Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Improved RAFT Architectures for Optical Flow Estimation

Johannes Schäufele

Course of Study:	Informatik
Examiner:	Prof. Dr.-Ing. Andrés Bruhn
Supervisor:	Prof. Dr.-Ing. Andrés Bruhn, Azin Jahedi, M. Sc., Jenny Schmalfuß, M. Sc.
Commenced:	July 6, 2020
Completed:	January 27, 2021

Abstract

The estimation of optical flow, that is computing the displacement field between two images, is a useful tool in computer vision that has many applications as part of larger frameworks. RAFT [70], a recent method for optical flow estimation, has significantly improved the quality of results on realistic benchmarks over previous approaches, while simultaneously reducing model complexity and training cost. Despite these advancements, RAFT still has several shortcomings including its flow upsampling that can only capture high-resolution details to a limited extent, simple cost volume without normalization, and limited incorporation of multiple frames in sequences. Due to its novelty, the method has also not been applied to related tasks, such as unsupervised optical flow estimation. To address this, we propose several remedies to these mentioned shortcomings of RAFT, including different cost volume normalization strategies and alternative matching cost functions, as well as different flow upsampling strategies that can capture more high-resolution details. We also extend the method to unsupervised training as well as online training, which involves multiple frames of sequences. In the context of unsupervised training, we introduce learned losses that can be applied to arbitrary model architectures and improve results over traditional photometric and smoothness losses. Our online learning approaches yield an improvement over RAFT's warm start and use multi-frame consistency to improve performance on video sequences. We evaluate our approaches on optical flow benchmarks and find that our modifications represent improvements over RAFT when working within a limited computational budget. We also argue that these results should scale for training configurations without such limitations.

Kurzfassung

Das Schätzen von optischen Fluss, genauer das Berechnen eines Verschiebungsfeldes zwischen zwei Bildern, stellt im Bereich des Maschinensehen ein wichtiges Hilfsmittel dar, das oft als Teil komplexerer Ansätze verwendet wird. Das vor Kurzem veröffentlichte RAFT-Verfahren [70] zum Schätzen von optischen Fluss erzielt auf realistischen Benchmarks deutlich bessere Ergebnisse als vorherige Ansätze und verringert sowohl die Parameteranzahl als auch den Trainingsaufwand für das Modell. Trotz dieser Ergebnisse weist das Verfahren einige Einschränkungen, wie beispielsweise die Interpolation des Flusses, die hochfrequente Details nur eingeschränkt miteinbeziehen kann, sowie die naive Berechnung des Kostenvolumens ohne Normalisierung und die eingeschränkte Miteinbeziehung von mehreren Bildelementen einer Sequenz, auf. Da das Verfahren erst vor Kurzem veröffentlicht wurde, existieren auch noch keine Ansätze, die die neue Architektur unüberwacht trainieren. Wir behandeln dies, indem wir Änderungen, die diese Einschränkungen beheben können, vorstellen. So führen wir beispielsweise verschiedene Normalisierungsstrategien für das Kostenvolumen, sowie alternative Matchingkostenfunktionen und Flussinterplationsstrategien, die hochfrequente Details besser miteinbeziehen können, ein. Des Weiteren führen wir Erweiterungen ein, die es erlauben die Architektur unüberwacht zu trainieren und in Form des Onlinelernens Informationen von mehreren Bildern einer Bildfolge miteinzubeziehen. Im Rahmen des unüberwachten Trainierens führen wir erlernte Kostenfunktionen ein, die bessere Ergebnisse als herkömmliche photometrische und Glattheitskostenfunktionen erzielen und unabhängig von der spezifischen Netzwerkarchitektur angewandt werden können. Unsere Ansätze zum Onlinelernen verbessern weiterhin mittels zeitlich transitiver Flusskonsistenz Ergebnisse für längere Bildfolgen und liefern akkuratere Flussfelder als RAFTs warm-start. Wir werten die Ergebnisse unserer Ansätze anhand von Benchmarks für optischen Fluss aus und können festhalten, dass unsere eingeführten Änderungen gegenüber RAFT eine Verbesserung darstellen, wenn der Rechenaufwand beschränkt ist. Weiterhin erwähnen wir Anzeichen, die andeuten, dass dies auch für uneingeschränkten Rechenaufwand der Fall ist.

Contents

1 Introduction

1.1 Motivation

Optical flow is a useful tool for a large variety of computer vision and image processing tasks concerning image time-series or videos, and multi-view images. Its applications range from frame interpolation [2, 53, 20] and video compression [40], over depth estimation [73, 83] to tracking [42] and action recognition [5].

This is makes the estimation of optical flow given input frames an important process and motivates methods for efficiently inferring high-quality optical flow. However, the task of optical flow estimation is very challenging and can not be regarded as a generally solved problem yet. Challenges include large displacements, small fast-moving objects, occlusions, ambiguities, optical effects and noise under realistic conditions, as well as missing information. State-of-the-art methods are continually getting better [9, 25, 68, 23], but still show room for improvement, especially for estimating high-quality optical flow for realistic video footage, potentially in real-time.

Many recent approaches involve leveraging Convolutional Neural Networks (CNNs) to tackle the problem. These approaches have much smaller evaluation times and can be run in real-time, due to only requiring a forward pass of the network for inference, but require extensive training beforehand. They also significantly outperform traditional variational approaches with regard to the quality of the inferred optical flow, especially in more challenging and realistic settings, such as those involving motion blur. Despite these advantages, methods using CNNs have not yet completely solved the problem of estimating optical flow either and are steadily being improved. Such deep-learning-based methods also generally have the drawbacks of lacking interpretability and missing robustness guarantees.

One CNN-based method in particular, a recently-published approach titled RAFT: Recurrent All-Pairs Field Transforms for Optical Flow (RAFT) [70], stands out as being extremely successful, especially regarding high-quality results in challenging, realistic scenes. This is evidenced by most of the top-performing entries to the MPI Sintel Final benchmark [6] being RAFT-based, which also translates well to other benchmarks. Similarly, all top 3 submissions to the Robust Vision Challenge[1] 2020 Flow Challenge utilize a variant of RAFT. RAFT does not only shine result-wise, but also uses some slightly tweaked paradigms and new concepts that allow for such performance to be achieved with a simpler model having less than six million parameters. This allows not only for fast inference and potentially more effective optimization, but also much less computationally-intensive training of the architecture.

[1] http://www.robustvision.net

Despite all its successes, the RAFT method is far from perfect, its main shortcomings being the low resolution at which the main flow estimation is performed. This leads to incorrect estimation for small and especially small fast-moving objects. Additionally, the handling of the cost volume and part of the architectural design is suboptimal. All this limits the potential performance the model can achieve. Further, the RAFT method could be extended to incorporate more information about frames other than the current two inputs in a video frame sequence and be made more adaptive to allow better results to be achieved on a wide variety of scenes.

As such, this new method also presents an opportunity to use its newly introduced concepts and insights gained to create an improved variant building on it, that alleviates some of the shortcomings of the original method and potentially surpasses it in terms of performance and being well-posed. It is also an opportunity to apply its findings to different domains, such as unsupervised training. This would represent a next step in the series of incrementally improving methods for optical flow.

We aim to take advantage of this opportunity and do so by proposing changes to RAFT that address some of its shortcomings and explore possible directions of improvement. This includes changes to the architecture, alternative ways of computing and processing cost volumes and new flow upsampling approaches. Further, to explore the new space of possibilities opened up by RAFT, we extend the method to unsupervised training and online learning. Our goal in all this is to improve methods of estimating optical flow for realistic video sequences.

We find that our modifications can improve the performance of RAFT-based architectures noticeably. The learned losses proposed in the context of unsupervised training also represent improvements over their traditional counterparts and are applicable to CNN-based methods for optical flow in general. Our online learning approach allows results to be improved even further at evaluation time by incorporating multi-frame information.

1.2 Thesis Organization

In Chapter 2, we introduce all basic concepts needed for the rest of our work. We will base our work upon RAFT, which is a CNN-based method for optical flow estimation. Thus, we require CNNs and the basics of optical flow and optical flow estimation to be introduced. We start by introducing the mathematical foundations of images and function derivatives. Then, (Artificial) Neural Networks (NNs) in general and CNNs specifically are detailed, including training procedures and architectures. Following this, optical flow and basic concepts for optical flow estimation, as well as optical flow datasets for training are introduced.

Chapter 3 then recapitulates previous CNN-based methods for optical flow estimation, culminating in RAFT itself being presented in more detail. We go over early CNN-based methods and mention progressive improvements in more recent methods. Relevant concepts later used for RAFT, such as cost volumes and ghosting, are also introduced. We end the chapter by detailing RAFT's architecture and training procedure, and go over the method's strengths and weaknesses.

With RAFT itself being established, we move on to present our contributions in Chapter 4. We detail our proposed modifications to RAFT. This spans from simple fixes over cost volume normalization to different upsampling strategies. We also propose alternative training strategies for RAFT. This includes unsupervised training, which trains models without ground truth flow, and online training, which attempts to incorporate multi-frame information to improve evaluation results.

Having stated our modified approaches, Chapter 5 performs an evaluation of our methods on optical flow benchmarks. Using ablation studies, we identify which of our proposed modifications represent improvements and attempt to interpret our results. Visualizations of our results are also included and we end the chapter by comparing our approach with RAFT.

We close our thesis in Chapter 6 with a more high-level view of our work and concluding remarks. We also give an outlook by mentioning possible future directions of improvement and unattempted ideas that seem promising.

2 Background

In this chapter we lay the foundation for our work by introducing fundamental concepts and notation required for all further methods. As we ultimately aim to train RAFT-like CNN-based models to take in images and output the estimated optical flow, we must first introduce concepts relating to images, CNNs, optical flow, and the estimation of optical flow.

Beginning with mathematical foundations in Section 2.1, we define images and their notation as well as derivatives of functions. The gradient, which is a specific arrangement of derivatives, is later needed for the optimization of NNs.
Next, as CNNs are a special variant of NN, we introduce NNs in general as well as CNNs specifically in Section 2.2. This includes how networks are trained and typical architectural patterns, which are also used by many existing CNN-based methods for optical flow estimation, including RAFT.
Afterwards, the focus shifts to optical flow in Section 2.3, defining it and related concepts, such as occlusions.
The transition is then made to the discussion of how optical flow can be estimated. This begins with optical flow datasets in Section 2.4, which are used to train supervised methods, such as RAFT. Flow datasets are also used to evaluate methods for optical flow estimation and we will use a subset of the introduced dataset to evaluate our work.
Finally, useful concepts from non-learned methods for optical flow estimation are introduced, as they are later used by CNN-based approaches including RAFT. These concepts stem from local block-matching and PatchMatch approaches in Section 2.5, as well as global variational methods in Section 2.6.

The concepts and notation introduced in this chapter form the basis to understanding CNN-based methods for optical flow estimation, and specifically RAFT as well as our proposed improvements to RAFT.

2.1 Mathematical Foundations

2.1.1 Images

Optical flow estimation pipelines, including RAFT, begin by taking in input images. Images also appear as a representation for intermediate features and can be used as an encoding for optical flow. Owing to the prevalence they have throughout this work, we begin by introducing our definition of images.

Herein, we define a continuous c-channel 2D image I to be the function $I : \mathbb{R}^2 \to \mathbb{R}^c$, with the short-form $I(x, y)$ for $x, y \in \mathbb{R}$ denoting the value of $I\left((x, y)^\top\right)$.

Further, a (domain-)discrete 2D image with c channels of width w and height h, where $w, h \in \mathbb{N}^+$, is defined as the function $I : \{0, \ldots, w-1\} \times \{0, \ldots, h-1\} \to \mathbb{R}^c$. The notation $I_{i,j} := I(i,j) = I((i,j)^\top)$ for $i \in \{0, \ldots, w-1\}, j \in \{0, \ldots, h-1\}$ further provides a convenient short-hand for image element values, also referred to as pixel values.

In practice, the codomain of images is also discretized as either 8-bit integer vectors $\{0, \ldots, 255\}^c$ or vectors of floating point numbers of various differing representations.

2.1.2 Calculus

Image derivatives can allow capturing important features, such as edges. Derivatives can also be used to measure the smoothness of flow fields, which is used for optical flow estimation. Additionally, NNs in general, and CNNs such as the RAFT architecture specifically, are trained using gradients, which are a construct based on function derivatives. Necessitated by these use cases further down the line, we introduce basic concepts from calculus in form of derivatives, the gradient, and the Hessian.

In the following, we assume all functions that we take a derivative of and are not further specified to be sufficiently differentiable.

Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$, we denote the partial derivative of f with respect to the variable x_i as $\frac{\partial f}{\partial x_i}$ for all $i \in \{1, \ldots, n\}$. Further, we define the Jacobian of f to be

$$\mathbf{J}(f) := \begin{pmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}. \tag{2.1}$$

For a scalar function $g : \mathbb{R}^n \to \mathbb{R}$ with $f(x_1, \ldots, x_n) \in \mathbb{R}$, we refer to the transposed Jacobian of g as the gradient of g, denoted by

$$\nabla g := \mathbf{J}(g)^\top = \begin{pmatrix} \frac{\partial g}{\partial x_1} \\ \vdots \\ \frac{\partial g}{\partial x_n} \end{pmatrix}. \tag{2.2}$$

Additionally, we refer to the Jacobian of the gradient of g as the Hessian of g: $\mathbf{H}(g) := \mathbf{J}(\nabla g)$.

2.2 (Artificial) Neural Networks

As we base our work on RAFT, which is a CNN-based approach, we need to introduce CNNs specifically and NNs in general. In this section we go over NNs and their structure, as well as how they can be optimized and trained, including notes on associated challenges and possible remedies. We additionally introduce reoccurring architectural patterns, that reappear in later methods, such as RAFT.

We begin by introducing the general concept of machine learning that allows harnessing the power of learning to achieve better results than possible with handcrafted approaches in Section 2.2.1.
Following this, the most basic type of NN is introduced. It consists of perceptrons, introduced in Section 2.2.2, that use activation functions, as described in Section 2.2.3, and is called a multilayer perceptron. The combination of these components into a multilayer perceptron is detailed in Section 2.2.4.
Now that we can create simple NN architectures, we would like to apply machine learning to train these architectures for a given objective. This optimization process is detailed in Section 2.2.5. The optimization process uses the gradient and has its speed regulated by a learning rate. How learning rates can be effectively chosen and varied in schedules to speed up training is detailed in Section 2.2.6.
Following this, we give some insight into how the use of the gradient during optimization can lead to issues with training and mention remedies to address the issues in Section 2.2.7.
At this point we are ready to discuss reoccurring architecture types, specifically Recurrent Neural Networks (RNNs) in Section 2.2.8 and CNNs in Section 2.2.9, which are the type of NN most relevant to our work. Both of these architecture types are highly relevant to RAFT, as it is a CNN-based approach that borrows from RNNs to realize its iterative incremental estimation of optical flow.
Having introduced the upsides and details of NNs, we mention some challenges with NNs in general in Section 2.2.10. This includes the comparatively their low interpretability when compared to hand-crafted approaches. We then mention some techniques that can provide more insight into NNs in Section 2.2.11, helping with the interpretability issues. This also includes outputting statistics than can be helpful when working with and debugging NNs in practice.

Through the introduction given in this section, all following CNN-based approach for optical flow estimation and RAFT in particular should be understandable in regards to their architecture and training. We also leverage some of the presented remedies to issues with NN training in our own method to improve training.

2.2.1 Machine Learning

Machine learning describes the approach of automatically learning a model for a task by fitting to an objective, possibly supported by training data, rather than manually designing an algorithm for the task directly. Instead of defining a function that realizes the task directly, a family of functions $\{f_\theta\}_\theta := \{f(\theta, \cdot)\}_\theta$, differentiated by their parameters θ, is used to search for suitable parameters θ. After search, the parameters should have values such that f_θ either minimizes the given objective function or failing that represents a good solution or local minimum amongst all functions in the family.

This is typically done by first manually choosing a family of functions thought appropriate for the task, choosing an initial set of parameters θ according to a manually decided distribution, and then iteratively updating the parameters to better fit the given objective function. It is also possible to optimize the family of functions used iteratively instead of manually choosing it, an example of this being neural architecture search [38].

(Artificial) Neural Networks (NNs) are a way of parametrizing models for arbitrary functions, loosely inspired by biological neural networks. A basic building block of NNs, directly inspired by biological neurons, is the perceptron.

2.2.2 Perceptron

A perceptron [61], as sketched in Figure 2.1, takes in a fixed number $k \in \mathbb{N}^+$ of scalar inputs and outputs a single scalar value. It is parametrized by a scalar value weight w_i for each input and an additional bias scalar b and is further defined by the choice of its activation function $f : \mathbb{R} \to \mathbb{R}$. The output is computed as

$$\text{Perceptron}(x_1,\ \ldots,\ x_k) = f\left(\left(\sum_{i=1}^{k} w_i x_i\right) + b\right). \tag{2.3}$$

The activation function or nonlinearity enables nonlinear relationships to be modeled.

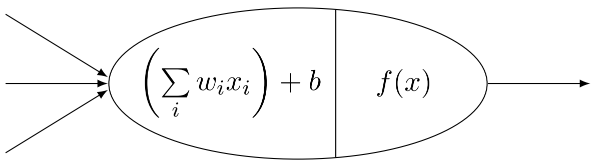


Figure 2.1: Schematic layout of a perceptron. The inputs x_i are weighted by their corresponding parameters w_i and summed up, before adding the learned bias b and finally passing the result through the activation function f and outputting the result as the perceptron's output.

2.2.3 Activation Functions

Functions typically chosen as the activations functions include the sigmoid function and its variant, the Tanh function, which restrict outputs to the range of $[0,\ 1]$ and $[-1,\ 1]$ respectively. Another activation function often chosen in practice is the Rectified Linear Unit (ReLU) function [11, 15, 51]. It is inspired by biological thresholding and only allows positive values to pass through, clamping the rest of the output range to zero:

$$\text{ReLU}(x) := \max(x,\ 0). \tag{2.4}$$

Empirically, ReLU activation functions seem to allow for better learning in general [13] and are thus very commonly chosen as nonlinearities [8]. There also exist many other possible activation functions, some of which are variants of the previously mentioned functions.

2.2.4 Multilayer Perceptron

Perceptrons can be combined by using the outputs of perceptrons as the inputs to further perceptrons, which allows stacking layers of perceptrons, yielding a Multilayer perceptron, as depicted in Figure 2.2. It is capable of learning more complex, indeed arbitrary, functions [18]. Multilayer perceptrons are typically the most basic form of NNs used as architectures for learned models.

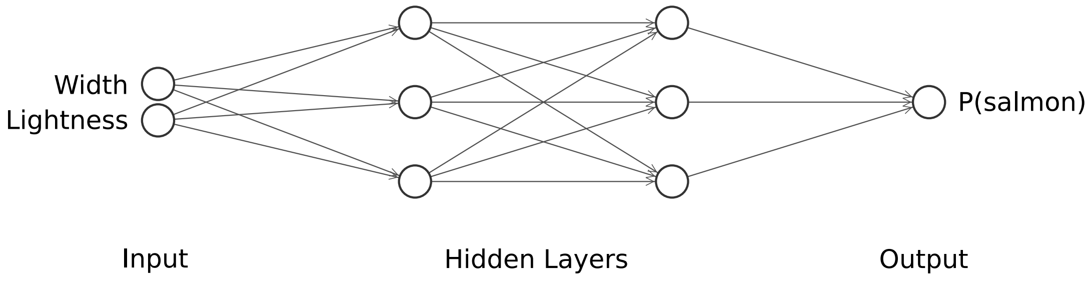


Figure 2.2: Example topology of a multilayer perceptron. From the left, two scalar inputs are fed into two consecutive layers of perceptrons, with three perceptrons in each layer, before their outputs are combined to the final output to be predicted by a single perceptron on the right.

2.2.5 Optimization

The parameters of NNs are generally first initialized by drawing initial parameters according to a weight initialization distribution, which aims to provide random weights that do not lead to numerical instabilities for typical inputs and often takes the chosen activation functions into account. After initialization, the parameters are iteratively optimized to reduce the value of the chosen loss function. Though evolutionary algorithms that allow for less restrictive optimization objectives and more flexible network topologies [66] and other optimization algorithms for NNs exist, NNs are primarily optimized via (stochastic) gradient descent, which is efficiently implemented via backpropagation. This requires objective or loss functions to be a differentiable scalar function (or at least a subgradient should be efficiently computable) and works by iteratively updating parameters by moving in the direction of steepest descent, given by the negative gradient of the loss function with respect to the parameters. Specifically, gradient descent updates parameters θ after time step or iteration k for a given loss function and model function f as follows:

$$\theta^{k+1} = \theta^k - \alpha \nabla \mathrm{loss}(f_{\theta^k}). \tag{2.5}$$

The size of the update step, determined by the learning rate α, influences the speed of learning. In stochastic gradient descent, only a small random subset of the training data, a so-called minibatch, is used to compute an approximation of the gradient for efficiency reasons, though this typically still yields a good enough estimate of the actual gradient.

Several optimizers improving upon stochastic gradient descent have been developed, including momentum-based methods, which aim to speed up convergence. One of the most notable optimizers is Adam [31], for which each parameter is assigned its own individual learning rate that is adapted during training, and the update direction is modified based on statistics of previous gradients. Adam has become the de facto default choice for optimizing NNs and works well for a wide variety of different architectures even given only minimal hyperparameter tuning, though in certain scenarios other optimizers can outperform Adam. AdamW [39], a variant of Adam, modifies the effect of weight decay in Adam to be more comparable to its effect with stochastic gradient descent and otherwise functions the same as Adam.

2.2.6 Learning Rates

Choosing appropriate learning rates for optimization is important for the speed of learning and quality of the resulting model. If the chosen learning rate α is too small, the model only improves very slowly, whereas when α is too large, the (estimated) gradient is no longer accurate enough for the large step size. This can potentially lead to a worse learned model, oscillation, or even divergence. Larger learning rates can on the other hand help parameters escape from local minima, where smaller learning rates would lead to the model getting stuck in said minima. To determine the value range appropriate for the learning rate, learning rate tests, as proposed in [63], can be employed to facilitate learning rate search in addition to typical exploration methods of the hyperparameter space.

Further, various learning rate schedules, which vary the learning rate over training iterations rather than keeping it a constant, have also been shown to be able to improve convergence. In case of optimizers with individual learning rates, such as Adam, the schedules instead vary the permissible range of values for each parameter. RAFT itself, and following it our method, uses a series of OneCycleLR schedules, which can be interpreted as a single instance of a cyclic learning rate schedule.

In a cyclic learning rate schedule [63], learning rates are continuously varied. This is typically done linearly, between the minimum and maximum value of the appropriate range for learning rates. Such appropriate ranges of learning rate can be determined though a learning rate test. Varying the learning rate between this minimum and maximum repeatedly over the course of training, potentially also decaying the overall learning rate, to encourage convergence, has been shown to be able to significantly reduce the required training iterations necessary for convergence. One possible interpretation is that decaying towards the minimum learning rate leads the model to a local minimum, which is periodically interrupted by larger learning rates allowing escape from bad local minima, resulting in a good local minimum being found more quickly than without the schedule.

Building on the idea of cyclic learning rates, a learning rate schedule was proposed that can exhibit even faster convergence [64] using a much larger maximum learning rate for only a single cycle, which quickly moves from the minimum learning rate to the maximum learning rate, and then slowly decays back to the minimum learning rate. This schedule, which we will also refer to as OneCycleLR, can be interpreted to lead the parameters close to a good local minimum utilizing the regularizing effect of large learning rates at the schedule's peak, before decaying the learning rate to converge towards a good local minimum.

2.2.7 Gradient Pitfalls

Gradients play a central role in optimizing NNs using gradient descent. Anomalies in the obtained gradients can cause significant issues with optimization or even catastrophic failures. When gradient magnitudes get very large, referred to as exploding gradients, optimization becomes unstable and may diverge or even lead to numerical failures. On the other hand, for very small gradient magnitudes, referred to as vanishing gradients, parameters only change very slowly, requiring many steps for convergence. Vanishing gradients can be observed in the close to flat regions of the sigmoid or TanH activation functions. In the extreme case where gradients are zero, parameters are not updated at all. This can occur in the flat region of the ReLU activation function, as defined by Equation (2.4). A ReLU activation can thus reach a state, in which its output will permanently be zero and may never recover, effectively removing the contribution of a subnetwork from the output. This effect of ReLU becoming permanently stuck in the flat region is referred to as the dying ReLU problem, which can be avoided by choosing a different activation function, such as one of the ReLU variants that does not feature a flat region. Such candidates include LeakyReLU [78], which allows negative values to pass through heavily dampened, instead of completely clamping their value to zero:

$$\mathrm{LeakyReLU}(x) := \begin{cases} 0.01x & x < 0 \\ x & x \geq 0 \end{cases}. \tag{2.6}$$

Whenever a weight parameter is multiplied by an input or an activation, that is, the output of an intermediate layer, the contribution towards the gradient of that parameter is also multiplied by this value. To prevent the issue of vanishing or exploding gradients to arise as a result, the magnitude of inputs and activations can be normalized, typically to a mean of zero and a standard deviation of one. As such normalization generally makes training more stable by preventing these issues and sometimes faster by increasing gradient magnitudes, normalization layers that perform this transformation are commonly employed in NN architectures. Since the originally proposed batch normalization [26], many other normalization strategies, such as instance normalization [72] or more the more general group normalization [76], have been introduced, that mainly differ in how the statistics for the mean and variance are computed and how activation are grouped before applying normalization.

