced by changing the number of recurrent iterations at runtime, without the need for additional training or distillation. Estimation at a single level, especially combined with the chosen sampling strategy, allows RAFT to incorporate smaller details, which is often a problem in the (until then predominant) coarse-to-fine warping strategies. By combining said principles, RAFT provides a significant improvement in performance over its baseline methods.

In the remainder of this section, we will present a somewhat detailed but general outline of the RAFT architecture and its components. We begin in Section 3.1.1 with a top-level overview, after which we will have a closer look at the most important concepts, including the cost volume, the lookup operator used for sampling the cost volume, and the recurrent refinement network. Finally, we will discuss the loss function and supervision approach taken by RAFT in Section 3.1.2. For more details, such as the number of channels for specific layers as well as their composition, we refer to the original paper by Teed and Deng [TD20] and its corresponding implementation.

[1] The density embeddings in HD^3 [YDY19] could be construed as hidden state, however, network weights are not shared across levels, i.e., iterations.

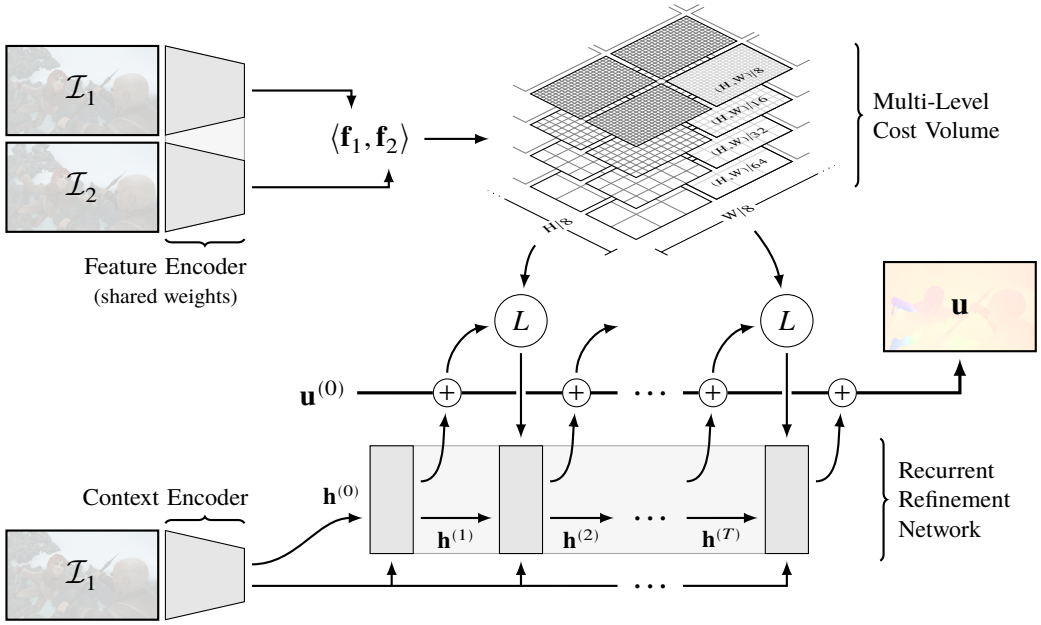

Figure 3.1: Overview of the RAFT method. Shared feature encoders transform input images $\mathcal{I}_1$ and $\mathcal{I}_2$ into embeddings $\mathbf{f}_1$ and $\mathbf{f}_2$, respectively. From those, costs are computed via the inner product $\langle \mathbf{f}_1, \mathbf{f}_2 \rangle$. The recurrent refinement network estimates flow by producing a residual update that is added to the current estimate $\mathbf{u}^{(t)}$. It samples from the cost volume via the lookup operator L, taking said estimate as input. A context encoder is used to both initialize the hidden state $\mathbf{h}^{(0)}$ and provide input to the recurrent network in each iteration. Learned neural network blocks are indicated in gray, with light gray background across multiple blocks indicating shared weights.

3.1.1 Architecture

Similar to the majority of other neural network based optical flow estimation methods, RAFT first transforms the input images $\mathcal{I}_1$ and $\mathcal{I}_2$ into a suitable feature representation $\mathbf{f}_1 = f(\mathcal{I}_1)$ and $\mathbf{f}_2 = f(\mathcal{I}_2)$ using a Siamese feature encoder network $f : [0,1]^{H\times W\times 3} \to \mathbb{R}^{H/8\times W/8\times D}$, i.e., a convolutional neural network (CNN) in which the encoders share the same weights and structure for both inputs. In contrast to many competing techniques, however, RAFT only produces a single-level feature representation, specifically by downsampling the input by 8 in resolution: Assuming an input resolution of $H \times W$ (with 3 color channels), it generates features of resolution $H/8 \times W/8$ with $D = 256$ channels. These features are then used to compute matching costs at that resolution via a scaled inner product. Sharing weights between encoders ensures that the learned embeddings are compatible and allows retaining symmetricity and other properties, such as positive semi-definiteness, of said (fixed) matching cost function (applied to the embeddings) with respect to the input images (cf. deep metric learning).

These initial matching costs are then used to create a hierarchical multi-level cost volume, which poses as the fixed basis for the subsequent recurrent estimation and refinement. Flow is estimated at a resolution of $H/8 \times W/8 = H_f \times W_f$ using a convolutional RNN. The hidden state $\mathbf{h}^{(t)}$ of this RNN is initialized from a context encoder network (applied to the first input image $\mathcal{I}_1$), which shares its structure (but not weights) with the feature encoders. In each iteration t, the RNN samples from the

cost volume via a lookup operator and computes a residual[2] update $\mathbf{u}_{\text{res}}^{(t)}$ to the current flow estimate $\mathbf{u}^{(t-1)}$ as $\mathbf{u}^{(t)} = \mathbf{u}^{(t-1)} + \mathbf{u}_{\text{res}}^{(t)}$. All three, the cost volume, the lookup operator, and the recurrent refinement network are described below in more detail. After the final iteration $t = T$, the updated flow estimate $\mathbf{u}^{(t)} \in \mathbb{R}^{H_f \times W_f \times 2}$ is upsampled to the full resolution estimate $\mathbf{u}_{\text{up}}^{(t)} \in \mathbb{R}^{H \times W \times 2}$ via a convex combination of a 3×3 neighborhood. The weights for this combination are computed from the hidden state $\mathbf{h}^{(t)}$ via a learned CNN. This allows for a significant reduction of computational costs during the main stages while still providing the details, structure, and clean motion boundaries only achievable with full resolution output and provides an improvement over simple bilinear upsampling [TD20]. An illustration of the RAFT architecture (neglecting the upsampling module) is given in Figure 3.1.

Cost Volume

Initial matching costs are computed at resolution $H/8 \times W/8 = H_f \times W_f$ via the scaled inner product

$$C_{i,j,k,l}^{(0)} = \frac{1}{\sqrt{D}} \left(\mathbf{f}_{1,i,j}\right)^{\top} \left(\mathbf{f}_{2,k,l}\right), \tag{3.1}$$

where the scaling factor $1/\sqrt{D}$ is used to retain unit variance (assuming all inputs have unit variance). As RAFT considers all feature pairs, this leads to the finest cost volume level $C^{(0)} \in \mathbb{R}^{H_f \times W_f \times H_f \times W_f}$. Due to the scaled dot product as matching cost function, computation can be implemented efficiently via a simple matrix multiplication, applied after reshaping the feature tensors. Coarser levels $m \in \{1, 2, 3\}$ are derived from the finest level by average-pooling in the domain of the second frame ($\mathcal{I}_2$), i.e., over indices k and l, resulting in $C^{(m)} \in \mathbb{R}^{H_f \times W_f \times (H_f/2^m) \times (W_f/2^m)}$ with

$$C_{i,j,k,l}^{(m)} = \frac{1}{2^{2m}} \sum_{p=0}^{2^m-1} \sum_{q=0}^{2^m-1} C_{i,j,2^m k+p,2^m l+q}^{(0)}. \tag{3.2}$$

Therefore, each source feature element in the first domain has a dedicated cost pyramid across multiple levels, which is illustrated in Figure 3.2 (left). RAFT employs four of these levels, with downsampling factors from 8 on the finest to 64 on the coarsest one. Keeping the higher resolution in domain of the first frame ($\mathcal{I}_1$) allows estimating flow at that fixed resolution and further helps incorporate details and smaller structures. Downsampling in the domain of the second frame provides the necessary spatial range when looking only at a small slice or neighborhood of the cost volume.

Lookup Operator

Processing and incorporating the cost volume directly into the recurrent estimation and refinement network is infeasible due to its prohibitively large size. Therefore, a lookup operator is used to sample a specific (fixed-size) slice of the multi-level cost volume, given by a set of displacement

[2]We borrow this terminology from numerical analysis, where it finds wide use in iterative solvers (to which, not incidentally, many optical flow estimation methods show similarities). In particular, the residual is given by the transformed error. More precisely, given some equation $g(x) = y$ to be solved with optimal solution x^*, the error is defined as $e := x^* - x$ whereas the residual is defined as $r := g(x^*) - g(x) = y - g(x)$. We can make the connection to optical flow by defining our estimation approach to be the function g mapping some internal network state x to the estimated flow y. RAFT therefore tries to estimate the residual r in each iteration, rather than the full flow y.

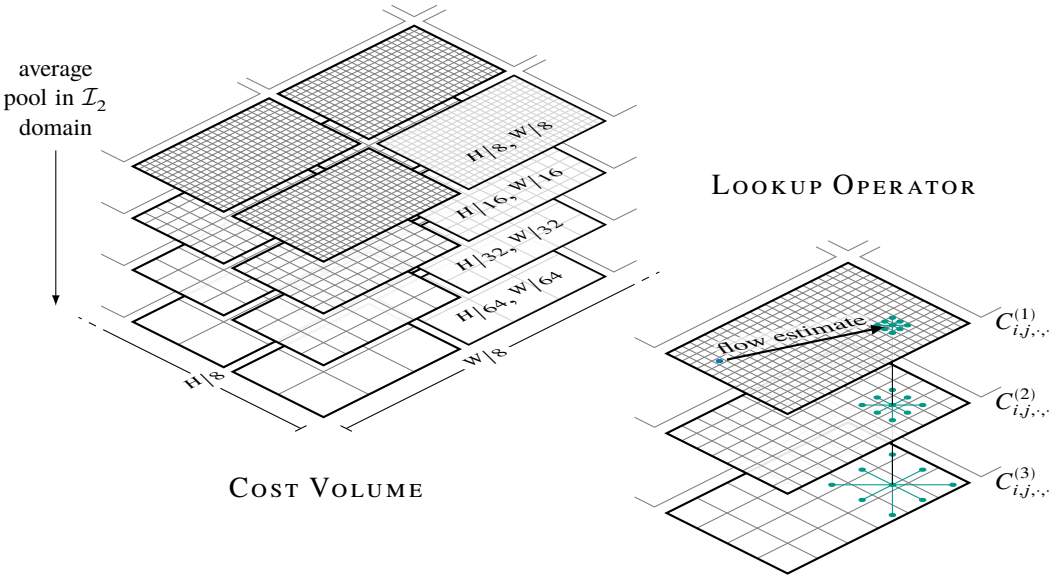

Figure 3.2: Slices of the RAFT multi-level cost volume (left) and lookup operator (right). The initial cost volume level is created at a resolution of $(H/8 \times W/8) \times (H/8 \times W/8)$ and represents the finest resolution. Coarser levels are obtained by average-pooling the initial level in domain of the second frame ($\mathcal{I}_2$). Size in domain of the first frame ($\mathcal{I}_1$) remains equal. The lookup operator uses the current flow estimate as an offset to the source pixel position $(i,j)^\top$ (blue dot) to compute the sampling position (center green dot). It samples values from the cost volume simultaneously over all levels (here 3 instead of the original 4), using a stencil (here 3×3) and bilinear interpolation.

hypotheses $\{\mathbf{d}_k\}_k \subset \mathbb{Z}^2$. Let us assume for now that we are only interested in costs and flow for a single feature pixel[3] $\mathbf{p}_{i,j} = (i,j)^\top$. Displacement hypotheses $\mathbf{d}_k$ represent the relative offset of the second-frame pixel $\mathbf{p}_{k,l} = (k,l)^\top$ to that source pixel $\mathbf{p}_{i,j}$, i.e., $\mathbf{p}_{k,l} = \mathbf{p}_{i,j} + \mathbf{d}_k$. Hence, by restricting the set of possible displacement hypotheses, we define a slice of the full cost volume. The lookup operator then returns the costs for that pixel and the predefined of set displacement hypotheses $\{\mathbf{d}_k\}_k$, usually restricted to a small range around zero. Crucially, it does so by taking the current flow estimate $\mathbf{u}^{(t-1)}_{i,k}$ into account. This can essentially be interpreted as moving the source pixel position in accordance with the flow estimate (without actually changing the feature tensors as is done in warping) and leads to the estimation of costs for the residual flow, therefore enabling the residual refinement and estimation scheme. As the current flow estimate is generally a real-valued vector, we cannot index the cost volume directly and instead have to resort to sampling it, for which RAFT uses bilinear interpolation. An illustration of the lookup operator is given in Figure 3.2 (right).

[3]We will use the phrase *feature pixel* to refer to the position (or other properties) of a single feature element, e.g., $\mathbf{f}_{1,i,j}$. When it is evident from the context that we are referring to features and not source image pixels, we may further abbreviate that to *pixel*.

In the following, we will use function notation to represent access via (bilinear) sampling. We use $C^{(m)}_{i,j}(\mathbf{x})$ to denote the cost volume entry for source pixel $(i,j)^\top$ and target position $\mathbf{x} \in \mathbb{R}^2$ on level m. The full lookup operator can then be described via a sampling grid $\mathbf{X}$. In particular, we define a grid sampling operation via

$$C^{(m)}_{i,j,k}(\mathbf{X}) \coloneqq C^{(m)}_{i,j}\left(\mathbf{X}_{i,j,k}\right), \quad \mathbf{X} \in \mathbb{R}^{H_f \times W_f \times N_d \times 2}, \tag{3.3}$$

where $\mathbf{X}_{i,j,k} \in \mathbb{R}^2$ is a single sampling position of the grid $\mathbf{X}$. The index $k = 1, \dots, N_d$ corresponds to the respective displacement hypothesis $\mathbf{d}_k$. The sampling grid employed by RAFT can then be given as

$$\mathbf{X}^{(m,t)}_{i,j,k} = \frac{1}{2^m}\left(\mathbf{p}_{i,j} + \mathbf{u}^{(t-1)}_{i,j}\right) + \mathbf{d}_k, \tag{3.4}$$

now additionally depending on the timestep t and cost volume level m. In conclusion, by using this sampling grid, we obtain a cost volume slice $C^{(m)}(\mathbf{X}^{(m,t)})$ where $C^{(m)}_{i,j,k}(\mathbf{X}^{(m,t)})$ is the matching cost for pixel $\mathbf{p}_{i,j} = (i,j)^\top$ and displacement hypothesis $\mathbf{d}_k$ on level m at timestep t.

By default, displacement hypotheses $\mathbf{d}_k$ are arranged in a 9×9 grid, i.e., $\{\mathbf{d}_k\}_k = \{-4, -3, \dots, 4\}^2$. Note that Equation (3.4) does not scale the $\mathbf{d}_k$, due to which they are relative to the level m. This means that on coarser levels the displacements are larger in absolute terms than on finer levels, allowing RAFT to incorporate a broader range via the coarser levels and yet also the details from finer levels. Lookup is performed on all cost volume levels m individually (and in parallel) and the results are concatenated before they are passed on to the recurrent refinement network.

Recurrent Estimation and Refinement

The recurrent estimation and refinement network produces a flow update in each iteration, using the lookup operator to access the cost volume. In its core, this network consists of a gated recurrent unit (GRU) [CvMBB14] that has been modified to fit in the convolutional setting. Specifically, Teed and Deng [TD20] replace the linear layers usually found in GRUs with convolutional layers. Further, they propose to stack two GRU-blocks with 1×5 and 5×1 sized convolutions (respectively) for each recurrent refinement unit, allowing them to increase the receptive field without a quadratic increase in model size.

This inclusion of information from the neighborhood gives the network the ability to (at least in part) resolve occlusions and ambiguous matches. Note that both can be directly conveyed by the cost volume and, by extension, the slice returned from the lookup operator, as the respective cost values essentially represent a confidence measure: Equally high costs over all displacement hypotheses indicate that there is no good match, leading to the conclusion that the source pixel must likely be occluded (or its true target out of bounds in the second frame). Similarly, equally low costs over all hypotheses indicate that there are multiple equally good matches, i.e., ambiguities. In either case, the network can then use the neighborhood as well as additional contextual cues provided via a context encoder to perform regularization and provide a better flow estimate.

The aforementioned context encoder network, following the same structure as the feature encoders, is used to both initialize the hidden state $\mathbf{h}^{(0)}$ and provide contextual embeddings for this RNN by taking in the first image $\mathcal{I}_1$. The initial hidden state and context embeddings are produced as two separate outputs of this encoder. The context embeddings stay fixed over all iterations. In each iteration $t = 1, \dots, T$, the recurrent unit combines costs sampled via the lookup operator (across

all cost volume levels), the current flow estimate $\mathbf{u}^{(t-1)}$, and the context embedding to produce the updated hidden state $\mathbf{h}^{(t)}$ and, from this, the residual flow update $\mathbf{u}_{\text{res}}^{(t)}$ via a decoder network. The flow estimate is then updated via $\mathbf{u}^{(t)} = \mathbf{u}^{(t-1)} + \mathbf{u}_{\text{res}}^{(t)}$. Flow is generally initialized with $\mathbf{u}^{(0)} = 0$, however, when dealing with samples in a sequence (i.e., video), it may also be initialized with the result of the prior sample (referred to by Teed and Deng [TD20] as "warm start"). This, at least in theory, allows for incorporation of preexisting knowledge, as we generally assume that motion does not change abruptly between consecutive frames of a video due to the physical momentum of moving objects.

A particular trick employed by RAFT as part of the recurrent estimation process is the use of gradient blocking: Specifically, RAFT does not propagate gradients via the flow in recurrent updates. Flow is instead "detached" and handled individually in each iteration. The benefit of this gradient blocking strategy has been independently verified by Hofinger et al. [HBP+20] for coarse-to-fine pyramid networks. Resulting improvements can be ascribed to the noisy gradient of the bilinear interpolation operation with respect to the sampling coordinate [HBP+20; JST+19]. As the current flow estimate is used to compute the sampling positions, this noise would directly affect the gradient of the flow. Therefore, Hofinger et al. [HBP+20] propose to get rid of this noise by simply disregarding (i.e., blocking) propagation of that gradient, a scheme also followed by RAFT. We will briefly elaborate on this in Section 3.4.2. Note that this requires each intermediate flow estimate to be considered in the training loss, as there would otherwise be no supervision signal for some parts of the unrolled recurrent network. Alternatively, Jiang et al. [JST+19] propose a different and more complex sampling strategy that aims to avoid this problem, which (to our knowledge) has, however, not been tested with optical flow estimation yet.

Our last note in this subsection is an advantage of the recurrent approach: The number of iterations during training does not have to be the same as the number of iterations during evaluation. Further, additional iterations require space only for the automatic differentiation buffers. Hence, memory only imposes a limit on this number during training. When evaluating, i.e., when no gradient computation and back-propagation is required, the number of recurrent iterations is restricted only by the amount of computation time one is willing to spend. Teed and Deng [TD20] have shown that RAFT does not diverge with more iterations and also analyzed their impact (see their supplementary material).

3.1.2 Supervision

During training, upsampling is performed after each iteration and the loss is computed using the upsampled flow estimates $\mathbf{u}_{\text{up}}^{(t)}$ of each step $t = 1, \dots, T$. Specifically, RAFT uses an exponentially weighted sequence loss

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \gamma^{T-t} \sum_{(i,j)^\top \in \Omega} \frac{1}{|\Omega|} \left\| \mathbf{u}_{\text{gt},i,j} - \mathbf{u}_{\text{up},i,j,\theta}^{(t)} \right\|_k , \tag{3.5}$$

where $\mathbf{u}_{\text{gt}}$ is the ground-truth optical flow, γ is the exponential weight, θ are the model parameters, T the number of recurrent iterations, and Ω the set of all pixels $(i,j)^\top$ for which ground truth is available. The parameter k determines the type of the norm. By default, RAFT uses the L^1 norm, i.e., $k = 1$, and an exponential weight of $\gamma = 0.8$ for training on FlyingChairs and FlyingThings3D

and $\gamma = 0.85$ for training on Sintel and KITTI. This loss weights the final output with a factor of one and earlier estimates with an exponentially decreasing weight, going last to first, taking the mean over all (valid) pixels.

3.2 DICL: Displacement-Invariant Matching Cost Learning

The cost volume has become a central component of modern neural network based optical flow estimation methods, already starting with the seminal works by Dosovitskiy et al. [DFI+15] and their FlowNetC approach. Notable exceptions that forgo any sort of cost or correlation measure entirely, such as FlowNetS [DFI+15] and Perceiver-IO [JBA+22], are nowadays few and far between. As discussed in Section 2.1, many attempts at improving the cost volume via refinement have been made. "Displacement-invariant matching cost learning" (DICL) by Wang et al. [WZD+20], however, establishes a novel approach that completely forgoes any fixed correlation or cost measures and instead uses a learned neural network to compute matching costs by processing displacement hypotheses independently of each other. They have shown that, integrated into a classical coarse-to-fine setting, their cost learning module can significantly out-perform the standard dot product and cosine similarity [WZD+20]. Further, their proposed optical flow estimation method, as a whole, shows considerable improvements over its baseline methods VCN [YR19], PWC-Net [SYLK18], and SelFlow [LLKX19], especially on the more challenging Sintel "final" dataset (cf. Section 2.2.1).

This section provides an overview of the DICL method. We will begin in Section 3.2.1 by outlining its architecture, with a particular focus on the novel cost learning method employed by it. Thereafter, in Section 3.2.2, we will discuss the loss used by DICL. As we are mostly interested in cost learning, we refer the reader to the original paper and corresponding implementation by Wang et al. [WZD+20] for more details, such as the composition and kernel sizes of individual submodules, as well as investigations into the qualitative differences of cost functions with their approach.

3.2.1 Architecture

The full DICL method follows, in large, the common coarse-to-fine warping scheme: First, a Siamese feature encoder network f transforms input images $\mathcal{I}_1, \mathcal{I}_2 \in [0,1]^{H \times W \times 3}$ into feature pyramids $\{\mathbf{f}_1^{(m)}\}_m = f(\mathcal{I}_1)$ and $\{\mathbf{f}_2^{(m)}\}_m = f(\mathcal{I}_2)$, respectively, where m denotes the pyramid level. Levels range from $m = 2$ to $m = 6$, inclusively. Feature tensors $\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)} \in \mathbb{R}^{H/2^m \times W/2^m \times D_m}$ are reduced in both height and width by a factor of two per level, with D_m specifying the corresponding number of feature channels. Flow is estimated iteratively over the different levels, starting on the coarsest ($m = 6$) and finishing on the finest ($m = 2$). Network weights are not shared across levels. To account for the difference in resolution, flow is upsampled via bilinear interpolation in between levels. After the finest level, flow is upsampled again, this time to the full input scale, i.e., $m = 0$.

Each level can be divided into two major parts: The first part, shown in Figure 3.3, estimates the residual flow $\mathbf{u}_{\text{res}}^{(m)}$ given the upsampled flow estimate of the prior level, $\mathbf{u}_{\text{up}}^{(m+1)}$ (initialized to zero on the coarsest level). This is then used to update the total flow estimate $\mathbf{u}^{(m)} = \mathbf{u}_{\text{up}}^{(m+1)} + \mathbf{u}_{\text{res}}^{(m)}$. Hereby, the residual flow $\mathbf{u}_{\text{res}}^{(m)}$ represents the additional motion of the finer and more detailed level m that could not be represented and estimated on the coarser level $m + 1$ due to the lack of spatial resolution. As the coarser level already contains motion of larger and faster objects, estimation of

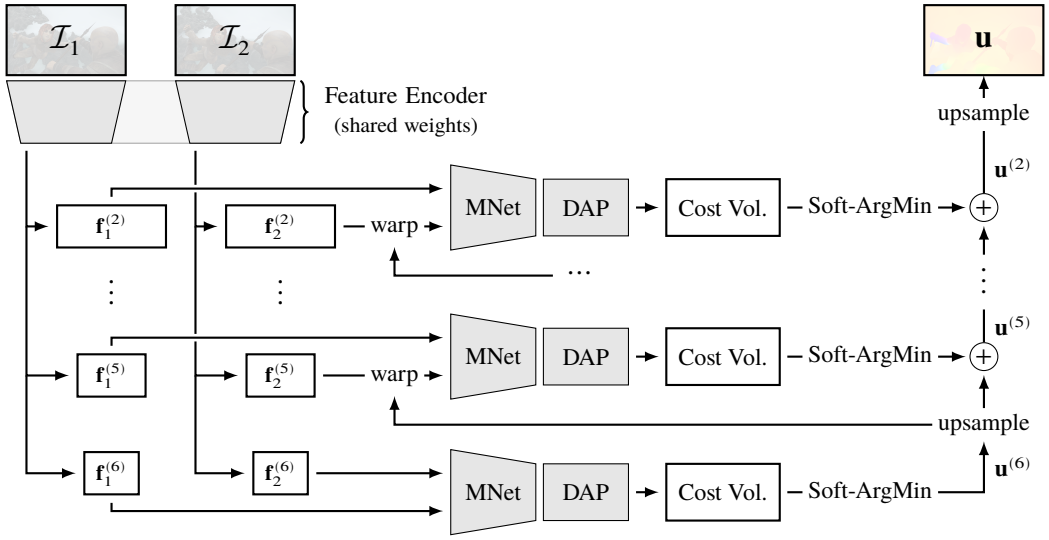

Figure 3.3: Overview of the DICL method. Input images $\mathcal{I}_1$ and $\mathcal{I}_2$ are transformed into feature pyramids $\mathbf{f}_1^{(m)}$ and $\mathbf{f}_2^{(m)}$, respectively, by tied feature encoders. Flow is produced iteratively in a coarse-to-fine manner, starting on the coarsest level (bottom, $m = 6$) and ending on the finest one (top, $m = 2$). In each level, the second frame features $\mathbf{f}_2^{(m)}$ are first warped back towards the first frame using the combined and upsampled flow estimate $\mathbf{u}^{(m-1)}$ of the coarser levels. Matching costs are computed from first and warped second frame features via a matching cost network (MNet) followed by a displacement-aware projection layer (DAP). A soft-argmin regression is used to decode the cost volume slice and produce the (residual) flow update. Learned neural network blocks are again indicated in gray.

the residual flow is generally restricted to a smaller range, reducing the amount of computation required. This, however, is a trade-off that creates difficulties in estimating small and fast-moving objects. In the second part of each level (omitted in Figure 3.3), the current total flow estimate is (optionally) refined via a network incorporating contextual information from the first-frame features $\mathbf{f}_1^{(m)}$ and the downsampled first image $\mathcal{I}_1$. As this refinement approach is of little relevance for this thesis, we refer to the original paper and corresponding implementation by Wang et al. [WZD+20] for more details thereon and will only discuss the first part in depth.

To estimate the residual flow $\mathbf{u}_{\text{res}}^{(m)}$, second-frame features $\mathbf{f}_2^{(m)}$ are first warped backwards using the upsampled total flow estimate $\mathbf{u}_{\text{up}}^{(m+1)}$ of the next coarser level to compensate for it. As the flow estimate is real-valued, we access the features by sampling via bilinear interpolation, which we again denote using function notation. Backward warping can therefore be expressed as

$$\tilde{\mathbf{f}}_{2,i,j}^{(m)} = \mathbf{f}_2^{(m)}\left(\mathbf{p}_{i,j} + \mathbf{u}_{\text{up},i,j}^{(m+1)}\right), \tag{3.6}$$

where $\tilde{\mathbf{f}}_2^{(m)}$ represents the warped features and $\mathbf{p}_{i,j} = (i,j)^\top$ a point in the feature domain of level m. Following the recommendations of Hofinger et al. [HBP+20] (and similar to RAFT, cf. Section 3.1.1), the gradient is blocked from propagating back across the upsampled flow to avoid any noise caused by the bilinear sampling operation with respect to it (see also Section 3.4.2). Features $\mathbf{f}_1^{(m)}$ of the first frame and warped features $\tilde{\mathbf{f}}_2^{(m)}$ of the second frame are then used to compute the matching costs $C_{i,j,k}^{(m)}$ for all source points $(i,j)^\top$ and a small set of displacement hypotheses $\{\mathbf{d}_k^{(m)}\}_k \subset \mathbb{Z}^2$.

The main contribution of DICL can be found in the novel way of computing these costs: Instead of using a fixed measure, such as subjecting the features to a cosine similarity score or dot product, Wang et al. [WZD+20] propose a learned network block, which we will refer to as the *DICL module* and discuss below in more detail.

Once matching costs have been computed, they are used to directly produce the residual flow estimate $\mathbf{u}_{\text{res},i,j}^{(m)}$ via a soft-argmin regression, i.e.,

$$\mathbf{u}_{\text{res},i,j}^{(m)} = \sum_{k=1}^{N_d} \varrho_{i,j,k}^{(m)} \cdot \mathbf{d}_k^{(m)} \tag{3.7}$$

with estimated match probabilities

$$\varrho_{i,j,k}^{(m)} = \sigma_1 \left(-\mathbf{C}_{i,j,\cdot}^{(m)}\right)_k, \tag{3.8}$$

computed via the standard softmax function σ_τ with temperature[4] τ defined as

$$\sigma_\tau(\mathbf{z})_k \coloneqq \frac{\exp(z_k/\tau)}{\sum_j \exp(z_j/\tau)}. \tag{3.9}$$

We use a dot in the displacement index field of the costs $\mathbf{C}_{i,j,\cdot}^{(m)}$ in Equation (3.8) to denote that we take all values over that dimension as a vector, input to the softmax function σ_τ. The displacement hypotheses $\mathbf{d}_k^{(m)} = \mathbf{p}_{k,l} - \mathbf{p}_{i,j}$ again represent the relative offset of target pixels $\mathbf{p}_{k,l} = (k,l)^\top$ in the second frame to source pixels $\mathbf{p}_{i,j} = (i,j)^\top$ in the first frame. Therefore, the soft-argmin regression given with Equation (3.7) is simply the sum of all displacement hypotheses weighted by their corresponding estimated match probability $\varrho_{i,j,k}^{(m)}$. These match probabilities, in turn, are computed directly by applying a softmax to the negative cost values, cf. Equation (3.8). Essentially, this means that we are treating the costs $C_{i,j,k}^{(m)}$ as negative log-probabilities (logits) for matches.

Cost Volume Learning

To generate the cost volume slices $C_{i,j,k}^{(m)}$, DICL uses a displacement invariant matching cost network (MNet) followed by a so-called displacement-aware projection (DAP) layer, both illustrated in Figure 3.4. The matching network computes preliminary costs from feature pairs in a way invariant to the actual displacement hypothesis $\mathbf{d}_k$, whereas the DAP reweighs these costs to incorporate displacement dependent information, such as biases, and to help focus the subsequent soft-argmin regression on the optimal match. To facilitate processing of feature pairs, the warped features $\tilde{\mathbf{f}}_2^{(m)}$ of the second frame are first offset in space by the respective displacement hypothesis $\mathbf{d}_k = (i_k, j_k)^\top \in \mathbb{Z}^2$ for which costs are computed, and then stacked on top of the first-frame features $\mathbf{f}_1^{(m)}$ along the feature dimension. We denote the offset features by $\hat{\mathbf{f}}_2^{(m)}$ with

$$\hat{\mathbf{f}}_{2,i,j,k}^{(m)} \coloneqq \tilde{\mathbf{f}}_{2,i+i_k,j+j_k}^{(m)}. \tag{3.10}$$

These stacks are then passed on to the matching network to compute the initial costs $\tilde{C}_{i,j,k}^{(m)}$. We can express this step as

$$\tilde{C}_{\cdot,\cdot,k}^{(m)} = \text{MNet}\left(\mathbf{f}_1^{(m)} \parallel \hat{\mathbf{f}}_{2,\cdot,\cdot,k}^{(m)}\right) \tag{3.11}$$

[4]The temperature $\tau \in \mathbb{R}^+$ can be seen as a parameter to change the base of the softmax from e to $e^{1/\tau}$, therefore controlling the hardness of the resulting distribution. The higher the temperature, the softer the distribution.

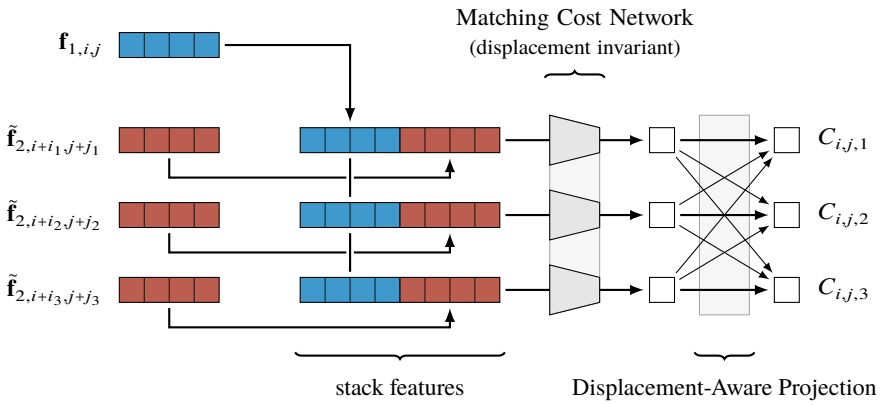

Figure 3.4: DICL cost learning module, encompassing the matching cost network (MNet) and displacement-aware projection (DAP). Given some source position $\mathbf{p}_{i,j} = (i,j)^\top$ and displacement hypothesis $\mathbf{d}_k = (i_k, j_k)^\top$, initial costs (pre-DAP) are computed by stacking first-frame features $\mathbf{f}_1$ at position $\mathbf{p}_{i,j}$ and warped second-frame features $\tilde{\mathbf{f}}_2$, offset at $\mathbf{p}_{i,j} + \mathbf{d}_k$, along the feature dimension before feeding them to the matching network. The matching network uses one instance per displacement hypothesis (sharing weights between all instances), but can incorporate information from other positions $\mathbf{p}_{q,r}$ in the local neighborhood of $\mathbf{p}_{i,j}$. Lastly, the initial costs are reweighed by the DAP layer to produce the final costs $C_{i,j,k}$ for position $\mathbf{p}_{i,j}$ and displacement $\mathbf{d}_k$.

Note that each displacement, indexed by k, is processed separately with the same network, i.e., the matching network shares the same weights for all displacement hypotheses, hence being displacement invariant. The dots replacing the indices i and j again indicate that we are retaining these dimensions, meaning the matching network is a 2D-convolutional network. In particular, given D_m as the number of feature channels for level m and under consideration of some fixed index k, we have $\mathbf{f}_1^{(m)}, \hat{\mathbf{f}}_{2,\cdot,\cdot,k}^{(m)} \in \mathbb{R}^{H/2^m \times W/2^m \times D_m}$. Using double bars to denote concatenation along the feature dimension, we obtain $(\mathbf{f}_1^{(m)} \parallel \hat{\mathbf{f}}_{2,\cdot,\cdot,k}^{(m)}) \in \mathbb{R}^{H/2^m \times W/2^m \times 2D_m}$ and, finally, $\tilde{C}^{(m)} \in \mathbb{R}^{H/2^m \times W/2^m \times N_d}$, where N_d represents the number of displacement hypotheses $\mathbf{d}_k$.

To compute the final costs $C^{(m)}$, the preliminary costs $\tilde{C}^{(m)}$ are recombined using the DAP. It can be understood as a linear layer over displacements k given some pixel $(i,j)^\top$, i.e.,

$$C_{i,j,k}^{(m)} = \sum_{l=1}^{N_d} w_{k,l} \cdot \tilde{C}_{i,j,l}^{(m)}, \tag{3.12}$$

where N_d is, again, the number of displacement hypotheses. Note that this can be implemented simply as a standard 1×1 sized 2D convolution without bias. This DAP layer is required as the combination of displacement invariant matching network and (fixed, i.e., non-learned) soft-argmin regression can consider neither biases nor correlation of different displacement hypotheses by itself. It further allows focusing the network on the optimal displacement hypothesis when the distribution provided by the matching network is multi-modal. See the paper by Wang et al. [WZD+20] for an analysis thereon. By default, the DAP layer is initialized as an identity mapping, i.e., $w_{k,l} = 1$ if $k = l$ and $w_{k,l} = 0$ otherwise.

To better understand why the DAP is beneficial, we need to have another look at the matching network, in particular the decision of processing displacements individually. The main reason behind this is cost savings in memory and compute, as processing the full set of stacked features via 4D convolutions is costly.[5] One could similarly decide to compute costs independent of position, however, using 2D convolutions across the spatial domain allows us to incorporate information from the neighborhood and with that produce better results when dealing, for example, with occluded pixels. The main problem now is that the subsequent soft-argmin regression, used to transform the costs into flow, cannot deal well with multi-modal distributions, whereas the matching network is prone to generate such as the displacements are handled independently. Further, being non-learned, the soft-argmin regression also cannot account for any directional biases which may, for example, be beneficial for automotive datasets where matches would generally be expected to follow the pattern of a forward driving direction.[6] The DAP is an attempt at alleviating these issues by allowing the network to recombine the initial costs in a learned way.

3.2.2 Supervision

The DICL loss is a weighted sum over individual losses for each level, which in turn are simply average endpoint errors according to some L^k-norm. Similar to RAFT's loss (cf. Section 3.1.2), it operates on the flow upsampled to the input resolution, $\mathbf{u}_{\text{up}}^{(m)}$. Upsampling is, again, performed via bilinear interpolation. The full loss is thus given as

$$\mathcal{L}(\theta) = \sum_{m \in \mathcal{M}} \alpha_m \sum_{(i,j)^\top \in \Omega^{(m)}} \frac{1}{|\Omega|} \left\| \mathbf{u}_{\text{gt},i,j} - \mathbf{u}_{\text{up},i,j,\theta}^{(m)} \right\|_k , \tag{3.13}$$

where

$$\Omega^{(m)} = \left\{ (i,j)^\top \in \Omega : \left| u_{\text{gt},i,j} \right| < r_u^{(m)} \wedge \left| v_{\text{gt},i,j} \right| < r_v^{(m)} \right\} \tag{3.14}$$

is an (optional) restriction of the pixels considered in the loss by limiting horizontal and vertical ground-truth flow $\mathbf{u}_{\text{gt}} = (u_{\text{gt}}, v_{\text{gt}})^\top$ by thresholds $r_u^{(m)}$ and $r_v^{(m)}$, respectively. Here, Ω is again the set of pixels $(i,j)^\top$ for which ground-truth flow is available. The weights α_m control the amount with which each level m contributes to the full loss, where $\mathcal{M} = \{2, 3, \dots, 6\}$ denotes the set of levels, ranging from $m = 2$ as the finest to $m = 6$ as the coarsest level. By default, the L^2 norm is used, i.e., $k = 2$. In the original DICL approach [WZD+20], this loss is applied to both flow directly produced by the soft-argmin regression and flow refined by help of the context network. To achieve this, the two flow outputs $\mathbf{u}_{\text{up}}^{(2m)}$ and $\mathbf{u}_{\text{up}}^{(2m+1)}$ produced per level m are weighted with individual weights α_{2m} and α_{2m+1} and restricted by corresponding thresholds. In other words, the set of levels is essentially doubled, with two elements per actual level.

[5] Note that, so far, we have taken the displacements as a set, without distinguishing between displacement directions. When processing the stacked features, however, we would need to distinguish between them, hence 4D convolutions being required.

[6] Although it is very much debatable whether we would want to incorporate such dataset dependent biases or have a method that performs well without them.

3.3 Combining RAFT and DICL

Having discussed the architectures of RAFT and DICL in the previous two sections, as well as other related methods in Section 2.1, we can now briefly revisit our motivation (stated previously in Section 1.2) and, finally, present our strategy for combining both techniques. We begin, however, in Section 3.3.1 by clarifying the potentially confusing notation regarding the denotation of cost volume and coarse-to-fine levels. After revisiting the motivation and objective in Section 3.3.2, we will derive our core concept for integrating DICL into RAFT. We start doing so in Section 3.3.3 by investigating a naive all-pairs approach. After concluding that this approach is infeasible, we will derive the basis of a feasible approach in Section 3.3.4 by reordering parts of RAFT and, finally, integrate DICL into it via generalization and relaxation in Section 3.3.5.

3.3.1 A Note on Level Nomenclature

In our discussions of RAFT (Section 3.1) and DICL (Section 3.2), we have used conflicting ways for denoting levels m. In both cases, m represents the base two exponent of the resolution divisor, giving the resolution of the level as $h/2^m \times w/2^m$ in relation to an anchor resolution $h \times w$. The important difference, however, is the definition of the anchor: For RAFT, we used m to denote the level of the cost volume, relative to its finest base level, whereas for DICL, we used m to denote the global level, relative to the input resolution. Those may not be and, in fact, are by default not the same.

In the remainder of this thesis, we will use the choice most appropriate to the current context: We use RAFT (local) notation whenever we deal with the RAFT cost volume, specifically the mathematics behind it, such as described later in this section. When in a model-centric context, such as when we discuss pyramid, coarse-to-fine, or other spatial levels in a more abstract way as part of a larger multiscale approach, we will use the global notation as this better relates to the actual resolution of the level. In particular, we will exclusively rely on the global notation during our evaluations in Part II.

3.3.2 Revisiting the Motivation and Objective

The main objective of this thesis is to evaluate whether we can improve the RAFT architecture by integrating the dynamic cost learning approach proposed within the DICL method. As we have already stated in Section 1.2, there are several indicators, such as Separable Flow by Zhang et al. [ZWPT21], hinting at potential qualitative improvements through such a combination. Via the insights we gained in the previous sections, we can now have a closer look at this: Our very first observation should be that *RAFT focuses on decoding the cost volume*, which is fixed during all iterative updates and (save for feature encoders) computed in a static way, whereas *DICL focuses on creating the cost volume* via a learned network and uses a static approach for decoding it. Therefore, we argue that both methods complement each other, and that combining them makes sense and is the next logical step on this abstract level.

However, a more important question to ask is whether this combination is sound on a technical level, i.e., whether the integration of DICL can provide additional inference capabilities beyond the ones that RAFT already possesses. For example, if DICL were only good at dealing with occlusions, we might argue that the benefits over plain RAFT would likely be limited, as the latter is (at least

in theory) already capable of using neighborhood context to handle occlusions via the learned recurrent decoder. In particular because information about occlusions and ambiguous matches can be extracted from the standard cost volume (similar high or similar low costs, respectively; cf. Section 3.1.1).

Hence, any extension to RAFT should provide further additions, such as a larger receptive field or a cost-accumulation strategy (cf. Separable Flow [ZWPT21]). We argue that DICL does so by, first, considering neighboring feature embeddings when estimating costs and, second, by generating costs in a learned way that can assign a different importance to feature channels based on the respective context and match candidate (i.e., feature pair). In contrast to this, the costs computed in RAFT are completely independent of each other given the embeddings. The decoder can only incorporate neighboring costs and features provided from the first frame via the context encoder but not features of the second frame, and it also cannot consider these while computing costs. Lastly, the dot product assumes that all feature channels are equally important, which may not necessarily be the case in all situations. We thus believe that an integration of the DICL module has the potential to benefit RAFT, especially on complex datasets with difficult matches, similar to the observations Wang et al. [WZD+20] made with coarse-to-fine methods.

3.3.3 An All-Pairs Approach

The conceptually simplest approach at combining RAFT and DICL can be obtained by computing the finest all-pairs cost volume level of RAFT via the DICL network. Thereafter, this level can be downsampled to create the coarser levels and, with that, the complete hierarchical cost volume, which can then be accessed via the lookup operator, just like in the original RAFT method. Unfortunately for us, however, this is not feasible due to memory requirements: Let us simplify our problem by assuming that we only need memory for storing the feature pairs $(\mathbf{f}_{1,i,j}, \mathbf{f}_{2,k,l})$. We can then calculate the number of floating-point values n by multiplying the number of match candidates (which, in our case, is the number of all feature pairs) with the number of feature channels per pair. Together, this yields

$$n = \left(H_f \cdot W_f\right)^2 \cdot 2D, \tag{3.15}$$

where H_f and W_f denote height and width in feature pixels and D is the number of feature channels.

We can now make further assumptions to obtain a rough estimate. Following DICL for the number of feature channels, we set $D = 32$. Following RAFT for the feature resolution, we set $H_f = {}^{H}/_{8}$ and $W_f = {}^{W}/_{8}$, where $W \times H$ is the original input resolution. As input, we assume the Sintel dataset, which, with appropriate padding in height, gives us a resolution of $W \times H = 1024 \times 440$. Plugging this into Equation (3.15) yields a memory requirement of approximately 12 GiB when using 32 bit precision. Note that this estimate is, first, only for a single input sample and, second, only for storing feature pairs. We would, at the very least, need additional memory during training for the activation buffers needed for back-propagation.

Hence, even though there are GPUs available with that amount of memory today, training will very likely not be possible at all. This approach would be infeasible even if memory would suffice, as we will not be able to obtain a good GPU compute utilization, therefore costing us computation time. Further, note that the costs are quadratically dependent on the input resolution: Repeating the calculation above with a resolution of 1920×1080 results in a requirement of approximately 250 GiB, which is not reasonably manageable by current standards. The all-pairs approach is only

feasible in RAFT because it relies on the dot product, due to which the matching cost computation of all feature pairs can be implemented efficiently via a simple matrix multiplication. This is not possible for DICL-based costs, which processes feature pairs via a learned network. We thus need a better strategy.

3.3.4 Towards a Feasible Approach

The approach we propose in this thesis is founded on an observation already made by Teed and Deng [TD20]: RAFT does not actually require matching costs for all feature pairs, but only the ones accessed via the lookup operator. Therefore, if we can find a method to compute only these costs on demand, we can avoid the quadratic memory dependency on the input size. We aim to do that in this subsection. In particular, we will derive a method to compute costs on demand and in a way that is easily extendable to arbitrary neural networks as cost or correlation function while still producing mathematically equivalent results when used with the dot product. This method will do so without the need for computing the costs of all feature pairs and without the need for pooling of costs, as used in the standard method to create the full multi-level cost volume (cf. Section 3.2.1). Costs for any level can therefore be computed independently of the ones for other levels. For this subsection, we will rely exclusively on inner products as correlation functions, whereas generalization to arbitrary functions will be discussed in the next subsection.