Building on these insights, we will employ both LeakyReLU activation functions and various normalization strategies in our approach to mitigate issues with vanishing and exploding gradients.

2.2.8 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network that can process a variable length sequence of inputs by iteratively applying the same operator. This operator takes in the next element of the input sequence and outputs the next element of the output in each iteration, while a hidden state is internally maintained, updated, and passed to subsequent iterations. Though typical architectures use LSTM units [17] to fill the role of the operator, more recently a variant called gated recurrent unit (GRU) [7] that represents a simplified, but less powerful version of LSTMs, has also been proposed for this purpose.

As the same operator is applied consecutively several times in RNNs, a transformation that multiplies its output would lead to the parameter gradients being multiplied several times in the completely unrolled version, potentially leading to exploding gradients for multipliers greater than one. To counteract this potentially destructive effect, gradient clipping [56] is often performed for RNNs, in which gradients are transformed in a way that preserves the update direction, but limits the magnitudes to be below a chosen threshold.

RAFT follows a strategy that iteratively estimates optical flow in increments. As such, it borrows from RNN architectures to create its own update operator. For similar reasons, RAFT also employs gradient clipping.

2.2.9 Convolutional Neural Networks

Ultimately, we want to use NNs to estimate optical flow by taking in and processing two images, then outputting optical flow, representable as an image. As such, we need to adapt the concept of NNs to allow images to be processed.
Though multilayer perceptrons can be used to process images by treating each pixel of an image as a separate input or output, this completely ignores the 2-dimensional nature of images. This also only allows images of fixed size to be processed, produces a very large number of parameters which leads to both overfitting and high computational costs, and has no spatial invariances or direct concepts of 2D neighborhoods. Convolutional Neural Networks (CNNs) are a special variant of neural networks, which can instead be used for processing images, though they are also occasionally employed for non-image data. The concept of convolutions can be generalized to an arbitrary number of dimensions, but in the following we will only discuss with 2-dimensional convolutions.

In the following, we introduce convolutional layers, the key component of CNNs and describe how they are used as part of larger convolution blocks that make up a CNN. We then go over to discussing residual connections, which are important for deep CNNs, which includes architectures like RAFT. Finally, we go over architectural patterns for CNNs, which are used in the upcoming CNN-based methods for optical flow.

2.2.9.1 Convolutional Layers

Instead of fully connected layers, as with the multilayer perceptron, weight sharing is employed to drastically reduce the number of parameters and add translational invariance while respecting the 2-dimensional nature of images. CNNs replace the many perceptrons in a layer with a single convolutional layer, which is based on convolution filters known from image processing and parametrized by learnable discrete convolution kernels. By applying the same filter at each location, only a small stencil needs to be parametrized and the operation is translationally invariant. Somewhat analogous to a perceptron, a convolutional layer takes in k 2-dimensional scalar image channels, applies the convolution filter at every location to produce j output channels, and finally adds a separate bias to each of the output channels. Activation functions are still typically applied after convolutional layers, but not part of the layer itself. The kernel is parametrized by its base size, which is often of the form $2n + 1 \times 2n + 1$ for $n \in \mathbb{N}_0$, and contains one 2D stencil for each unordered pair of an input and an output channel, defining the contribution of that input channel's neighborhood to the output channel's value, giving it an overall size of $j \times k \times 2n + 1 \times 2n + 1$.

2.2.9.2 Convolution Blocks

For a kernel with a base size larger than 1×1, locations near image boundaries do not have a sufficient amount of neighbors for a value to be computed. Thus, if the image size should be kept the same, the inputs can be padded to make up for this. Typical padding includes zero padding, where padded regions are filled with the constant value of zero, and reflection padding, which mirrors image values in the padded regions. Though zero padding is frequently employed before convolutions, it introduces artificial boundaries that can introduce artifacts, such as a non-existent edge being detected, due to the values at image boundaries and the padded regions being dissimilar. In contrast, reflection padding does not introduce as many artifacts as zero padding.

When the resolution of channels should be decreased, strided convolutions or pooling can be used. Strided convolutions only evaluate output channels at a subset of locations, such as only evaluating at one of the locations in a 2×2 square, leading to the output channel with a quarter of the input resolution. Instead, the full-resolution output can also be computed and subsampled by pooling, such as by taking the minimum, mean, or maximum in each 2×2 square as the subsampled value. Should the resolution be increased instead, transposed convolution or upscaling interpolation can be employed. As transposed convolutions introduce undesirable checkerboard artifacts [54], bilinearly upsampling channels followed by performing a normal convolution is typically preferable to employing transposed convolutions.

Convolutional layers are thus generally part of a larger block of layers that first pads the input appropriately for the following convolution, followed by the convolutional layer itself. The convolutional layer may optionally be strided or followed by a pooling layer for downsampling, and is followed by an activation layer, that applies the activation function for each channel, and finally a normalization layer, such as a 2D variant of batch or instance normalization. Though there is some debate about whether the activation layer or the normalization layer should be applied first, we choose activation followed by normalization. This leads to the output of each block, which is also the input for the next block and thus the activation that affects gradients for the weights in the next block, being normalized. Normalization should not be performed for the last block, to enable the model output to follow arbitrary distributions, or when information about the magnitude of activations should be preserved [30].

2.2.9.3 Residual Connections

In deep CNNs, where the depth measured by the number of consecutive applications of convolutions can be large, typical architectures show noticably worse results for higher depths. This issue has been shown to be alleviated by introducing residual or skip connections [16] that add the outputs of earlier convolutions to the input of convolutions further down the chain, effectively skipping over all convolutions in between, thus reducing the minimal depth of the network measured as the shortest path between the input and the loss in the computational graph. It has also been shown that introducing such residual connections effectively smooths the loss surface for the architecture, allowing for better optimization [34].

2.2.9.4 Architectural Patterns

When a scalar or a few scalars should be inferred from a single image, such as for classification tasks, encoder architectures are often employed. These architectures repeatedly downsize activations after one or more blocks of convolutions until activations have a very small spatial size and then feed these small activations into fully-connected layers, consisting of multilayer perceptrons, which output the scalars to be predicted. If images should be synthesized from a small feature set, such as for generative models, a decoder architecture can be used instead. It often begins with the input being passed through a multilayer perceptron, and then repeatedly upsizes activations between convolution blocks before outputting a higher-resolution image.

Encoders and decoders can be combined into encoder-decoder architectures that take in an image, internally downscale it to a bottleneck representation before upscaling it again, and output an image of the same resolution as the input, in this case fully-connected layers are often dropped. This architecture can be used to transform images, while taking features from several scales of resolution into account and benefit from a large effective receptive field. U-Net [60] architectures are an improved variant of encoder-decoder, which adds residual connections from the encoder outputs to the decoder inputs for each resolution scale. The added connections reduce the minimal depth of the model and allow the decoder to preserve high-resolution details not captured by the lower resolution outputs preceding the decoder blocks.

2.2.10 Challenges

Neural networks and especially deep learning are still active fields of research, in which many insights, such as used architectures, layers, normalization, and objective choices are only practically justified by their empirical success, but not well-understood or motivated theoretically [62]. Further, neural networks show issues with interpretability and strong guarantees, as an intuitive understanding can not easily be gained for larger, more complex architectures, the way the single component of a perceptron can be understood. Automated analysis and reasoning techniques are also still developing and currently do not lend themselves to interpretation of many models in real-world use.

2.2.11 Insights into Neural Networks

Unlike for traditional software, it can often be very hard to tell whether a NN is functioning properly, due to their learned nature, and tooling that grants insights into or debugs NNs being limited and often not universally applicable. Errors in the implementation might not manifest themselves directly and NNs may still appear to behave correctly and learn some behavior, even if the actual implementation differs significantly from what one expects it to be. If, on the other hand, the network does not learn properly as expected, it can be very hard to pin down where exactly the problem lies. The training data, loss function, training loop implementation, model architecture, hyperparameters, numerical issues, or even the feasibility of the task itself could all be responsible.

This makes outputting and visualizing auxiliary information about the model, data, and training invaluable, both to pin down errors and even realize that unexpected behavior exists in the first place. Such information includes statistics, such as the minimum, mean, mean magnitude, and maximum, and visualizations of input data, outputs, intermediary activations, and parameter gradients. Gradient magnitudes in particular are useful for determining issues with improper learning rates, vanishing or exploding gradients, and parts of the architecture through which gradients only flow with small magnitude, potentially motivating additional normalization or skip connections. On a more fundamental level, gradient magnitudes can help determine if the model or certain parts of it are even learning at all.

At a higher level, advanced visualization techniques, such as guided backpropagation [65] can be used to reason about specific outputs or the general relationship between inputs and outputs, though these are often limited to certain types of tasks, most prominently classification. A reasoning technique that is easily available is to determine the derivatives of certain parts of the output, intermediate activations, or combined metrics thereof with respect to certain parts of the input or activations to determine their relationship in a small neighborhood around the examined sample, though this does not allow for as much insight as with more advanced techniques.

2.3 Optical Flow

As we eventually aim to pursue the task of optical flow estimation, we must introduce this task and concepts related to optical flow. This includes error measure for optical flow, which we will later use to evaluate the results of different methods for optical flow estimation. Further, the concept of occlusions is significant to optical flow estimation, as the task is more difficult in occluded region. Taking occlusions into account can also improve the result of methods for optical flow estimation.

Estimating optical flow is the task of finding a displacement field corresponding to the motion between two images. More formally, given two continuous 2D images f and g, also referred to as input frames, both mainly defined on the same, typically rectangular, domain $\Omega \subset \mathbb{R}^2$, which are commonly images taken at subsequent times or from different perspectives, a motion field $d : \Omega \to \mathbb{R}^2$ is estimated, such that each location $f(x, y)$ in f corresponds to $g(x+u, y+v)$ for $(u, v)^\top = d(x, y)$ for all $(x, y)^\top \in \Omega$. In practice, a domain-discrete version of this problem is solved, however, the displacements $(u, v)^\top$ are still real- or float-valued and locations $g(x + u, y + v)$ are seen as being on a continuous domain of g, typically realized by higher than 0th order interpolation.

The location pairs $(x, y)^\top$ and $(x + u, y + v)^\top$ are correspondences in the sense that if the real object and its location that mainly contributed to the value of $f(x, y)$ had been physically marked with a sufficiently small marker, that marker would end up mainly contributing towards the value of $g(x+u, y+v)$ in the second frame. Alternatively, if one imagines perfect geometrical descriptions of the recorded frames to exist, the first object and location hit by light originating from the sensors responsible for recording the values of $(x, y)^\top$ in the first frame and $(x + u, y + v)^\top$ in the second frame in direction of the focal point, or equivalent for different camera models, would be the same, barring more complex optical effects.

2.3.1 Error Measures

A typical error measure for optical flow is the Endpoint Error (EPE), which is the per-pixel L2 distance between the predicted and the ground truth flow vectors, denoted by $\tilde{d}$ and d respectively:

$$\mathrm{EPE}(\tilde{d},\ d)_{i,j} := \left\| \tilde{d}_{i,j} - d_{i,j} \right\|_2 . \tag{2.7}$$

The measure is usually averaged over the entire flow domain to obtain an overall measure for a flow field. Although such an averaged metric is sometimes referred to as the Average Endpoint Error (AEE), herein we also refer to it as EPE, when a single averaged value is used for an entire flow field.

2.3.2 Occlusions

Unlike a scene's actual motion, which typically happens in a 3-dimensional space of unconstrained size, optical flow is seen from the perspective of a projection of that scene onto a limited-size 2-dimensional domain. This discrepancy gives rise to occlusions, such as when objects leave the limited domain of a frame or overlap each other, causing multiple objects to be mapped to the same projected location, though barring transparency, only one of them will be visible. Such occlusions can be detected as inconsistencies between the forward and backward flows, that is, following the motion from the first frame to the second frame using the forward flow and then back to the first frame using the backward flow leads to a location different from the starting point, as illustrated in Figure 2.3. Using such a forward-backward consistency check, locations exceeding a threshold of distance between the original location and the followed one, regions can be marked as occluded. Generally speaking, estimating optical flow is more difficult in occluded regions, as objects involved

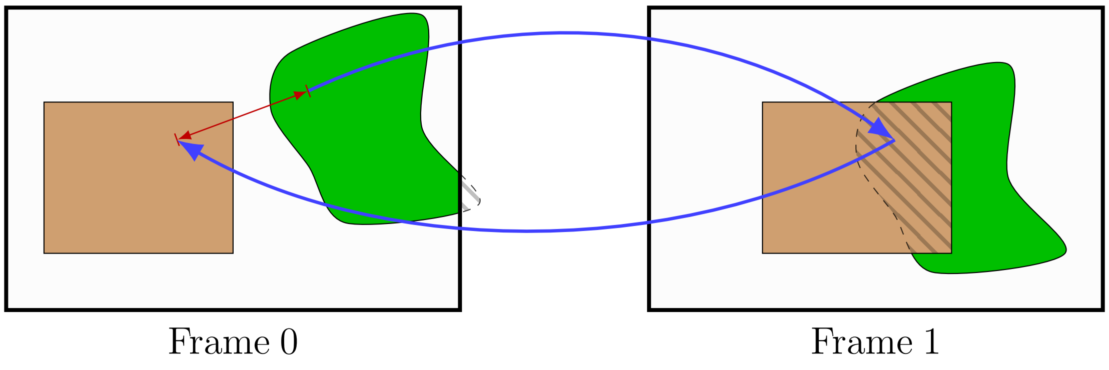
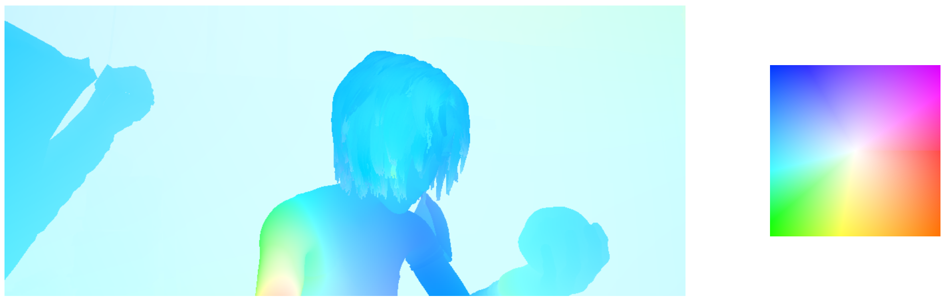


Figure 2.3: Illustration of occlusions being detectable via failure of forward-backward consistency check. Two objects change their positions from one frame to another. Following the forward flow from the first frame into the second frame and then the backward flow at that same location back to the first frame, one notices a large distance between the destination and the original starting point, indicating an occlusion.

in the motion to be estimated may no longer be visible in one of the frames, due to being covered up or out of the frame. This only leaves a reduced amount of information that can be used for the estimation of motion in occluded regions, leading to estimation being more difficult in such regions.

2.3.3 Visualization of Flow Fields

Flow fields can be densely visualized by viewing flow vectors at each location in polar coordinate form, then coloring each location using polar angle (flow direction) to select hue and radius (flow magnitude) to determine lightness. This allows a compact representation of both the direction and magnitude of the flow, though the hue mapping may take some getting used to. In practice, flow magnitudes are first normalized and clipped to equalize the brightness range of visualizations across different flows and constrain colors to the limited range of values supported by display devices or print. A sample flow visualization and visualization guide can be found in Figure 2.4.



(a) Sample flow field visualization **(b)** Flow visualization guide

Figure 2.4: Sample flow field visualization and flow visualization guide. The flow field is taken from the alley_1 sequence of the Sintel dataset [6]. The visualization guide shows the color-coding of a flow vector originating in the center pixel for each location. Note that flow magnitudes are scaled to make the range of color values in the visualization be more in line with the range of possible output colors.

2.4 Optical Flow Datasets

CNN-based methods for optical flow including RAFT need to be trained on training data. When the task to be learned is optical flow estimation, optical flow datasets are used for this purpose.

Deep neural networks, such as those typically used for challenging tasks in computer vision, often possess on the order of millions of parameters on the low end. Fitting a model with such a large number of degrees of freedom to data requires a similarly enormous amount of data and appropriate regularization. Furthermore, increasing the training set size with more high-quality data is an aspect that significantly improves performance and keeps improving performance logarithmically [67], across different architectures, tasks, and optimization objectives for such models, as well as representing the most important and often time-consuming factor for practical applications.

Training CNNs to estimate optical flow in a supervised manner requires input frame images and corresponding ground truth flows, whereas unsupervised training only requires input frames, which can be obtained from arbitrary videos. Obtaining ground truth flow is much harder than obtaining input frames and can be done in a number of ways. Firstly, ground truth flow can manually be created by humans, matching points of the input frames accordingly. As this is extremely tedious, time-consuming, and potentially error-prone, only a very limited amount of data can be annotated this way. Alternatively, additional sensors and calibration can be used when recording input frames, to determine flow via recorded geometry and motion, hidden textures, or using other information captured with special sensors to reconstruct motion. This allows for capture of real-world images with corresponding flow fields within a smaller margin of error and can be done for larger video sequences, but requires additional, potentially costly tools and may not yield dense accurate estimates. Finally, scenes can be synthetically rendered, allowing the optical flow to be directly determined from the known underlying geometry, resulting in accurate dense flow fields. Though rendered scenes may not be as realistic as real-world images, this acquisition method gives the highest-quality flow fields and can be done automatically and much more cheaply for a potentially arbitrary amount of scenes.

Early datasets and benchmarks for optical flow only included few samples, did not always feature dense ground truth flow and were not always realistic. They were sourced from simple renderings, manually annotated images, or objects with hidden texture [4, 55, 21, 1]. A more complete overview of flow datasets and augmentation with focus on use for CNNs can be found in [48].

2.4.1 KITTI Datasets

The KITTI datasets, KITTI2012 [12] and KITTI2015 [50], are driving-related datasets that were obtained by attaching multiple cameras and special sensors to a car, followed by recording video and auxiliary data while driving the car and thus falls into the category of measured datasets. The obtained flow fields are not dense and contain mostly rigid motion, some artifacts have also been introduced by post-processing. As a benchmark, it offers realistic data, especially for approaches in the area of autonomous driving. Though the dataset is larger than many previous datasets, it does not contain a sufficient number of samples to be directly used for training CNNs, making it more relevant for fine-tuning and as a benchmark.

2.4.2 Synthetic Datasets

Synthetic datasets offer the advantages of providing dense high-quality ground truth optical flow that can be generated at scale cheaply and automatically, at the cost of being less realistic. The feasibility of generating a large number of samples makes synthetic datasets the most suitable candidate as training data for data-hungry CNNs.

FlowNet [9], the first method to infer optical flow using only CNNs, introduced the FlyingChairs dataset, which consists of photographic background images with superimposed objects, in this case chairs, a portion of which exhibit random 2-dimensional geometric transformations in the next frame, and corresponding flow fields. As such samples can be generated easily, the dataset's size was chosen according to the needs of training data for CNNs. Further variants of the dataset, FlyingChairsOcc [23] and FlyingChairs2 [24], which include additional data aspects, such as backward flow and occlusions, have also been generated.

Since then, the FlyingThings3D [47] dataset has emerged, which is an optical flow and scene flow dataset created by rendering 3D scenes with random objects and motions, making it somewhat more realistic and challenging, while still including a large enough number of samples for CNNs.

2.4.2.1 Sintel Dataset

The Sintel dataset [6] is a synthetic dataset, based on the Blender open movie Sintel, and consists of rendered scenes from the movie, optionally including optical effects such as motion blur. Though not consisting of real-world images, it is a more realistic dataset, with its complex motion and optical effects making it more challenging than other benchmarks. As the amount of scenes and samples is fairly limited, the Sintel dataset primarily serves as a benchmark, and can be seen as a relevant benchmark for estimating motion in realistic free-form videos.

2.4.3 Data Augmentation

Data augmentation creates more data from a limited dataset by applying various transformations many times to each sample of the original dataset. This creates a larger amount of data, which may be required for training, though the quality of augmented data is not comparable to additional samples from the original dataset distribution. The application of such transformations can also be used to encourage robustness of the trained model under such transformations, such as color and brightness changes, geometric transformations, or noise.

Typical transformations include modifications of the image values, such as hue, brightness, and contrast changes, geometric transformations in the form of translation, scaling, and rotation, as well as crops, which can serve as an augmentation as well as a way of bringing data samples to the required resolution for a model's input.

As the change of the flow field between augmented frames can easily be computed from the underlying transformations, augmentations can even serve to create training samples with known ground truth flow in an unsupervised setting, or serve as a self-supervision constraint.

ScopeFlow [3] has also highlighted the importance of not altering the distribution of flow fields between the original and the augmented datasets, especially when fine-tuning, as this can lead to the model fitting to a skewed distribution, leading to worse performance on the dataset with the original distribution. This was most prominently observed when continuously cropping and can be avoided by cropping in discrete steps, such as dividing the original image into a fixed number of equal-sized pieces and uniformly choosing one of them as the augmented cropped image.

2.5 Concepts from Local Methods for Optical Flow

Block matching approaches are simple methods for optical flow estimation that locally minimize matching costs. Despite their simplicity, they provide important concepts also used in more advanced methods for flow estimation. Matching costs specifically have been adapted to cost volumes and are used in CNN-based methods for flow estimation. One of RAFT's advancements specifically involves cost volumes and uses cross correlation. Correlation is one of the three matching cost functions that will be introduced in this section. Correlation having been adapted to cost volumes also makes the other introduced cost functions starting points for possible cost volume calculation and we will adapt the presented sum of absolute differences to cost volumes in our method.

A basic family of approaches for optical flow estimation are PatchMatch approaches. These function by assigning a matching cost to any pair of patches, that is small local neighborhoods around a point, in each of the two input frames, where small costs indicate more similar or conforming patches. A flow field is then estimated by comparing the patch for each location in the first frame to many other candidate patches from the second frame, typically in limited windows around the original patch location to reduce computational cost, and choosing the displacements that correspond to minimal matching costs. This step is often used as part of a larger optimization routine that propagates displacements among neighboring patches and adds random search. This is motivated by the idea that certain feature descriptions based on a point and its local neighborhood are invariant or corresponding under the motion between the frames, due to the motion of points being caused by motion of larger objects that are largely unchanged by the motion.

Potential ways of choosing matching costs, also used in practice for block matching, include the sum of absolute differences and (normalized) cross correlation. Both can either be directly applied to the frames represented as gray value, color images, or on multi-channel feature images derived from each input frame. The sum of absolute differences (SAD) is based on the L1 metric of the difference of the two patches p^a and p^b with domain Ω,

$$\mathrm{SAD}(p^a,\, p^b) := \left\| p^a - p^b \right\|_1 = \sum_{i \in \Omega} \left| p^a_i - p^b_i \right|, \tag{2.8}$$

is inexpensive to compute, robust under outliers, due to the sub-quadratic penalization, and can be interpreted as the distance between feature vectors for each patch.

(Cross) Correlation (CC), on the other hand, is computed through channel-wise multiplication of feature vectors reduced by summation and is instead maximized for corresponding patches:

$$\mathrm{CC}(p^a,\, p^b) := p^a \cdot p^b = \sum_{i \in \Omega} p^a_i \cdot p^b_i, \tag{2.9}$$

and can be normalized for mean and variance:

$$\mathrm{NCC}(p^a,\, p^b) := \frac{\left(p^a - \overline{p^a}\right) \cdot \left(p^b - \overline{p^b}\right)}{\left\| p^a - \overline{p^a} \right\|_2 \cdot \left\| p^b - \overline{p^b} \right\|_2}, \tag{2.10}$$

where $\overline{p}$ refers to the mean of p. Normalized Cross Correlation (NCC) is invariant under affine value transformations by virtue of the normalization performed. Correlation in general is more expensive to compute, though still reasonable in cost, and can be geometrically interpreted as the (normalized) inner product of feature vectors, relating to the cosine of the angle between vectors. If patch features are corresponding, point in a similar direction, and have values of large magnitude and the same sign for many channels, correlation is maximized, indicating matching patches.

Though correlation is commonly used for block matching directly on image gray values, the sum of absolute differences, or other simple distance metrics, appear to be more intuitive and possibly effective when dealing with feature vectors or descriptors rather than the images themselves.

2.6 Concepts from Variational Methods

Variational methods represent a more advanced approach of optical flow estimation that attempts to globally optimize flow fields. The energy functionals used by variational methods to assign a global cost to a flow field are often used as a starting point for unsupervised losses for CNN-based methods. This is due to unsupervised training requiring a loss function that assigns a loss value to a flow field given only input frames and no ground truth flow, just as for energy functionals. As we later present a method for training RAFT in an unsupervised setting, we introduce components of variational energy functionals as concepts from variational methods. We also introduce pyramid schemes used by variational methods, as this concept too can be and has been applied to CNN-based methods for optical flow. This also applies to the associated warping strategy.

Beyond brute-force, block and feature-matching approaches, and local-energy methods, variational methods are traditionally an effective tool for optical flow estimation. Fundamentally, variational approaches formulate an energy functional that assigns a single energy value to each potential flow field, where low energy values correspond to more desirable flow fields, based on the input frames and other objectives. After defining an initial flow field, they attempt to successively modify the current flow field to one of lower energy.

Though variational approaches are not utilized in the kinds of methods mainly discussed in this work and will thus not be introduced as thoroughly, some concepts originating from variational models can and have been applied in more recent works using learned approaches.

Energy functionals of variational methods for optical flow typically comprise a data term, a smoothness term, and possibly other terms for regularization or based on additional constraints, such as material or depth.

2.6.1 Data Term

Data terms for variational methods are the counterpart of matching costs for block or patch matching methods described previously, in that they penalize discrepancies of the visual structure between the first frame and the second frame at the corresponding locations, implied by the current flow field, possibly also involving their neighborhoods. Though quadratic penalization of the difference between image or image derivative channels is often used for easy differentiation, subquadratic penalizers quite similar to the sum of absolute differences are also used for better robustness under outliers.

We will also refer to functions that assign a cost value to a flow field based on the assumption that photometric invariants hold for corresponding locations as photometric losses. Data terms for variational methods represent instances of photometric losses.

2.6.2 Smoothness Term

Smoothness terms measure the smoothness of a given flow field, favoring smooth flow fields with lower energy, formalizing the assumption that the motion comprises larger objects moving continuously and encouraging good matches of the data term in certain locations to propagate to the surrounding area. As smoothness typically does not hold at the boundary between objects moving in different directions, the influence of smoothness may need to be reduced at certain locations, which is often done at image boundaries, on the assumption that these often coincide with object and thus motion boundaries.

Further, smoothness can be chosen as first order smoothness, which penalizes flow fields using the magnitude of first order derivatives, completely discounting only constant motion, and second order smoothness, which in turn penalizes second order derivatives, and allows affine motion to stay unpenalized. Depending on the underlying motion, both first and second order smoothness may be more appropriate, sometimes even differing between various image locations, leading to adaptive approaches, which combine both first and second order smoothness by introducing per-image-location weights for each order, choosing them as is most appropriate at that location.

2.6.3 Pyramid Schemes

More advanced variational methods are sometimes embedded in a pyramid-based coarse-to-fine framework, wherein optical flow is first estimated at a very coarse scale before iteratively increasing the resolution of the estimated flow field back to the original frame resolution. This allows for better optimization of more complex energy functionals and allows large displacements to be correctly estimated, but typically only for large objects rather than small, fast-moving objects.

2.6.3.1 Warping

Warping appears in the context of these pyramid schemes and allows the second frame to be compensated by a motion field, in order to bring it into the same coordinate system as the first frame. This is done by backward registration, which samples the second frame at the locations corresponding to the first frame's coordinate grid, that is the flow field being added to original coordinate grid.

Optical flow can subsequently be estimated between the first frame and the compensated second frame once more, and the resulting flow field can be added to the initial flow field to obtain a combined estimate of the motion from the first to the original second frame, while respecting the refinement of the flow field obtained after compensation.

Chapter Outlook

Having introduced the basic concepts required regarding CNNs and optical flow and discussed traditional non-learned approaches for optical flow, we can now move on to learned methods for optical flow and specifically look at how optical flow can be and has been estimated using CNNs.

3 Related Work

To put our work into context and introduce the methods we will be building upon, in the following, we introduce a body of previously proposed approaches and concepts for optical flow estimation.

Having established some non-learned approaches for optical flow estimation in the previous chapter, we first discuss how the concept of learning can be applied to optical flow estimation in Section 3.1. It details how and at which different levels at learning can be incorporated.
We then move to CNN-based approaches for optical flow in Section 3.2. We mention the first CNN architectures for optical flow estimation and more recent iterations on this. Recurring patterns and shared concepts, such as the used supervision signal and architectural patterns of these methods are also noted, some of which are used in RAFT.
We then take an excursion from supervised learning to unsupervised learning of optical flow estimation in Section 3.3. We mention the typical loss functions, building in part on variational methods, and self-supervision. Semi-supervised learning is also briefly introduced. This knowledge forms the basis for the RAFT-based unsupervised and semi-supervised approach we will later propose.
We then move on to the missing two aspects for understanding RAFT, the first of which is cost volumes, introduced in Section 3.4. We explain how cost volumes related to the previously introduced matching costs and how previous CNN-based methods have used them for optical flow estimation.
Next, the issues occurring when working with warped input frames are detailed in Section 3.5. Additionally, previous work that achieves better results by avoiding these issues is mentioned. RAFT will join these methods in avoiding warping and the issues that are associated with it.
Finally, we introduce RAFT itself in more detail in Section 3.6. Having introduced all necessary concepts for understanding the method, we explain RAFT's components and how they are used to estimate optical flow. We also go over RAFT's strengths and impressive results, as well as some shortcomings of the method which could be improved.

With RAFT and other previous CNN-based methods of optical flow being introduced, this chapter will form the basis of understanding for our work, which builds on these methods and aims to improve upon them.

3.1 Towards Learned Methods for Optical Flow

Machine learning and optimization processes in general can allow improvement beyond what is typically feasible with manually designed solutions, making leveraging this potential for improvements desirable for optical flow estimation. Bridging the gap from variational methods, where the flow field is optimized according to a hand-crafted objective, to learned approaches can be done by introducing learning at various different stages. This is mainly characterized by which components are kept fixed and which are to be parametrized and optimized. The choice of how to parametrize and optimize such components can also have a large influence on the quality of the resulting method.

At the most basic level, the image features used for feature matching or in the photometric losses of variational methods can be learned. Going a step further, the entire matching cost or the data and smoothness terms themselves, or just a single combined term, could be made subject to optimization.

Leaving the framework of traditional approaches, one can also see optical flow estimation as a general optimization process that begins with an initial flow estimate, then iteratively improves the current estimate through an update. Learning a suitable update operator immediately yields a method for inferring flow, parametrizable in the number of iterations and potentially the way flow is initialized, if an initialization operator was not learned or initialization was set fixed. At its most opaque, an estimator for optical flow takes in input frames and outputs a corresponding flow field. Thus, a model realizing this transformation can be learned through end-to-end training.

Finally, meta-learning embeds the architecture [38], optimization objectives (see Section 4.6.1), or even training process of models [58], like the ones previously mentioned, into an optimization framework.

From here on out, we focus on learned approaches involving CNNs, due to RAFT and our work building on it being CNN-based.

3.2 Previous CNN-based Approaches

Motivated by the then-recent success of deep learning and CNNs for computer vision tasks [33, 10], FlowNet [9] was the earliest published successful attempt to infer optical flow using only CNNs and did so in an end-to-end manner. Though its results were only of limited use and not inherently impressive compared to the state of the art, even at the time, it had a significant runtime advantage. More importantly, it introduced many concepts laying the foundation of optical flow estimation using CNNs, some of which are still used in many recent approaches.

Following FlowNet, CNN-based methods have generally been supervised approaches and used the EPE loss, also adapting the training schedule of training on the FlyingChairs dataset first, followed by the FlyingThings3D dataset, and finally a finetune on benchmarks such as KITTI or Sintel, with learning rates decreasing for each subsequent dataset. Though there are often other small changes, most methods mainly differed in architecture, which still typically employed a U-Net-like architecture like FlowNet. Unsupervised methods using CNNs exist, but typically perform much worse than their supervised counterparts. Nonetheless, the potential to use arbitrary videos as training data and the ability to train on frames for which flow should be inferred without requiring ground truth makes this a promising direction of research.

Notable recent improvements on FlowNet include PWC-Net [68], which estimates flow in a coarse to fine manner, inspired by variational methods with pyramids, as well as IRR [23], which uses weight-sharing to reduce parameters and estimates forward flow, backward flow, and occlusions in a rolled iterative scheme.

3.3 Unsupervised Training

Unsupervised training aims to train a model without using any ground truth. For optical flow estimation this implies a network being trained using only input frames, without any ground truth flow. Training in an unsupervised manner has the advantage of being able to train on arbitrary video data, which is available in high quality in abundance, as well as being able to train on input frame scenes before estimating the optical flow between them. Despite these promising advantages, unsupervised methods typically perform much worse than their supervised counterparts, as proper guidance through a loss function is significantly more difficult without the availability of ground truth.

Unsupervised losses often borrow from variational energy functionals, which similarly assign a cost to estimated flow fields using only input frames as external data. In addition to data terms and smoothness terms similar to those from variational energy functionals, some methods also add self-supervised loss components, which use the model itself or data and data transformations to synthesize pseudo ground truth, which can in turn be used with typical supervised losses. One example of self-supervision would be to obtain a flow field for two frames using the current model, then applying a known transformation to the input frames, and using the appropriately adjusted flow field as ground truth for the transformed frames [35]. A pretrained teacher model can also provide flow fields based on input model, which the trained model learns from, essentially distilling the knowledge of the teacher model [37]. There also exist data-driven approaches [36] that generate frame pairs and corresponding flow fields using geometric transformations, very similar to how some synthetic datasets, such as the FlyingChairs dataset, are created. Self-supervised loss components can enforce consistency constraints on a model and allow capturing the actual metric used for penalizing errors in the flow fields, such as the EPE or L1 distance, which can not easily be done using photometric losses.

UFlow [29] is a recent survey that compares the effect of different components for unsupervised optical flow. It finds the census loss [49] to be a good choice for the photometric loss, suggests stopping gradients at occlusions, and introduces cost volume normalization, which normalizes features for mean and variance before correlation, similar to normalized cross correlation, amongst other findings.

3.3.1 Semi-supervised Training

Supervised and unsupervised training can be combined into semi-supervised approaches, where ground truth exists for some, but not all inputs in the training set. Applied to optical flow, one could first train a network in the typical supervised fashion on synthetic datasets, followed by unsupervised training on realistic data or the benchmark chosen to evaluate. Such an approach could benefit from both the typically better results from supervised training as well as improved performance by specifically fitting to the data later used for evaluation, without requiring ground truth to do so. A potential pitfall could be that the unsupervised loss in the latter part of training degrades model performance, due to the worse guidance exhibited. A possible remedy could be to mainly employ self-supervised losses and drop data or smoothness terms, or weigh them down considerably. Self-supervision is an appropriate choice, as it is not biased, in the sense that a perfect model should have a self-supervised loss of 0, preventing performance degradation. Further, the model has already been trained fairly well and is this a suitable teacher for itself, already yielding quite reasonable flow estimates.

3.4 Cost Volumes and Correlation

One established way of determining dense correspondences is to formulate a function to determine how conforming locations in the first and the second frame are. Examples of such functions are the cost functions for block matching or data terms in variational methods. The resulting cost volume is 4-dimensional and stores a cost for every possible displacement (2D) for each location in the first frame's domain (2D). Due to the computational and memory requirements, the range of possible displacements is sometimes limited to a small local neighborhood, as is often done for block matching. Correspondences can then be determined by choosing the displacement with minimal cost value for each image location in the cost volume.

When FlowNet [9] introduced a CNN architecture for optical flow, in addition to a fully convolutional architecture, called FlowNetS, a variant including a layer that computes correlation cost, FlowNetC, was also introduced. The correlation layer used to do so computes the correlation of feature maps for the first and second frame for a limited set of small displacements, sampling a small part of the corresponding cost volume. Initially this variant was introduced, as it was not clear whether a fully convolutional network could effectively infer flow on its own, but correlation layers showed merit of their own and were continued to be included in approaches building on FlowNet [25, 68, 23]. The variant without it, FlowNetS, and architectures based on it were used less frequently, due to the quality of inferred flow being worse.

Some recent approaches, such as VCN [81], have lifted the restriction on displacement ranges and instead compute and process full 4D cost volumes. These methods have seen improved performance on benchmarks as a result, though processing 4D cost volumes still remains comparatively expensive.

Cost volumes are related to warping in that each slice of a cost volume across the entire image domain with a fixed displacement field corresponds to the cost metric applied to the first frame and the second frame at displaced locations or alternatively the same location in the second frame warped according to the displacement field.

3.5 Warping, Ghosting, and Neighborhoods

Warping introduced for variational approaches using pyramids reappears for CNN-based methods, such as for PWC-Net [68], which similarly uses a pyramid approach. Further, approaches that estimate flows iteratively, such as IRR [23], also warp frames to estimate the remaining displacement field between the first frame and the second frame warped by the currently estimated flow.

Despite such widespread use of warping, it is accompanied by two significant issues. Firstly, occlusions can lead to so-called ghosting [82, 27], where the warped frame includes two copies of a moving object: one at the actual new position and another around its original location to the frame, which is the undesirable artifact introduced by warping. The second appearance is caused by the occluded geometry behind the object in the second frame being mostly stationary, but attempting to sample this occluded geometry at the location instead results in sampling the object in front of it. If flow estimation is now performed with a warped image as the second frame, methods might incorrectly find correspondences involving the ghost object, rather than the actual object, due to its similar or even identical appearance, resulting in incorrectly estimated flow. One way of prevent correspondences involving such ghosted regions, is to detect and mask away all affected regions, as is done by MaskFlownet [82], which lends it improved performance on benchmarks.

A second issue with estimating optical flow for warped frames is the non-uniform displacement caused by warping itself, which leads neighborhoods around points to be distorted. This causes feature descriptors based on neighborhood information to be less consistent around more distorted neighborhoods and may have more dramatic effects when neighborhoods are effectively cut off or partially moved far away by discontinuities in the flow. Possible remedies that were proposed include deformable [41] or even learned [79] neighborhoods, with which the distortion of neighborhoods can be sidestepped, by never directly performing warping to begin with, or counteracted to an extent by allowing neighborhood samples to be repositioned inversely to the distortion.

3.6 The RAFT Method

RAFT: Recurrent All-Pairs Field Transforms for Optical Flow (RAFT) [70], a recently published CNN-based method for optical flow, introduced several new paradigms and significantly improved state-of-the-art performance on benchmarks, such as MPI Sintel. It iteratively estimates optical flow in increments and uses a novel lookup operator to sample a full 4-dimensional cost volume, while simultaneously reducing both model complexity in terms of the number of parameters and the number of training iterations required without sacrificing the quality of the estimated optical flow.

We first give a general overview of RAFT's components and architecture, and then look at individual components in more detail. This begins with the cost volume and corresponding cost volume sampling, followed by its flow upsampling strategy. We then review the training procedure and how evaluation is performed on benchmarks. We end by summarizing RAFT's strengths and shortcomings, providing candidates for improvement in our approach following this.

3.6.1 Architecture

RAFT estimates optical flow iteratively in increments. Beginning with an initial flow estimation, each update iteration computes an incremental update of the estimated optical flow, based on samplings from a cost volume. RAFT takes in two input frames, internally estimates optical flow at $\frac{1}{8}$ of the resolution of the frames, to benefit from pooled information and lower computational costs, before finally upsampling flow back to the full resolution and outputting the upsampled flow field.

The architecture features two encoders, one infers features used for correlation and the other provides initialization and context input in each iteration. Both take in input frames and output computed features at $\frac{1}{8}$ resolution. The feature encoder shares its weights between both input frames, while the context encoder is only applied to the first frame.

The first step in the pipeline consists of inferring the features to be used for correlation from the input frames using the feature encoder and precomputing the correlation for all displacement pairs. Next, the context encoder is applied to the first frame, yielding context information of the reference frame and an initial hidden state.

RAFT's updates borrow from the design of recurrent neural networks. Each iteration, a hidden state is updated based on the iteration inputs of the static context information, the currently estimated flow field, and the result of the correlation lookup for the currently estimated flow field. This is a sampled version of the cost volume, as described further below. An incremental update to the current flow field is them computed from the new hidden state. This process can be interpreted as fixed-point iteration, allowing the number of iterations to be varied between training and evaluation. The update operator features a ConvGRU, which is a convolutional version of a GRU [7], wherein perceptrons are replaced with convolutional layers. Optionally, standard convolutions are replaced with separable convolutions in the ConvGRU, to allow for a more lightweight parametrization of convolutions with larger kernel sizes. A complete update begins with passing the correlation and flow components through a small network that compacts their representation, that includes convolutions with large kernel sizes for the flow component, enabling propagation of flow values in a larger neighborhood. This compact representation is then passed to the ConvGRU along with the previous hidden state, to compute the new hidden state. From the new hidden state, small decoder networks compute the flow update and additional values used for flow upsampling.

3.6.2 Cost Volume

The "all-pairs" in RAFT refers to the method constructing a full 4D cost volume. Though VCN [81] has already employed 4D cost volumes for optical flow estimation previously, the sampling and processing of the cost volume is done differently, as described in the next section. Using (unnormalized) correlation as the cost metric, the volume is efficiently precomputed using a single matrix multiplication. Storing a full 4D cost volume, even at $\frac{1}{8}$ resolution as is done for RAFT, requires a significant amount of memory, which is why RAFT also proposes an alternative method of computation for correlation, in which only the values actually sampled from the cost volume are computed ad hoc every iteration. The alternative computation offers the benefit of having only linear memory and time complexity in the number of image pixels, as opposed to the quadratic complexity of the original implementation.

Where VCN uses 4D convolutions to filter the cost volume directly, RAFT samples 2D slices of the cost volume each iteration and processes them using 2D convolutions. This limits the amount of information from the cost volume taken into consideration, but significantly reduces the complexity and runtime of the processing.

3.6.3 Correlation Lookup

In each iteration, the cost volume is sampled using the current flow estimate to index the displacement at each location. Similar to but distinct from Devon [41], the cost volume is also sampled in a square neighborhood of displacements around the current flow estimate. In particular, this sampling effectively yields the correlation between features of the first frame at a single pixel with every element of a square neighborhood in the second frame around the same location displaced by the current flow estimate.

This is also performed for several downsampled versions of the correlation, where the cost volume is pooled over the displacements, leading to larger effective neighborhoods for coarser versions of the cost volume, without having to proportionally increase the neighborhood size. Intuitively, this allows correspondences to be found by selecting neighbors with maximal correlation, prioritizing coarse neighbors, and using the respective displacement to update and improve the current estimate of the flow. Once correlation is maximal for the central location on coarser levels, the finer neighborhoods can be used to locate the correct displacement at higher resolution.

The sampled neighborhoods are undistorted and do not directly involve warped images, avoiding the pitfalls otherwise encountered when using warping, as described in Section 3.5. This strategy is an alternative to previously used correlation layers and is easily adaptable to multi-scale approaches. The lookup itself is very cheap if the cost volume has already been precomputed. In the alternative implementation, correlation is only computed for values that should be looked up.

3.6.4 Flow Upsampling

The flow field is upsampled by choosing the high-resolution flow to be a weighted average of the flow values in a 3x3 neighborhood of the low-resolution flow, using learned interpolation weights. This constrains high-resolution flow values to be in the range of the low-resolution neighborhood values and allows meaningful interpolation to be learned easily, especially in regions where the flow field is affine, but also prevents small or thin objects with a flow value significantly different from the larger surroundings to be properly captured by this upsampling scheme. As nine interpolation weights have to be learned for each of the 8x8 high-resolution values corresponding to a single low-resolution value, this also introduces a significant amount of additional parameters for interpolation. Nevertheless, this interpolation strategy improves results compared to constant or bilinear interpolation of the flow field and allows high-resolution details to be captured more accurately.

The upsampling is performed after each iteration before applying the loss, effectively disconnecting the low-resolution flow field from the ground truth flow field, but the chosen parametrization still leads to proper guidance.

3.6.5 Training

RAFT uses an L1 loss between the predicted and the ground truth flow to train the network in a supervised manner. The L1 loss is similar to the sum of absolute differences and is defined as the L1 norm of the difference between the predicted flow $\tilde{d}$ and ground truth flow vector d:

$$\mathrm{L1}(\tilde{d},\, d) = \left\| d - \tilde{d} \right\|_1 . \tag{3.1}$$

It stands in contrast to the EPE loss, which is the L2 norm of this difference instead. Much like for the EPE loss, an average is taken over all image locations as the single loss value for an entire flow field. The loss is applied on the upsampled flow after every iteration, where later iterations are weighted higher than early ones.

Gradients for the flow and lookup indices are stopped after each iteration, which limits the size of the computation graph and improves performance, but also encourages the update operator to be more independent of the previous iterations, though information still flows through the hidden state from one iteration to the next.

Gradient clipping is performed for much the same reason it is used for RNNs, due to the update operator's recurrent nature. AdamW is used as the optimizer with a small weight decay component.

3.6.5.1 Learning Rate Schedule

RAFT follows the typical training schedule already mentioned in Section 3.2. It first trains on the FlyingChairs dataset, followed by training on FlyingThings3D, and then finetuning on Sintel. Each dataset is used to train the network for 100k iterations with learning rates decreasing for datasets later in the schedule.

A separate instance of OneCycleLR is used as the learning rate schedule for each dataset. Separating learning rates between datasets is advisable due to the different distributions and loss (gradient) magnitudes of the datasets as well as the desire to only finetune on datasets used for later schedules. The series of OneCycleLR schedules can also be seen as a form of a single cyclic learning rate schedule over all datasets, which decays the learning rate for each dataset.

3.6.6 Evaluation

At evaluation time, more iterations of the update operator are performed to achieve better results. Optionally, a warm-start can be used for inference on sequence datasets, which uses the forward flow estimated directly prior in the sequence and projects it into the space of the current frame, to be used as the initialization of the flow field instead of the usual zero initialization.

3.6.7 Shortcomings

RAFT introduces a novel way of computing correlation and an incremental update operator, achieving significantly improved results on benchmarks with a simpler model and cheaper training. Despite this, the method has several shortcomings and aspects that could be improved.

The architecture mostly uses simple convolution blocks that do not take into account more recent insights into activation functions and architectures for deep CNNs, such as discussed in Section 2.2.9. Due to performance concerns of correlation and other operations, the resolution of the flow and features is limited to $\frac{1}{8}$ of the original resolution and the upsampling strategy is unable to capture certain high-resolution details, limiting accuracy for small details at high resolution. The correlation still consumes a significant amount of memory, even at this reduced resolution. Evaluation on sequence datasets offers warm-start initialization, but much more information could potentially be derived from more than two input frames or entire sequences. The newly introduced concepts also represent a first iteration of a new idea, that may still have room for refinement and could be applied to different domains, such as unsupervised training.

Chapter Outlook

These shortcomings and potential for application to different domains represent an opportunity to build on the strengths of RAFT, improve upon the method, and apply it to other areas. We will attempt to take advantage of this opportunity and present potential improvements and application of the method to other problems in Chapter 4, describing our method.

4 Improved RAFT Architectures and Training Procedures

The previous chapters have established the foundations of optical flow estimation using CNNs, as well as previous methods for doing so and RAFT itself. We can now move on to investigating how the RAFT approach can be improved and built upon to create new methods for optical flow estimation.
Our aim in all this is to create methods of estimating optical flow that provide good results in realistic scenarios, for use in larger computer vision pipelines. We specifically want to estimate high-quality optical flow for realistic video sequences, and do so within the confines of our computational budget. This stems from realistic video being the most abundant and frequently used source for input frames, as well as many practical applications benefiting from high-quality results.
As it is difficult to directly measure the desirable metrics previously described, we use the performance on realistic benchmarks as a stand-in. While this does not lead to the same metrics, the benefit of being able to easily perform an evaluation outweighs this concern. We specifically choose the Sintel benchmark [6] to fill this role, as it is the benchmark closest to real free-form video, that can easily be evaluated from with the available ground truth flow. The Sintel dataset is also challenging enough that results on it should generally translate well to other applications.
We follow a holistic approach that examines all areas of RAFT for potential improvements and propose modifications to different parts of the architecture, the used loss functions and training process in general, as well as the estimated outputs and how flow is estimated for frame sequences. While this does mean that we may examine some areas more shallowly compared to an approach that only focuses on only a single area, we believe that our approach is the most promising for the overall improvement of results. This allows finding the areas that allow for the largest improvements and enables combining modifications from different areas that can eclipse the results for each modification on its own. It stands in contrast to being limited by the potentially smaller improvements that can be gained when restricted to a single area, even with more focus on this area.

In this chapter, we present our approach that builds on, improves, and extends RAFT to further domains. We limit ourselves to introducing our approach conceptually in this chapter, and go over implementation details and obtained results in our evaluation chapter, following this. We begin by introducing baseline modifications to the RAFT architecture in Section 4.1. This concerns the RAFT architecture and represents small changes motivated by the empirical findings discussed in Section 2.2.
Following this, we examine the cost volume computation, downsampling, and indexing more closely in Section 4.2. We propose fixes, normalization strategies, and alternative computations that aim to improve the quality of the matches derivable from the cost volume. Such improved correspondences can directly elevate the quality of the resulting optical flow.
After the cost volume, we turn to the flow upsampling used by RAFT in Section 4.3. The shortcomings of the upsampling discussed in Section 3.6.4 are addressed by our three proposed

alternative upsampling strategies, aiming to improve full-resolution flow results and capture high-frequency details of the motion.
Next, the training data and used augmentations are discussed in Section 4.4. Our inclusion of occlusion information and focus on realistic data motivates some changes in the selected datasets and performed augmentations.
The training process itself is then examined more closely in Section 4.5. We propose using a different supervised objective that could improve performance on benchmarks and adjust training parameters to better fit our computational budget.
An unsupervised approach for training RAFT is then presented in Section 4.6. Unsupervised training can improve results, as it can use large quantities of readily available high-quality input frames without requiring ground truth flow. It can also be used to train on benchmarks before evaluating on them, allowing to improve benchmark results by taking the specific distribution of the input frames into consideration.
Training at evaluation time also forms the basis of online learning, presented in Section 4.7. It takes the idea further by not only training on the entire benchmark dataset, but directly fitting to the specific sequence or even frame pair. Additionally, multi-frame information is taken into account, all in all allowing for more specific details to be considered and results to be improved even further. These improvements generalize to practical applications, where estimating flow on images sequences is often the norm, leading us to the improved methods for optical estimation we aim to create.