Before we start to derive our method, we briefly need to recap the standard method for generating and accessing the cost volume, more specifically its three main operations: Cost computation on the finest level ($m = 0$) via the dot product to obtain $C^{(0)}_{i,j,k,l}$ (cf. Equation (3.1)), followed by average-pooling in domain of the second frame ($\mathcal{I}_2$) to create the coarser levels $C^{(m)}_{i,j,k,l}$ with $m > 0$ (cf. Equation (3.2)), and, finally, sampling via bilinear interpolation to allow for access via real-valued position vectors (cf. Equations (3.3) and (3.4)). For simplicity, we will derive the method below for an arbitrary inner product $\langle \cdot , \cdot \rangle : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. Note that this includes the scaled dot product used in RAFT, as we can simply define

$$\left\langle \mathbf{f}_{1,i,j} , \mathbf{f}_{2,k,l} \right\rangle \coloneqq \frac{1}{\sqrt{D}} \left(\mathbf{f}_{1,i,j} \right)^\top \left(\mathbf{f}_{2,k,l} \right) , \tag{3.16}$$

while still retaining all properties of an inner product, in particular linearity in both arguments for real values, which we will exploit below.

Step 1: Feature Pooling

For the first step, we follow an optimization proposed by Teed and Deng [TD20]: They have shown that, due to the linearity of the inner product, first computing the cost $C^{(0)}_{i,j,k,l}$ at the finest level and then pooling to obtain $C^{(m)}_{i,j,k,l}$ with $m > 0$ is equivalent to pooling the second frame features $\mathbf{f}_2$ to obtain $\mathbf{f}^{(m)}_2$ and then computing costs $C^{(m)}_{i,j,k,l}$ independently for all levels m using those. We formulate this as the following theorem:

Theorem 3.3.1
When computing the multi-level cost volume, pooling over the domain of the second frame and the inner product are interchangeable. Specifically, the following approaches are equivalent (in terms of results):

- *First compute costs on the finest level using feature vectors, then pool costs in domain of the second frame to obtain coarser cost levels.*
- *First pool the second frame features to obtain coarser feature levels, then compute costs from those and non-pooled first frame features.*

Proof As shown by Teed and Deng [TD20]. Let $\mathbf{f}_1, \mathbf{f}_2 \in \mathbb{R}^{H_f \times W_f \times D}$ be the feature tensors of the first and second frame, respectively, with $\mathbf{f}_{1,i,j}, \mathbf{f}_{2,k,l} \in \mathbb{R}^D$ being the corresponding feature vectors and i, j, k, l the element indices. Further, let $\mathcal{C} = \{C^{(0)}, C^{(1)}, \dots, C^{(M)}\}$ denote the multi-level cost volume with $C^{(m)} \in \mathbb{R}^{H_f \times W_f \times (H_f/2^m) \times (W_f/2^m)}$ representing a single level m, where $m = 0$ denotes the finest level. The specific cost value for level m at location (i, j) in the first frame and location (k, l) in the second frame is then given by $C^{(m)}_{i,j,k,l}$. Due to linearity of the inner product, we obtain

$$C^{(m)}_{i,j,k,l} = \frac{1}{2^{2m}} \sum_{p=0}^{2^m-1} \sum_{q=0}^{2^m-1} \underbrace{\left\langle \mathbf{f}_{1,i,j} \, , \, \mathbf{f}_{2,2^m k+p,2^m l+q} \right\rangle}_{= C^{(0)}_{i,j,2^m k+p,2^m l+q}} \tag{3.17}$$

$$= \left\langle \mathbf{f}_{1,i,j} \, , \, \underbrace{\frac{1}{2^{2m}} \left(\sum_{p=0}^{2^m-1} \sum_{q=0}^{2^m-1} \mathbf{f}_{2,2^m k+p,2^m l+q} \right)}_{=: \mathbf{f}^{(m)}_{2,k,l}} \right\rangle, \tag{3.18}$$

where Equation (3.17) represents the first and Equation (3.18) represents the second approach. Therefore, both approaches yield the same result. ■

The implications of this are that we do not need to store the full all-pairs cost volume. Instead, we can store the pooled features $\mathbf{f}^{(m)}_2$ and use those to compute costs both on demand and (given these features) independently for each level. This leads to an updated order of steps: First we pool the $\mathbf{f}_2$ features, then compute the costs we need, and, finally, use bilinear interpolation to allow for real-valued sampling positions. We are therefore swapping operations one and two of the original strategy. Note that, for the sampling step, we now only need to compute costs for the four neighboring integer-valued positions of each sampling point, hence avoiding quadratic dependency on the input size. While Teed and Deng [TD20] have implemented this optimization for the dot product via a custom CUDA kernel, both computing the required costs and performing interpolation/sampling as one operation (without the need for storing feature pairs explicitly), it is somewhat more complex to adapt this to general networks (where we do need to store feature pairs to process them), thus leading to our second step.

Step 2: Feature Sampling

We can now further show that first computing the costs and then sampling from the computed costs via bilinear interpolation is equivalent to bilinearly sampling from the $\mathbf{f}_2$ features and then computing the costs with the sampled features:

Theorem 3.3.2
To compute bilinearly sampled costs, we can either sample costs from the cost volume level pooled in the domain of the second frame, or, alternatively, sample feature vectors from the pooled feature representation of the second frame and then use those in the inner product. Both approaches produce equal results.

Proof As above, let $\mathbf{f}_1, \mathbf{f}_2 \in \mathbb{R}^{H_f \times W_f \times D}$ be the feature tensors of the first and second frame, respectively. Given element indices i, j, k, l, let $\mathbf{f}_{1,i,j}, \mathbf{f}_{2,k,l} \in \mathbb{R}^D$ be the corresponding feature vectors and $\mathbf{f}_2^{(m)} \in \mathbb{R}^{(H_f/2^m) \times (W_f/2^m) \times D}$ be the pooled second frame features as defined in Equation (3.18) for some level m. Further, we again denote the cost volume entry of that level at location (i,j) in the first and location (k,l) in the second frame by $C^{(m)}_{i,j,k,l}$. We use function notation as a shorthand for bilinear sampling. In particular, for some discrete location (i,j) in the first frame and some real-valued position $\mathbf{x} = (x_1, x_2)^\top$ in the second frame, we let $C^{(m)}_{i,j}(\mathbf{x})$ be the cost volume taken at that first location and sampled at the second one.

Writing this down explicitly yields Equation (3.19) with interpolation weights $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in \mathbb{R}_0^+$, which we can then reformulate via

$$
\begin{aligned}
C^{(m)}_{i,j}(\mathbf{x}) &= \alpha_1 C^{(m)}_{i,j,\lfloor x_1 \rfloor,\lfloor x_2 \rfloor} + \alpha_2 C^{(m)}_{i,j,\lfloor x_1 \rfloor,\lfloor x_2 \rfloor+1} + \alpha_3 C^{(m)}_{i,j,\lfloor x_1 \rfloor+1,\lfloor x_2 \rfloor} + \alpha_4 C^{(m)}_{i,j,\lfloor x_1 \rfloor+1,\lfloor x_2 \rfloor+1} && (3.19)\\
&= \alpha_1 \left\langle \mathbf{f}_{1,i,j}\,,\, \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor,\lfloor x_2 \rfloor} \right\rangle + \alpha_2 \left\langle \mathbf{f}_{1,i,j}\,,\, \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor,\lfloor x_2 \rfloor+1} \right\rangle && (3.20)\\
&\quad + \alpha_3 \left\langle \mathbf{f}_{1,i,j}\,,\, \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor+1,\lfloor x_2 \rfloor} \right\rangle + \alpha_4 \left\langle \mathbf{f}_{1,i,j}\,,\, \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor+1,\lfloor x_2 \rfloor+1} \right\rangle \\
&= \Big\langle \mathbf{f}_{1,i,j}\,,\, \underbrace{\alpha_1 \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor,\lfloor x_2 \rfloor} + \alpha_2 \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor,\lfloor x_2 \rfloor+1} + \alpha_3 \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor+1,\lfloor x_2 \rfloor} + \alpha_4 \mathbf{f}^{(m)}_{2,\lfloor x_1 \rfloor+1,\lfloor x_2 \rfloor+1}}_{= \mathbf{f}_2^{(m)}(\mathbf{x})} \Big\rangle. && (3.21)
\end{aligned}
$$

Equation (3.19) is expanded to Equation (3.20) using Equation (3.18) from the previous theorem. Bilinearity of the inner product yields Equation (3.21). Sampling from the pooled costs, as given by Equation (3.19), therefore produces the same result as computing costs from the pooled and sampled $\mathbf{f}_2$ features, as stated by Equation (3.21). ■

This leads to the final order of operations: First, we pool the $\mathbf{f}_2$ features to obtain $\mathbf{f}_2^{(m)}$, just as in step one. Then, we sample from the pooled features $\mathbf{f}_2^{(m)}$ via bilinear interpolation, and, finally, compute the costs from these sampled features. In summary, we swap operations two and three from the previous order, proposed in step one.

RAFT Reordered

Our final approach of this subsection is the direct result of step two and Equation (3.21). In particular, we compute the sampled costs $C^{(m)}_{i,j}(\mathbf{x})$ for some source location $(i,j)^\top$ and some target position $\mathbf{x}$ on level m via

$$
C^{(m)}_{i,j}(\mathbf{x}) = \left\langle \mathbf{f}_{1,i,j}\,,\, \mathbf{f}^{(m)}_2(\mathbf{x}) \right\rangle, \tag{3.22}
$$

where

$$
\mathbf{f}^{(m)}_{2,k,l} = \frac{1}{2^{2m}} \sum_{p=0}^{2^m-1} \sum_{q=0}^{2^m-1} \mathbf{f}_{2,2^m k+p,2^m l+q} \tag{3.23}
$$

represent the pooled and to-be-sampled second-frame features. As above, $\mathbf{f}_{1,i,j}$ and $\mathbf{f}_{2,i,j}$ denote the standard first- and second-frame features, respectively. Note that we can use Equations (3.22) and (3.23) as a direct replacement for Equations (3.1) and (3.2) by plugging them into Equation (3.3). The full sampling grid remains the one given by Equation (3.4). Further, note that this is still mathematically equivalent to RAFT and only represents a reordering of operations. In contrast to the original implementation, however, we can now simply take the first-frame features $\mathbf{f}_{1,i,j}$ and

sample from the pooled second-frame features $\mathbf{f}_2^{(m)}$ at some position $\mathbf{x}$ to directly compute the corresponding costs $C_{i,j}^{(m)}(\mathbf{x})$ on demand and independently for each level. This thus allows for easy extension and generalization, which we will discuss in the next subsection.

Revisiting Memory Requirements

Before generalization, however, we will briefly revisit the (highly simplified) memory requirements for our updated strategy. By allowing for costs to be computed on demand, we have traded the quadratic dependency on the feature resolution $W_f \times H_f$ against a linear dependency thereon in addition to further linear dependencies on the number of displacement hypotheses N_d and the number of levels N_m in the multi-level cost volume. As a result of this, we can now give the number of floating-point values needed for storing all required feature pairs $(\mathbf{f}_{1,i,j}, \mathbf{f}_2^{(m)}(\mathbf{x}))$ as

$$n = H_f \cdot W_f \cdot N_d \cdot N_m \cdot 2D. \tag{3.24}$$

In contrast to the all-pairs memory requirements, given in Equation (3.15), this now describes the requirements for a single iteration. During evaluation, we can simply reuse the same buffers in each iteration, however, during training, we need to store network activations for back propagation, which we can approximate by multiplying Equation (3.24) by the number of training iterations.

Following the same assumptions as in Section 3.3.3 (in particular using $D = 32$ feature channels as well as $H_f = {}^{H}/_{8}$ and $W_f = {}^{W}/_{8}$) and setting $N_d = 9 \times 9$ and $N_m = 4$ based on RAFT, we can again compute a rough memory estimate. For the Sintel dataset with $W \times H = 1024 \times 440$ (again using appropriate padding), we now require approximately 557 MiB per iteration or 6.5 GiB for 12 training iterations when using 32 bit precision (cf. approximately 12 GiB for the all-pairs approach). Using a larger resolution of 1920×1080 now results in 2.5 GiB per iteration (cf. approximately 250 GiB for the all-pairs approach).

This is still considerably large (especially when compared to the costs of original RAFT, in which explicit storing of the feature pairs is not required, therefore foregoing the factor $2D$), but we can now trade off memory requirements against model capabilities by adjusting the number of displacement hypotheses N_d as well as the number of levels N_m of the multi-level cost volume. Further, we can use lower resolutions for training and process higher resolutions when evaluating with the same overall memory budget, as the latter can reuse the buffers in each iteration.

3.3.5 Combining RAFT and DICL via Generalization

We can now build our integration of DICL into RAFT by generalizing the reordered RAFT approach we derived in the previous subsection and Equation (3.22) in particular. Our method can be obtained via three successive modifications: First we relax the requirement for our cost function from an inner product to arbitrary (but differentiable) functions $c : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, yielding

$$C_{i,j}^{(m)}(\mathbf{x}) = c\left(\mathbf{f}_{1,i,j}, \mathbf{f}_2^{(m)}(\mathbf{x})\right). \tag{3.25}$$

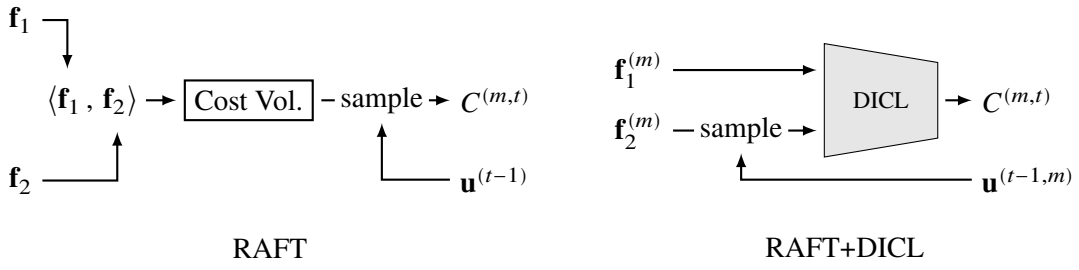

Figure 3.5: Cost computation and sampling as used in the standard RAFT implementation (left) and its replacement in our RAFT+DICL approach (right). RAFT first computes costs $C^{(0)}$ for all feature pairs on the finest level, then pools them to create the cost volume levels $C^{(m)}$, and finally samples from those by incorporating the current flow estimate $\mathbf{u}^{(t-1)}$ for some timestep t to obtain the cost volume slice $C^{(m,t)}$. RAFT+DICL, in contrast, first creates feature representations $\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)}$ for each level m, then samples from the second-frame features $\mathbf{f}_2^{(m)}$ by incorporating the current flow estimate $\mathbf{u}^{(t-1,m)}$ for level m and timestep t, and finally computes the cost volume slice $C^{(m,t)}$ via the DICL matching network and displacement-aware projection.

Note again that cost and correlation are inverse concepts and that we can use either for the function c. Equation (3.25) now allows c to be represented by a neural network, however, costs/correlation-scores are still computed individually for each match candidate, without considering context from the neighborhood. Generalizing c to enable this is our second step, yielding

$$C^{(m)}(\mathbf{X}) = c\left(\mathbf{f}_1, \mathbf{f}_2^{(m)}(\mathbf{X})\right) \tag{3.26}$$

with $c : \mathbb{R}^{H_f \times W_f \times D} \times \mathbb{R}^{H_f \times W_f \times N_d \times D} \to \mathbb{R}^{H_f \times W_f \times N_d}$, using again a sampling grid $\mathbf{X}$ as, for example, given in Equation (3.4). Equation (3.26) now computes the full cost volume slice for level m and is therefore no longer restricted to a single feature pair. Note that this already allows for the integration of the DICL module (i.e., DICL matching network and displacement-aware projection layer). Our third and last step is a relaxation of the inputs:

$$C^{(m)}(\mathbf{X}) = c\left(\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)}(\mathbf{X})\right). \tag{3.27}$$

In particular, we generalize to different feature encodings $\mathbf{f}_1^{(m)}$ and $\mathbf{f}_2^{(m)}$ for different levels m, i.e., allow for true feature encoder stacks instead of feature pooling. Note, however, that the resolution of the first-frame features and the sampling grid still defines the output resolution, which needs to be equal over all levels m for the standard RAFT setting. The resolution for the second frame features can, in contrast, be arbitrary on each level (due to sampling). We will later relax this condition as well by (re)arranging RAFT in a coarse-to-fine approach.

In summary, Equation (3.27) is a generalization of the RAFT reordered lookup operator, as illustrated in Figure 3.5. We can obtain the original (reordered) lookup operator as a special case where c is a broadcast version of the scaled dot product, applied to each feature pair individually, and the features are the finest level first-frame features and average-pooled second-frame features. In contrast, however, our generalized approach allows for direct processing of the feature pairs associated with each accessed match candidate via an (at least in theory) arbitrary convolutional neural network and is not restricted to the DICL matching network. This also enables a wide variety

of modifications, as we could now easily return arbitrary vectors from the network c instead of costs for each displacement hypothesis. The RAFT recurrent network already expects a cost vector flattened over all displacement hypotheses, therefore little adaption of this network is needed, with the modifications essentially only representing a change in the number of channels fed to it.

3.4 Common Principles

Before we discuss our experiments and approaches in more detail, we will briefly look at common principles used throughout them. In particular, we will discuss the differences between warping and sampling, including issues related to warping, in Section 3.4.1 as well as gradient blocking in Section 3.4.2. While both have already been extensively discussed by Hofinger et al. [HBP+20], to which we refer for a more detailed investigation thereon, we believe that an inclusion in this thesis is warranted for completeness.

3.4.1 Comparing Warping and Sampling

Both RAFT and DICL iteratively estimate residual flow to update a total flow estimate. Their approach and, in particular, the core technique used to enable this residual estimation, however, is significantly different: DICL, similar to many other coarse-to-fine approaches, uses backward warping whereas RAFT relies on a less common sampling method. Hofinger et al. [HBP+20] have shown that, at least in the context of coarse-to-fine models, the choice between these two techniques matters and has a considerable impact on performance, with sampling leading to better results. We reason that their arguments (on which we will expand below) can be extended to any residual estimation attempt, as evidenced by Teed and Deng [TD20] and their findings that RAFT performs significantly worse when replacing the sampling strategy with a warping-based approach.

To better understand where this difference in performance stems from, we apply both techniques to obtain the second-frame features $\mathbf{f}_{2,i,j,k}$ for some source location $\mathbf{p}_{i,j} = (i,j)^\top$, displacement hypothesis $\mathbf{d}_k = (i_k,j_k)^\top$, and current flow estimate $\mathbf{u}_{i,j}$ from the original features $\mathbf{f}_{2,i,j}$. For sampling, this yields

$$\mathbf{f}_{2,i,j,k} = \mathbf{f}_2\left(\mathbf{p}_{i,j} + \mathbf{u}_{i,j} + \mathbf{d}_k\right), \tag{3.28}$$

whereas, for the backward-warping-based approach, this yields

$$\tilde{\mathbf{f}}_{2,i,j} = \mathbf{f}_2\left(\mathbf{p}_{i,j} + \mathbf{u}_{i,j}\right), \tag{3.29}$$

$$\mathbf{f}_{2,i,j,k} = \tilde{\mathbf{f}}_{2,i+i_k,j+j_k}. \tag{3.30}$$

We again use function notation to indicate bilinear sampling. Note that warping also relies on sampling and that, therefore, the algorithmic difference between both techniques is solely the order of operations: The sampling approach performs the sampling operation as the last step, using the full position derived from the current flow estimate $\mathbf{u}_{i,j}$ and displacement hypothesis $\mathbf{d}_k$. The warping-based approach, in contrast, first creates a warped set of features $\tilde{\mathbf{f}}_2$ having the same resolution as the original $\mathbf{f}_2$ features via sampling by incorporating the current flow estimate $\mathbf{u}_{i,j}$. This represents the actual warping step. Only thereafter is the respective offset for the displacement hypothesis $\mathbf{d}_k$ applied to obtain the final result from said warped intermediary tensor $\tilde{\mathbf{f}}_2$.

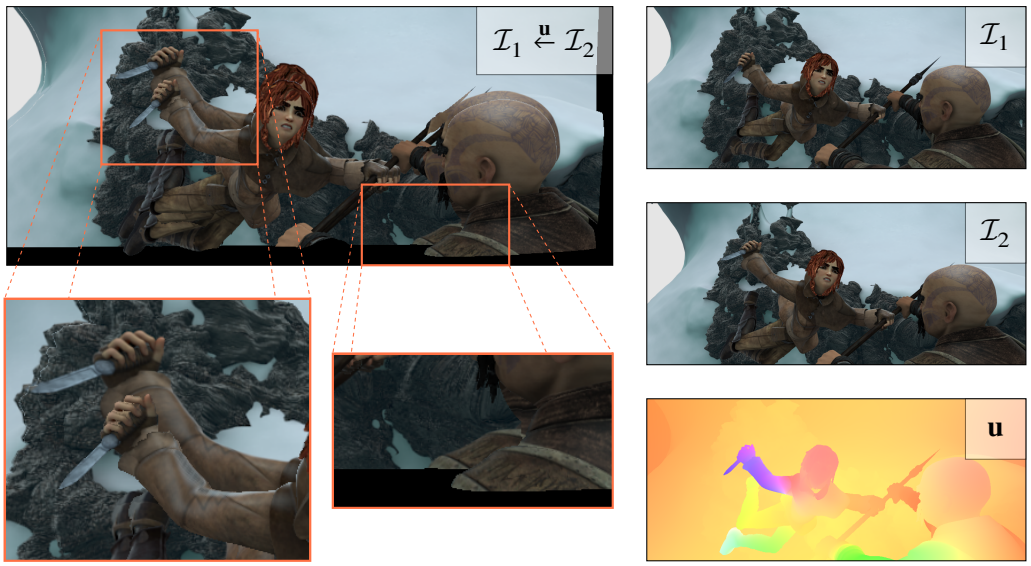

Figure 3.6: Illustration of artifacts created as a result of backward warping the second image $\mathcal{I}_2$ towards the first image $\mathcal{I}_1$ using optical flow $\mathbf{u}$. Top-left: Warped image $\mathcal{I}_1 \leftarrow \mathcal{I}_2$. Bottom-left: Excerpt of the warped image, showing ghosting. Bottom-center: Excerpt of the warped image, showing loss of information (black) at the image borders. Right: Input images $\mathcal{I}_1$ and $\mathcal{I}_2$ as well as optical flow $\mathbf{u}$ used for warping (from top to bottom).

It is precisely this order that distinguishes between a destructive operation, given by warping, and a non-destructive one, given by sampling. Being a destructive operation, warping results in loss of information and unwanted artifacts, some of which are illustrated in Figure 3.6. In particular, warping can lead to *ghosting*, which expresses itself as duplicate feature areas stemming from a fast-moving foreground and a slow-moving background, i.e., two or more flow vectors pointing to the same source position in the second-frame features (cf. Figure 3.6 bottom left). As the original features $\mathbf{f}_{2,i,j}$, however, can only show foreground and not background objects, both target locations retain the feature descriptor from the same foreground object. Whereas, in reality, the target location associated with the background flow should obtain the background feature descriptor. In the worst case, this duplication can lead to false ambiguous matches being encoded in the cost volume.

Loss of information, on the other hand, can occur when there exists any pixel to which no flow points, or when flow points outside the frame, i.e., to a place where no information is available (cf. Figure 3.6 bottom center). Note that the latter can be seen as a cause for the former, as the resolution is maintained across warping and thus, for any flow pointing outside the frame, there is a corresponding element missing that would need to point inside the frame to retain all information. Similarly, shrinking and expansion of areas can also be seen as a cause for this. Lastly, loss of information is also likely to occur when using warping to account for residual flow across different scales, such as in coarse-to-fine models. In this case, flow from the coarser level cannot represent fine details due to which, when using it as basis for warping, these may get erased by moving a larger object that dominates the flow on top of them. An illustration for this can be found in the paper by Hofinger et al. [HBP+20].

A final problem of warping is that neighboring pixels in the first frame share large parts of their search window for matches in the warped second frame, whereas, with sampling, there is no such restriction and individual pixels can have completely individual and disjoint search windows. This, however, also constitutes the major downside of the sampling approach: Due to the order of operation that allows this independence, more points need to be sampled (in particular one point per displacement hypothesis and location instead of just one point per location). This results in higher compute and memory requirements when compared to warping. As we already need to store the feature pairs for each match candidate explicitly, making the additional requirements less severe, and due to the findings by Teed and Deng [TD20] in regard to RAFT, we will exclusively rely on sampling for our methods.

3.4.2 Gradient Blocking

A technique used by both RAFT and DICL is blocking the gradient of the flow across different estimation iterations. Specifically, both methods allow the gradient to pass only through the residual flow update $\mathbf{u}_{\text{res}}^{(t)}$ for some iteration (in case of RAFT, or level in case of DICL) t, but not across the previous full flow estimate $\mathbf{u}^{(t-1)}$. This is because bilinear interpolation, intrinsic to both warping and sampling-based residual estimation approaches, has a noisy gradient in regard to its positional argument, which is derived from the (previous) full flow estimate $\mathbf{u}^{(t-1)}$ at step t. Therefore, allowing the gradient to pass backwards from this sampling operation to the flow $\mathbf{u}^{(t-1)}$ would result in a noisy flow gradient, which, again, Hofinger et al. [HBP+20] have shown to be detrimental to performance in a coarse-to-fine setting. However, a minor drawback of this is that, for many methods, the full flow estimate provides the only connection across iterations. Hence, blocking the gradient from propagating across the flow removes any connection of the gradient between iterations and, as a result, individual supervision of each iteration is needed.

The existence of this noise can be attributed to the sudden switching of basis points. This is best illustrated for the one-dimensional case, i.e., linear interpolation [HBP+20][7]: Given a set of sampled basis points $\{f_i\}_i$ with $i \in \mathbb{Z}$ and some position $\mathbf{x} \in \mathbb{R}$ we can write linear interpolation as

$$f(x) = (1 - \alpha) \cdot f_i + \alpha \cdot f_j \tag{3.31}$$

with interpolation weight

$$\alpha = \frac{x - i}{j - i}, \tag{3.32}$$

where $i \leq x \leq j$, for example $i = \lfloor x \rfloor$ and $j = \lfloor x \rfloor + 1$. We can then give the gradient as

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) = \frac{f_j - f_i}{j - i}. \tag{3.33}$$

[7] We deviate from the simplified formulation provided by Hofinger et al. [HBP+20] as we believe there is a certain beauty in the slightly more complex version, explicitly accounting for the step width $j - i$. In particular, this highlights that the gradient of the bilinear sampling operation is a direct approximation to the original underlying function g from which the points f_i have been sampled: Compare Equation (3.33) with the standard definition of the derivative

$$\frac{\mathrm{d}g}{\mathrm{d}x}(x) = \lim_{h \to 0} \frac{g(x + h) - g(x)}{h}.$$

Therefore, letting $(j - i) \to 0$ produces the original gradient. Further, the step width directly relates to the quality of approximation. Just like we are approximating the original function g via a piecewise linear function f, we are also approximating the original gradient g' via a piecewise constant function g'.

Note that the gradient is discontinuous and, more specifically, piecewise constant. Therefore, crossing over from one basis point to another causes a sudden jump in the gradient value. As bilinear interpolation is simply linear interpolation twice applied, an equal observation can be made for the two-dimensional case. There, the gradient is piecewise constant in the dimension of derivation and piecewise linear in the other dimension. Similar to warping and sampling, we refer to Hofinger et al. [HBP+20] for a more detailed investigation thereon. We will, due to their results and the usage of gradient blocking in both RAFT and DICL, also make use of this technique in all methods presented in this thesis.

Part II

Experiments

4 Evaluated Methods

Chapters 1 to 3 have established the methodological foundations of our approaches, giving us the necessary means to now construct complete optical flow estimation methods, implementing both our core RAFT+DICL idea derived in Section 3.3 and the principles discussed in Section 3.4. Specifically, we propose three approaches: First, a single-level method, presented in Section 4.1, giving us a simplified model to gauge feasibility of our overall approach. Second, a multi-level method, discussed in Section 4.2, following the multiscale approach taken by RAFT [TD20] (using a hierarchical cost volume), intended to extend the receptive field of the (in this regard) rather limited single-level model. And third, a coarse-to-fine method, given in Section 4.3, which attempts to resolve the shortcomings and limitations of our multi-level approach, mainly resulting from its considerable memory requirements.

As the respective model architecture is only one part of a successful method, Section 4.4 follows up with a generalization of the sequence loss used by RAFT [TD20]. Thereafter, Sections 4.5 and 4.6, respectively, discuss the training and evaluation strategies employed throughout the remainder of this thesis. We conclude this chapter by detailing our next steps, providing the rationale and plan behind the evaluations performed in Chapters 5 and 6.

4.1 RAFT+DICL on a Single Level

We start with a simplified approach to gauge the feasibility and viability of our key concept, derived in Section 3.3. In particular, we start with a single-level model, using only the finest cost volume level. Apart from this simplification, this approach is a fairly direct integration of DICL into RAFT. While removal of the coarser cost volume levels is expected to negatively impact performance on larger motions, due to the missing capacity of the model for these, performance should still be good on smaller displacements. In consideration of the heightened memory requirements of multi-level methods (as well as additional design choices that need to be made therefore), we believe that this single-level model should be able to pose as a good baseline for our subsequent experiments.

4.1.1 Architecture

The overall framework of our single-level method is based on RAFT (cf. Section 3.1). In particular, we use the RAFT feature encoder f to create the (single-level) features $\mathbf{f}_1^{(3)} = f(\mathcal{I}_1)$ and $\mathbf{f}_2^{(3)} = f(\mathcal{I}_2)$ from input images $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. The features $\mathbf{f}_1^{(3)}, \mathbf{f}_2^{(3)} \in \mathbb{R}^{H/8 \times W/8 \times D}$ have $D = 32$ feature channels. To reduce the number of feature channels in comparison to the $D = 256$ of RAFT, we modify only the 1×1 output convolution that is already present in this encoder.

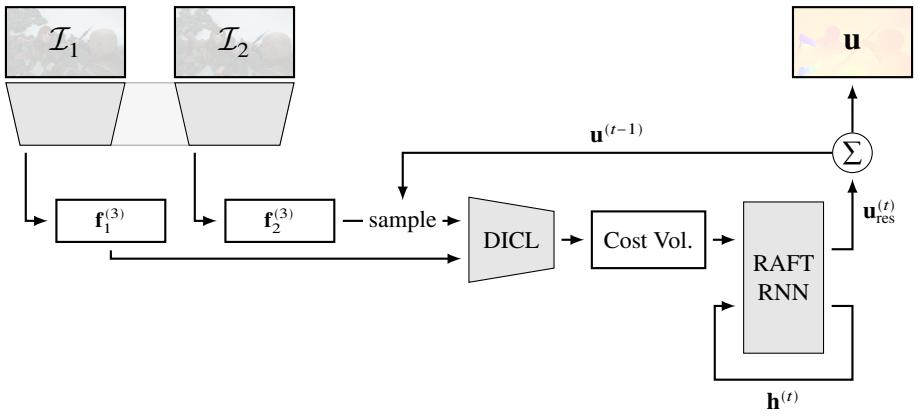

Figure 4.1: Architecture of the single-level RAFT+DICL approach. Input images $\mathcal{I}_1$ and $\mathcal{I}_2$ are transformed into a single-level feature representation $\mathbf{f}_1^{(3)}$ and $\mathbf{f}_2^{(3)}$, respectively, using a feature encoder with tied weights (indicated by shared gray background). In each recurrent iteration, costs are computed directly on that single level by sampling from the second frame features $\mathbf{f}_2^{(3)}$ using the current total flow estimate $\mathbf{u}^{(t-1)}$ and processing feature pairs via the DICL module, consisting of matching network and displacement-aware projection. This cost volume slice is then used in the RAFT recurrent network to estimate the residual flow $\mathbf{u}_{\text{res}}^{(t)}$, which is summed up to obtain the total flow estimate $\mathbf{u}^{(t)}$. Learned network blocks are indicated in gray.

In each recurrent iteration, we compute a cost volume slice from the first-frame features $\mathbf{f}_1^{(3)}$ and the sampled second frame features $\mathbf{f}_2^{(3)}(\mathbf{X})$, using the standard DICL module (i.e., matching network and displacement-aware projection) as described in Section 3.2 and the approach derived in Section 3.3. The sampling grid $\mathbf{X}$ is derived from the current full flow estimate $\mathbf{u}^{(t-1)}$ via Equation (3.4). As we only have a single feature representation and our approach does not (easily) allow for downsampling of cost values, we only generate the cost volume slice at that single level. RAFT, in comparison, uses four cost volume levels by default. To account for this difference, we simply adapt the corresponding input layer of the recurrent network employed by RAFT, reducing its number of input channels. Note that this is possible as RAFT flattens the cost values over all displacements and levels, resulting in one cost vector per pixel.

Just like in RAFT, a residual flow update $\mathbf{u}_{\text{res}}^{(t)}$ is then generated from that cost volume slice using said convolutional RNN. The hidden state of the RNN is initialized via a context encoder network (again taken from RAFT), which also provides a (static) context encoding as input to the RNN for each iteration. The residual flow update is summed up to obtain the full flow estimate $\mathbf{u}^{(t)}$, which, in turn, is upsampled to the resolution of the original input images using the convex upsampling module of RAFT (cf. Section 3.1.1). During training, upsampling is performed in each iteration, with the upsampled flow of each iteration being used in the loss function. The full RAFT RNN is discussed in Section 3.1.1 in more detail. Apart from the aforementioned adaptation of the input layer, we make no changes. An overview of the architecture, neglecting the upsampling module and context encoder, is provided in Figure 4.1.

4.1.2 Default Parameters

We adapt all default parameters from either RAFT [TD20] or DICL [WZD+20]. As already mentioned above, we follow DICL for the layout of the DICL module and choose $D = 32$ feature channels. Based on RAFT, we select a displacement range of $R = 4$ in each direction for a total of $N_d = 9 \times 9$ displacement hypotheses. For training, we use the sequence loss employed by RAFT, presented in Section 3.1.2, with the loss parameters following our training strategy, discussed in Section 4.5.

4.2 RAFT+DICL Multi-Level

The most notable drawback of the single-level approach is its limited suitability for larger motions. Teed and Deng [TD20] therefore propose downsampling the cost volume in dimension of the second frame, providing a larger displacement range through coarser levels. We will essentially do the same in our multi-level approach, with the exception that we base this on the reordered version derived in Section 3.3, in particular Equation (3.27), for feasibility. Hence, the multi-level RAFT+DICL approach can essentially be seen as an extension of the single-level approach (presented in Section 4.1), or, in other words, the single-level approach can be seen as a special case of the multi-level one. This, however, comes with the downside of significantly heightened memory requirements, which we will address in the next section.

4.2.1 Architecture

As implied by Section 3.3 and Equation (3.27), the creation of a multi-level cost volume requires establishing a multi-level feature representation. Quite crucially, this is not a classical dual feature pyramid as found in the common coarse-to-fine methods, but rather a feature stack for the first frame, retaining the same resolution for features $\mathbf{f}_1^{(m)}$ over all levels m, and a feature pyramid for the second frame, where the resolution of features $\mathbf{f}_2^{(m)}$ is reduced across levels. This is required as RAFT, and therefore our adaption, estimates flow at a single scale, equal in resolution to the first-frame features.

We investigate two approaches for the feature encoders creating this representation: First, we simply use the original feature encoder used in RAFT to create features $\mathbf{f}_1^{(3)}, \mathbf{f}_2^{(3)}$ and then pool the $\mathbf{f}_2^{(3)}$ features to obtain $\mathbf{f}_2^{(m)}$ with $m > 3$. For the first frame, we simply use the features $\mathbf{f}_1^{(3)}$ without modifications, i.e., we let $\mathbf{f}_1^{(m)} = \mathbf{f}_1^{(3)}$ for all m. Note that this is conceptually closest to the original RAFT approach. In particular, if we would use the dot product for cost computation, we would obtain the reordered RAFT approach derived in Section 3.3.4. As an alternative to this, we also explore an "asymmetric" extension of the RAFT feature encoder, retaining its residual structure. We again use the original RAFT feature encoder as a base to create the features $\mathbf{f}_1^{(3)}$ and $\mathbf{f}_2^{(3)}$ via shared weights, but then process them independently to obtain $\mathbf{f}_1^{(m)}$ and $\mathbf{f}_2^{(m)}$ with $m > 3$. To retain the feature resolution for the first frame, we employ a mix of normal and dilated convolutional layers, whereas we use normal convolutional layers to reduce resolution when processing the second frame features. Similar to the original RAFT encoder, we employ a 1×1 convolutional layer as output for each level, transforming the features to have a common number of $D = 32$ feature channels. In theory, however, different levels may have a different number of feature channels.

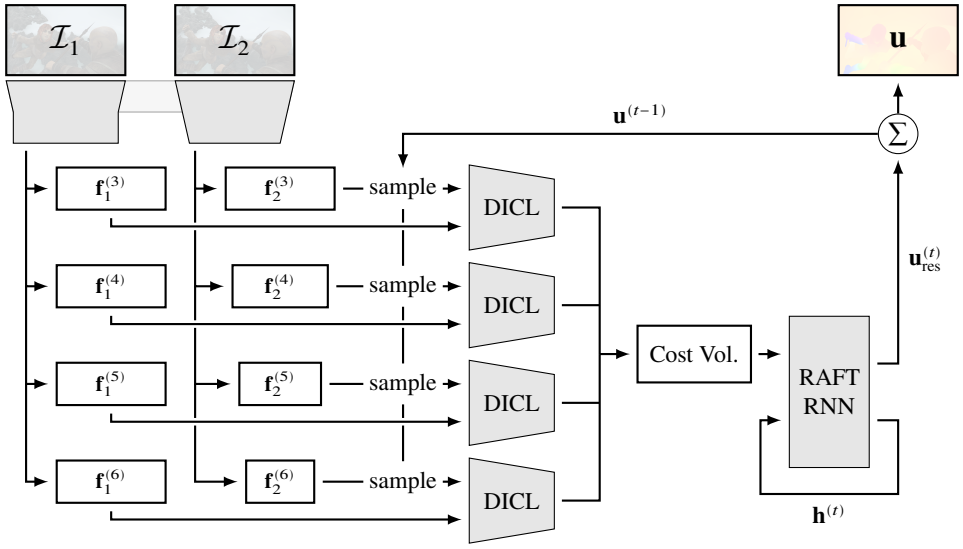

Figure 4.2: Architecture of the multi-level RAFT+DICL approach. An asymmetric multi-level feature representation $\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)}$ is generated from the input images $\mathcal{I}_1, \mathcal{I}_2$, respectively, via a feature encoder. Crucially, the first-frame features $\mathbf{f}_1^{(m)}$ have a common resolution across all levels m, whereas the second frame features $\mathbf{f}_2^{(m)}$ form a feature pyramid by reducing the resolution to obtain the coarser levels. Costs are computed individually for each level m and iteration t, by sampling from the second frame features $\mathbf{f}_2^{(m)}(\mathbf{X})$ using the current flow estimate $\mathbf{u}^{(t-1)}$ and combining those with the first frame features $\mathbf{f}_1^{(m)}$ via the DICL module. In each iteration, costs are collected over all levels and evaluated displacements, before being fed to the RAFT RNN to estimate a residual flow update $\mathbf{u}_{\text{res}}^{(t)}$. This residual update is summed up to compute the new full flow estimate $\mathbf{u}^{(t)}$. Note that flow is only estimated at the single resolution defined by the first-frame feature stack $\mathbf{f}_1^{(m)}$. Learned network blocks are indicated in gray, shared weights via a gray background.

The remaining architecture follows the same scheme as the single-level approach and, in large, RAFT. Flow is estimated iteratively. In each iteration t, we first sample from the second frame features $\mathbf{f}_2^{(m)}$ by incorporating the current full flow estimate $\mathbf{u}^{(t-1)}$, using the sampling grid $\mathbf{X}$ as defined in Equation (3.4). Feature pairs for each displacement hypothesis are then processed individually per level m via the DICL module, consisting of matching network and displacement-aware projection (DAP), to produce the cost volume slice for that level. By default, we do not share weights between the DICL modules (specifically, matching networks) across levels. This enables the network to learn different cost functions for each level, and therefore potentially focus on different aspects of the features on each level. While we could use one DAP layer per level as part of the DICL module, we instead choose to employ a single combined layer across all levels.

Costs are collected and flattened over all levels and displacement hypotheses, and then passed on to the RAFT RNN. Depending on the number of levels used, we again simply modify the number of input channels accepted by that network to reflect the overall number of cost values. The RAFT RNN is initialized by a context encoder network (again taken from RAFT) and produces a residual flow update $\mathbf{u}_{\text{res}}^{(t)}$ in each iteration, which is summed up to create the new full flow estimate $\mathbf{u}^{(t)}$. The context encoder also provides context embeddings as input to the RNN in each iteration. As a last

step, the full flow estimate is upsampled via the convex upsampling module of RAFT to create flow at the original input resolution. During training, this is done for each iteration, just like original RAFT, with the upsampled flow of each iteration being used in the loss function. The RAFT RNN is therefore kept as proposed by Teed and Deng [TD20] and discussed in Section 3.1.1, with a potential adaptation of the input layer to account for a different number of cost volume levels. An overview of this approach (again neglecting the convex upsampling module and context encoder) is illustrated in Figure 4.2.

4.2.2 Default Parameters

We again choose our default parameters based on RAFT [TD20] and DICL [WZD+20]. In particular, we employ $D = 32$ feature channels (as already stated above), and use a displacement range of $R = 4$ per direction on each level for a total of $N_d = 9 \times 9$ displacement hypotheses per level. We perform experiments with both $M = 2$ and $M = 4$ cost volume levels. The DAP layer is initialized as identity mapping. For training, we employ the sequence loss of RAFT, described in Section 3.1.2. Loss parameters follow our training strategy, presented in Section 4.5.

4.3 RAFT+DICL Coarse-to-Fine

The coarse-to-fine RAFT+DICL approach can be understood as a rearrangement of the single-level RAFT+DICL model presented in Section 4.1 into a coarse-to-fine scheme (or, by extension, a rearrangement of the multi-level RAFT+DICL approach presented in Section 4.2). In essence, flow is being estimated iteratively from the coarsest to the finest level, while also being estimated iteratively on each level. Our main rationale behind this approach is to lower the memory requirements compared to the multi-level method. The coarse-to-fine setting is ideal for this, as the high memory requirements of the multi-level approach are a direct result of estimating flow at a single and therefore, by necessity, comparatively fine level. RAFT requires the first-frame dimensions over all levels of the multi-level cost volume to be the same, due to which there are no cost savings associated with coarser levels, at least in our reordered approach (cf. Section 3.3.4). The coarse-to-fine approach does not have this restriction as flow is estimated over multiple levels, leading to a lower memory requirement. This is also the deciding factor as to why we only employ a single cost volume level per coarse-to-fine level (multiple levels would again negate the cost savings), however, a generalization to multiple cost volume levels per coarse-to-fine level is also possible (in essence, allowing for the creation of a hybrid model).

4.3.1 Architecture

This being a coarse-to-fine approach, we start with a classical dual feature pyramid, i.e., we, using feature encoders with shared weights, transform the input images $\mathcal{I}_1$ and $\mathcal{I}_2$ into feature embeddings $\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)} \in \mathbb{R}^{H/2^m \times W/2^m \times D}$, respectively, for multiple levels m. By default, we generate features from the finest level $m = 3$ to the coarsest level $m = M$, with some $M \in \{4, 5, 6\}$. For the feature encoder itself, we simply extend the RAFT encoder by continuing its residual structure to produce

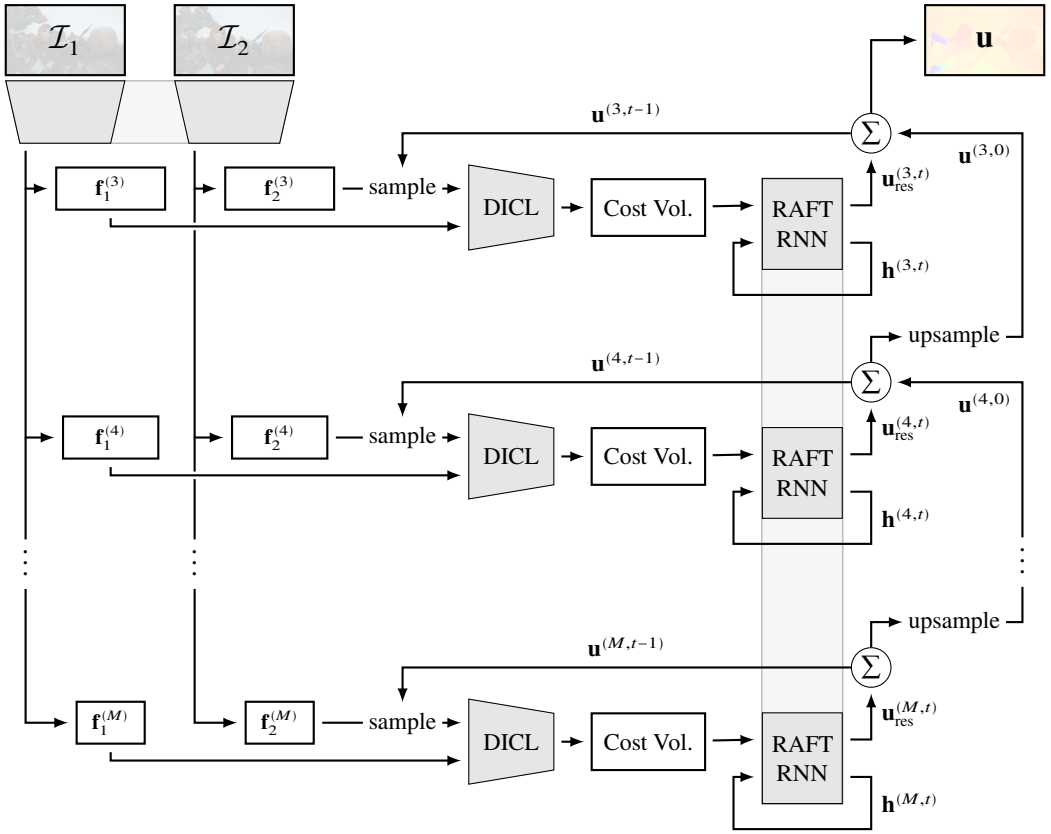

Figure 4.3: Architecture of the coarse-to-fine RAFT+DICL approach. Input images $\mathcal{I}_1, \mathcal{I}_2$ are transformed into feature pyramids $\mathbf{f}_1^{(m)}, \mathbf{f}_2^{(m)}$, respectively, using two feature encoders with shared weights. Flow is estimated iteratively over all feature levels m, starting on the coarsest level $m = M$. On each level, in turn, flow is again estimated iteratively in a RAFT-like fashion: First, second-frame features $\mathbf{f}_2^{(m)}$ are sampled using the current full flow estimate $\mathbf{u}^{(m,t-1)}$ and combined with the first-frame features $\mathbf{f}_1^{(m)}$ to be fed to the DICL module. This produces the single-level cost volume slice for level m and iteration t, which is then processed by the RAFT RNN to estimate a residual flow update $\mathbf{u}_{\text{res}}^{(m,t)}$. Weights for the RAFT RNN are shared across all levels. The residual estimate is summed up to obtain the full flow estimate $\mathbf{u}^{(m,t)}$ for that level. After all iterations on the level have been completed, the final full flow estimate $\mathbf{u}^{(m,T)}$ is upsampled to produce the initial flow estimate $\mathbf{u}^{(m-1,0)}$ on the next finer level. Learned network blocks are indicated in gray, shared weights via a gray background.

multiple feature levels. To obtain a flexible number of output channels, we again employ a 1×1 output convolution. By default, we choose $D = 32$ feature channels for all levels, however, the number of channels could also be chosen independently for each level.