4.1 Baseline Modifications

We begin by introducing a number of smaller modifications to the RAFT architecture. These are motivated by Section 2.2, which introduced the vanishing and exploding gradient problem, their relation to activation functions and normalization, as well as their effect on training. The considerations for deep CNNs mentioned by ResNet [16] also lead us to introduce skip connections. These mainly aim to improve gradient flow and can thus lead to better training and results. We also opt to use the alternative correlation implementation to fit our computational budget.

To prevent dying ReLU issues, as described in Section 2.2.7, we replace all ReLU activations with LeakyReLU activation functions.

Instead of zero-padding, which can result in artifacts at boundaries, we use reflection-padding where appropriate. This includes padding of inputs and all padding before convolutions and the padding before `unfold` for the convex upsampling. Notably, padding the cost volume using reflection padding is not appropriate, as this would result in spurious costs for displacements corresponding to padded locations, so this is kept as zero padding. The need for this modification and the change in coordinate downscaling mentioned in the next section becomes evident when using visualization techniques as described in Section 2.2.11, and may otherwise easily go unnoticed.

Normalization layers in the form of instance normalization are added after convolutions in the feature and context encoders, to improve gradient flow and prevent exploding and vanishing gradients, as described in Section 2.2.7. Normalization is not introduced into the update block, to prevent inferred flow magnitude information being lost due to normalization.

Following [16], we introduce residual connections into the update block, to reduce the shortest paths from the output to the parameters in the computation graph and work with a smoother loss landscape [34] more favorable for optimization. Specifically, a skip connection is introduced to skip the ConvGRU and another skip connections allows skipping over the flow encoding convolutions in the update block. We also replace the flow decoder with a residual block.

Due to the computational and memory costs for all pairs correlation being quadratic in the input size, we create an optimized, more efficient implementation of all pairs correlation and neighborhood sampling written in CUDA, based on the alternative implementation described in RAFT and Section 3.6.3.

4.2 Cost Volume Processing

The construction and sampling of the cost volume is the main way in which RAFT identifies correspondences. Further, the quality of the matches derivable from the sampled cost volume significantly influences the quality of the estimated optical flow, as such matches can directly yield the flow vectors in unoccluded regions. This makes improving the matches derivable from the sampled cost volume a primary target for improving the overall result quality of RAFT-based architectures.

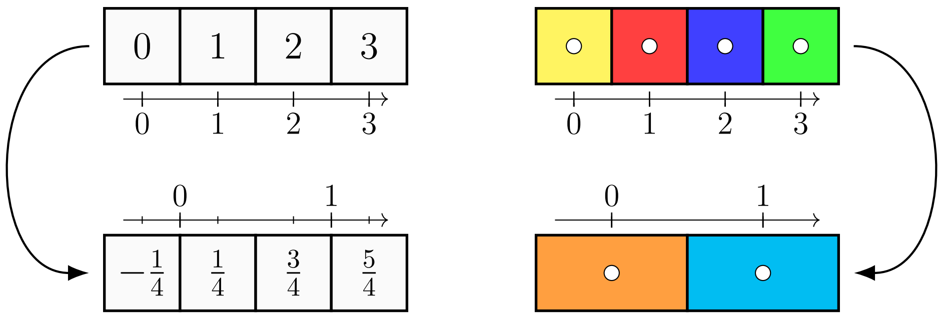
In this section, we introduce several modifications to the calculation and sampling of the cost volume, all aiming to improve derived match quality. We begin with modifications to the cost volume downsampling in Section 4.2.1. This includes a simple bug fix that properly centers downsampled versions of the cost volume and a transformation of the cost volume pyramid that can allow for better interpretation of the pyramid's individual levels.
Following this, we propose cost volume normalization strategies in Section 4.2.2, which can help the network better compare different match candidates against each other after magnitudes are normalized. They also achieve the same positive effects typical normalization layers for NNs bring with them.
Finally, Section 4.2.3 introduces several different variants that fundamentally change how the cost volume is calculated, with the aim of improving match quality by allowing for more general and learnable matching costs. We also take inspiration from block matching to compare correlation to an L1-based cost function, similar to the sum of absolute differences defined in Section 2.5.

4.2.1 Cost Volume Downsampling

RAFT samples the cost volume using coordinates that correspond to adding the currently estimated flow to a standard coordinate grid. When indexing downsampled versions of the cost volume, the following modified coordinates c^i are used for indexing level i, where c^0 represent the original coordinates without downsampling, g the coordinate grid with value ranges $[0, w-1] \times [0, h-1]$ for width w and height h, and $\tilde{d}$ denotes the flow field used to index the correlation volume:

$$c^i = \frac{c^0}{2^i} = \frac{g}{2^i} + \frac{\tilde{d}}{2^i}. \tag{4.1}$$

Adjusting the coordinates in this manner does not take into account that the location of the values in the downsampled cost volume lie at different positions, as illustrated in Figure 4.1.



(a) Coordinate downscaling **(b)** Image downsampling

Figure 4.1: 1-dimensional illustration of downsampling and correct coordinate downscaling. Small circles on the image to the right symbolize the point-like position of each pixel. The downscaled coordinates are obtained as the position of the high-resolution grid points on the low-resolution coordinate axis.

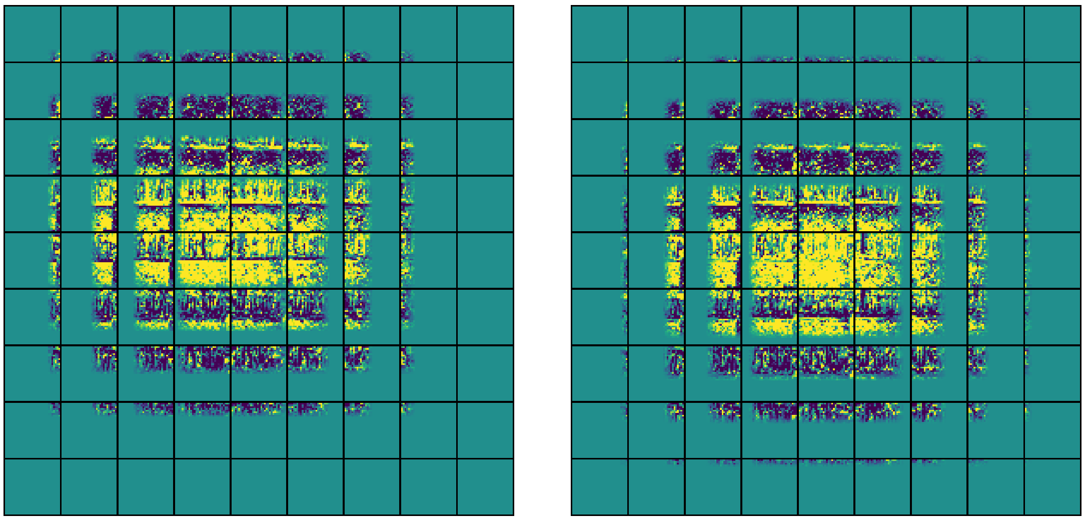
To address this, we instead use the following downscaling for coordinates:

$$c^i = \frac{c^0 + \frac{1}{2}}{2^i} - \frac{1}{2} = \left(\frac{g + \frac{1}{2}}{2^i} - \frac{1}{2}\right) + \frac{\tilde{d}}{2^i}. \tag{4.2}$$

The effect on the downsampled correlation is visualized in Figure 4.2.

We optionally transform the pyramid of subsampled cost volumes by subtracting an upsampled version the next-lower layer from each layer of the pyramid save the last, resulting in levels similar to those of a laplacian pyramid. This localizes correlation on specific scales, resulting in small localized peaks for correspondences at higher resolution levels, as opposed to having a spread-out peak around the correspondence with a slightly higher value for the peak localized at the higher resolution level. This transformation only affects the visualization of the cost volume and possible metrics applied directly to samplings of the cost volume. It does not change the expressibility of the network taking in the sampled correlation, as the subtracted layers can easily be added back in by the network, even on sampled versions of the pyramid. For visualizations, this change can make peaks in the cost volume caused by correspondences to be more clearly localized at higher resolution scales.

Sampled versions of the cost volume can be used to formulate metrics that measure the quality of the features and the inferred flow. This can be done by penalizing peaks in the sampled cost volume that do not correspond to the central location for the displacements given by the estimated or even ground truth optical flow on each pyramid layer. Such metrics encourage the estimated optical flow to coincide with the displacements of good correspondences, characterized by a large correlation response for the features at the locations encoded by the correspondence. They could even be used as part of a loss function for training. We opt for a slightly different approach directly penalizing feature maps instead, independently of the estimated flow, as described later in this section.



(a) Original downsampling **(b)** Our downsampling

Figure 4.2: Visualization of downsampled versions of the cost volume for the original and our modified downsampling. The pictured correlation sampling was downsampled by a factor of 8. The 9×9 box grid contains a box for each neighborhood offset, with the center box corresponding to an offset of 0. Each box shows the correlation value for each image location, with brighter colors representing larger values. One can observe that the correlation using our downsampling is properly centered, whereas the original version is skewed towards the top left corner.

4.2.2 Cost Volume Normalization

Using (unnormalized) correlation as a cost function is inspired by block-matching methods, but it is not clear that this choice is optimal. Normalized cross correlation improves results for block-matching methods and [29] finds similar improvements for correlation normalization with CNNs, suggesting that correlation normalization in RAFT-based architecture may improve results. To this end, we investigate different normalization approaches for correlation and cost volumes in general.

Traditionally, correlation is normalized for blocks or patches, however, the cost volume used in RAFT computes matching costs between two individual points, with a neighborhood given around the point in the second image. This motivates normalization across the entire domain or normalization across each neighborhood in the second image instead.

Normalizing features across the entire image domain represents the most straightforward approach and serves as the equivalent of normalized cross correlation. Such normalization was already proposed by UFlow [29] and can be realized by and also interpreted as a standard normalization layer. It is realized by passing the feature maps for both input frames through such a normalization layer before computing correlation. We employ group normalization with a single group for all channels on the features used to compute the cost volume for this purpose. We will also refer to this modification as feature normalization.

Normalizing each neighborhood patch in the sampled cost volume can additionally be employed to more easily pick the displacement corresponding to the best match in each patch. This is similar to normalizing all values in a patch, as is done for block matching. One downside to this is that correlation can no longer be compared between different neighborhoods, preventing selection of better matches to be propagated to locations with worse matches. To counteract this, the information can be re-introduced by concatenating the unnormalized center cost to each normalized neighborhood. We perform this normalization and concatenate unnormalized center costs, and refer to this modification as neighborhood normalization.

We employ both feature and neighborhood normalization for our cost volume. Both normalization types first subtract the mean value and then divide by the standard deviation, across the image domain or in each patch respectively.

4.2.3 Alternative Cost Functions

Beyond normalizing correlation, correlation-based cost functions can be further modified, replaced with cost functions based on other distance metrics, or cost functions can be learned themselves. Through harnessing the power of learning and allowing for more general cost functions, learned matching costs could improve the quality of matches derived from the cost volume.
Recently, several CNN-based methods have proposed modifications to cost volume calculations that appear to noticeably improve the quality of the estimated optical flow. Motivated by the success of these modifications, we propose changes to the computation of RAFT's cost volume. This includes simplified variants of the modifications proposed by the mentioned methods, typically to keep computational costs reasonable, while still benefiting from their improvements.

LCV [77], a recently published approach that modifies the correlation used in methods utilizing 4D cost volumes, including RAFT, proposes multiplying the feature map for the second frame with a learned positive definite matrix with a special parametrization. This allows weighing each feature channel separately and can account for correlation between the feature channels, allowing for a more general and potentially more effective computation of the cost volume. When using correlation, we employ the proposed approach by multiplying the feature map of the second frame with a learned matrix before correlation, but limit the matrix to be diagonal for simplicity. This allows reweighing the importance of each feature channel individually.

GOCor [71] instead proposes learning a function that transforms the second feature map in a way that incorporates information from both frames to make the resulting correlation matches more globally unique. We attempt to incorporate such a transformation without having to incur the costly components and optimization used by GOCor by computing new feature maps using convolutional layers using information from both feature maps. We perform this operation with unwarped feature maps, which does not allow for proper treatment of information between feature maps for correspondences, but still allows transformations of each feature map individually and information to be shared between feature maps if averaged over larger neighborhoods, assuming flow magnitudes that are sufficiently small. To reduce parameter count, we apply the same transformation for each feature channel, still allowing for transformations very similar to the initialization proposed by GOCor.

Beyond correlation, which was inspired by cost functions for block-matching approaches, other distance metrics, such as L1 distance, can also be used as a cost function, similar to the sum of absolute differences for block matching. In particular, L1 distance seems promising as a cost function between samples of a feature map, as it is more robust to outliers and it does not change the gradient magnitude in the backward pass, unlike correlation, in which gradient magnitudes are multiplied by the magnitudes of the values in the feature map. It is also computationally cheaper, though it no longer allows efficient precomputation via a single matrix multiplication. We optionally use L1 distance instead of correlation as the cost function to compute our cost volume, computing it when the cost volume is indexed using a variant of our alternative CUDA correlation implementation. Feature maps are normalized as with correlation, but we drop the multiplication of the second feature map with a matrix. For simplicity and efficiency of implementation, values are unmodified even if one of feature maps is not defined at the used offset, resulting in spurious values for this location, compared to the constant zero value one would get for correlation. To remedy this, a valid map with stopped gradients is passed to the network in each iteration that marks which displacements are valid and which fall out of the range of the feature map. Typically, spurious values are larger than values for corresponding locations, as the spurious values equal the L1 norm of the features for a single image as opposed to matching values which are typically close to zero, somewhat mitigating this issue.

Finally, DICL [75] proposes dropping correlation entirely in favor of a learned cost function. As learned 4D cost volumes incur a significant computation and memory cost, DICL embeds its flow estimation into a pyramid approach. This computational and memory cost prevents us from employing a similar learned cost volume, though the approach can directly be applied to the RAFT architecture by sampling the concatenated feature maps instead of their correlation when indexing the cost volume, optionally followed by additional processing, which could be a promising direction of research.

Inspired by DICL's efforts to make cost volume peaks more unique, we add a feature loss component that encourages the cost volume to have more corresponding values for ground truth flow displacements d than for random displacements $\tilde{d}$:

$$\mathcal{L}_{\text{feature}} = \left\| m_0 - \text{warp}(m_1,\ d) \right\|_1 - \left\| m_0 - \text{warp}(m_1,\ \tilde{d}) \right\|_1, \tag{4.3}$$

for the feature maps m_0, m_1 of the first and second input frame respectively, where warp refers to the warping operation described in Section 2.6.3.1. The feature loss is only applied in non-occluded regions, using ground truth occlusions to determine where the loss should be applied. Further, we add a uniqueness loss component that encourages feature map values to be more unique by encouraging the difference between feature map values at different locations to be large:

$$\mathcal{L}_{\text{unique}} = -\left\| m_0 - \text{warp}(m_0,\ r) \right\|_1, \tag{4.4}$$

where differing locations are realized via a random displacement field r. This loss component only features a single difference, as the difference between the first feature map and itself is trivially zero. Note that feature maps are normalized, thus preventing the uniqueness loss from becoming arbitrary small. The norms containing warped elements are only evaluated when the displacements used to warp fall into the corresponding image domain and the contributions are re-weighted accordingly for both losses. We observe that these loss components decrease, even when they are not optimized for, confirming that they represent useful properties for the network to have. This may also suggest that it is not strictly necessary to optimize for these objectives, but optimizing them might improve results nonetheless.

4.3 Upsampling

Upsampling of flow fields is required in the RAFT architecture, as flow is only estimated at $\frac{1}{8}$ resolution in the main part of the network, which significantly reduces the computational load and allows the use of all-pairs correlation. Ultimately, full resolution flow needs to be computed and naive upsampling, such as bilinear upsampling, limits the quality of the result, as it fails to capture high-resolution details of the flow. The convex upsampling strategy employed by RAFT improves results by yielding sharper flow changes at object boundaries and flow discontinuities, but fails to capture the motion of small and thin objects. Using alternative upsampling strategies that do not suffer from the same issues as RAFT's upsampling strategy may improve the quality of the estimated flow.

In this section, we attempt to improve results through better flow upsampling, and propose three different strategies to do so. We begin by explaining why we do not choose to follow a pyramid scheme in Section 4.3.1, despite seeming like an obvious solution that can overcome most of the shortcomings of RAFT's upsampling. This stems from RAFT not being easily adaptable into a pyramid scheme, without significantly increasing the computational cost and memory required.
Next, drop-in methods for flow upsampling from previously published work and their applicability to RAFT are discussed in Section 4.3.2. This gives us our first class of upsampling strategies, directly taken from traditional methods.
Following, a modified version of the convex upsampling used by RAFT, that can alleviate some of its issues, is proposed in Section 4.3.3. This gives us our second upsampling strategy candidate.
Finally, a novel flow-based upsampling approach is proposed to replace RAFT's upsampling in Section 4.3.4. This third candidate allows flow values to be adjusted at full resolution, without disproportionately increasing computational cost.

4.3.1 Pyramid Schemes

A prominent strategy for capturing the motion of large objects with reasonable computational effort used by variational approaches and previous methods involving CNNs is a pyramid structure, which infers flow incrementally on coarse to fine scales. Such approaches are somewhat incompatible with all-pairs correlation, which unlike pyramid approaches can capture the motion of small fast-moving objects well, but is associated with very high computational effort at high resolutions. Naively extending RAFT to a pyramid scheme runs into runtime performance and potentially quality issues, as re-using the same invariant update operator across different scales incurs extremely high computation effort at full resolution, due to 64 times as many pixels needing to be processed at that scale. Instead using less complex update operators at higher scales can cause very simple update operators to be unable to properly learn with their constrained expressibility, while still incurring a considerable amount of computational effort. In our prototype experiments with pyramid schemes based on RAFT, we were significantly limited by the computational effort and memory required to use RAFTs components at full resolution and were unable to successfully train a variant with a significantly reduced amount of layers to fit our computational budget, likely due to insufficient complexity of the slimmed-down network.

PRAFlow_RVC [74] is a recent attempt to reconcile the RAFT architecture with pyramid schemes, making concessions on the pyramid itself, and consists of two instances of RAFT with shared weights, one internally inferring flow at $\frac{1}{8}$ resolution as usual and another at $\frac{1}{4}$ resolution. Both instances share the same encoders and the upsampled output of the $\frac{1}{8}$ instance is used as the initial estimate for the $\frac{1}{4}$ instance, finally using its output as the inferred flow. Although this approach shows improved accuracy for small displacements, it significantly increases model complexity and training cost by adding a second instance which operates at higher resolution and performs the final upsampling with RAFTs convex upsampling, just at a slightly higher resolution.

4.3.2 Traditional Approaches

Though simple constant or bilinear interpolation can be used to interpolate optical flow, methods specific to flow interpolation also exist. Flow interpolation has traditionally been required when inferred flow from a lower resolution level needs to be transferred to a higher one in a pyramid scheme, or sparse correspondences, such as from feature descriptor matches, need to be unified into a dense flow field. Traditional approaches, employed in variational pyramid schemes include EpicFlow [59] and RIC [19], which are mainly based on preserving flow discontinuities based on input frame edges and segments.

Some of these methods could directly be used to upsample the low-resolution flow inferred by RAFT. However, the approaches are typically not differentiable, preventing losses from being applied on the upsampling flow. Instead, RAFT can be trained as usual, followed by using the traditional upsampling method at evaluation time after the model has been trained. Another consideration is that optimal upsampling depends on the downsampled representation, which can not easily be learned to fit non-differentiable upsampling strategies.

We propose using RIC, which represents an improvement over EpicFlow, as a possible way to upsample flow at evaluation time. For this, each pixel in the low resolution flow is regarded to encode a correspondence of an 8×8 superpixel at full resolution. The correspondences are interpreted to be at the top-left origin of each superpixel, as this experimentally yielded better results than using the superpixel's center. We do not use variational refinement after the upsampling, due to its computational costs and differing function compared to all other upsampling approaches. Variational refinement or refinement in general could be applied after all possible upsampling methods, but we regard such refinement to be independent and not part of the actual upsampling strategy.

4.3.3 Convex Upsampling

The convex upsampling strategy used by RAFT is already a reasonable starting point for an upsampling scheme, with the benefits mentioned in Section 3.6.4. We propose a modified version of this strategy that addresses two of its shortcomings.

We propose first upsampling the $\frac{1}{8}$ resolution flow field to $\frac{1}{4}$ resolution via constant interpolation, resulting in 2×2 blocks of the same flow value. Next, a learned 2×2 block is added to this flow for each location in the low-resolution image. Finally, convex upsampling is performed as usual with the modified $\frac{1}{4}$ resolution flow.

This modification limits convex upsampling to a factor of 4 instead of 8, allowing the motion of objects twice as thin in each direction to be captured. Additionally, the learned blocks added to the flow are decoupled from the flow used to index the cost volume in each iteration, allowing for more flexibility of the chosen values. Finally, adding blocks to the low-resolution flow rather than using learned blocks directly allows the added blocks to be skipped initially, similar to residual connections. This allows the low-resolution flow to be learned as for the original upsampling, mostly ignoring the learned blocks initially. Afterwards, the learned blocks can add higher-resolution details, while still taking the general motion of the larger neighborhood into account.

4.3.4 Flow-based Upsampling

Finally, we investigate completely replacing RAFT's convex upsampling with a learned approach, rather than just modifying it. For this, we want to perform convolutions at full resolution, to take high-resolution information into account without suffering from the artifacts transposed convolutions introduce. As full-resolution convolutions incur $8 \cdot 8 = 64$ times higher computation and memory costs, we need to limit the complexity of the layers used.

Loosely inspired by UPFlow [43], we replace convex upsampling with flow-based upsampling. This works by bilinearly upsampling low-resolution flow, followed by warping the upsampled flow by a learned full-resolution displacement field. For better efficiency, the low-resolution flow can be sampled with offset grid locations instead.

This operation is similar to convex upsampling, but interpolates between four neighbors instead of nine, and is not limited to the neighbors directly around the location, but can sample values from further away. This means that for thin extensions of larger objects the flow values from the main part of the object can be used, even if they are further away. The upsampling weight count is also reduced from nine to four.

The displacements used for warping are computed by a residual layer that takes in the part of the finest level of the encoder feature pyramid and a bilinearly upsampled version of layers derived from the low-resolution hidden state. By using only few channels and a single residual block, the overhead of the full-resolution convolutions can be kept reasonable, though runtimes are still somewhat higher. This approach allows flow values to be taken from further away and incorporates full-resolution information without introducing artifacts. The concessions due to performance and lack of cost volume computation may still limit the quality of its results, however.

4.4 Data

The typical optical flow datasets available were introduced in Section 2.4 and pose candidate training datasets that we can choose from. We restrict our choice of datasets to those which contain the image aspects of backward flow and occlusions in addition to the input frames and forward flow, to enable us to evaluate and train using these aspects. This rules out the FlyingChairs dataset [9], but allows for the FlyingChairsOcc [23] and FlyingChairs2 [24] datasets to be used, which have essentially the same distribution as the FlyingChairs dataset. We refer to the combination of the FlyingChairsOcc and FlyingChairs2 dataset as chairs. As FlyingThings3D [47], which we also refer to as things, only provides occlusions for a subset of the data, we restrict our use of it to the subset with ground truth occlusions. Unlike RAFT, we do not use the HD1K [32] or KITTI [12, 50] datasets as training data, due to our focus on the Sintel benchmark. We still perform evaluation on the KITTI15 dataset to provide a point of comparison.

4.4.1 Augmentations

Having both forward and backward flow available allows us to swap input frames and other data aspects such as flow and occlusions as an augmentation. Augmentations carried over from RAFT include color, brightness, and saturation changes to both frames, potentially varying between the frames, and spatial flipping of the input frames and corresponding data aspects.

We do not scale input frames and following ScopeFlow [3] use discrete crops to get images to their desired resolution, to limit the change in distribution that would arise from scaling images and continuous crops, as described in Section 2.4.3. The augmentation of cropping introduces additional occlusions, as parts of the frame that contained correspondences are now cut off. To provide accurate occlusions for cropped frames, we adjust occlusion masks by marking all locations with correspondences outside the cropped frame as occluded in addition to the ground truth occlusions from the dataset.

We normalize input frames before passing them into the network, which is done for the same reason activations are often normalized using normalization layers and additionally already introduces simple invariances under global affine value transformations.

4.5 Training

Though we largely follow RAFT's supervised training procedure, we propose using a different loss function for better results on benchmarks. We also adjust the iteration count used during training and tweak the batch size and training iteration steps in order to best make use of our computational budget.

In contrast to RAFT, we use the EPE as our supervision signal, instead of an L1 loss. This means penalizing the L2 norm of the difference between the predicted and ground truth flow, rather than the L1 norm. The EPE metric has the advantage of being rotationally invariant and not favoring any specific directions, unlike the L1 loss that favors the coordinate axes. It is also the metric used to evaluate a model's results for all relevant benchmarks, making it the objective of choice for better evaluation results. One potential concern with using the EPE loss as the objective function is that the derivative of the loss with respect to the horizontal and vertical components of the flow output realized by 2 output channels no longer has constant magnitude and exhibits magnitude changes for small flow differences. Despite this, the EPE loss is an adequate choice that has also already been used by the majority of previous supervised methods estimating optical flow using CNNs, as

described in Section 3.2.
Our complete supervised loss for predicted flow $\tilde{d}$ and ground truth flow d thus reads

$$\mathcal{L}_{\text{supervised}} = \text{EPE}(\tilde{d},\ d) + \lambda_{\text{feature}} \cdot \mathcal{L}_{\text{feature}} + \lambda_{\text{unique}} \cdot \mathcal{L}_{\text{unique}}, \tag{4.5}$$

for parameters λ_{feature} and λ_{unique}.

We apply this loss to the flow predictions after each update iteration and use a weighted sum as the combined loss:

$$\mathcal{L}_{\text{total}} = \frac{1}{\sum_i \gamma^i} \sum_i \gamma^i \cdot \mathcal{L}_{\text{base}}\left(\tilde{d}_i\right), \tag{4.6}$$

for $\gamma > 1$, where $\mathcal{L}_{\text{base}}\left(\tilde{d}_i\right)$ is our supervised loss for predicted flow $\tilde{d}_i$ for each update iteration $i \geq 0$. This allows each incremental update to be individually supervised and prevents parts of the computational graph where gradients are stopped from lacking a supervision objective. The weighted sum places more importance on the output of later iterations and normalizes the loss magnitude to be independent of the number of iterations and the specific choice of γ.

Following RAFT, we use the AdamW optimizer with a small weight decay component, stop flow gradients after each iteration, and employ gradient clipping.

To cope with the constraints of our computational and memory resources, we reduce the number of unrolled update iterations during training from 12 to 10 and reduce the number of channels used in the encoders. These changes make experiments more feasible given our hardware, but reduce the expressive power of the network making direct comparisons with RAFT more difficult, further details are discussed in the evaluation chapter.

Given a fixed computational budget, the required time to train a network is very similar for different configurations of iteration count and batch size, as long as the product of the number of iterations and batch sizes stays the same, given large enough batch sizes and especially when using virtual batch repetition, that is performing gradient descent multiple times and averaging the gradients before performing a single parameter optimization step. Larger batch sizes yield a more accurate estimate of the gradient and can improve the optimization quality significantly, allowing for larger learning rates to be chosen. On the other hand using smaller batch sizes can lead parameters to sharper local minima. This motivates re-evaluating which combination of batch size and iteration count leads to the best learned models for a fixed computation budget or product of batch size and iteration count. We find that training with larger batch sizes for fewer iterations can lead to competitive results, even for smaller computation budget. Our specific training configurations and results leading to our choice can be found in Section 5.2.3.

4.6 Unsupervised Training

Unsupervised training of optical flow estimation models, introduced in Section 3.3, can be performed using only input frames and does not require ground truth optical flow. This allows overcoming the significant issues with optical flow training data, as described in Section 2.4. Specifically, the realism of input frames does not need to be sacrificed in favor of dense ground truth flow fields, and realistic high-quality training data is available in abundance in the form of video recordings. The importance of having large amounts of high-quality training data for CNNs makes unsupervised training even more desirable for CNN-based methods. Further, footage can easily be captured for use cases with different input frame distributions, whereas obtaining the corresponding ground-truth flow is often infeasible. This makes extending the method to applications with slightly different distributions much more tenable with unsupervised training. Finally, unsupervised training allows models to be fit to data at evaluation time, allowing fine-tuning to the specific distribution later evaluated on without the need for ground truth flow. Taking the information from the specific inputs into account can significantly improve the resulting optical flow for those inputs and makes the method more adaptable. This can even be done in combination with supervised training as semi-supervised training, or form the basis of online learning, which will be discussed in Section 4.7.

As we aim to estimate high-quality optical flow for realistic video sequences with RAFT-based architectures, unsupervised training represents a promising approach for the reasons mentioned above. Additionally, unsupervised methods for optical flow can benefit from and be improved by applying RAFT's advancements to unsupervised training. Unsupervised training will also be required for our subsequent online learning approach, which aims to improve results even further.

Thus, we introduce an unsupervised training approach in this section. Unsupervised loss functions represent the main difference to its supervised counterpart and will be the main point of discussion in this section. Such losses typically consist of a photometric loss, a smoothness loss, and a self-supervised loss, as described in Section 3.3. Following previous CNN-based approaches for unsupervised learning of optical flow, we also use these three components in our unsupervised loss. We begin by introducing are photometric and smoothness loss components in Section 4.6.1. Unlike previous methods that use hand-crafted functions, we instead learn these loss functions to benefit from the feature-matching capabilities of CNNs.
Following this, we propose our self-supervised loss component in Section 4.6.2. This allows us to formulate our complete unsupervised loss function in Section 4.6.3.
We end on the note of how we combine this unsupervised loss function with the previous supervised approach to create a semi-supervised approach in Section 4.6.4.

4.6.1 Learned Losses

Photometric and smoothness losses have previously been chosen to be similar to their variational counterparts, with UFlow [29] settling on the census photometric loss and first order smoothness for Sintel or second order smoothness for KITTI. In constrast, we propose learning photometric and smoothness losses, to allow the loss components to benefit from CNNs' strength of feature learning, rather than relying on the invariance of hand-crafted or traditional edge-detection approaches to weigh smoothness penalization.

We employ CNNs to realize learned photometric and smoothness losses, first training the loss functions via a meta-loss, that is a loss for optimizing the learned losses. This pretraining is done in a supervised manner and then followed by unsupervised training of the actual model using the learned loss functions. This arguably makes our approach involving learned loss functions semi-supervised, though frame pairs and corresponding flow fields could easily be generated from single frames, such as described in Section 3.3. Ground truth flow only leaks in form of its derivatives to the learned smoothness loss and indirectly through warped frames to the learned photometric loss. An additional constraint that learned loss functions should follow is that their magnitudes and gradient magnitudes should be consistent across different trained loss functions, so that weights for the loss functions can be chosen independently of the current run.

We realize both learned losses via U-Net architectures. The learned photometric loss passes both inputs frames through a U-Net with shared weights between the frames to infer feature maps for each frame. Both feature maps are then normalized and the feature map for the second frame is then warped by the estimated flow, before taking the L1 distance between the first and the warped second feature map, averaged across the image, as the photometric loss. The normalization ensures consistent loss magnitudes across runs.
The learned smoothness loss takes in the first frame and passes it through a U-Net to obtain location-specific weights for first and second order smoothness. One weight is learned per location for each entry of the gradient of the flow field realizing first order smoothness and each entry of the Hessian of the flow field, for second order smoothness. The gradient and Hessian are approximated using discrete differences and the absolute value of each entry is taken before multiplying with the learned weights. The final weighted sum is taken as the smoothness loss.

Both losses are trained by evaluating the losses for the ground truth flow d and a flow $\tilde{d}$ that is distinct from the ground truth flow, such as the zero flow, or the ground truth flow plus a small random smooth flow field. The base loss components are

$$\mathcal{L}_{\text{real}} = \mathcal{L}_{\text{learned}}(I_0, I_1, d) \tag{4.7}$$

and

$$\mathcal{L}_{\text{fake}} = \mathcal{L}_{\text{learned}}(I_0, I_1, \tilde{d}), \tag{4.8}$$

where $\mathcal{L}_{\text{learned}}$ refers to the learned photometric or learned smoothness loss respectively and I_0 is the first and I_1 the second input frame. The difference between the real, ground-truth flow field, and fake, other flow field, loss is minimized, such that the ground truth flow results in small loss values and the fake flow field in large ones, similar to adversarial losses. Additionally, a small penalty is placed on the loss magnitude for the real flow field, as a tie-breaker in favor of smaller loss values. The complete meta loss with regularization parameter λ_{reg} thus reads

$$\mathcal{L}_{\text{meta}} = (\mathcal{L}_{\text{real}} - \mathcal{L}_{\text{fake}}) + \lambda_{\text{reg}} \cdot \mathcal{L}_{\text{real}} \tag{4.9}$$

for both the learned photometric and learned smoothness loss, using their respective real and fake components for their loss.

We use these learned losses as our photometric and smoothness losses. Following UFlow [29], we limit the photometric loss in unoccluded regions and stop gradients at occlusion masks. Occluded regions do not feature visibly corresponding objects, making the application of the photometric loss in appropriate in occluded regions. Stopping gradients at occlusion masks additionally allows computing occlusions without gradients for better performance and improves overall results, as reported by UFlow. As occlusion ground truth is not available for unsupervised training, we estimate occlusions using the forward-backward consistency check described in Section 2.3.2. The estimated occlusions being inaccurate, especially towards the beginning of training, leads us to weigh down the photometric loss in occluded regions, rather than completely disabling it. This still allows proper training, even with slightly inaccurate estimated occlusions.

4.6.2 Self-Supervision

We use self-supervision based on a teacher model and the consistency of flow for transformed input frames with known transformations. We choose our RAFT-based architecture with an exponential moving average of the model's parameters as our teacher model. This limits the impact of feedback effects and provides a more stable teacher model, while still incorporating parameters from more recent iterations, allowing the teacher to continually improve.

We use our teacher's predictions on the original frames as the first part of our self-supervised loss. Similar to ARFlow [35], we also employ known random geometric transformations that introduce additional occlusions, and use the EPE loss between the inferred flow field for the transformed frames and the adjusted flow field of the teacher model as the self-supervision signal. This adds a good supervision signal for occluded regions, in which the photometric loss can not provide guidance and a naive self-supervised loss without the introduction of occlusions through augmentation would provide a worse supervision signal, due to the occlusions for the teacher. Specifically, we deform the first frame by a random smooth flow field and apply an affine geometric transformation to the second frame.

We generate random smooth flow fields using independent fractal simplex octave noise for each channel of the flow field. The affine transformation is realized via random scalings, translation, and rotations in an interval of parameters. An affine transformation is chosen rather than warping by a flow field for the second frame, as it is easy to invert an affine transformation and compute the corresponding flow field. It is also possible to directly invert a flow field, but this is slightly more involved. The combined flow for the augmented frames can be computed as the combination of the random flow field used to warp the first frame, followed by the teacher's predicted flow from the augmented first to the augmented second frame, followed by the flow for the inverse of the affine transformation applied to the second frame. Two flow fields can be consecutively combined by adding the first flow field d_1 to the second flow field d_2 warped by the first flow field:

$$\text{combine}(d_1,\ d_2) = d_1 + \text{warp}(d_2,\ d_1). \tag{4.10}$$

This is equivalent to following a location in the first flow, then following the second flow from the location prescribed by the first flow.
The resulting combined flow is only valid for locations that do not have occlusions for the first or the second flow. A simple way of convincing oneself of this is to imagine combining a flow field and its inverse, that is the forward flow and the backward flow. The combined flow field should be zero everywhere, but from the forward-backward consistency check we know that this only holds in unoccluded regions. Following an occluded correspondence from the first flow in the second flow leads to following a different object, resulting in an incorrect combined flow. The valid mask for the combined flow can be computed as

$$\text{combine\_valid}(d_1,\ v_1,\ d_2,\ v_2) = v_1 \cdot \text{warp}(v_2,\ d_1), \tag{4.11}$$

given the valid masks v_1 for the first flow and v_2 for the second flow. The individual valid masks should be zero in occluded and invalid regions and one everywhere else. Warping returns zero values for offsets that lie out of frame.

Our self-supervised loss thus consists of two components, based on two signals from the teacher model. The first signal uses the teacher directly on the input frames to obtain the teacher's prediction d_{teacher}. The other first transforms both frames and uses the teacher's prediction on the transformed frames to compute a combined flow $d_{\text{teacher,aug}}$, as described above. We use the EPE loss to penalize deviations from the teachers signal, for the same reason we use it in our supervised loss. The self-supervised loss component reads

$$\mathcal{L}_{\text{self}} = \lambda_{\text{teacher}} \cdot \text{EPE}(\tilde{d},\ d_{\text{teacher}}) + \lambda_{\text{teacher,aug}} \cdot \text{EPE}(\tilde{d},\ d_{\text{teacher,aug}}) \tag{4.12}$$

for a flow prediction $\tilde{d}$ and parameters λ_{teacher} and $\lambda_{\text{teacher,aug}}$. The augmented teacher predictions facilitate guidance for occluded regions, but are not valid everywhere. The teacher prediction on the unaugmented frames is thus used in addition, as it is valid for all locations.

The self-supervision loss is not applied for the first few iterations and only slowly increased to its full magnitude over the course of many iterations. This is done, as the model itself is not a suitable teacher when training first starts and a sudden rather than a smooth introduction of a loss component can lead to undesirable jumps or even divergence.

4.6.3 Unsupervised Loss

Our complete unsupervised loss thus reads

$$\mathcal{L}_{\text{unsupervised}} = \lambda_{\text{photo}} \cdot \mathcal{L}_{\text{photo}} + \lambda_{\text{smooth}} \cdot \mathcal{L}_{\text{smooth}} + \lambda_{\text{self}} \cdot \mathcal{L}_{\text{self}} + \lambda_{\text{unique}} \cdot \mathcal{L}_{\text{unique}}, \tag{4.13}$$

where $\mathcal{L}_{\text{photo}}$ refers to our learned photometric loss, $\mathcal{L}_{\text{smooth}}$ to our learned smoothness loss. Each component is given its own weight parameter λ. It incorporates the previously introduced uniqueness loss, but does not use the feature loss as it requires ground truth optical flow. The unsupervised loss is applied to the predicted flow after each iteration, in exactly the same fashion as is done for the supervised loss.

We employ a variant of boundary dilated warping [44] for the photometric and self-supervised loss. This functions by warping an image with a margin around the augmented cropped image, and cropping the image down to its expected size after warping. It effectively removes occlusions caused by corresponding locations lying outside of the cropped region. This allows the photometric loss to provide guidance in occluded regions and can improve the teacher's flow prediction in the self-supervised component, due to more information being available. We choose to only add small margins rather than using full frames to limit the computation overhead while still allowing for a degree of additional information to be incorporated.

4.6.4 Semi-Supervised Training

We extend our unsupervised approach to a semi-supervised approach by first training on datasets with available ground truth using the supervised approach, followed by training in an unsupervised manner on the dataset that should be evaluated. This allows the model to learn from high-quality ground truth flow and receive better guidance than typically possible with unsupervised losses as well as being able to fit to the dataset used for evaluation without requiring any ground truth with the unsupervised loss. The model trained via supervision provides an excellent starting point for the teacher model of the unsupervised approach. This limits the bias otherwise introduced by the unsupervised loss for a poorly-performing teacher model. Combining the strengths of both supervised and unsupervised approaches in this fashion can improve results beyond what would be possible using only supervised or only unsupervised training.

4.7 Online Learning

Online learning or test-time training is the process of fitting a model to the data it should be evaluated on before the model is used to perform the actual evaluation on those inputs. Alternatively it can also mean to continuously learn from predictions and potential feedback while a model is deployed. We restrict ourselves to the former angle, that is fitting a model to data before evaluating on it. Much like for unsupervised training, there is no ground truth available for the data at this time, but a larger set of potentially related inputs may be available before having to evaluate the model on each of the inputs. For the problem of optical flow, it is common to receive a sequence of consecutive frames all taken from the same video and be given the task to estimate optical flow for each pair of consecutive frames in the sequence, which is the case for many optical flow benchmarks including Sintel. This allows exploiting additional information from other frames in the sequence on top of fitting to a pair of specific input frames when evaluating on each consecutive frame pair.

We aim to improve RAFT-based models using online learning and present our approach to do so in this section. We begin by discussing multi-frame approaches in general in Section 4.7.1. This includes RAFT's warm-start, which is a simple multi-frame approach, and previous approaches that can incorporate information from multiple frames.
Having introduced these previous approaches, we can present our approach to online learning in Section 4.7.2. We describe how our model is trained and online flow predictions are obtained.
Finally, we explain how we combine our online predictions with the pretrained predictions in a fusion approach in Section 4.7.3. This is done to adaptively select the best predictions from both the pretrained and online flow. It allows incorrect predictions both from the pretrained flow that lacks context and the online flow that lacks a direct supervision signal to be mitigated.

4.7.1 Multi-frame Approaches

RAFT itself takes multi-frame information into account at evaluation time with so-called warm-starts by taking the estimated forward flow field of the previous frame pair, warping it by itself, and using the warped flow field as the initial flow field for the next frame pair. Assuming constant projected velocities, this would yield the correct flow at all non-occluded locations. This does not typically yield a dense flow field due to warping and occlusions and is also generally not the correct flow field due to non-constant projected velocities, but still typically yields a better initial estimate than the zero flow field. It should be noted that this approach only takes one additional frame into account and could be improved by using the negative backward flow from the current first frame to the previous frame as initialization instead, as this yields a dense estimate and would still be correct at all locations assuming constant projected velocities. In our modified warm-start we use this negative backward flow instead of the warped previous forward flow.

A previous approach involving CNNs and online learning is ProFlow [46], which learns a small CNN from scratch that predicts the forward flow from the backward flow to the previous frame, using selected samples from initial flow estimates as training data and combines the initial and predicted flow into a final flow prediction along with some post-processing. This approach also only uses one additional frame as input and can be seen as transforming the previous backward flow into the predicted forward flow using a learned transformation, rather than just using the negative previous backward flow as the estimate for initialization as with the previous example, allowing it to take non-constant velocities into account.

Though multiple frames in a sequence can be taken into account during normal training on a dataset with longer sequences, such as FlyingThings3D, by constructing a cost volume spanning multiple frames [27, 80] or adding temporal connections to a model [52, 14], we do not pursue this approach directly due to the added complexity and computational and memory cost. This is also true for a more recent approach [28] that learns to complete missing parts of the cost volume in RAFT, but requires a large amount of memory.

4.7.2 Our Approach

One additional consideration for online learning is that the learned model will only be used for inference on a specific known sequence of frame pairs, making overfitting to that specific sequence or even frame pair a legitimate strategy. In the latter case, one essentially optimizes a single flow field, similar to variational methods, but instead of directly parametrizing the flow field, one optimizes the flow field through a parametrized model, which potentially makes for more well-posed optimization and allows for initial parameters from a pretrained model to be used for better results. This view motivates the introduction of additional inputs to a model, such as the coordinates for each frame position as a grid and the times of the frames in the sequence, effectively removing the translational invariance of convolutional layers, but allowing for specific characteristics of the flow field at certain locations or points in time to be captured by the model. Nevertheless, overfitting might still pose a problem for online training when the distribution being fit to differs from the distribution used for evaluation. This can also be the case when the training data used is low-quality, such as when it originates from bad initial flow estimates.

Though it is possible to follow ProFlow and design a separate model that is trained from scratch for online training, we instead choose to fit our pretrained architecture to specific sequences and frame pairs, motivated by the success of online training of pretrained models using consistency constraints across frame pairs in a video sequence demonstrated in [45]. Although online training could be restricted to certain parts of the network by freezing the parameters for certain components, we allow all parameters to be trained to allow adjustments to be made to all parts of the architecture. As no ground truth is available, a simple candidate for an online loss would be the loss used for unsupervised training, which is directly applicable, but does not take multi-frame information into account.

We perform our online training by fitting a pretrained model to a single sequence using an online loss. For our online loss, we use our photometric and smoothness losses as they appear in our unsupervised loss and replace the self-supervised loss with direct supervision using the pretrained model as a teacher and a multi-frame consistency loss, that takes in three possibly non-consecutive frames and penalizes deviations of the combined flow from the first to the second and the second to the third frame to the flow directly from the first to the third frame. It reads

$$\mathcal{L}_{\text{consistency}} = \text{EPE}(\text{combine}(\tilde{d_1},\ \tilde{d_2}),\ d_{12}), \tag{4.14}$$

where $\tilde{d_1}$ is the output of the online model for the first and second frame and $\tilde{d_2}$ the output for the second and third frame, while d_{12} comes from the pretrained model given the first and third frame.

We select frame triples randomly from the sequence, but ensure that they are either in correct or reverse temporal order and are not more than a fixed number of frames apart, which depends heuristically on the average flow magnitude for the sequence, determined via the flow predictions of the pretrained model. We perform base flow and occlusion estimation at full resolution using the pretrained model, whereas other loss components are calculated on cropped versions of the frames.

Flow-combination is done as for unsupervised training and only holds in transitively non-occluded regions. As the consistency loss does not hold everywhere, we still add teacher supervision through the pretrained model as follows

$$\mathcal{L}_{\text{pretrain}} = \text{EPE}(\tilde{d_1},\ d_1) + \text{EPE}(\tilde{d_2},\ d_2), \tag{4.15}$$

where d_1 and d_2 are the predictions of the teacher model given the first and second, and second and third input frame, respectively.

Although even the consistency loss on its own already allows improving results via online learning, adding pretrained teacher supervision and photometric and smoothness losses can yield further improvement. Our overall online loss is thus defined as

$$\mathcal{L}_{\text{online}} = \lambda_{\text{photo}} \cdot \mathcal{L}_{\text{photo}} + \lambda_{\text{smooth}} \cdot \mathcal{L}_{\text{smooth}} + \lambda_{\text{consistency}} \cdot \mathcal{L}_{\text{consistency}} + \lambda_{\text{pretrain}} \cdot \mathcal{L}_{\text{pretrain}}. \tag{4.16}$$

Due to the high memory requirements of our online approach, we only apply the photometric and smoothness losses for the flow prediction of the final iteration, all other loss components are applied to all iterations' outputs as a weighted sum in the same way as for our supervised and unsupervised loss.

We additionally supply the network with inputs stating the grid positions of the frame crop and time for both frames in sequence. We use the negative backward flow to the previous frame using the pretrained model as initialization for both the pretrained model and the online model.

4.7.3 Flow Fusion

The flow estimated by the online model may yield better results after fitting to the data to be evaluated on, or might be outperformed by the pretrained model, which had better loss guidance and a more balanced dataset to train on. To improve evaluation results as much as possible, we only want to use the flow estimated by the online model if it outperforms the pretrained model. This can either be done by binarily choosing one of the flow fields based on a metric, or combining both flow fields, potentially using the pretrained prediction at some location and the online prediction at others. As ground truth is not available, an alternative metric that approximates it needs to be used for selection. A simple candidate for this could be based on photometric and smoothness losses for the flow field, which are already available. This may however be a poor choice if the same loss functions were already directly used to train the online network.

We propose a flow fusion strategy that combines two or more flow fields into a single prediction, attempting to select for the best prediction available at each location. A cost image is constructed for each input flow that uses a photometric loss, specifically our learned photometric loss, which is smoothed spatially to limit irregular selection of pixels different from their neighborhood, likely signaling outliers. Incorporating smoothness in this manner allows to independently pick the flow with minimal cost at each location, rather than having to perform more costly optimization necessary when smoothness losses are incorporated into the cost directly. In occluded regions all costs other than that of the base flow are penalized, as the photometric loss is likely to not hold in these regions.

After discounting the cost of the base flow estimate by a constant, to break ties in favor of the pretrained flow and only use online flow if it improves cost beyond a certain threshold, the flow with minimal corresponding cost is selected at every location as the fused flow. The overall cost for a flow candidate $\tilde{d}_i$ and input images $I_0,\ I_1$ is

$$F_i = \left(K_\rho * \mathcal{L}_{\text{photo}}(I_0, I_1, \tilde{d}_i)\right) + t_i, \tag{4.17}$$

where K_ρ is a gaussian kernel with standard deviation ρ that is used to spatially smooth the cost volume via the convolution operator $*$. The additive component t_i penalizes occluded regions for flows other than the pretrained flow candidate $\tilde{d}_0$ and discounts $\tilde{d}_0$ by the constant threshold. The fused flow is chosen as the flow candidate with minimal cost F_i for each location, which can be approximated by using a softmin function, which smoothly approximates $\arg\min$, to make the operation differentiable.

This fusion procedure can also be applied to arbitrary flow candidates and can improve results beyond simply selecting one of the candidates. It is however limited by the quality of the photometric loss used, only incorporates smoothness in a limited fashion, and suffers in occluded regions. Nevertheless, it is generally better than constantly selecting either the pretrained or the online flow and thus serves our purpose well enough.

Chapter Outlook

Having proposed many different changes, variants, and extensions to the RAFT method, we want to investigate whether the proposed changes actually represent improvements, which combination of different variants gives the best results, and how our extensions of RAFT for unsupervised learning and online learning fare against previous approaches for these tasks. In the next chapter, we perform an evaluation of our method to investigate and answer these questions by obtaining performance results of our variants on optical flow benchmarks and comparing and interpreting the results amongst themselves and with previous literature.