Following the common coarse-to-fine scheme, flow is estimated iteratively from coarsest to finest level. On each level, flow is estimated recurrently via the RAFT RNN and the DICL module. More specifically, for some level m we again follow the approach derived in Section 3.3 by first sampling from second-frame features $\mathbf{f}_2^{(m)}$ using the current flow estimate $\mathbf{u}^{(m,t-1)}$. This flow estimate is either initialized with zero at the coarsest level, i.e., $\mathbf{u}^{(M,0)} = 0$, or initialized from the upsampled final estimate of the next coarser level, i.e., $\mathbf{u}^{(m,0)} = \mathrm{up}(\mathbf{u}^{(m+1,T_{m+1})})$. Here, T_{m+1} denotes the last iteration on level $m + 1$, "up" the bilinear upsampling operator, and $m < M$ the level, with the coarsest level given by M. The sampling grid $\mathbf{X}$ again follows Equation (3.4), however, note that the grid is now relative to the given coarse-to-fine level.

From the first-frame features $\mathbf{f}_1^{(m)}$ and sampled second-frame features $\mathbf{f}_2^{(m)}(\mathbf{X})$, we can then compute the costs for this level and recurrent iteration via the DICL module, i.e., matching network and displacement-aware projection. In short, each coarse-to-fine level again follows the overall approach derived in Section 3.3 as well as Equation (3.27). Note that we only compute costs at a single cost volume level for each coarse-to-fine level. An extension to multiple cost volume levels is possible by, for example, pooling the second frame features of that coarse-to-fine/feature level to obtain a feature pyramid for those. Depending on the feature encoder, coarser second-frame feature representations of it could also be chosen directly for this.[1]

Once costs have been computed, they are again fed into the RAFT RNN (cf. Section 3.1.1) to estimate a residual flow update $\mathbf{u}_{\mathrm{res}}^{(m,t)}$, this time for both the current level m and iteration t. As before, the hidden state of the RNN is initialized via a context encoder network that shares its structure (but not weights) with the feature encoder network. This context encoder also provides contextual embeddings as input to each RNN iteration. In contrast to the prior approaches, context embeddings are now created for each (coarse-to-fine) level. The full flow estimate $\mathbf{u}^{(m,t)}$ for some level m is computed by summing up all prior residual updates ($t' \leq t$) for that level. On the finest coarse-to-fine level, we use the RAFT convex upsampling module to upsample the final full flow estimate. An overview of the approach, neglecting upsampling module and context encoder, is given in Figure 4.3.

4.3.2 Default Parameters

By default, we do not share weights for the DICL modules across levels. The rationale behind this is that this allows the network to learn level dependent feature representations and cost functions. As we believe that the representation of motion via a cost volume should be universal across all levels, however, we share weights for the RAFT recurrent network across those. Other defaults are again based on RAFT [TD20] and DICL [WZD+20]. In particular, we choose $D = 32$ feature channels for each coarse-to-fine level. This also allows us to easily experiment with weight sharing for the DICL modules. We further follow RAFT [TD20] and employ a displacement range of $R = 4$

[1] This may, however, be limited in success as the first frame features need to retain the intrinsic resolution of the coarse-to-fine level (i.e., form a stack whereas the second-frame features need to form a pyramid, cf. Sections 3.3.4 and 3.3.5). Choosing coarser features for the second-frame only could therefore lead to mismatching representations.

in each direction, for a total of $N_d = 9 \times 9$ displacement candidates per level. As we save memory via the reduction in spatial resolution on coarser levels, it would, however, also be possible to use a larger number of feature channels or a larger number of displacements on those. In case of the latter, however, sharing of RAFT RNN weights would no longer be (easily) possible.

During training, we again upsample each full flow estimate (on all levels) and consider it in the loss term. On the finest level, we use the RAFT upsampling module, whereas, on coarser levels, we use bilinear upsampling instead. Note that, as a result, we still obtain a single full-resolution flow sequence (even though the original output sequence is a multi-resolution one), due to which we can again use the standard RAFT sequence loss. As before, all loss parameters follow our training strategy, elaborated in Section 4.5.

4.4 Multi-Sequence Loss

While we can train all approaches proposed above with the standard RAFT sequence loss given in Equation (3.5), we, nevertheless, suggest a generalization to it for multiple sequences. We will later employ this loss for an improved supervision technique. Similar to the original sequence loss, our *multi-sequence loss* is applied to the estimated flow, upsampled to the resolution of the input images. We define it as

$$\mathcal{L}(\theta) = \sum_{m \in \mathcal{M}} \alpha_m \sum_{t=1}^{T_m} \gamma^{T_m - t} \sum_{(i,j)^\top \in \Omega} \frac{1}{|\Omega|} \left\| \mathbf{u}_{\text{gt},i,j} - \mathbf{u}_{\text{up},i,j,\theta}^{(m,t)} \right\|_k . \tag{4.1}$$

Here, $\mathbf{u}_{\text{gt}}$ denotes the ground-truth flow, $\mathbf{u}_{\text{up}}^{(m,t)}$ the upsampled (full) flow estimate for sequence m and iteration t of that sequence, Ω the set of all pixels $(i,j)^\top$ for which ground-truth flow is available, and θ the model parameters. Further, $\mathcal{M} \subset \mathbb{N}_0$ denotes the set of sequence indices and T_m the number of recurrent iterations of that specific sequence m. The loss parameter α_m represents the weight of that sequence, whereas the exponential weight γ defines weighting of elements internal to a sequence. Lastly, the parameter k controls the norm type. Note that by letting $\mathcal{M} = \{0\}$ be a single-element set and $\alpha_0 = 1$ this falls back to the standard sequence loss (cf. Equation (3.5)).

We can apply this loss to our coarse-to-fine approach, presented in Section 4.3, by interpreting its result as a multi-level sequence, with one sequence per level. This allows levels to be weighted independently, which provides more flexibly. In this case, the set of sequences is simply the set of levels $\mathcal{M} = \{3, 4, \dots, M\}$, ranging from the finest level $m = 3$ to the coarsest level $m = M$. Note that we can still make this application scheme fall back to the standard sequence loss over all levels by choosing appropriate α_m. In particular, by letting $\alpha_m = \gamma^{T_m^+}$ with $T_m^+ = \sum_{m' \in \mathcal{M}, m' < m} T_m$ denoting the number of iterations from the current sequence element to the last element.

4.5 Training Strategy

As all of our approaches follow the general architectural scheme of RAFT, although adapted for our needs, we also base our training strategy on the one used by Teed and Deng [TD20] and adjust it to make training feasible on the hardware available to us. In particular, we adjust the strategy so that the majority of our experiments can be run on a single NVIDIA RTX 3090 GPU with 24 GiB

VRAM. Additionally, we train on NVIDIA Tesla V100 GPUs with 32 GiB VRAM each, by default using two in parallel to speed up training. We verified that, with our strategy, results are equal when training on one and on two GPUs (except for runs in which we enable batch normalization, which we always train on a single GPU). To ensure comparability, all of our models are trained with the same strategy, only using minor adjustments to ensure convergence when required. This section provides an overview of our training strategy. Section 4.5.1 will discuss a preparatory epoch to ensure convergence, followed by the main training stages in Section 4.5.2 and a short overview of padding in Section 4.5.3. Lastly, we provide a summary of all augmentations applied to the training data in Section 4.5.4 and discuss supervision, including loss parameters, in Section 4.5.5.

Training is divided into three major stages, spread over three datasets: Models are pre-trained on FlyingChairs2 [ISKB18], followed by FlyingThings3D [MIH+16], and, finally, fine-tuned on the MPI Sintel dataset [BWSB12]. This curriculum-learning strategy has been shown effective by earlier approaches, such as PWC-Net [SYLK18; SYLK20] and DICL [WZD+20]. We refrain from an additional training stage targeting the KITTI datasets [GLU12; MHG15], as is used by RAFT and the aforementioned approaches, in favor of evaluating more models and variations. In contrast to RAFT, we chose FlyingChairs2 over FlyingChairs [DFI+15] as the former provides both forward and backward flow, whereas the latter only provides forward flow. Both datasets are conceptually equal and provide the same amount of data. If required, backward flow could be estimated, for example using the techniques described by Sánchez et al. [SSM15].

4.5.1 Preparatory Epoch

Due to instabilities, in particular convergence problems experienced during the initial training steps of some of our models (later discussed in more detail in Section 6.1), we start training with a specialized first epoch on the FlyingChairs2 dataset. In this epoch, we combine both forward and backward flows of the same image pair into a single batch, i.e., for some input image pair $\mathcal{I}_1, \mathcal{I}_2$ we use both $\mathbf{u}_{1\to2}$ and $\mathbf{u}_{2\to1}$ in the same batch, leading to a batch structure of

$$\mathcal{B} = \{\dots, (\mathcal{I}_1, \mathcal{I}_2, \mathbf{u}_{1\to2}), (\mathcal{I}_2, \mathcal{I}_1, \mathbf{u}_{2\to1}), \dots\}. \tag{4.2}$$

This forward-backward-batching has previously been employed by Jaegle et al. [JBA+22] in their Perceiver-IO approach and seems to aid convergence.[2] We believe that combining forward and backward flow in this manner may potentially balance out directional biases and components, thus helping the model learn motion cues independently of their orientation. A similar result could maybe be obtained by using a larger batch size (we use a batch size of 6 for this epoch), and we theorize that forward-backward-batching may simulate such. To further improve stability and convergence, we only employ basic augmentations during this initial epoch, in particular randomized cropping as well as horizontal and vertical flipping, and adapt the learning rate on a per-model basis, with one of 4×10^{-4} (default), 1×10^{-4}, 5×10^{-5}, or 5×10^{-6}. The learning rate is modulated by a linear one-cycle learning rate scheduler, equal to the one used by Teed and Deng [TD20]. We found that, combined, this first epoch is sufficient for ensuring convergence and that the learning rate can be raised after initial convergence has been reached. We will discuss instability problems of our approaches in more detail in Section 6.1.

[2] See also https://github.com/deepmind/deepmind-research/issues/266 for more information on the stability issues of Perceiver-IO. Accessed on April 19, 2022.

4.5.2 Main Training Stages

After this preparatory epoch, training continues on FlyingChairs2 for 14 epochs (for a total of 15 epochs on this dataset), followed by 10 epochs on FlyingThings3D and, finally, 250 epochs on Sintel. This represents approximately 100 000 steps per stage. We will later refer to the former two stages using their abbreviated names "Chairs2" and "Things", respectively. For the FlyingThings3D and Sintel datasets, we train on both the "clean" and "final" subsets mixed together. We use the original training splits for the FlyingChairs2 and FlyingThings3D datasets, and use the split proposed by MaskFlowNet [ZSD+20][3] for training and evaluation on Sintel.

In contrast to Teed and Deng [TD20], we chose to train on the Sintel dataset only instead of a mix combining Sintel, FlyingThings3D, KITTI 2015, and HD1K[4] for simplicity, especially as their original mix proposes to use the full Sintel dataset, which would only allow us to validate using data we train with. For some selected models, we additionally train exactly 100 000 steps on a RAFT-like mix using the MaskFlowNet Sintel split as an alternative to the full Sintel dataset. This new mix follows the same overall composition as the original mix (i.e., the lower number of samples of the Sintel split has been compensated to obtain the same relative share).[5] Note that other approaches, such as MaskFlowNet [ZSD+20] and DICL [WZD+20], also use a simple Sintel-only stage.[6]

We use a batch size of 6 for FlyingChairs2 and 4 for FlyingThings3D and Sintel. It has been chosen such that the majority of our experiments can run on a single RTX 3090 GPU, i.e., it is limited by the available memory of that GPU and the memory requirements of our approaches. The batch size is therefore smaller than the one used in RAFT (with 12, 6, and 6 according to their paper [TD20]). An increase of the batch size would be possible via gradient accumulation.[7] In addition, we also reduce the number of iterations for our RAFT-based approaches from the original 12 to 4. For the coarse-to-fine approaches, we use 4 iterations on the coarsest level and 3 iterations on all other levels. The rationale behind this choice is the facilitation of a fairer comparison by (roughly) keeping memory requirements equal: In terms of memory, one iteration on a finer level has similar requirements as 4 iterations have on the next coarser level due to the resolution being halved in both height and width when downscaling to that level, leading to an overall (theoretical)

[3] The MaskFlowNet split uses the ambush_2, ambush_6, bamboo_2, cave_4, market_6, and temple_2 scenes for validation and all other scenes for training.

[4] Specifically, this mix uses a composition of 71% Sintel, 13.5% FlyingThings3D, 13.5% KITTI 2015, and 2% HD1K.

[5] One may question, however, whether such compensation should actually be performed: Retaining the same portion of the Sintel dataset with, however, now less variation leads to (relatively) less variation in the final mix. A simple replacement without compensation would still lead to (absolutely) less variation overall compared to the original mix, which, however, is now shared across the larger mix, therefore still obtaining a larger amount of variation than direct compensation. This question is a part of why we originally decided to use a simple Sintel-only stage. We, however, found that a mixed Sintel training stage may be useful for investigating potential overfitting issues.

[6] As Sun et al. [SYLK20] note, however, training matters a great deal and an optimal training strategy may depend on the model to be evaluated. For final benchmark-ready results, evaluation on the proposed split may therefore prove beneficial. We believe that, for the scope of this thesis, a simple Sintel stage and (selectively) the compensated Sintel mix should be enough to provide qualitative results for comparative evaluation.

[7] Gradient accumulation describes the process of accumulating the gradient over multiple backward passes. This can be used to simulate a larger batch size by collecting the gradient over multiple sub-batches and only running the optimizer once a sufficient number of sub-batches have been processed. Using n as the number of accumulated sub-batches, together forming one simulated batch, this leads to one optimizer step per n backward passes. As a downside, this has implications on techniques depending on the "true" batch size during forward-/backward passes, such as batch normalization [IS15] (which could be replaced by weight standardization [QWL+20] to avoid this problem).

reduction by a factor of 4. To obtain the number of iterations for our coarse-to-fine evaluations, we therefore essentially trade one iteration on a finer level for 4 iterations on the next coarser level. For benchmark-submission-ready results, one may want to think about a more clever allocation of iterations across levels during training and evaluation.

Due to the reduced batch size, especially when compared to DICL (using a batch size of 8 per GPU for a total of 64), and since batch normalization [IS15] generally requires a sufficiently large batch size (see, e.g., the works by Qiao et al. [QWL+20] and Wu and He [WH18]), we disable batch normalization layers during all training stages (i.e., do not use batch normalization). RAFT, in contrast, only freezes batch normalization on the second and third stage. We use crop sizes of 496×368 for FlyingChairs2, 720×400 for FlyingThings3D, and 768×368 for Sintel, as well as learning rates of 4×10^{-4} for FlyingChairs2 and 1.25×10^{-4} for FlyingThings3D and Sintel. The learning rate is again modulated using a linear one-cycle scheduler with the same parameter values as were used for training RAFT [TD20]. Additionally, we clip gradients to a norm of one, again similar to the original RAFT strategy.

4.5.3 Padding

As another difference to the original RAFT evaluation strategy, we use zero-based padding instead of replication-based padding. Zero-based padding is used, among other approaches, by DICL and has in our informal testing, using the original checkpoints provided by Teed and Deng [TD20], lead to slightly better results. We believe that this is sound, since replication-based padding, which works by mirroring the border area of the input images, intrinsically leads to ambiguous matches, whereas zero-based padding avoids that. Note that RAFT does not use any padding during training as the image sizes are already cropped to a multiple of 8. Ideally, one would adjust the crop size to the model in an effort to avoid padding, however, as our strategy is intended to be general and used across multiple methods we chose to stick with the original crop sizes proposed by Teed and Deng [TD20]. Further, Bar-Haim and Wolf [BW20] have shown that decreasing the crop size can lead to worse results.

4.5.4 Augmentations

To prevent overfitting and improve generalization, we adopt the data augmentations used for training RAFT [TD20]. Specifically, we employ randomized scaling, cropping, and horizontal as well as vertical flipping of input samples to spatially augment them. We further, on top of the augmentations used by RAFT, add a small randomized translation up to a maximum of 10 px in each direction. To allow for generalization to different lighting situations, we augment input images via randomized perturbations to brightness, contrast, saturation, and hue (again following Teed and Deng [TD20]). These augmentations can be applied either symmetrically to both input images or, asymmetrically, to each input image individually. This decision is again randomized. Lastly, we add artificial occlusions by replacing small randomized parts of the second image with the average color of that image. This is intended to improve generalization in case of occlusions. Similar to RAFT, we do not add any noise, such as additive Gaussian noise, which is used by other approaches, notably the original PWC-Net [SYLK18] (in particular, as Sun et al. [SYLK20] have later shown that additive noise can be detrimental).

4.5.5 Supervision

We train all DICL baselines with the standard DICL loss given in Section 3.2.2, using the same loss parameters (i.e., weights α_m and restriction thresholds $r_u^{(m)}, r_v^{(m)}$) as proposed by Wang et al. [WZD+20] if not specified otherwise. This also includes choosing the L^2 norm as base measure. The DICL loss is applied to both the direct result of the soft-argmin regression and the subsequently refined flow. For our RAFT baselines as well as all RAFT-based architectures, we employ the standard RAFT sequence loss, given in Section 3.1.2. This loss is applied to the flow upsampled via the convex upsampling module. In case of the coarse-to-fine approaches, we again upsample flow on the finest level via the convex upsampling module, however, we also upsample flow from coarser levels via bilinear interpolation and apply the loss to the full sequence over all levels. Following the original Sintel stage of RAFT [TD20], we choose $\gamma = 0.85$ as exponential weight for all stages and use the L^1 norm as base measure.

4.6 Evaluation Strategy

We evaluate our approaches during and after each training stage (see Section 4.5 for their structure and parameters), with the specific evaluation datasets depending on the respective stage. For FlyingChairs2, we rely on the official FlyingChairs2 test split. In contrast, we evaluate our FlyingThings3D results on the full Sintel training sets, with separate evaluations for the "clean" and "final" subsets. We believe that this produces more meaningful results, in particular as our intended training target is the Sintel benchmark. This further allows us to obtain an indicator for cross-dataset generalization, as, in this stage, the Sintel dataset has not been used for training yet. Note that the FlyingThings3D stage is the first that introduces true three-dimensional motion. The prior FlyingChairs2 stage employs two-dimensional transformations only (cf. Section 2.2), due to which we do not evaluate the results of it on the Sintel dataset, believing that doing so may not be reflective of the true model performance. Lastly, we evaluate both (alternative) Sintel training stages (Sintel-only and Sintel Mix) on the MaskFlowNet [ZSD+20] validation split (cf. Section 4.5). Results suggest that this split highlights (some of) the more complex aspects of the Sintel dataset during evaluation.

Following the established standards of neural network based optical flow estimation methods, we use the average endpoint error (AEE), discussed in Section 2.3.1, as the main evaluation measure. To facilitate the evaluation of multiple samples, i.e., complete datasets and validation splits, we take the mean over all of said samples. By default, we employ the same model parameters as were used when training the model. This, in particular, relates to the number of iterations for our recurrent approaches, which has been detailed (for training) in Section 4.5.

Similar to the training stages above, we will also abbreviate the evaluation datasets. Moreover, we will use a shorthand notation for the combination of both by separating them with a slash. For example, we will use "Things/Sintel (clean)" to refer to evaluation on the "clean" (full) Sintel dataset after training on the FlyingThings3D stage. Equivalently, "Sintel/Sintel" then refers to both "clean" and "final" evaluations on the MaskFlowNet Sintel validation split after training on the MaskFlowNet Sintel(-only) training split.

4.7 Rationale and Plan

Within this chapter, we have established the basis for the evaluations performed in the remaining part of this thesis. We begin in Chapter 5, investigating both RAFT and DICL baselines, as well as the methods discussed in Sections 4.1 to 4.3. This chapter is intended to form the fundament from which we can then explore improvements and discuss encountered difficulties in Chapter 6. During evaluations, we mainly focus on one respective base model, from which we derive variations to gain a better understanding of the model itself. Specifically, we attempt to find its limitations, its strengths, and potential starting points for improvements. By providing additional ablations for our respective RAFT and DICL baselines, we aim to understand where potential limitations may stem from and how they could potentially be resolved.

5 Base Results

Having presented our three main RAFT+DICL architectures in Sections 4.1 to 4.3 as well as our training strategy in Section 4.5, we can now start with some basic evaluations of these models. In particular, this chapter aims to provide a foundation upon which we will improve in Chapter 6. We begin in Section 5.1 by analyzing our baseline models, RAFT and DICL. In Section 5.2, we then investigate the single-level RAFT+DICL model, which we use as a simplified starting point to gauge the viability of our general RAFT+DICL approach. Section 5.3 evaluates its naive extension to multiple levels, following the original RAFT strategy. Section 5.4 investigates the coarse-to-fine strategy as an alternative to the multi-level approach, which requires less memory and is therefore somewhat more feasible. Finally, Section 5.5 summarizes and concludes our findings, preparing us for the next chapter, in which we will discuss and analyze problems found in the evaluated approaches and attempt to resolve them as well as improve our models further.

5.1 Baselines and Baseline Ablations

We begin our evaluation by investigating our baselines, RAFT and DICL (cf. Sections 3.1 and 3.2), as well as variations thereof. The intended goal of this small study is to obtain a frame of reference on how our combined approaches should perform and to assess the impact of the trade-offs taken to make them feasible. Via this, we hope to gain a better understanding, not just of the baseline methods but also of our proposed techniques, by obtaining a solid foundation for analysis, including discovery and investigation of potentially limiting factors.

5.1.1 RAFT

For RAFT, our variations focus on the standard model parameters as well as the training strategy, more specifically the employment of batch normalization [IS15] during training. The particular model parameters investigated are the specific cost volume levels being used for flow estimation, the lookup range per direction in the cost volume, the number of channels in the feature representation, and the number of recurrent iterations performed when training. Results in terms of endpoint error are shown in Tables 5.1 to 5.4.

Receptive Field

Both the cost volume levels and the lookup operator range R, determining the number of displacement hypotheses N_d via $N_d = (2R + 1)^2$, allow us to investigate the importance of the size of RAFT's total per-iteration receptive field. While the lookup range allows for a fairly direct control thereof, the selection of levels provides a trade-off between range and spatial resolution and therefore has a

Parameters					Chairs2	Things		Sintel	
Levels	*Rng.*	*Chn.*	*Iter.*	*BN*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
{3, 4, 5, 6}	4	256	4	✗	1.2832	1.9707	3.2727	2.3113	3.5204
{3, 4, 5}	4	256	4	✗	1.2977	1.9972	3.3378	2.2981	3.5269
{3, 4}	4	256	4	✗	1.3118	2.0402	3.3270	2.5337	4.2979
{3}	4	256	4	✗	1.4209	2.3015	3.5155	2.9718	4.5847
{3}	7	256	4	✗	1.3301	2.1118	3.4481	2.5551	4.4555

Table 5.1: RAFT ablations for analysis of the impact of the receptive field size. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including the specific cost volume levels (*Levels*), lookup operator/displacement candidate range in each direction (*Rng.*), number of channels in the feature representation (*Chn.*), number of recurrent iterations (*Iter.*), and whether batch normalization has been employed (*BN*).

comparably smaller impact on computational and memory requirements. One can further argue that providing cost values at different scales may give the model an additional architectural advantage that could help resolve ambiguous matches.

Results for the investigation of the receptive field size are provided in Table 5.1. Levels are denoted by their base-two downsampling exponent with respect to the input image size (i.e., level m is a factor of 2^m smaller in both height and width than the input image). The largest gains in qualitative performance can be observed when going from a single to two cost volume levels, with the notable exception of the Sintel/Sintel (final) result. There, the biggest improvement can be seen when going from two to three levels.

In general, the relative improvement seems to somewhat follow a negative exponential curve (i.e., smaller improvements with increasingly larger receptive fields), which is not surprising considering dataset statistics: About 90% of the ground-truth flow is below a magnitude of 29 px in the FlyingChairs2 test set, 36 px in the full Sintel dataset, and 47 px in the MaskFlowNet Sintel validation split. Similarly, about 95% of the ground-truth flow is below a magnitude of 39 px in the FlyingChairs2 test set, 65 px in the full Sintel dataset, and 73 px in the MaskFlowNet Sintel validation split. Therefore, the finest level, with a default lookup operator range of 4 per level and thus a total range of 32 px per iteration[1], is already capable of capturing a significant number of ground-truth feature matches.

Extending the model to two levels extends its capabilities to a total range of 64 px and therefore above or somewhat close to capturing 95% of the ground-truth matches for all employed evaluation datasets but the MaskFlowNet Sintel split. By going up to three levels, we extend the model capabilities to a total range of 128 px, which is then able to capture close to 99% of correct matches on the MaskFlowNet split and even more so on the other datasets. Hence, diminishing returns

[1]Note that the model range is given in pixels for the x-/y-directions and not the true theoretical maximum magnitude, which is higher at the corners. This therefore represents the minimum over all orientations of the maximum flow magnitude (per orientation) that can be truthfully handled by the model.

Parameters			Chairs2	Things		Sintel	
Levels	*Rng.*	**u**-*max*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
{3, 4, 5, 6}	4	64	0.9836	1.3506	2.4567	1.2041	2.0859
{3, 4, 5}	4	64	0.9869	1.4093	2.4962	1.2260	2.1257
{3, 4}	4	64	0.9923	1.4408	2.4897	1.2579	2.1109
{3, 4, 5, 6}	4	32	0.7693	1.2684	2.2611	1.3385	2.2019
{3, 4, 5}	4	32	0.7722	1.3271	2.2837	1.3495	2.3873
{3, 4}	4	32	0.7790	1.3317	2.2178	1.4574	2.0902
{3}	4	32	0.8343	1.4233	2.2456	1.4537	2.1696

Table 5.2: RAFT ablations for analysis of the impact of multiple cost volume levels on match candidate regularization. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including the specific cost volume levels (*Levels*) and lookup operator/displacement candidate range in each direction (*Rng.*). The average endpoint error is computed only for pixels with maximum flow magnitude below the specified **u**-*max* threshold (in pixel).

beyond this point are expected. The aforementioned anomaly with the performance jump from two to three cost volume levels on the Sintel stage can likely, at least partially, be attributed to said statistics.

Some improvements can, nevertheless, be seen when going from three to four cost volume levels, especially after training on FlyingChairs2 and FlyingThings3D. This shows that there could potentially be benefits beyond the increased total per-iteration range (such as the aforementioned improved regularization for ambiguous matches), however, one needs to take into account that these high-magnitude matches likely produce a higher endpoint error compared to their lower-magnitude counterparts, therefore having a relatively higher impact on the end result. To address this, we reevaluate our RAFT ablations by restricting computation of the endpoint error to pixels with a ground-truth flow magnitude of 32 px and 64 px only[2], i.e., the range capabilities of the one- and two-level models.

Table 5.2 shows, that, overall, there indeed seem to be some smaller benefits of adding additional cost volume levels beyond the range extension. The most notable increase in performance can be observed when going from a single cost volume level to two levels. Beyond two levels, gains are somewhat inconsistent on the Sintel "final" sets, whereas other test sets still show small improvements. We argue that this may also be visible in Table 5.1 when going from a lookup range of 4 to a lookup range of 7 on the single-level model and comparing that with the addition of a second coarser level. While addition of the second level would correspond to a lookup range of 8 on the finer level (as opposed to 7)[3], we believe that results still show a significant performance uplift that cannot only be attributed to the increased range alone, especially considering the "final" evaluation sets. Evaluations with range 8, possibly for multiple level combinations, may be advisable to confirm this.

[2]Expressed mathematically, we redefine the set of pixels used for evaluation as $\Omega_t = \{\mathbf{x} : \mathbf{x} \in \Omega \wedge \mathbf{u}_{\text{gt}}(\mathbf{x}) \leq t\}$ and then use Ω_t (here with $t \in \{32, 64\}$) instead of Ω for our evaluation metrics as defined in Section 2.3.

[3]We chose 7 instead of 8 due to memory limitations with the RAFT+DICL single-level approach evaluated later in this chapter (training was performed using a single NVIDIA RTX 3090 GPU with 24 GiB VRAM).

Parameters		Chairs2	Things		Sintel	
Levels	*Chn.*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
{3, 4, 5, 6}	256	1.2832	1.9707	3.2727	2.3113	3.5204
{3, 4, 5, 6}	64	1.2985	1.9984	3.3936	2.4627	3.7454
{3, 4, 5, 6}	32	1.3762	2.1607	3.4531	2.4667	3.8451
{3}	256	1.4209	2.3015	3.5155	2.9718	4.5847
{3}	64	1.4221	2.3531	3.6052	2.8499	4.8030
{3}	32	1.4823	2.3525	3.6665	2.9657	4.7662

Table 5.3: RAFT ablations for analysis of the number of feature channels. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including the specific cost volume levels (*Levels*) and number of channels in the feature representation (*Chn.*).

Feature Channels

The number of feature channels and the number of recurrent iterations, on the other hand, do not influence the per-iteration receptive field, however, similar to the former two parameters provide a trade-off between quality of the result and computational requirements. Modifying the number of feature channels allows us to control the amount of information passed to the cost computation and therefore influence the quality of the cost volume. To modify the number of feature channels, we simply adapt the 1×1 convolutional output layer of the RAFT feature encoder. While this does not allow us to encode any new information, i.e., we should not expect better cost values by simply increasing the number of feature channels beyond a certain point[4] without changing the earlier layers of the feature encoder, this gives us the means for creating an information bottleneck by reducing the number of feature channels. Notably, we introduce such a (potential) bottleneck in our RAFT+DICL combination as we reduce the number of feature channels to a default of 32 (from 256 in RAFT) due to computational limitations. Therefore, we consider it an important parameter for our investigation.

Results are shown in Table 5.3. On the FlyingChairs2 evaluation set, the largest drop in performance can be observed when going from 64 down to 32 feature channels, whereas going from 256 to 64 channels has a substantially smaller impact. For the four-level RAFT model, the (largely) same behavior can be observed for Things/Sintel (clean), however, especially Sintel/Sintel evaluations show that a larger number of feature channels benefits feature matching in more complex scenarios. We believe that this is further reinforced by the results of the single-level model, which does not show significant differences between 32 and 64 channels in the evaluation of the FlyingThings3D stage. It may be plausible that, there, the increased number of feature channels helps in detecting out-of-range (i.e., out-of-domain) matches. While the benefits of a larger number of feature channels are visible, it is not clear whether they can be attributed to the resulting higher capacity for information or to improvements in cost computation, as the dot product may be better at approximating the optimal

[4] Increasing the number of feature channels from pre-output layer to post-output layer provides the option for the dot product to incorporate more linear combinations of the pre-output-layer channels, which may result in a better cost function even though we do not provide additional information (we just make it more accessible to the dot product). We could now compute the combinatorial maximum of such combinations, but notable gains are quite likely exhausted before that, hence the somewhat loosely formulated “beyond a certain point”.

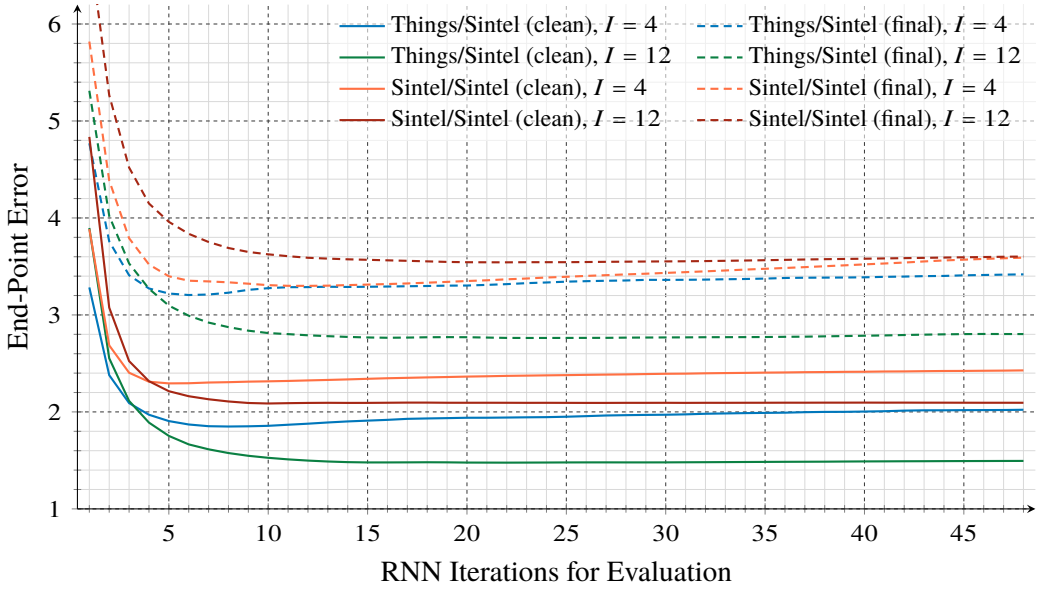

Figure 5.1: Impact of the number of recurrent iterations on RAFT during training and evaluation. We compare RAFT trained with both 4 (blue/orange) and 12 (green/red) recurrent iterations by evaluating using 1 to 48 iterations (horizontal axis). Runs are evaluated after the FlyingThings3D (blue/green) and after the Sintel (orange/red) training stages on both "clean" (solid) and "final" (dashed) Sintel datasets (using the full Sintel dataset after training on FlyingThings3D and the MaskFlowNet validation split after training on the MaskFlowNet Sintel training split).

cost function with a larger number of inputs. Likely, both play a role. Somewhat anomalous, the single-level model results on the Sintel "clean" split improve with a reduction in feature channels, with best scores for 64 channels. We believe that these results may not be fully representative for evaluation of channels alone, as a considerable portion (about 17%) of the ground-truth flow of this validation split is outside this model's receptive field.

Refinement

For the same reason that we reduce the number of feature channels, namely computational and (more importantly) memory requirements, we also reduce the number of recurrent iterations during training, compromising on quality through the refinement capabilities of the model. This trade-off can be chosen independently for training and evaluation, potentially allowing for higher accuracy during evaluation as, for that, memory usage does not grow with the number of iterations (cf. Section 3.3.4). We provide a separate investigation of this in Figure 5.1 by training RAFT once with four and once with twelve iterations (representing the first and second entry in Table 5.4) and evaluating both runs over a varying number of iterations, ranging from one to 48. For our results in Tables 5.1 to 5.4, we keep the number of training and evaluation iterations equal. Note that increasing the number of recurrent iterations can also increase the model's receptive field across all iterations, however, finding the best feature correspondence when it is outside the initially considered ones then depends on a sequence moving that match into the per-iteration receptive field.

Parameters					Chairs2	Things		Sintel	
Levels	*Rng.*	*Chn.*	*Iter.*	*BN*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
{3, 4, 5, 6}	4	256	4	✗	1.2832	1.9707	3.2727	2.3113	3.5204
{3, 4, 5, 6}	4	256	12	✗	1.1468	1.4979	2.7898	2.0929	3.5893
{3, 4, 5, 6}	4	256	4	✓	1.2794	2.0768	3.5011	2.3124	3.3398

Table 5.4: RAFT ablations for analysis of recurrent refinement and batch normalization. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including the specific cost volume levels (*Levels*), lookup operator/displacement candidate range in each direction (*Rng.*), number of channels in the feature representation (*Chn.*), number of recurrent iterations (*Iter.*), and whether batch normalization has been employed (*BN*).

It is therefore questionable whether this aspect provides any true benefit. As this extension of the receptive field is, however, difficult to separate from the pure refinement aspect, we do not evaluate it separately.

Table 5.4 shows clear improvements when increasing the number of iterations from four to twelve across all training stages and evaluation sets, with the notable exception of Sintel/Sintel (final). Figure 5.1 shows more details, comparing the impact of different numbers of recurrent iterations during evaluation. For a sufficiently large number of iterations during evaluation (larger than or equal to four), the larger number of iterations during training shows (with the already mentioned exception) clear benefits.

We can additionally see some interesting differences between four and twelve training iterations: In particular, the models trained with four iterations produce better initial results (for less than four evaluation iterations) and seem to converge faster than their twelve iteration counterparts. However, we can also observe that the four iteration models exhibit some unexpected convergence behavior, showing best results initially and then converging to a slightly worse result. This can be especially observed for the FlyingThings3D stage, but also for the Sintel stage. Together, this could potentially be interpreted as the model learning to overshoot during training (i.e., maybe in some way predicting the next refinement steps to estimate a more optimal result), leading to worse results when evaluated with a larger number of iterations. The twelve iteration models do not seem to exhibit this behavior, or at least not with the same magnitude. It therefore seems that a certain number of training iterations is required to obtain this kind of stability, and that four iterations may be too little.

Batch Normalization

Lastly, we compare training RAFT with and without batch normalization. The original training strategy by Teed and Deng [TD20] employs batch normalization on their first training stage (FlyingChairs) and freezes it on the later ones. Due to a lower batch size, we decided to disable batch normalization across all stages. We therefore investigate whether our concerns are valid for our training strategy when looking at RAFT alone (performing a similar investigation for selected RAFT+DICL approaches later).

As can be observed in Table 5.4, training with batch normalization leads to mixed results when compared to the model without it. For the FlyingChairs2 stage as well as Sintel/Sintel (final), results improve. For the FlyingThings3D stage, however, results with batch normalization enabled are notably worse. This may be an indication as to why Teed and Deng [TD20] decided to freeze batch normalization after the first stage. A further evaluation following the original RAFT scheme may prove beneficial for finding an improved training strategy. We will keep batch normalization disabled by default due to these mixed results in addition to the model differences with respect to our RAFT+DICL approaches, but will also investigate its use later for a subset of these models.

Summary

With the experiments performed in this section, we aimed to gain insight into RAFT and its success. We have tried to identify the impact of important parameters, especially in regard to the trade-offs required to make our RAFT+DICL combination feasible. We found that cost volume levels not only serve to increase the receptive range of the model, but may also help with regularization during the recurrent cost volume decoding. This seems to lead to improved optical flow even when virtually all ground-truth feature matches are already in the receptive field. Based on dataset statistics, three cost volume levels are sufficient to adequately capture the large majority of ground-truth matches, but adding a fourth level can nevertheless lead to small gains.

The quality of the cost values themselves depends on the number of feature channels. It has been observed that a certain minimum number of channels is required to appropriately reflect the visual complexity of the input data inside the model. This minimum has shown to be higher on the "final" datasets, containing imagery that is more visually challenging due to effects like motion blur and depth of field. As we reduce the number of feature channels to 32 to make our RAFT+DICL approaches feasible, this could pose as a bottleneck. However, gains related to an increased number of feature channels could also be related to the dot product being able to better model the "true" cost function when made available more input dimensions.

Recurrent refinement similarly seems to require a certain minimum of iterations to show its full potential. We found that, with four iterations only, the model seems to overshoot and reach its best results with a smaller number of evaluation iterations, however converges to an overall worse result when applying further iterations. The model trained with twelve iterations does not exhibit this behavior, at least not in that magnitude, and also leads to better results in general.

Batch normalization does show benefits when training on FlyingChairs2. Results are, however, mixed beyond that. Overall performance is worse when evaluating the FlyingThings3D stage on Sintel and better again when evaluating the Sintel stage on the Sintel "final" MaskFlowNet split.

5.1.2 DICL

For our experimental analysis of DICL, we focus on the parameters and aspects relevant for its integration into RAFT, following the approach outlined in Section 3.3, as well as our training strategy, largely derived from the one employed by RAFT [TD20]. This entails the employed coarse-to-fine levels, the norm used in the loss function, whether loss computation is restricted to ground-truth flow within the receptive field of the corresponding level, and the use of batch normalization. Results in terms of endpoint error are presented in Table 5.5.

Parameters				Chairs2	Things		Sintel	
Levels	L^k	*Restr.*	*BN*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
$\{3,4,5,6\}$	L^2	✗	✓	2.3952	2.8113	4.2636	4.0811	5.8047
$\{2,3,4,5,6\}$	L^2	✗	✓	2.3271	2.5393	4.0304	3.3712	5.4540
$\{2,3,4,5,6\}$	L^2	✗	✗	1.7794	2.3704	3.6626	3.2897	4.6445
$\{2,3,4,5,6\}$	L^2	✓	✓	2.1976	2.2221	3.7913	3.4950	5.4286
$\{2,3,4,5,6\}$	L^2	✓	✗	1.7533	2.3313	3.6828	3.0999	5.1157
$\{2,3,4,5,6\}$	L^1	✓	✗	1.8838	2.3891	3.8790	3.3158	5.0695

Table 5.5: DICL baselines and ablations. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including number of coarse-to-fine levels (*Levels*), loss norm (L^k), whether the loss has been restricted to the model's receptive field (*Restr.*), and whether batch normalization has been employed (*BN*).

Coarse-to-Fine Levels

In contrast to our investigation of RAFT, where we dropped coarser cost volume levels in an attempt to measure the impact of the receptive field size, we will look at the impact of the finest cost volume level for DICL. The reasoning behind this is that the finest level of RAFT, and therefore our combined approach, uses a coarser spatial resolution: Given level resolutions as $H/2^m \times W/2^m$ with original input resolution $H \times W$, we (as well as RAFT) employ $m = 3$ for the finest cost volume level, whereas DICL uses $m = 2$ for the finest coarse-to-fine level. Dropping the level with $m = 2$ sets the finest feature resolution of DICL equal to RAFT and our combined approach, and may thus give us some valuable insights. In particular, we expect a loss of detail, expressed both in terms of smaller objects and accuracy of motion boundaries. Note that RAFT employs an upsampling network for the final flow estimate, thus producing more accurate boundaries in comparison to DICL, which relies on bilinear upsampling. Nevertheless, the amount of detail in terms of small objects captured should remain mostly equivalent when given the same finest feature resolution.

Results in Table 5.5 show that there is comparatively little impact on the FlyingChairs2 dataset. We believe that this is likely due to its simplified structure, relying on fairly simple shapes (compared to other datasets), leading to simpler motion boundaries, and employing only two-dimensional affine transformations to express flow (cf. Section 2.2). There is likely not much detail to be lost, which could explain these results. In the evaluations of the FlyingThings3D and Sintel stages, we can identify a tendency showing a relatively smaller drop in performance when removing the $m = 2$ level in the "final" evaluation sets than can be observed in the "clean" sets. We argue that this can likely be attributed to the structure of the "final" datasets: Due to the visual effects applied to those, such as motion blur and depth of field, smaller objects may already be hidden or harder to distinguish. Therefore, they may already not be as relevant in the results of the "final" sets compared to the "clean" sets, hence limiting the ability to detect them has a lesser impact. We believe that this also shows that having finer levels may not be as important as one may initially think, and that employing an upsampling module similar to RAFT may be a good compromise to not miss out on accurate motion boundaries, especially considering its benefit-cost ratio.

Loss Norm

Differences between RAFT and DICL are not limited to the architecture only, but also include the loss function. One such particular difference is the employed norm: DICL uses the euclidean L^2 norm, whereas RAFT uses the L^1 norm. In general, the L^1 norm is less susceptible to large outliers, such as pixels with ground-truth flow outside the receptive field of the model. On the other hand, the L^2 norm focuses more on large errors, which may not always be outliers and may arguably have a higher potential for comparably easier achievable improvements.

Table 5.5 (5th and 6th row) shows that DICL trained with the L^2 norm provides consistently better results than it being trained with the L^1 norm, except for Sintel/Sintel (final). One possible explanation for this can again be found in dataset statistics: On the Sintel MaskFlowNet training split, 90% of ground-truth flow is below a magnitude of 33 px, whereas on the FlyingThings3D training set, the same portion is below a significantly higher magnitude of 79 px. On the Sintel MaskFlowNet validation split, this 90% threshold is at 45 px. We therefore believe that during the FlyingThings3D stage, the model focuses adequately on both higher and lower magnitude flow due to the distribution, ultimately leading to the obtained result when testing on the full Sintel dataset. However, when training on the Sintel MaskFlowNet split, focus turns towards lower magnitude flows. This may happen as a combined result of both dataset statistics and the L^1 norm behavior. Therefore, we theorize that after training on Sintel the model may focus more on smaller motions, leading to improved accuracy on the "final" set, where retaining small details may be more difficult. In turn, this could then also lead to performance drops on the "clean" set (due to potential relative higher importance of larger motions), where these details are unaltered by visual effects.

A significant aspect to note here is that we restrict the loss computation to the receptive field of the model, i.e., do not include true outliers in the loss. Therefore, this, in essence, compares preference distributions over the magnitudes of motion. Results (with the noted exception) show that it is generally advisable to choose the L^2 norm for DICL. This further indicates that RAFT could benefit from a similar approach, relying on the L^2 norm instead of the L^1 one. We will investigate the impact of including outliers in the loss next.

Loss Restriction

The DICL loss is computed by accumulating one loss term per level (cf. Section 3.2.2). These per-level terms are computed only over ground-truth flow contained inside the respective receptive field of the level (determined by the current and coarser coarse-to-fine levels), meaning true outliers are not considered in the loss and simply ignored. RAFT does not make such a distinction.

Table 5.5 (comparing rows two and four as well as three and five) shows general improvements through restriction of the loss. Somewhat notably, however, the best result of the Sintel/Sintel (final) evaluation is achieved without loss restriction (row three, compared to row five) and with a substantial margin. A similar reversal of gains can be observed for the same pair of rows, again on the Sintel "final" set, this time however for Things/Sintel (final), as well as for Sintel/Sintel (clean) in rows two and four. We therefore believe that, while loss restriction improves results in most cases, there may also be a regulatory character to including outliers. Especially, considering that this behavior seems to be more prominent when batch normalization is disabled, which itself normally provides a regulatory aspect.

Batch Normalization

Lastly, we again consider batch normalization. A significant difference between our training strategy and the original DICL training strategy proposed by Wang et al. [WZD+20] is the batch size, which is of considerable importance for batch normalization. In the original training strategy, a total batch size of 64 is used, with eight samples per GPU. Similar to our implementation, the batch normalization layer is not shared (i.e., synchronized) across GPUs, therefore resulting in an effective batch size of eight in regard to that. This is still notably larger than our batch size of six for FlyingChairs2 and four for all other training stages. We train all evaluated methods with batch normalization enabled on a single GPU to not further reduce the effective batch size.

Results in Table 5.5 (comparing rows two and three as well as four and five) show clear and substantial improvements when disabling batch normalization. A single exception to this can be found in the Things/Sintel (clean) evaluation when training with loss restriction. As hinted to above, we believe that the effective batch size of our training strategy may be too small for batch normalization to be successful. Other normalization strategies, such as group normalization [WH18] or weight standardization [QWL+20] may produce better results.

Summary

In conclusion, we found that having a fine spatial resolution may not be as important as one may first think. While there is a loss of detail leading to an increased endpoint error when dropping the finest DICL level, this loss seems smaller on the "final" sets containing visual effects. We therefore believe that this indicates that an upsampling strategy as employed by RAFT is a suitable alternative to a higher (maximum) spatial resolution, leading to accurate motion boundaries with less computational and memory costs, therefore providing a good compromise.

We found that using the L^2 norm works better for training DICL than the L^1 norm (used by RAFT), at least when not including outliers outside the receptive field of the model in the loss term. Due to these results, a similar investigation may be beneficial for RAFT. Further, results show that, while including outliers may have a small regulatory character, restricting the loss to the (per-level) receptive field generally leads to improvements. Lastly, we found that disabling batch normalization is beneficial for our training strategy, likely due to the reduced batch size.

One notable difference to RAFT that we did not investigate in this subsection is the per-level receptive field range for which cost values are computed. In RAFT, this is four by default, whereas DICL uses three. While this again heightens the memory and compute requirements, a compromise without the finest coarse-to-fine level could be tested.

5.2 Single-Level

With the insights from the previous section, we can now briefly look at our first and simplest approach for combining RAFT and DICL, using only a single cost volume level as described in Section 4.1. As before, we follow the training strategy detailed in Section 4.5. We train using a lookup operator range of four by default, but also experiment with a lookup operator size of seven. All other parameters of our method are left at their previously elaborated defaults. For fair

Method	Parameters		Chairs2	Things		Sintel	
	Chn.	*Rng.*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
DICL (baseline)	32	3	1.7533	2.3313	3.6828	3.0999	5.1157
RAFT (baseline)	256	4	1.2832	1.9707	3.2727	2.3113	3.5204
RAFT (SL)	256	4	1.4209	2.3015	**3.5155**	2.9718	4.5847
RAFT (SL)	64	4	1.4221	2.3531	3.6052	2.8499	4.8030
RAFT (SL)	32	4	1.4823	2.3525	3.6665	2.9657	4.7662
RAFT+DICL (SL)	32	4	**1.4103**	**2.2605**	3.6062	**2.7448**	**4.2528**
RAFT (SL)	256	7	**1.3301**	**2.1118**	**3.4481**	**2.5551**	4.4555
RAFT+DICL (SL)	32	7	1.3865	2.1374	3.4678	2.5739	**4.1015**

Table 5.6: RAFT+DICL on a single level (SL). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including number of channels in the feature representation (*Chn.*) and lookup operator/displacement candidate range in each direction (*Rng.*). Baselines not directly comparable are shown in gray.

comparison, we use a single-level version of RAFT, which is equal to RAFT with the exception that no cost pooling is taking place (i.e., only a single cost volume level is being used). We further add ablations thereof for reference, using 64 and 32 feature channels.

5.2.1 Base Results

Results in Table 5.6 show promise: For the model trained with a lookup range of four, we can see clear improvements on all "clean" datasets, including FlyingChairs2. The Sintel/Sintel (final) evaluation equally shows improvements over its baselines, however, Things/Sintel (final) does not show any gains in performance over the single-level RAFT version. Looking at the models trained with a lookup operator range of seven, results of our combined approach are worse in relation. The only improvement is achieved on Sintel/Sintel (final). Otherwise, however, results are somewhat close to the respective single-level version of RAFT, with the largest difference of the average endpoint error being 0.05 px.

Considering that we, in comparison, obtain good results on the Sintel "final" evaluation sets (especially after the Sintel training stage), this approach does not seem to express the same limiting behavior as our investigation into restricting the number of feature channels for RAFT in Section 5.1.1 shows. We therefore believe that RAFT+DICL approaches are likely not inherently hindered from producing good results by their low number of feature channels. Models may still be limited in a less severe way, however, as such a limitation may be expressed differently due to the replacement of the dot product as cost function. The somewhat anomalous result of Things/Sintel (final) could be an indication of this.

Method	Parameters		Eval.	Chairs2	Things		Sintel	
	Chn.	*Rng.*	**u**-*max*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (baseline)	256	4	32	**0.7693**	**1.2684**	2.2611	**1.3385**	2.2019
RAFT (SL)	256	4	32	0.8343	1.4233	**2.2456**	1.4537	2.1696
RAFT (SL)	64	4	32	0.8349	1.4538	2.3322	1.5373	2.2370
RAFT (SL)	32	4	32	0.8740	1.4676	2.3575	1.5983	2.2947
RAFT+DICL (SL)	32	4	32	0.8248	1.4116	2.2853	1.5548	**2.1607**
RAFT (baseline)	256	4	56	**0.9266**	**1.3242**	2.4023	**1.1603**	2.0279
RAFT (SL)	256	7	56	0.9577	1.4611	2.4442	1.2326	2.1019
RAFT+DICL (SL)	32	7	56	0.9885	1.4271	**2.3877**	1.3125	**2.0172**

Table 5.7: Results for the single-level (SL) RAFT+DICL method considering only flow inside the receptive field. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including number of channels in the feature representation (*Chn.*) and lookup operator/displacement candidate range in each direction (*Rng.*). The endpoint error is computed only for pixels having a ground flow magnitude below or equal to the given **u**-*max* (in pixel).

5.2.2 Results for Flow Inside the Receptive Field

Comparing models across different lookup ranges, we can see that, as expected from our studies in Section 5.1.1, the size of the receptive field is a major limitation. We therefore reevaluate our models and restrict endpoint error computation to ground-truth flow inside the receptive field. For models with a lookup operator range of four, this equates to flow with a ground-truth magnitude below or equal to 32 px and for models with a lookup operator range of seven to flow with a ground-truth magnitude below or equal to 56 px. We now additionally provide equally restricted evaluations for a four-level (i.e., standard baseline) RAFT version for comparison.

Results are shown in Table 5.7 and are somewhat more mixed. Comparing against the four-level RAFT baseline again shows that, even when restricting flow to the single-level receptive field, having multiple cost volume levels is a benefit. When comparing only the single-level approaches, our RAFT+DICL models still achieve good results, however error values are now notably closer for versions with a lookup range of four. For models with a lookup range of seven, the RAFT+DICL approach now outperforms its single-level RAFT baseline on more evaluation sets compared to before, although margins are again small.

Comparing results for models with lookup range of four between Table 5.6 and Table 5.7 shows higher margins of improvement on the unrestricted evaluations, which could indicate that the RAFT+DICL models may be better at extrapolating outside their receptive field or may detect such outliers better, at least in some cases. Results for models with a lookup range of seven, however, do not support such a theory outright. Comparing results for "clean" and "final" datasets over all evaluations, especially in Table 5.7 as well as for the Sintel training stage, may hint at the RAFT+DICL approach performing, in relation, better at more complex matches.

Lastly, we want to note that we did not restrict training to the subset of ground-truth flow inside the receptive field of the model. On the FlyingThings3D training set, a non-negligible portion of the training data is outside the (initial) receptive field: About 36% of the ground-truth flow

is larger than 32 px, whereas on FlyingChairs2 it is only 8% and on Sintel 11%. As seen in our experiments regarding DICL in Section 5.1.2, this can impact model performance. Therefore, evaluations restricting the training loss itself to pixels with small enough ground-truth flow only may be beneficial. Given the overall mostly promising results, however, we instead chose to continue our experiments by expanding the receptive field via multi-level and coarse-to-fine strategies, which we discuss in the two next sections.

5.3 Multi-Level

Our first attempt at increasing the area of the receptive field follows the strategy taken by RAFT, using multiple cost volume levels as described in Section 4.2. We can therefore directly compare these models with the corresponding RAFT variations employing the same cost volume levels. For our multi-level RAFT+DICL models, we chose to evaluate two types of feature encoders: First, we evaluated an average-pooling encoder that simply downsamples the finer feature embeddings to create the coarser ones. Note that this relates more directly to the original RAFT method and would resemble a reordered version thereof if we would choose to compute costs via the dot product (as discussed in its derivation in Section 3.3).

As an alternative to this, we also experiment with a somewhat more classical encoder, relying on a CNN to create the coarser embeddings. Due to the asymmetric nature of the (reordered) RAFT feature embeddings, it consists of two sub-encoders, one per input image: The encoder for the first frame retains the spatial size after the first encoding stage via dilated convolutions, whereas the encoder for the second frame downsamples its input via strided ones (see Section 4.2 for details). We therefore cannot share all weights for both input image paths in the encoder. For the pooling encoder, we share the weights of DICL matching network across all levels, whereas for the CNN-based one we do not. Our rationale behind this is that the pooling encoder retains the overall channel structure across all levels whereas the CNN-based encoder may choose to create a different encoding structure per level. In both cases, we use a single combined DAP layer over all levels, initialized to be an identity mapping.

All RAFT+DICL models shown in this section have been trained with a base learning rate of 1×10^{-4} during the preparatory epoch. Other parameters follow our training strategy presented in Section 4.5.

5.3.1 Base Results

As Table 5.8 shows, our multi-level RAFT+DICL models were unable to produce good results. Our approaches performed consistently worse than their respective baselines, with the four level RAFT+DICL model (using the CNN-based encoder) even having a worse endpoint error than the two-level RAFT baseline across all datasets, except for the Sintel/Sintel (final) evaluation. Apart from this, the (two-level) version with the CNN-based encoder performed notably better on the first two training stages (FlyingChairs2 and FlyingThings3D) than the pooling one. This is reversed on the last training stage (Sintel).

Method	Parameters		Chairs2	Things		Sintel	
	Levels	*Enc.*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (baseline)	{3, 4, 5, 6}	pool	**1.2832**	**1.9707**	**3.2727**	**2.3113**	**3.5204**
RAFT+DICL (ML)	{3, 4, 5, 6}	CNN	1.5099	2.3272	3.4719	2.8434	4.0606
RAFT (baseline)	{3, 4}	pool	**1.3118**	**2.0402**	**3.3270**	**2.5337**	**4.2979**
RAFT+DICL (ML)	{3, 4}	pool	1.6012	2.4013	3.7188	3.0117	4.3087
RAFT+DICL (ML)	{3, 4}	CNN	1.5512	2.2992	3.4465	3.1343	4.4985
RAFT+DICL (SL)	{3}	N/A	1.4103	2.2605	3.6062	2.7448	4.2528

Table 5.8: RAFT+DICL with multiple levels (ML). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including cost volume levels (*Levels*) and feature encoder type (*Enc.*).

Lastly, we note that the single-level RAFT+DICL model generally out-performs its multi-level counterparts, except for the Sintel "final" test sets. This hints at a fundamental problem with the model, as we would expect the multi-level models to improve over the single-level one. We can also see this by comparing the two- and four-level CNN-based versions after the FlyingThings3D training stage, where the two-level model out-performs the four-level one. We will investigate this in Chapter 6 in more detail.

5.3.2 Results for Smaller Flow Magnitudes

As before, we again evaluate our trained approaches on subsets of the ground-truth data containing only flow with a magnitude below or equal to 128 px, 64 px, or 32 px, respectively. Results for this are shown in Table 5.9. Here, we can see that the multi-level RAFT+DICL models are capable of improving upon the single-level version in some scenarios. In particular, when restricting evaluation to ground-truth flow below or equal to 128 px, both multi-level methods are able to outperform the single-level method on the FlyingThings3D stage. For the Sintel stage, the multi-level methods outperform the single-level method when restricting the evaluation to flow below or equal to 32 px. For flow below 64 px (and somewhat more so for flow below 32 px), the RAFT+DICL models can even outperform their RAFT baselines in some cases and get (relatively) closer in others. Most notably, for the Things/Sintel (final) evaluation, hinting at a potentially greater capability for generalization. This also again indicates (as seen before in the evaluation of the single-level model) that the DICL-based cost computation likely benefits more in respect to complex feature matches than it does in respect to simpler ones.

Apart from this, we believe that the relative improvement on small (ground-truth) flow compared to large flow also again shows that multiple levels are beneficial for regularization. This seems to be somewhat extreme for the RAFT+DICL models, especially below 32 px. Here, the CNN-based encoder performs better than the pooling-based one on the Sintel MaskFlowNet validation split (the order is reversed for larger flows), even though both encoders share the same structure up to the first feature (and therefore also cost volume) level (which has a receptive field range of 32 px). We theorize that the CNN-based encoder provides additional information on coarser levels not encoded

Method	Parameters		Eval.	Chairs2	Things		Sintel	
	Lvl.	*Enc.*	**u**-*max*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (baseline)	4	pool	128	**1.2609**	**1.5377**	**2.7865**	**1.5730**	**2.7339**
RAFT (baseline)	2	pool	128	1.2890	1.6102	2.8108	1.6746	3.3051
RAFT+DICL (ML)	4	CNN	128	1.4825	1.7744	2.8511	1.9116	2.9826
RAFT+DICL (ML)	2	CNN	128	1.5220	1.7935	2.8436	2.1349	3.3514
RAFT+DICL (ML)	2	pool	128	1.5736	1.9000	3.1438	2.0042	3.2830
RAFT+DICL (SL)	1	N/A	128	1.3834	1.8002	3.1026	1.8938	3.2912
RAFT (baseline)	4	pool	64	**0.9836**	**1.3506**	2.4567	**1.2041**	**2.0859**
RAFT (baseline)	2	pool	64	0.9923	1.4408	2.4897	1.2579	2.1109
RAFT+DICL (ML)	4	CNN	64	1.1575	1.4288	**2.3774**	1.4242	2.1452
RAFT+DICL (ML)	2	CNN	64	1.1700	1.4793	2.3870	1.4278	2.2723
RAFT+DICL (ML)	2	pool	64	1.2135	1.6434	2.6706	1.4141	2.3115
RAFT+DICL (SL)	1	N/A	64	1.0510	1.5372	2.6749	1.3889	2.1743
RAFT (baseline)	4	pool	32	**0.7693**	**1.2684**	2.2611	**1.3385**	2.2019
RAFT (baseline)	2	pool	32	0.7790	1.3317	2.2178	1.4574	2.0902
RAFT+DICL (ML)	4	CNN	32	0.9062	1.2870	**2.0324**	1.4886	**2.0064**
RAFT+DICL (ML)	2	CNN	32	0.9186	1.3356	2.0374	1.4662	2.2748
RAFT+DICL (ML)	2	pool	32	0.9422	1.5147	2.3269	1.5347	2.3632
RAFT+DICL (SL)	1	N/A	32	0.8248	1.4116	2.2853	1.5548	2.1607

Table 5.9: Results for the multi-level RAFT+DICL method considering only ground-truth flow below or equal to the respective magnitude **u**-*max* (in pixel). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective model parameters, including cost volume levels (*Levels*) and feature encoder type (*Enc.*).

in the finest feature level, which is then used in the recurrent refinement network for resolving matches. Considering the (relatively) worse performance on larger flows, this regularization even seems to take a larger functional aspect in the network than extending its receptive range.

5.3.3 A Note on Feasibility

Unfortunately, our multi-level models exhibit another significant problem besides the suboptimal results: High memory requirements. Memory requirements scale linearly with the number of levels used (cf. Section 3.3.4) and the RAFT+DICL cost computation makes up for a considerable amount of the overall memory usage of these approaches. This means that, while the single-level model could easily be trained on a single NVIDIA RTX 3090 GPU with 24 GiB video memory using our training strategy presented in Section 4.5, we need two such GPUs for the two-level model and four for the four-level model.

For the four-level model, this also means that, with the batch size of four on the FlyingThings3D and Sintel stages, we process one sample per GPU, which makes batch normalization essentially unusable. Further, this makes increasing the number of training iterations difficult, as the memory requirements of the multi-level approach also scale linearly in respect to these (note again that

testing iterations do not impact memory requirements). We therefore limited ourselves to the few experiments shown within this thesis and consider these approaches to be borderline infeasible (given the hardware available to us at the time of writing).

5.4 Coarse-to-Fine

The high memory requirements of the multi-level approach make it difficult to investigate its problems and scale up the approach for larger images, a larger receptive field, or better refinement through more training iterations. Our analysis of RAFT in Section 5.1.1 has shown the latter two to be important for producing good results. In particular, a sufficient receptive field size is required to adequately capture the majority of ground-truth motion in the common evaluation datasets and ultimately also to be useful in real-world applications.

We therefore need another strategy for extending the receptive field. More specifically, one that scales sub-linearly with a quadratic size increase, such as an extension by adding coarser spatial levels. This is given in the classical coarse-to-fine scheme. The reason for the linear scaling of our multi-level approach is the constant resolution of the first input feature frame across cost levels (cf. Section 3.3). Flow is produced at this single level, due to which first-frame features and costs need to be computed at the same spatial resolution.

By downsampling the first-frame features, therefore creating a spatial feature pyramid, and reordering our approach we obtain the coarse-to-fine approach described in Section 4.3, resolving our memory scaling issue for the receptive field size. Crucially, this coarse-to-fine approach only uses a single cost volume level per coarse-to-fine level. While, in theory, hybrid approaches are possible using multiple cost volume levels per coarse-to-fine level, this would again heighten memory requirements (most notably on the finest level). Due to the significant difference between the regular (multi-level) RAFT model and our coarse-to-fine approach, we use corresponding coarse-to-fine RAFT versions as baselines for comparison. These baseline models equally use only a single cost volume level (however, as the dot product can be implemented more efficiently, here multiple levels could be used without forcing trade-offs in other places).

By default, all evaluated models share weights of the recurrent refinement network across all coarse-to-fine levels, whereas DICL module weights are not shared. We assume that the cost representation is universal and therefore in large equal on each level, whereas the feature representation may be different. This, in essence, allows for the model to learn a dedicated cost function per level. All models have been trained following the training strategy detailed in Section 4.5, with RAFT+DICL models using a base learning rate of 1×10^{-4}, 5×10^{-5}, or 5×10^{-6} for the two-, three- and four-level coarse-to-fine variants, respectively, during the preparatory epoch to avoid initial stability issues.

5.4.1 Base Results

Results are presented in Table 5.10. Our coarse-to-fine RAFT+DICL approaches perform again generally worse than their respective RAFT-based baselines. However, endpoint errors are now significantly closer than with the multi-level methods, evaluated in the previous section. We can even observe some improvements over said baselines on the Sintel stage, specifically in the Sintel/Sintel (final) results with the two-level coarse-to-fine model and in the Sintel/Sintel (clean) results for the

Method	Parameters	Chairs2	Things		Sintel	
	Levels	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (baseline)	{3, 4}	1.3118	2.0402	3.3270	2.5337	4.2979
RAFT (baseline)	{3, 4, 5}	1.2977	1.9972	3.3378	2.2981	3.5269
RAFT (baseline)	{3, 4, 5, 6}	1.2832	1.9707	3.2727	2.3113	3.5204
RAFT (baseline, $I = 12$)	{3, 4, 5, 6}	1.1468	1.4979	2.7898	2.0929	3.5893
RAFT+DICL (SL)	{3}	1.4103	2.2605	3.6062	2.7448	4.2528
RAFT (CtF-2)	{3, 4}	**1.1731**	**1.7454**	**3.0181**	**2.1083**	3.9801
RAFT+DICL (CtF-2)	{3, 4}	1.3773	1.8723	3.1438	2.2506	**3.6760**
RAFT (CtF-3)	{3, 4, 5}	**1.2092**	**1.5938**	**3.0188**	2.1514	**3.1364**
RAFT+DICL (CtF-3)	{3, 4, 5}	1.2745	1.7436	3.0417	**2.0666**	3.2351
RAFT (CtF-4)	{3, 4, 5, 6}	**1.2723**	**1.6782**	**3.0476**	**2.0644**	**3.1022**
RAFT+DICL (CtF-4)	{3, 4, 5, 6}	1.3097	1.7958	3.0890	2.2874	3.4266

Table 5.10: RAFT+DICL in coarse-to-fine (CtF) arrangement. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective coarse-to-fine levels (*Levels*). Baselines not directly comparable are shown in gray, including a standard RAFT model with $I = 12$ training and evaluation iterations (default is $I = 4$).

three-level one. Further, results now also improve as expected when adding more levels, at least up to (and including) three. The four-level model performs worse than the three-level one, however, all coarse-to-fine methods perform better than the RAFT+DICL single-level baseline. Considering that these models additionally require less memory than their multi-level counterparts, we believe that they form a good foundation for further experiments and attempted improvements.

Most notably, we find that the coarse-to-fine methods can outperform standard RAFT models (using a hierarchical cost volume). These results are, however, not easily comparable as their numbers of training iterations differ (e.g., 4 vs. 4+3+3). As indicated by the model trained and evaluated with 12 iterations, this difference could very well be responsible for the improved results, but the three- and four-level RAFT coarse-to-fine methods still outperform that model in the Sintel/Sintel (final) evaluation. We therefore believe that this may be a topic of interest for further research, which, however, is outside the scope of this thesis.

5.4.2 Results for Smaller Flow Magnitudes

To gain more insights, we again evaluate our methods on pixels with ground-truth flow below or equal to a certain magnitude. In particular, we find that, for better gauging the model's actual accuracy, it is important to investigate results on ground-truth flow fully contained inside the model's receptive field, neglecting outliers. This shows that, for both the two-level (Table 5.11) and three-level (Table 5.12) coarse-to-fine models, improvements over their respective baselines on the unrestricted evaluation sets (Sintel/Sintel-final for the two-level model and Sintel/Sintel-clean for the three-level one) disappear as soon as evaluation is restricted to the respective receptive field of the coarsest level (64 px for two levels, 128 px for three levels). On the Sintel stage, the

Method	Eval.	Chairs2	Things		Sintel	
	u-*max*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-2)	∞	**1.1731**	**1.7454**	**3.0181**	**2.1083**	3.9801
RAFT+DICL (CtF-2)	∞	1.3773	1.8723	3.1438	2.2506	**3.6760**
RAFT (CtF-2)	128	**1.1501**	**1.3582**	**2.4597**	**1.4215**	3.0242
RAFT+DICL (CtF-2)	128	1.3505	1.4563	2.6077	1.5318	**2.8173**
RAFT (CtF-2)	64	**0.8705**	**1.1773**	**2.1410**	**1.0799**	**1.8635**
RAFT+DICL (CtF-2)	64	1.0248	1.2410	2.2888	1.1874	1.9209
RAFT (CtF-2)	32	**0.6669**	**1.1054**	**1.8705**	**1.2273**	**1.9069**
RAFT+DICL (CtF-2)	32	0.7870	1.1633	2.0399	1.3884	2.0884

Table 5.11: Results for the two-level coarse-to-fine (CtF-2) RAFT+DICL method, considering only ground-truth flow below or equal to the respective magnitude **u**-*max* (in pixel). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

Method	Eval.	Chairs2	Things		Sintel	
	u-*max*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-3)	∞	**1.2092**	**1.5938**	**3.0188**	2.1514	**3.1364**
RAFT+DICL (CtF-3)	∞	1.2745	1.7436	3.0417	**2.0666**	3.2351
RAFT (CtF-3)	256	**1.2092**	**1.4965**	**2.9058**	**1.8671**	**2.8768**
RAFT+DICL (CtF-3)	256	1.2745	1.6558	2.9245	1.8699	2.9680
RAFT (CtF-3)	128	**1.1849**	**1.2074**	**2.4010**	**1.4052**	**2.2633**
RAFT+DICL (CtF-3)	128	1.2497	1.3280	2.4066	1.5104	2.3605
RAFT (CtF-3)	64	**0.8906**	**1.0202**	**2.0425**	**1.0401**	**1.7129**
RAFT+DICL (CtF-3)	64	0.9394	1.1186	2.0453	1.1684	1.7877

Table 5.12: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method, considering only ground-truth flow below or equal to the respective magnitude **u**-*max* (in pixel). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

relative performance gap between our RAFT+DICL and the RAFT coarse-to-fine approaches seems to increase with lower maximum ground-truth flow magnitude, with our approaches ultimately performing worse across the board.

This could indicate that the RAFT+DICL approaches are better at extrapolation to data outside the receptive field, which we have also seen in our single-level evaluation (cf. Section 5.2.2). We believe, however, that such conclusions need to be considered with care and results outside the receptive field could also be up to chance, especially as other stages do not (consistently) express such behavior: The standard (i.e., "pure") cost volume, consisting of cost/correlation scores for specific displacement hypotheses, does not provide any means for the encoding of information outside the receptive field. Further, apart from cost computation, the RAFT and RAFT+DICL approaches are equal. Note that the bulk of DICL-based cost computation (specifically, the matching network) still happens in a manner invariant to displacements and only for the specified hypotheses,

Method	Parameters	Chairs2	Things		Sintel	
	Channels	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-3)	256	1.2092	1.5938	3.0188	2.1514	3.1364
RAFT (CtF-3)	64	**1.2610**	**1.6016**	3.0027	**2.0333**	**3.1963**
RAFT+DICL (CtF-3)	64	1.2667	1.7018	**2.9868**	2.0687	3.2501
RAFT (CtF-3)	32	1.2814	**1.6683**	3.0209	2.2031	3.3963
RAFT (CtF-3, FS)	32	1.2785	1.7076	**3.0063**	**2.0179**	3.3210
RAFT+DICL (CtF-3)	32	**1.2745**	1.7436	3.0417	2.0666	**3.2351**

Table 5.13: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method with 64 feature channels instead of 32. We additionally evaluate a reordered version of RAFT using feature sampling (FS) instead of cost sampling (default). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective number of channels in the feature representation (*Channels*).

therefore confining contained information to matches inside the receptive field (i.e., we see no possibility of extrapolation in the cost volume itself). It might, however, be possible that a higher quality cost volume as input to the recurrent refinement network could (together with the context network) allow for better extrapolation in some cases.

5.4.3 Impact of Feature Channels

Now that we have a basic assessment of the RAFT+DICL coarse-to-fine model performance, we can try to find out where its limitations come from and what potential starting points for improvements could be. We first investigate the impact of the number of channels in the feature representation. This is, apart from the method of cost/correlation computation and the required reordering due to it, the major difference between our combined approaches and the RAFT baseline. By reducing the number of feature channels in the RAFT coarse-to-fine models (or, equivalently, raising the number of channels in our RAFT+DICL methods), we can gauge whether there is some systemic problem in our approach, as we would at the very least expect our RAFT+DICL models to perform on par with their respective baseline using the same number of channels, if not better.

We can even go one step further in this validation and also compare the combined and baseline approaches with the reordered RAFT version derived in Section 3.3.4 (again with an equal number of channels). This model uses the same feature-sampling approach as our RAFT+DICL models, but instead of the DICL module uses the dot product to compute costs, making this their only difference. We evaluate approaches for both 32 and 64 feature channels. Note that we did not fully scale up the DICL matching network with the number of feature channels and only adapt its input layer. We will briefly experiment with different DICL matching network compositions in Section 6.3.

As shown in Table 5.13, results are generally fairly close, for both models with 32 and models with 64 feature channels. While there are some differences, most notably for the Things/Sintel (clean) evaluation (on both 32 and 64 channel models) as well as for Sintel/Sintel on the 64 channel models, results do not show a clear overall deficiency of the RAFT+DICL approach. Interestingly, the RAFT feature-sampling approach seems to be slightly closer to the RAFT+DICL approach in cases where there are larger differences in endpoint error. While the feature-sampling approach should

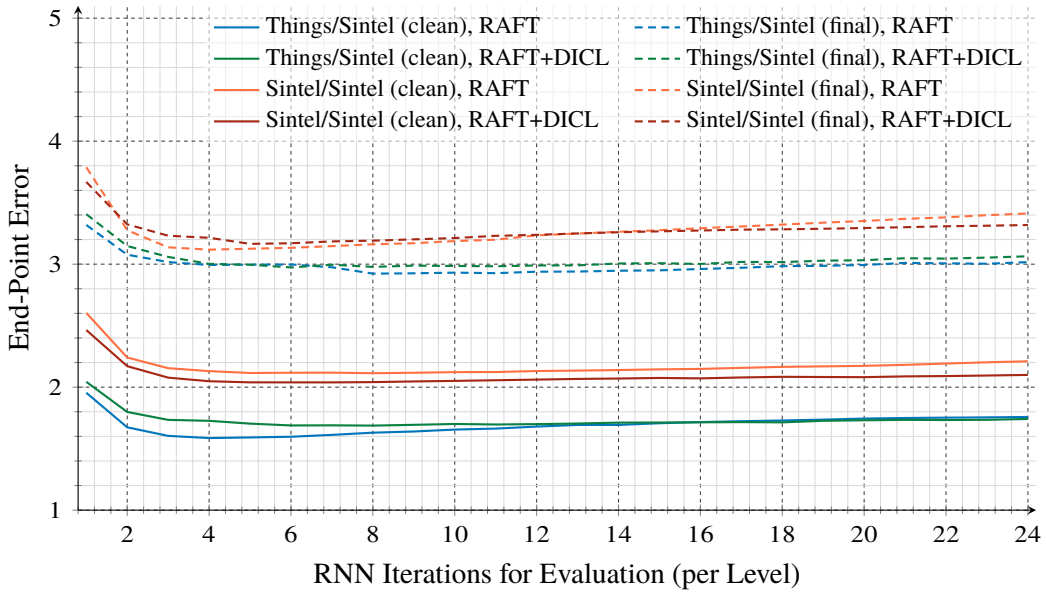

Figure 5.2: Impact of the number of recurrent iterations on three-level coarse-to-fine RAFT and RAFT+DICL during evaluation. We compare coarse-to-fine RAFT and RAFT+DICL approaches trained with 4, 3, and 3 iterations going from coarsest to finest level. Runs are evaluated after the FlyingThings3D (blue/green) and after the Sintel (orange/red) training stages, on both "clean" (solid) and "final" (dashed) Sintel datasets (using the full Sintel dataset after training on FlyingThings3D and the MaskFlowNet validation split after training on the MaskFlowNet Sintel training split).

theoretically be equal to the standard RAFT model (as proven in Section 3.3.4), small variances between rearrangements are to be expected, among other reasons due to floating-point operations not necessarily being associative.

Going from 32 to 64 channels shows larger gains for the RAFT baseline model. This may be due to the DICL matching network being conceived for 32 channels. Note that we only adapt its first layer to accommodate for the higher number of feature channels. Not fully scaling it up as well could lead to a bottleneck, due to which a larger number of feature channels may not have as large of an impact. This could further explain why results for our RAFT+DICL model stay (roughly) the same on the Sintel stage.

5.4.4 Impact of Iterations

We can now again analyze the endpoint error across different numbers of evaluation iterations. A visualization thereof is provided in Figure 5.2 (note that, here, we use the same number of evaluation iterations across all levels). We can see that the problems observed in Section 5.1.1, particularly in Figure 5.1, are still present: The best results are generally achieved when keeping the number of evaluation iterations similar or slightly higher than the number of training iterations, with the endpoint error getting notably worse when increasing iterations too much. Somewhat notably, however, this problem seems to be less severe on our coarse-to-fine RAFT+DICL approach than on the coarse-to-fine RAFT one.

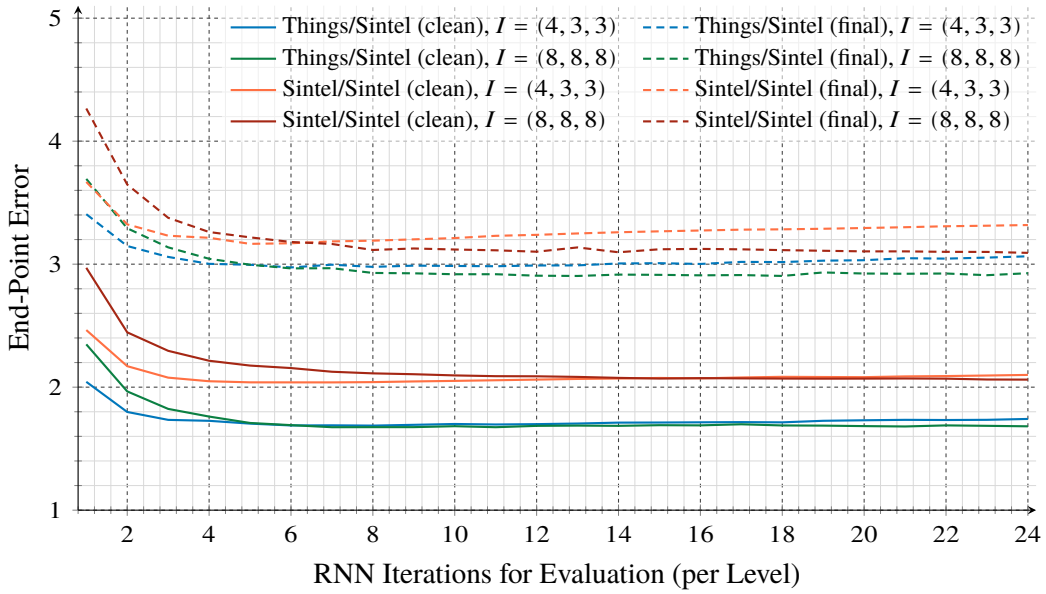

Figure 5.3: Impact of the number of recurrent iterations on three-level coarse-to-fine RAFT+DICL during training and, especially, evaluation. We compare coarse-to-fine RAFT+DICL approaches trained with 4, 3, and 3 iterations going from coarsest to finest level with approaches trained with 8 iterations on all levels. Runs are evaluated after the FlyingThings3D (blue/green) and after the Sintel (orange/red) training stages, on both "clean" (solid) and "final" (dashed) Sintel datasets (using the full Sintel dataset after training on FlyingThings3D and the MaskFlowNet validation split after training on the MaskFlowNet Sintel training split).

As indicated by our earlier findings, this seems to be caused by the number of training iterations being too small. We can indeed verify that this is the case via a RAFT+DICL model trained with eight iterations per level, as shown in Figure 5.3. Observations are again similar to the ones made in Section 5.1.1. While we share the recurrent network weights across all levels, therefore training it with a combined number of $4 + 3 + 3 = 10$ iterations per sample, we reinitialize the hidden state on each level. This, in turn, means that training behavior regarding the recurrent network is more similar to three separate samples, trained with 4, 3, and 3 iterations each. A potential solution for this which avoids increasing the number of iterations could be upsampling of the hidden state between levels to (at least partially) retain it and establish one single recurrent sequence.

5.4.5 Normalization Layers

By default, we disable all batch normalization layers according to our training strategy (cf. Section 4.5). We now investigate whether enabling them or replacing them with group normalization layers can be beneficial. Note that we still use instance normalization for the feature encoder (as does RAFT [TD20]), meaning we only enable or replace normalization layers in the context encoder and the DICL matching network.

Method	Parameters	Chairs2	Things		Sintel	
	Norm	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-3)	none	1.2092	1.5938	3.0188	2.1514	**3.1364**
RAFT (CtF-3)	batch	**1.1945**	1.6311	2.9811	**1.9644**	3.1603
RAFT (CtF-3)	group	1.2219	**1.5841**	**2.9549**	2.1950	3.2603
RAFT+DICL (CtF-3)	none	1.2745	1.7436	3.0417	**2.0666**	3.2351
RAFT+DICL (CtF-3)	batch	**1.2395**	**1.7271**	**2.9763**	2.0992	**3.1172**
RAFT+DICL (CtF-3)	group	1.2493	1.7525	3.0453	2.2871	3.5325

Table 5.14: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method with batch and group normalization. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic) when using the respective norm type (*Norm*).

Results in Table 5.14 show that, for the coarse-to-fine models, normalization layers can improve results, even with our comparatively small batch size of six (for FlyingChairs2) and four (for Sintel and FlyingThings3D). In particular, batch normalization seems to slightly improve results overall for the RAFT+DICL approach, whereas group normalization, however, seems to perform worse across the board except for the FlyingChairs2 stage. For the RAFT coarse-to-fine models, results seem to be more mixed. Batch normalization can lead to gains for the FlyingChairs2 and Sintel stages, but group normalization performs better on the FlyingThings3D stage. These results stand in contrast to our RAFT and DICL baseline evaluations in Sections 5.1.1 and 5.1.2, which did not show clear benefits and instead (in some cases) even performance losses when using batch normalization. We expect that with larger batch sizes, results with batch normalization would likely improve yet again.

While this evaluation shows the general potential of batch normalization, additional tests may be beneficial. In particular, one may want to evaluate a different normalization type for the encoder or different normalization types for matching network and context, as we so far have only tested their types together. Further, in the original RAFT training strategy, batch normalization layers are frozen after the first stage, which we also did not test. Due to the small batch size, weight standardization [QWL+20] could also prove fruitful.

5.4.6 Weight Sharing

Lastly, we investigate different ways of sharing weights between neural network submodules. By default, we share weights for the recurrent refinement network as we expect the cost volume representation to be a general representation across all coarse-to-fine levels, whereas the feature representation may be adapted for each level. A similar argument can, however, also be made for the feature representation and the DICL module: If the feature embeddings are compatible across levels, the DICL module, consisting of matching cost network (MNet) and displacement-aware projection (DAP), can be thought of as a general operator to compute cost values (much like the dot product is generally applicable, given appropriate feature representations).

Table 5.15 shows that our initial intuition regarding weight sharing between the recurrent network instances was correct. Sharing weights between RNNs generally improves results for both coarse-to-fine RAFT and our RAFT+DICL approach over fully separate RNN instances. Similarly, however,

Method	Shared Weights		Chairs2	Things		Sintel	
	DICL	*RNN*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-3)	N/A	✓	**1.2092**	**1.5938**	**3.0188**	2.1514	**3.1364**
RAFT (CtF-3)	N/A	✗	1.2843	1.6235	3.0264	**2.0720**	3.2318
RAFT+DICL (CtF-3)	✗	✓	1.2745	1.7436	3.0417	**2.0666**	3.2351
RAFT+DICL (CtF-3)	✓	✓	**1.2344**	**1.6663**	**3.0089**	2.1456	**3.1710**
RAFT+DICL (CtF-3)	✗	✗	1.3467	1.8765	3.1106	2.1434	3.2478

Table 5.15: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using different weight sharing strategies. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic), sharing weights for the indicated network parts, specifically the DICL module (*DICL*), consisting of matching network and DAP, and RAFT's recurrent refinement network (*RNN*).

sharing weights between DICL modules also improves results, in some instances considerably (with the single exception of the Sintel/Sintel-clean evaluation). We achieve overall best results by sharing weights for both DICL modules and recurrent refinement networks. These results are somewhat similar to the ones of Hur and Roth [HR19], who have shown that clever sharing of weights combined with iterative estimation and refinement can transform existing methods in better performing (iterative) ones. In addition to our experiments, one could also try separate DAP layers per level.

5.4.7 Summary

In summary, our coarse-to-fine methods perform overall slightly worse than their respective baselines, with the three-level coarse-to-fine one showing the most promise. This also holds for smaller flow magnitudes, especially flow restricted to (roughly) the receptive field size, for which previously observed improvements over the baselines disappear. Most notably, we find that coarse-to-fine approaches (both RAFT and RAFT+DICL) can outperform the standard RAFT multi-level method, although a fair comparison thereof is difficult and would require further evaluations.

A downside of coarse-to-fine models over the alternative multi-level approach is, however, that the latter, when dealing with a consecutive image sequence, can be initialized using the flow estimate of the previous sample of this sequence. This is referred to as "warm-start" by Teed and Deng [TD20]. The nature of coarse-to-fine models prevents easy application of this on all but the coarsest levels, which likely makes it less useful overall.

Comparing our RAFT+DICL models against RAFT baselines with a (similar) lower number of feature channels does not exhibit any clear and overall deficiencies of our approach, yielding roughly comparable results for 32 feature channels. The comparably low number of feature channels may be a limitation of our approach. For more than 32 feature channels (as validated by our testing with 64 channels), the original DICL matching network may become a bottleneck and therefore may need to be scaled up in more than just its input layer. We will experiment with this in Section 6.3.

While coarse-to-fine approaches lessen the dependency on a large amount of video memory compared to the standard RAFT-based multi-level technique, we still run into notable limitations. In particular, we found that the number of recurrent iterations performed during training is too little for both our three-level coarse-to-fine RAFT+DICL approach and its respective RAFT baseline. Increasing the number of iterations to eight per level helps resolve this, however, comes at a significant cost of both memory required and computing time consumed. Alternatively, we may be able to retain the information encoded in the hidden state by upsampling it in between levels, bypassing said problem.

Additional improvements can be found with both normalization layers and weight sharing. In particular, we observed that, in contrast to our baseline evaluations in Section 5.1, batch normalization applied to the DICL matching cost network and the context encoder network can lead to considerable performance gains, even for the comparatively small batch size employed by us. Similarly, we found that sharing the weights of both DICL modules and the recurrent refinement network across levels generally improves performance, in some instances considerably so.

5.5 Summary

Evaluations performed throughout this chapter have shown mixed results. The single-level RAFT+DICL approach (Section 5.2) initially showed promise, but upon closer inspection, performance improvements over its RAFT baseline can mostly be attributed to flow outside the (direct) receptive field. When limiting our evaluations to the receptive field, earlier gains disappear. Similar results can be observed for our coarse-to-fine approaches (Section 5.4). In contrast to these, our multi-level approaches (Section 5.3) seem to fail outright, producing generally worse results than the single-level RAFT+DICL method. This hints at a fundamental problem within, at the very least, those specific models. As we will see in the next chapter, its likely cause, however, also affects our other approaches, leading to yet undiscussed training difficulties and smaller performance deficiencies.

In addition, the multi-level RAFT+DICL methods prove difficult to train due to their heightened memory requirements, making investigation of issues slow and cumbersome. Due to both this and its results, the overall most promising RAFT+DICL model, which we will focus on in the remainder of this thesis, is the three-level coarse-to-fine one. Even though considerable improvements have been achieved by sharing weights across levels, in particular those of the DICL module, its performance is still subpar when compared to its RAFT baselines. Limitations discovered so far may include the number of recurrent refinement iterations as well as the number of feature channels. We believe that the latter may express itself differently than with our RAFT baselines (Section 5.1.1) as, for those, the bottleneck may lie within the dot product (specifically, the number of its input channels) and not the capacity of the feature representation itself.

6 Difficulties and Improvements

With the evaluations performed in the previous chapter, we have built the very basis from which we now aim to improve our method. As hinted at in Section 5.5, we encountered some stability issues during training, especially of our more complex coarse-to-fine RAFT+DICL methods. We will discuss these problems in Section 6.1. Within this section, we also aim to understand why the multi-level RAFT+DICL approaches may have failed as catastrophically as they did. Section 6.2 expands on this by looking at the produced cost volume specifically and discusses means with which we can improve it. Thereafter, we will briefly explore variations of the DICL matching cost network, representing the core ingredient of the learnable cost function module, in Section 6.3.

6.1 Stability Problems

Even though we have presented baseline models (Chapter 3), our evaluated models (Chapter 4), as well as our training and evaluation strategies (Sections 4.5 and 4.6), there is one piece missing on our way to obtain the results shown in the previous chapter that needs discussion: The stability problems we encountered when training our approaches, and how we can avoid them. These issues can manifest themselves twofold: First, on a global basis, causing the model to converge towards producing a constant zero-flow output, from which it may not recover, and second, on a per-level basis, causing somewhat more subtle performance degradation.

We will discuss these issues in Section 6.1.1 and Section 6.1.2, respectively, and present means with which they can be avoided. Section 6.1.3 will present a theory as to why these issues are predominant in our RAFT+DICL approaches, especially the coarse-to-fine ones, whereas the initial difficulties do not seem to be shared by either RAFT and DICL. Lastly, we will investigate both RAFT and DICL and find that, to some extent, they also share the per-level issues.

6.1.1 Initial Convergence

The most prevalent issue of our models is their stability during the initial training steps. In such a case, instabilities express themselves as a failure of the model to converge to the correct result. Instead, it converges to produce near zero-magnitude flow across the whole domain. Severity of this issue varies with the model. In particular, we have found the four-level RAFT+DICL coarse-to-fine model to be essentially untrainable without forward-backward batching (cf. Section 4.5.1) and lowering the learning rate substantially, whereas the single-level RAFT+DICL model rarely exhibits such problems. We therefore believe that the increasing complexity of the model plays a role in its cause.