5 Evaluation

In the previous chapter, we introduced modifications to and extensions of RAFT, with the aim of improving the quality of the estimated flow for realistic video sequences. Many of the proposed changes may seem reasonable and likely to improve results. However, the complexity and opacity of the impact of changes to NN models necessitates implementation of the method and measurement of the method's results to determine whether those changes actually lead to an improvement. Using any of the proposed changes in models or even practical applications is not advisable if they have not been shown to lead to improvements over the original RAFT method and other previously published approaches. We thus perform an evaluation of our methods, to determine to which extent the aim of our work has been met by our presented modifications.

In this chapter, we investigate the efficacy and usefulness of our previously proposed changes. We do so by evaluating the results of an implementation of our method on benchmarks for optical flow estimation. We also attempt to interpret these results and try to determine why certain variants perform better than others.
We begin by detailing our evaluation methodology in Section 5.1. It motivates and explains in concept how we measure the overall result quality of a variant. This ranges from training a model, over evaluating on a dataset, to summarizing the results in a few numerical values that can be used for comparison with other variants.
Next, we go into more detail of our implementation and evaluation in Section 5.1. This includes implementation details and parameter choices, as well as specific details of the training procedure and system configuration used to run our implementation. We include these details to make our results more comprehensible with knowledge of this background and provide specific parameter values to make our results more reproducible.
We close the introductory part of the evaluation by mentioning a number of caveats in Section 5.3. Though we do not believe that these completely undermine or invalidate our results, they should be kept in mind for the rest of the evaluation. This allows to critically examine our evaluation and determine whether the presented interpretations are actually supported by the numerical results, to prevent incorrect conclusions from being drawn.
We move on to present our results and begin with our supervised results and a baseline ablation study in Section 5.4. It focuses on the baseline modifications proposed in Section 4.1 as well as the different choice of supervised loss function. As this is the first time we present actual results of our work, we also add a few notes on our results in general.
Following this, we evaluate our proposed changes to the cost volume in Section 5.5. We investigate the effect of the different possible normalization strategies and choice of matching cost function.
Next, the different possible upsampling strategies are compared in Section 5.6. This spans the traditional upsampling approaches including RIC, as well as learned upsampling strategies in the form of convex and flow-based upsampling.
We then evaluate our unsupervised approach in Section 5.7. This includes comparing our learned losses with traditional photometric and smoothness losses, as well as semi-supervised training.

Our online learning approach and its results are then examined more closely in Section 5.8. We compare how the incorporation of multi-frame information affects results for our modified warm-start and multi-frame consistency-based online training.
Having evaluated all of our individual modifications, we finally compare our methods with the original RAFT method in Section 5.9. We examine the merit of our different categories of changes and look at the overall impact of our combined changes. This will allow us to formulate our conclusions in the next chapter.

5.1 Methodology

Our aim with this work is to create a method for optical flow estimation that infers high-quality optical flow for realistic video sequences. As detailed in the introduction of Chapter 4, we choose the Sintel dataset [6] as a stand-in benchmark to measure. Choosing such a stand-in metric is required to allow quantitative evaluation of different variants against each other. Of the publicly available datasets for optical flow, the Sintel dataset is one of the most realistic and challenging benchmarks, making it a suitable choice. We also include results on KITTI15 [50] as a point of comparison, but do not focus on these results. The KITTI-specific metric Fl-all describes the percentage of locations with EPE larger than 3 or have an EPE error of more than 5%.
The Sintel dataset consists of a number of different sequences. Each sequence consists of 20 to 50 frames that are consecutively sourced from the same scene at constant frame rate, without any drastic visual discontinuities or cuts. The benchmark provides a training split, for which ground truth forward flow is available in addition to the frame sequences, and a test split, which only contains the frames of sequences. As evaluation for the test split is locked off behind a restricted submission system, we evaluate on the training split.
To ensure that our results generalize ion a meaningful way, we do not use Sintel ground truth flow during training in any way. Leaking ground truth flow in such a way could lead to the model merely memorizing previously seen information rather than actually estimating optical flow and make the corresponding results invalid. The sequences frames on the other hand, can optionally be given to the model before evaluation. This would also be possible for the test split and other datasets as well. When inferring optical flow for videos recorded in advance, methods are also able to access this information. For realtime estimation of optical flow, this would instead be limited to examining past frames. Thus, access having access to input frames before evaluation does not make results invalid or limit their ability to generalize.
We use the EPE as the error metric, which also chosen for public results on the Sintel benchmark and was used as an error metric for optical flow throughout this work. We specifically measure the average EPE between the predicted and ground truth flow of across all forward frame pairs of the entire dataset. All reported EPE figures are rounded to two decimal places, unless explicitly stated otherwise. We train models from scratch and report evaluation results either after every individual step of the training schedule or only once after all steps of the training schedule have finished, for brevity. We use 32 update iterations during evaluation. All results use our modified warm-start described in Section 4.7.1, unless otherwise noted. We will use the shortcut C for the chairs datasets, T for things, and S for Sintel. The combination $\mathrm{C} + \mathrm{T}$ denotes first training on chairs, followed by training on things, and similarly for other combinations. Two consecutive entries for the same variant, such as C and $\mathrm{C} + \mathrm{T}$ denote that the parameters from the training run on chairs were used to initialize the parameters for the subsequent things training, rather than runs being independent.

We have to perform all training and evaluation runs within our limited computational budget. Iterating on designs and performing evaluation for many different variants requires performing many training runs. To fit these runs into our computational budget, we have to limit the amount of time a single run takes. We do so by limiting our model complexity and training complexity. This means that our results are no longer directly comparable to RAFT, but this allows us to perform iteration and sufficient evaluation in the first place, which we prioritize.

5.2 Configuration Details

We now give more details about the specific implementation of our method. We include this information to aid reproduction of our results and give more insight into how the results were obtained. This includes technical details and the more specific details of the architecture, as well as parameter choices previously only mentioned symbolically. We also go over how our training is performed, including details on the underlying hardware. We end by presenting our most commonly used training schedule, used to obtain the majority of our results, and explain why we settled on this specific schedule.

5.2.1 Implementation Details

We implement our approach using the PyTorch framework [57].

For the forward-backward consistency check, we regard locations as fully occluded for distance inconsistencies greater than 10 and fully unoccluded for a discrepancy of 0. When using non-binary occlusions, we linearly scale this relationship. This means that a distance of 5 is associated with an occlusion value of 0.5, meaning the location is “half occluded”. When only applying loss components in non-occluded regions we do so by multiplying the loss with $(1 - o)$ for the occlusion mask o at each location, and divide the average result by the average value of the mask we multiplied by. The same is done for valid masks.

We remove variational refinement from RIC, and otherwise leave all of its parameters at their default values.

5.2.1.1 Architecture

Our convolution block consists of symmetric reflection padding, with padding of size $\left\lfloor \frac{k}{2} \right\rfloor$ at all four image edges for uneven kernel size k. Our default kernel size is 3. This is followed by a convolution, which is optionally strided with stride 2 to achieve downsampling. Biases are dropped if the block contains a normalization. This is followed by an optional activation, which is set to be a LeakyReLU activation by default and finally an optional normalization layer. We use instance normalization as the normalization layer in all instances other than our feature normalization and neighborhood normalization. The latter utilize group normalization with a single group for all channels.

Our residual blocks consist of two convolution blocks and an optional additional convolution. The first convolution block takes in the input channels and outputs an intermediary amount of channels. The intermediary amount of channels is chosen to be the minimum of the input and output channel count by default. It is followed by the second convolution block, which takes in the intermediary channels and outputs the preliminary output channels. It optionally applied a downsampling factor of two and does not contain an activation or a normalization layer. The original input is used as the skip input, if the input and output layer count are identical and no downsampling is performed. Otherwise, a bias-less and optionally strided convolution is applied to the input to obtain the skip input. The skip input is added to the preliminary output, after which a final activation and normalization layer are applied. This yields the output of the residual block.

All U-Net architectures have configurable depth and apply begin by applying a single convolution block with kernel size 1 to obtain an input with the desired number of channels. It then applies a configurable number of residual blocks with kernel size 3 on the narrowing encoder part, by default this is only a single residual block. The first residual blocks performs downsampling by factor 2. The skip input is taken as the output of the last output of a convolution block for each possible resolution. The decoder part bilinearly upsamples the previous output by a factor of 2, concatenates the corresponding skip input, and passes the result through a convolution block with kernel size 3 on each level. The final convolution block does not contain an activation function or normalization layer, to allow for arbitrary output distributions.

Our feature and context encoders are designed in the same way as the contracting part of the U-Net, using the lower-resolution output as their final output. The skip inputs at each resolution are optionally accessible, which is later used for our flow-based upsampling. We use only one residual block per level by default, and use a kernel size of 3. We use channel counts $\max\left(\frac{c}{4^{d-i}},\ 8\right)$ at level i, for total depth d and final output channel count c. Following RAFT, we use $c = 256$, though the resulting channel counts on the other levels differ.

We replace all ReLU activation functions with LeakyReLU. We add a skip connection to the ConvGRU by passing its input through a convolution with kernel size 1 to adjust layer counts and add this to the final output. We add a convolution that skips over the two consecutive flow convolutions in the motion encoder. We replace the flow decoder with a residual block without norms or final activation.

We use our L1-based cost function for cost volume computation, unless otherwise specified. We employ both feature and neighborhood normalization, unless otherwise specified.

Our modified convex upsampling adds eight output channels to the original mask block to obtain 2-dimensional flow values for each 2×2 block. Unlike RAFT, we do not multiply the mask output by a factor. Our flow-based upsampling instead utilizes a residual block with kernel size 5 without normalization that outputs 4 channels. These channels are then bilinearly upsampled to full resolution and concatenated with the first four full-resolution output channels of the feature encoder. This is then passed through another residual layer with kernel size 9 without norms or final activation and the output is used as the flow displacements, used to sample the original upsampled flow. We utilize our modified convex upsampling, unless otherwise specified.

Our GOCor-inspired transformation first concatenates both feature maps and reshapes them to consist of two channels and treat every feature dimension the same. The first feature channel corresponds to a channel in the first feature map and the second in the second for each dimension. We pass this through a residual block with kernel size 5, 2 input, 4 intermediary and 2 output channels. After reverting the reshaping, the features are further processed as usual.

Our learned losses are both based on U-Nets with depth 2. The learned photometric loss uses 2^{i+2} channels on level i and 8 output channels, whereas the learned smoothness loss uses 2^{i+1} channels on level i and 10 output channels. The learned weights for the smoothness loss are obtained by applying the function $x \mapsto e^{-|x|}$ to each output channel.

We disable photometric augmentations in the form of hue, saturation, and brightness changes for unsupervised training, to make the distribution of the images from the training data more in line with that of the test set. Photometric augmentations are still used when training our learned losses on things.

Components otherwise unmentioned in this work are consistent with the original RAFT implementation.

5.2.1.2 Parameters

We choose our loss parameters as follows: $\lambda_{\text{feature}} = \lambda_{\text{unique}} = 0.1$, $\lambda_{\text{reg}} = 0.001$ $\lambda_{\text{teacher}} = 0.2$, $\lambda_{\text{teacher, aug}} = 0.8$, $\lambda_{\text{photo}} = \lambda_{\text{smooth}} = 1$, $\lambda_{\text{consistency}} = 1$, and $\lambda_{\text{pretrain}} = 0.1$. We scale λ_{self} linearly from 0.0 to 1.0 over the first 10k iterations after the first 1k iterations. Before that, we linearly scale it from 0.0 to 1.0 over the first 100k iterations after the first 100 iterations. If these two schedules are conflicting, we take the maximum, and limit it to the range $[0, 1]$. We use $\gamma = 1.2$ for our online loss, and $\gamma = 1.05$ otherwise.

For AdamW, we use a weight decay of 10^{-4}. We use a learning rate of $8 \cdot 10^{-4}$ for training on chairs, and a learning $2 \cdot 10^{-4}$ for finetuning on things and Sintel. These learning rates are approximately four times larger than those used in RAFT, allowing for faster learning in combination with the larger batch size, similar to [64].

5.2.2 System Configurations

We use two different systems for our experiments which we will refer to as `H1` and `H2`. The `H1` system features a single NVIDIA GeForce RTX 2080 Ti GPU, while `H2` uses a single NVIDIA GeForce RTX 3090 GPU. Both machines are only bottlenecked by GPU performance and GPU memory, making the specific CPU and memory details less relevant, as they more than keep up with the GPU's computations for both machines. Due to the specific configuration of `H2`, the PyTorch JIT had to be disabled when running on it, impacting both runtimes and memory usage negatively and generally making training times between the systems incomparable, even if the hardware differences were to be overlooked. We use `H1` for most of our runs and use `H2` predominately for unsupervised training, which has higher memory requirements and benefits from the additional GPU memory.

5.2.3 Training Configurations

We use a learning rate of $8 \cdot 10^{-4}$ when training on chairs and $2 \cdot 10^{-4}$ on things and Sintel, unless otherwise specified. We use a separate OneCycleLR learning rate schedule on each dataset, which has a factor of 100 between the minimum and maximum learning rate and reaches its maximum after 20% of the total dataset iterations.

Our basic training schedule is as follows: We train on chairs with an image size of 256×256 and a batch size of 24 for 10k iterations, followed by training on things with an image size of 384×384 and a batch size of 6 for 10k iterations. These are chosen such that training with half the batch size and the given crop size is possible within `H1`'s GPU memory, and batches are virtually repeated twice. We use this training schedule, unless explicitly stated otherwise.

As discussed in 4.5, we choose the batch size and iteration count by comparing the results and training times for different iteration counts and batch sizes. A set of different configurations and their results can be seen in Table 5.1. We settle on our basic schedule by picking an option that has low training time, but still gives good results.

Iteration count	Batch size factor	Training duration	Sintel train EPE	
			Clean	Final
10k	1	3.6h	2.32	3.42
10k	2	6.8h	2.08	3.32
10k	4	13.1h	2.05	3.22
20k	2	13.3h	1.99	3.30

Table 5.1: Overview of results for different batch sizes and iteration counts. Models are trained using our basic training schedule, the base batch size is 12 for chairs and 3 for things. The used batch size is calculated as the base batch size multiplied by the batch size factor. The same noted iteration count is used for both training on chairs an things. This means that our chosen training configuration corresponds to 10k iterations and a batch size factor of 2. Results are obtained after training on both chairs and things and the total training time is noted as the training duration. All runs were performed on `H1`.

For unsupervised and semi-supervised training, we additionally train on Sintel with an image size of 256×218, which is exactly a quarter of the frame size, and a batch size of 12. For unsupervised training, we reduce the batch size when training on chairs from 24 to 12 in order to keep training times reasonable, as unsupervised training requires longer training times. We only train on input frames from the Sintel train split for all of our methods, though results could potentially be improved by incorporating frames from the Sintel test split when training without ground truth flow labels. However, this would change the setting and invalidate our results on the Sintel train split to a certain extent. We do not choose to do so, as a result.

Pretraining is done using the things dataset with an image size of 512×512 and a batch size of 24 for 5k iterations. This is again chosen to fit into GPU memory. We use a learning rate of $8 \cdot 10^{-4}$ for the photometric loss and $8 \cdot 10^{-3}$ for our smoothness loss, with our typical OneCycleLR learning rate schedule. When evaluating on the Sintel test split, one could train using the Sintel train dataset instead, likely allowing for better results to be achieved.

Online training is done on the Sintel dataset with an image size of 256×218 and a batch size of 12 for 300 iterations per sequence. We use a learning rate of $5 \cdot 10^{-4}$ and follow our typical OneCycleLR learning rate schedule.

5.3 Caveats

Note that in most cases, only the result of a single training run is reported. As weight initialization, data sampling and augmentations, and other details including non-deterministic implementation of network layers all introduce non-determinism into training, there can be some variance of results between different training runs. This means that small differences in results between different variants may be the result of stochastic fluctuations between runs rather than actual improvements caused by the modifications of the variant. Ideally one would repeat the same training run many times in order to get a more accurate representation of the expected results and be able to estimate the magnitude of fluctuations between runs. We were not able to do so in all cases, due to our computational and time budget, necessitating more scrutiny to be applied when comparing similar results for different variants. For variants that we were able to perform more training runs of, we find that the run-to-run variance for Sintel results is typically below 0.2 EPE, often below 0.1 EPE or even lower. This is measured after completing the training on both chairs and things, but is not dissimilar, though sometimes more pronounced, for training runs on chairs only.

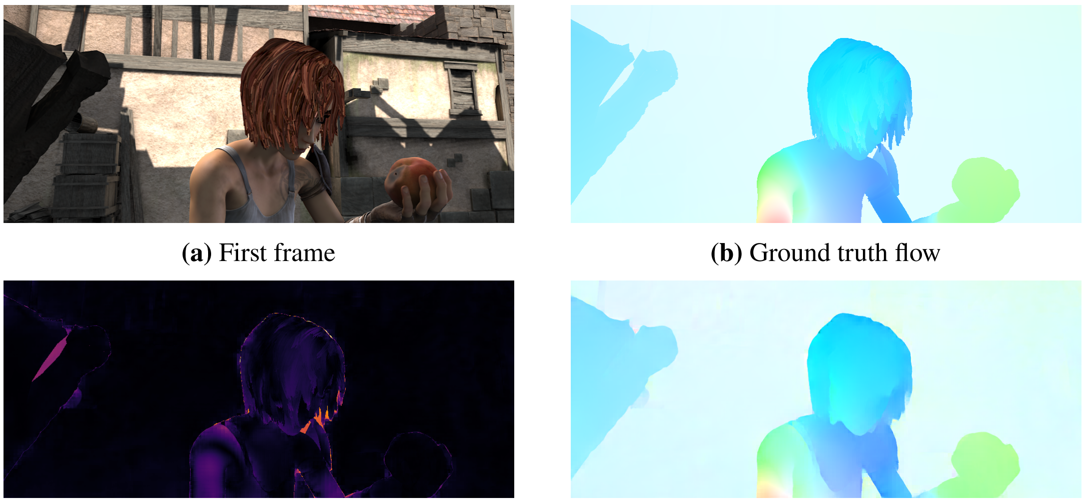
Though some effort was put into adjusting the overall learning rates, not all hyperparameters were tuned, and hyperparameters were not tuned for specific variants. This means that results should fall short of those achievable for models with proper hyperparameter tuning and certain variants might happen to benefit more from the specific hyperparameters than others, which may lead to skewed results between different variants.

Our changed approach to training and evaluation, mainly aimed at reducing the computational effort required for a single training run, also prevents a large number of runs from being directly comparable with RAFT's original results or other results in previous literature. Our results are still internally comparable amongst themselves. We instead reduce the computational effort used to train RAFT, to make it more comparable with the computational budget we use. Comparing the results of this more limited version of RAFT against our results can help compare our method to RAFT when using a fixed computational budget. Though it seems reasonable that improved results should scale given a larger computation budget, this is not guaranteed or verified experimentally. We go into more detail on this in our overall comparison with RAFT in Section 5.9.

We evaluate on Sintel train without ever using Sintel ground truth flow during training. For supervised training, we also do not use Sintel input frames, meaning that we use no data from the Sintel dataset during training. For unsupervised, semi-supervised, and online training, we additionally use Sintel input frames during training. We believe this to be valid, as even in real-world unsupervised settings, input frames are available for use. When performing optical flow estimation in real time, this is only true for past frames. It should be noted that we evaluated on Sintel when iterating on our proposed modifications. Though our learning rate is unaffected, design choices such as the layer count, training configurations, and some modifications themselves were decided on after observing their impact on Sintel train evaluations. This means that through our iteration, we may have inadvertently overfit to the Sintel train dataset, potentially leading to our results not generalizing well to other datasets. We do not believe that this is too much of a source for concern, as we only tested few, reasonable configurations for each decision, and it is difficult to completely overfit on a dataset as complex as Sintel by choosing from only tens of possible configurations. Nevertheless, this should be kept in mind when looking at our Sintel train results. Our evaluation results on KITTI are legitimate, as we do not use any KITTI data during training and did not extensively evaluate on KITTI during iteration.

5.4 Baseline Ablation Study

We begin by examining our supervised results. Visual results of our supervised approach can be found in Figure 5.1. In addition to showing numeric results for our supervised approach, we also investigate whether our baseline changes actually yield improved results. We do so by performing an ablation study, which replaces or removes individual components and observes the effect on this change on results. If removing a component leads to noticeably worse results, we conclude that the component was important for the model to achieve good results. This allows us to estimate the importance and effect of individual components and can be used to judge whether these individual changes are worthwhile. As the number of training runs we can perform is limited, we focus on ablation variants for changes with unpredictable or presumably large impact. In particular, we do not perform individual ablation studies for seemingly straight-forward and low-impact changes, such as the change of activation functions or the feature and uniqueness loss components. The combined effect of all of our modifications can still be seen when comparing the results of our full method to RAFT.



(a) First frame

(b) Ground truth flow

(c) EPE visualization

(d) Predicted flow

Figure 5.1: Sample visual results for the Sintel dataset. The results come from a model trained on chairs and things using our basic training configuration. The maximum flow magnitude is kept consistent across visualizations. In the EPE visualization, black corresponds to an EPE of 0 and bright yellow to an EPE of 10. The values in between vary linearly with lightness.

Table 5.2 shows the results of an ablation study concerning our proposed baseline modifications proposed in Section 4.1 and choice of supervised loss function. We specifically compare the results of our full supervised approach with a variant that drop the or skip connections, as well as a variant that replaces the EPE loss component of our supervised loss with an L1 loss. The original RAFT implementation is used to obtain results with our used batch size, image size, and iteration count. We use RAFT's original learning rate for its run. RAFT's results are reported without its warm-start, as it makes results worse in this case. Note that RAFT's warm-start makes results

Variant	Training schedule	Sintel train EPE		KITTI15 train		Parameter Count
		Clean	Final	EPE	Fl-all	
RAFT (2-view)	C	2.95	4.01	12.97	36.88%	5.3M
	C + T	2.57	3.74	**10.62**	**33.15%**	
Ours (supervised)	C	2.54	3.79	12.73	40.39%	3.7M
	C + T	2.03	**3.33**	11.63	38.85%	
Ours w/o update skip connections	C	2.96	4.16	14.45	46.57%	3.6M
	C + T	2.62	3.86	15.10	48.29%	
Ours w/ L1 loss	C	2.44	3.61	11.89	34.70%	3.7M
	C + T	**2.02**	3.34	10.79	34.22%	

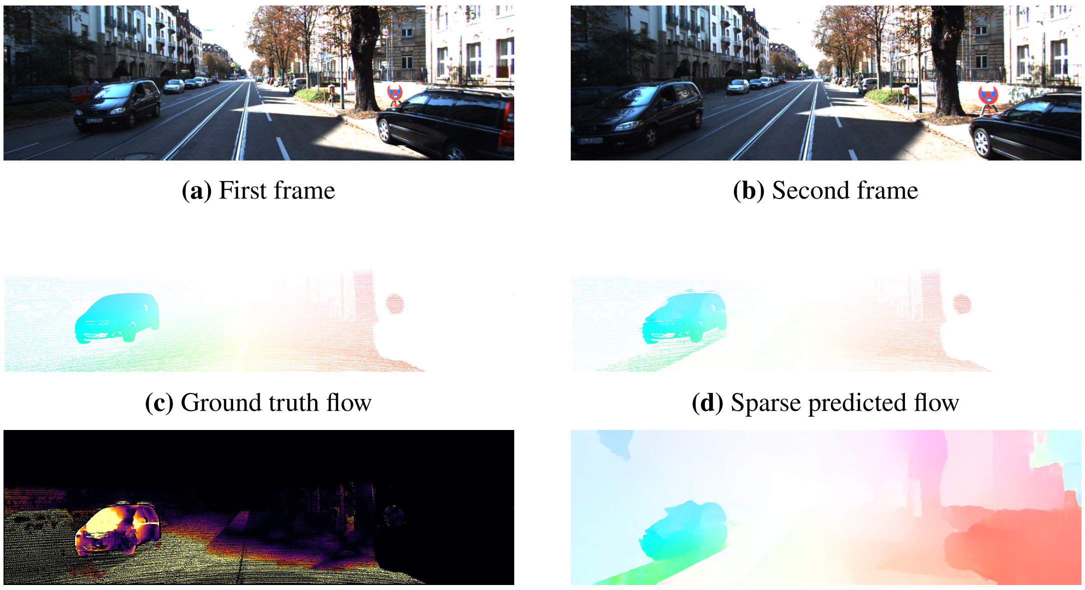
Table 5.2: Results of the baseline ablation study. Bold entries mark the best result in each column. We abbreviate with as w/ and without as w/o. The original implementation and learning rate is used for RAFT, otherwise all runs follow our basic training configuration. All models were trained with supervised loss functions and all runs were performed on `H1`.

slightly worse even for the original results on the Sintel train split. An evaluation of the different warm-start procedures can be found in Section 5.8. By obtaining results on RAFT using our training configuration, we can more easily compare the original method to our approach. Note that RAFT still uses more update iterations, parameters, possibly better augmentation, and more computational power overall, meaning that results are not completely comparable, but as comparable as they can be for the original RAFT method.