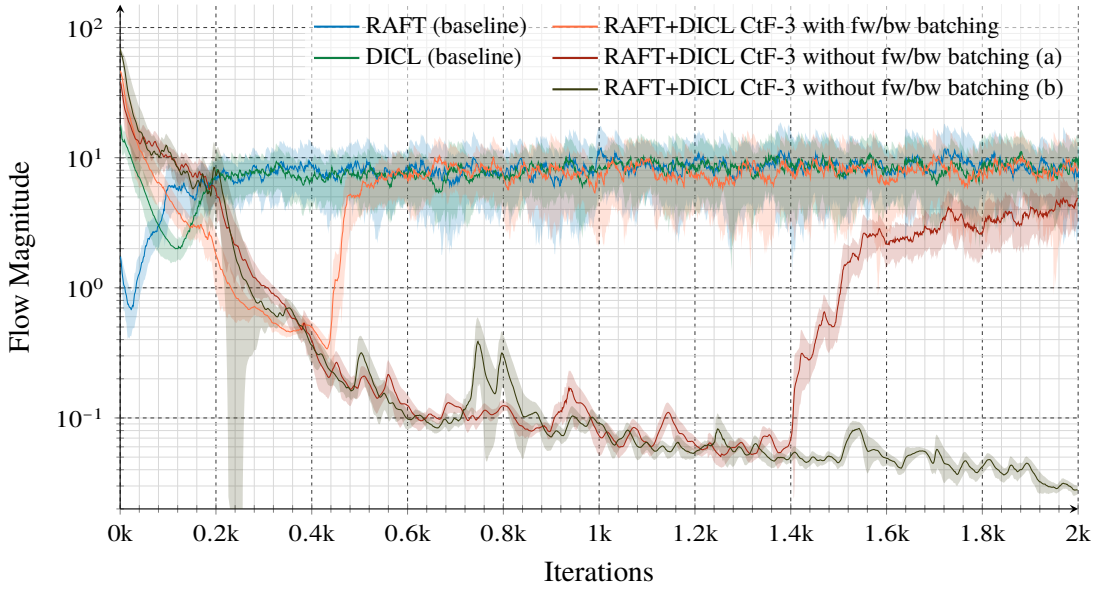

Figure 6.1: Flow magnitude during the initial training steps on the FlyingChairs2 dataset. Our RAFT+DICL models (here exemplary shown for the three-level coarse-to-fine variant) have a tendency to converge towards producing zero flow. Forward-backward batching can help the model to converge towards the intended output. Values have been smoothed using an exponentially weighted moving average, their standard deviation is indicated by the surrounding shaded area.

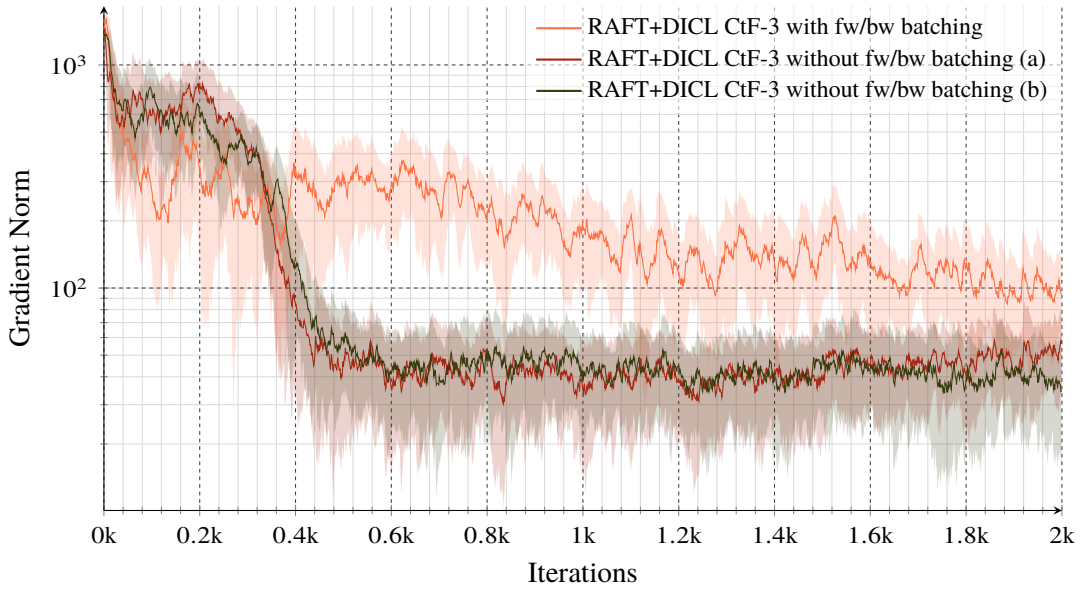

Figure 6.2: Gradient norm during the initial training steps on the FlyingChairs2 dataset. Models experiencing convergence problems (i.e., without forward-backward batching) seem to be caught in a saddle point (cf. Figure 6.1). Values have been smoothed using an exponentially weighted moving average, their standard deviation is indicated by the surrounding shaded area.

Symptoms

Figure 6.1 illustrates this problem by showing the flow magnitude for the initial training steps of the three-level coarse-to-fine model, with our RAFT and DICL baselines for comparison. All RAFT+DICL models shown here have been trained with a base learning rate of 5×10^{-5} during the preparatory epoch, whereas the baseline models use a learning rate of 4×10^{-4}. Other parameters follow our training strategy, described in Section 4.5. We can observe that, initially, all RAFT+DICL models tend to converge towards producing zero-magnitude flow. A similar but considerably less severe trend can also be seen in both baselines. Notably, however, variants trained without forward-backward batching converge to the correct target after significantly more training iterations, if at all. As shown in Figure 6.1, even with an appropriately reduced learning rate, models can fail to produce the desired results, leaving proper convergence up to chance. Note that variants (a) and (b) have been trained with the exact same parameters but a different random seed.

Treatment

In our testing, forward-backward batching, in combination with an appropriate learning rate, was able to ensure convergence for all tested models. We can attempt to understand where these instabilities come from (and how forward-backward batching may help) by looking at the gradient norm, provided in Figure 6.2. While all approaches start off with a similar gradient norm, variants without forward-backward batching have a considerably higher gradient norm during the first 300 steps. Thereafter, their gradient norm drops sharply and remains constantly low. We believe that this may show that the model is running into a saddle point, from which escape is difficult. A similar notion has been made by Jaegle et al. [JBA+22], who found forward-backward batching necessary for convergence of their Perceiver-IO approach.

Forward-backward batching, on the other hand, results in somewhat smaller gradients during the very first steps. This, in turn, seems to be crucial in circumnavigating the saddle point. Our theory as to why forward-backward batching has this impact on the gradient norm is that it can cancel out directional biases: By providing both forward and backward flow, any (predominant) direction in a forward-flow sample gets canceled out in its corresponding backward-flow sample. Therefore, a batch containing forward and backward flows for all samples should not contain any (significantly) predominant direction. We believe that a similar observation could also be made with a considerably larger batch size due to the law of large numbers, assuming the dataset is balanced and does not exhibit directional biases itself. See Appendix A.1 for additional discussion of forward-backward batching.

Lastly, we found that models behave stable after initial convergence to the target output has been reached, even when altering the training strategy by dropping forward-backward batching and increasing the learning rate. It is therefore not necessary to train models from start to end with forward-backward batching (or with a low learning rate), which may be preventative as some datasets (notably Sintel and KITTI) do not provide backward flow, meaning training on these would require estimation thereof.

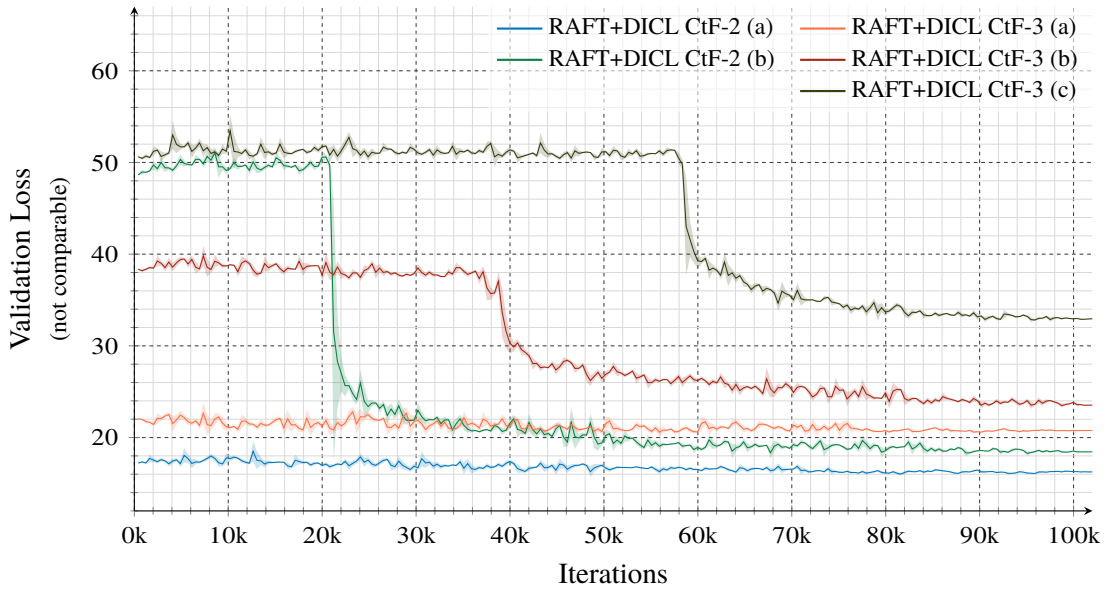

Figure 6.3: Validation loss for two- and three-level coarse-to-fine RAFT+DICL approaches on the Sintel stage. The exact time when more levels start being actively involved can be seen by sudden drops in validation loss. Approaches (a) have been trained with an initial learning rate of 1×10^{-4} and 5×10^{-5} for the two- and three-level models, respectively, whereas (b) and (c) have been trained with an initial learning rate of 4×10^{-4}. Values have been smoothed using an exponentially weighted moving average, their standard deviation is indicated by the surrounding shaded area.

6.1.2 Per-Level Convergence

A significantly more subtle problem of our coarse-to-fine approaches is estimation of zero-flow on a per-level basis. This does not directly lead to a failure to train the model, but instead to a somewhat less noticeable performance degradation. In its essence, this issue resembles the initial convergence problems, described in the previous subsection, however, it is instead constrained to one or multiple coarse-to-fine levels: Flow is estimated correctly at the finest level, creating the appearance that the model works correctly, but coarser level converge to produce zero flow.

We have found our RAFT+DICL coarse-to-fine models to actively start involving coarser levels during later training stages. This can be seen as sudden anomalous drops in the validation loss, presented in Figure 6.3. By looking at the per-level model estimates, shown in Figures 6.4 and 6.5, we can directly attribute the drops to the respective coarse-to-fine levels. For instance, the two-level (b) approach can be seen to produce zero-flow on the coarsest level before training on Sintel, whereas it produces flow on that level after this stage. Similarly, the three-level (b) approach starts actively using the coarsest level during the Sintel stage, whereas the three-level (c) approach only starts using the second-coarsest level. This aspect is also apparent in the validation loss (cf. Figure 6.3).

The notable difference between the successful and failed variants shown in Figures 6.3 to 6.5 is their learning rate. As before, we chose 1×10^{-4} for the successful two-level approach, 5×10^{-5} for the successful three-level one, and 4×10^{-4} for the failed approaches. We found that selecting an appropriately small initial learning rate is crucial in avoiding these problems. In particular, the

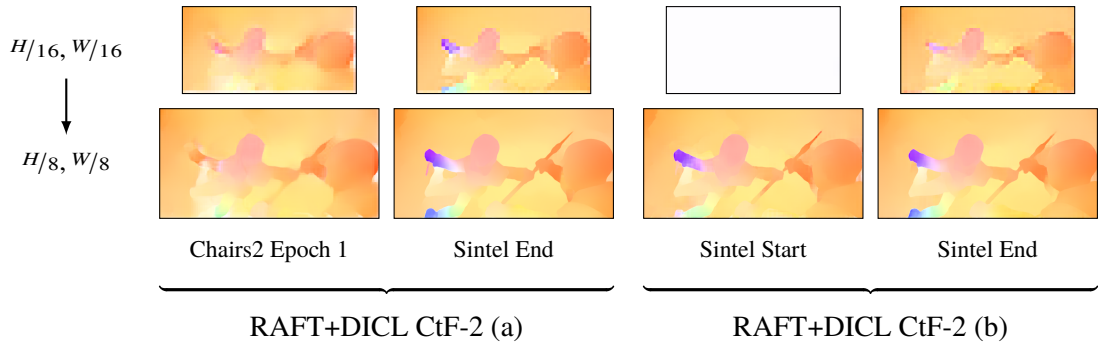

Figure 6.4: Per-level flow estimate for the two-level coarse-to-fine RAFT+DICL approach. The model trained with initial learning rate 1×10^{-4} (a) shows the desired output on the coarsest level across all training stages, whereas the model trained with learning rate 4×10^{-4} (b) produces zero flow on the coarsest level up to the start of the Sintel stage.

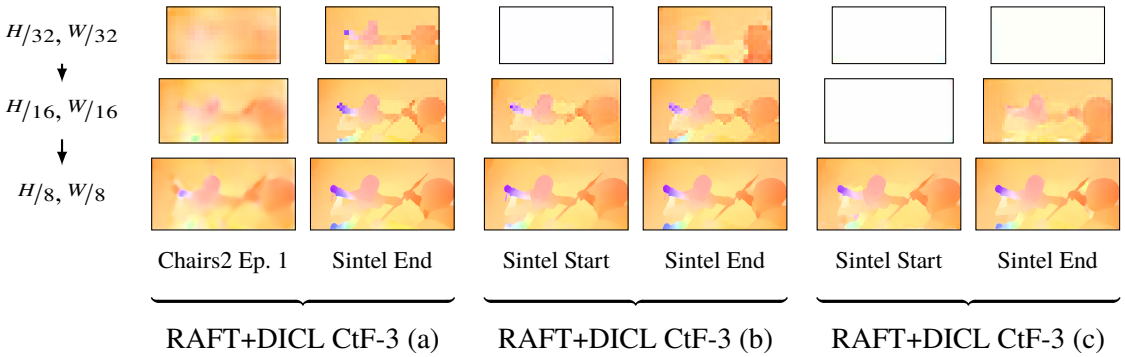

Figure 6.5: Per-level flow estimate for the three-level coarse-to-fine RAFT+DICL approach. The model trained with initial learning rate 5×10^{-5} (a) shows the desired output on all level across all training stages, whereas the models trained with learning rate 4×10^{-4} (b) produces zero flow on coarser levels up to or even during the Sintel stage.

learning rate again needs to be smaller with an increasing number of levels. Similar to the issue discussed in the previous section, once a level produces non-zero estimates, the learning rate can be increased without the level going back to producing zero-flow.

6.1.3 A Theory on the Convergence Problems of RAFT+DICL Approaches

Due to their inherent similarities, we believe that the problems discussed in Sections 6.1.1 and 6.1.2 stem from the same underlying cause. In this subsection, we aim to provide a theory as to what this cause is and why such issues seem to be prevalent in our RAFT+DICL approaches, but are not apparent in both DICL and RAFT.

We theorize that the fundamental cause can be summed up as a lack of guidance for the cost learning approach: Both of our baseline competitors, RAFT and DICL, directly encode a notion of cost or correlation in the network. More specifically, they provide an inductive bias for how a good or a bad match is encoded. In RAFT, this is done via, first, symmetricity of the feature encoder

and the dot product and, second, the positive definiteness of the dot product itself. This results, in essence, in a positive semi-definite correlation function, therefore directly encoding that high values represent good matches whereas low values represent bad matches. DICL, on the other hand, cannot rely on symmetricity and positive (semi-)definiteness. Instead, however, it directly supervises the cost/correlation function via the soft-argmin regression (cf. Equation (3.7)). Note that this regression is not a learned operator, therefore using the resulting flow in the loss term represents a direct supervision of the cost values.

Our RAFT+DICL approaches have neither a direct supervision of the cost values nor a sufficiently guiding inductive bias in any other way, as the DICL matching network is neither symmetric nor positive (semi-)definite. The network is instead completely free to choose an appropriate representation itself, which we believe may cause two related problems:

Cost Representation and Association

First, the network may experience difficulties in associating cost volume entries with their respective displacement vectors. The RAFT recurrent refinement network, decoding the cost volume, takes in the flattened vector of all costs. It therefore has to find a mapping from cost/correlation entry to resulting flow/displacement vector itself. In RAFT, this is aided by the positive semi-definite and symmetric correlation function, which inherently provides a means of similarity (as already discussed above).[1] We believe that this aids the network in finding these associations.

Without such constraints, the network has to both learn a representation and then also associate cost volume entries with displacements. These may be somewhat conflicting goals, which we can see by giving an example: Let us assume that, for one feature pair, the randomly initialized matching network and feature encoder stack outputs, by chance as yet untrained, high values for matching features. For another matching pair, however, the network may output low values. If both feature pairs now share the same displacement vector, the decoder network is essentially asked to associate both cost values with the same resulting flow.

A perfectly randomly initialized cost function, outputting a truly random and unbiased cost value for any (distinct) feature pair, will therefore always provide conflicting cost values and, due to this, prevent the refinement and decoding network from being able to form associations. This indicates that, before associations for decoding can be formed, a common cost representation needs to be established and that, in essence, we rely on the initialization to provide a bias towards one representation. We believe that this is the root cause for the network converging to zero flow: The decoder part cannot, at least initially, rely on the cost values. It is nevertheless being trained to reduce the flow error. Without any valid input, it therefore reduces the error by converging towards the average flow of its training targets, which we expect to be zero on a balanced dataset.

[1] Note that a small hint of training these associations may still be visible in our RAFT baseline in Figure 6.1.

Representation Conflicts across Levels

Second, as our approaches (by default) do not share DICL module weights across levels, the above problem is repeated on each level. From our experiments, it seems that coarser levels may be more difficult to train correctly. For the coarse-to-fine models, we now also need a common cost representation over all levels, as we (again by default) share weights for the recurrent refinement network decoding the cost volume. However, there is again no guiding factor towards compatible representations. This means that, especially initially, representations may conflict with each other until a dominant representation is established and all levels have been converted to it. We believe that this, likely together with coarser resolutions, is the root cause for coarse-to-fine models with more levels being more difficult to train successfully.

In theory, this second problem should be less severe with approaches sharing DICL module weights or with approaches not sharing the recurrent network weights. Our experiences from training show that (for the coarse-to-fine variants) these models do seem somewhat easier to train. More specific evaluations would, however, be required to ensure that this is indeed the case.

The described difficulties could also explain the failure of the multi-level RAFT+DICL models. In their instance, the model may decide to simply ignore coarser cost volume levels, as costs are collected simultaneously and over all levels. In contrast to the coarse-to-fine methods, there is no active supervision enforcing estimation of (largely) correct flow on each level independently, making it much more likely that certain levels may never learn to produce valid cost values or other motion information. The data provided on these levels may then rather confuse the model than help it, explaining the generally worse performance of the multi-level methods over the single-level one.

6.1.4 Per-Level Convergence Problems in RAFT and DICL

Naturally, we may ask ourselves if the somewhat less obvious per-level estimation problems also exist (at least in some form) in other approaches, in particular in RAFT and DICL. Especially, as these methods do not seem to exhibit the initial training difficulties leading to outright failure. We will first investigate DICL, which is again straight-forward, and thereafter discuss RAFT, where we attempt to gauge the importance of certain cost volume levels by zeroing them during evaluation.

DICL

Figure 6.6 shows per-level flow estimates for DICL methods using three different training strategies. In particular, we compare the original model checkpoints provided by Wang et al. [WZD+20] with our re-trained baseline (adhering to Section 4.5 with an initial learning rate of 5×10^{-5} and batch normalization disabled), as well as an approach using a batch size of 32 by performing gradient accumulation over four steps with a batch size eight per step.

From the depicted results, we can see that the per-level issues are not exclusive to our RAFT+DICL approaches. We observe similar problems on all of our DICL baseline to various degrees. Specifically, our re-trained baseline shows issues on the coarsest level. Similarly, for the original checkpoints, notable artifacts are visible in the coarsest level output of the FlyingChairs stage, however, they disappear after training on the FlyingThings3D dataset. As one of the most significant differences

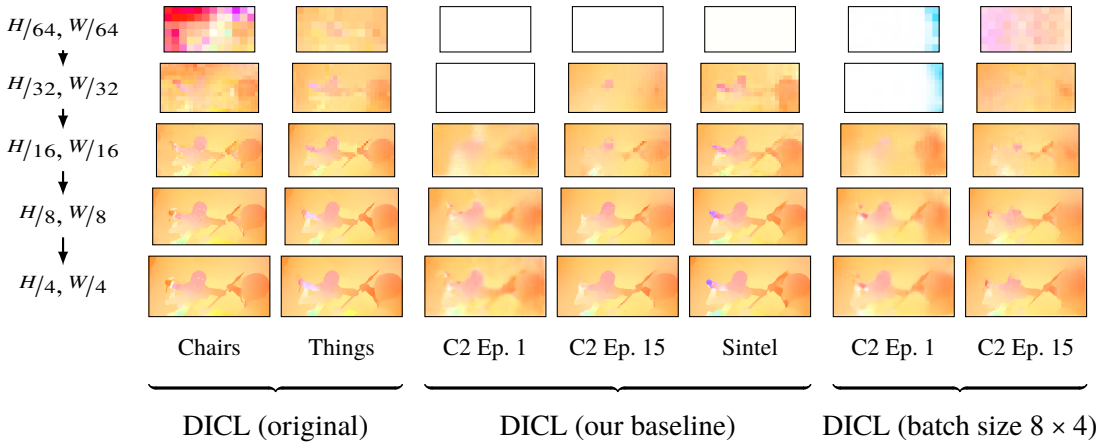

Figure 6.6: Per-level flow estimate for the DICL method. The original DICL checkpoint shows artifacts on the coarsest level after the first training stage, whereas our re-trained version does not produce any meaningful flow on the coarsest level across all stages. Re-training with a batch size of 32 through batch accumulation (size 8 with 4 accumulation steps) shows a reasonable estimate after 15 epochs of FlyingChairs2.

between our training strategy and the one suggested by Wang et al. [WZD+20] is the considerably lower batch size of our strategy, we re-train our approach with an effective batch size of 32. Results show that the per-level estimation issues are still present after the first epoch of FlyingChairs2, however, they have (largely) disappeared after epoch 15. This indicates that a larger batch size may be able to resolve such issues as well.

RAFT

In contrast to DICL and our coarse-to-fine approaches, investigating the impact of different levels of RAFT is not straightforward. In particular so, as we are now dealing with cost volume levels (collected and fed to a singular decoder) and not separate coarse-to-fine levels (where flow estimation is performed separately, apart from sharing of weights). To gauge the contribution of specific levels, we therefore attempt to mask cost volume levels by zeroing them, only providing correct cost values on the remaining levels. This approach, however, may not be fully reliable and results need to be taken with care: The recurrent network may expect a certain structure in the cost vector provided to it, such as introduced by downsampling costs, which may be violated by our masking.

Results for this process, applied to the two-level RAFT and our two-level RAFT+DICL models, are shown in Figure 6.7 (four-level models show overall similar behavior, see Appendix A.2). We can observe that, when comparing the complete models to the models using only the finest cost level (level 3) for estimation, results are near identical. Only minor details, for example in the left hand (purple), differ (this is more apparent in the four-level model, cf. Figure A.2).

When relying only on the coarsest level, we can see significant differences between models. Specifically, our RAFT baseline still provides a somewhat accurate estimate, however, our RAFT+DICL models fail to do so. This failure is not limited to the model with CNN-based encoder and separate DICL modules, but also, somewhat less severe, to the approach using average-pooling and a DICL

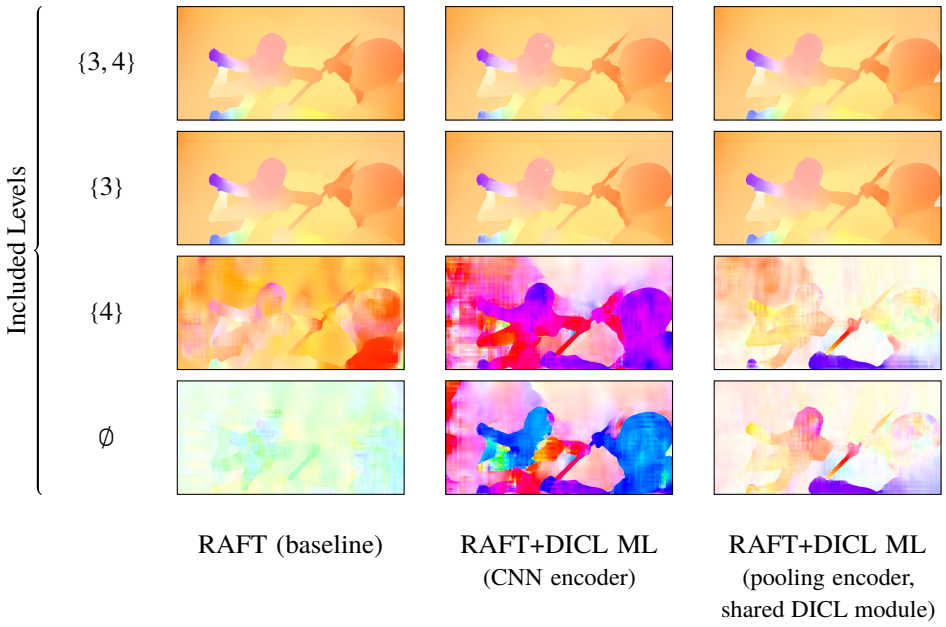

Figure 6.7: Flow estimates of the two-level RAFT and RAFT+DICL approaches, including only specified cost volume levels. Excluded cost levels are set to zero. Results are taken after four refinement iterations. The RAFT baseline shows somewhat truthful flow when only the coarsest level (level 4) is actively used, whereas results for the RAFT+DICL methods are largely incorrect.

module shared between levels. These results are somewhat similar to results when providing a completely zeroed out cost volume, indicating that at least parts of the output may stem from the refinement network rather than from the cost information.

We believe that this could also indicate that, for our methods, the coarsest level does not actively determine flow, but rather serves as an addition to the finer level (if considered at all), helping to provide regularization, as previously theorized in Section 5.3.2. The approach with CNN-based encoder provides significantly different results on the coarser levels than the pooling-based approach, which could mean that it does not provide matching cost values at all, but rather some explicit values trained to enhance cost values from the first level. This, in turn, could explain its relative performance improvement over the pooling-based approach. Looking back at the results in Section 5.3.2 and specifically the comparison to the single-level approach, however, these additional levels seem to confuse the model more than help.

Lastly, we want to take note of a somewhat interesting aspect of the models presented in Figure 6.7 in regard to the sample used for our visualization: The flow for the left hand (purple) is correctly estimated on both our two-level models and our single-level models (not shown here), however, has a flow magnitude of about 40 px to 64 px. This is well outside the 32 px cost volume range for the single-level approach and the finest level of the multi-level approaches. For the two-level RAFT baseline, we found that, in the first iteration, flow in this part is estimated accurately when relying solely on the coarsest level, indicating that it is indeed used for larger flows (note that our figures show results after iteration four). Our RAFT+DICL approaches, in turn, do not show this. As the single-level approach also estimates this part correctly, this does not necessarily mean that, for our

RAFT+DICL models, coarser cost volume levels contain any information helping to increase the range, but may rather indicate that such information is obtained elsewhere, for example through the feature representation encoding a somewhat larger context area.

6.2 Improving the Cost Volume

In the last section, we theorized that the stability issues encountered within our RAFT+DICL approaches may stem from these models having difficulties finding a common cost (or, alternatively, correlation) representation. But how do their cost values look in reality? And, more importantly, can we improve them while maybe also resolving the aforementioned stability problems in a better way? We aim to answer these questions in this section.

To this end, we begin by investigating the cost volumes produced by various approaches in Section 6.2.1, starting with our RAFT and DICL baselines for comparison, before finishing with two of our RAFT+DICL methods. Using the insights obtained therewith, Section 6.2.2 introduces a direct supervision approach inspired by DICL [WZD+20], intended to shape and, through this, improve the learned cost function. Section 6.2.3 takes further inspiration from Wang et al. [WZD+20], proposing a modification of the loss function. Finally, Section 6.2.4 provides results for the techniques introduced in the two previous subsections, after which Section 6.2.5 summarizes our findings.

6.2.1 Comparing Cost Volumes

Before we can attempt to improve the cost volume, we find it beneficial to first gain a better understanding of what we are currently working with. To this end, we visualize slices thereof following the technique described in Section 2.4, focusing on level $m = 3$ (i.e., the finest for our RAFT+DICL approaches) and the last refinement iteration (where applicable). If not specified explicitly, we use the models fully trained using our default training strategy (Section 4.5) and the standard Sintel stage.

To gauge the quality of the cost volumes produced by our RAFT+DICL methods, we compare them with the cost volumes created by various baselines. In particular, we chose cost volumes from the original RAFT method using the checkpoint provided by Teed and Deng [TD20], our single-level RAFT baseline, our three-level coarse-to-fine RAFT baseline, as well as the original DICL method using the checkpoint provided by Wang et al. [WZD+20]. The sample used for our evaluation is shown in Figure 6.8.

RAFT

Results for the RAFT baselines are shown in Figure 6.9. In all three visualizations, we can see a halo surrounding the two moving characters, with the clarity of the halo varying with the method. This seems to stem from the occlusions caused by the motion of said characters and gives an indication that the cost volume can not only convey information about motion, but also some degrees of uncertainty of it. We can further see that similar object structure and texture leads to similar

Figure 6.8: The sample used for our cost volume evaluation (Sintel training set, scene "ambush 6", frame 4). Shown are first and second input frame ($\mathcal{I}_1$ and $\mathcal{I}_2$, respectively) and ground-truth flow (**u**).

correlation scores, creating an equal means of uncertainty. In our example, this is especially visible for the shaft of the spear shown in the excerpt. Here, correlation scores are equally large across the length of the spear, resulting in line-like structures in the cost volume.

While a distinct area (and often also a center) of focus is visible in most of the unoccluded cases, costs seem to be somewhat noisy outside it. This is likely owed to the use of the dot product as the correlation function, and is an aspect that we would expect to be improved by the DICL approach. Note that these areas are significantly more spread out in the single-level RAFT baseline. This could be due to the larger amount of flow outside the receptive field compared to the other two methods, creating a larger amount of uncertainty inside the model.

DICL

Cost volume slices for DICL are shown in Figure 6.10. We can again see the expected areas of uncertainty, which are especially noticeable after the displacement-aware projection (DAP) layer. This layer, in particular, seems to reshape the fairly focused match scores produced by the matching cost network (MNet) into a broader Gaussian-like shape. Considering the nature of the soft-argmin flow regression employed by DICL (cf. Equations (3.7) to (3.9)) this is not all that surprising. The scale of spread should be controllable through the temperature parameter τ of the employed softmax function. Notably, however, the DAP layer also seems to create some artifacts at the fringes of the displacements for individual source pixels.

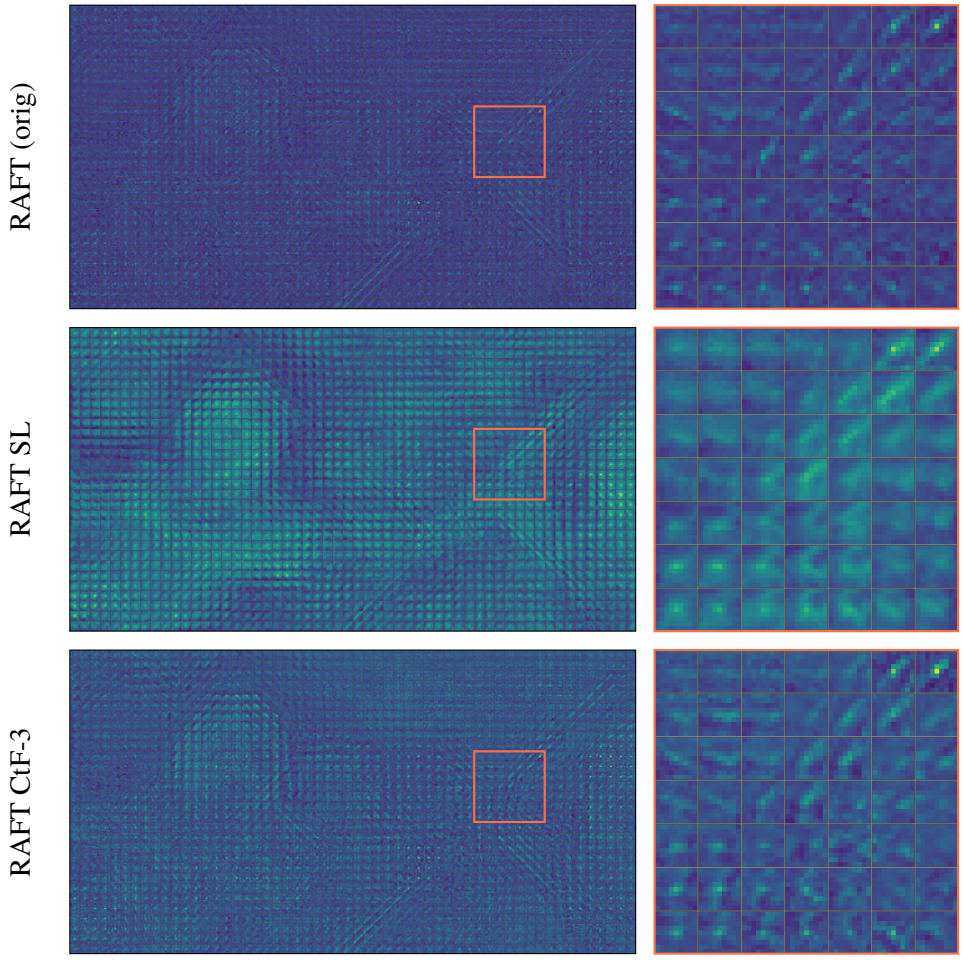

Figure 6.9: Cost volume slices for different RAFT variations, with excerpts to the right. Top: Original checkpoint by Teed and Deng [TD20]. Middle: Our single-level RAFT baseline. Bottom: Our three-level coarse-to-fine (CtF-3) RAFT baseline. All slices have been taken at the finest level during the last refinement iteration.

In contrast to the cost volumes produced by our RAFT baselines, the correlation scores produced by the matching network seem to have, in general, a clearer center point of focus. Nevertheless, the focus area around this point still follows the shape of ambiguous textures and objects. Less noise is visible outside said area, confirming our intuition above.

RAFT+DICL

Cost volume slices for the single-level and coarse-to-fine RAFT+DICL approaches are shown in Figure 6.11. Visualizations for the multi-level methods are provided in Appendix B. As is directly apparent, the cost volumes of our approaches stand in stark contrast to the cost volumes used by the baseline models. We can observe two major differences:

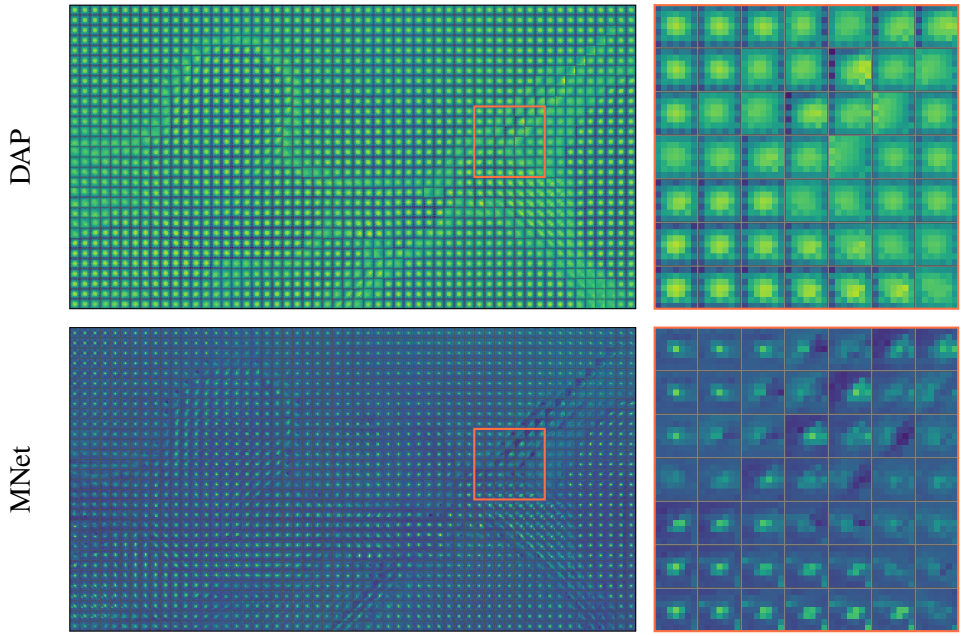

Figure 6.10: Cost volume slices for the original DICL checkpoint by Wang et al. [WZD+20] (for level $m = 3$ with resolution $W/8 \times H/8$). Costs are visualized after the matching network (MNet, bottom) and after the displacement-aware projection (DAP, top), with marked excerpts to the right.

First, the cost representation is different. For the single-level model, in particular, both low (purple) and high (yellow) scores seem to indicate feature matches, whereas non- or badly matching feature pairs are encoded by medium (green) values. We believe that this may be one reason for suboptimal results, as the matching network now needs to handle two cases for matching feature pairs and the maximum dissimilarity score is bounded between those. Notably, this issue seems to be less pronounced on the coarse-to-fine approach. However, there still seem to be some (source) pixels (bottom right in the excerpt) where higher scores seem to indicate a match, whereas for the vast majority of pixels lower scores indicate matches.

Second, we can observe a checkerboard-like pattern. This is most notable after the matching network, whereas the DAP layer seems to balance values out slightly. Especially in the single-level model, however, these checkerboard-like artifacts also contain source pixels where cost information is significantly less discernible or in some cases even seems to be missing outright. Further, while the DAP can balance some of these artifacts out, it also seems to introduce noise, which, in turn, seems more pronounced in the coarse-to-fine model. As we will confirm in the next subsection, the checkerboard-like pattern stems from the structure of the DICL matching network, which internally downsamples the input feature pairs to half their original resolution and then upsamples them again using transposed convolutions. This upsampling step explains the pattern of these artifacts.

The reason why both issues are seemingly less severe in the coarse-to-fine network could be that this network is forced to choose a common cost representation across all levels. To do so, the network must actively resolve potentially conflicting representations, which, we believe, could lessen these artifacts.

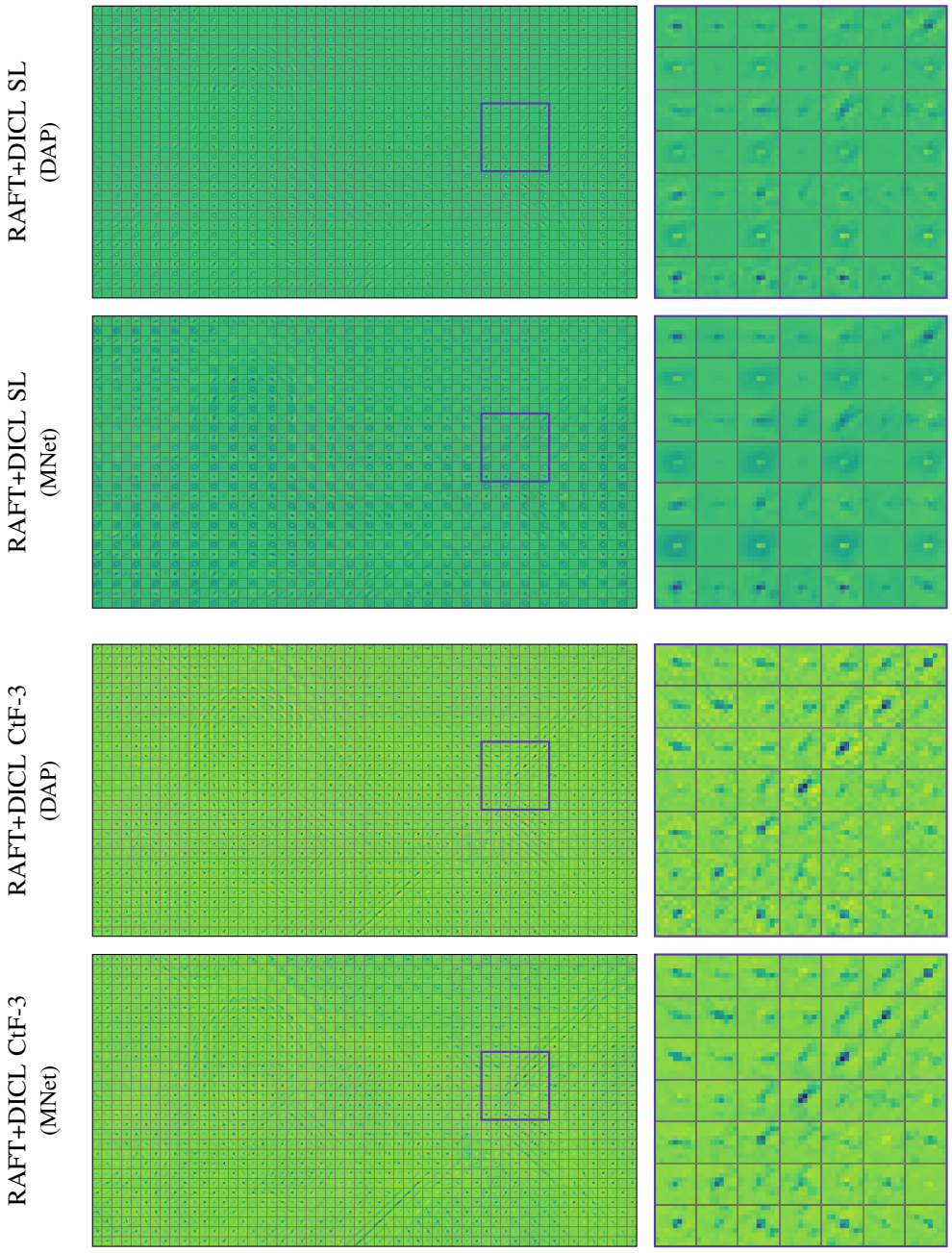

Figure 6.11: Cost volume slices for our single-level (top two) and coarse-to-fine (bottom two) RAFT+DICL approaches. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP), with marked excerpts to the right.

As can be seen in Appendix B, these problems and artifacts also affect our multi-level RAFT+DICL approaches. Using these visualizations, we can additionally verify our earlier findings of Section 6.1.4, in particular that the coarser levels of these approaches do not provide any discernible qualitative information about motion.

6.2.2 Guided Cost Learning by Direct Regression

Overall, the visual inspections performed in the previous subsection lend further credence to our theory on the stability issues presented in Section 6.1.3. They additionally indicate that the results of our current RAFT+DICL approaches can likely be improved if we manage to find a way to form a better cost representation and, simultaneously, prevent the discovered artifacts. We therefore attempt to guide the cost learning process by applying a more direct supervision to the cost values.

Cost Supervision via Soft-Argmin Regression

To supervise the learned cost function, we propose to employ the soft-argmin regression approach used by DICL, previously discussed in Section 3.2.1. The soft-argmin regression is directly applied to the cost volume to produce an intermediate flow output, which we then use in the multi-sequence loss (Section 4.4) as a secondary output sequence for training. In contrast to DICL, we do not use the intermediately estimated flow during the (recurrent) refinement stage. We believe that doing so could be a topic for further research.

The downside of this approach is the addition of several new hyperparameters. In particular, we now have an additional temperature value for the regression itself, as well as new hyperparameters for the loss term. By default, we choose a weight of $\alpha_1 = 1$ for the main sequence and $\alpha_2 = \lambda = 0.85$ for the new secondary sequence, as well as a standard temperature value of $\tau = 1$. We did not specifically tune these parameters.

Figure 6.12 shows a slice produced by the three-level coarse-to-fine RAFT+DICL approach trained with this direct cost regression approach. In comparison to our previous approaches, cost values look significantly better. Unfortunately, we can still observe a checkerboard-like pattern in the cost values produced by the matching network. This is again balanced out by the DAP layer, which, however, in turn seems to introduce some noise-like artifacts. We therefore explore how these artifacts can be avoided next.

Preventing Artifacts

As already discussed above, the checkerboard-like pattern matches the upsampling step performed in the DICL matching network. However, as we have seen in Section 6.2.1 and Figure 6.10, the original DICL approach does not express such behavior. Hence, we investigate three additional variations of our approach to better understand what exactly causes these artifacts:

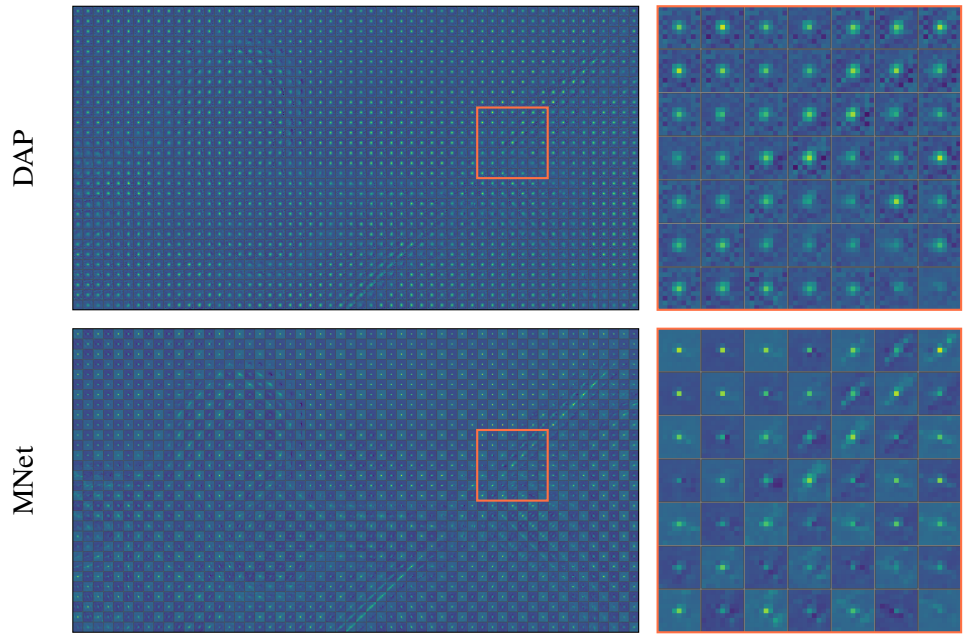

Figure 6.12: Cost volume slices for the three-level coarse-to-fine RAFT+DICL method with direct cost regression. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP), with marked excerpts to the right.

Gradient Stopping First, we stop the gradient from propagating backwards from the recurrent refinement network to the cost learning module. This essentially divides our model into two separate and individually trainable parts, which, however, we still trained in combination. The first part produces cost values and is supervised exclusively by the soft-argmin regression, whereas the second part takes these cost volumes as fixed input and decodes them into flow. The second part is supervised using the standard RAFT loss. Crucially, it therefore cannot influence the first part during training.

Cost volume slices for this experiment are shown in Figure 6.13. We can see that this approach does indeed not show the checkerboard-like artifacts and, in general, produces a cost volume much more similar to the DICL one than to those of our previous approaches. It stands to reason, however, that this may not be the best solution for avoiding these artifacts, as the refinement network may provide valuable guidance to improve the learned cost function.

Disabling the Displacement Aware Projection Layer Second, we can observe in all prior approaches, and especially Figure 6.12, that the displacement-aware projection (DAP) layer generally tries to balance out and remove the checkerboard pattern. We thus surmise that there is at least some supervision signal, likely strengthened by direct cost regression, which pulls the network towards a cost representation free of such checkerboard-like artifacts. We therefore attempt to exploit this signal by removing the DAP layer, forcing it to act directly upon the matching network. Note that we still employ direct cost regression.

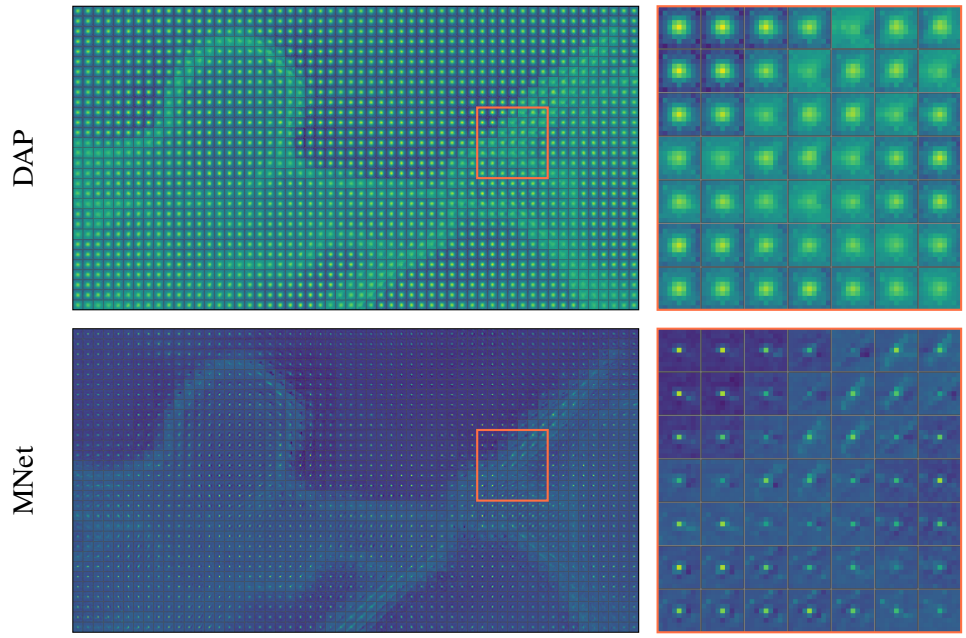

Figure 6.13: Cost volume slices for the three-level coarse-to-fine RAFT+DICL method with direct cost regression and gradient stopping. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP), with marked excerpts to the right.

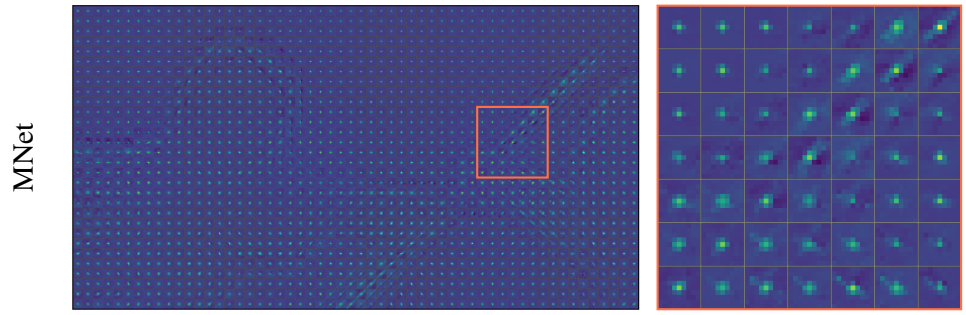

Figure 6.14: Cost volume slices for the three-level coarse-to-fine RAFT+DICL method with direct cost regression and without DAP layer. Costs are visualized after the matching network (MNet) pass.

The resulting cost volume is presented in Figure 6.14. This shows that removing the DAP layer gives the cost regression indeed a better control over the matching network, which prevents said artifacts (almost completely). We believe that a DAP layer may not be necessary for the RAFT refinement network, as this already employs a similar 1×1 convolutional layer as input layer. Notably, the original DICL training strategy proposed by Wang et al. [WZD+20] initially disables the DAP layer and only enables it after the model has received a certain amount of training. A similar strategy may also be enough to reduce artifacts in our case.

While we could outright remove it, the DAP layer may still be useful for the soft-argmin flow regression. In particular, we think that it could act as a negotiation layer between soft-argmin regression and the refinement network. Therefore, it could make sense to move the DAP layer into the cost regression module and let the refinement network directly access the costs produced by the matching network.

A Matching Cost Network without Upsampling Lastly, we can confirm our statements above and explore a matching network that does not contain transposed convolutions, and does neither down- nor upsample feature pairs. To this end, we use a network exclusively consisting of 1×1 convolutional layers. The overall network structure follows the DICL matching network, with down- and upsampling layers removed.

Figure 6.15 shows that this network produces (visually) good cost values, even when not paired with direct cost regression. Costs look similar to the ones produced by our RAFT baselines (cf. Figure 6.9) and show some noise outside the area of focus, likely due to the network structure not taking neighboring feature pairs into account (similar to the dot product). When combined with the soft-argmin cost regression, shown in Figure 6.16, cost values are notably more focused and noise is less visible. In both cases, the DAP layer seems to focus cost values more directly on a single displacement, but also adds noise. This becomes especially apparent in the center of the excerpt shown in Figure 6.16.

6.2.3 Loss Restriction

To train their DICL approach, Wang et al. [WZD+20] propose to restrict the loss term to the receptive field (cf. Section 3.2.2). This is done by limiting loss computation to pixels with ground-truth flow below the corresponding per-level threshold. We believe that this may also be beneficial for our soft-argmin cost regression, as this soft-argmin regression is the same mechanism with which the original DICL method by Wang et al. [WZD+20] supervises its cost volume. However, as the recurrent refinement network adjusts the center of the receptive field in each iteration, we cannot use the simple per-level threshold on the ground-truth flow employed by DICL. Instead, we need to determine the pixels for which to compute the loss dynamically in each iteration.

To this end, we (re)define the set of pixels $\Omega_{\text{restr}}^{(m,t)}$ on which we compute the training loss as

$$\Omega_{\text{restr}}^{(m,t)} \coloneqq \left\{ (i,j)^\top \in \Omega : \left| u_{\text{gt},i,j} - u_{\text{est},i,j}^{(m,t-1)} \right| < r_u^{(m)} \wedge \left| v_{\text{gt},i,j} - v_{\text{est},i,j}^{(m,t-1)} \right| < r_v^{(m)} \right\}, \tag{6.1}$$

where t denotes the current iteration, m the current level, $\mathbf{u}_{\text{gt}} = (u_{\text{gt}}, v_{\text{gt}})^\top$ the ground-truth flow, and $\mathbf{u}_{\text{est}}^{(m,t-1)} = (u_{\text{est}}^{(m,t-1)}, v_{\text{est}}^{(m,t-1)})^\top$ the flow estimate produced in the previous iteration, $t-1$. The thresholds $r_u^{(m)}$ and $r_v^{(m)}$ are determined by the receptive field size of the respective level m. Given

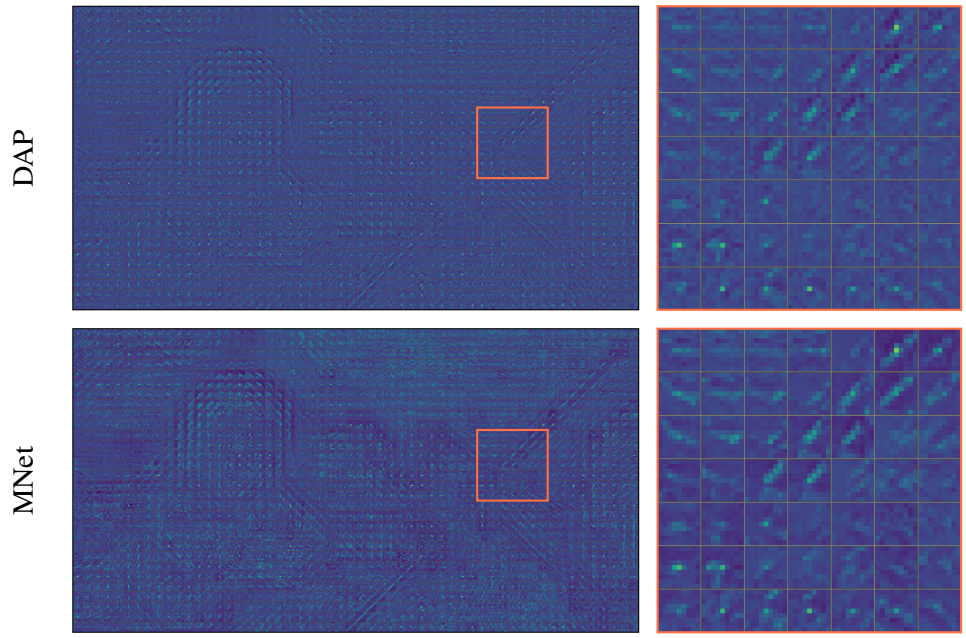

Figure 6.15: Cost volume slices for the three-level coarse-to-fine RAFT+DICL method with 1×1 convolutional matching network. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP), with marked excerpts to the right.

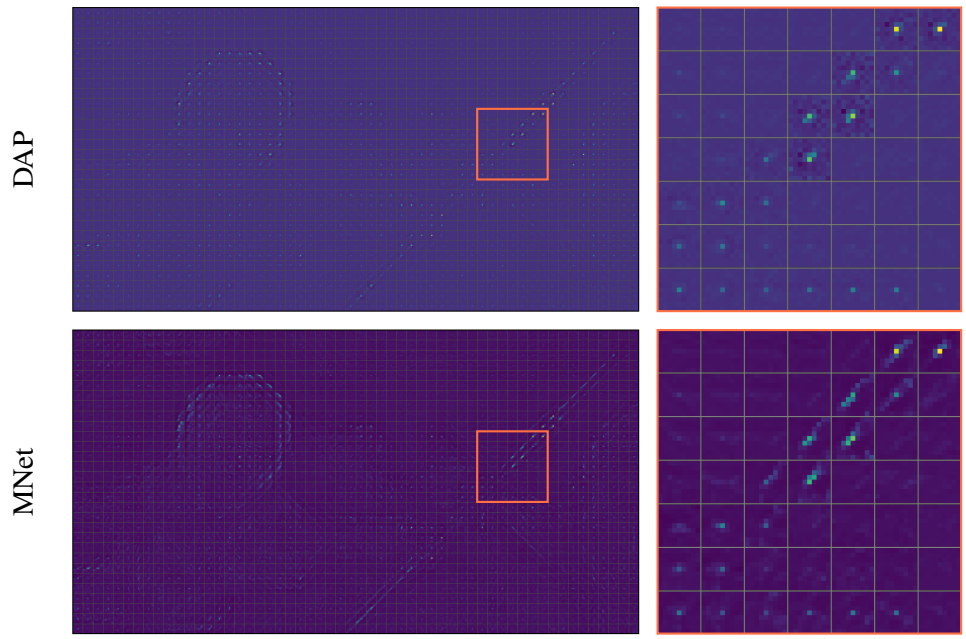

Figure 6.16: Cost volume slices for the three-level coarse-to-fine RAFT+DICL method with 1×1 convolutional matching network and direct cost regression. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP), with marked excerpts to the right.

a lookup range R_u for level m, we can compute the corresponding threshold as $r_u^{(m)} = R_u \cdot 2^m$. Note that the flow estimate produced in the previous iteration is only used to determine the set of pixels on which to evaluate the loss on, and does not play any other role in gradient computation.

We can then use the set defined in Equation (6.1) as basis for any of our previously defined cost functions, such as the standard RAFT sequence loss (Section 3.1.2) or our multi-sequence loss (Section 4.4), replacing Ω therein. For methods trained with the multi-sequence loss, such as our direct cost regression approaches, we can additionally decide whether we want to apply loss restriction to the full loss term or only to an individual sequence, for example the one created by the soft-argmin regression.

6.2.4 Results

While visual inspection of the cost volume can tell us about potential problems within our approaches, it is no replacement for the classical error measures applied to the full models and, in particular, the endpoint error (Section 2.3.1). In this subsection, we evaluate variations of the cost regression, gradient stopping, and loss restriction approaches introduced above. We will again mainly focus on the three-level coarse-to-fine method. In the given order, we will discuss base results, DAP layer placements, results for smaller flow magnitudes, loss restriction, gradient stopping, and sharing of DICL modules. At the end of this subsection, we will briefly investigate whether this cost regression approach can also benefit our two-level RAFT+DICL method. As we found that results are generally fairly close, we also evaluate on a mixed Sintel training set similar to the one employed by Teed and Deng [TD20] for training RAFT (adapted to use the MaskFlowNet [ZSD+20] validation split as discussed in Section 4.5), which we will refer to as "Sintel Mix".

For our evaluations, we compare methods trained without direct cost regression and their corresponding counterparts trained with direct cost regression, applied in different training stages. As this regression is intended to be a means of guidance for the cost volume, our experiments specifically focus on using it only in the initial stages. We mark methods trained fully to the end with cost regression with "CR", methods where cost regression has been dropped with the start of the Sintel stage with "CR/S" and methods where cost regression has been dropped with the start of the FlyingThings3D stage with "CR/T". For a small selection of the models shown here, we provide visualizations of the estimated flow and corresponding per-pixel endpoint error in Appendix C.

Base Results

Before we begin with the evaluations targeting our coarse-to-fine RAFT+DICL approaches, we pose the question whether cost regression can also benefit our coarse-to-fine RAFT baseline. Theoretically, as this baseline does not exhibit any of the issues encountered within our RAFT+DICL approaches, direct soft-argmin regression should not provide many benefits. It may be possible that cost regression helps somewhat more directly in forming the feature representation, however it may equally be a source of disturbances, pulling costs into a representation that is less suited for the matching network and more suited for the direct regression itself. As we can see in Table 6.1, the latter seems to be the case. Results of plain coarse-to-fine RAFT are notably better than the version with cost regression in all instances.

Method	Parameters	Chairs2	Things		Sintel	
	Channels	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-3)	256	1.2092	1.5938	3.0188	2.1514	3.1364
RAFT (CtF-3)	32	1.2814	1.6683	3.0209	2.2031	3.3963
RAFT (CtF-3, FS)	32	1.2785	1.7076	3.0063	2.0179	3.3210
RAFT (CtF-3, CR)	256	1.2906	1.6882	3.0585	2.1996	3.4476

Table 6.1: Results for the three-level coarse-to-fine (CtF-3) RAFT method using direct cost regression (CR). We compare results with RAFT baselines across different number of feature channels and optional feature sampling (FS) instead of cost sampling. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

Method	Parameters	Chairs2	Things		Sintel		Sintel Mix	
	DAP	*Chairs2*	*S/clean*	*S/final*	*S/clean*	*S/final*	*S/clean*	*S/final*
RAFT (CtF-3)	N/A	**1.2092**	**1.5938**	3.0188	2.1514	**3.1364**	**1.8385**	3.3207
RAFT (CtF-3, FS)	N/A	1.2785	1.7076	**3.0063**	**2.0179**	3.3210	1.9394	**3.3024**
R+D (CtF-3)	✓	1.2745	1.7436	3.0417	2.0666	3.2351	1.8810	3.4520
R+D (CtF-3, CR)	✓	1.2671	**1.6991**	3.1063	2.0491	3.2799	**1.8378**	**3.3307**
R+D (CtF-3, CR/S)	✓	1.2671	**1.6991**	3.1063	2.0227	3.2623	1.8829	3.3497
R+D (CtF-3, CR/T)	✓	1.2671	1.7761	**3.0580**	**1.9882**	**3.1867**	1.9515	3.3702
R+D (CtF-3, CR)	✗	1.2630	**1.7440**	3.1373	2.1414	3.1507	**1.9371**	**3.3706**
R+D (CtF-3, CR/S)	✗	1.2630	**1.7440**	3.1373	2.0423	**3.1197**	1.9375	3.4035
R+D (CtF-3, CR/T)	✗	1.2630	1.7427	**3.0445**	**1.9961**	3.3905	1.9845	3.3961
R+D (CtF-3, CR)	CR	1.2781	1.7349	2.9519	2.1117	3.0929	1.9794	3.1646
R+D (CtF-3, CR/S)	CR	1.2781	1.7349	2.9519	2.0868	**2.9569**	1.9576	**3.1038**
R+D (CtF-3, CR/T)	CR	1.2781	**1.6867**	**2.9379**	**2.0505**	2.9576	**1.9213**	3.1097

Table 6.2: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using direct cost regression (CR). Models were fine-tuned by disabling cost regression starting with the FlyingThings3D (CR/T) or Sintel/Sintel Mix (CR/S) stage. We compare results for different displacement-aware projection (*DAP*) types, specifically DAP enabled (✓), DAP disabled (✗), and DAP moved into the cost regression module (CR). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

Results for our three-level coarse-to-fine RAFT+DICL approaches are shown in Table 6.2. We use the same training parameters as before (specifically, we again use a learning rate of 5×10^{-5} for our three-level coarse-to-fine approaches). In general, cost regression seems to improve results somewhat compared to the non-regression approach. However, specifics differ between evaluation datasets and training stages. Further, differences are generally fairly small and improvements are not sufficient to make our methods notably better than (and in many cases even to make them on-par with) their RAFT baselines (especially considering the additional requirements and through these implied limitations of our models). For the Things/Sintel (final) and Sintel/Sintel (final) evaluations, it seems best to enable cost regression only while pre-training on FlyingChairs2, whereas enabling cost regression throughout all training stages seems to be best for the Sintel Mix stage.

DAP Layer Placement

Table 6.2 also shows results for different positions of the displacement-aware projection (DAP) layer. In particular, we experiment with three variations: For the first variation, we choose the standard placement as part of the DICL module. Here, costs are computed via both matching cost network and DAP layer, after which they are used in both the recurrent refinement network and the soft-argmin cost regression. For the second variation, we disable the DAP layer, computing costs only via the matching network. Third and last, we move the DAP layer into the cost regression module. In this variation, costs produced by the matching network are directly forwarded to the refinement network without any DAP layer in between. The cost regression, which now takes in the matching network results, however, then employs an exclusive DAP layer applied directly before the soft-argmin computation.

Generally, results get slightly worse when disabling the DAP layer, indicating that it has indeed a valid use in our RAFT+DICL approaches, at least when combined with direct cost regression. A notable exception to this can be found in the RAFT+DICL approach trained with cost regression and its Sintel/Sintel (final) evaluation. We believe it to be possible that removing the DAP layer gives the refinement network more direct supervision over the costs (as corroborated by Figure 6.14), which might lead to this improvement.

When moving the DAP layer into the cost regression module, results notably get better on the "final" evaluation sets for all stages, even beating our RAFT baselines. This, however, seems to come at the cost of a slightly worse performance on the "clean" evaluation sets. Two of the overall best approaches seem to consist of moving the DAP layer into the cost regression module and employing cost regression only on the FlyingChairs2 or FlyingChairs2 and FlyingThings3D stages. We theorize that the DAP layer acts as an intermediary layer in this configuration, improving compatibility between the soft-argmin regression and the matching network.

Results for Smaller Flow Magnitudes

Similar to our previous evaluations in Chapter 5 and, in particular, Section 5.4.2, we can now again evaluate our methods on pixels with a limited ground-truth flow magnitude. In particular, we are again interested in the case where flow is restricted to the receptive field of the model. For the coarsest level (and first iteration) of the three-level coarse-to-fine method, this (approximately) represents flow below or equal to a magnitude of 128 px.

Results for this are shown in Table 6.3. Similar to our observations in Section 5.4.2, the tested RAFT baseline performs notably better than our RAFT+DICL approaches on the FlyingChairs2, FlyingThings3D, and basic Sintel stages when (approximately) restricting the flow to the receptive field. A new exception to this can be found in the Things/Sintel (final) evaluation. However, here results of the respective RAFT+DICL approach with cost regression are only slightly better, whereas its results are worse across the other mentioned stages.

On the mixed Sintel training stage, results look considerably different. Here, our RAFT+DICL approaches using cost regression start to perform better than their respective RAFT and RAFT+DICL (without cost regression) baselines when restricting flow to 128 px. As the evaluation sets used in

Method	Eval.	Chairs2	Things		Sintel		Sintel Mix	
	u-*max*	*Chairs2*	*S/clean*	*S/final*	*S/clean*	*S/final*	*S/clean*	*S/final*
RAFT (CtF-3)	∞	**1.2092**	**1.5938**	**3.0188**	2.1514	**3.1364**	1.8385	**3.3207**
R+D (CtF-3)	∞	1.2745	1.7436	3.0417	2.0666	3.2351	1.8810	3.4520
R+D (CtF-3, CR)	∞	1.2671	1.6991	3.1063	2.0491	3.2799	**1.8378**	3.3307
R+D (CtF-3, CR/S)	∞	1.2671	1.6991	3.1063	2.0227	3.2623	1.8829	3.3497
R+D (CtF-3, CR/T)	∞	1.2671	1.7761	3.0580	**1.9882**	3.1867	1.9515	3.3702
RAFT (CtF-3)	256	**1.2092**	**1.4965**	**2.9058**	1.8671	**2.8768**	1.6330	**3.0500**
R+D (CtF-3)	256	1.2745	1.6558	2.9245	1.8699	2.9680	1.6827	3.1809
R+D (CtF-3, CR)	256	1.2671	1.6021	2.9929	1.8271	2.9886	**1.6134**	3.0527
R+D (CtF-3, CR/S)	256	1.2671	1.6021	2.9929	1.8137	2.9820	1.6369	3.0729
R+D (CtF-3, CR/T)	256	1.2671	1.6635	2.9390	**1.7593**	2.9103	1.7315	3.0861
RAFT (CtF-3)	128	**1.1849**	**1.2074**	2.4010	**1.4052**	**2.2633**	1.3426	2.5215
R+D (CtF-3)	128	1.2497	1.3280	2.4066	1.5104	2.3605	1.3543	2.5340
R+D (CtF-3, CR)	128	1.2396	1.2993	2.4566	1.4492	2.4253	1.3126	**2.4581**
R+D (CtF-3, CR/S)	128	1.2396	1.2993	2.4566	1.4359	2.4291	**1.2741**	2.4859
R+D (CtF-3, CR/T)	128	1.2396	1.3376	**2.3981**	1.4175	2.3550	1.3095	2.4983
RAFT (CtF-3)	64	**0.8906**	**1.0202**	2.0425	**1.0401**	**1.7129**	1.0475	1.9592
R+D (CtF-3)	64	0.9394	1.1186	2.0453	1.1684	1.7877	1.0873	1.9055
R+D (CtF-3, CR)	64	0.9235	1.0932	2.0715	1.1294	1.8088	1.0787	**1.8742**
R+D (CtF-3, CR/S)	64	0.9235	1.0932	2.0715	1.1275	1.7619	**1.0307**	1.8801
R+D (CtF-3, CR/T)	64	0.9235	1.1281	**2.0345**	1.1220	1.7328	1.0569	1.8980

Table 6.3: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using direct cost regression (CR), considering only ground-truth flow below or equal to the respective magnitude **u**-*max* (in pixel). Models were fine-tuned by disabling cost regression starting with the FlyingThings3D (CR/T) or Sintel/Sintel Mix (CR/S) stage. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

combination with the Sintel and Sintel Mix stages are equal, this difference in behavior cannot be explained via the evaluation data itself, showing that gaining access to the full performance of our RAFT+DICL models may require further training improvements and considerations.

Loss Restriction

Table 6.4 shows results for the training loss restriction approach described in Section 6.2.3. We chose to apply loss restriction to both the loss term supervising the cost volume via the soft-argmin regression and the loss term supervising the complete model via the output produced by the refinement network.

On the FlyingThings3D and Sintel stages, loss restriction seems to improve results on the Sintel "final" evaluation sets to varying degrees, but at the same time worsen results slightly on the "clean" evaluation sets. On the Sintel Mix stage, results are generally worse. The strict focus on flow inside the receptive field during training could help improve feature embeddings, which may explain the performance gains on the "final" sets. We theorize that including flow outside the receptive field

Method	Parameters	Chairs2	Things		Sintel		Sintel Mix	
	Loss Restr.	*Chairs2*	*S/clean*	*S/final*	*S/clean*	*S/final*	*S/clean*	*S/final*
RAFT (CtF-3)	✗	**1.2092**	**1.5938**	3.0188	2.1514	**3.1364**	**1.8385**	3.3207
RAFT (CtF-3, FS)	✗	1.2785	1.7076	**3.0063**	**2.0179**	3.3210	1.9394	**3.3024**
R+D (CtF-3)	✗	1.2745	1.7436	3.0417	2.0666	3.2351	1.8810	3.4520
R+D (CtF-3, CR)	✗	1.2671	**1.6991**	3.1063	2.0491	3.2799	**1.8378**	**3.3307**
R+D (CtF-3, CR/S)	✗	1.2671	**1.6991**	3.1063	2.0227	3.2623	1.8829	3.3497
R+D (CtF-3, CR/T)	✗	1.2671	1.7761	**3.0580**	**1.9882**	**3.1867**	1.9515	3.3702
R+D (CtF-3, CR)	✓	1.3280	1.7987	2.9971	2.0763	**3.0592**	1.9358	3.3787
R+D (CtF-3, CR/S)	✓	1.3280	1.7987	2.9971	2.0360	3.1203	**1.9113**	**3.3044**
R+D (CtF-3, CR/T)	✓	1.3280	**1.7929**	**2.9733**	**2.0246**	3.0701	2.0265	3.4357

Table 6.4: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using direct cost regression (CR) and a restricted training loss (*Loss Restr.*). Models were fine-tuned by disabling cost regression starting with the FlyingThings3D (CR/T) or Sintel/Sintel Mix (CR/S) stage. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

Method	Parameters		Chairs2	Things		Sintel	
	Grad. Stop	*Restr.*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (CtF-3)	✗	✗	**1.2092**	**1.5938**	3.0188	2.1514	**3.1364**
RAFT (CtF-3, FS)	✗	✗	1.2785	1.7076	**3.0063**	**2.0179**	3.3210
R+D (CtF-3, CR)	✗	✗	**1.2671**	**1.6991**	3.1063	**2.0491**	3.2799
R+D (CtF-3, CR)	✗	✓	1.3280	1.7987	**2.9971**	2.0763	**3.0592**
R+D (CtF-3, CR)	✓	✗	1.3835	1.8591	3.1749	2.2739	3.3822
R+D (CtF-3, CR)	✓	✓	1.3869	1.8646	3.1088	2.1715	3.1508

Table 6.5: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using direct cost regression (CR) and gradient stopping (*Grad. Stop*) for the cost/correlation module. When the gradient is stopped, cost/correlation scores are only supervised by the cost regression. We compare results with and without training loss restriction (*Restr.*). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

could, however, also act as a regulatory measure, which may explain the relatively worse results on the "clean" sets. Further evaluations where loss restriction is applied to the cost regression term only may be helpful.

Gradient Stopping

Table 6.5 shows results for direct cost regression combined with gradient stopping. In this configuration, we prevent the gradient from being propagated back from refinement network to cost learning module, causing the latter to be exclusively supervised by direct cost regression. We additionally explore variations trained with loss restriction (again applied to both network parts) as this more closely resembles the original DICL training protocol.

Method	Parameters		Chairs2	Things		Sintel		Sintel Mix	
	Shared	*DAP*	*Chairs2*	*S/clean*	*S/final*	*S/clean*	*S/final*	*S/clean*	*S/final*
RAFT (CtF-3)	N/A	N/A	**1.2092**	**1.5938**	3.0188	2.1514	**3.1364**	**1.8385**	3.3207
RAFT (CtF-3, FS)	N/A	N/A	1.2785	1.7076	**3.0063**	**2.0179**	3.3210	1.9394	**3.3024**
R+D (CtF-3)	✗	✓	1.2745	1.7436	3.0417	**2.0666**	3.2351	**1.8810**	**3.4520**
R+D (CtF-3)	✓	✓	**1.2344**	**1.6663**	**3.0089**	2.1456	**3.1710**	1.9360	3.4842
R+D (CtF-3, CR)	✗	✓	1.2671	**1.6991**	3.1063	2.0491	3.2799	**1.8378**	**3.3307**
R+D (CtF-3, CR/S)	✗	✓	1.2671	**1.6991**	3.1063	2.0227	3.2623	1.8829	3.3497
R+D (CtF-3, CR/T)	✗	✓	1.2671	1.7761	**3.0580**	**1.9882**	**3.1867**	1.9515	3.3702
R+D (CtF-3, CR)	✓	✓	1.2462	1.6413	3.0646	2.0370	3.1396	**1.8530**	3.4279
R+D (CtF-3, CR/S)	✓	✓	1.2462	1.6413	3.0646	2.0261	**3.1336**	1.9123	3.3898
R+D (CtF-3, CR/T)	✓	✓	1.2462	**1.5973**	**3.0384**	**2.0183**	3.1511	1.8624	**3.3484**
R+D (CtF-3, CR)	✓	✗	1.2709	1.7288	**3.0134**	2.1559	**2.9847**	1.9606	3.1583
R+D (CtF-3, CR/S)	✓	✗	1.2709	1.7288	**3.0134**	2.1391	3.1029	1.9781	**3.1575**
R+D (CtF-3, CR/T)	✓	✗	1.2709	**1.7010**	3.0304	**2.0467**	3.1066	**1.9022**	3.2970

Table 6.6: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using direct cost regression (CR) and a shared DICL module (*Shared*). Models were fine-tuned by disabling cost regression starting with the FlyingThings3D (CR/T) or Sintel/Sintel Mix (CR/S) stage. We compare results with displacement-aware projection (*DAP*) enabled and disabled. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

Results show that gradient stopping leads to considerably worse performance, regardless of loss restriction. This indicates that the gradient from the refinement network is indeed beneficial and, maybe more importantly, that the refinement network may not work well with certain cost representations, such as the one forced by the soft-argmin regression. Further evaluations, such as combining gradient stopping and moving the DAP layer to the cost regression module may prove useful to understand this better. This could provide the refinement network with costs not subject to the rather soft-argmin specific refocusing, spreading values out into a broader Gaussian-like shape (which can be observed in Figure 6.13).

Sharing DICL Modules

In Section 5.4.6, we have seen that sharing weights between DICL modules can improve results. Therefore, we also evaluate weight sharing in combination with our direct cost regression approach. We again share both matching cost network and DAP layer weights, and further explore disabling the DAP layer while doing so.

Results, given in Table 6.6, show that sharing DICL module weights across levels again leads to performance gains in most cases. Exceptions to this can be found in the evaluations of the mixed Sintel stage. Disabling the DAP layer leads to yet more gains on the Sintel "final" evaluation sets, but, as also noticed before, in turn results in generally worse performance on the "clean" sets. Somewhat

Method	Eval.	Chairs2	Things		Sintel		Sintel Mix	
	u-max	*Chairs2*	*S/clean*	*S/final*	*S/clean*	*S/final*	*S/clean*	*S/final*
RAFT (CtF-3)	∞	**1.2092**	**1.5938**	3.0188	2.1514	3.1364	**1.8385**	**3.3207**
R+D (CtF-3)	∞	1.2344	1.6663	**3.0089**	2.1456	3.1710	1.9360	3.4842
R+D (CtF-3, CR)	∞	1.2462	1.6413	3.0646	2.0370	3.1396	1.8530	3.4279
R+D (CtF-3, CR/S)	∞	1.2462	1.6413	3.0646	2.0261	**3.1336**	1.9123	3.3898
R+D (CtF-3, CR/T)	∞	1.2462	1.5973	3.0384	**2.0183**	3.1511	1.8624	3.3484
RAFT (CtF-3)	128	**1.1849**	**1.2074**	2.4010	**1.4052**	2.2633	1.3426	**2.5215**
R+D (CtF-3)	128	1.2087	1.2780	2.3750	1.4803	**2.2604**	**1.3251**	2.5969
R+D (CtF-3, CR)	128	1.2191	1.2571	2.4150	1.4652	2.3468	1.3325	2.5981
R+D (CtF-3, CR/S)	128	1.2191	1.2571	2.4150	1.4772	2.3204	1.3732	2.5677
R+D (CtF-3, CR/T)	128	1.2191	1.2264	**2.3577**	1.4644	2.3037	1.3578	2.5411
RAFT (CtF-3)	64	**0.8906**	**1.0202**	2.0425	**1.0401**	1.7129	1.0475	1.9592
R+D (CtF-3)	64	0.9026	1.0603	2.0252	1.1279	1.6793	**1.0413**	1.8903
R+D (CtF-3, CR)	64	0.9008	1.0678	2.0386	1.1247	1.7353	1.0564	1.9220
R+D (CtF-3, CR/S)	64	0.9008	1.0678	2.0386	1.1168	1.7018	1.0758	1.8771
R+D (CtF-3, CR/T)	64	0.9008	1.0366	**1.9802**	1.1077	**1.6680**	1.0656	**1.8760**
RAFT (CtF-3)	32	**0.6733**	**0.9412**	**1.7605**	**1.1678**	**1.7416**	1.1988	2.1296
R+D (CtF-3)	32	0.6839	0.9785	1.7660	1.2574	1.7654	1.1792	1.9126
R+D (CtF-3, CR)	32	0.6769	1.0017	1.9737	1.2629	1.8325	**1.0911**	1.8871
R+D (CtF-3, CR/S)	32	0.6769	1.0017	1.9737	1.2528	1.7918	1.1374	1.9025
R+D (CtF-3, CR/T)	32	0.6769	0.9771	1.8626	1.1748	1.7877	1.1154	**1.8474**

Table 6.7: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using direct cost regression (CR) and a shared DICL module, considering only ground-truth flow below or equal to the respective magnitude **u**-*max* (in pixel). Models were fine-tuned by disabling cost regression starting with the FlyingThings3D (CR/T) or Sintel/Sintel Mix (CR/S) stage. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

notably, the method using a shared DICL module, a standard DAP layer, as well as cost regression during the FlyingChairs2 stage only is our first coarse-to-fine RAFT+DICL method providing results on par with its corresponding RAFT baseline for the Things/Sintel (clean) evaluation.

Comparing the approach with shared DICL module and (optional) soft-argmin regression against its RAFT baseline directly over multiple maximum flow magnitudes in Table 6.7 shows that there is no clearly superior method. Only when restricting evaluated flow to a magnitude of 32 px and below can we find RAFT dominating on multiple training stages. Generally, our RAFT+DICL approach seems to perform, in relation, better on the "final" evaluation sets.

Multi-Level RAFT+DICL

Lastly, we investigate whether direct cost regression can help resolve the problems encountered within our multi-level RAFT+DICL approaches. To this end, we apply cost regression to our two-level RAFT+DICL approach with pooling-based encoder. Specifically, a soft-argmin regression is applied to each cost volume level independently, leading to three output sequences in total. We chose

Method	Parameters		Chairs2	Things		Sintel	
	Levels	*Enc.*	*Chairs2*	*Sintel clean*	*Sintel final*	*Sintel clean*	*Sintel final*
RAFT (SL)	{3}	N/A	1.4209	2.3015	**3.5155**	2.9718	4.5847
RAFT+DICL (SL)	{3}	N/A	**1.4103**	**2.2605**	3.6062	**2.7448**	**4.2528**
RAFT (baseline)	{3, 4}	pool	1.3118	2.0402	3.3270	2.5337	4.2979
RAFT+DICL (ML)	{3, 4}	CNN	**1.5512**	**2.2992**	**3.4465**	3.1343	4.4985
RAFT+DICL (ML)	{3, 4}	pool	1.6012	2.4013	3.7188	3.0117	**4.3087**
RAFT+DICL (ML, CR)	{3, 4}	pool	1.6431	2.3729	3.6307	**2.9356**	4.6785

Table 6.8: Results for the two-level RAFT+DICL method using direct cost regression (CR). We compare results with single-level RAFT and RAFT+DICL baselines, a two-level RAFT baseline, as well our earlier two-level RAFT+DICL variants using different feature encoder types (*Enc.*). Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

a loss term multiplier of $\alpha_1 = 1$ for the main output sequence, $\alpha_2 = \lambda = 0.85$ for cost regression on the finest level, and $\alpha_3 = 0.72 \approx \lambda^2$ for cost regression on the coarsest one. Additionally, we share a single DICL module across both levels and disable the DAP layer to force a more direct control of the soft-argmin regression over the matching network. We trained our approach with an initial learning rate of 5×10^{-5} during the preparatory epoch.

As results in Table 6.8 show, accuracy improves over the prior two-level RAFT+DICL module with pooling-based encoder and shared DICL module on the FlyingThings3D stage and on the Sintel/Sintel (clean) evaluation. Overall performance, however, is still worse in comparison to the single-level RAFT+DICL approach. Looking at a cost volume slice produced by this new approach (provided in Appendix B.1, specifically Figure B.3) shows that, even though we attempt to explicitly shape both cost volume levels, the coarser level still does not seem to show any qualitative matching-cost-like information. In addition to this, the finest cost volume level still shows a noticeable (but slightly different) checkerboard-like pattern. Stopping gradient propagation from refinement back to the matching network might help in both regards. Compared to the previous approaches, however, the finer level of the cost volume seems to have (at least visually) improved.

6.2.5 Summary

The additional flexibility introduced by the dynamically learned cost function has lead to difficulties in training. Section 6.2.1 has shown that these difficulties also extend to the quality of the cost volume, leading to artifacts and inconsistencies therein. To help guide the learning process towards a consistent representation of cost values, we proposed the use of a direct soft-argmin cost regression (Section 6.2.2) and an (optional) loss restriction (Section 6.2.3), both based on their respective counterparts introduced by Wang et al. [WZD+20]. Similar approaches using metric learning techniques, such as a contrastive loss term, could potentially be used as an alternative way of shaping the cost function (cf. Section 2.1.3).

Results (Section 6.2.4) combined with visual inspection (Section 6.2.2) indicate that forming the cost volume via direct cost regression can indeed improve performance by guiding the learned cost function towards a more consistent and a more artifact-free output. Further, in contrast to the

RAFT+DICL coarse-to-fine approaches without cost regression (cf. Section 5.4.2), models trained with cost regression do no longer significantly deteriorate in performance (relative to their baselines) when evaluation is restricted to the receptive field. While we assume that cost regression also lessens the severity of the stability issues discussed in Section 6.1, we have not explicitly investigated this. Concessions in learning rate, or further measures such as forward-backward batching, may still be necessary for more complex multiscale models. In particular, we were unable to significantly improve the RAFT-like multi-level approach, which still suffers from severe problems on coarser levels.

We found that the DAP layer generally helps to improve results, but can lead to noise-like artifacts in the cost representation. Additionally, it can lead to retention of checkerboard-like artifacts in the output of the matching cost network, which we found stem from its inherent structure. The DAP layer balances these out, therefore preventing the direct cost regression from acting against them. Disabling the DAP layer leads to an artifact-free output, which, however, comes at the price of an overall worse performance. Wang et al. [WZD+20] propose to disable the DAP layer during some initial training steps, which we believe may (possibly together with a larger batch size) be the required technique for (properly) removing both noise- and checkerboard-like artifacts.

Moving the DAP layer into the cost regression module, taking costs for the recurrent refinement network directly as outputs of the matching network, can improve results on the more complex Sintel "final" evaluation datasets, but trades this against mostly worse performance on the "clean" sets. We believe that, in this case, the DAP layer acts as an intermediary, making the representation of costs as imposed by the soft-argmin regression more compatible with the recurrent refinement network. Two separate DAP layers, one exclusive to the cost regression and one exclusive to the recurrent refinement network, may prove beneficial.

Restricting the full training loss (including both main and cost regression terms) to flow inside the receptive field can similarly improve results on the Sintel "final" evaluation sets, but degrade performance on the "clean" sets as well as on the mixed Sintel stage. Interestingly, this is opposite to our findings for the DICL baseline approaches (cf. Section 5.1.2). We believe that flow outside the receptive field may act as regularization, and that better results might be achieved by restricting the loss term for the cost regression only.

While we have shown that preventing gradient propagation from the recurrent refinement network back to the DAP layer and matching cost network can lead to a visually pleasing and (largely) artifact free cost volume in Section 6.2.2, doing so also leads to a significantly worse performance. This shows that the gradient signal provided by the recurrent refinement network is indeed beneficial and, further, that the refinement network may not work equally well with different cost representations. Accuracy improves slightly when gradient stopping is combined with loss restriction, however, it is still worse than without gradient stopping, providing further hints that loss restriction applied to the cost regression term only may prove beneficial. We conclude that gradient stopping may, at best, be useful for pre-training.

Lastly, we again investigated sharing of DICL module weights across coarse-to-fine levels. As before, doing so generally improves our results. Accuracy of the three-level coarse-to-fine RAFT+DICL approach is now overall on-par with its RAFT baseline. Experiments restricting evaluated flow to different maximum magnitudes show that there is no clearly superior method.

6.3 Improving the Matching Network

Except for the visual investigation of cost volumes in Section 6.2.1, we have so far exclusively used the original DICL matching network, proposed by Wang et al. [WZD+20]. Our general approach, however, is not bound to this network type and can, at least when ignoring potential hardware limitations, work with any imaginable network structure as a replacement to it. In this section, we will therefore explore a scaled-up version of the original matching network as well as the previously presented 1×1 convolutional alternative.

Both of the alternative networks are, in large, based on the original DICL matching network. For the scaled-up version, we simply doubled the number of channels in each layer, except for the last one, producing the final cost values. In addition, we also doubled the number of channels in the feature representation, i.e., the channels input to this network. This allows us to check whether the DICL module may pose as a bottleneck due to the reduced number of feature channels in comparison to RAFT. For the 1×1 convolutional alternative, we, as the name implies, set the size of all convolution kernels to be one in each direction. In addition, we removed the down- and upsampling layers employed by the original DICL module, meaning the resulting network has two fewer layers. The number of channels was kept the same.

Results for both networks, used within the three-level coarse-to-fine RAFT+DICL framework, are shown in Table 6.9. To provide a better context, we also include results for the 64-channel version of coarse-to-fine RAFT+DICL, previously shown in Section 5.4.3. This version simply uses an adapted input layer of the DICL module to accommodate the larger number of feature channels, but is otherwise kept unchanged. We can see that this can already lead to some improvements over the (default) 32-channel variant, most notably on the “final” evaluation sets. Results on the “clean” sets are mixed and, in some cases, even degrade. The network may trade accuracy on the “clean” sets against accuracy on the “final” ones.

The method using the scaled-up DICL module achieves further gains on the FlyingThings3D and (plain) Sintel training stages, with a fine-tuned version consistently outperforming our RAFT baselines on three of the four evaluation sets and being on-par on the remaining one (Things/Sintel-clean). Interestingly, results are less clear on the mixed Sintel stage. We nevertheless believe that this shows a limitation of the DICL-based cost learning approach in the number of channels, both of the feature representation and of the matching cost network. Whether the achieved gains, however, are worth the additional computational costs and heightened memory requirements is a more difficult question to answer. We therefore believe that other opportunities for improvement, such as weight sharing and DAP layer placement, should be explored first. A compromise could be achieved by scaling-up the feature representation and matching cost networks on coarser levels only, which, however, would prevent sharing of weights across (all) levels.

Lastly, we found that the 1×1 convolutional variant of the matching network still performs comparatively well, even though it has a significantly less complex structure. Quite notably, best results (with the single exception of the Sintel Mix/Sintel-final evaluation) are achieved without any use of direct soft-argmin cost regression. This may indicate two things, which we have also noted before:

First, cost regression may not guide the matching network towards the optimal cost function for the refinement network. Potential solutions for this could be lowering the impact of the cost regression by confining its use to the very first training steps only (similar to but more extreme than our fine-tuning approaches) or reducing its contribution to the complete training loss by reducing the

Method	Parameters		Chairs2	Things		Sintel		Sintel Mix	
	Corr.	*Chn.*	*Chairs2*	*S/clean*	*S/final*	*S/clean*	*S/final*	*S/clean*	*S/final*
RAFT (CtF-3)	dot	256	**1.2092**	**1.5938**	3.0188	2.1514	**3.1364**	**1.8385**	3.3207
RAFT (CtF-3)	dot	64	1.2610	1.6016	**3.0027**	2.0333	3.1963	1.8764	**3.2872**
RAFT (CtF-3)	dot	32	1.2814	1.6683	3.0209	2.2031	3.3963	1.9651	3.6756
RAFT (CtF-3)	FS/dot	32	1.2785	1.7076	3.0063	**2.0179**	3.3210	1.9394	3.3024
R+D (CtF-3)	DICL	32	1.2745	1.7436	**3.0417**	2.0666	3.2351	1.8810	3.4520
R+D (CtF-3, CR)	DICL	32	**1.2671**	**1.6991**	3.1063	2.0491	3.2799	**1.8378**	**3.3307**
R+D (CtF-3, CR/S)	DICL	32	**1.2671**	**1.6991**	3.1063	2.0227	3.2623	1.8829	3.3497
R+D (CtF-3, CR/T)	DICL	32	**1.2671**	1.7761	3.0580	**1.9882**	**3.1867**	1.9515	3.3702
R+D (CtF-3)	DICL	64	**1.2667**	1.7018	**2.9868**	**2.0687**	3.2501	**1.8554**	3.3023
R+D (CtF-3, CR)	DICL	64	1.2810	1.7453	3.0230	2.1357	3.1191	1.9012	3.2190
R+D (CtF-3, CR/S)	DICL	64	1.2810	1.7453	3.0230	2.1420	**3.0252**	1.9254	**3.1740**
R+D (CtF-3, CR/T)	DICL	64	1.2810	**1.6812**	3.0299	2.1539	3.1369	1.9067	3.2036
R+D (CtF-3)	DICL×2	64	1.2755	1.7261	3.0140	2.0261	3.2259	1.8883	**3.2134**
R+D (CtF-3, CR)	DICL×2	64	**1.2433**	**1.5972**	2.9999	2.0547	3.1741	**1.8676**	3.2778
R+D (CtF-3, CR/S)	DICL×2	64	**1.2433**	**1.5972**	2.9999	2.0114	3.2528	1.8742	3.3139
R+D (CtF-3, CR/T)	DICL×2	64	**1.2433**	1.6288	**2.9254**	**1.9576**	**3.0824**	1.9027	3.3006
R+D (CtF-3)	1×1	32	**1.2501**	**1.6550**	**3.0349**	**2.0785**	**3.2297**	**1.8381**	3.4132
R+D (CtF-3, CR)	1×1	32	1.2514	1.7293	3.0631	2.2701	3.2635	2.0157	**3.2024**
R+D (CtF-3, CR/S)	1×1	32	1.2514	1.7293	3.0631	2.1858	3.2620	1.9287	3.3636
R+D (CtF-3, CR/T)	1×1	32	1.2514	1.7258	3.0552	2.1439	3.3136	1.9221	3.2984

Table 6.9: Results for the three-level coarse-to-fine (CtF-3) RAFT+DICL method using different matching network types. We compare the RAFT baseline models with dot product and (optional) feature sampling (FS) with RAFT+DICL modules using the standard DICL matching network, a version with two times the feature channels per layer (DICL×2), and a 1×1 convolutional replacement. We evaluate methods with and without direct cost regression (CR), optionally fine-tuned starting with FlyingThings3D (CR/T) and Sintel/Sintel Mix (CR/S) stages. Results are given as average endpoint error for training stages (in bold) over evaluation datasets (in italic).

corresponding multiplier. Both may, however, lead the model into a local optimum. Using a different and less constrained approach to supervise the cost volume, potentially relating to metric learning techniques, may also be possible.