For the introduced skip connections, one can observe that the EPE increase when the modification is removed. This suggests that the usage of these extra skip connections noticeably improve results. This is in line with our aim and expectations when proposing extra skip connections in, based on the findings of [16].

When comparing the choice of supervised loss, we notice that results for both the EPE loss and L1 loss lie fairly closely together. The L1 loss seems to slightly outperform the EPE loss, which is especially noticeable for the KITTI results. Though this does not definitively show that the L1 loss is a superior choice in this case, due to the difference between the results falling well within the margin of error, we believe that this is likely to be the case. This would explain why RAFT chose the L1 loss over the EPE loss that previous methods went with. A possible explanation for this is that the gradients of the L1 loss function are more suitable for the chosen parametrization of the optical flow field. The flow field output is parametrized as the vector component in x-direction and the vector component in y-direction, each represented by an individual output channel. The L1 loss directly and independently penalizes deviations of these components between the predicted and ground truth flow. It also leads to constant gradient magnitudes, regardless of the magnitude of the difference between flows. For the EPE loss on the other hand, the penalization of each individual channel depends on the value of the other channel, as do gradient magnitudes. As the EPE loss is rotationally invariant and used as the metric to evaluate on benchmarks, this is still surprising, but could reasonably explain the difference in results. Nevertheless, both losses seem viable and can achieve good results, meaning that the EPE loss is still a viable supervision signal choice.

The comparison with the original RAFT method shows that our supervised results on Sintel are noticeably better when using the training configuration that fits our computational budget. This is in spite of the RAFT model being more complex and actually more computationally expensive even in this configuration. We also note that our results are better than the reported results for the RAFT small configuration. The computational effort required to obtain our results is smaller than that for RAFT small, but our model is still more complex and the small configuration does not appear to have been the main focus of RAFT. We go into a more detailed comparison between the original method and our approach in Section 5.9.



(a) First frame

(b) Second frame

(c) Ground truth flow

(d) Sparse predicted flow

(e) EPE visualization

(f) Dense predicted flow

Figure 5.2: Sample visual results for the KITTI15 dataset. Note how the reflections on the windshield of the moving car to the left cause the model to incorrectly match locations with similar brightness. The results come from a model trained on chairs and things using our basic training configuration. The maximum flow magnitude is kept consistent across visualizations. In the EPE visualization, black corresponds to an EPE of 0 and bright yellow to an EPE of 20. The values in between vary linearly with lightness. As the EPE is clamped to be at most 20, bright yellow areas may also correspond to larger EPE values.

5.4.1 A Note on KITTI Results

Although we did not intend to focus on KITTI results, we find the achieved results very surprising. They indicate very poor performance on KITTI data, compared to the respectable performance on Sintel. The Fl-all metric in particular does not match up with what would typically be expected. As Sintel results are good without having used any Sintel training data, and training losses on chairs and things are reasonable, we do not believe that this is caused by catastrophic overfitting on Sintel. We suspect that this may be caused by the difference in distributions between KITTI and Sintel. RAFT reports significantly better results on KITTI, but our results for RAFT using our configuration are similar to those we obtained. As the results were obtained with RAFT's original implementation and evaluation code, we can be fairly certain that these results are not caused by an issue in our implementation. This suggests that the difference in results is caused by our modified training configuration itself, rather than being caused by our proposed modifications. Visual results on the KITTI15 dataset can be found in Figure 5.2.

Upon further inspection, we notice that most of the error is contributed by locations near the edges of the frame, which are occluded due to moving out of frame. The motion in this region is mostly affine and follows a similar pattern across most frames, and is caused by the egomotion of driving forward. A specific example of this is the overall EPE value of 11.63 being made up by an EPE of 32.91 in the region with out of frame correspondences and 6.26 everywhere else. It may be that this type of motion very much lies out of distribution for our model, causing these unexpected results.

5.5 Cost Volume Processing

We move on to evaluating our proposed changes to the cost volume calculation and processing. We compare standard correlation with our proposed L1-based cost volume and perform an ablation study for the proposed cost volume normalization.

The results in Table 5.3 suggest that both correlation and the L1-based cost are feasible choices for cost functions. The results for correlation seem to be slightly better, but overall results are similar and the differences fall into the margin of error. This means that L1-based cost volume computation on its own does not appear to yield a significant improvement over the previously used correlation. One way this could change is if the more direct relationship between feature maps could be exploited elsewhere. Currently, this makes the viability of an L1-based cost volume an interesting discovery, but it does not actually seem practically useful. This would suggest that correlation-based cost volumes should continued to be used instead.

The ablation study for cost volume normalization shows more noticeable changes. It indicates that using both normalization types significantly improves results over using no such normalization. To break the effect down further into the individual contributions from the feature normalization and neighborhood normalization, we also run configurations that only remove one of the normalization types. Unfortunately, training becomes unstable when removing only neighborhood normalization. This is likely due to the exploding gradients problem manifesting upon removal of the normalization. We thus do not report results with only neighborhood normalization being removed. Results without feature normalization are not too far from results with both normalization types, meaning that its impact must not be that large. We conclude that using our normalization strategies yields improvements and that the majority of these improvements is contributed by our neighborhood

Variant	Training schedule	Sintel train EPE		KITTI15 train	
		Clean	Final	EPE	Fl-all
Ours w/ L1-based cost	C	2.54	3.79	12.73	40.39%
	C + T	2.03	**3.33**	11.63	38.85%
Ours w/ correlation	C	2.48	3.63	11.68	34.29%
	C + T	**2.01**	3.38	**10.57**	**34.00%**
Ours w/o normalization	C	3.20	4.13	16.23	41.58%
	C + T	2.50	3.51	11.72	40.99%
Ours w/o feature normalization	C	2.58	3.76	12.96	37.21%
	C + T	2.04	3.41	11.21	35.29%

Table 5.3: Results of an ablation study for different choices of cost volume processing. Bold entries mark the best result in each column. We abbreviate with as w/ and without as w/o. Ours w/o normalization refers to removing both feature and neighborhood normalization from the model. All models were trained using our supervised approach on `H1`.

normalization. Though feature normalization does not seem harmful and has sound theoretical motivation, we do not observe a large benefit from using it. Neighborhood normalization should thus be employed to improve results, and feature normalization can optionally be used, if the motivations behind it seem attractive.

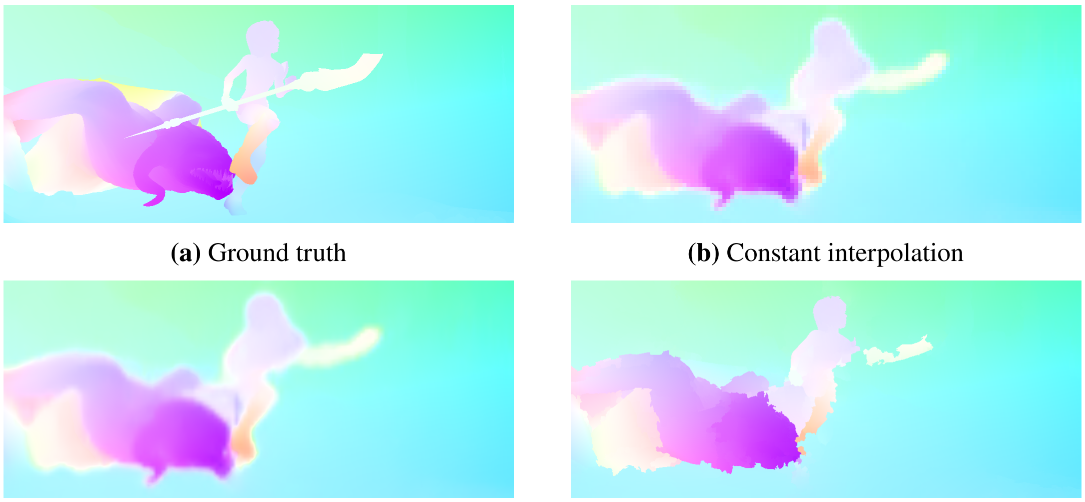
5.6 Upsampling

Next, we compare different possible upsampling strategies for RAFT-based architectures, which were mentioned in Section 4.3.

Variant	Sintel train EPE	
	Clean	Final
Ours w/ constant interpolation	2.38	3.53
Ours w/ bilinear interpolation	2.41	3.56
Ours w/ RIC upsampling	**2.37**	**3.51**

Table 5.4: Results for different traditional upsampling strategies. All results were obtained using the same low-resolution flow inputs from a model trained on chairs and things in a supervised manner. We apply RIC without subsequent variational refinement. Bold entries mark the best result in each column. We abbreviate with as w/.

Beginning with traditional upsampling approaches, the results in Table 5.4 show that RIC [19] yields an improvement over simple constant or bilinear interpolation, even without variational refinement. The results obtained with RIC also have sharper contours, as can be seen in Figure 5.3. However, the results for traditional upsampling approaches are very close and noticeably fall short of our baseline learned upsampling approach.



(a) Ground truth **(b)** Constant interpolation

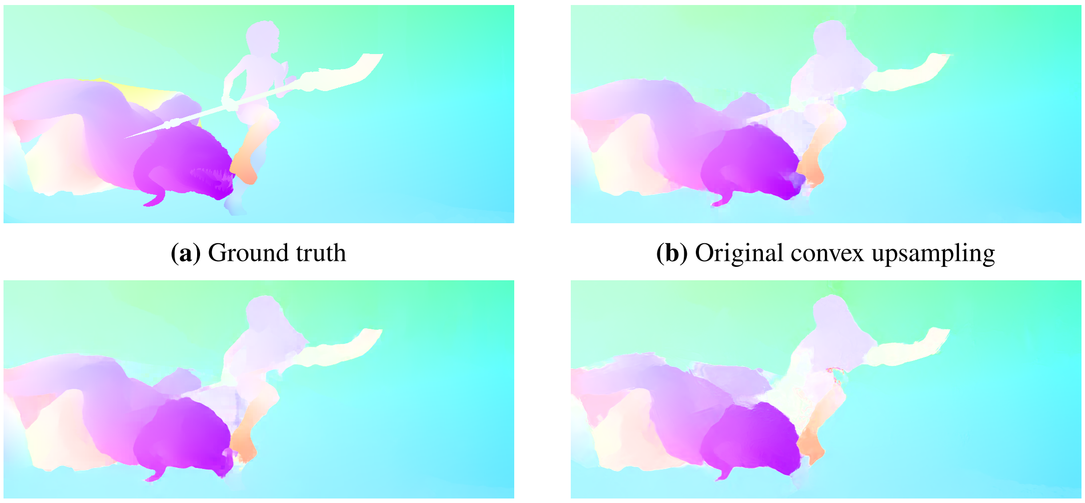
(c) Bilinear interpolation **(d)** RIC upsampling

Figure 5.3: Sample visualization of flow fields obtained for the different traditional upsampling strategies. All visualizations use the same low-resolution flow input and maximum flow magnitude for visualization. The scene is taken from the Sintel clean dataset.

Variant	Training schedule	Sintel train EPE		KITTI15 train		Parameter Count
		Clean	Final	EPE	Fl-all	
Ours w/ RAFT's upsampling	C	2.73	3.87	14.08	42.66%	3.7M
	C + T	2.19	3.34	13.29	40.73%	
Ours w/ modified convex upsampling	C	2.54	3.79	12.73	40.39%	3.7M
	C + T	**2.03**	**3.33**	**11.63**	**38.85%**	
Ours w/ flow-based upsampling	C	2.72	3.65	11.43	37.20%	3.4M
	C + T	2.20	3.51	12.62	39.14%	

Table 5.5: Results for different choices of learned upsampling strategies. The models were trained in a supervised manner. We use bold entries to highlight the best result in each column and abbreviate with as w/. All runs were performed on `H1`.

Table 5.5 shows the results for the different learned upsampling approaches. This compares RAFT's original convex upsampling and our modified version with the proposed flow-based upsampling. Though the proposed flow-based upsampling reduces the model's parameter count, the full-resolution convolutions lead to a higher overall computational efforts. Its obtained results are mostly on par with RAFT's original upsampling, meaning that it comes at a higher cost without yielding the desired benefits. The visual results in Figure 5.4 show that flow-based upsampling does not lead to block artifacts appearing, whereas block patterns are visible for both the original and modified convex upsampling. However, the flow-based upsampling strategy still produces artifacts of its own. As a result, we do not believe flow-based upsampling should be used as-is. Our modification of RAFT's convex upsampling on the other hand leads to improved results at negligible cost, even if only by a smaller margin.



(a) Ground truth

(b) Original convex upsampling

(c) Modified convex upsampling

(d) Flow-based upsampling

Figure 5.4: Sample visualization of flow fields obtained for the different learned upsampling strategies. The results are taken from a different training run for each strategy, to allow for model differences. The results were obtained after training on both chairs and things using our basic training configuration. All visualizations use the same maximum flow magnitude for visualization. The scene is taken from the Sintel clean dataset.

All learned approaches beat every traditional approach, making them the more attractive choice. Among learned approaches, our modified convex upsampling achieves the best results, making it the upsampling strategy for choice from the strategies we compared.

5.7 Unupervised Results

We next evaluate our unsupervised approach, introduced in Section 4.6. We investigate the effectiveness of our learned losses and compare our results with those of previous unsupervised approaches.

Table 5.6 shows the results for our unsupervised approaches with different loss functions and UFlow [29], which is a previous unsupervised method for optical flow achieving state-of-the-art results. We specifically compare against UFlow, as this provides a direct point of comparison with its suggested traditional loss functions. Though unsupervised methods with slightly better results exist, these generally use more Sintel data, from the evaluation set or even from the original movie,

Variant	Training schedule	Sintel train EPE		KITTI15 train	
		Clean	Final	EPE	Fl-all
Ours w/ traditional losses	C	5.98	6.65	17.06	37.61%
	C + T	4.66	5.39	14.24	38.38%
	C + T + S	4.24	5.05	10.73	33.46%
Ours w/ traditional smoothness and learned photometric loss	C	4.54	5.46	16.80	37.50%
	C + T	3.73	4.76	14.60	38.35%
	C + T + S	3.32	4.23	11.21	32.03%
Ours w/ learned losses	C	4.64	5.55	18.85	43.73%
	C + T	3.61	4.68	15.22	43.17%
	C + T + S	3.24	4.11	13.05	37.44%
UFlow [29] (reported)	C + S	**3.01**	**4.09**	**2.84**	**9.39%**

Table 5.6: Results for unsupervised training. We use our previously described unsupervised training schedule. Note that models were trained on the Sintel train input frames. Traditional losses refer to the census loss for the photometric loss and edge-aware first order smoothness for the smoothness loss. We highlight the best result in each column in bold face and abbreviate with as w/. Our runs were performed on H2.

making the comparison with our results more difficult. UFlow's results are not that far behind these methods' results and are still better than our results, making our comparison with UFlow results as a stand-in for state-of-the-art approaches valid. We first note that our approach is successful in using RAFT-based architectures for unsupervised optical flow estimation. Our best unsupervised results on the Sintel benchmark are not too far from state-of-the-art unsupervised results, despite our training configuration being significantly more limited due to our computational budget. That being said, one would expect that the advantages of the RAFT architecture could be used to improve unsupervised results. We believe that this may still be achievable using configurations more similar to those of the original RAFT.

For loss functions, we compare our learned losses with the traditional losses found to be a good choice for the Sintel dataset by UFlow. This is the census loss [49] for the photometric loss and edge-aware first order smoothness for the smoothness loss. The Sintel dataset results show that our learned losses outperform traditional losses by a noticeable margin. The difference in KITTI results is likely caused by the traditional smoothness function dealing better with the flow near frame edges. The learned smoothness loss was optimized to fit the FlyingThings3D dataset, which differs significantly from the KITTI datasets. To break down the contribution of the individual losses to the overall results, we also perform a run with our learned photometric loss and a traditional smoothness loss. As this run has somewhat similar, but slightly worse results on Sintel after training is completed, we conclude that both of our learned losses individually outperform their traditional counterparts. It does seem that that our learned photometric is a more substantial upgrade over its traditional counterpart, whereas the learned smoothness loss represents an improvement, but only a small one. We conclude our learned losses are effective and can outperform traditional losses. These loss functions are applicable for unsupervised methods for optical flow in general and are not limited to our specific approach.

5.7.1 Semi-Supervised Results

It is unsurprising that our unsupervised results are worse than our supervised results, due to the poorer guidance. We next investigate whether our proposed semi-supervised approach can lead to an overall improvement over supervised result.

Variant	Training schedule	Sintel train EPE		KITTI15 train	
		Clean	Final	EPE	Fl-all
Ours (supervised)	C	2.52	3.57	12.37	38.64%
	C + T	**2.09**	3.36	11.79	37.03%
Ours (semi-supervised)	C + T + S	2.26	3.27	10.73	25.41%
Ours (semi-supervised, 2x batch size)	C + T + S	2.17	**3.17**	**7.83**	**25.22%**

Table 5.7: Results for semi-supervised training. Bold entries mark the best result in each column. The semi-supervised training uses our unsupervised Sintel schedule after the model has been trained on chairs and things. The second semi-supervised run doubles the batch size compared to the first. The unsupervised portions for both results were started from the same pretrained supervised model, that has its results displayed above. The supervised part of training was performed on `H1` and the unsupervised part on `H2`.

The semi-supervised results in Table 5.7 show that our semi-supervised approach can improve results to a certain extent. The KITTI and Sintel final results are improved by the additional unsupervised training, but Sintel clean results get worse. A second run with doubled batch size shows that larger training configurations can continue to improve results. It seems possible that using a sufficiently large training configuration, results could be improved over the supervised results in general, though one can not be certain of this. The significant improvement of KITTI results suggests that the unsupervised part of training may allow the model to generalize better to datasets with different distributions. In this configuration, semi-supervised training on its own does not increase results over supervised models across the board. Our semi-supervised approach thus can improve results, but is potentially somewhat limited by our training configuration.

5.8 Online Learning

We now turn to evaluating our methods incorporating multi-frame information with the aim of achieving better results.

Variant	Sintel train EPE	
	Clean	Final
Ours w/o warm-start	2.15	3.43
Ours w/ RAFT warm-start	2.13	3.54
Ours w/ our warm-start	**2.09**	**3.36**

Table 5.8: Comparison of different warm-start variants. The model used for this completed the $\mathrm{C} + \mathrm{T}$ training schedule using our supervised approach. Only the warm-start variant differs between entries, the underlying model is the same for all variants. Bold entries highlight the best result in each column.

We begin by comparing the different warm-start approaches, results for this can be found in Table 5.8. RAFT's original warm-start gives slightly better results for the clean pass, but worsens performance for the final pass. Our modified warm-start on the other hand leads to larger improvements and improves results across the board. We conclude that our modified warm-start successfully incorporates multi-frame information to improve results. It also outperforms the original warm-start by being defined for whole image domain and not having to incorporate warping. It should be noted that our warm-start requires one additional forward pass of the network to be performed, though this is generally a comparatively cheap operation.

To make our runtimes comparable with the original RAFT, we similarly report inference times when using 10 update iterations. The original RAFT reports a runtime of 100ms for this on a system with a single NVIDIA GeForce GTX 1080 Ti GPU. On `H1`, which possesses a more powerful GPU, our model can process a batch of 8 frame pairs in an average of 620ms. This implies an amortized inference time of about 78ms per frame pair. For single frame pairs, our model takes an average of 95ms per frame pair, which is similar to RAFT's timings.

Sintel sequence	Pretrained EPE		Online EPE		Fused EPE	
	Clean	Final	Clean	Final	Clean	Final
alley_1	0.369	0.416	**0.318**	0.345	0.318	**0.343**
alley_2	0.360	0.406	**0.303**	**0.334**	0.304	0.344
ambush_2	**6.129**	**20.015**	7.472	23.104	6.857	21.762
ambush_4	**11.367**	21.591	13.845	**18.760**	12.686	18.810
bamboo_2	0.949	1.073	0.891	1.070	**0.878**	**1.040**
bandage_1	0.630	0.756	**0.563**	**0.673**	0.563	0.681
sleeping_1	0.255	0.259	**0.194**	**0.197**	0.195	0.198
temple_3	**3.550**	**5.654**	4.121	6.168	3.888	6.047

Table 5.9: Results of online training on different Sintel sequences. All EPE figures are rounded to three decimal places instead of two, due to values being closer together. Bold entries mark the best values for each modality in each row. Entries marked bold respect more decimal places than the ones displayed here. All runs were performed on `H1`.

Moving on to our online training, Table 5.9 shows the result of our approach for a selection of Sintel sequences. We were not aware of the online training results for these sequences before selecting them, but selected them after visually inspecting the input frames for the sequences. We favored sequences with smooth motion and smaller flow magnitudes, but also included more challenging sequences and tried to include a variety of different sequences. All results were obtained using the same pretrained model that was trained on chairs and things with our basic supervised training configuration. Note that each sequence is trained on separately for each modality, that is the clean pass and the final pass. We trained a single online model for each sequence and modality combination and used these to obtain all of our results. We can observe that for most sequences, our online training is able to improve results beyond the pretrained prediction on its own. Training on a single sequence with our online training configuration takes about one hour on `H1`. Better results are likely achievable with higher iteration counts. It seems that for more challenging sequences with high pretrained EPE, such as the ambush sequences, online learning can lead to worse results. This may be due to the baseline pretrained flow predictions being to inaccurate, causing invalid consistencies to be enforced. We find that our fusion strategy is effective, as it dampens the effect online flow being worse than the pretrained flow and can occasionally yield better results than the online flow on its own.

Sintel sequence	Perfect fusion		Average fusion		Our fusion	
	Clean	Final	Clean	Final	Clean	Final
alley_1	0.260	0.288	**0.303**	**0.340**	0.318	0.343
alley_2	0.248	0.279	**0.295**	**0.337**	0.304	0.344
ambush_2	4.572	17.444	**6.136**	**20.784**	6.857	21.762
ambush_4	10.010	13.495	**12.484**	19.352	12.686	**18.810**
bamboo_2	0.724	0.764	**0.857**	**0.984**	0.878	1.040
bandage_1	0.468	0.550	**0.545**	**0.654**	0.563	0.681
sleeping_1	0.163	0.170	0.196	0.204	**0.195**	**0.198**
temple_3	2.931	4.568	**3.679**	**5.682**	3.888	6.047

Table 5.10: Online results for different fusion strategies. All EPE figures are rounded to three decimal places instead of two, due to values being closer together. Bold entries mark the best values for each modality in each row, the perfect fusion row is disregarded for this, as it uses a ground truth oracle. All runs were performed on `H1`.

As we know that our fusion strategy is only a simple approach that could be improved, we still evaluate our fusion strategy to gauge how much room for improvement there is. The results in Table 5.10 show our fusion approach in contrast to a perfect fusion strategy, that binarily picks the flow candidate with minimal EPE at each location using ground truth flow fields, and an average fusion strategy, that returns the average between all flow field candidates. Surprisingly, our baseline averaging strategy outperforms our fusion strategy on most sequences. Our fusion strategy may thus be improved if we attempt to build a weighted average of the candidate flow fields, rather than choosing a single candidate at each location. The perfect fusion results also show that there is more room for improvement, even when only binarily picking candidates, rather than more complex combination schemes. Note that although we only use two flow candidates for fusion, if many candidates are used even random flow fields will yield low EPE values when using the perfect fusion strategy, making the results somewhat unrealistic.

We conclude that our proposed modified warm-start and online training can improve overall results on video sequences. Our warm-start in particular improves results noticeably even over RAFT's warm-start and online training can further yield a small boost in performance. Though our fusion strategy is somewhat effective, it is lacking and should be improved to achieve better results.

5.9 Comparison with RAFT

Having examined our proposed change individually, we now try to take a more high-level view and compare our methods to the original RAFT method.

We have shown that our architectural changes lead to noticeable improvements when evaluating on Sintel in our setting. The introduction of skip connections and additional normalization were particularly effective. Note that our proposed changes to cost volume processing can also be utilized in architectures other than RAFT using cost volumes, to benefit from these improvements. Though our chosen upsampling strategy significantly outperforms bilinear upsampling, it is only slightly better than RAFT's convex upsampling. Nonetheless, it improves results at little to no additional cost.

Our unsupervised results do not represent improvements over RAFT's results, but are competitive with and may be able to outperform previous unsupervised methods. Our learned losses in particular provide an advantage over traditional losses and can be employed for models of arbitrary architecture. The semi-supervised approach can partially improve results over those of our supervised model. This shows that the incorporation of unsupervised approaches does have the potential to uplift results over pure supervised training.

With our online results, we are able to improve the accuracy of optical flow further by incorporating multi-frame information. Our modified warm-start is a significant improvement over RAFT's warm-start, and leads to larger, more consistent improvements. Online training can further lead to a small improvement on video sequences.

Viewed as a method in general, and not directly in comparison with RAFT, our approaches allows achieving very good results on Sintel, even when limited to a small computational budget. Though computationally cheaper models for optical flow estimation exist [22], they typically sacrifice results to a more significant extent to allow for this. They also generally still require more expensive training than our method. On the other hand, a few hours of training on a single GPU can lead to comparatively good results on benchmarks with our method. Compared to RAFT, this represents an alternative when having only limited computational budget, but still requiring high-accuracy optical flow. This is especially useful if one has to perform their own training runs, such as when the distribution of images one wants to infer flow for significantly differs from that of typical datasets.

Getting back to a more direct comparison with RAFT, our experiments have shown that our methods can noticeably outperform RAFT, when working in a setting with our limited computation budget. Ideally, we would like to be able to show the same without these limits on computational cost. As this is not feasible for us, we instead try to reason about why these results might scale to more expensive configurations such as that of RAFT, and note some potential reasons that may prevent this.

Our changes are generally motivated with a reason for why they should improve results, which are often backed up by previous work observing this effect. The individual ablations show that they can present an improvement when working within a limited computational budget. From experiments with slightly higher computational cost, it can also be seen that our method can scale, at least to a certain extent. One limiting factor for such scaling is our model's reduced complexity. As these reductions in complexity were motivated by our limited computation budget in the first place, they longer need to be employed when more computational power is available. This should thus be addressed by reverting our simplifications including the reduction of channel counts, the number of update iterations, and the number of residual blocks on each level in the encoders. This also applies to our decreased image size and iteration count when training. One final consideration is that our increased batch size may need to be sustained for our results to carry over to more expensive training. This may increase the overall computation cost compared to RAFT. Though this is not necessarily bad in and of itself, we have also seen that batch sizes and iteration counts can be modulated to fit a fixed computation budget, while still obtaining good results. It is likely that such tweaking could also be applied to achieve similar results for a computational budget roughly equivalent to that of RAFT.

We thus believe that it is likely for our changes to represent improvements over the original RAFT method, even in the general setting. To achieve such results, some of the simplifications to our model and training configuration may have to be reverted. Individual hyperparameters and training schedules may also need to be re-tuned to fit the new setting.

As RAFT currently outperforms all published non-RAFT-based approaches, the comparison with RAFT also gives a point of comparison against other previous literature.

Chapter Outlook

We have now performed an evaluation of our work and know the effectiveness of our proposed changes. We have also seen what our results look like and where we stand in comparison to RAFT. With this, the last thing left to do is to review out work as a whole. In the next chapter, we will examine to which extent the results of our work have met our aim, give concluding remarks, and leave off with potential directions of future work.

6 Conclusion

We end by putting our work and results into a greater context, giving concluding remarks and an outlook on possible directions of future work.

We have proposed modifications to the RAFT method and shown that many of them can lead to noticeably improved results in realistic scenarios for our setting of a fixed computational budget. Our changes to cost volumes and their improvements are even applicable to methods using cost volumes in general. Further, we have presented an approach for training RAFT in an unsupervised manner, and introduced learned losses that can significantly improve results in an unsupervised setting and are applicable to models with arbitrary architectures. We also applied online learning to optical flow estimation to further improve results on video sequences. Our online approach is also adaptable to learned methods for optical flow estimation in general. All in all, we addressed many of RAFT's shortcomings, successfully applied it to different domains, and introduced useful concepts relevant to learned models for optical flow estimation in general along the way.

We originally set out to improve optical flow results for realistic video sequences by improving upon RAFT. Within our limited computation budget, we certainly outperform RAFT and give competitive results for realistic video sequences. However, our concrete results are limited to this fixed scenario.

This is not necessarily a bad thing. Though this is not what we set out to do, as a result of optimizing results within a limited computational budget, our approaches allow achieving very good results within very low training times even with a single GPU. If low training times on limited hardware are required, our methods can be trained in less time than the vast majority of previous approaches and still often yield better results than these previous approaches.

Despite this, we can not say that our aim was completely met, and we have only demonstrably met our aim within the limitation of a fixed, smaller computational budget. Nevertheless, we have shown that many of our modifications represent noticeable improvements and have reasonably argued that these results should scale to computational budgets similar to that of RAFT. The contributed concepts including learned losses and our online learning approach are generally applicable and promise improved results.

6.1 Future Work

Though we have explored many different ideas and directions, there are still other promising directions we did not get to pursue. Further, due to us exploring different areas, not all aspects of our methods were developed as deeply as they could be. The newly introduced concepts also give a starting point, from which further steps towards improvement can be taken.

In the following, we mention some unexplored ideas and possible directions for future work as an outlook, concluding this thesis.

First and foremost, our experiments should be repeated with more computational resources, and potentially reverted simplifications, to confirm whether our results scale beyond our limited computation budget. With more computational resources, more extensive hyperparameter tuning and variants of architectures could also be explored.

The estimation of occlusions could also be improved and made part of the network, as is done by IRR [23]. Such an integration could similarly improve optical flow estimation results. Alternatively, a separate network could be trained to estimate occlusions, which would allow for better occlusion estimation even when training in an unsupervised setting, where occlusion maps are often used as part of loss functions.

Cost volumes could also be made fully learnable, similar to DICL [75], as mentioned in Section 4.2.3.

On the case of flow upsampling, one could learn separate models for flow upsampling, as has been previously explored by InterpoNet [84]. This would allow learned upsampling strategies to use more complex models, without incurring the associated computational cost when training the main flow estimation network.

Incorporating additional data aspects, such as depth, camera pose, material properties, and stereo frames could additionally improve results for scenarios where such additional data is readily available. From there, models could also be extended to infer full scene flow rather than just projected optical flow.

To improve results for scenes similar to those in the KITTI datasets, the concept of rigid motion could be directly embedded into the approach, such as is done for RAFT-based architectures in [69].

The robustness of RAFT-based approaches could also be investigated, and typical defenses to adversarial attacks could be applied as a defense strategy to improve robustness. This would be particularly valuable for real-world applications, where lacking robustness guarantees could lead to catastrophic failures of systems utilizing optical flow information.

Our unsupervised training scheme could benefit from further tuning and the newly introduced learning losses may be improved through further iteration. Our semi-supervised approach also only naively divides supervision between datasets and could be made more involved to improve results.

For video sequences, cost volumes taking in information from multiple frames could improve results in occluded regions. The modified warm-start strategy we follow could be further improved upon through learning, by using initialization strategies similar in function to ProFlow [46]. Our online training could also be taken further by not just tuning to the specific sequence, but also tuning to each individual frame pair. The fusion strategy we employ also clearly leaves room for improvement, one could potentially even train a separate model to perform fusion of multiple flow candidates.

Finally, the architectural design of models for optical flow estimation could be fundamentally changed. This could be done by applying learning at a different level of abstraction, as described in Section 3.1. At higher levels of abstraction, one could employ meta-learning to search for architectures or even objectives. On lower levels, one might search for suitable energy functionals for variational approaches or use NNs to produce implicit systems of equations as part of an otherwise more traditional method. Alternatively, one could also optimize flow fields parametrized by NNs or use adversarial learning often used for generative methods to learn a model for optical flow estimation in conjunction with a discriminator that effectively learns a suitable energy functional. The latter could even enable new kinds of unsupervised training.

Acknowledgments

A small portion of the computational work required for training during iteration was performed on the computational resource bwUniCluster funded by the Ministry of Science, Research and the Arts Baden-Württemberg and the Universities of the State of Baden-Württemberg, Germany, within the framework program bwHPC.

Bibliography

[1] 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 (Mar. 2011), pp. 1–31. ISSN: 1573-1405. DOI: 10.1007/s11263-010-0390-2 (cit. on p. 26).

[2] W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, M.-H. Yang. “Depth-aware video frame interpolation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3703–3712. DOI: 10.1109/CVPR.2019.00382 (cit. on p. 9).

[3] A. Bar-Haim, L. Wolf. “ScopeFlow: dynamic scene scoping for optical flow”. In: 2020, pp. 7995–8004. DOI: 10.1109/CVPR42600.2020.00802 (cit. on pp. 27, 51).

[4] J. L. Barron, D. J. Fleet, S. S. Beauchemin. “Performance of optical flow techniques”. In: *International Journal of Computer Vision* 12.1 (Feb. 1994), pp. 43–77. ISSN: 1573-1405. DOI: 10.1007/BF01420984 (cit. on p. 26).

[5] H. Bilen, B. Fernando, E. Gavves, A. Vedaldi. “Action recognition with dynamic image networks”. In: vol. 40. 12. 2018, pp. 2799–2813. DOI: 10.1109/TPAMI.2017.2769085 (cit. on p. 9).

[6] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black. “A naturalistic open source movie for optical flow evaluation”. In: *Proc. European Conference on Computer Vision (ECCV)*. LNCS 7577. Springer, 2012, pp. 611–625. DOI: 10.1007/978-3-642-33783-3_44 (cit. on pp. 9, 25, 27, 41, 62).

[7] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. “Learning phrase representations using RNN encoder–decoder for statistical machine translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179 (cit. on pp. 19, 36).

[8] D.-A. Clevert, T. Unterthiner, S. Hochreiter. “Fast and accurate deep network learning by exponential linear units (elus)”. In: *arXiv e-prints* (2015). arXiv: 1511.07289 [cs.LG] (cit. on p. 16).

[9] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, 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)*. 2015, pp. 2758–2766. DOI: 10.1109/iccv.2015.316 (cit. on pp. 9, 26, 32, 34, 51).

[10] D. Eigen, C. Puhrsch, R. Fergus. “Depth map prediction from a single image using a multi-scale deep network”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’14. Montreal, Canada: MIT Press, 2014, pp. 2366–2374 (cit. on p. 32).

[11] K. Fukushima, S. Miyake. "Neocognitron: a self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and Cooperation in Neural Nets*. LNBM 45. Springer Berlin Heidelberg, 1982, pp. 267–285. DOI: 10.1007/978-3-642-46466-9_18 (cit. on p. 16).

[12] 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*. IEEE. 2012, pp. 3354–3361 (cit. on pp. 26, 51).

[13] X. Glorot, A. Bordes, Y. Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323 (cit. on p. 16).

[14] P. Godet, A. Boulch, A. Plyer, G. L. Besnerais. "STaRFlow: a spatiotemporal recurrent cell for lightweight multi-frame optical flow estimation". In: *arXiv e-prints* (2020). arXiv: 2007.05481 [cs.CV] (cit. on p. 58).

[15] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, H. S. Seung. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: vol. 405. 6789. Nature Publishing Group, 2000, pp. 947–951. DOI: 10.1038/35016072 (cit. on p. 16).

[16] K. He, X. Zhang, S. Ren, J. Sun. "Deep residual learning for image recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on pp. 21, 42, 43, 69).

[17] S. Hochreiter, J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 19).

[18] K. Hornik, M. Stinchcombe, H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Netw.* 2.5 (1989), pp. 359–366. ISSN: 0893-6080 (cit. on p. 17).

[19] Y. Hu, Y. Li, R. Song. "Robust interpolation of correspondences for large displacement optical flow". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4791–4799. DOI: 10.1109/CVPR.2017.509 (cit. on pp. 49, 73).

[20] Z. Huang, T. Zhang, W. Heng, B. Shi, S. Zhou. "RIFE: real-time intermediate flow estimation for video frame interpolation". In: *arXiv e-prints* (2020). arXiv: 2011.06294 [cs.CV] (cit. on p. 9).

[21] F. Huguet, F. Devernay. "A variational method for scene flow estimation from stereo sequences". In: *2007 IEEE 11th International Conference on Computer Vision*. 2007, pp. 1–7. DOI: 10.1109/ICCV.2007.4409000 (cit. on p. 26).

[22] T.-W. Hui, C. C. Loy. "LiteFlowNet3: resolving correspondence ambiguity for more accurate optical flow estimation". In: *Computer Vision – ECCV 2020*. LNCS 12365. Cham: Springer International Publishing, 2020, pp. 169–184. ISBN: 978-3-030-58565-5. DOI: 10.1007/978-3-030-58565-5_11 (cit. on p. 79).

[23] J. Hur, S. Roth. "Iterative residual refinement for joint optical flow and occlusion estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5754–5763. DOI: 10.1109/CVPR.2019.00590 (cit. on pp. 9, 26, 32, 34, 51, 82).

[24] 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. Cham: Springer International Publishing, 2018, pp. 626–643. ISBN: 978-3-030-01258-8. DOI: 10.1007/978-3-030-01258-8_38 (cit. on pp. 26, 51).

[25] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox. "Flownet 2.0: evolution of optical flow estimation with deep networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2462–2470. DOI: 10.1109/CVPR.2017.179 (cit. on pp. 9, 34).

[26] S. Ioffe, C. Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456. DOI: 10.5555/3045118.3045167 (cit. on p. 19).

[27] J. Janai, F. Güney, A. Ranjan, M. Black, A. Geiger. "Unsupervised learning of multi-frame optical flow with occlusions". In: *Computer Vision – ECCV 2018*. LNCS 11220. Cham: Springer International Publishing, 2018, pp. 713–731. ISBN: 978-3-030-01270-0. DOI: 10.1007/978-3-030-01270-0_42 (cit. on pp. 34, 58).

[28] Y. Jiao, G. Shi, T. D. Tran. "Optical flow estimation via motion feature recovery". In: *arXiv e-prints* (2020). arXiv: 2101.06333 [cs.CV] (cit. on p. 58).

[29] R. Jonschkowski, A. Stone, J. T. Barron, A. Gordon, K. Konolige, A. Angelova. "What matters in unsupervised optical flow". In: *Computer Vision – ECCV 2020*. LNCS 12347. Cham: Springer International Publishing, 2020, pp. 557–572. ISBN: 978-3-030-58536-5. DOI: 10.1007/978-3-030-58536-5_33 (cit. on pp. 33, 45, 53, 54, 74, 75).

[30] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, T. Aila. "Analyzing and improving the image quality of stylegan". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8110–8119. DOI: 10.1109/CVPR42600.2020.00813 (cit. on p. 21).

[31] D. P. Kingma, J. Ba. "Adam: a method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 17).

[32] 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*. Ed. by D. Cremers, I. Reid, H. Saito, M.-H. Yang. LNCS 9007. Cham: Springer International Publishing, 2015, pp. 595–610. ISBN: 978-3-319-16814-2. DOI: 10.1007/978-3-319-16814-2_39 (cit. on p. 51).

[33] A. Krizhevsky, I. Sutskever, G. E. Hinton. "ImageNet classification with deep convolutional neural networks". In: vol. 60. 6. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 84–90. DOI: 10.1145/3065386 (cit. on p. 32).

[34] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein. "Visualizing the loss landscape of neural nets". In: *Advances in neural information processing systems*. 2018, pp. 6389–6399 (cit. on pp. 22, 43).

[35] L. Liu, J. Zhang, R. He, Y. Liu, Y. Wang, Y. Tai, D. Luo, C. Wang, J. Li, F. Huang. "Learning by analogy: reliable supervision from transformations for unsupervised optical flow estimation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6489–6498. DOI: 10.1109/CVPR42600.2020.00652 (cit. on pp. 33, 55).

[36] 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)*. 2019, pp. 4566–4575. DOI: 10.1109/CVPR.2019.00470 (cit. on p. 33).

[37] P. Liu, I. King, M. R. Lyu, J. Xu. “DDFlow: learning optical flow with unlabeled data distillation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8770–8777. DOI: 10.1609/aaai.v33i01.33018770 (cit. on p. 33).

[38] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. Yen. “A survey on evolutionary neural architecture search”. In: *arXiv e-prints* (2020). arXiv: 2008.10937 [cs.NE] (cit. on pp. 15, 32).

[39] I. Loshchilov, F. Hutter. “Decoupled weight decay regularization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7 (cit. on p. 18).

[40] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, Z. Gao. “DVC: An end-to-end deep video compression framework”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11006–11015. DOI: 10.1109/CVPR.2019.01126 (cit. on p. 9).

[41] 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)*. 2020, pp. 2694–2702. DOI: 10.1109/WACV45572.2020.9093590 (cit. on pp. 35, 36).

[42] J. Luiten, T. Fischer, B. Leibe. “Track to reconstruct and reconstruct to track”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1803–1810. DOI: 10.1109/LRA.2020.2969183 (cit. on p. 9).

[43] K. Luo, C. Wang, S. Liu, H. Fan, J. Wang, J. Sun. “UPFlow: upsampling pyramid for unsupervised optical flow learning”. In: *arXiv e-prints* (2020). arXiv: 2012.00212 [cs.CV] (cit. on p. 50).

[44] K. Luo, C. Wang, N. Ye, S. Liu, J. Wang. “OccInpFlow: occlusion-inpainting optical flow estimation by unsupervised learning”. In: *arXiv e-prints* (2020). arXiv: 2006.16637 [cs.CV] (cit. on p. 56).

[45] X. Luo, J.-B. Huang, R. Szeliski, K. Matzen, J. Kopf. “Consistent video depth estimation”. In: 39.4 (2020). DOI: 10.1145/3386569.3392377 (cit. on p. 58).

[46] D. Maurer, A. Bruhn. “ProFlow: learning to predict optical flow”. In: *Proc. British Machine Vision Conference (BMVC)*. BMVA Press, 2018 (cit. on pp. 57, 82).

[47] 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 clow estimation”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4040–4048. DOI: 10.1109/CVPR.2016.438 (cit. on pp. 26, 51).

[48] 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: vol. 126. 9. 2018, pp. 942–960. DOI: 10.1007/s11263-018-1082-6 (cit. on p. 26).

[49] S. Meister, J. Hur, S. Roth. “UnFlow: unsupervised learning of optical flow with a bidirectional census loss”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (cit. on pp. 33, 75).

[50] M. Menze, A. Geiger. “Object scene flow for autonomous vehicles”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3061–3070 (cit. on pp. 26, 51, 62).

[51] V. Nair, G. E. Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077 (cit. on p. 16).

[52] M. Neoral, J. Šochman, J. Matas. “Continual occlusion and optical flow estimation”. In: *Computer Vision – ACCV 2018*. Ed. by C. Jawahar, H. Li, G. Mori, K. Schindler. LNCS 11364. Cham: Springer International Publishing, 2019, pp. 159–174. ISBN: 978-3-030-20870-7. DOI: 10.1007/978-3-030-20870-7_10 (cit. on p. 58).

[53] S. Niklaus, F. Liu. “Softmax splatting for video frame interpolation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5437–5446 (cit. on p. 9).

[54] A. Odena, V. Dumoulin, C. Olah. “Deconvolution and checkerboard artifacts”. In: *Distill* (2016). DOI: 10.23915/distill.00003. URL: http://distill.pub/2016/deconv-checkerboard (cit. on p. 21).

[55] M. Otte, H.-H. Nagel. “Estimation of optical flow based on higher-order spatiotemporal derivatives in interlaced and non-interlaced image sequences”. In: *Artificial Intelligence* 78.1 (1995). Special Volume on Computer Vision, pp. 5–43. ISSN: 0004-3702. DOI: 10.1016/0004-3702(95)00033-X (cit. on p. 26).

[56] R. Pascanu, T. Mikolov, Y. Bengio. “On the difficulty of training recurrent neural networks”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318 (cit. on p. 20).

[57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. “PyTorch: an imperative style, high-performance deep learning library”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, R. Garnett. Vol. 32. Curran Associates, Inc., 2019, pp. 8024–8035 (cit. on p. 63).

[58] E. Real, C. Liang, D. So, Q. Le. “AutoML-zero: evolving machine learning algorithms from scratch”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8007–8019 (cit. on p. 32).

[59] 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)*. 2015, pp. 1164–1172. DOI: 10.1109/CVPR.2015.7298720 (cit. on p. 49).

[60] O. Ronneberger, P. Fischer, T. Brox. “U-Net: convolutional networks for biomedical image segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. LNCS 9351. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28 (cit. on p. 22).

[61] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Vol. 85. Report: Cornell Aeronautical Laboratory. Issues 460-461. Cornell Aeronautical Laboratory, 1957 (cit. on p. 16).

[62] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry. “How does batch normalization help optimization?” In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 2483–2493 (cit. on p. 22).

[63] L. N. Smith. “Cyclical learning rates for training neural networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, pp. 464–472. DOI: 10.1109/WACV.2017.58 (cit. on p. 18).

[64] L. N. Smith, N. Topin. “Super-convergence: very fast training of neural networks using large learning rates”. In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. Vol. 11006. International Society for Optics and Photonics. SPIE, 2019, pp. 369–386. DOI: 10.1117/12.2520589 (cit. on pp. 18, 65).

[65] J. T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller. “Striving for simplicity: the all convolutional net”. In: (2015) (cit. on p. 23).

[66] K. O. Stanley, R. Miikkulainen. “Evolving neural networks through augmenting topologies”. In: *Evolutionary computation* 10.2 (2002), pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811 (cit. on p. 17).

[67] C. Sun, A. Shrivastava, S. Singh, A. Gupta. “Revisiting unreasonable effectiveness of data in deep learning era”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 843–852. DOI: 10.1109/ICCV.2017.97 (cit. on p. 25).

[68] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. “PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8934–8943. DOI: 10.1109/CVPR.2018.00931 (cit. on pp. 9, 32, 34).

[69] Z. Teed, J. Deng. “RAFT-3D: scene flow using rigid-motion embeddings”. In: *arXiv e-prints* (2020). arXiv: 2012.00726 [cs.CV] (cit. on p. 82).

[70] Z. Teed, J. Deng. “RAFT: recurrent all-pairs field transforms for optical flow”. In: *Computer Vision – ECCV 2020*. LNCS 12347. Cham: Springer International Publishing, 2020, pp. 402–419. ISBN: 978-3-030-58536-5. DOI: 10.1007/978-3-030-58536-5_24 (cit. on pp. 3, 4, 9, 35).

[71] P. Truong, M. Danelljan, L. V. Gool, R. Timofte. “GOCor: bringing globally optimized correspondence volumes into your neural network”. In: vol. 33. 2020 (cit. on p. 46).

[72] D. Ulyanov, A. Vedaldi, V. Lempitsky. “Instance normalization: the missing ingredient for fast stylization”. In: *arXiv e-prints* (2016). arXiv: 1607.08022 [cs.CV] (cit. on p. 19).

[73] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, T. Brox. “DeMoN: depth and motion network for learning monocular stereo”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5038–5047. DOI: 10.1109/CVPR.2017.596 (cit. on p. 9).

[74] Z. Wan, Y. Mao, Y. Dai. “PRAFlow_RVC: pyramid recurrent all-pairs field transforms for optical flow estimation in robust vision challenge 2020”. In: *arXiv e-prints* (2020). arXiv: 2009.06360 [cs.CV] (cit. on p. 49).

[75] J. Wang, Y. Zhong, Y. Dai, K. Zhang, P. Ji, H. Li. “Displacement-invariant matching cost learning for accurate optical flow estimation”. In: vol. 33. 2020 (cit. on pp. 47, 82).

[76] Y. Wu, K. He. “Group normalization”. In: *Computer Vision – ECCV 2018*. LNCS 11217. Cham: Springer International Publishing, 2018, pp. 3–19. DOI: 10.1007/978-3-030-01261-8_1 (cit. on p. 19).

[77] 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. Cham: Springer International Publishing, 2020, pp. 483–499. ISBN: 978-3-030-58545-7. DOI: 10.1007/978-3-030-58545-7_28 (cit. on p. 46).

[78] B. Xu, N. Wang, T. Chen, M. Li. “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv e-prints* (2015). arXiv: 1505.00853 [cs.LG] (cit. on p. 19).

[79] H. Xu, J. Zhang. “AANet: adaptive aggregation network for efficient stereo matching”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 1959–1968. DOI: 10.1109/CVPR42600.2020.00203 (cit. on p. 35).

[80] F. Yang, Y. Cheng, J. V. D. Weijer, M. G. Mozerov. “Improved discrete optical flow estimation with triple image matching cost”. In: *IEEE Access* 8 (2020), pp. 17093–17102. DOI: 10.1109/ACCESS.2020.2968180 (cit. on p. 58).

[81] G. Yang, D. Ramanan. “Volumetric correspondence networks for optical flow”. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019, pp. 794–805 (cit. on pp. 34, 36).

[82] S. Zhao, Y. Sheng, Y. Dong, E. I. Chang, Y. Xu, et al. “MaskFlownet: asymmetric feature matching with learnable occlusion mask”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6278–6287. DOI: 10.1109/CVPR42600.2020.00631 (cit. on pp. 34, 35).

[83] Y. Zou, Z. Luo, J.-B. Huang. “DF-Net: unsupervised joint learning of depth and flow using cross-task consistency”. In: *Proceedings of the European conference on computer vision (ECCV)*. LNCS 11209. 2018, pp. 36–53. DOI: 10.1007/978-3-030-01228-1_3 (cit. on p. 9).

[84] 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)*. 2017, pp. 6363–6372. DOI: 10.1109/CVPR.2017.674 (cit. on p. 82).

Acronyms

CNN Convolutional Neural Network. 5, 9, 10, 13, 14, 15, 20, 21, 25, 26, 27, 29, 30, 31, 32, 34, 35, 38, 41, 42, 45, 46, 48, 51, 52, 53, 57

EPE Endpoint Error. 24, 32, 33, 37, 51, 55, 56, 62, 66, 67, 68, 69, 70, 71, 72, 73, 75, 76, 77, 78, 79

NN (Artificial) Neural Network. 5, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 43, 61, 83

RAFT RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. 1, 3, 4, 6, 7, 9, 10, 11, 13, 14, 15, 18, 20, 25, 27, 29, 31, 32, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 60, 61, 62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 77, 79, 80, 81, 82

ReLU Rectified Linear Unit. 16, 19, 42, 63, 64

RNN Recurrent Neural Network. 5, 15, 19, 20, 38