Second, the difficulties regarding a consistent cost representation are, at least in part, specific to the matching cost network being used. While consistency across levels is a problem that we expect to be independent of the network, the level-internal issues, such as internal consistency and checkerboard-like patterns, are not. The checkerboard-like patterns, in particular, seem to be caused by the transposed convolutional layer used by the DICL matching network to upsample cost values. Additional experiments with non-upsampling network variations may therefore prove beneficial.

7 Conclusions

7.1 Conclusions

In this thesis, we have discussed the estimation of optical flow via the combination of two recent methods: RAFT by Teed and Deng [TD20], estimating flow iteratively and incrementally via a recurrent refinement network, and DICL by Wang et al. [WZD+20], using a learned convolutional network to produce better matching costs. To this end, Part I presented the fundamental concept of optical flow in Chapter 1, discussed preliminaries such as related work, training and evaluation data, as well as error measures in Chapter 2, and provided a more detailed look at both RAFT and DICL individually in Chapter 3. Moreover, in Section 3.3 we derived a generic RAFT-based framework, allowing for the integration of arbitrary (optionally learned) but differentiable cost functions.

We then used this novel framework in combination with the DICL cost learning approach as the basis for Part II. In Chapter 4, we proposed three specific instances of this framework, shaped into a single-level (Section 4.1), a RAFT-like multi-level (Section 4.2), and a more classical coarse-to-fine (Section 4.3) RAFT+DICL method. We evaluated these approaches in combination with their respective RAFT and DICL baselines in Chapter 5. Chapter 6 builds on top of these results and investigates the difficulties (in particular stability issues) we encountered, the means with which they can be prevented, and ways in which results can be improved.

Due to the results obtained in Chapter 5, our investigations focused mainly on the three-level coarse-to-fine RAFT+DICL approach, finding that, in some cases, RAFT-based coarse-to-fine approaches can out-perform the standard RAFT methods using a hierarchical multi-level cost volume. Our RAFT-like multi-level approaches, however, suffered from systemic problems, which we were unable to resolve. In particular, we found that the added flexibility of our approach can lead to significant instabilities during the initial training steps (Section 6.1) as well as suboptimal, and at times even internally conflicting, cost representations (Section 6.2). The latter can, at least in part, be caused by specific network structures, such as the transposed convolution used in the DICL matching cost network. By guiding the learning process of the cost function via a direct soft-argmin regression (Section 6.2.2), we were able to show improvements thereof. We, however, have also seen that the soft-argmin cost regression approach may not always lead to the cost representation best suited for the recurrent refinement approach.

We further found that the placement of the displacement-aware projection (DAP) layer can have a considerable impact on results. Making the DAP layer exclusive to the cost regression can, in some cases, lead to significant improvements. We believe that it can act as an intermediary, improving compatibility between cost regression and matching network. Using two separate instances thereof, one exclusive to the recurrent refinement network and another one exclusive to the cost regression, might prove beneficial. More training considerations and investigations may be required, as we have

shown that the DAP layer can be prone to introducing artifacts into the cost volume. Additional gains can be achieved by sharing weights of both recurrent refinement network and DICL modules across spatial levels.

A considerable limitation of our overall approach and framework is its high memory requirement. During training, this limits our models in two notable ways: First, in their batch size and, second, in the number of recurrent iterations performed. The former may impact the usefulness of normalization layers, such as batch normalization, which we nevertheless still found to be beneficial. The latter has shown to impact convergence of the recurrent estimation process. In particular, when trained with too few iterations, results diverge slightly during evaluation when increasing the numbers of iterations. For our coarse-to-fine approaches, we believe that this could be resolved by not re-initializing the hidden state on each new level, but instead upsampling it from the previous (coarser) one, therefore retaining its information. The in comparison to RAFT considerably smaller number of channels, both in the feature representation and in the matching cost network, has also shown to be a limiting factor. It is similarly constrained by memory and computational requirements.

Two of our best methods are obtained by making the DAP layer exclusive to the cost regression (Table 6.2) and by sharing DICL and RNN module weights across levels (Tables 6.6 and 6.7). Individually, they are both roughly equal regarding accuracy and in only some instances outperform our RAFT baselines. We believe, however, that, combined and extended with the propositions given above, they may be able to show notable gains over said baselines.

7.2 Future Work

While we were yet unable to show significant and consistent improvements over our RAFT baseline methods, we believe that the general RAFT-based cost-learning framework derived in this thesis shows the potential for such. Ideas for future work can be roughly divided into three categories: Evaluations (Section 7.2.1), adjustments (Section 7.2.2) and extensions (Section 7.2.3).

7.2.1 Evaluations

First, we believe that additional evaluations are needed. Not only for the aforementioned combination of our best variations so far, but also using an updated full-scale training strategy (similar to the original RAFT training strategy, which we down-scaled for ease-of-use due to the heightened requirements of our approaches), including a larger batch size, batch normalization, (potentially) more training iterations, and hyperparameter tuning. While somewhat more time and resource consuming, we believe that such a strategy will be essential to achieve state-of-the-art results.

To allow for a more broadly spread investigation of our approaches, we have, throughout this thesis, limited ourselves to evaluations on the Sintel dataset only. This dataset is, however, of synthetic nature. We believe that evaluations on real-world datasets, in particular the KITTI ones, are crucial as these may better reflect the actual use-cases of optical flow methods. Submission to benchmarks, specifically the Sintel and KITTI ones, is an additional next step that should be taken once a sufficiently good model and training strategy (including hyperparameters) have been found. This then provides for direct comparability with other state-of-the-art approaches.

Lastly, we want to note that our approaches should (in subsequent work) be compared to (and evaluated against) the Separable Flow method proposed by Zhang et al. [ZWPT21] as they both share the same larger goal: Improvement of the 4D cost volume used by RAFT, by providing more accurate and focused cost values via a learned approach. Ours does so by learning the cost function itself, whereas Separable Flow does so by aggregating costs over a semi-global context (combined with local refinement). They are, however, fundamentally incompatible with each other due to reasons of feasibility: Even though Separable Flow internally employs a decomposition into two separated (semi-global) 3D cost volumes to remain practicable, it requires the full all-pairs cost volume as input (to be decomposed). Feasibility of our proposed approach, in contrast, depends on only a subset of the cost volume being actually required.

7.2.2 Adjustments

A second aspect for future work are adjustments made to the model. This includes some of the aforementioned ideas, such as the use of weight standardization [QWL+20] instead of batch normalization. We also want to again emphasize hidden state upsampling as a potential means for our coarse-to-fine approaches to considerably reduce its dependency on larger number of iterations by retaining the hidden state of the recurrent refinement network across levels. Additionally, we believe that the cost-regression approach could be improved (besides hyperparameter tuning), potentially through techniques used in metric learning.

For our coarse-to-fine approaches, we chose a somewhat simplistic extension of the encoder used by RAFT. Replacing the feature and context encoder networks with more sophisticated versions, for example the GA-Net [ZPYT19] based encoder used by DICL [WZD+20] or the recently proposed detail-preserving residual feature pyramid modules by Long and Lang [LL22], should therefore be considered. This may provide higher quality feature embeddings, which in turn could reduce the dependency on the number of feature channels and channels in the matching cost network. Many of these smaller adjustments could also benefit the plain RAFT coarse-to-fine approach.

7.2.3 Extensions

Third and last, we believe that there are considerable amounts of possibilities to extend and make use of our general framework. Most importantly, this framework is not limited to the DICL matching cost network, but only by feasibility of respective replacements thereof. This creates the potential for various extensions: For example, replacement modules may produce arbitrary vectors instead of scalar cost values. Note that this, however, may yet again lead to further stability problems not easily resolved by the soft-argmin regression, due to which the specific network architecture needs to be chosen with care. In a more concrete example, the flow produced by the soft-argmin regression could be provided as input to the matching network.

Lastly, the recurrent approach could be extended to the matching costs by creating a feedback loop, adding a connection from the recurrent refinement network back to the cost learning module. Via this, costs could be adapted for each new iteration, which may further help refinement by focusing on different aspects in different iterations. This could also be extended to the feature representation itself: A network could be designed to reduce a more information-rich feature representation with a larger number of channels to a smaller representation with fewer channels using a channel-based

attention mechanism depending on inputs from the previous recurrent iteration. In each iteration, the model would thus be able to actively query for parts of the larger representation, choosing only the information relevant at this time, while keeping the resulting vectors smaller and therefore more feasible for subsequent cost computation via, for example, an adapted DICL module.

Appendices

A Stability Problems

A.1 Forward–Backward Batching: Speed of Convergence

To gauge the usefulness of forward-backward batching besides improving initial stability, we compare the training error of the initial preparatory epoch (as described in Section 4.5.1) between approaches trained with and without it. The particular question we want to answer is whether forward-backward batching can improve the speed of convergence, i.e., the speed with which the error is reduced. One may argue that using both forward and backward flow can cancel out biases, thus leading to more accurate batches, meaning a better estimation of the gradient, and therefore an improved gradient descent (i.e., optimizer) step. Having a better gradient estimate could then result in faster convergence of the model, i.e., a faster reduction of the training error. We therefore evaluate the training error across both RAFT and DICL baselines, trained with and without forward-backward batching. The results of which are shown in Figure A.1. They do not show any noticeable difference in training error, hence the aforementioned theory is false.

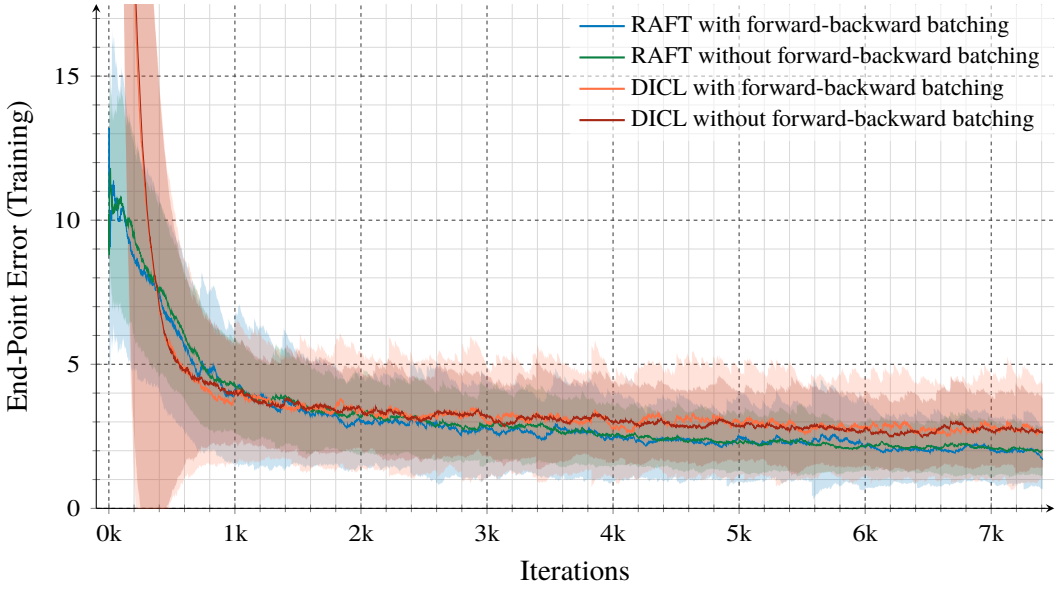

Figure A.1: Plot of the training error for RAFT and DICL baselines during the first (i.e., preparatory) epoch. Both models are trained once with and once without forward-backward batching to assess the impact thereof on model convergence. The training error is smoothed using an exponentially weighted moving average, with the shaded areas indicating the standard deviation. As indicated by the graphs, there is no significant impact on convergence speed.

A.2 Per-Level Convergence Issues in Four-Level RAFT

An additional visualization for estimates with masked cost volume levels in RAFT (cf. Section 6.1.4) is provided in Figure A.2. Notably, both original checkpoints provided by Teed and Deng [TD20] and our re-trained baseline exhibit similar issues, estimating largely incorrect flow when only levels 5 and 6 provide correct cost values (setting all other levels to zero). Both methods still work comparatively well when zeroing the finest cost volume level. Our four-level RAFT+DICL method, on the other hand, fails catastrophically at this.

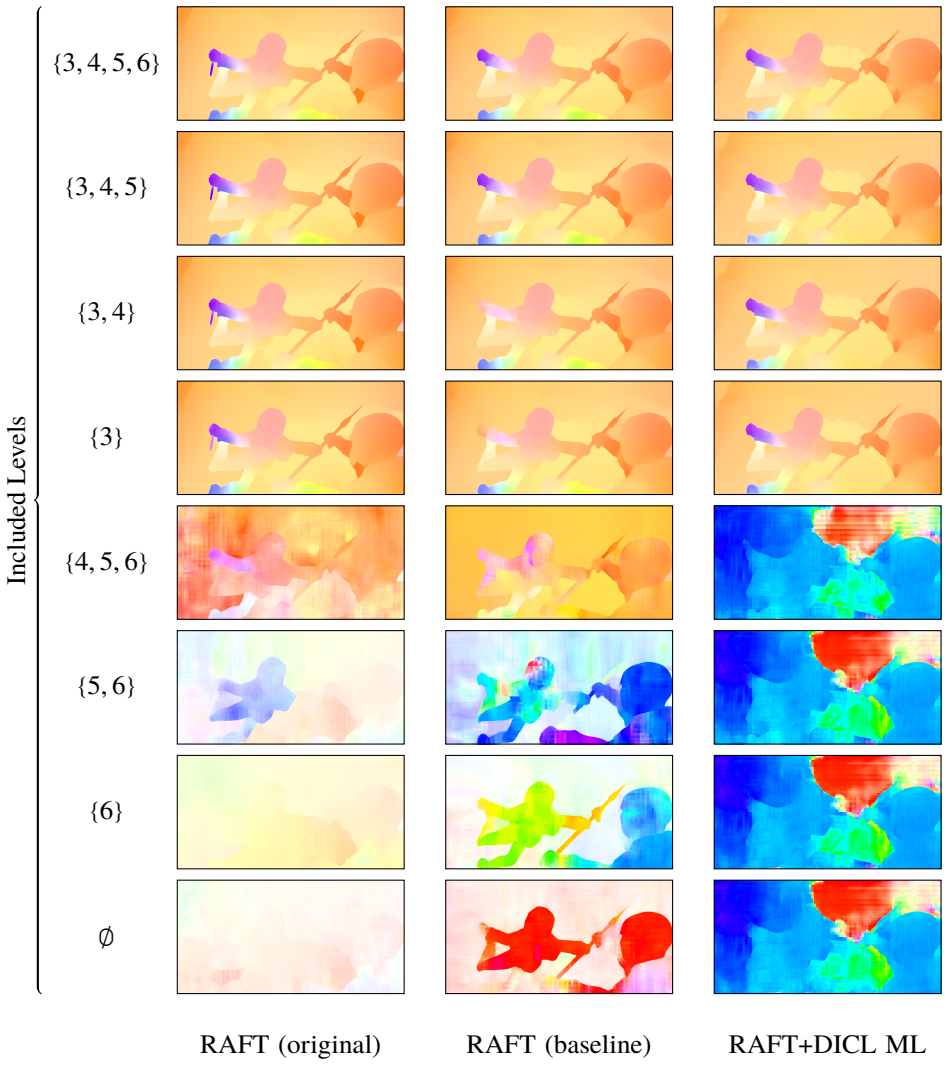

Figure A.2: Flow estimates of the four-level RAFT and RAFT+DICL approaches, including only specified cost volume levels. Excluded cost levels are set to zero. Results are taken after twelve (original) and four (baseline, ours) refinement iterations. The original and baseline RAFT models show mostly truthful flow when the finest level has been excluded (level 3), whereas results for our RAFT+DICL method are notably wrong.

B Additional Cost Volume Visualizations

B.1 Multi-Level RAFT+DICL with Two Levels

Additional cost volume visualizations for the two level RAFT+DICL approach are provided in Figures B.1 and B.2 for, respectively, the variant with CNN-based encoder and separate DICL modules, and the variant using a pooling-based encoder and a shared DICL module. A visualization for the approach trained with soft-argmin cost regression (employing a pooling-based encoder, sharing one single DICL module, and not using a DAP layer; as discussed in Section 6.2.4), is shown in Figure B.3. Note the lack of distinctive correlation or cost-like information in the coarser level of all shown approaches, as well as the different artifacts present in both levels.

B.2 Multi-Level RAFT+DICL with Four Levels

Additional cost volume visualizations for the four-level RAFT+DICL model are provided in Figures B.4 and B.5. Note the lack of any cost-/correlation-like information on the coarser levels as well as the artifacts, such as a checkerboard pattern, present in most levels. Notably, the coarsest and second-coarsest levels do not seem to contain much information at all.

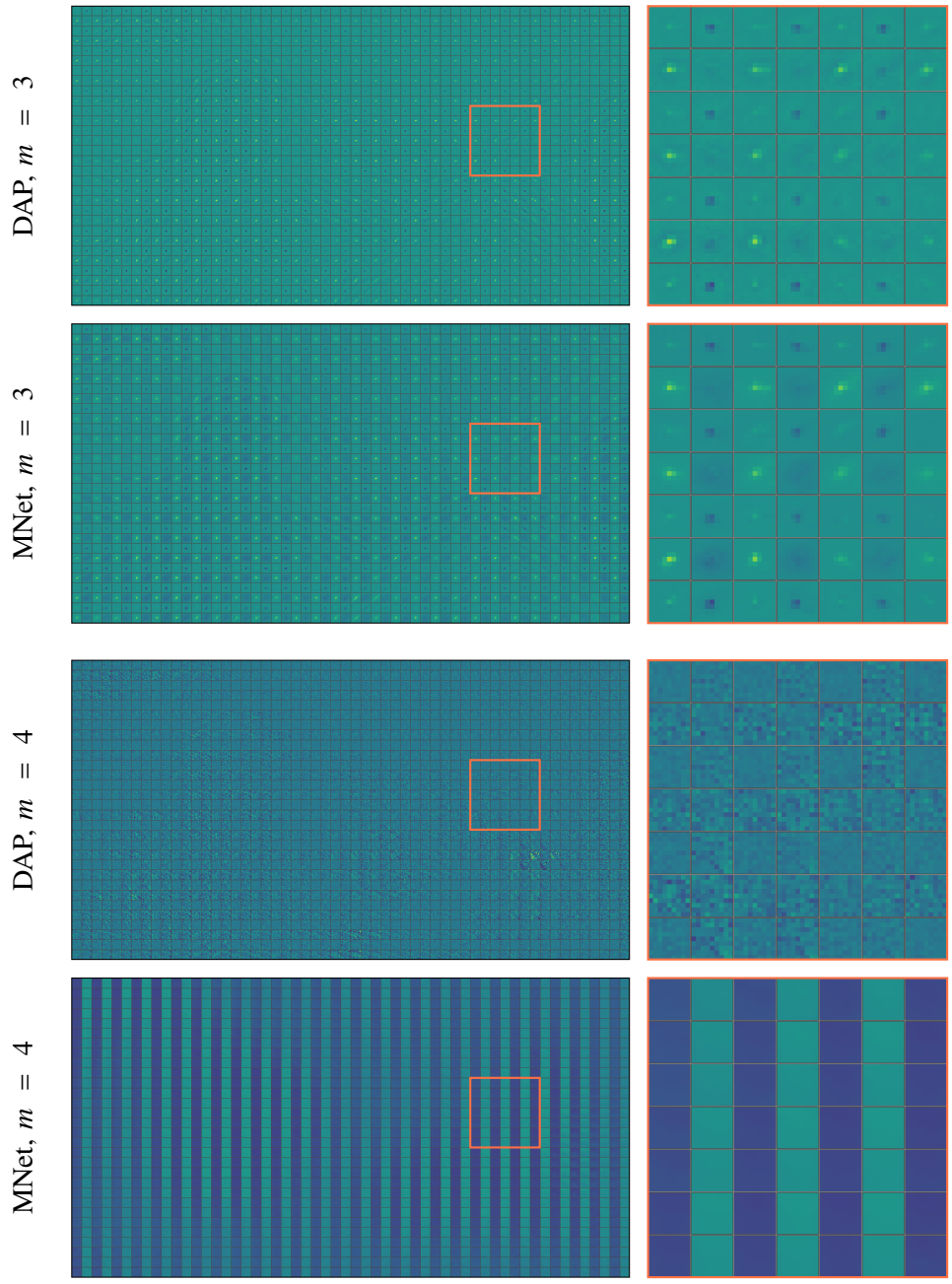

Figure B.1: Cost volume slices for the two-level RAFT+DICL method with CNN-based encoder. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP) of the respective level m, with marked excerpts to the right. All slices have been taken during the fourth (last) refinement iteration.

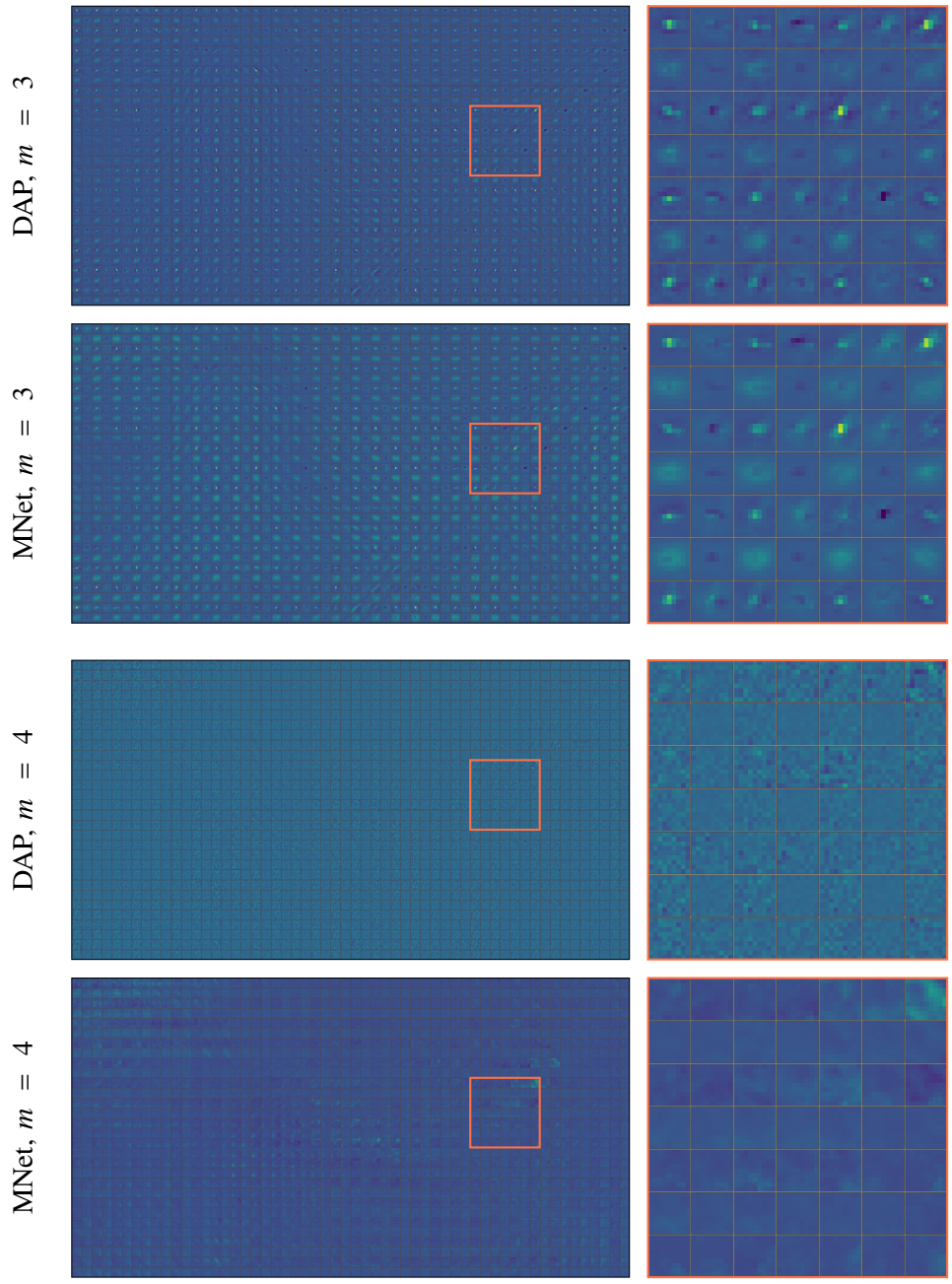

Figure B.2: Cost volume slices for the two-level RAFT+DICL method with pooling-based encoder and shared DICL module. Costs are visualized after the matching network (MNet) and after the displacement-aware projection (DAP) of the respective level m, with marked excerpts to the right. All slices have been taken during the fourth (last) refinement iteration.

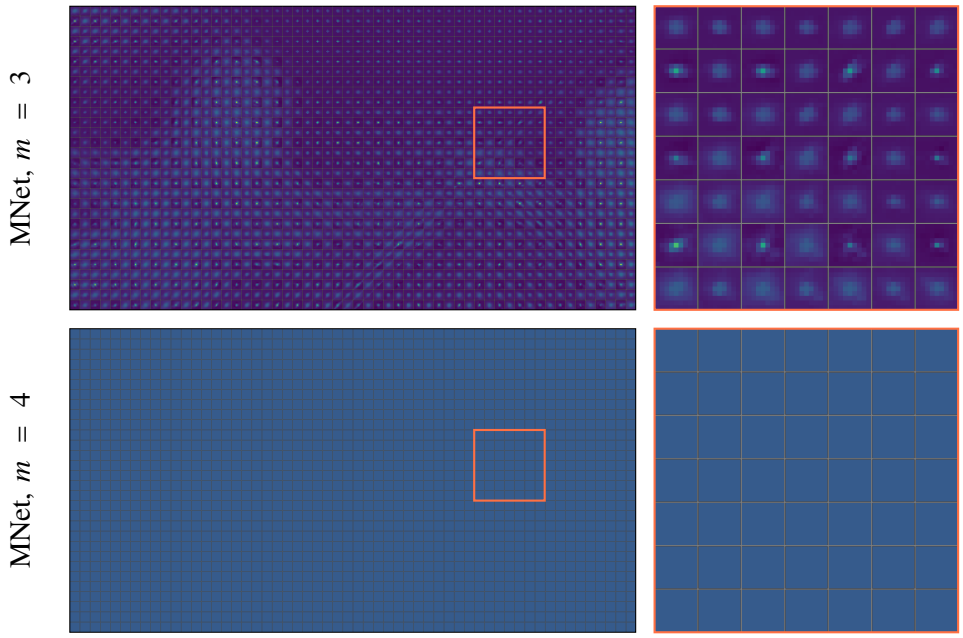

Figure B.3: Cost volume slices for the two-level RAFT+DICL method with pooling-based encoder, shared DICL module without DAP, and with direct cost regression applied separately to each level. Costs are visualized after the matching network (MNet) of the respective level m, with marked excerpts to the right. All slices have been taken during the fourth (last) refinement iteration.

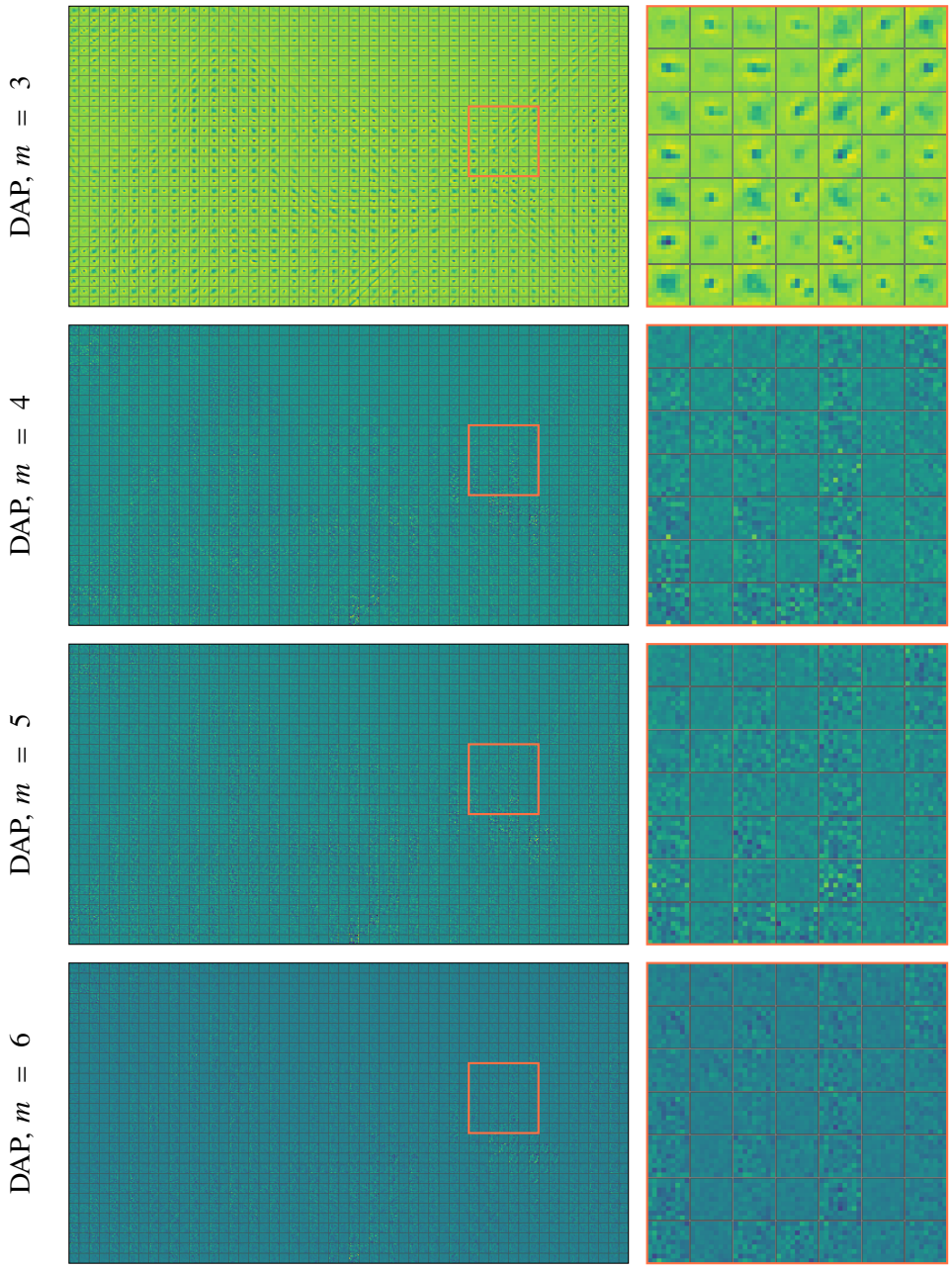

Figure B.4: DAP cost volume slices for the four-level RAFT+DICL method with CNN-based encoder. Costs are visualized after the displacement-aware projection (DAP) layer of the respective level m, with marked excerpts to the right. All slices have been taken during the fourth (last) refinement iteration.

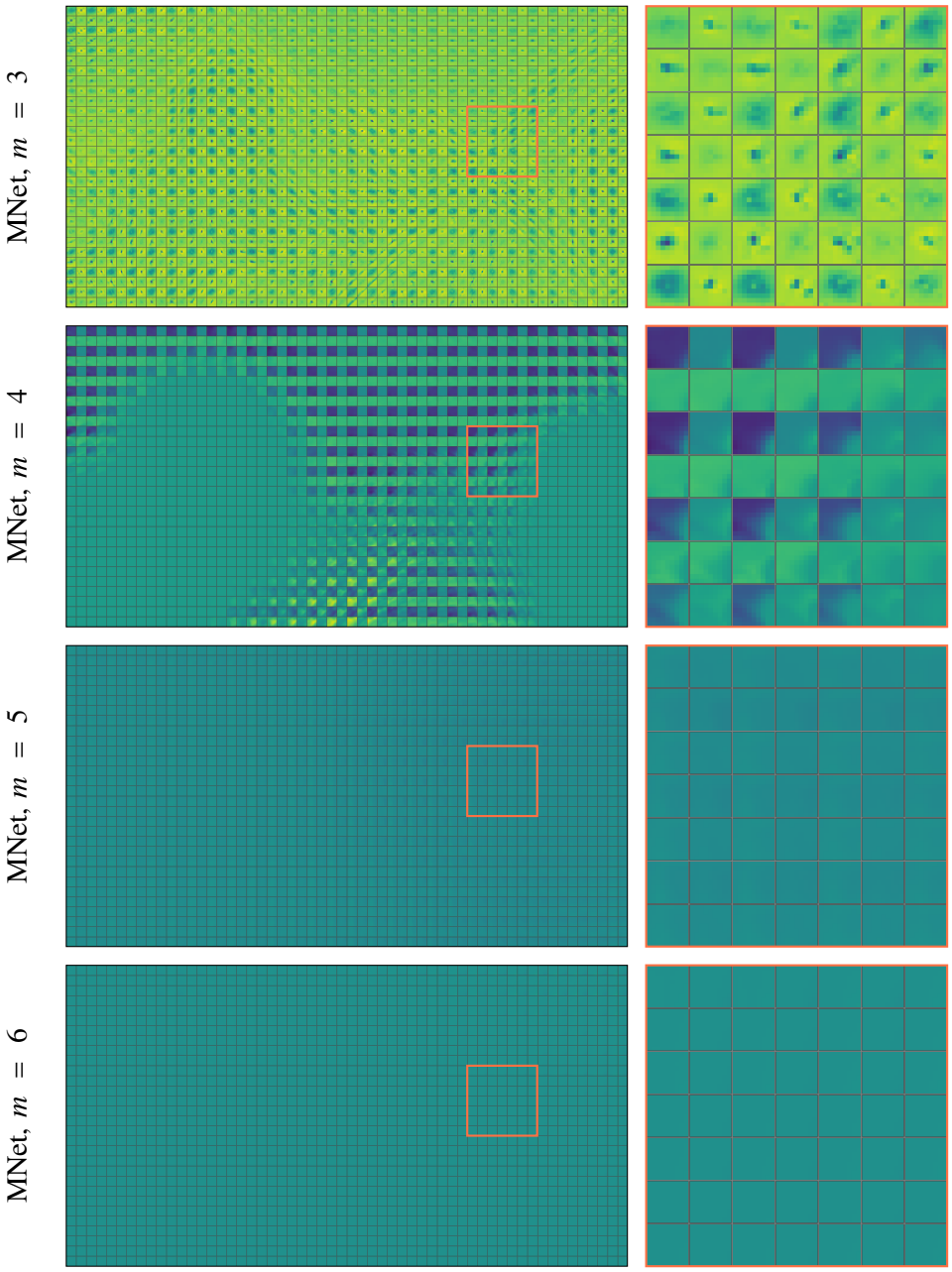

Figure B.5: MNet cost volume slices for the four-level RAFT+DICL method with CNN-based encoder. Costs are visualized after the matching network (MNet) of the respective level m, with marked excerpts to the right. All slices have been taken during the fourth (last) refinement iteration.

C Visual Flow Samples

Figures C.1 to C.6 show example flow estimates and their respective endpoint error maps for selected three-level coarse-to-fine RAFT+DICL methods, as well as the corresponding RAFT baseline model. All methods have been trained using the training strategy described in Section 4.5. As before, RAFT+DICL methods have been trained with an initial learning rate of 5×10^{-5}, RAFT models with an initial learning rate of 4×10^{-4}. Models are evaluated after training on the Sintel Mix stage. Note that all samples have been taken from the validation set. See Sections 6.2 and 6.3 for details on the naming scheme and abbreviations used. See Section 2.4 (in particular Figure 2.2) for more information on the endpoint error visualization being employed.

Note the differences in thin structures, such as the knife or spear. RAFT+DICL approaches can improve results in some instances, most notably the blurred spear tip in Figure C.6, however, can also have deficiencies, for example regarding the knife in Figures C.1 and C.2.

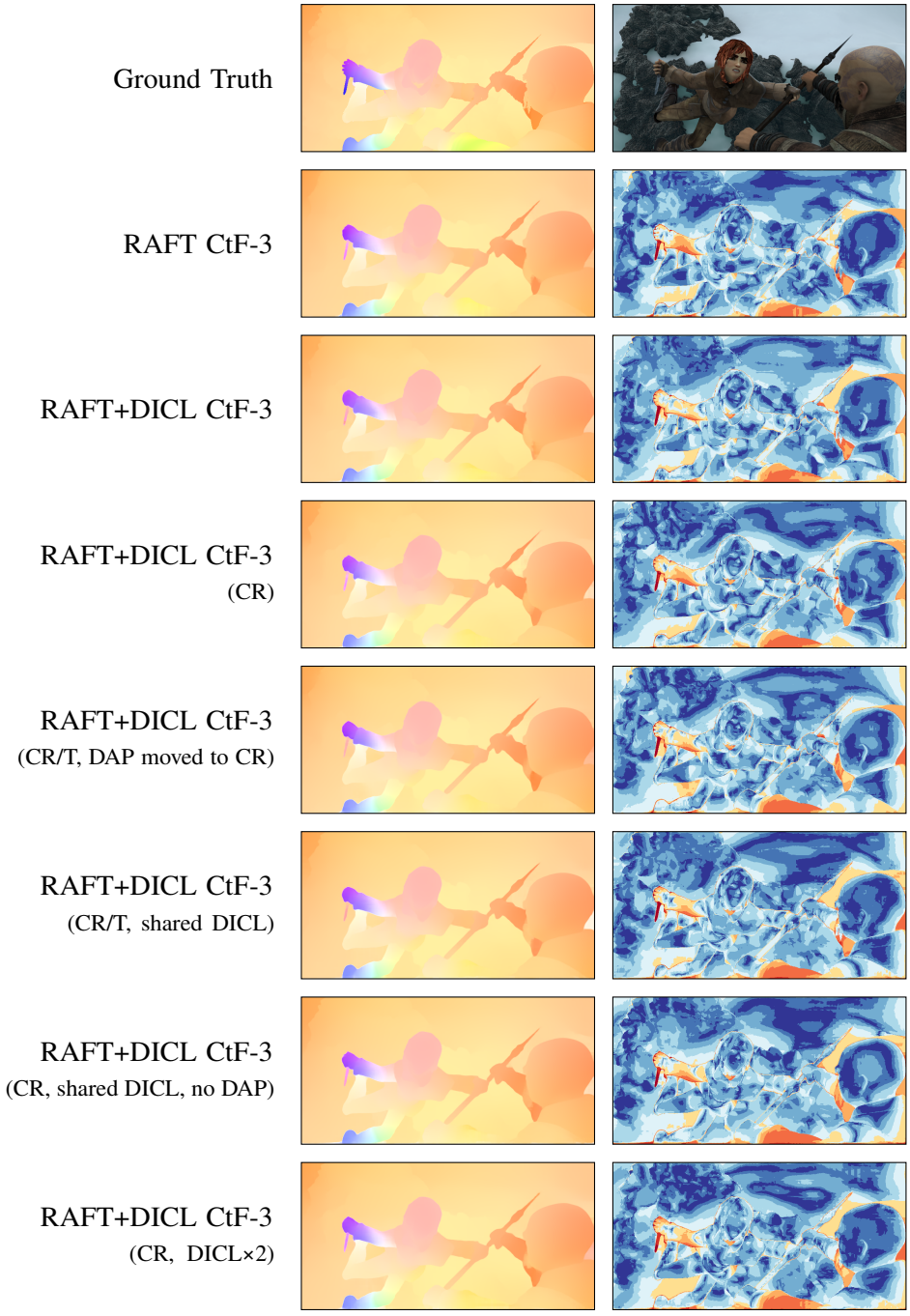

Figure C.1: Flow samples (left) and endpoint error visualization (right) for the "clean" version of frame 4 of the Sintel "ambush 6" scene. Top row: Ground-truth flow (left) and first input image (right). Results are shown for various three-level coarse-to-fine (CtF-3) RAFT+DICL methods, with variations in use of cost regression (CR) and use/placement of the DAP layer. Methods have been trained on and evaluated after the Sintel Mix stage.

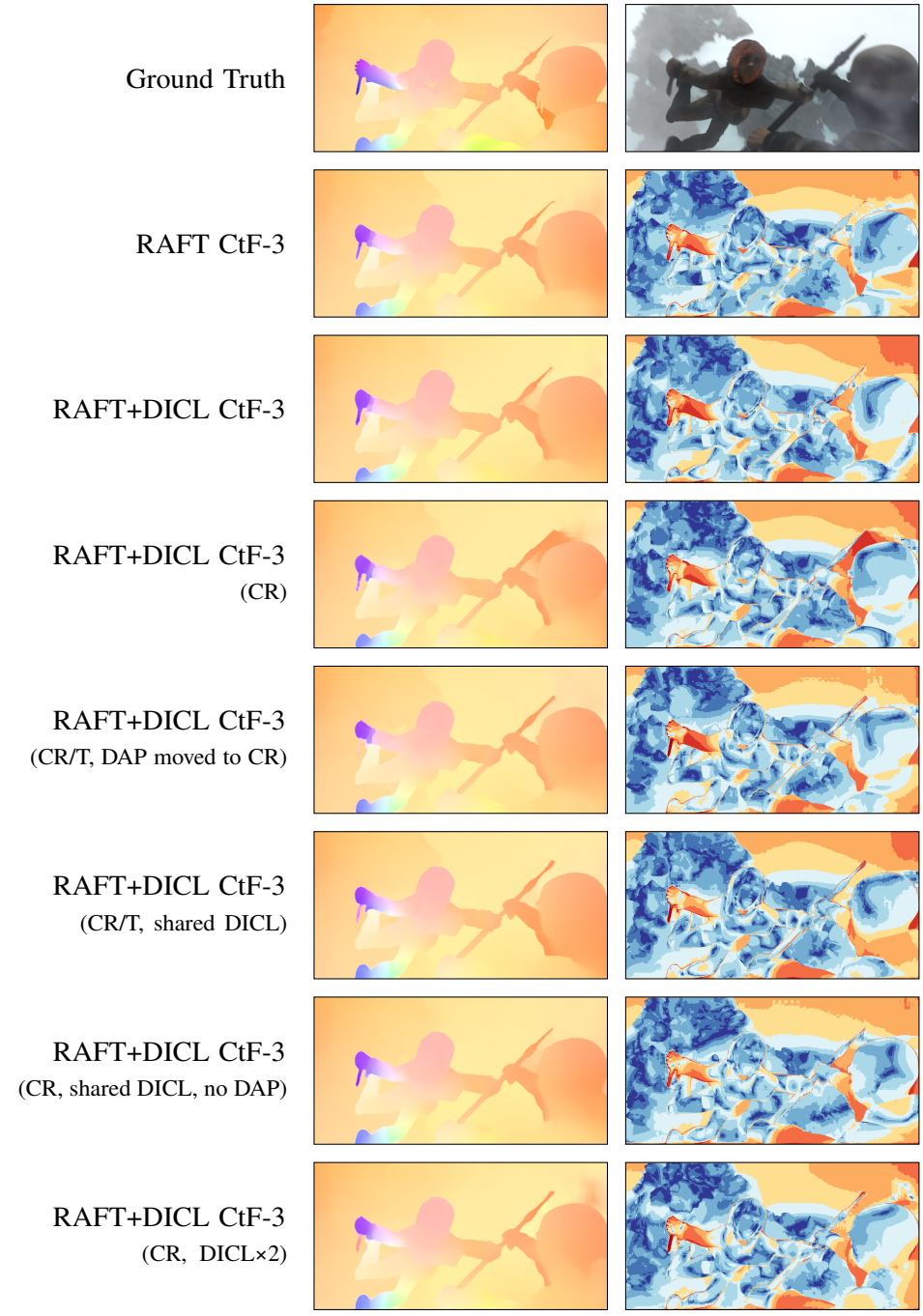

Figure C.2: Flow samples (left) and endpoint error visualization (right) for the "final" version of frame 4 of the Sintel "ambush 6" scene. Top row: Ground-truth flow (left) and first input image (right). Results are shown for various three-level coarse-to-fine (CtF-3) RAFT+DICL methods, with variations in use of cost regression (CR) and use/placement of the DAP layer. Methods have been trained on and evaluated after the Sintel Mix stage.

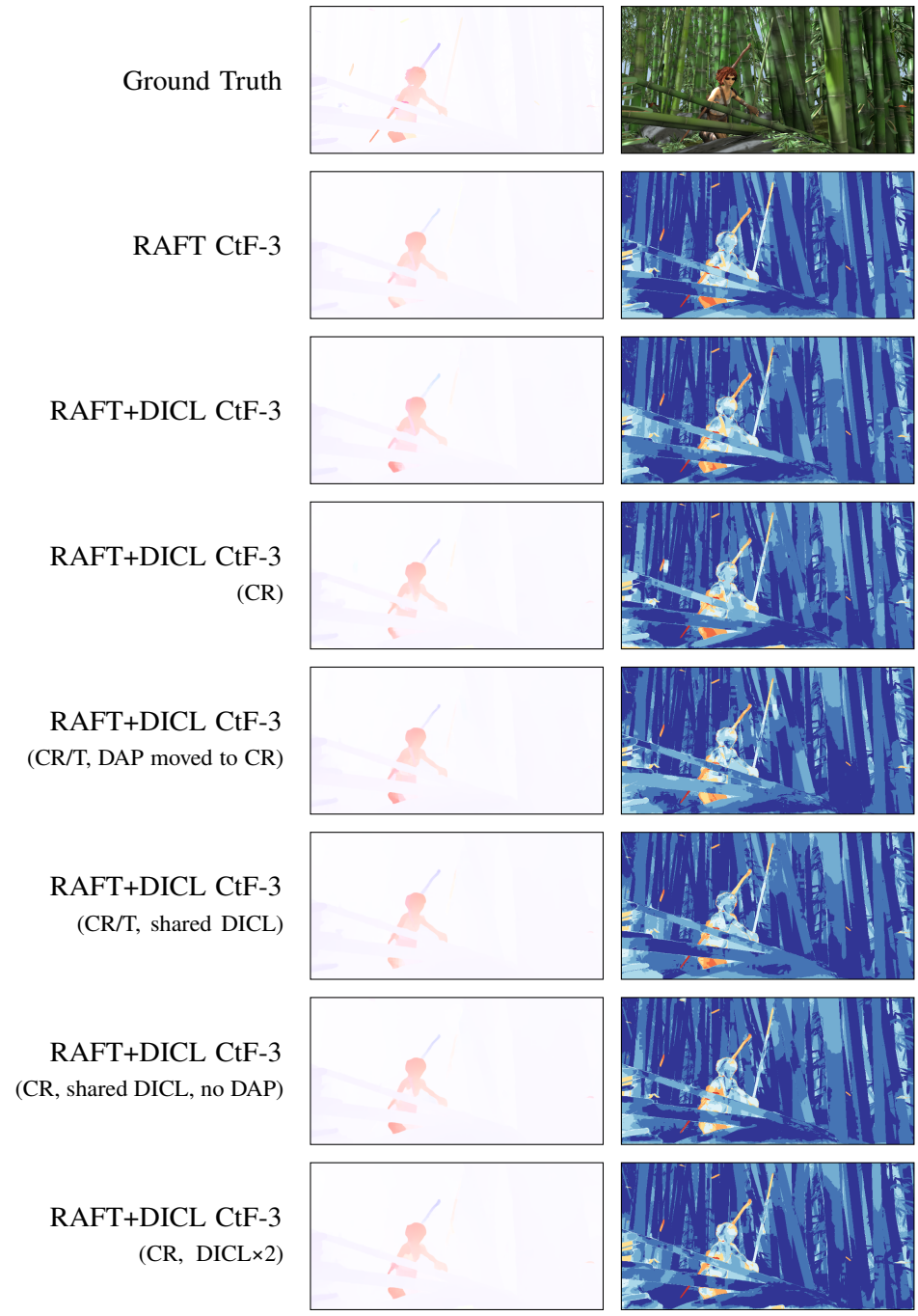

Figure C.3: Flow samples (left) and endpoint error visualization (right) for the "clean" version of frame 29 of the Sintel "bamboo 2" scene. Top row: Ground-truth flow (left) and first input image (right). Results are shown for various three-level coarse-to-fine (CtF-3) RAFT+DICL methods, with variations in use of cost regression (CR) and use/placement of the DAP layer. Methods have been trained on and evaluated after the Sintel Mix stage.

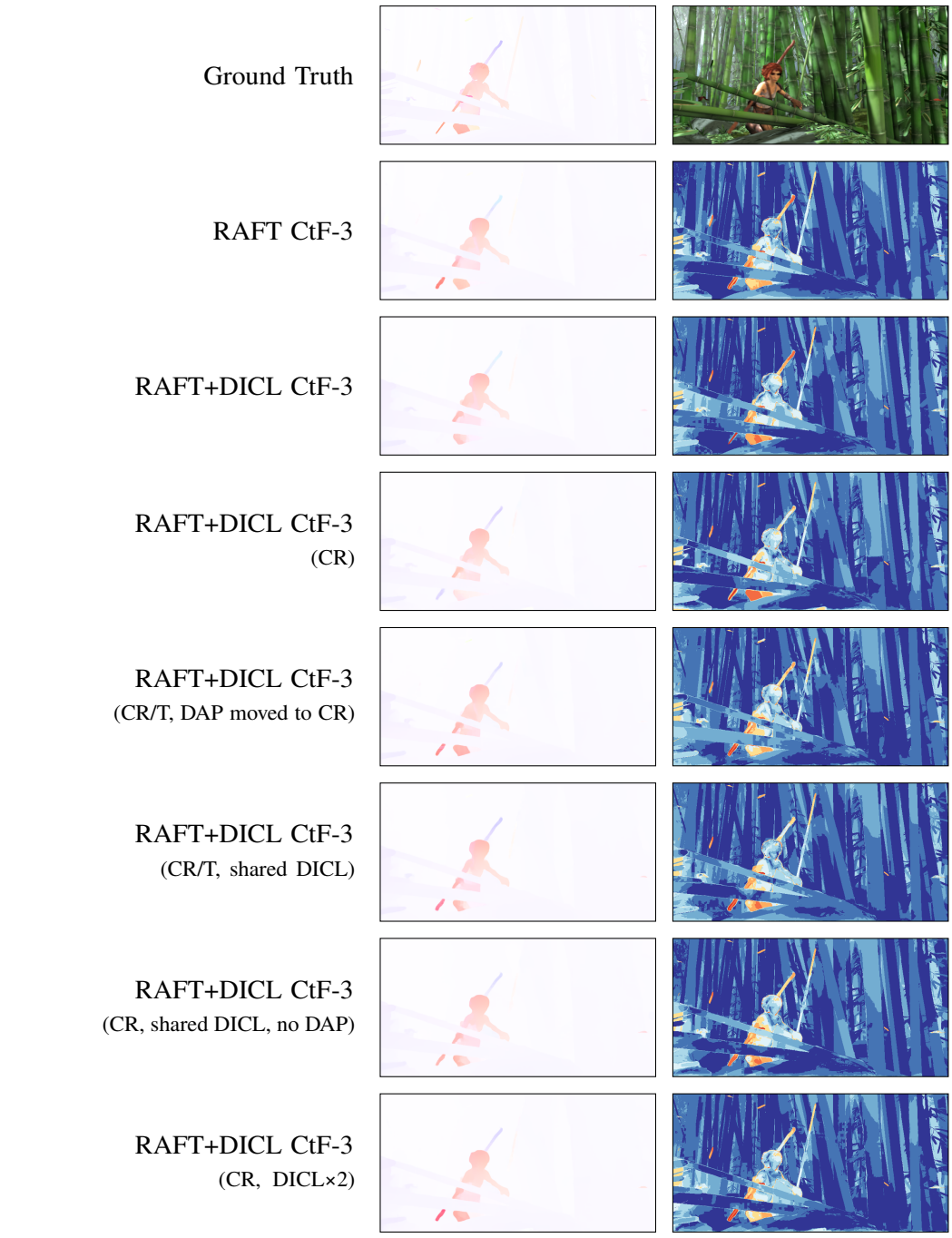

Figure C.4: Flow samples (left) and endpoint error visualization (right) for the "final" version of frame 29 of the Sintel "bamboo 2" scene. Top row: Ground-truth flow (left) and first input image (right). Results are shown for various three-level coarse-to-fine (CtF-3) RAFT+DICL methods, with variations in use of cost regression (CR) and use/placement of the DAP layer. Methods have been trained on and evaluated after the Sintel Mix stage.

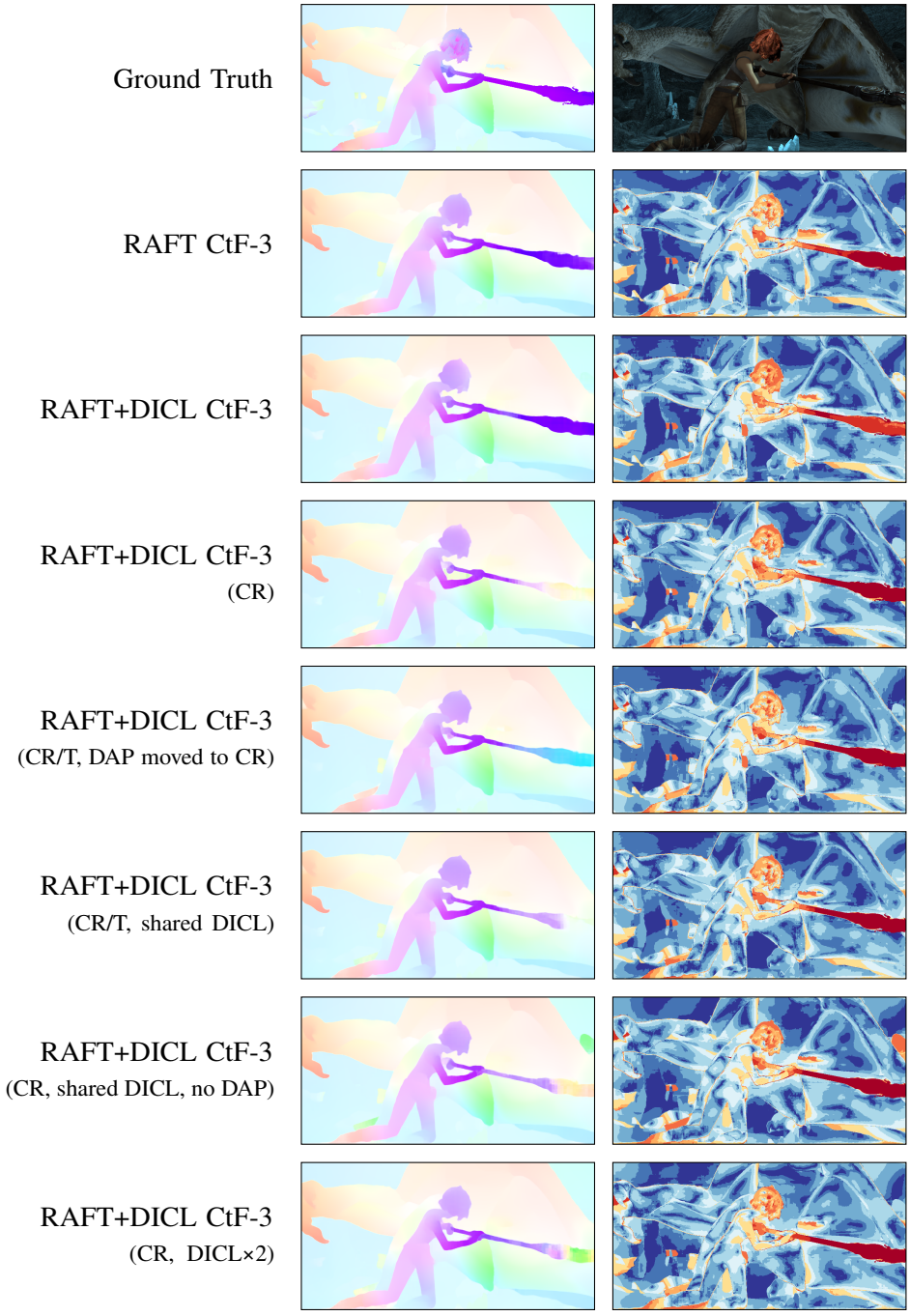

Figure C.5: Flow samples (left) and endpoint error visualization (right) for the "clean" version of frame 11 of the Sintel "cave 4" scene. Top row: Ground-truth flow (left) and first input image (right). Results are shown for various three-level coarse-to-fine (CtF-3) RAFT+DICL methods, with variations in use of cost regression (CR) and use/placement of the DAP layer. Methods have been trained on and evaluated after the Sintel Mix stage.

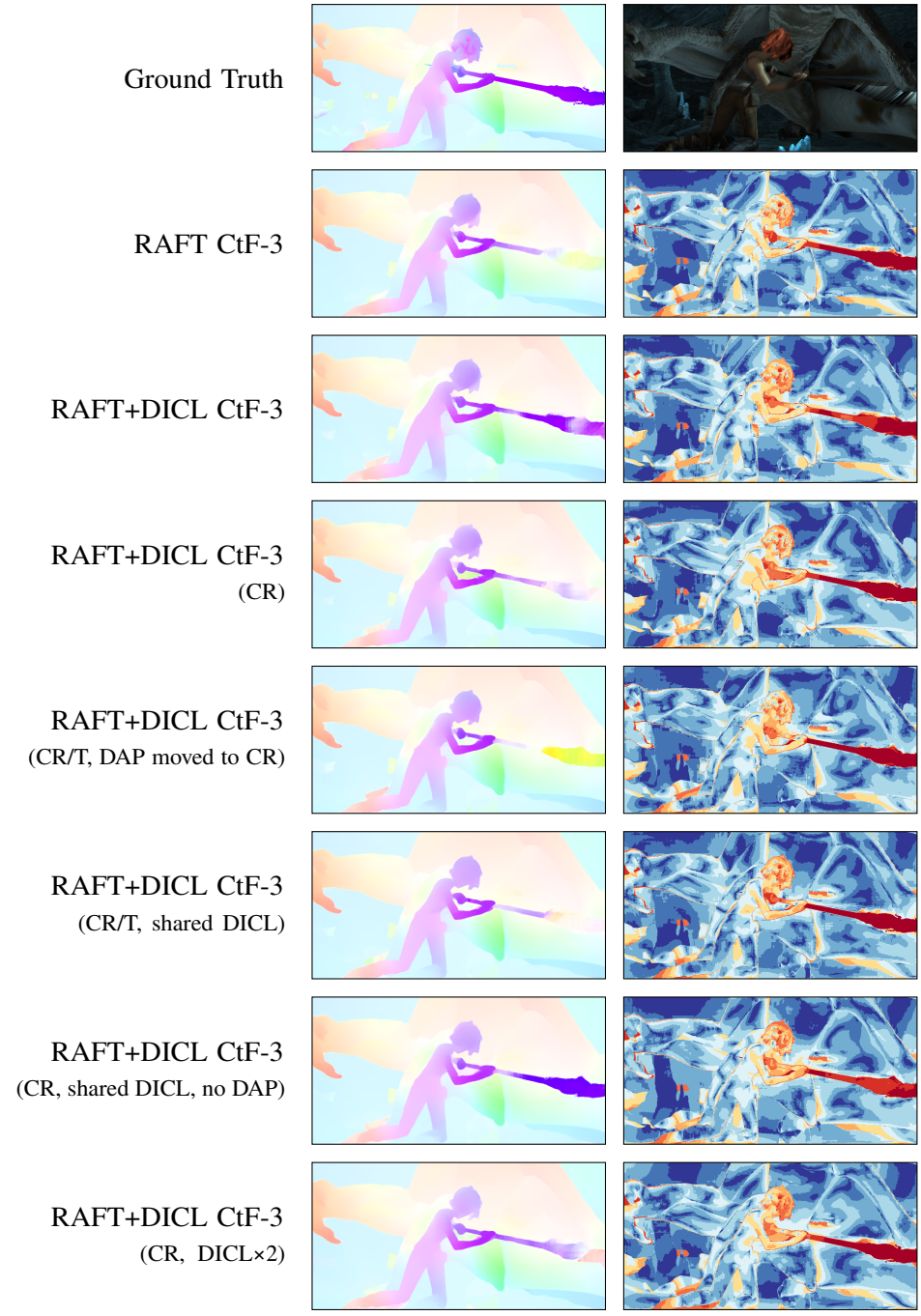

Figure C.6: Flow samples (left) and endpoint error visualization (right) for the "final" version of frame 11 of the Sintel "cave 4" scene. Top row: Ground-truth flow (left) and first input image (right). Results are shown for various three-level coarse-to-fine (CtF-3) RAFT+DICL methods, with variations in use of cost regression (CR) and use/placement of the DAP layer. Methods have been trained on and evaluated after the Sintel Mix stage.

Bibliography

[AJN+16] A. Ali, A. Jalil, J. Niu, X. Zhao, S. Rathore, J. Ahmed, M. Aksam Iftikhar. "Visual Object Tracking — Classical and Contemporary Approaches". In: *Frontiers of Computer Science* 10.1 (2016), pp. 167–188 (cit. on p. 21).

[Ana89] P. Anandan. "A Computational Framework and an Algorithm for the Measurement of Visual Motion". In: *International Journal of Computer Vision* 2.3 (1989), pp. 283–310 (cit. on p. 27).

[BA96] M. J. Black, P. Anandan. "The Robust Estimation of Multiple Motions: Parametric and Piecewise-Smooth Flow Fields". In: *Computer Vision and Image Understanding* 63.1 (1996), pp. 75–104 (cit. on p. 27).

[BBPW04] T. Brox, A. Bruhn, N. Papenberg, J. Weickert. "High Accuracy Optical Flow Estimation Based on a Theory for Warping". In: *Computer Vision - ECCV 2004*. LNCS 3024. Springer, 2004, pp. 25–36 (cit. on p. 27).

[BC08] D. Brezeale, D. Cook. "Automatic Video Classification: A Survey of the Literature". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.3 (2008), pp. 416–430 (cit. on p. 22).

[BFB94] J. L. Barron, D. J. Fleet, S. S. Beauchemin. "Performance of Optical Flow Techniques". In: *International Journal of Computer Vision* 12.1 (1994), pp. 43–77 (cit. on pp. 34, 40).

[BGG03] V. Bruce, P. R. Green, M. A. Georgeson. *Visual Perception: Physiology, Psychology, & Ecology*. 4th ed. Psychology Press, 2003. 483 pp. (cit. on p. 26).

[BGL+94] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, R. Shah. "Signature Verification Using a "Siamese" Time Delay Neural Network". In: *Advances in Neural Information Processing Systems*. Vol. 6. Morgan Kaufmann Publishers, 1994, pp. 737–744 (cit. on p. 28).

[BLCW09] Y. Bengio, J. Louradour, R. Collobert, J. Weston. "Curriculum Learning". In: *Proceedings of the International Conference on Machine Learning (ICML)*. Association for Computing Machinery, 2009, pp. 41–48 (cit. on pp. 34, 38).

[Bru06] A. Bruhn. "Variational Optic Flow Computation: Accurate Modelling and Efficient Numerics". PhD thesis. Department of Mathematics and Computer Science, Saarland University, 2006. 206 pp. (cit. on p. 41).

[BSL+11] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, R. Szeliski. "A Database and Evaluation Methodology for Optical Flow". In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31 (cit. on pp. 27, 34, 35, 41).

[BVS17] C. Bailer, K. Varanasi, D. Stricker. “CNN-Based Patch Matching for Optical Flow with Thresholded Hinge Embedding Loss”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 2710–2719 (cit. on pp. 28, 33).

[BW20] A. Bar-Haim, L. Wolf. “ScopeFlow: Dynamic Scene Scoping for Optical Flow”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 7995–8004 (cit. on pp. 38, 79).

[BWSB12] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black. “A Naturalistic Open Source Movie for Optical Flow Evaluation”. In: *Computer Vision – ECCV 2012*. LNCS 7577. Springer, 2012, pp. 611–625 (cit. on pp. 35, 36, 39, 77).

[CCZH17] W. Chen, X. Chen, J. Zhang, K. Huang. “Beyond Triplet Loss: A Deep Quadruplet Network for Person Re-identification”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1320–1329 (cit. on p. 33).

[CFG+15] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, F. Yu. *ShapeNet: An Information-Rich 3D Model Repository*. Stanford University / Princeton University / Toyota Technological Institute at Chicago, 2015. arXiv: 1512.03012 [cs] (cit. on pp. 36, 37).

[CGN13] H. Chao, Y. Gu, M. Napolitano. “A Survey of Optical Flow Techniques for UAV Navigation Applications”. In: *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2013, pp. 710–716 (cit. on p. 21).

[CHL05] S. Chopra, R. Hadsell, Y. LeCun. “Learning a Similarity Metric Discriminatively, with Application to Face Verification”. In: *2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. IEEE, 2005, pp. 539–546 (cit. on p. 33).

[CMH20] Y. Cabon, N. Murray, M. Humenberger. “Virtual KITTI 2”. 2020. arXiv: 2001.10773 [cs, eess] (cit. on p. 37).

[CMS12] D. Ciregan, U. Meier, J. Schmidhuber. “Multi-Column Deep Neural Networks for Image Classification”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3642–3649 (cit. on p. 27).

[CvMBB14] K. Cho, B. van Merriënboer, D. Bahdanau, Y. Bengio. “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics, 2014, pp. 103–111 (cit. on p. 49).

[CZM+19] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, Q. V. Le. “AutoAugment: Learning Augmentation Strategies From Data”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 113–123 (cit. on p. 38).

[CZSL20] E. D. Cubuk, B. Zoph, J. Shlens, Q. Le. “RandAugment: Practical Automated Data Augmentation with a Reduced Search Space”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 18613–18624 (cit. on p. 38).

[DBK+21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*. 2021 (cit. on pp. 28, 31).

[DFI+15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, T. Brox. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 2758–2766 (cit. on pp. 29, 36, 38, 51, 77).

[DGXZ19] J. Deng, J. Guo, N. Xue, S. Zafeiriou. "ArcFace: Additive Angular Margin Loss for Deep Face Recognition". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 4685–4694 (cit. on p. 33).

[FBK15] D. Fortun, P. Bouthemy, C. Kervrann. "Optical Flow Modeling and Computation: A Survey". In: *Computer Vision and Image Understanding*. Image Understanding for Real-world Distributed Video Networks 134 (2015), pp. 1–21 (cit. on p. 27).

[FJ90] D. J. Fleet, A. D. Jepson. "Computation of Component Image Velocity from Local Phase Information". In: *International Journal of Computer Vision* 5.1 (1990), pp. 77–104 (cit. on pp. 39, 40).

[Fuk80] K. Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36.4 (1980), pp. 193–202 (cit. on p. 27).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, 2016. 775 pp. (cit. on p. 28).

[GCT98] A. Giachetti, M. Campani, V. Torre. "The Use of Optical Flow for Road Navigation". In: *IEEE Transactions on Robotics and Automation* 14.1 (1998), pp. 34–48 (cit. on p. 21).

[Gib50] J. J. Gibson. *The Perception of the Visual World.* The Perception of the Visual World. Houghton Mifflin, 1950. 242 pp. (cit. on pp. 19, 26).

[Gib86] J. J. Gibson. *The Ecological Approach to Visual Perception*. Psychology Press, Taylor & Francis Group, 1986. 315 pp. (cit. on pp. 19, 26).

[GLSU13] A. Geiger, P. Lenz, C. Stiller, R. Urtasun. "Vision Meets Robotics: The KITTI Dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237 (cit. on pp. 34, 35).

[GLU12] A. Geiger, P. Lenz, R. Urtasun. "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3354–3361 (cit. on pp. 21, 35, 39, 41, 77).

[GW16] D. Gadot, L. Wolf. "PatchBatch: A Batch Augmented Loss for Optical Flow". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 4236–4245 (cit. on pp. 28, 33).

[GWCV16] A. Gaidon, Q. Wang, Y. Cabon, E. Vig. "VirtualWorlds as Proxy for Multi-object Tracking Analysis". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 4340–4349 (cit. on p. 37).

[Had02] J. Hadamard. “Sur Les Problèmes Aux Dérivées Partielles et Leur Signification Physique”. In: *Princeton University Bulletin* (1902), pp. 49–52 (cit. on p. 26).

[HBL17] A. Hermans, L. Beyer, B. Leibe. “In Defense of the Triplet Loss for Person Re-Identification”. 2017. arXiv: 1703.07737 [cs] (cit. on p. 33).

[HBP+20] M. Hofinger, S. R. Bulò, L. Porzi, A. Knapitsch, T. Pock, P. Kontschieder. “Improving Optical Flow on a Pyramid Level”. In: *Computer Vision – ECCV 2020*. LNCS 12373. Springer, 2020, pp. 770–786 (cit. on pp. 32, 50, 52, 63–66).

[HCL06] R. Hadsell, S. Chopra, Y. LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In: *2006 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. IEEE, 2006, pp. 1735–1742 (cit. on p. 34).

[HDK17] A. W. Harley, K. G. Derpanis, I. Kokkinos. “Segmentation-Aware Convolutional Networks Using Local Attention Masks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 5048–5057 (cit. on p. 29).

[HDP11] A. Hadid, J.-L. Dugelay, M. Pietikäinen. “On the Use of Dynamic Features in Face Biometrics: Recent Advances and Challenges”. In: *Signal, Image and Video Processing* 5.4 (2011), pp. 495–506 (cit. on p. 22).

[Hel25] H. Helmholtz. *The Perceptions of Vision*. Red. by J. Southall. Vol. III. Treatise on Physiological Optics. Optical Society of America, 1925. 736 pp. (cit. on p. 26).

[HL20] T.-W. Hui, C. C. Loy. “LiteFlowNet3: Resolving Correspondence Ambiguity for More Accurate Optical Flow Estimation”. In: *Computer Vision – ECCV 2020*. LNCS 12365. Springer, 2020, pp. 169–184 (cit. on pp. 22, 30–32).

[Hom16] Homer. *The Iliad*. Trans. by A. S. Kline. Poetry In Translation, 2016. 744 pp. (cit. on p. 19).

[HPB+16] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, R. Cipolla. “Understanding Real World Indoor Scenes with Synthetic Data”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 4077–4085 (cit. on p. 37).

[HR19] J. Hur, S. Roth. “Iterative Residual Refinement for Joint Optical Flow and Occlusion Estimation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 5747–5756 (cit. on pp. 32, 36, 105).

[HR20] J. Hur, S. Roth. “Optical Flow Estimation in the Deep Learning Age”. In: *Modelling Human Motion*. Springer, 2020, pp. 119–140 (cit. on p. 28).

[HS81] B. K. P. Horn, B. G. Schunck. “Determining Optical Flow”. In: *Artificial Intelligence* 17.1 (1981), pp. 185–203 (cit. on p. 26).

[HTL18] T.-W. Hui, X. Tang, C. C. Loy. “LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 8981–8989 (cit. on pp. 29, 30).

[HTL21] T.-W. Hui, X. Tang, C. C. Loy. “A Lightweight Optical Flow CNN — Revisiting Data Fidelity and Regularization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.8 (2021), pp. 2555–2569 (cit. on pp. 29, 30).

[HXL+11] W. Hu, N. Xie, L. Li, X. Zeng, S. Maybank. “A Survey on Visual Content-Based Video Indexing and Retrieval”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41.6 (2011), pp. 797–819 (cit. on p. 22).

[IÇG+18] E. Ilg, Ö. Çiçek, S. Galesso, A. Klein, O. Makansi, F. Hutter, T. Brox. “Uncertainty Estimates and Multi-hypotheses Networks for Optical Flow”. In: *Computer Vision – ECCV 2018*. LNCS 11211. Springer, 2018, pp. 677–693 (cit. on p. 29).

[IMS+17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox. “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1647–1655 (cit. on pp. 29, 34, 36, 38).

[IS15] S. Ioffe, C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 448–456 (cit. on pp. 78, 79, 83).

[ISKB18] E. Ilg, T. Saikia, M. Keuper, T. Brox. “Occlusions, Motion and Depth Boundaries with a Generic Network for Disparity, Optical Flow or Scene Flow Estimation”. In: *Computer Vision – ECCV 2018*. LNCS 11216. Springer, 2018, pp. 626–643 (cit. on pp. 29, 36, 77).

[JBA+22] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, O. J. Henaff, M. Botvinick, A. Zisserman, O. Vinyals, J. Carreira. “Perceiver IO: A General Architecture for Structured Inputs & Outputs”. In: *International Conference on Learning Representations (ICLR)*. 2022 (cit. on pp. 31, 51, 77, 109).

[JCL+21] S. Jiang, D. Campbell, Y. Lu, H. Li, R. Hartley. “Learning To Estimate Hidden Motions With Global Motion Aggregation”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2021, pp. 9752–9761 (cit. on pp. 31, 45).

[JDO+17] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, K. Kavukcuoglu. “Population Based Training of Neural Networks”. 2017. arXiv: 1711.09846 [cs] (cit. on p. 37).

[JGB+21] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, J. Carreira. “Perceiver: General Perception with Iterative Attention”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 4651–4664 (cit. on p. 31).

[JL21] H. Jiang, E. Learned-Miller. “DCVNet: Dilated Cost Volume Networks for Fast Optical Flow”. 2021. arXiv: 2103.17271 [cs] (cit. on pp. 30, 32).

[JST+19] W. Jiang, W. Sun, A. Tagliasacchi, E. Trulls, K. M. Yi. “Linearized Multi-Sampling for Differentiable Image Transformation”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2019, pp. 2988–2997 (cit. on p. 50).

[JSZK15] M. Jaderberg, K. Simonyan, A. Zisserman, K. Kavukcuoglu. “Spatial Transformer Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015, pp. 2017–2025 (cit. on p. 32).

[KB19] M. Kaya, H. Ş. Bilge. “Deep Metric Learning: A Survey”. In: *Symmetry* 11.9 (9 2019), p. 1066 (cit. on p. 33).

[KMK15] I. Kajo, A. S. Malik, N. Kamel. “Motion Estimation of Crowd Flow Using Optical Flow Techniques: A Review”. In: *2015 9th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE, 2015, pp. 1–9 (cit. on p. 21).

[KNH+16] D. Kondermann, R. Nair, K. Honauer, K. Krispin, J. Andrulis, A. Brock, B. Güssefeld, M. Rahimimoghaddam, S. Hofmann, C. Brenner, B. Jähne. “The HCI Benchmark Suite: Stereo and Flow Ground Truth with Uncertainties for Urban Autonomous Driving”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2016, pp. 19–28 (cit. on p. 35).

[KNH+21] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, M. Shah. “Transformers in Vision: A Survey”. In: *ACM Computing Surveys* (2021) (cit. on p. 28).

[KNM+15] D. Kondermann, R. Nair, S. Meister, W. Mischler, B. Güssefeld, K. Honauer, S. Hofmann, C. Brenner, B. Jähne. “Stereo Ground Truth with Error Bars”. In: *Computer Vision – ACCV 2014*. LNCS 9007. Springer, 2015, pp. 595–610 (cit. on p. 35).

[KSH12] A. Krizhevsky, I. Sutskever, G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates Inc., 2012, pp. 1097–1105 (cit. on pp. 27, 38).

[Kul13] B. Kulis. “Metric Learning: A Survey”. In: *Foundations and Trends in Machine Learning* 5.4 (2013), pp. 287–364 (cit. on p. 33).

[KV08] D. Kalligeropoulos, S. Vasileiadou. “The Homeric Automata and Their Implementation”. In: *Science and Technology in Homeric Epics*. History of Mechanism and Machine Science. Springer Netherlands, 2008, pp. 77–84 (cit. on p. 19).

[Lai10] A. F. Laine. “In the Spotlight: Biomedical Imaging”. In: *IEEE Reviews in Biomedical Engineering* 3 (2010), pp. 7–9 (cit. on p. 22).

[LHW+18] W.-S. Lai, J.-B. Huang, O. Wang, E. Shechtman, E. Yumer, M.-H. Yang. “Learning Blind Video Temporal Consistency”. In: *Computer Vision – ECCV 2018*. LNCS 11219. Springer, 2018, pp. 179–195 (cit. on p. 22).

[LHZ17] J. Lu, J. Hu, J. Zhou. “Deep Metric Learning for Visual Understanding: An Overview of Recent Advances”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 76–84 (cit. on p. 33).

[LK81] B. D. Lucas, T. Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision”. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence*. Vol. 2. IJCAI’81. Morgan Kaufmann Publishers, 1981, pp. 674–679 (cit. on p. 26).

[LL22] L. Long, J. Lang. “Detail Preserving Residual Feature Pyramid Modules for Optical Flow”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. IEEE, 2022, pp. 2100–2108 (cit. on p. 139).

[LLKX19] P. Liu, M. Lyu, I. King, J. Xu. “SelFlow: Self-Supervised Learning of Optical Flow”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 4566–4575 (cit. on p. 51).

[Lou18] C. Loukas. “Video Content Analysis of Surgical Procedures”. In: *Surgical Endoscopy* 32.2 (2018), pp. 553–568 (cit. on p. 22).

[LVW+20] Y. Lu, J. Valmadre, H. Wang, J. Kannala, M. Harandi, P. H. S. Torr. “Devon: Deformable Volume Network for Learning Optical Flow”. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2020, pp. 2694–2702 (cit. on p. 32).

[LWY+17] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, L. Song. “SphereFace: Deep Hypersphere Embedding for Face Recognition”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 6738–6746 (cit. on p. 33).

[LXM+21] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, T.-K. Kim. “Multiple Object Tracking: A Literature Review”. In: *Artificial Intelligence* 293 (2021), p. 103448 (cit. on p. 21).

[MBP10] O. Mac Aodha, G. J. Brostow, M. Pollefeys. “Segmenting Video into Classes of Algorithm-Suitability”. In: *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, pp. 1054–1061 (cit. on p. 35).

[MG15] M. Menze, A. Geiger. “Object Scene Flow for Autonomous Vehicles”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 3061–3070 (cit. on pp. 21, 35, 40, 42).

[MHG15] M. Menze, C. Heipke, A. Geiger. “Joint 3D Estimation of Vehicles and Scene Flow”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* II-3/W5 (2015), pp. 427–434 (cit. on pp. 35, 77).

[MHG18] M. Menze, C. Heipke, A. Geiger. “Object Scene Flow”. In: *ISPRS Journal of Photogrammetry and Remote Sensing*. Geospatial Computer Vision 140 (2018), pp. 60–76 (cit. on pp. 35, 39).

[MHLD17] J. McCormac, A. Handa, S. Leutenegger, A. J. Davison. “SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?” In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2697–2706 (cit. on p. 37).

[MHPB13] O. Mac Aodha, A. Humayun, M. Pollefeys, G. J. Brostow. “Learning a Confidence Measure for Optical Flow”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.5 (2013), pp. 1107–1120 (cit. on p. 35).

[MIF+18] N. Mayer, E. Ilg, P. Fischer, C. Hazirbas, D. Cremers, A. Dosovitskiy, T. Brox. “What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation?” In: *International Journal of Computer Vision* 126.9 (2018), pp. 942–960 (cit. on p. 38).

[MIH+16] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, T. Brox. “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 4040–4048 (cit. on pp. 36, 37, 77).

[Miu05] K. Miura. “Tracking Movement in Cell Biology”. In: *Microscopy Techniques*. Vol. 95. Advances in Biochemical Engineering. Springer, 2005, pp. 267–295 (cit. on p. 22).

[MMBH18] Z. Mahfouf, H. F. Merouani, I. Bouchrika, N. Harrati. “Investigating the Use of Motion-Based Features from Optical Flow for Gait Recognition”. In: *Neurocomputing* 283 (2018), pp. 140–149 (cit. on p. 22).

[MP98] E. Memin, P. Perez. "A Multigrid Approach for Hierarchical Motion Estimation". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. IEEE, 1998, pp. 933–938 (cit. on p. 27).

[NŠM19] M. Neoral, J. Šochman, J. Matas. "Continual Occlusion and Optical Flow Estimation". In: *Computer Vision – ACCV 2018*. LNCS 11364. Springer, 2019, pp. 159–174 (cit. on p. 30).

[QWL+20] S. Qiao, H. Wang, C. Liu, W. Shen, A. Yuille. "Micro-Batch Training with Batch-Channel Normalization and Weight Standardization". 2020. arXiv: 1903.10520 [cs] (cit. on pp. 78, 79, 92, 104, 139).

[RB17] A. Ranjan, M. J. Black. "Optical Flow Estimation Using a Spatial Pyramid Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 2720–2729 (cit. on p. 29).

[RHK17] S. R. Richter, Z. Hayder, V. Koltun. "Playing for Benchmarks". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2232–2241 (cit. on pp. 34–36).

[RHT+20] A. Ranjan, D. T. Hoffmann, D. Tzionas, S. Tang, J. Romero, M. J. Black. "Learning Multi-human Optical Flow". In: *International Journal of Computer Vision* 128.4 (2020), pp. 873–890 (cit. on p. 37).

[RHW86] D. E. Rumelhart, G. E. Hinton, R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. Computational Models of Cognition and Perception. MIT Press, 1986, pp. 318–362 (cit. on p. 28).

[RWHS15] J. Revaud, P. Weinzaepfel, Z. Harchaoui, C. Schmid. "EpicFlow: Edge-preserving Interpolation of Correspondences for Optical Flow". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 1164–1172 (cit. on pp. 28, 29).

[Sch15] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61 (2015), pp. 85–117 (cit. on p. 28).

[SET20] S. Savian, M. Elahi, T. Tillo. "Optical Flow Estimation with Deep Learning, a Survey on Recent Advances". In: *Deep Biometrics*. Unsupervised and Semi-Supervised Learning. Springer, 2020, pp. 257–287 (cit. on p. 28).

[SGH21] J. L. Suárez, S. García, F. Herrera. "A Tutorial on Distance Metric Learning: Mathematical Foundations, Algorithms, Experimental Analysis, Prospects and Challenges". In: *Neurocomputing* 425 (2021), pp. 300–322 (cit. on p. 33).

[SJAS21] J. P. Singh, S. Jain, S. Arora, U. P. Singh. "A Survey of Behavioral Biometric Gait Recognition: Current Success and Future Perspectives". In: *Archives of Computational Methods in Engineering* 28.1 (2021), pp. 107–148 (cit. on p. 22).

[SKP15] F. Schroff, D. Kalenichenko, J. Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 815–823 (cit. on p. 33).

[SLG+19] L. Sevilla-Lara, Y. Liao, F. Güney, V. Jampani, A. Geiger, M. J. Black. "On the Integration of Optical Flow and Action Recognition". In: *Pattern Recognition*. LNCS 11269. Springer, 2019, pp. 281–297 (cit. on p. 21).

[SM16] K. Sitara, B. M. Mehtre. “Digital Video Tampering Detection: An Overview of Passive Techniques”. In: *Digital Investigation* 18 (2016), pp. 8–22 (cit. on p. 22).

[SRB10] D. Sun, S. Roth, M. J. Black. “Secrets of Optical Flow Estimation and Their Principles”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 2432–2439 (cit. on p. 27).

[SRB14] D. Sun, S. Roth, M. J. Black. “A Quantitative Analysis of Current Practices in Optical Flow Estimation and the Principles Behind Them”. In: *International Journal of Computer Vision* 106.2 (2014), pp. 115–137 (cit. on p. 27).

[SS02] D. Scharstein, R. Szeliski. “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms”. In: *International Journal of Computer Vision* 47.1 (2002), pp. 7–42 (cit. on p. 40).

[SSM+16] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. S. Kruthiventi, R. V. Babu. “A Taxonomy of Deep Convolutional Neural Nets for Computer Vision”. In: *Frontiers in Robotics and AI* 2 (2016) (cit. on p. 28).

[SSM15] J. Sánchez, A. Salgado, N. Monzón. “Computing Inverse Optical Flow”. In: *Pattern Recognition Letters* 52 (2015), pp. 32–39 (cit. on p. 77).

[SVH+21] D. Sun, D. Vlasic, C. Herrmann, V. Jampani, M. Krainin, H. Chang, R. Zabih, W. T. Freeman, C. Liu. “AutoFlow: Learning a Better Training Set for Optical Flow”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2021, pp. 10088–10097 (cit. on pp. 37, 38).

[SW18] K. Sundararajan, D. L. Woodard. “Deep Learning for Biometrics: A Survey”. In: *ACM Computing Surveys* 51.3 (2018), 65:1–65:34 (cit. on p. 21).

[SXJS16] H. O. Song, Y. Xiang, S. Jegelka, S. Savarese. “Deep Metric Learning via Lifted Structured Feature Embedding”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 4004–4012 (cit. on p. 33).

[SYLK18] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, pp. 8934–8943 (cit. on pp. 29, 33, 37, 38, 51, 77, 79).

[SYLK20] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. “Models Matter, So Does Training: An Empirical Study of CNNs for Optical Flow Estimation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.6 (2020), pp. 1408–1423 (cit. on pp. 28, 29, 34, 36–38, 77–79).

[SZ14] K. Simonyan, A. Zisserman. “Two-Stream Convolutional Networks for Action Recognition in Videos”. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014, pp. 568–576 (cit. on p. 21).

[SZS12] K. Soomro, A. R. Zamir, M. Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. CRCV-TR-12-01. Center for Research in Computer Vision, University of Central Florida, 2012. 7 pp. arXiv: 1212.0402 [cs] (cit. on p. 36).

[TBS12] P. Torkashvand, H. Behnam, Z. A. Sani. “Modified Optical Flow Technique for Cardiac Motions Analysis in Echocardiography Images”. In: *Journal of Medical Signals and Sensors* 2.3 (2012), pp. 121–127 (cit. on p. 22).

[TD20] Z. Teed, J. Deng. “RAFT: Recurrent All-Pairs Field Transforms for Optical Flow”. In: *Computer Vision – ECCV 2020*. LNCS 12347. Springer, 2020, pp. 402–419 (cit. on pp. 19, 22, 23, 31, 33, 34, 36–38, 45, 47, 49, 50, 58, 59, 63, 65, 69, 71, 73, 75–80, 88, 89, 103, 105, 116, 118, 126, 137, 144).

[TDGT20] P. Truong, M. Danelljan, L. V. Gool, R. Timofte. “GOCor: Bringing Globally Optimized Correspondence Volumes into Your Neural Network”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 14278–14290 (cit. on p. 31).

[TH17] D. Teney, M. Hebert. “Learning to Extract Motion from Videos in Convolutional Neural Networks”. In: *Computer Vision – ACCV 2016*. LNCS 10115. Springer, 2017, pp. 412–428 (cit. on p. 31).

[TYB16] Y.-H. Tsai, M.-H. Yang, M. J. Black. “Video Segmentation via Object Flow”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 3899–3908 (cit. on p. 21).

[VHW16] D. K. Vig, A. E. Hamby, C. W. Wolgemuth. “On the Quantification of Cellular Velocity Fields”. In: *Biophysical Journal* 110.7 (2016), pp. 1469–1475 (cit. on p. 22).

[WBSB12] J. Wulff, D. J. Butler, G. B. Stanley, M. J. Black. “Lessons and Insights from Creating a Synthetic Optical Flow Benchmark”. In: *Computer Vision – ECCV 2012. Workshops and Demonstrations*. LNCS 7584. Springer, 2012, pp. 168–177 (cit. on pp. 34, 35).

[WH18] Y. Wu, K. He. “Group Normalization”. In: *Computer Vision – ECCV 2018*. LNCS 11217. Springer, 2018, pp. 3–19 (cit. on pp. 79, 92).

[WRHS13] P. Weinzaepfel, J. Revaud, Z. Harchaoui, C. Schmid. “DeepFlow: Large Displacement Optical Flow with Deep Matching”. In: *2013 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2013, pp. 1385–1392 (cit. on p. 28).

[WS13] H. Wang, C. Schmid. “Action Recognition with Improved Trajectories”. In: *2013 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2013, pp. 3551–3558 (cit. on p. 21).

[WWZ+18] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, W. Liu. “CosFace: Large Margin Cosine Loss for Deep Face Recognition”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018, pp. 5265–5274 (cit. on p. 33).

[WZD+20] J. Wang, Y. Zhong, Y. Dai, K. Zhang, P. Ji, H. Li. “Displacement-Invariant Matching Cost Learning for Accurate Optical Flow Estimation”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 15220–15231 (cit. on pp. 19, 22, 23, 30, 34, 36, 38, 51–55, 57, 71, 73, 75, 77, 78, 80, 92, 113, 114, 116, 119, 124, 133–135, 137, 139).

[WZLQ16] Y. Wen, K. Zhang, Z. Li, Y. Qiao. “A Discriminative Feature Learning Approach for Deep Face Recognition”. In: *Computer Vision – ECCV 2016*. LNCS 9911. Springer, 2016, pp. 499–515 (cit. on p. 33).

[XRK17] J. Xu, R. Ranftl, V. Koltun. “Accurate Optical Flow via Direct Cost Volume Processing”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 5807–5815 (cit. on p. 28).

[XWL+22] Y. Xu, H. Wei, M. Lin, Y. Deng, K. Sheng, M. Zhang, F. Tang, W. Dong, F. Huang, C. Xu. “Transformers in Computational Visual Media: A Survey”. In: *Computational Visual Media* 8.1 (2022), pp. 33–62 (cit. on p. 28).

[XYS+20] T. Xiao, J. Yuan, D. Sun, Q. Wang, X.-Y. Zhang, K. Xu, M.-H. Yang. “Learnable Cost Volume Using the Cayley Representation”. In: *Computer Vision – ECCV 2020*. LNCS 12354. Springer, 2020, pp. 483–499 (cit. on pp. 22, 31).

[Yan06] L. Yang. “Distance Metric Learning: A Comprehensive Survey”. Department of Computer Science and Engineering, Michigan State University, 2006. 51 pp. (cit. on p. 33).

[YBH15] K. Yousif, A. Bab-Hadiashar, R. Hoseinnezhad. “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics”. In: *Intelligent Industrial Systems* 1.4 (2015), pp. 289–311 (cit. on p. 21).

[YDY19] Z. Yin, T. Darrell, F. Yu. “Hierarchical Discrete Distribution Decomposition for Match Density Estimation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 6037–6046 (cit. on pp. 30, 45).

[YLZ19] G. Yao, T. Lei, J. Zhong. “A Review of Convolutional-Neural-Network-based Action Recognition”. In: *Pattern Recognition Letters*. Cooperative and Social Robots: Understanding Human Activities and Intentions 118 (2019), pp. 14–22 (cit. on p. 21).

[YR19] G. Yang, D. Ramanan. “Volumetric Correspondence Networks for Optical Flow”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019, pp. 794–805 (cit. on pp. 22, 30, 33, 51).

[ŽL15] J. Žbontar, Y. LeCun. “Computing the Stereo Matching Cost with a Convolutional Neural Network”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015, pp. 1592–1599 (cit. on pp. 28, 31).

[ŽL16] J. Žbontar, Y. LeCun. “Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2287–2318 (cit. on pp. 28, 31, 33, 34).

[ZPYT19] F. Zhang, V. Prisacariu, R. Yang, P. H. Torr. “GA-Net: Guided Aggregation Net for End-To-End Stereo Matching”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 185–194 (cit. on pp. 31, 139).

[ZSD+20] S. Zhao, Y. Sheng, Y. Dong, E. I.-C. Chang, Y. Xu. “MaskFlownet: Asymmetric Feature Matching with Learnable Occlusion Mask”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 6277–6286 (cit. on pp. 32, 78, 80, 126).

[ZW17] S. Zweig, L. Wolf. “InterpoNet, a Brain Inspired Neural Network for Optical Flow Dense Interpolation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 6363–6372 (cit. on p. 28).

[ZWPT21] F. Zhang, O. J. Woodford, V. A. Prisacariu, P. H. S. Torr. “Separable Flow: Learning Motion Cost Volumes for Optical Flow Estimation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, 2021, pp. 10807–10817 (cit. on pp. 23, 30, 31, 45, 56, 57, 139).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature