

Institute for Visualization and Interactive Systems

University of Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Masterarbeit

# Multi-Frame Approaches for Learning Optical Flow Predictions

Patrick Tobien

Course of Study:	Informatik
Examiner:	Prof. Dr. Andrés Bruhn
Supervisor:	Dr. Daniel Maurer, Lukas Mehl, M.Sc.
Commenced:	November 13, 2019
Completed:	June 29, 2020



## Abstract

In recent years, optical flow estimation techniques have shifted from classical variational approaches to learning-based methods, which show promising results with respect to both accuracy and speed. However, only few attempts have been made at using more than two frames of the input sequence for optical flow prediction. ProFlow by Maurer and Bruhn [MB18] is a self-supervised learning approach that uses three consecutive frames in order to estimate forward motion from the corresponding backward motion. By capitalizing on the temporal information contained in the sequence, ProFlow manages to improve performance in occluded regions, where a good estimation is particularly difficult. In this work, ProFlow is extended such that it uses backward motions over multiple frames to predict the corresponding forward motion, thus leveraging additional information from the past. Four different multi-frame modifications are introduced that each enable a forward flow estimation from multiple backward flows. The best performing approach employs several convolutional neural networks (CNNs) where each network is trained on a separate backward flow. By explicitly combining the respective predictions, the model achieves an improvement of between 10% and 15% on the Sintel datasets, proving that additional temporal information can be leveraged for optical flow prediction.



# Contents

1	Introduction	7
1.1	Motivation . . . . .	7
1.2	Contributions . . . . .	8
1.3	Outline . . . . .	8
2	Background	11
2.1	Optical Flow . . . . .	11
2.2	Convolutional Neural Networks . . . . .	12
2.3	Error Measures . . . . .	16
3	Related Work	19
3.1	Datasets . . . . .	19
3.2	Optical Flow Estimation with CNNs . . . . .	20
3.3	Optical Flow Estimation with Multiple Frames . . . . .	25
3.4	ProFlow . . . . .	28
4	General Approach: ProFlowS	33
4.1	CNN Architecture . . . . .	33
4.2	Results . . . . .	35
4.3	Multi-Frame Challenges . . . . .	36
5	Multi-Frame Modifications	37
5.1	Notation . . . . .	37
5.2	Multiple Backward Flows . . . . .	38
5.3	Flow Differences . . . . .	42
5.4	Multiple Warped Backward Flows . . . . .	45
5.5	Separate CNNs . . . . .	47
5.6	Sparse Convolution . . . . .	49
6	Results and Evaluation	53
6.1	Multiple Backward Flows: Evaluation . . . . .	53
6.2	Flow Differences: Evaluation . . . . .	60
6.3	Multiple Warped Backward Flows: Evaluation . . . . .	62
6.4	Separate CNNs: Evaluation . . . . .	66
6.5	Sparse Convolution: Evaluation . . . . .	69
6.6	Summary . . . . .	73
7	Conclusion and Future Outlook	75
	Bibliography	77

## Contents

---

A	Additional Evaluation Data	83
A.1	Validity Flags . . . . .	83
A.2	Multiple Backward Flow Models . . . . .	83
A.3	Flow Differences . . . . .	86
A.4	Multiple Warped Backward Flows . . . . .	86
A.5	Separate CNNs . . . . .	87
A.6	Sparse Convolution Models . . . . .	88

# 1 Introduction

Optical flow estimation is a fundamental problem in computer vision that has many applications, including in optical mice [Ng03], autonomous cars [SB95], video processing [Tek95] and action recognition [WKSL11]. Typically, one is interested in a displacement per pixel, also called *flow field*, between two consecutive frames of an image sequence. Despite many years of continuous research and development, accurate flow estimation still remains a challenging task for most image sequences due to non-rigid motion, large displacements, illumination changes and, in particular, occlusions.

Traditional optical flow estimations rely on variational approaches where an energy functional is minimized, as established by the seminal work of Horn and Schunck [HS81]. This generally requires solving a system of partial differential equations that can be derived analytically. In practice, the resulting system of equations is solved numerically, for instance with Successive Over-Relaxation (SOR) [BBPW04].

In recent years and with the surge in popularity of deep learning methods in many computer vision tasks, convolutional neural networks (CNNs) have been introduced to solve the optical flow problem. While the classical energy functionals rely on some kind of constancy assumption, CNNs are more flexible regarding the image features they use for estimation. CNNs with multiple layers can extract and make use of far more complex features in the input data. More importantly, CNN-based algorithms are often significantly faster than their variational counterparts. CNNs come with their own share of drawbacks, however. Since neural networks learn their parameters from (labeled) input data, their accuracy heavily relies on the size and quality of the training dataset. Obtaining ground truth flow fields for natural scenes, however, is a non-trivial task. Additionally, the large amount of parameters in CNNs leads to a large memory footprint and also impairs the ability of analyzing the network.

The most accurate learning-based algorithms incorporate existing domain knowledge from classical approaches such as coarse-to-fine pyramids and warping [RB17; SYLK18b]. However, while some classical methods have exploited the temporal coherence in a sequence [BA91; VBZ11], few multi-frame attempts have been made with CNN-based approaches, even though temporal coherence is undeniably a valuable source of information in image sequences. One exception is the work of Maurer and Bruhn [MB18], in which they introduced a self-supervising learning-based approach called ProFlow that uses three consecutive images as input data. With their method they show that leveraging additional temporal information in CNNs can lead to improvements in occluded areas.

## 1.1 Motivation

The work presented here is based on the ProFlow architecture introduced by Maurer and Bruhn [MB18]. ProFlow uses three consecutive frames at time steps  $t-1$ ,  $t$  and  $t+1$ . Using these frames it predicts the forward flow ( $t$  to  $t+1$ ) from the backward flow ( $t$  to  $t-1$ ). Both the backward flow and

the corresponding forward flow are precomputed during the estimation with a (partially variational) baseline approach which are then used to train a CNN model. ProFlow therefore does not rely on a true ground truth dataset for training. Instead, it aims at improving the estimation in regions where the variational baseline fails to achieve good accuracy. Notably, ProFlow performs among the top approaches on all of the major optical flow benchmarks. However, it still struggles in regions where the input data from the baseline is sparse. More precisely, for large regions where the baseline does not produce reliable ground truth estimations, the CNN can not learn an adequate model due to insufficient training data. Leveraging additional temporal information can potentially help increase the available training data in these regions and lead to better results in general. This thesis therefore introduces modifications to the ProFlow architecture that use four or more frames to predict the forward flow of the reference frame. The here presented work thus focuses on answering the following three questions:

- Can additional temporal information increase CNN-based optical flow estimation?
- What are the challenges when using multi-frame approaches for prediction and how can we overcome them?
- What are suitable data structures to leverage temporal information in the context of CNNs?

Since ProFlow learns during the estimation process and, thus, does not need a ground truth dataset, it is a good candidate for examining and answering these questions.

### 1.2 Contributions

This work introduces and evaluates four different approaches that enable optical flow predictions from multiple frames. Among these, the best methods achieve a performance increase of between 10% and 15% percent when more than three frames are used as the input data. These results prove the usefulness of the temporal information contained in the sequence using CNNs for optical flow prediction.

Since the backward flows that are used for training are generally sparse, an additional method of explicitly handling training data sparsity based on Uhrig *et al.*'s Sparse Convolution Layers [USS+17] is introduced. Some preliminary results of combining this method with the established multi-frame modifications are also shown and evaluated.

Additionally, general multi-frame challenges and difficulties arising from particular learning methods are discussed and possible solutions established. Therefore, the here presented research provides a deeper understanding of multi-frame learning-based optical flow estimation as well as practical foundations for future work.

### 1.3 Outline

The following chapter provides an overview of relevant concepts regarding optical flow in general, popular prediction techniques and convolutional neural networks. Chapter 3 relates the content of this thesis to past and current research in the field of computer vision. Chapter 4 introduces some



general modifications to the original ProFlow architecture that allows for easier integration of the subsequent multi-frame techniques which are presented in Chapter 5 and evaluated in Chapter 6. Finally, a summary of the findings and an outlook for future work is given in the last chapter.



## 2 Background

The following sections introduce basic concepts that are vital for the presented work. First the definition of optical flow as well as important use cases are given. Following that, convolutional neural networks are discussed, in particular in the context of optical flow. Finally, important error measures for the optical flow problem are characterized.

### 2.1 Optical Flow

The optical flow concept was first introduced in the 1940s by James J. Gibson, who described the visual stimuli of moving objects and the importance to animals thereof [Gib50]. More recently, the term optical flow is often framed as the “apparent motion of brightness patterns” in an image sequence [HKH86]. The optical flow problem is thus best described as the computation of a *displacement field*  $(u, v)$  that represents the transformation of brightness patterns from one image of a sequence to the next.

More precisely, the displacements vector (optical flow vector)  $w_{i,j} = (u_{i,j}, v_{i,j})^T$  gives the motion in  $x$  and  $y$  direction at image coordinates  $(i, j)$ . If such a vector exists for all  $i \in \{1, \dots, N\}$  and  $j \in \{1, \dots, M\}$ , where  $N \times M$  is the size of the image, the flow field is called *dense*, otherwise it is called *sparse*. Note that  $u$  and  $v$  are usually not integers, i.e. the flow typically has *sub-pixel precision*. Since the displacement field describes the *relative motion* between the scene and an observer, optical flow can either be induced by a moving object or a change in perspective and should, thus, not be confused with the *true motion* in the scene.

Figure 2.1 depicts a typical flow field (left) for the reference frame (middle) to the subsequent frame in the sequence (right). Typically, flow fields are visualized in a HSV color space, where the direction of the displacement is encoded in the hue (color) and the magnitude (or velocity) in the value (brightness).



(a) Visualized flow field of the sequence to the right. (b) Reference frame at time step  $t$ . (c) Frame at time step  $t+1$ .

**Figure 2.1:** Visualization of the flow field (left) from frame  $t$  (middle) to frame  $t+1$  (right). The color denotes the direction of the displacement, while the brightness represents its magnitude.

There are numerous important applications for optical flow, varying from autonomous cars and traffic tracking [GLU12; MG15; SB95] to video processing and compression [Tek95; TJN98]. Optical computer mice, for instance, capture small low-resolution images with a camera and compute the self-motion as optical flow between consecutive frames [Ng03]. Unmanned aerial vehicles can use optical flow measurements to estimate their velocity [GBG12] or to stabilize the orientation of the vehicle [RSL09].

Traditional methods compute the optical flow field as a minimizer with respect to some kind of constancy assumption. The pioneering work of Horn and Schunck [HS81], which first introduced a variational approach for optical flow estimation, incorporates the minimization of an energy functional consisting of a data term and a smoothness term. Horn and Schunck's (linearized) data term penalizes deviations from the brightness constancy assumption, that is, the assumption that a moving object (or pixel) can be discerned by its constant brightness in both images of the sequence. The smoothness term penalizes flow deviations in the neighborhood and is based on the assumption that neighboring pixels usually belong to the same object and should therefore exhibit a similar displacement. The smoothness term also ensures a dense flow field by causing the so-called *filling-in-effect* as it propagates flow information from the neighborhood.

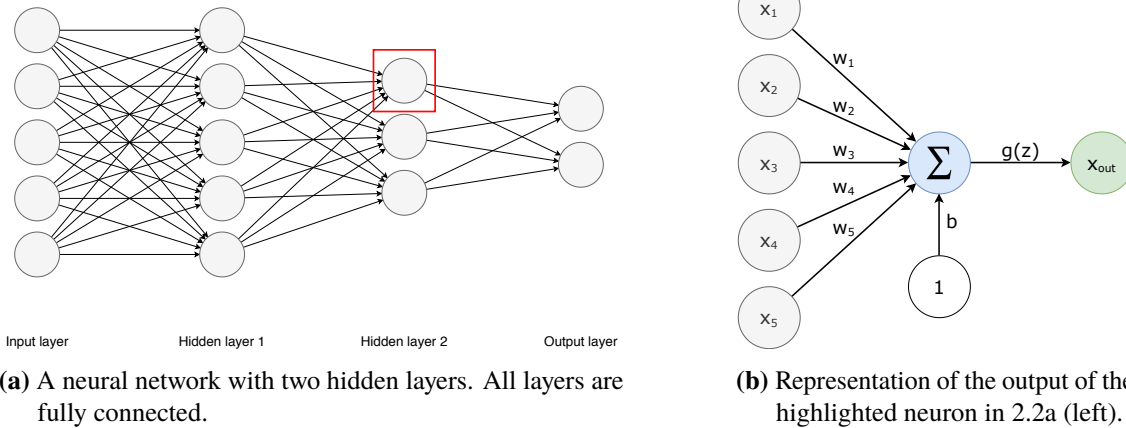
Based on the work of Horn and Schunck, variational approaches dominated the field for many years, whereby various data and smoothness terms were thoroughly explored and improved over time, often at the cost of computation speed. More modern variants make numerous modifications to the energy functional such as by incorporating multiple different constancy assumptions [UGVT88], robust (non-quadratic) data terms [BA91; BW05], or coarse-to-fine schemes and warping [BBPW04], which can help with large displacements. Generally speaking, however, the more complex the data and smoothness terms are, the more computationally expensive they become.

Newer methods thus take a whole new approach by capitalizing on current advancements in the field of Machine Learning. In recent years convolutional neural networks (CNNs) have gained traction within the field of computer vision [KSH12] and have also been proven useful for the estimation of optical flow with the work of Dosovitskiy *et al.* [DFI+15].

The following section describes convolutional neural networks in general. Recent and promising learning-based approaches are explored in Chapter 3.

## 2.2 Convolutional Neural Networks

In Machine Learning, a neural network is a computing system which loosely resembles an animal brain. Neural networks are able to discern patterns in the given input data. Here, we typically differentiate between so-called *supervised* and *unsupervised* learning. A supervised model learns via fully labeled datasets (or ground truth). After training the model, it is then able to predict a value for new, unseen, input data. The prediction value can either be discrete, in which case we talk about a *classification* problem, or continuous, which refers to a *regression* problem. In the context of optical flow, we are interested in (real) displacement values. Therefore, flow prediction is a regression problem. Alternatively, an unsupervised model typically attempts to automatically find patterns in the given input data, without explicit instructions or correct answers (i.e. no labeled training data).



**Figure 2.2:** A typical neural network and the exemplary computation of a single neuron's output.

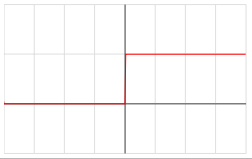
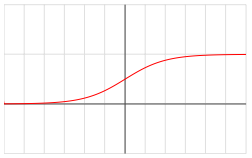
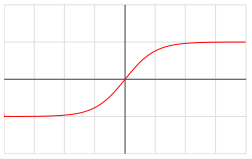

Inspired by the human brain, a neural network consists of nodes (neurons) and connectors (synapses) between nodes. Typically, the neurons are organized into layers, where the output of one layer serves as the input to the next layer (see Figure 2.2a). A network with multiple *hidden layers*, i.e. layers between the input and output layer, is also referred to as a *deep* neural network (DNN). As shown in Figure 2.2b, each neuron combines its inputs  $x_i$  with weights  $\omega_i$ , which either amplify or dampen the input. The products are then summed up and passed through an activation function  $g(\cdot)$  which determines if the signal is passed through to the next layer, and if so, to what extent. In other words, the output  $\hat{y}$  of input  $x$  is given as:

$$\begin{aligned}
 f(x) &= x_1 \cdot \omega_1 + x_2 \cdot \omega_2 + \dots + x_n \cdot \omega_n + 1 \cdot b \\
 &= b + \sum_{i=1}^n (x_i \omega_i) =: z \\
 \hat{y} &= x_{out} = g(z),
 \end{aligned}
 \tag{2.1}$$

where  $b$  is a bias that ensures that even if all inputs are 0, the neuron can still be activated. Table 2.1 shows some of the more popular activation functions.

In order to train most networks, large amounts of training data are necessary, that is, input data with labeled outputs such that the network can learn and adjust the weights that lead to the given outputs. The network does this by minimizing a loss function (such as *true value* – *predicted value*) via an optimization algorithm typically based on gradient descent (also called *backpropagation*). *Adam* (adaptive momentum estimation) [KB15] is, at this time, the most popular optimizer for neural networks. Note that because of the backpropagation algorithm, the activation function  $g(z)$  typically introduces non-linearities to the network. Otherwise the derivative of  $z$  would be a constant which has no relation to the input and, thus, the weights could not be updated via gradient descent.

*Convolutional neural networks* (CNNs) are typically a type of deep neural network that make use of convolutions with various *filters* (also called *kernels*) instead of general matrix multiplications. More precisely, a neuron in a CNN consists of a kernel (defined by a width and a height) instead of a single weight. Given an image as input, a CNN passes *patches of pixels* through each filter,

Name	Equation	Plot
Heaviside	$g(z) = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$	
Sigmoid	$g(z) = \frac{1}{1+e^{-z}}$	
TanH	$g(z) = \tanh(z) = \frac{2}{1+e^{-2z}} - 1$	
Rectified Linear Unit (ReLU) [NH10]	$g(z) = \begin{cases} 0 & \text{for } z < 0 \\ z & \text{for } z \geq 0 \end{cases}$	

**Table 2.1:** Various possible activation functions.

*convolving* the image patch with the kernel (see Figure 2.4). The output of that layer is an abstract *feature map* which then serves as the input for the next convolutional layer. A (discrete) convolution of, e.g., an image  $I$  with a kernel  $\omega$  is defined as follows:

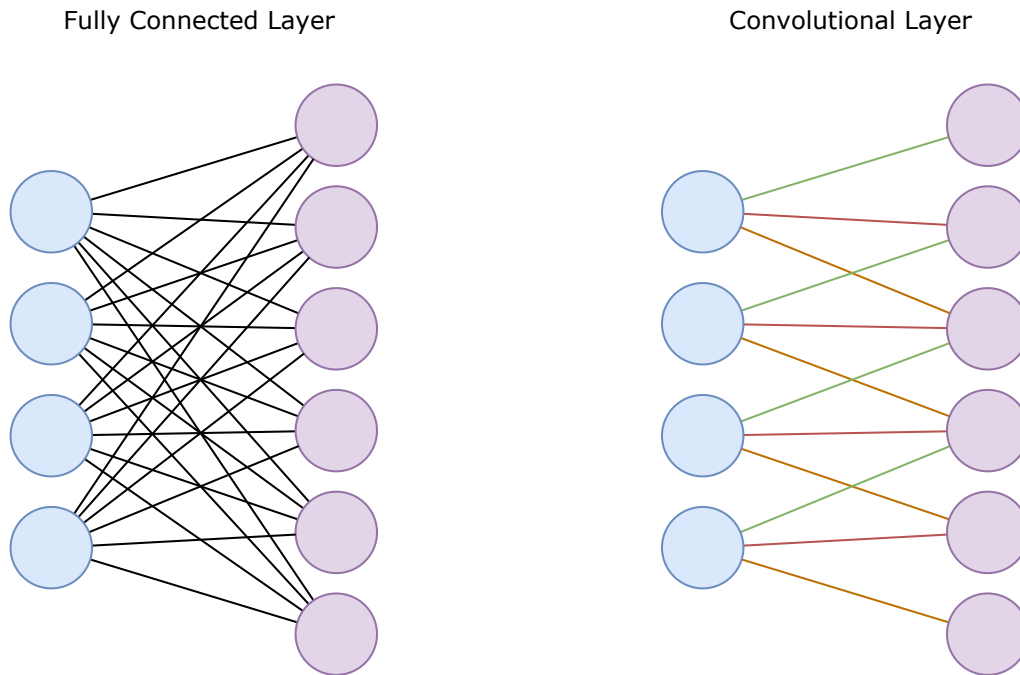
$$h(i, j) = \omega * I(i, j) = \sum_{s=-a}^a \sum_{t=-b}^b \omega(s, t) \cdot I(i - s, j - t), \quad (2.2)$$

where  $i$  and  $j$  are the image coordinates,  $*$  is the convolution operation, and  $a \times b$  is the kernel size. With feature  $x$  as input and a kernel size of  $2k - 1$ , the output of a convolutional layer in a standard CNN is then

$$f_{i,j}(x) = \sum_{s,t=-k}^k x_{i+s,j+t} \cdot \omega_{s,t} + b, \quad (2.3)$$

where  $\omega$  is the kernel's weight,  $b$  is a bias, and  $(i, j)$  is the positional index. The step size with which the kernel is applied is called the *stride*. For instance, a stride of one implies that the kernel is applied to each pixel in the image.

Convolving an image with various filters has several advantages. For example, CNNs share weights. That is, in contrast to a *fully connected layer*, the number of trainable variables does not depend on the size of the input, but instead on the size and number of filters (cf. Figure 2.3). Thus, CNNs are more compact and efficient than traditional (fully connected) neural networks. Additionally, CNNs are able to capture *features* from the data, given that the input is locally coherent. This is usually

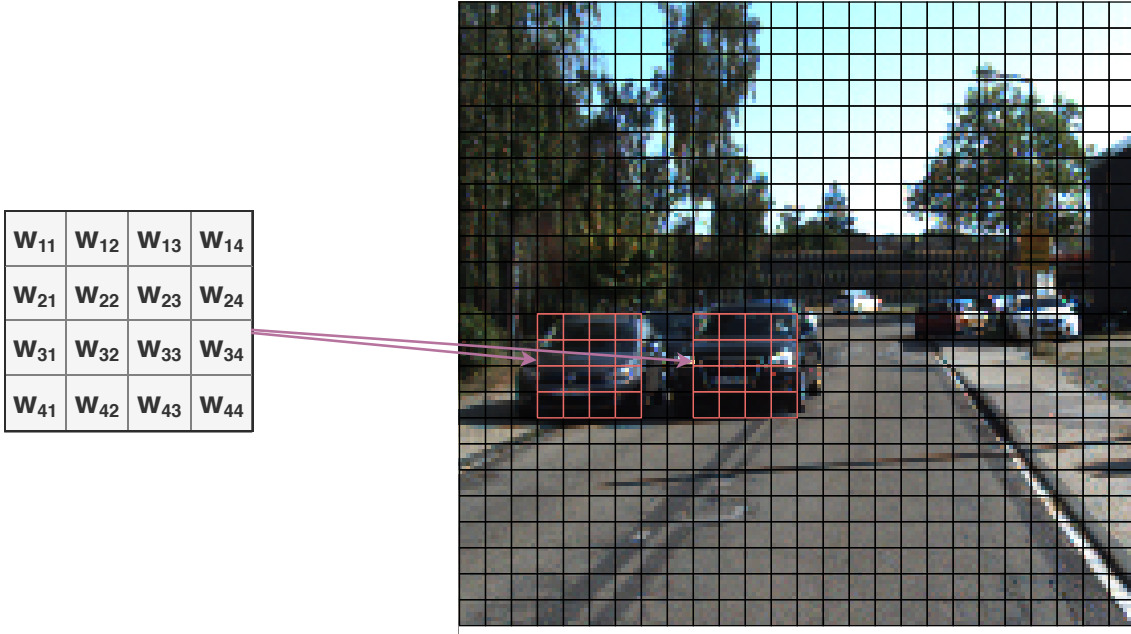


**Figure 2.3:** Left: A fully connected layer in a typical neural network. Right: A convolutional layer of a typical CNN. All neurons share weights, as indicated by the same line colors.

the case for images as neighboring pixels typically share similar properties. Filters can, for instance, detect simple features such as horizontal or vertical edges (typically in early layers of the CNN), or more complex ones such as human faces or cars (typically in deeper layers of the CNN). This is also called *feature extraction* which, as another benefit of CNNs, is location invariant. E.g., a filter that detects horizontal edges will produce similar outputs for the same edge no matter where it is located within the image. Figure 2.4 emphasizes this with the depicted cars.

In order to make training feasible, most CNNs employ a technique called *pooling*, which basically describes a downsampling or subsampling operation. A typical choice for pooling is to use a stride larger than one. Another popular approach is the so-called *max-pooling* operation. Similar to the convolution itself, this is a per-patch instead of a per-pixel operation. Following a convolution layer, only the highest value per patch is preserved and all other values are discarded. Therefore, only the locations that correlated the most with the detected feature are preserved. The result is a subsampled, lower-resolution output. Apart from reducing the computational cost, this has the advantage of generalizing the network which reduces the chance of *overfitting*. Overfitting happens when the learned model too closely resembles the input data, i.e. the network thinks of noise and randomness as underlying concepts of the data. Thus, during prediction, the network produces poor results as it does not generalize well. Since the pooling operation usually merges multiple values into a single one, outliers (noise) are filtered, making overfitting less likely.

Since the success of *AlexNet* for large-scale image classification in 2012 [KSH12], CNNs have surged in popularity within the field of Computer Vision. Today, CNNs are an invaluable tool for numerous Computer Vision tasks such as object detection [RDGF16; RHGS15; SEZ+14], segmentation [BKC17; HAGM15; HGDG17; LSD15], image reconstruction [LRS+18; OKK16] and colorization [ISI16; ZIE16], and, for some years now, optical flow estimation [DFI+15]. In



**Figure 2.4:** The convolution filter on the left is applied across the image on the right. It likely produces similar outputs for both cars.

the context of optical flow, CNNs are not only significantly faster, but do not depend on predefined constraints such as the brightness constancy assumption. As elaborated above, CNNs are able to extract relevant and complex features on their own. Their biggest caveat, however, is the necessity of large training datasets in order to learn a suitable model. Since getting ground truth optical flow data is very difficult, only few datasets exist for training or even testing. The most popular available datasets are discussed in Section 3.1. The error measures used to compare results are given in the following section.

### 2.3 Error Measures

In order to assess the performance of an optical flow estimation (including the estimations of the approaches suggested later in this work), it has to be evaluated with respect to a given ground truth flow. Typically, to do so, some kind of difference or deviation has to be computed. Because displacements are vectors, there are various measures we can take based on, e.g., angles or distances which we will see in the following.

Given an estimated flow field  $w^e$ , ground truth  $w^{gt}$  and an image size of  $N \times M$ , the first popular error measure is the **average angular error (AAE)**:

$$AAE(w^{gt}, w^e) = \frac{1}{NM} \sum_{x=1}^N \sum_{y=1}^M \arccos \left( \frac{u_{x,y}^{gt} u_{x,y}^e + v_{x,y}^{gt} v_{x,y}^e + 1}{\sqrt{u_{x,y}^{gt}{}^2 + v_{x,y}^{gt}{}^2 + 1} \sqrt{u_{x,y}^e{}^2 + v_{x,y}^e{}^2 + 1}} \right). \quad (2.4)$$



The AAE basically measures the (average) angular difference between both vectors. One downside is that it does not take the magnitude of the vectors, i.e. the velocity of the motion, into account. As an alternative, the **average endpoint error (AEE)** can be computed as

$$AEE(w^{gt}, w^e) = \frac{1}{NM} \sum_{x=1}^N \sum_{y=1}^M \sqrt{(u_{x,y}^{gt} - u_{x,y}^e)^2 + (v_{x,y}^{gt} - v_{x,y}^e)^2}. \quad (2.5)$$

Most major optical flow benchmarks (see Section 3.1) use the AEE as a measurement which, as the name implies, gives the average distance between the two vector endpoints. For this reason, it is the main error measure used in this work. However, some benchmarks additionally make use of the **bad pixel error (BP)**, which is defined as follows:

$$BP(w^{gt}, w^e) = \frac{100}{NM} \sum_{x=1}^N \sum_{y=1}^M \mathbf{1}_{\left(\sqrt{(u_{x,y}^{gt} - u_{x,y}^e)^2 + (v_{x,y}^{gt} - v_{x,y}^e)^2} > T\right)}, \quad (2.6)$$

where  $\mathbf{1}_{exp}$  is an indicator function that is 1 if  $exp$  evaluates to true and 0 otherwise, and  $T$  is a given threshold (typically around 2). In other words, BP gives the percentage of pixels for which the estimated displacement differs more than  $T$  pixels from the ground truth.



## 3 Related Work

In order to determine the performance of the multi-frame approaches introduced in the following chapters, we need some benchmark datasets to evaluate them against. Therefore, we will first look at the available benchmarks for optical flow estimation. Additionally, larger datasets used for training CNN-based approaches are discussed in the same section. Following that, we will look at general learning-based approaches in order to get an overview of the state of current research in the field of optical flow predictions. Section 3.3 will then look at existing multi-frame approaches which are closest in spirit to the here presented work. Finally, the last section explores the ProFlow architecture [MB18], which is the method this work is based on.

### 3.1 Datasets

In order to reliably compare different approaches, standardized datasets need to be established. Evidently, for the optical flow problem, the image data have to be annotated with the corresponding ground truth flow to ensure a meaningful evaluation of the estimation. In contrast to many other computer vision tasks, however, it is difficult to obtain proper ground truth data on real-world image sequences. Many datasets therefore rely on synthetic image data, such as the famous Yosemite sequence, or image data produced in controlled environments, such as the Middlebury dataset [BSL+11]. Due to the difficulties of producing such artificial or controlled sequences, these early datasets are very small and contain mostly small motions and lighting changes.

Clearly, for real-world applications like autonomous driving, larger and more realistic data would be preferable. The two *KITTI* datasets from 2012 [GLU12] and 2015 [MG15], respectively, try to achieve exactly that. Ground truth data were gathered using LIDAR (*light detection and ranging*), i.e. by mounting a laser scanner on top of a car and measuring the surrounding distances to obtain a 3D point cloud which is then projected onto consecutive frames to obtain a non-dense flow field. The 2012 dataset contains only static scenes, while the 2015 dataset also includes dynamic scenes with large motions and occlusions. The latter is thus more challenging and possibly a more realistic benchmark for current research.

Another popular dataset is the *Sintel* set from 2012 [BWSB12]. The dataset is based on the open source 3D animated short film Sintel. While artificially produced, the movie provides fairly naturalistic scenes with large motions, severe illumination changes, motion blur and atmospheric effects such as fog. To make things easier, the Sintel benchmark offers two render passes: While the *clean* pass already includes smooth shading and specular reflections, only the *final* pass contains blur due to motion or depth of field and atmospheric effects. Sintel's final render pass in particular is therefore among the most challenging benchmarks up to date. As of now, KITTI and Sintel are the most widely-used benchmarks for optical flow estimation and, thus, serve as the reference point for this thesis.

However, while the datasets are fairly large compared to earlier sequences like Yosemite and Middlebury, they are still too small for most learning-based approaches as CNNs generally require very large datasets during the training process. Therefore, Dosovitskiy *et al.* have created an additional, very large artificial dataset called *FlyingChairs* which they used to train their *FlowNet* [DFI+15] (see Section 3.2.1). It contains over 22000 image pairs and has subsequently been employed by other learning-based approaches.

Finally, Mayer *et al.*'s *FlyingThings3D* dataset is another artificially created set used for network training [MIH+16]. Inspired by the *FlyingChairs* set, it contains even more frames, longer sequences, and true 3D motion. For now, these are the only datasets which contain enough image sequences to sufficiently train most CNNs for optical flow, albeit at the cost of realism.

## 3.2 Optical Flow Estimation with CNNs

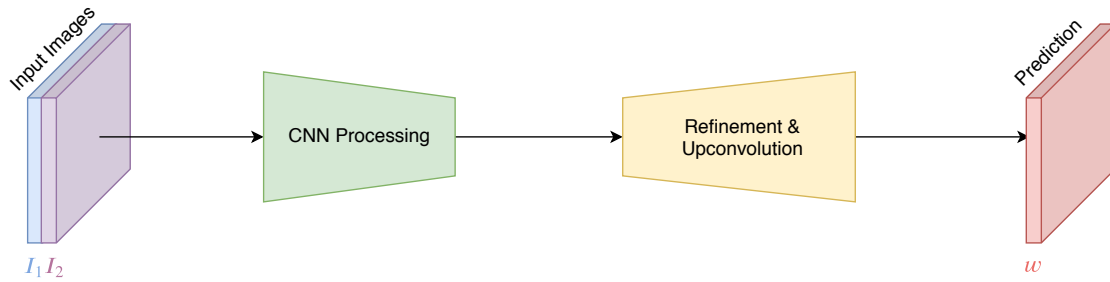
In this section, some of the most important learning-based approaches are discussed. The presented works build the foundation for all modern learning-based optical flow estimation techniques and are, thus, important to understand for the methods in this work as well. While the first attempts did not quite reach the accuracy of state-of-the-art variational approaches, more recent CNN architectures achieve not only top speeds, but also top results on the aforementioned benchmarks.

### 3.2.1 FlowNet

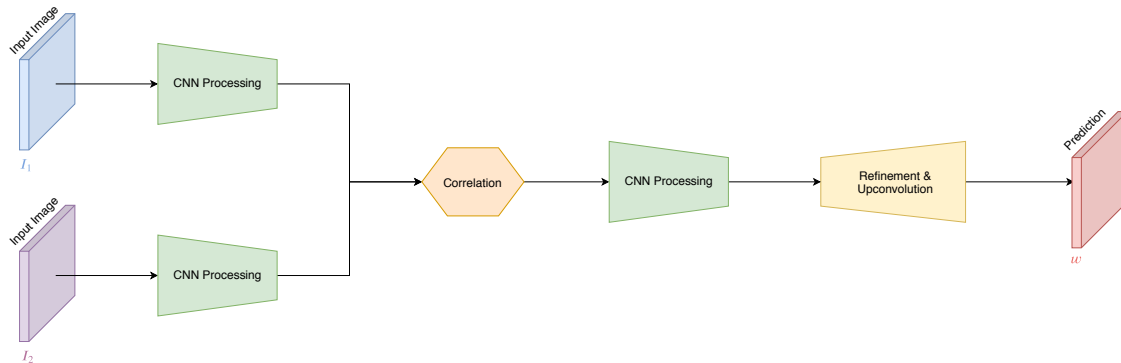
Since the success of Krizhevsky *et al.* [KSH12] regarding the performance of CNNs for image classification, CNNs have been explored and employed for numerous computer vision tasks. Dosovitskiy *et al.* [DFI+15] have laid the foundation for end-to-end learning-based optical flow estimation. In their 2015 work, they propose two CNN architectures, FlowNetS and FlowNetC, which are trained to learn a flow field (output) from a given image pair (input). The simple version of their network structure, FlowNetS, stacks the two images together and treats them as a single input. Thus, the network has to decide how to correlate the images and learn the motion by itself. In contrast, FlowNetC separates the input images via two identical processing streams and later joins them in an explicit correlation layer. That is, FlowNetC first extracts meaningful features of each image, which are then correlated as follows:

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\sigma \in [-k, k] \times [-k, k]} \langle I_1(\mathbf{x}_1 + \sigma), I_2(\mathbf{x}_2 + \sigma) \rangle, \quad (3.1)$$

where  $\mathbf{x}_1$  and  $\mathbf{x}_2$  denote the centers of two square patches of size  $2k + 1$  in image  $I_1$  and  $I_2$ , respectively. Note the similarity to a standard convolution (cf. Equation 2.2). However, in contrast to a convolution in a CNN layer, there are no trainable weights in this correlation layer and, therefore, this layer does not increase the model size. To further reduce computational costs, the maximum displacement (between  $\mathbf{x}_1$  and  $\mathbf{x}_2$ ) is limited with the assumption that the motion between two images is generally not too large. Figure 3.1 shows the two proposed network architectures from a top-level perspective.



(a) The FlowNetS Architecture.

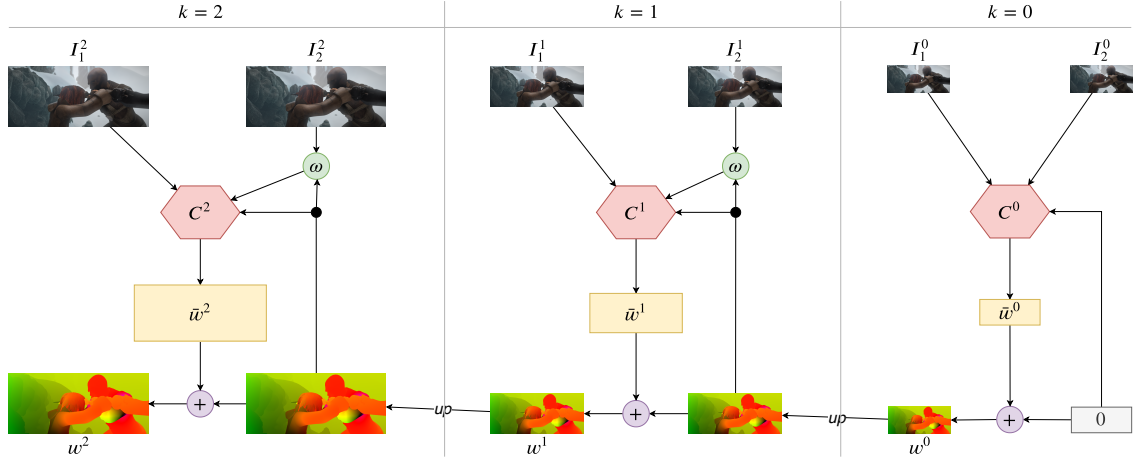


(b) The FlowNetC Architecture.

**Figure 3.1:** Abstract illustration of FlowNetS and FlowNetC as proposed by Dosovitskiy *et al.* [DFI+15]. Green indicates the contracting parts of the network, while Yellow represents the expanding part.

Both CNN architectures make use of pooling by employing a stride of two (cf. Section 2.2). This, however, leads to lower resolution outputs which would result in non-dense flow fields. In order to get a flow estimation per pixel, Dosovitskiy *et al.* employ an expanding part in their network that refines the flow via so-called *upconvolutions* or *transposed convolutions* [ZF14; ZTF11]. More precisely, during this expanding step, feature maps are upconvolved and then concatenated with the corresponding feature map from the contracting part of the network. The final upscaling, however, is either realized with bilinear interpolation to increase computation speed or with a more costly variational refinement, i.e. the coarse-to-fine scheme proposed by Brox and Malik [BM10].

As no optical flow dataset large enough to train a CNN existed at the time, Dosovitskiy *et al.* created an artificial dataset, called FlyingChairs, by applying random affine transformations to 3D models of chairs which were superimposed on various background images from Flickr (cf. Section 3.1). Despite the unrealistic dataset, FlowNetS and FlowNetC produce surprisingly good results on the Sintel and KITTI benchmarks, in particular with variational refinement. Dosovitskiy *et al.* did not find better results with FlowNetC than with FlowNetS, suggesting that the neural network is able to find correlations by itself. However, their follow-up work found that FlowNetC outperforms FlowNetS when different training schemes [IMS+17] are employed. While FlowNet does still not achieve accuracies comparable to the top variational approaches on these benchmarks, it shows remarkable results despite the lack of an adequate training dataset. Additionally, FlowNetC (without refinement) achieves competitive frame rates of up to 10 frames per second, thus proving the usefulness of CNNs for optical flow estimation.



**Figure 3.2:** Visualization of SPyNet’s network architecture with three pyramid levels [RB17].

### 3.2.2 SPyNet

Following the work of Dosovitskiy *et al.*, Ranjan and Black [RB17] try to incorporate classical optical flow estimation principles into a learning-based approach. More precisely, they employ the spatial pyramid structure from [DCSF+15], a coarse-to-fine approach that allows detection of large motion. The assumption is that large motion in the sequence results in a displacement of at most a few pixels at the coarsest level of the pyramid. Consequently, the convolution filters in the network (which typically have the size of a few pixels) can then capture and learn meaningful features of the temporal structure, something that the preceding FlowNet struggled with when the sequence contained large motions.

An overview of SPyNet (Spatial Pyramid Network) is given in Figure 3.2. The residual flow  $\bar{w}^k$  is computed with a CNN at each level  $k$  of the pyramid. That is, in every layer below the top layer, only the flow *increment* is learned and added to the output of the previous layer. Thus, the displacements at each level should stay small. The flow is then up-sampled to the next (finer) level and the second image is warped towards the first using the upsampled flow from the previous pyramid level. More precisely, the warping operator  $\phi(I, w)$  warps the image  $I$  with the given displacement  $w$  via bilinear interpolation (due to sub-pixel precision) such that  $\phi(I, w) = I(\mathbf{x} + w)$ , where  $\mathbf{x}$  is the pixel index. This kind of image warping is another principle taken from classical approaches [SRB14]. The residual flow at level  $k$  of the pyramid is then given by

$$\bar{w}^k = C^k(I_1^k, \phi(I_2^k, up(w^{k-1})), up(w^{k-1})), \quad (3.2)$$

where  $C^k$  is the CNN at level  $k$  of the pyramid,  $I_1^k$  and  $I_2^k$  are the frames at level  $k$ , and  $up(w^{k-1})$  is the upsampled flow from the previous pyramid level (cf. Figure 3.2). The flow  $w^k$  at the current pyramid level  $k$  is then computed as

$$w^k = up(w^{k-1}) + \bar{w}^k. \quad (3.3)$$

By learning residual flows instead of the actual flow, the range of flow fields in the output space is vastly restricted which results in a very small model size. In fact, compared to FlowNet, SPyNet is 96% smaller. Additionally, at the time of Ranjan and Black’s writing, SPyNet offered the best speed-accuracy ratio among all optical flow estimations, i.e. no faster method was more accurate. Thus, SPyNet shows that incorporating classical principles into new learning-based approaches can have significant benefits with respect to both computational costs and accuracy.

### 3.2.3 FlowNet 2.0

Based on the original FlowNet model, Ilg *et al.* [IMS+17] prove that learning-based methods can compete with traditional variational methods and produce state-of-the-art flow field estimations. Ilg *et al.* achieve this with three major improvements over the original FlowNet. For one, they use an additional training dataset, called FlyingThings3D [MIH+16], which contains more object variety, brightness changes and true 3D motion. Next, they show that the order in which the training data are presented matters, by experimenting with the two datasets, FlyingChairs and FlyingThings3D. Finally, they stack multiple FlowNet-CNNs on top of each other via a warping operation. Additionally, they introduce a subnetwork that specializes in small motion.

The best results were achieved by first training on FlyingChairs and then merely fine-tuning on FlyingThings3D. Similar to SPyNet’s architecture, FlowNet 2.0 stacks multiple networks via a warping operation and adopts an iterative refinement scheme from traditional estimation methods by focusing on the flow increment  $\bar{w}_k$  between the first image and the warped second image. Thus, Ilg *et al.* combine multiple FlowNetS and FlowNetC networks and evaluate different combinations thereof. When training one network, the weights of the others are fixed to prevent over-fitting. According to their experiments, the best results were achieved with one FlowNetC followed by two FlowNetS. The whole architecture is called FlowNet2-CSS. For accurate estimation of small motion, Ilg *et al.* introduce a new dataset, called ChairsSDHom, that mostly focuses on sub-pixel displacements. A slightly altered FlowNetS architecture is trained on ChairsSDHom and then fused with FlowNet2-CSS, resulting in the final FlowNet2 architecture.

The result is a learning-based approach that is on par with state-of-the-art variational methods on all major benchmarks. Regarding computation speed, FlowNet2 is vastly superior to traditional methods and, therefore, shows that CNNs might be the future of optical flow estimation.

### 3.2.4 PWC-Net

Sun *et al.* [SYLK18b] introduce the first learning-based approach that outperforms all other previously published methods on both the Sintel and KITTI 2015 benchmarks. In particular, PWC-Net yields higher accuracy than the best (and much slower) variational approaches. Inspired by FlowNet, SPyNet and FlowNet2, they combine classical principles into a CNN architecture. More precisely, they use a pyramid scheme, warping and a cost volume in their PWC-Net (PWC being the abbreviation for pyramid, warping and cost volume). By doing so, Sun *et al.* hope to increase the accuracy while simultaneously decreasing the size of the model.

Instead of a fixed image pyramid (as in SPyNet, see Figure 3.2), PWC-Net implements the coarse-to-fine approach via a learnable *feature pyramid*. At each level  $k$ , features  $c_t^k$  ( $t \in \{0, 1\}$ ) are generated with convolutional filters which downscale the features of the previous level,  $c_t^{k-1}$ . At the lowest

level ( $k = 0$ ), the features are set to the input images, i.e.  $c_1^0 = I_1$  and  $c_2^0 = I_2$ , where  $I_1$  and  $I_2$  are the first and second image of the sequence, respectively. Similarly, at each level  $k$ , the features of the second image are warped towards the features of the first image, using the upsampled flow of level  $k+1$  and bilinear interpolation. This process is similar to the warping in SPyNet, with the difference of feature representations being warped instead of the image itself, i.e:

$$c_\phi^k = c_2^k(\mathbf{x} + up(w^{k+1})), \quad (3.4)$$

with  $\mathbf{x}$  being the pixel index and  $c_2^k$  being evaluated via bilinear interpolation. Following that, a cost volume is constructed between the features of the first image and the warped features of the second image:

$$cost^k(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{N} (c_1^k(\mathbf{x}_1))^T c_\phi^k(\mathbf{x}_2), \quad (3.5)$$

where  $\mathbf{x}_1$  and  $\mathbf{x}_2$  are the pixel indexes,  $T$  is the transpose operator, and  $N$  is the length of vector  $c_1^k(\mathbf{x}_1)$ . This cost volume is similar to the correlation layer in FlowNetC (cf. Equation 3.1); the difference being the correlation of features instead of images. In practice, the search space for  $\mathbf{x}_1$  and  $\mathbf{x}_2$  is limited and, thus, only a partial cost volume is computed, which keeps computational costs small without sacrificing accuracy due to displacements being generally small in a coarse-to-fine scheme. More importantly, the warping and cost volume layers are fixed and, thus, keep the model size small. The cost volume, features of the first image  $c_1^k$  and the upsampled optical flow  $up(w^{k+1})$  then serve as the input to a multi-layer CNN at level  $k$  of the pyramid. Its output is the new flow estimate  $w^k$ . It should be noted, that the CNNs at each level of the pyramid do not share parameters. Finally, a so-called *context network* is employed which serves as a post-processing technique by employing dilated convolutions [YK16] in a small subnetwork.

The resulting PWC-Net is the first end-to-end learning-based method that outperforms the traditional variational approaches on the Sintel and KITTI benchmarks. It achieves higher accuracy at better and even interactive speeds of up to 35 frames per second with a fairly small model size, in particular compared to FlowNet and FlowNet2. PWC-Net thus again proves the benefits of combining existing domain knowledge with a CNN-based learning approach.

### 3.2.5 Other Methods

Since the introduction of FlowNet in 2015, there has been a lot of additional research with learning-based flow estimation [HTC18; ISKB18; ZLB17]. PWC-Net in particular proved to be very influential: Most more recently published methods base their approach on the PWC-architecture [HR19; LLKX19; NŠM18; SYLK18a; YR19].

Note that the above methods are all supervised learning approaches. Some effort has also been put into unsupervised optical flow estimation. Generally, most unsupervised methods warp the second image towards the first image via the estimated flow. The difference between the first image and the warped second image is then minimized with respect to the flow [AP16; JHD16; RYN+17]. This can lead to fairly good results in non-occluded regions, but does not work well for occluded pixels. Although some more recent methods provide specific occlusion reasoning with occlusion maps [MHR18; WYY+18], most unsupervised approaches generally do not perform on par with



supervised methods. One exception is the so-called self-supervised, but partially learning-based, ProFlow [MB18]. Another promising approach is SelFlow [LLKX19] which is a self-supervised method based on the PWC-Net architecture. Both methods use additional temporal information from multiple frames in the sequence and are discussed in the following sections, among other multi-frame approaches.

### 3.3 Optical Flow Estimation with Multiple Frames

This section focuses on approaches that try to improve accuracy by using additional temporal information, usually by incorporating more than 2 frames of a sequence. Some earlier research has been conducted with variational approaches [BA91; GRA13; KT15; SS07; SSB10; SWS+13; VBZ11]. Typically, either the energy functional is extended to include the assumption of smooth flow over time (e.g. via constant velocity or acceleration) or different estimates are fused together. For CNN based methods, however, there have only been very few attempts at leveraging temporal information, some of which are discussed in the following.

#### 3.3.1 PWC-Fusion

Ren *et al.* [RGS+19] build a separate fusion network with which they extend the pretrained PWC-Net models. They use past motion information by warping flow estimations of previous frames to the current frame, thereby gaining multiple estimates. Each estimate represents a valid candidate flow for the current frame. The fusion network then fuses these estimates together, resulting in a new optical flow field that capitalizes on the temporal information from previous frames. They achieve better results than PWC-Net on all major benchmarks, showing that temporal information can be leveraged for learning-based approaches as well. Since the fusion network is independent of the optical flow estimation itself, their approach can, in theory, be used with any optical flow algorithm and is not restricted to the PWC-Net architecture.

Importantly, Ren *et al.* conduct an “oracle” study to observe the usefulness of additional temporal information. To do so, they take three frames at  $t-1$ ,  $t$  and  $t+1$ . In addition to the flow  $w_{t \rightarrow t+1}$  from  $t$  to  $t+1$ , they compute the flow from  $t-1$  to  $t$  and warp it such that the coordinates coincide with the reference frame  $I_t$ . More precisely, the warped flow  $\hat{w}_{t-1 \rightarrow t}$  is computed with bilinear interpolation as follows:

$$\hat{w}_{t-1 \rightarrow t}(\mathbf{x}) = w_{t-1 \rightarrow t}(\mathbf{x} + w_{t \rightarrow t-1}), \quad (3.6)$$

where  $\mathbf{x}$  is the pixel coordinate and  $w_{t \rightarrow t-1}$  is the backward flow from  $t$  to  $t-1$ . They then compare the warped flow and the current flow to the actual ground truth and, for each pixel, select the better estimate. The resulting “oracle” flow achieves higher accuracy than the simple two-frame approach, i.e. PWC-Net. This confirms that the warped previous flow contains complementary information and, therefore, optical flow estimation can benefit from the temporal coherence in the scene.

### 3.3.2 ContinualFlow

Neoral *et al.* [NŠM18] include temporal connections into the PWC-Net architecture. That is, the flow of the previous time step ( $t-1$ ) is added as an additional input to the network. To do so, the previous flow is warped such that the image coordinates coincide with the reference frame. The flow can be transformed either via forward warping or backward warping. Forward warping takes the optical flow from  $t-1$  itself to transform the coordinates. More precisely, the warped flow  $\hat{w}_{t-1 \rightarrow t}$  is computed as

$$\hat{w}_{t-1 \rightarrow t}(\mathbf{x} + \text{round}(w_{t-1 \rightarrow t}(\mathbf{x}))) = w_{t-1 \rightarrow t}(\mathbf{x}), \quad (3.7)$$

where  $\mathbf{x}$  is the pixel position and  $w_{t-1 \rightarrow t}$  is the flow estimate of the previous frame. Alternatively, coordinates can be transformed by warping  $w_{t-1 \rightarrow t}$  with the backward flow  $w_{t \rightarrow t-1}$  from frame  $I_t$  to frame  $I_{t-1}$  as follows:

$$\hat{w}_{t-1 \rightarrow t}(\mathbf{x}) = w_{t-1 \rightarrow t}(\mathbf{x} + w_{t \rightarrow t-1}(\mathbf{x})). \quad (3.8)$$

Since this requires the additional estimation of the backward flow  $w_{t \rightarrow t-1}$ , it is computationally more expensive than PWC-Net. Neoral *et al.*, however, suggest to concatenate the forward transformation, backward transformation and backward flow as input to their altered PWC-Net for the best results. Additionally, they extend the network with an occlusion estimator based on the cost volume. If the cost of all possible displacements of a pixel is high, the pixel is likely to be occluded in the following frame. The final so-called *ContinualFlow* architecture achieves better results than the standard PWC-Net on both Sintel and KITTI 2015.

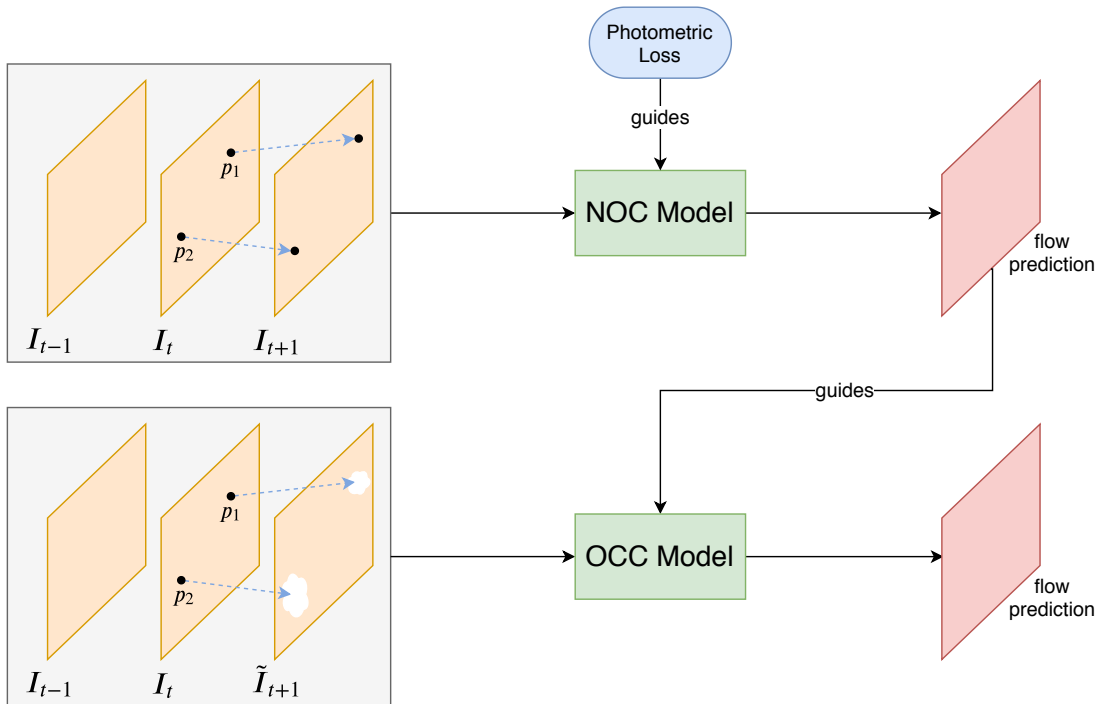
### 3.3.3 SelfFlow

At time of writing, SelfFlow [LLKX19] is the best performing estimation method on the Sintel final pass and produces state-of-the-art estimations on the KITTI benchmarks. Liu *et al.* base their approach on PWC-Net but extend it to three consecutive input frames  $I_{t-1}$ ,  $I_t$  and  $I_{t+1}$ , with  $I_t$  being the reference frame. In addition to the forward flow  $w_{t \rightarrow t+1}^k$  from  $t$  to  $t+1$  and the forward cost volume  $\text{cost}_{t \rightarrow t+1}^k$ , a backward flow  $w_{t \rightarrow t-1}^k$  from  $t$  to  $t-1$  and a backwards cost volume  $\text{cost}_{t \rightarrow t-1}^k$  is computed at each pyramid level  $k$ . Additionally, by swapping images  $I_t$  and  $I_{t+1}$  as input, a backward flow  $w_{t+1 \rightarrow t}$  can be estimated using the same network architecture. The estimated forward and backward flows can then be used to generate an occlusion map  $O$  via forward-backward (or *bi-directional*) consistency checking. More precisely, the reversed forward flow can be computed as follows:

$$\tilde{w}_{t \rightarrow t+1} = w_{t+1 \rightarrow t}(\mathbf{x} + w_{t \rightarrow t+1}(\mathbf{x})), \quad (3.9)$$

where  $x$  is the pixel index. If  $|w_{t \rightarrow t+1}(\mathbf{x}) + \tilde{w}_{t \rightarrow t+1}(\mathbf{x})|^2$  exceeds a certain threshold, the pixel  $\mathbf{x}$  is considered occluded.

The general idea of the approach is to train two CNNs. The first network is only trained on non-occluded regions (which are determined by the occlusion map). For the second network, random noise is generated in image areas to simulate additional occlusions. Due to the annotations learned



**Figure 3.3:** Illustration of SelfFlow’s two network architecture [LLKX19]. After initial unsupervised training of the NOC-model, the frame  $I_{t+1}$  is locally perturbed with random noise, resulting in  $\tilde{I}_{t+1}$  which is then used as an input to the OCC-Model. The OCC-model training is guided by the flow estimation from the NOC-model.

by the first network for these regions, the second network can then still learn how to estimate flow for occluded regions without supervision. Figure 3.3 illustrates this idea. Each of the two CNNs is based on the (extended) PWC architecture. The first CNN, called NOC-Model, is trained on all non-occluded regions, using a photometric loss between the first image and the warped second image (using the estimated flow). The loss is set to zero at occluded pixels. This corresponds to the basic idea of unsupervised optical flow estimation which works fine for non-occluded regions. The second CNN, called OCC-Model, is then additionally trained on the synthetically occluded pixels, i.e. with  $\tilde{I}_{t+1}$ . Here, the learning is guided by the flow that was estimated with the NOC-Model.

The result is an unsupervised learning approach that surpasses all previously published unsupervised methods on all major benchmarks and which even outperforms FlowNet and SPyNet. To achieve even better results, however, Liu *et al.* finally fine-tune their model on real-world annotated data, i.e. by employing a supervised learning step (albeit on small datasets). In doing so, they achieve the highest accuracy on the Sintel benchmark among all published methods and state-of-the-art results for KITTI 2012 and KITTI 2015. Liu *et al.* hence show that unsupervised learning can be feasible for optical flow and that large synthetic training datasets are not always necessary to produce state-of-the-art estimations.

### 3.4 ProFlow

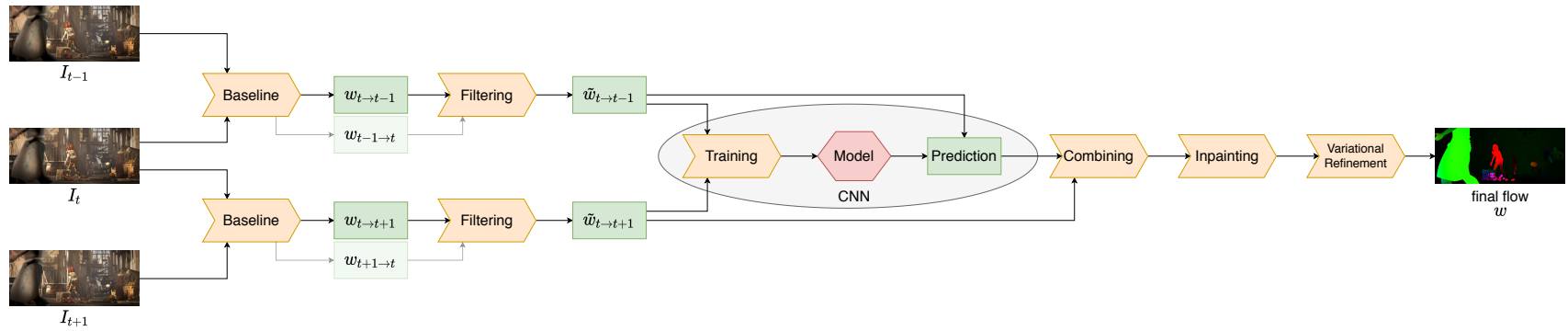
Maurer and Bruhn [MB18] propose a unique self-supervised approach with *ProFlow* that vastly differs from the above methods as it combines a classical, partially variational, estimation with a CNN. More precisely, ProFlow embeds a self-supervised network into a classical optical flow pipeline and as such is a partially-learning based hybrid. ProFlow learns online, i.e. during the estimation, and does not require labeled input data or supervised fine-tuning like the more recent SelfFlow does. In the following section, an overview regarding the general idea of ProFlow is given, followed by a more in-depth explanation of the architecture.

#### 3.4.1 Overview

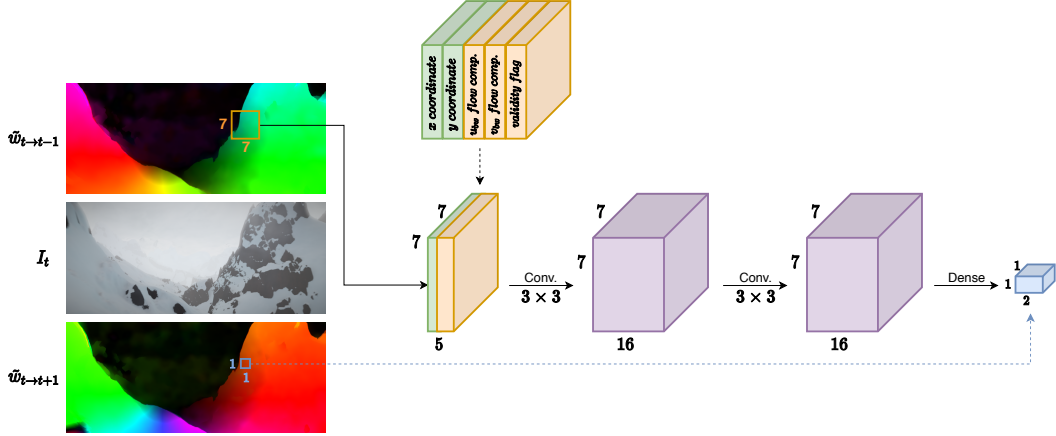
Figure 3.4 outlines the architecture as proposed by Maurer and Bruhn. ProFlow considers image triplets, that is, frames  $I_{t-1}$ ,  $I_t$  and  $I_{t+1}$ , with  $I_t$  being the reference frame. From these images, forward and backward flows are computed, i.e.  $w_{t \rightarrow t+1}$  and  $w_{t \rightarrow t-1}$ , respectively. Both flow fields are estimated with a conventional (partially variational) approach which serves as the baseline. Following that flow estimation, occluded image regions are filtered via a bi-directional consistency check. Therefore,  $w_{t+1 \rightarrow t}$  and  $w_{t-1 \rightarrow t}$  also need to be estimated with the baseline approach. Only regions where both the forward and backward flow remain after filtering, are then used to train a CNN. More precisely, the network is trained with the backward flow as input and the corresponding forward flow as the label. The trained network is then used to predict a forward flow field from the (filtered) backward flow field, including those regions that are valid in the filtered backward flow field but not in the filtered forward flow field. Thus, the predicted flow field of the network is used in regions that were occluded between  $I_t$  and  $I_{t+1}$ , but not occluded between  $I_t$  and  $I_{t-1}$ . The baseline flow field is then combined with the prediction of the network. Finally, the remaining regions (i.e. regions that are occluded in both the backward flow field and the forward flow field) are filled in by applying inpainting and improved via variational refinement. The final flow field is significantly more accurate than the estimation produced by the baseline.

#### 3.4.2 Architecture

Maurer and Bruhn chose EpicFlow’s [RWHS15] four step pipeline as the baseline. EpicFlow (Edge-Preserving Interpolation of Correspondences) is one of the most accurate variational approach on the Sintel final pass while being relatively fast compared to most other state-of-the-art classical methods. The four steps are (i) matching, (ii) outlier filtering, (iii) inpainting and (iv) variational refinement. However, apart from the bi-directional consistency check for the outlier filtering in the original EpicFlow approach, Maurer and Bruhn substituted each step with a more recent and more promising approach. The matching step is implemented with PatchMatch introduced by Hu *et al.* [HSL16], a coarse-to-fine approach that handles large displacements well. Inpainting was substituted with the robust interpolation technique (RIC) of Hu *et al.* [HLS17] and, finally, the refinement step was implemented via the order-adaptive illumination-aware refinement (OIR) proposed by Maurer *et al.* [MSB17].



**Figure 3.4:** Overview of the ProFlow pipeline as proposed by Maurer and Bruhn [MB18]. Note the reverse directions  $w_{t-1 \rightarrow t}$  and  $w_{t \rightarrow t+1}$  of the backward and forward flows, which are necessary for the following bi-directional filtering step.  $\tilde{w}_{t \rightarrow t-1}$  and  $\tilde{w}_{t \rightarrow t+1}$  denote the filtered flow fields.



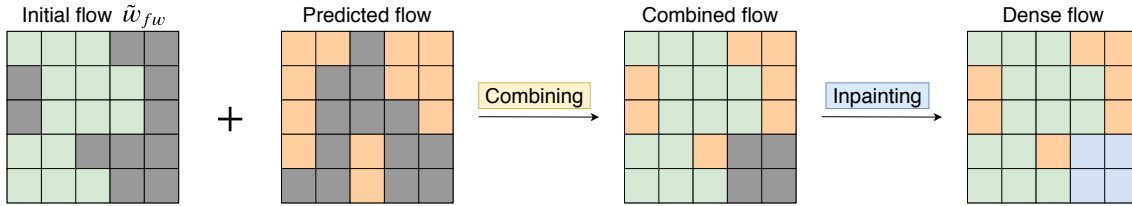
**Figure 3.5:** Architecture of the CNN in ProFlow [MB18]. The left side illustrates the training data extraction from the filtered baseline flows  $\tilde{w}_{t \rightarrow t-1}$  and  $\tilde{w}_{t \rightarrow t+1}$ .

After computing the initial flow fields  $w_{t \rightarrow t+1}$  and  $w_{t \rightarrow t-1}$  with the baseline, outliers are filtered via the bi-directional consistency check (see also Equation 3.9). Note that this is fundamentally the same as the filtering during the baseline estimation, but now the consistency check is applied after the final refinement step (instead of after the first matching step). Therefore, the whole baseline also has to be applied to the reverse directions, i.e. from  $I_{t+1}$  to  $I_t$  and from  $I_{t-1}$  to  $I_t$ , respectively. Only pixel indices where both the corresponding forward and backward flow are valid after filtering, are considered as potential candidates for the subsequent network training.

The training sample candidates are then sub-sampled via an equidistant grid spacing of 10 pixels. For each of those samples, a  $7 \times 7$  patch with the sample at its center is extracted from the backward flow. The flow patch is then stacked with the positional index of each pixel and a corresponding validity flag. More precisely, each training sample is of dimensions  $7 \times 7 \times 5$ , the former two dimensions ( $7 \times 7$ ) being the patch size, and the latter (5) resulting from backward flow components  $u_{t \rightarrow t-1}$  and  $v_{t \rightarrow t-1}$ , a validity flag  $c \in \{0, 1\}$  indicating if the location was valid after filtering, and the pixel location components  $x$  and  $y$  (see Figure 3.5). The latter allows the trained model to make predictions depending on the pixel location, which is particularly important for independently moving objects and non-rigid deformations in the input sequence.

Finally, the output (label) for each training sample is given by  $u_{t \rightarrow t+1}$  and  $v_{t \rightarrow t+1}$ , i.e. the corresponding forward flow  $w_{t \rightarrow t+1}$ . Based on these training samples, a CNN is then trained such that the model is able to predict a forward flow from the given backward flow. Note that all steps are processed automatically. That is, training samples and corresponding outputs are extracted without human interference, making the network training unsupervised.

The CNN is depicted in Figure 3.5. The network architecture is kept fairly simple since a new model has to be learned for each frame of the sequence. The CNN consists of two convolutional layers followed by a fully connected layer with a 2-dimensional output. Thus, the network predicts the  $u_{t \rightarrow t+1}$  and  $v_{t \rightarrow t+1}$  components of the forward flow corresponding to the center pixel of the  $7 \times 7$  input patch. The convolutional layers each have 16 kernels with a window size of  $3 \times 3$ .



**Figure 3.6:** Combination of the initial (filtered) baseline flow and the predicted flow, followed by the inpainting step.

For the loss function, the absolute difference between the predicted forward flow and the actual forward flow (given by the baseline as discussed above) is minimized, i.e.

$$L = |w_{t \rightarrow t+1}^b - w_{t \rightarrow t+1}^e|, \quad (3.10)$$

where  $w_{t \rightarrow t+1}^b$  is the forward flow as given by the baseline (which serves as the ground truth) and  $w_{t \rightarrow t+1}^e$  is the estimation from the CNN. Note that this coincides with the *average endpoint error* (see Equation 2.5).

The network is implemented in TensorFlow [ABC+16], with ReLUs as non-linearities [NH10] and Adam [KB15] as the chosen optimizer. The network was trained over 4000 steps, with the initial learning rate at 0.01 and an exponential decay of base 0.8 every 200 steps.

After training the network, it can now be used to predict a forward flow from the filtered backward flow. In particular, the CNN estimates a forward flow in those areas where a (baseline) backward flow but no (baseline) forward flow is available after the filtering step. Figure 3.6 illustrates how the forward flow predicted by the network and the initial forward flow from the baseline are selected during ProFlow’s combination step. As shown, the resulting flow field is sparse due to locations where neither a forward nor a backward flow are available. Thus, as a final step, inpainting followed by variational refinement is applied and analogously to the final two steps of the baseline, RIC [HLS17] and OIR [MSB17] are used, respectively.

### 3.4.3 Results

The baseline itself achieves fairly good results on all major benchmarks. According to Maurer and Bruhn, only the DF + OIR approach [MSB17] achieves better results, but is significantly more time-consuming as it employs the more complex variational method called *DiscreteFlow* as introduced by Menze *et al.* [MHG15]. However, ProFlow performs considerably better on all benchmarks than the pure baseline approach, achieving state-of-the-art results. On Sintel, in particular, ProFlow ranked first on the final render pass and second on the clean render pass at the time of Maurer and Bruhn’s writing. Experiments show particularly good results in occluded regions of the Sintel benchmark. Maurer and Bruhn thus show the benefits of handling occlusions with a CNN and additional temporal information. While ProFlow does not achieve interactive frame rates due to both the (partially) variational baseline and the fact that the model is learned online for every image in the sequence, it is unsupervised and neither requires large training datasets nor human

### 3 Related Work

---

assistance. ProFlow serves as the foundation of this thesis, whereby the following chapters focus on improving the flow estimation by incorporating even more temporal information, i.e. more than three frames.



## 4 General Approach: ProFlowS

The following work focuses on adapting the ProFlow prediction step such that more than three frames are used. The general idea is that with multiple frames more temporal information is available which, in turn, should allow for better optical flow predictions. In the standard three-frame setting, the CNN predicts a corresponding forward flow to the given backward flow. In the following chapter, we will instead see how we can modify this approach such that a forward flow estimation from *multiple* backward flows is possible. To make the necessary multi-frame modifications easier, a few general changes were made to the ProFlow CNN architecture which are presented in this chapter.

First, an overview of the new CNN architecture is given including layer modifications and a slightly different loss function. Following that, a brief evaluation of the CNN's performance for the standard three-frame setting is presented. Finally, challenges that present itself in a multi-frame setting are discussed. We will refer to this new model as *ProFlowS* in the following chapters.

### 4.1 CNN Architecture

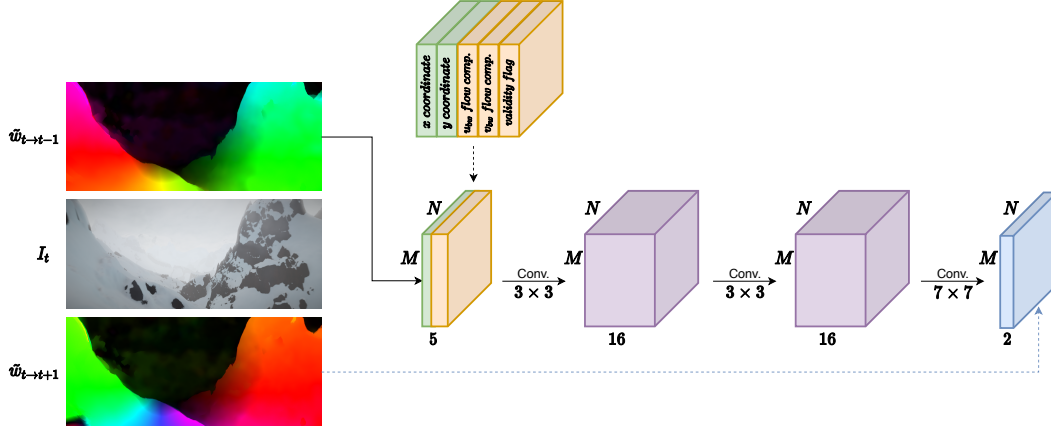
We will now look at some general modifications that were made to the ProFlow network architecture to simplify the original ProFlow approach. This more unified model allows for easier extensions and, in particular, comparisons with the multi-frame adaptations in the following chapters.

For now, let us assume the standard setting with *three* consecutive frames  $I_{t-1}$ ,  $I_t$  and  $I_{t+1}$ . Modifications of varying degrees were made to (i) the input and hidden layers, (ii) the final output layer and (iii) the loss function. We will introduce and discuss those in the following.

#### 4.1.1 Input and Hidden Layers

As presented in Section 3.4, the original ProFlow CNN selects potential training candidates by sub-sampling the filtered backward flow. More precisely,  $7 \times 7$  patches with a grid spacing of 10 pixels are extracted, each serving as a single training input labeled with the corresponding forward flow of the center pixel.

The here adapted CNN instead takes the whole image size as a single input. That is, only one training sample is selected consisting of the whole (filtered) backward flow, labeled with the whole (filtered) forward flow. As before, flow components  $u$  and  $v$  are stacked with validity flag  $c$  and the pixel coordinates  $x$  and  $y$ . Therefore, the input of the new network is of shape  $N \times M \times 5$ , where  $N \times M$  is the size of the image. Figure 4.1 depicts the new CNN architecture (cf. Figure 3.5 for a comparison).



**Figure 4.1:** The new ProFlowS network architecture as used in this work. Note the differences to the original architecture in Figure 3.5.

Following the original architecture, the two hidden layers are convolutional layers, each with 16 kernels of size  $3 \times 3$ . Since, here, the network input consists of the whole backward flow at once, however, each of these convolutional layers is significantly larger, producing an output of  $N \times M \times 16$  which then serves as the input to the next layer.

Note that with this configuration the network potentially learns from all positions, including those where forward or backward flows are invalid, which is clearly undesirable. This is explicitly addressed in Section 4.1.3 with a modified loss function.

#### 4.1.2 Output Layer

The original ProFlow network architecture implements the output layer as a fully connected layer with a 2-dimensional output. That is, each  $7 \times 7$  input patch is mapped to a 2-dimensional output representing the estimated flow components  $u_{t \rightarrow t+1}$  and  $v_{t \rightarrow t+1}$  corresponding to the center pixel.

As elaborated above, the input now consists of the whole backward flow. In other words, the input is an  $N \times M$  patch. Therefore, a fully connected layer with a single 2-dimensional output does not make sense here, as it would map the whole backward flow field to a single forward flow estimate. Similarly, a fully connected layer with output dimensions  $N \times M \times 2$  would not only vastly increase the model size and computational cost, but also learn each flow estimate with respect to the whole image, instead of only considering a local neighborhood. Since, typically, motion is locally coherent, keeping a configuration akin to the original patch-wise input scheme seems reasonable.

Therefore, the last layer is implemented as a convolutional layer consisting of two kernels with a kernel size of  $7 \times 7$  (see Figure 4.1). The such produced 2-dimensional output at position  $(x, y)$  is analogous to the output of the fully connected layer used in the standard ProFlow architecture (given an input patch with  $(x, y)$  at its center).

	Sintel Clean	Sintel Final	KITTI 2015	
	AEE	AEE	AEE	BP
baseline	1.94	3.78	6.64	18.59
ProFlowS	2.23	4.00	5.49	18.00

**Table 4.1:** Results of the ProFlowS architecture and the baseline approach on Sintel clean, Sintel final and KITTI 2015.

### 4.1.3 Loss Function

Following the original ProFlow architecture, the *average endpoint error* (AEE) is used (cf. Equations 2.5 and 3.10) as the loss function. However, since the network is trained with the whole backward flow, this also includes occluded regions, that is, regions which were considered invalid during the filtering step. Evidently, regions where either the forward or backward flow was previously filtered should not be used as training data. To account for those regions, the loss function has to be slightly adjusted. More precisely, the endpoint error is set to zero for each such occluded pixel.

Let  $\vec{w}^b$  be short for the filtered baseline forward flow, i.e.  $\vec{w}^b = w_{t \rightarrow t+1}^b$ , and  $\overleftarrow{w}^b$  short for the filtered baseline backward flow, i.e.  $\overleftarrow{w}^b = w_{t \rightarrow t-1}^b$ . Similarly, let  $w^e$  be the estimated forward flow from the network, i.e.  $w^e = w_{t \rightarrow t+1}^e$ . The final loss is then

$$L_{AEE}(\vec{w}^b, w^e) = \frac{1}{\tilde{N}\tilde{M}} \sum_{x=1}^N \sum_{y=1}^M \sqrt{\left(\vec{u}_{x,y}^b - u_{x,y}^e\right)^2 + \left(\vec{v}_{x,y}^b - v_{x,y}^e\right)^2} \cdot \mathbf{1}_{\left(\vec{c}_{x,y}^b=1 \wedge \overleftarrow{c}_{x,y}^b=1\right)}, \quad (4.1)$$

where  $\vec{c}_{x,y}^b$  is the validity flag of the forward flow at position  $(x, y)$ ,  $\overleftarrow{c}_{x,y}^b$  is the validity flag of the backward flow at position  $(x, y)$ , and  $\mathbf{1}$  is the indicator function. Additionally,  $\tilde{N}\tilde{M}$  is the number of pixels where both forward and backward flow are valid. In other words, the loss consists of the average endpoint error for those regions that are valid in both the filtered forward flow and filtered backward flow.

Finally, the whole network is re-implemented in TensorFlow 2.0 [ABC+16] using the Keras API. As in the original ProFlow, the CNN is built with ReLUs as non-linearities. Adam was chosen as the optimizer with standard parameters as suggested in [KB15]. It is trained over 150 epochs, unless specified otherwise. Note, however, that in the given architecture, one epoch corresponds to a single sample being passed through the network. More precisely, since we have exactly one sample consisting of the whole backward flow field, passing it through the network once corresponds to one iteration which also amounts to one entire epoch.

## 4.2 Results

The next chapter presents five different approaches that further extend the ProFlowS architecture such that multiple frames can be used for prediction. In order to reliably evaluate these models, the performance of the here introduced ProFlowS architecture serves as the basis for comparison. Table

4.1 shows the average endpoint error (AEE) for three major benchmarks, Sintel clean, Sintel final and KITTI 2015 (cf. Section 3.1). Additionally, for KITTI 2015 the bad pixel error (BP) is given, as that is the preferred error measure for the KITTI benchmarks. Both the results for ProFlowS and for the baseline are presented.

Evidently, the baseline itself performs better than ProFlowS on both Sintel benchmarks. While this suggests that the filtered flow values of the baseline are better than the prediction of ProFlowS, the here presented architecture merely serves us as a comparison base for multi-frame approaches. Since (a) the performance of the baseline is a constant in all following models and (b) we are mostly interested in a potential improvement by leveraging additional temporal information in the prediction step of the ProFlow pipeline, it suffices to compare the results of different multi-frame approaches to the ProFlowS prediction. It should be noted, however, that in some cases we will have to slightly adjust the models to ensure that they can be evaluated solely on their multi-frame aspects. This will be addressed later. Also note that when we skip the prediction step of the ProFlow pipeline, the estimation slightly improves. That is, filtering the baseline and applying inpainting to all filtered positions, followed by variational refinement, yields, for instance, an AEE of 1.93 on Sintel clean, further suggesting that inpainting performs better than the ProFlowS network prediction.

### 4.3 Multi-Frame Challenges

In order to leverage the temporal information of an image sequence, a few challenges have to be faced. The main difficulty resides in having to coincide the coordinate systems of each input frame. More precisely, in order to preserve meaningful and locally variant estimations, all inputs to the network have to be locally correlated to the reference frame. ProFlow as presented in Chapter 3.4 solves this problem by correlating the backward flow to the forward flow. By doing so, the reference coordinate system in both flows is that of the reference frame  $I_t$ . Since the approaches presented in this work attempt to incorporate more than three frames of a sequence, the made modifications have to ensure that the pixel coordinates of each input still coincide with the reference frame.

The other major challenge concerns the way training samples are selected. In ProFlowS as well as the original ProFlow architecture, a position  $(x, y)$  is considered a valid training sample if both the backward flow  $w_{t \rightarrow t-1}(x, y)$  and the corresponding forward flow  $w_{t \rightarrow t+1}$  are valid. With multiple frames and, therefore, multiple backward flows at  $(x, y)$  with potentially different validity flags, the question is in which cases the position should be considered by the loss function during training. Similarly, with multiple backward flows, ProFlow's combination step (Figure 3.6) offers multiple variations as we will see in the next chapter. Both, the selection of positions used for training and the combination step handling the resulting predictions, are further addressed in the following chapter and explored experimentally in Chapter 6 using different approaches.

## 5 Multi-Frame Modifications

In this chapter, five different methods that extend the ProFlowS CNN architecture are introduced. Four of those approaches primarily try to leverage more temporal information of the input image sequence, while the final approach focuses on counteracting the difficulties that, in particular, arise when multiple frames are used for optical flow prediction.

First, an overview of the notation for the upcoming modifications is given, followed by an introduction of the new models. Each model is then evaluated in the next chapter on the Sintel and KITTI 2015 datasets with an emphasis on the Sintel clean pass (cf Section 3.1).

### 5.1 Notation

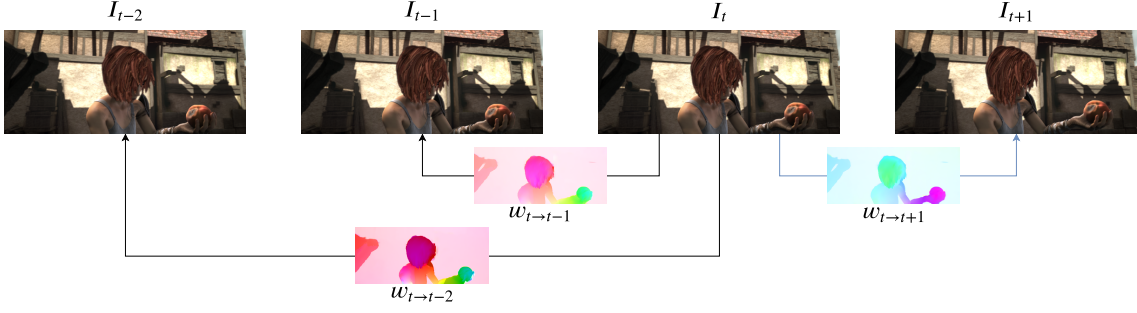
Given  $h$  consecutive frames  $I_{t-(h-2)}, I_{t-(h-3)}, \dots, I_{t-1}, I_t, I_{t+1}$ , the goal is to estimate the forward flow from  $I_t$  to  $I_{t+1}$ . That is,  $I_t$  is the reference frame of the sequence. While the standard ProFlow approach takes three consecutive frames as inputs, i.e.  $I_{t-1}, I_t, I_{t+1}$ , the following models, typically, incorporate four or more frames (e.g. from  $I_{t-3}$  to  $I_{t+1}$ ). The amount of frames from which the forward flow is estimated is denoted by the so-called *input history*  $h \in \mathbb{N}$ . For instance, in the three frame setting, we have an input history of  $h = 3$ .

Let  $w_{i \rightarrow j}$  denote the flow from  $I_i$  to  $I_j$ , e.g.,  $w_{t \rightarrow t+1}$  denotes the forward flow from  $I_t$  to  $I_{t+1}$  and  $w_{t \rightarrow t-1}$  denotes the backward flow from  $I_t$  to  $I_{t-1}$ . Thus,  $w_{j \rightarrow i}(\mathbf{x})$  is a 2-dimensional vector consisting of flow components  $u_{i \rightarrow j}(\mathbf{x})$  and  $v_{i \rightarrow j}(\mathbf{x})$  at pixel position  $\mathbf{x}$ . If not specified otherwise,  $w_{i \rightarrow j}$  denotes the *baseline* flow. Conversely,  $w_{i \rightarrow j}^e$  denotes the estimation from a network. Typically, we are only interested in  $w_{t \rightarrow t+1}^e$  which is evaluated against the ground truth provided by the respective dataset.

Given a flow  $w_{i \rightarrow j}$ , an additional validity map  $c_{i \rightarrow j}$  is appended, where the binary flag  $c_{i \rightarrow j}(\mathbf{x})$  indicates the validity of the flow at position  $\mathbf{x}$ . Assuming the original three-frame setting, the forward flow is predicted from a single backward flow and the respective validity map. In short, we write

$$(w_{t \rightarrow t-1}, c_{t \rightarrow t-1}) \xrightarrow{\text{predict}} w_{t \rightarrow t+1}^e. \quad (5.1)$$

Note that, by default, the pixel coordinates are provided as an additional input (also see Section 3.4.2). To allow for locally variant estimations, the pixel positions are always part of the input in all modifications. Thus, for the sake of brevity, these are generally omitted here.



**Figure 5.1:** Illustration of the new input to the network for an input history of 4. The two backward flows  $w_{t \rightarrow t-2}$  and  $w_{t \rightarrow t-1}$  are concatenated and serve as a single input to the CNN which, in turn, predicts the forward flow  $w_{t \rightarrow t+1}$ .

## 5.2 Multiple Backward Flows

The most intuitive approach is to stack multiple backward flows and use them as a single input for the convolutional neural network. In other words, the here presented CNN predicts the desired forward flow from two or more backward flows. Thus, with input history  $h$  we have

$$[(w_{t \rightarrow t-1}, c_{t \rightarrow t-1}), (w_{t \rightarrow t-2}, c_{t \rightarrow t-2}), \dots, (w_{t \rightarrow t-(h-2)}, c_{t \rightarrow t-(h-2)})] \xrightarrow{\text{predict}} w_{t \rightarrow t+1}^e. \quad (5.2)$$

See Equation 5.1 for a comparison to the standard ProFlow setting. Note that every backward flow  $w_{t \rightarrow j}$  is computed with respect to the reference frame. In doing so, the coordinate system of each backward flow is always aligned with the reference frame.

Analogously to the first backward flow, all additional flows are computed with the baseline and appended with a validity map obtained via a bi-directional consistency check. This also means that the reverse direction for each of those backward flows has to be computed with the baseline, e.g. the flow  $w_{t-2 \rightarrow t}$  is needed to filter  $w_{t \rightarrow t-2}$  and so on. Figure 5.1 illustrates the concept of this approach for  $h = 4$ . The two backward flows serve, together with the validity map and pixel coordinates, as the new combined input to the CNN which then predicts the corresponding forward flow. By simply concatenating the backward flows, the network has to learn meaningful features on its own.

To accommodate for the additional input to the CNN, the input layer shape has to be adjusted. Assuming four input frames, for instance, the input shape becomes  $N \times M \times 8$ , consisting of the first backward flow components  $(u_{t \rightarrow t-1}, v_{t \rightarrow t-1})$  and validity flag  $c_{t \rightarrow t-1}$ , the second backward flow components  $(u_{t \rightarrow t-2}, v_{t \rightarrow t-2})$  and validity flag  $c_{t \rightarrow t-2}$ , and the image coordinates  $(x, y)$ . In general, the input layer size is  $N \times M \times 3 \times (h - 2) + 2$ : 3 dimensions for each backward flow (flow components and validity flag) plus the  $x$  and  $y$  coordinates for each pixel.

While these multi-frame modifications appear to be straightforward, there are two parts of the ProFlowS architecture where different variations are feasible that can result in greatly varying outcomes. These are namely (a) the loss function used for training the network and (b) the combination step of the ProFlow pipeline where the predicted flow estimates are combined with the baseline flow. For each, two major alternatives are viable for examination. The possible variations for both parts are explained in the following, with results being shown in the next chapter.

### 5.2.1 Loss Function

Section 4.1.3 introduced a new loss function for ProFlowS that makes sure that only pixels are considered for training the network where both forward and backward flow are valid. Since we have multiple backward flows now, we also need to incorporate the validity of each backward flow into the loss function. Therefore, we define a new validity flag  $c^{\text{all}}$  that meaningfully combines the validity flags of all flows such that the new loss  $\hat{L}_{AEE}$  reads as follows:

$$\hat{L}_{AEE}(w^b, w^e) = \frac{1}{\tilde{N}\tilde{M}} \sum_{x=1}^N \sum_{y=1}^M \sqrt{(u_{x,y}^b - u_{x,y}^e)^2 + (v_{x,y}^b - v_{x,y}^e)^2} \cdot \mathbf{1}_{(c_{x,y}^{\text{all}}=1)}, \quad (5.3)$$

where  $w^b$  is the baseline forward flow and  $w^e$  is the estimated forward flow. Note that in ProFlowS  $c_{i,j}^{\text{all}} = \vec{c}_{i,j}^b \cdot \overleftarrow{c}_{i,j}^b$ , i.e. only positions where both the baseline forward and backward flow are valid are taking into account (cf. Equation 4.1).

With multiple backward flows, there are now two major possibilities of combining the validity flags of the backward flows into a single value: *conjunctively* or *disjunctively*. Both methods are explained in the following.

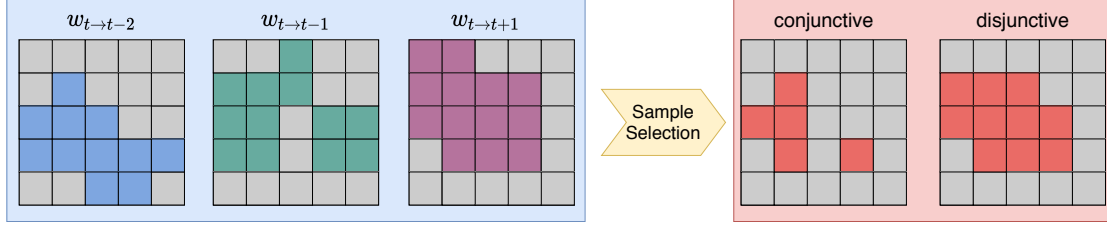
#### Conjunctive Validity Flag

In the conjunctive setting, a pixel at position  $(x, y)$  is considered by the loss function if (i) the forward flow  $w_{t \rightarrow t+1}(x, y)$  is valid and (ii) *all* backward flows  $w_{t \rightarrow t-n}(x, y)$ ,  $n \in 1, \dots, h-2$  are valid. That is,

$$c_c^{\text{all}}(x, y) = \begin{cases} 1, & \text{if } c_{t \rightarrow t+1}(x, y) = 1 \\ & \wedge c_{t \rightarrow t-1}(x, y) = 1 \wedge \dots \wedge c_{t \rightarrow t-(h-2)}(x, y) = 1 \\ 0, & \text{otherwise.} \end{cases} \quad (5.4)$$

Note that the conjunction of the (backward) validity flags can also be expressed with a *min* operator of all flags at each respective position.

While this reduces the amount of pixels used for evaluating the loss, it ensures that the network learns only from positions where information is available at every time step and, thus, potentially increases the quality of predictions. Contrarily, if the number of valid pixels decreases by too much, the network might not be able to reliably learn the underlying motion due to a lack of sufficient training data.



**Figure 5.2:** Illustration of the difference between a conjunctive and disjunctive validity flag.

### Disjunctive Validity Flag

The alternative is to combine the backward validity flags *disjunctively*. Here, a pixel at position  $(x, y)$  is considered by the loss function if (i) the corresponding forward flow is valid and (ii) *at least one* corresponding backward flow is valid. More precisely,

$$c_d^{\text{all}}(x, y) = \begin{cases} 1, & \text{if } c_{t \rightarrow t+1}(x, y) = 1 \\ & \wedge (c_{t \rightarrow t-1}(x, y) = 1 \vee \dots \vee c_{t \rightarrow t-h-2}(x, y) = 1) \\ 0, & \text{otherwise} \end{cases} \quad (5.5)$$

Note that the disjunction of the (backward) validity flags can also be expressed with a *max* operator of all flags at each respective position.

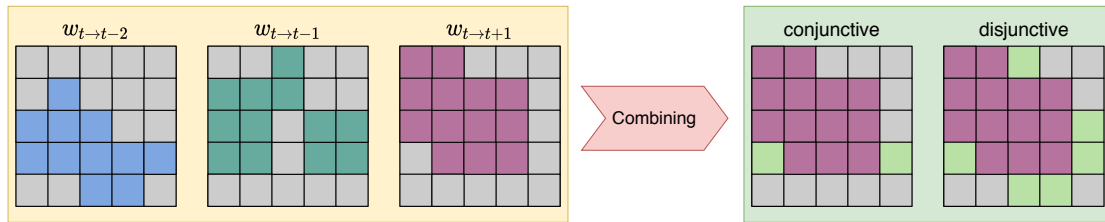
While this increases the amount of pixels considered by the loss function, it also potentially incorporates positions where one or more backward flows were evaluated as invalid during the filtering step. The network then learns from partially incorrect backward flow values which could decrease the quality of the prediction. However, since the network receives the validity flags of each backward flow as well, it might be able to recognize and learn the partial backward flow sparsity. Generally, the question is if the increased number of positions considered during training outweighs the outliers introduced in the form of invalid flows. Figure 5.2 illustrates the difference between the *conjunctive* and *disjunctive* validity flag setting for an input history of 4.

### 5.2.2 Combination Step

After training the model, its prediction has to be meaningfully combined with the baseline flow. In the original architecture, the prediction is taken at every position where the backward flow was valid after filtering but the forward flow was not. More precisely, the final flow consists of three parts:

1. The forward flow from the baseline is taken at all positions where it is considered valid after the filtering step.
2. For positions where the forward flow is invalid but the backward flow is valid, the network estimation is taken.
3. All other positions are filled in via inpainting.





**Figure 5.3:** Illustration of the differences between a conjunctive and disjunctive combination step for  $h = 4$ . Violet pixels denote the baseline estimation, light green pixels show the additional positions gained through the prediction of the network. All other positions need to be filled in via inpainting.

In the multiple backward flow setting the second step extends to two possible choices. Similarly to the combined validity flag methods above, we can either view the backward flows *conjunctively* or *disjunctively*. That is, the prediction of the CNN is taken either at positions where *all* backward flows are valid after filtering (*conjunctive*) or, alternatively, where *at least one* backward flow is valid (*disjunctive*).

Figure 5.3 illustrates the difference between both combination steps for an input history of 4. Note that the number of positions used from the baseline stays the same for all multi-frame methods. With a conjunctive combination step, however, it is likely that the prediction is taken for less pixels than in the original three-frame ProFlow setting. This means that more positions have to be filled in via inpainting. Assuming the prediction to be more accurate than flow values gained by inpainting, it is likely that the overall accuracy of the final flow suffers in the conjunctive configuration. On the other hand, however, the positions where all backward flows are valid can be expected to have the best predictions which, in turn, could lead to better inpainting performance.

The disjunctive combination instead increases positions for prediction and, thus, has the higher potential for an overall performance gain if the prediction is good. However, due to estimating a forward flow at positions where one or more backward flows are invalid, the prediction at these positions might be less accurate.

More importantly, however, ProFlowS performs worse than just the baseline with inpainting on Sintel. More precisely, after the final variational refinement step, inpainted flow values are more accurate than the estimated values of ProFlowS on the Sintel dataset (see Section 4.2). Therefore, the conjunctive combination step as introduced above is not employed in the following sections, because decreasing the amount of positions where the prediction is taken likely increases the flow accuracy as more pixels are evaluated via the better performing inpainting step. Since we are interested in a comparison of the multi-frame setting versus the original three-frame setting, we have to limit the amount of pixels where inpainting is used. Otherwise, a possible performance increase could be attributed to the better performing inpainting step instead of a better performing prediction step.

Therefore, in the following, the conjunctive combination step is substituted by the original ProFlowS combination step. That is, a prediction is taken at all positions where the *first* backward flow, i.e.  $w_{t \rightarrow t-1}$ , is valid (and the forward flow is not). This means that the validity of all additional backward flows is disregarded and, instead, we have the same pixels being estimated by the network as in the original ProFlowS setting. Thus, in case the overall flow accuracy increases, it is solely due to the

additional temporal information contained in the multi-frame setting. We will see the performance of both combination steps as well as both validity flag methods in Section 6.1, by evaluating all possible combinations against each other.

### 5.3 Flow Differences

While the general approach in the previous section directly makes use of multiple estimated backward flows, the approach presented in the following focuses on flow *differences* between backward flows. The idea is that by computing the flow *increment* at each time step, we get a more expressive description of the underlying motion model.

More precisely, each flow increment corresponds to the *velocity* in the respective time step. By sequencing these velocities for multiple frames, we get a better representation of the *acceleration* contained in the image sequence. In contrast, stacking multiple backward flows merely gives a direct relation between distance (displacement) and time, which only works well for constant velocities. Thus, with flow differences, we can describe a more expressive (constant) acceleration model, i.e.

$$v = v_0 + a \cdot t, \quad (5.6)$$

where  $v$  is the velocity,  $v_0$  is the initial velocity,  $a$  is the acceleration and  $t$  is the time step. Note that in the special case of constant (linear) motion ( $a = 0$ ), we get  $v = v_0$ . In this case each flow increment would be identical.

In terms of flow prediction, the idea is that the CNN can then learn to predict the forward flow (velocity) from multiple backward velocities. More broadly speaking, the network might be able to learn the forward *velocity* due to the (implicit) *acceleration* contained in the backward flow increments. Or in other words, from the given past increments the network can learn the next increment in the sequence, which corresponds to the desired forward flow.

#### 5.3.1 Implementation Details

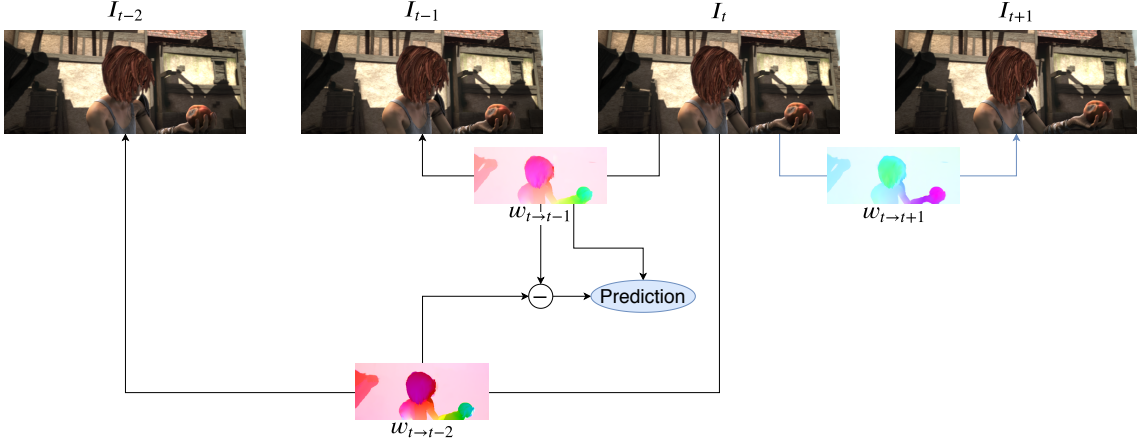
For each time step before  $t-1$ , the CNN is only given the respective flow increment. To get the increments, all backward flows are first computed with respect to the reference frame (as in the previous approach). Given an input history of  $h = 4$ , the flow increment between  $I_{t-1}$  and  $I_{t-2}$  is then

$$w'_{t-1 \rightarrow t-2} = w_{t \rightarrow t-2} - w_{t \rightarrow t-1}, \quad (5.7)$$

i.e. the difference between the two backward flows. The prediction then looks as follows:

$$[(w_{t \rightarrow t-1}, c_{t \rightarrow t-1}), (w'_{t-1 \rightarrow t-2}, c'_{t-1 \rightarrow t-2})] \xrightarrow{\text{predict}} w_{t \rightarrow t+1}^e, \quad (5.8)$$

where  $c'_{t-1 \rightarrow t-2}$  is the validity flag of the flow increment, which will be addressed later.



**Figure 5.4:** Illustration of the CNN for the flow difference model for four frames ( $h = 4$ ). The flow increment between time steps  $t-1$  and  $t-2$  is computed by subtracting the second backward flow from the first backward flow. The flow increment in addition to the first backward flow  $w_{t \rightarrow t-1}$  serve as input to the CNN.

Similarly, extending this to more than four frames yields

$$[(w_{t \rightarrow t-1}, c_{t \rightarrow t-1}), \dots, (w'_{t-(h-3) \rightarrow t-(h-2)}, c'_{t-(h-3) \rightarrow t-(h-2)})] \xrightarrow{\text{predict}} w_{t \rightarrow t+1}^e, \quad (5.9)$$

where  $h$  is the input history and the increment  $w'_{i \rightarrow i-1}$  can be computed as

$$w'_{i \rightarrow i-1} = w_{t \rightarrow i-1} - w_{t \rightarrow i}. \quad (5.10)$$

Additionally, note that since  $w_{t \rightarrow t}$  is a zero flow, it follows that

$$w_{t \rightarrow t-1} = w_{t \rightarrow t-1} - w_{t \rightarrow t} = w'_{t \rightarrow t-1}, \quad (5.11)$$

i.e. the first backward flow (from  $I_t$  to  $I_{t-1}$ ) can also be described as the first flow difference.

Figure 5.4 illustrates the approach with four frames. Note that in contrast to stacking increasingly large backward flows, the flow increments generally stay small and within the same margins which, in a sense, normalizes the network input. This, in turn, makes it potentially easier for the network to extract meaningful features as part of the learning process. Or, from another point of view, by standardizing the input data, it might be easier to detect inconsistencies in a particular backward flow. This, combined with the validity flag, could enable the network to better interpret the partial sparsity of the training data.

## 5.3.2 Difference Flow Validity

We have seen how to compute the flow increment  $w'_{i \rightarrow i-1}(\mathbf{x})$  from the baseline backward flows. However, we have yet to determine how to get the corresponding validity flag  $c'_{i \rightarrow i-1}(\mathbf{x})$ . Evidently, we need to append the validity map to each flow difference in order to only train the network on valid positions. Depending on the choice for  $c^{\text{all}}$  (conjunctive or disjunctive, cf. Section 5.2.1), multiple  $c'_{i \rightarrow i-1}(\mathbf{x})$  need to be combined accordingly. Thus, we need a validity flag for each flow difference.

Clearly, if both  $c_{t \rightarrow i} = 0$  and  $c_{t \rightarrow i-1} = 0$ , then  $c'_{i \rightarrow i-1}$  should be 0 as well. Similarly, if both validity flags are 1, then  $c'_{i \rightarrow i-1}$  should be 1 as well. However, if only one of the two backward flows is valid, the question arises whether the resulting flow difference could still be considered valid.

The most intuitive solution is to set  $c'_{i \rightarrow i-1}$  to 0 if either one of the backward flows is invalid:

$$c'_{i \rightarrow i-1} = c_{t \rightarrow i-1} \vee c_{t \rightarrow i}. \quad (5.12)$$

As a reminder, we are only talking about the validity of a single flow difference here. Equation 5.12 should not be confused with the *combined* validity flag  $c^{\text{all}}$  that is later used in the loss function as described in Section 5.2.1.

Taking this approach, however, means that the amount of valid flow differences is smaller than the amount of valid flows in either backward flow. More importantly, note that with the disjunctive validity flag variant described in Section 5.2.1, the loss for  $h = 4$  then considers all positions where (in addition to the forward flow) either  $w_{t \rightarrow t-1}$  or  $w'_{t-1 \rightarrow t-2}$  is valid. Since  $w'_{t-1 \rightarrow t-2}$  is only valid if both  $w_{t \rightarrow t-1}$  and  $w_{t \rightarrow t-2}$  are valid, this comes down to

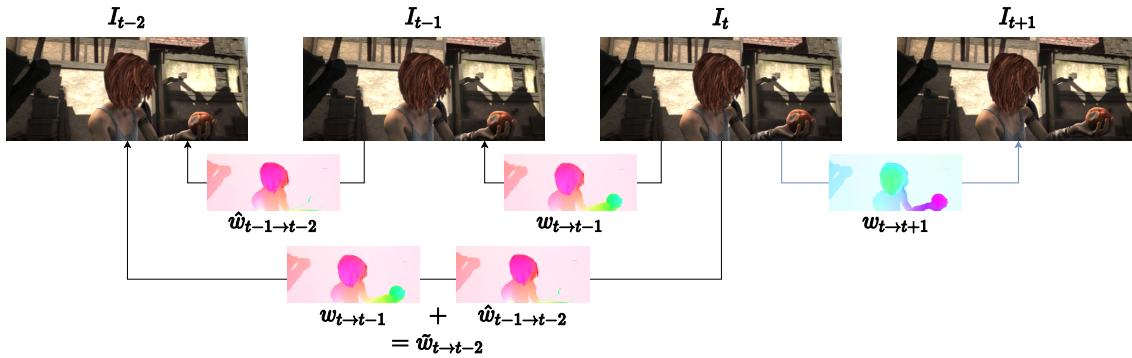
$$c_{t \rightarrow t-1} \vee c'_{t-1 \rightarrow t-2} \equiv c_{t \rightarrow t-1} \vee (c_{t \rightarrow t-1} \wedge c_{t \rightarrow t-2}) \equiv c_{t \rightarrow t-1}. \quad (5.13)$$

In other words, only the validity flag of the first backward flow is considered. Therefore, despite using more backward flows as well as the disjunctive validity flag method, we do not get the benefit of using additional positions to train the network (in contrast to the first approach).

One alternative would be to keep the validity flag of the backward flow with the larger time step, based on the assumption that it is more likely to be inaccurate. That is,

$$c'_{i \rightarrow i-1} = c_{t \rightarrow i-1}. \quad (5.14)$$

The benefit is that with the disjunctive validity flag variant as part of the loss function we have the same amount of positions the network can be trained on as in the first approach. The downside is that this likely introduces some additional errors as a given flow increment could be set to valid despite the shorter backward flow used in its computation being invalid. We will see which of the suggested technique yields the best results in Section 6.2.



**Figure 5.5:** Illustration of how the second backward flow is computed by warping and summing up (with four input frames). Note the difference to Figure 5.1.

## 5.4 Multiple Warped Backward Flows

Both previous approaches have a common problem: We first need to compute multiple baseline backward flows, each with respect to the reference frame. It is likely that backward flows over larger time steps exhibit more errors than that of a single time step. Thus, more invalid positions are introduced into the network training which might make reliably predictions more difficult.

Therefore, in the following approach only backward flows of a single time step are considered. More precisely, each backward flow is first computed pair-wise and then warped via bilinear interpolation such that the image coordinates spatially coincide with those in the reference frame. Thus, assuming that the baseline is generally more accurate for smaller time steps, using pair-wise backward flows should give us more valid flows to work with. The downside with this approach, however, is that the warping operation itself can introduce errors in the form of flow artifacts which we will see later in Section 6.3.

### 5.4.1 Implementation Details

Let us first assume a four-frame setting ( $h = 4$ ). Figure 5.5 illustrates the difference to the first two approaches (cf. Figures 5.1 and 5.4). Starting from the reference frame  $I_t$ , the first backward flow  $w_{t \rightarrow t-1}$  is computed as usual from  $I_t$  to  $I_{t-1}$ . For the previous frame ( $I_{t-2}$ ), however, first the flow  $w_{t-1 \rightarrow t-2}$  from  $I_{t-1}$  to  $I_{t-2}$  is computed with the baseline and filtered via the bi-directional consistency check, i.e. by also computing  $w_{t-2 \rightarrow t-1}$ . Following that, the flow is warped using the backward flow  $w_{t \rightarrow t-1}$ . More precisely, the warped backward flow is

$$\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x}) = w_{t-1 \rightarrow t-2}(\mathbf{x} + w_{t \rightarrow t-1}(\mathbf{x})), \quad (5.15)$$

where  $\mathbf{x}$  is the pixel coordinate and  $w_{t-1 \rightarrow t-2}$  is evaluated via bilinear interpolation. Finally, the new (warped) backward flow is obtained by combining the first backward flow and the warped previous backward flow. That is,

$$\begin{aligned}\tilde{w}_{t \rightarrow t-2}(\mathbf{x}) &= w_{t \rightarrow t-1}(\mathbf{x}) + \hat{w}_{t-1 \rightarrow t-2}(\mathbf{x}) \\ &= w_{t \rightarrow t-1}(\mathbf{x}) + w_{t-1 \rightarrow t-2}(\mathbf{x} + w_{t \rightarrow t-1}(\mathbf{x})).\end{aligned}\quad (5.16)$$

Therefore, we get an input of similar proportion to the first approach as described in Section 5.2. The main reason for summing up both flows instead of using  $\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x})$  directly, is that it can help reduce some of the downsides of the warping strategy. This will be addressed later in Section 6.3. For now, note that in doing so this approach is more in line with the first approach (Section 5.2) than the second approach (Section 5.3).

Finally, the CNN is trained with the first backward flow  $w_{t \rightarrow t-1}$  and the warped second backward flow  $\tilde{w}_{t \rightarrow t-2}$ , i.e.

$$[(w_{t \rightarrow t-1}, c_{t \rightarrow t-1}), (\tilde{w}_{t \rightarrow t-2}, \tilde{c}_{t \rightarrow t-2})] \xrightarrow{\text{predict}} w_{t \rightarrow t+1}^e. \quad (5.17)$$

In other words, the warped backward flows are concatenated and serve as a single input to the CNN (in addition to the respective validity maps), similar to the first approach. Generally, for more than four frames, the warped backward flow  $\tilde{w}_{t \rightarrow t-i}$  from frame  $I_t$  to  $I_{t-i}$  can be computed as follows:

$$\begin{aligned}\tilde{w}_{t \rightarrow t-i}(\mathbf{x}) &= \tilde{w}_{t \rightarrow t-(i-1)}(\mathbf{x}) + \hat{w}_{t-(i-1) \rightarrow t-i}(\mathbf{x}) \\ &= \tilde{w}_{t \rightarrow t-(i-1)}(\mathbf{x}) + w_{t-(i-1) \rightarrow t-i}(\mathbf{x} + \tilde{w}_{t \rightarrow t-(i-1)}(\mathbf{x})),\end{aligned}\quad (5.18)$$

with  $\tilde{w}_{t \rightarrow t-1}(\mathbf{x}) = w_{t \rightarrow t-1}(\mathbf{x})$ . The CNN then looks as follows:

$$[(w_{t \rightarrow t-1}, c_{t \rightarrow t-1}), \dots, (\tilde{w}_{t \rightarrow t-(h-2)}, \tilde{c}_{t \rightarrow t-(h-2)})] \xrightarrow{\text{predict}} w_{t \rightarrow t+1}^e. \quad (5.19)$$

As with the previous two approaches, we have the choice of either using the disjunctive or conjunctive combined validity flag in the loss function. To do so, however, we need to first address the validity of a warped flow, i.e.  $\tilde{c}_{t \rightarrow t-i}$ .

#### 5.4.2 Warped Flow Validity

Similar to the previous approach, we are computing new inputs (here: warped backward flows) from baseline flows. Therefore, we need to adjust the validity of a warped flow depending on the flow validities used in its computation.

Intuitively, the validity flag  $\hat{c}_{t-1 \rightarrow t-2}(\mathbf{x})$ , for instance, can also be evaluated via bilinear interpolation with respect to  $c_{t-1 \rightarrow t-2}$ , after which it is rounded to 0 or 1. More precisely,

$$\hat{c}_{t-1 \rightarrow t-2}(\mathbf{x}) = \text{round}(c_{t-1 \rightarrow t-2}(\mathbf{x} + c_{t \rightarrow t-1}(\mathbf{x}))). \quad (5.20)$$

This works and makes sense for positions where  $c_{t \rightarrow t-1}$  is 1 and the displacement  $w_{t \rightarrow t-1}$  does not leave the image boundaries. For all other positions, however, evaluating the new validity flag is more problematic.

The most straightforward idea is to set  $\hat{c}_{t-1 \rightarrow t-2}(\mathbf{x})$  to 0 if  $c_{t \rightarrow t-1}(\mathbf{x})$  is 0. The intuition here is that if the displacement we use to warp  $w_{t-1 \rightarrow t-2}$  is invalid, the resulting interpolated flow can not be valid either. However, by doing so, the sample size is limited to that of  $w_{t \rightarrow t-1}$ . Thus, this approach defeats the purpose of using a loss function with the disjunctive validity flag methods.

The other idea is to ignore the validity flag of the first backward flow completely. More precisely, the validity flag of  $\hat{w}_{t-1 \rightarrow t-2}$  is only determined by bilinear interpolation, regardless of the validity of  $w_{t \rightarrow t-1}$ . While this might seem unintuitive at first, the idea here is that due to the appended validity flag  $c_{t \rightarrow t-1}$ , the network might be able to identify the dubiousness of such a position even if  $\hat{c}_{t-1 \rightarrow t-2}$  happens to be set to 1. More importantly, however, with this method the sample size can potentially increase in the disjunctive loss setting.

The final problem arises for positions where  $w_{t \rightarrow t-1}$  leaves the image boundaries. Here, we have no flow information of  $w_{t-1 \rightarrow t-2}$  which makes warping difficult. Therefore, it makes the most sense to set the validity flag of the warped flow at such a position to 0. However, when using the disjunctive validity flag variant, it might still be considered as a training sample if  $w_{t \rightarrow t-1}$  is valid. Additionally, if a neighboring position (within the kernel size of the CNN) is valid, it will also be considered in the convolution of that neighbor. Therefore, we still need to adjust the flow.

One option is to assume linear motion and set  $\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x})$  simply to  $w_{t \rightarrow t-1}(\mathbf{x})$ . Another option is to get the last available flow of  $w_{t-1 \rightarrow t-2}$  in the direction of  $w_{t \rightarrow t-1}(\mathbf{x})$ . In other words, we clamp  $(\mathbf{x} + w_{t \rightarrow t-1}(\mathbf{x}))$  at the image boundaries. A more in-depth discussion of both methods as well as the evaluation of the warping approach are presented in Section 6.3.

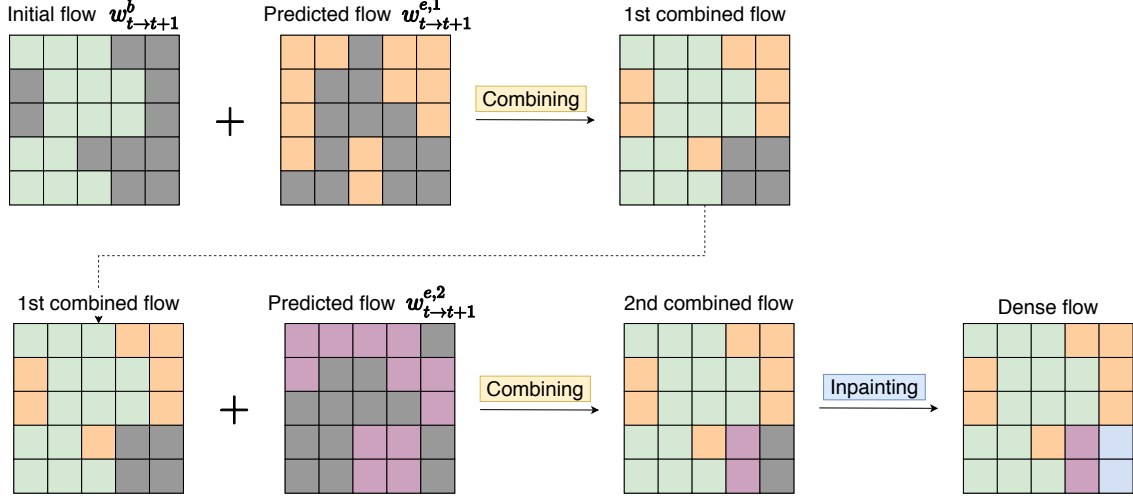
## 5.5 Separate CNNs

Here, we are exploring a slightly different angle of the multi-frame setting. Instead of focusing on the network inputs, the overall pipeline architecture of the prediction step is addressed. Until now, we have seen different possibilities of combining multiple backward flows which are then used as the input to a single CNN. These methods all have one thing in common: The network has to learn meaningful temporal connections between multiple backward flows (or the precomputed equivalents) by itself.

In contrast, the idea of this approach is to explicitly handle the extraction of temporal features. More precisely, multiple CNNs are trained independently, each with only one backward flow as input. Their predictions are then joined explicitly during the combination step of the ProFlow pipeline.

### 5.5.1 Implementation Details

As in the previous approaches, all backward flows are first computed by the baseline and filtered with respect to the reference frame. However, instead of concatenating the backward flows and feeding them into a single network, a separate CNN is trained for each of the flows. Each CNN is based on the general ProFlowS architecture as described in Chapter 4.1, but trained separately (i.e. weights are not shared). For an input history of 4, two networks are necessary, each producing their own estimation. That is,



**Figure 5.6:** Illustration of the adjusted combination step for four frames ( $h = 4$ ). Note the difference to Figure 3.6.

$$\begin{aligned}
 (w_{t \rightarrow t-1}, c_{t \rightarrow t-1}) &\xrightarrow{\text{predict}} w_{t \rightarrow t+1}^{e,1} \\
 (w_{t \rightarrow t-2}, c_{t \rightarrow t-2}) &\xrightarrow{\text{predict}} w_{t \rightarrow t+1}^{e,2},
 \end{aligned} \tag{5.21}$$

where  $w_{t \rightarrow t+1}^{e,1}$  is the prediction of the first CNN and  $w_{t \rightarrow t+1}^{e,2}$  is the prediction of the second CNN. Evidently, this approach can be extended to the next larger input history by computing the next backward flow and training an additional CNN on it.

### 5.5.2 Combination Step

Since we have a separate prediction per backward flow now, we need to combine both predictions into a single estimate. To do so, the combination step of the original ProFlow architecture is slightly adapted. Figure 5.6 illustrates the new combination step for  $h = 4$ , i.e. for two predictions: First, the baseline forward flow is directly carried over to the final flow at positions where it is still valid after filtering. Then, for each position where  $w_{t \rightarrow t-1}$  is valid (but not the corresponding forward flow), the estimation from the first CNN is taken, i.e.  $w_{t \rightarrow t+1}^{e,1}$ . Following that, for each position where  $w_{t \rightarrow t-2}$  is valid (but neither the corresponding forward flow nor the first backward flow  $w_{t \rightarrow t-1}$ ), the estimation from the second CNN is taken, i.e.  $w_{t \rightarrow t+1}^{e,2}$ . Finally, only for positions where neither the forward flow nor *any* backward flow is valid, inpainting is used.

### 5.5.3 Intuition

Note that with the above combination method, we then use a prediction at any position where at least one backward flow is valid. That is, the positions where a prediction is taken coincides with the positions of the disjunctive combination (Section 5.2.2). The benefit is that if the predictions are good, the results should be significantly better with larger input histories as less pixels have to be filled in via inpainting.



The other major benefit of separating the CNNs is that it combines the benefits of both the conjunctive and disjunctive validity flag variants (Section 5.2.1). In contrast to the conjunctive setting, all valid backward flows are considered by the loss function during training, and, in contrast to the disjunctive setting, the positions used for training contain only valid flow values. (Note, however, that invalid flows can still enter the learning process due to the convolution filters incorporating neighboring positions.)

Finally, separating the CNNs allows for explicitly prioritizing the estimations. More precisely, the first CNN estimates the forward flow from the backward flow  $w_{t \rightarrow t-1}$ , i.e. the backward flow of a single time step. The second CNN estimates the forward flow based on the backward flow  $w_{t \rightarrow t-2}$ , a flow over *two* time steps. Since the temporal difference and therefore the motion is larger in the second backward flow, it is reasonable to assume that the first CNN yields better estimations. Only for positions that are occluded between  $I_{t-1}$  and  $I_t$  but not between  $I_{t-2}$  and  $I_t$ , e.g. objects leaving and re-entering the scene, the second CNN's estimation might be beneficial. Thus, by prioritizing the predictions during the combination step, it is always assured that the better estimation is taken as the final optical flow.

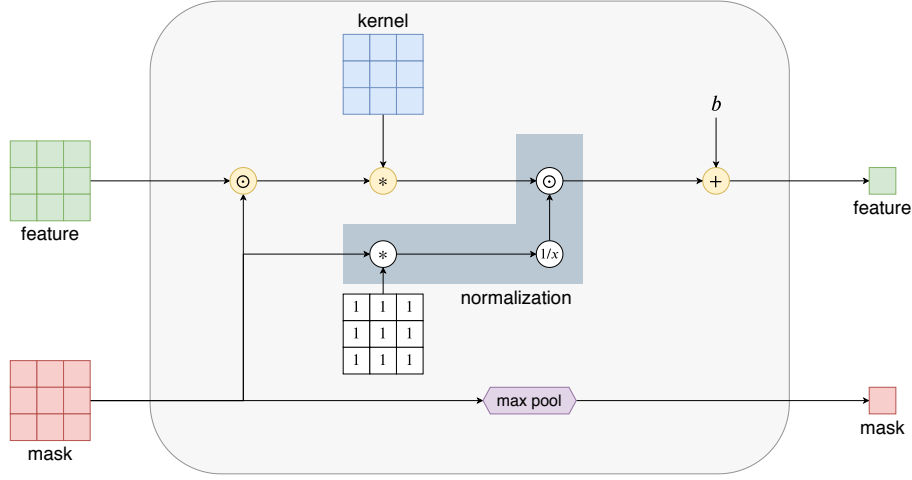
This process is easy to extend to more than four frames, i.e. for  $h > 4$ . For each additional backward flow, an additional CNN is trained and its respective prediction added during the combination step. One major caveat is that each CNN has the same amount of inputs and outputs and, thus, takes the same amount of time to be trained. The time necessary for the prediction step therefore grows linearly in the amount of used backward flows. However, each additional CNN becomes less likely to add meaningful flow estimations at many positions, in particular since the amount of valid pixels used for training presumably decreases significantly for most sequences. The gained information does therefore most probably not outweigh the additional computational costs for a large input history. We will see the results of this approach with different input histories in Section 6.4.

## 5.6 Sparse Convolution

The networks in the previous approaches suffer from one common difficulty: They all have to deal with sparse input data. Until now, we have assumed that the networks are capable of accurately interpreting the sparsity due to the appended validity flags. While the loss function already prevents training from positions where the validity is zero (depending on the chosen combined flag  $c^{\text{all}}$ ), such positions can still affect the learning process due to the convolution filters in the network.

More precisely, applying a convolution filter to a pixel results in a new value for that position. Depending on the size of the filter, however, multiple neighboring pixels are incorporated into that convolution operation (see Equation 2.3). Therefore, even if these neighboring positions are considered invalid, they affect the outcome of the convolution. As long as the center pixel is valid, the network is still trained on these values. Thus, the sparsity of the input data and the resulting problems even apply to the separate CNNs approach of the previous section.

The following approach therefore focuses on modifying the CNN layers itself, such that invalid positions do not affect the outcome of a convolution. This approach is inspired by Uhrig *et al.* [USS+17]. In their 2017 work, they introduce a so-called *Sparse Convolutional Layer* that explicitly handles sparse input data. While their work primarily focuses on non-dense depth maps, it is reasonable to assume that it can also be applied in the field of optical flow.



**Figure 5.7:** Illustration of a Sparse Convolutional Layer as introduced by Uhrig *et al.* [USS+17]. Here,  $*$  denotes a convolution,  $\odot$  element-wise multiplication,  $1/x$  inversion and  $b$  is a bias. Source: Uhrig *et al.* [USS+17].

### 5.6.1 Implementation Details

Up until now, for each backward flow, a validity map is appended to it and added as a separate input to the network. Thus, with multiple backward flows, we also have multiple validity maps. The CNN then has to learn by itself how to interpret the respective validity flags. Instead, a Sparse Convolutional Layer as suggested by Uhrig *et al.* takes a single *observation mask* (here: validity map) as one additional input to explicitly handle the sparsity.

More precisely, given an observation mask  $o$ , where  $o_{i,j} = 1$  if the value is observed (valid) at position  $(i, j)$  and  $o_{i,j} = 0$  otherwise, the modified convolution is then

$$f_{i,j}(x, o) = \frac{\sum_{s,t=-k}^k o_{i+s,j+t} x_{i+s,j+t} \omega_{s,t}}{\sum_{s,t=-k}^k o_{i+s,j+t} + \epsilon} + b, \quad (5.22)$$

where  $x$  is the input,  $\omega$  is the kernel's weight,  $b$  is a bias and the kernel size is  $2k + 1$ . Here,  $\epsilon$  is a small value that makes sure the denominator does not evaluate to zero if all values are non-valid. Note that if all values are valid, the equation comes down to a standard (scaled) convolutional layer as in Equation 2.3.

Since the network has to then propagate the validity information to the next layer, the observation mask has to be updated accordingly. Uhrig *et al.* suggest that the new observation value  $o_{i,j}^{n+1}$  should only be set to zero (non-valid) if *none* of the previous observations in the filter region were valid. Therefore, the max-pooling operation can be used to determine the mask of the next layer:

$$o_{i,j}^{n+1} = \max_{s,t=-k,\dots,k} o_{i+s,j+t}^n, \quad (5.23)$$

where  $n$  is the current layer,  $n + 1$  the next layer and  $2k + 1$  the kernel size. Figure 5.7 illustrates a single Sparse Convolutional Layer.

Uhrig *et al.* show that their so-called *SparseConvNet* outperforms most state-of-the-art approaches when the input is sparse. While these results are based on depth-map estimations instead of optical flow, it can be assumed that the flow estimations will also improve, in particular for sequences where only few positions remain after filtering.

The Sparse Convolution Layer as defined above was implemented in TensorFlow 2.0 with the Keras API such that it can be used in the ProFlowS network. The overall network architecture remains the same as in all other approaches which is described in Chapter 4.1. The only difference is the implementation of the convolutional layers. Each layer in the network now employs the Sparse Convolution (Equation 5.22) instead of a standard convolution. This also means that we can combine this layer architecture with each of the above multi-frame modifications.

### 5.6.2 Combined Validity Map

Except for the Separate CNNs approach in Section 5.5, each method uses multiple backward flows (or increments) in a single network, each appended with its respective validity map. The here presented Sparse Convolution Layer makes explicit use of a *single* observation map which is appended to the input as a whole.

Thus, when combining the new layer architecture with one of those approaches, we have to decide how to combine multiple validity maps into said observation mask. Since we already combine multiple validity flags into a single flag in the loss function, it makes sense to use the same method for the new observation mask. More precisely, the validity maps for each backward flow are merged into a single observation mask  $o$  according to the chosen validity flag method in the loss function, i.e. either conjunctively (see Equation 5.4) or disjunctively (see Equation 5.5).

One additional question is whether the forward flow validity should be included in the observation mask—that is, whether we only merge the backward flows to a new observation value or whether we need to consider the validity of the corresponding label in our Sparse Convolutions. This will be addressed later in Section 6.5.

### 5.6.3 Prospect

Evidently, this approach does not explicitly address the multi-frame setting and can instead be applied to the three-frame ProFlowS architecture as well. However, with larger input histories and more backward flows, we potentially also have more sparsity, making the Sparse Convolutional Layers more relevant. Thus, it makes sense to examine this approach in combination with the above multi-frame modifications.

Since in this work we are mostly interested in multi-frame approaches improving optical flow predictions, the models evaluated in Section 6.5 only serve as a first insight into the here introduced layer architecture. More in-depth studies of Sparse Convolutions in the field of optical flow prediction and also with respect to multi-frame modifications are left for future research.



## 6 Results and Evaluation

In this chapter we are going to look at and discuss the results of the five approaches introduced in the previous chapter. We are primarily going to investigate each of the modifications on the Sintel clean pass but additionally present and evaluate results for different input histories on the Sintel final pass as well as on KITTI 2015. This should give us a good overview of each approach and its performance under different circumstances.

### 6.1 Multiple Backward Flows: Evaluation

Section 5.2 introduced an approach that uses multiple backward flows as the CNN input to predict the corresponding forward flow. As discussed, there are two parts of the prediction step that have different implementation possibilities: The validity flag of the loss function (see Section 5.2.1) and the combination step method (see Section 5.2.2), each with two possible variations.

In the following, models that employ the conjunctive validity flag are denoted by the suffix *-c* and models with the disjunctive validity flag are denoted by *-d*. Similarly, models that use the disjunctive combination step are denoted by an additional suffix *-d*. Models that follow the original ProFlowS combination step, on the other hand, are denoted by the suffix *-s*. Note that we are not going to use the conjunctive combination step due to the reasons given in Section 5.2.2.

Finally, all models in this section are denoted by ProFlow-MBF, followed by the respective suffixes expressing the employed loss function and combination step. Since there are two possible validity flags for the loss function and two possible combination steps, we have four models to be evaluated and compared to each other:

1. ProFlow-MBF-c-s: conjunctive validity flag and ProFlowS combination step
2. ProFlow-MBF-c-d: conjunctive validity flag and disjunctive combination step
3. ProFlow-MBF-d-s: disjunctive validity flag and ProFlowS combination step
4. ProFlow-MBF-d-d: disjunctive validity flag and disjunctive combination step

We will look at the performance of each model in the following.

Sequence	ProFlowS AEE	ProFlow-MBF-c-s AEE	ProFlow-MBF-c-d AEE	ProFlow-MBF-d-s AEE	ProFlow-MBF-d-d AEE
alley_2	<b>0.17</b>	0.19	0.19	<b>0.17</b>	<b>0.17</b>
ambush_2	6.48	9.68	9.43	<b>5.92</b>	6.03
ambush_4	11.72	22.09	21.87	<b>10.94</b>	10.97
ambush_7	0.57	0.64	0.58	<b>0.53</b>	0.55
bandage_2	<b>0.17</b>	0.19	0.19	<b>0.17</b>	<b>0.17</b>
market_2	0.48	0.55	0.55	<b>0.47</b>	0.48
market_5	8.42	12.60	12.40	<b>8.23</b>	8.25
temple_3	4.90	8.00	8.00	4.75	<b>4.72</b>
all	2.22	3.27	3.27	<b>2.11</b>	2.15

**Table 6.1:** Results of the different models for individual exemplary sequences of the Sintel clean pass. Except for ProFlowS (three frames), the given results are for an input history of  $h = 4$ . For a breakdown of each sequence refer to Appendix A.2.

### 6.1.1 Results

Table 6.1 shows the results for each of the four models on the Sintel clean pass in comparison to ProFlowS. For evaluation purposes, the total AEE of each method as well as a breakdown for a few individual sequences of the dataset is given. While ProFlowS uses the classical three-frame setting, all other models are evaluated on an input history of  $h = 4$ .

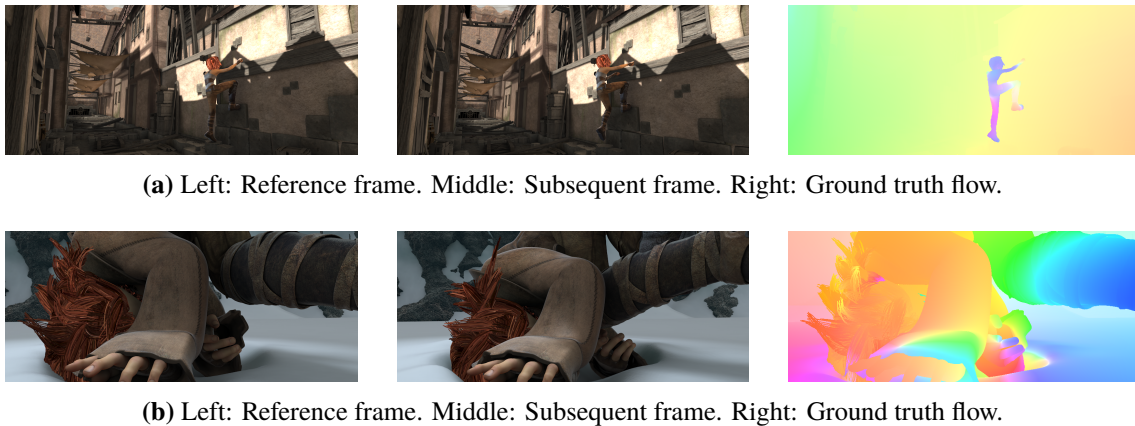
Both conjunctive validity flag models (denoted by the suffix *-c*) perform significantly worse than ProFlowS. The disjunctive models (denoted by the suffix *-d*), on the other hand, perform generally better than ProFlowS. ProFlow-MBF-d-s, as the best performing model, provides an accuracy increase of about 5% (from an AEE of 2.22 in ProFlowS to 2.11 with one additional input frame).

### 6.1.2 Discussion

To better understand the benefits and drawbacks of both the validity flag method and the combination step mode, we will discuss the models in two parts. We will first evaluate the models that employ the conjunctive validity flag against the models that use the disjunctive validity flag. Following that, we will look at the performance difference arising from the chosen combination step.

#### Conjunctive versus Disjunctive Validity Flag

Let us first focus on the two models that use the conjunctive validity flag, ProFlow-MBF-c-s and ProFlow-MBF-c-d. Both perform significantly worse than ProFlowS. Looking at individual sequences, we can see that this is particularly true for *ambush\_2*, *ambush\_4*, *market\_5* and *temple\_3* with an error increase between 50% and 90%. For the other shown sequences, however, the error is only slightly worse than in ProFlowS (i.e. about 10%). Comparing the AEE of the former sequences with the other sequences, we can assume that these sequences are harder to estimate in general because ProFlowS itself performs significantly worse on, e.g., *ambush\_4* than on *alley\_2*.



**Figure 6.1:** ‘Easy’ versus ‘hard’ sequences in Sintel clean. Top: Frame 29 of the *alley\_2* sequence. Bottom: Frame 6 of the *ambush\_4* sequence.

Sequence	$\mathbf{c}_{t \rightarrow t+1} \wedge \mathbf{c}_{t \rightarrow t-1}$ valid positions (%)	$\mathbf{c}_{t \rightarrow t+1} \wedge \mathbf{c}_{t \rightarrow t-2}$ valid positions (%)	Conjunctive Flag valid positions (%)	Disjunctive Flag valid positions (%)
alley_2	93	91	90	93
ambush_2	60	48	47	61
ambush_4	60	49	47	62
ambush_7	94	92	92	94
bandage_2	96	93	93	96
market_2	92	89	88	93
market_5	63	52	51	65
temple_3	67	58	57	68
all	86	82	81	87

**Table 6.2:** Amount of valid positions for each backward flow as well as the two combined validity flag methods for  $h = 4$ . The amounts are presented for the same sequences as in Table 6.1. Note that since the corresponding forward flow  $w_{t \rightarrow t+1}$  serves as the label in training, it always has to be valid for a particular pixel to be potentially considered by the loss function. For a breakdown of each sequence refer to Appendix A.2.

Figure 6.1 further emphasizes this assumption. The two frames of the *alley\_2* sequence visibly contain only small motion between them, while the two frames of *ambush\_4* exhibit large and more complex motion patterns between them. Intuitively, this means that the baseline itself performs significantly worse on such sequences which makes training and, thus, reliable prediction even harder. More precisely, it is likely that considerably less valid flows are available after the filtering step which would be especially true for the second backward flow  $w_{t \rightarrow t-2}$ . Thus, using a conjunctive validity flag further reduces the amount of positions that are considered by the loss function during training, leading to worse performance than ProFlowS.

In order to investigate this assumption, we need to look at the amount of valid positions for both backward flows. Table 6.2 shows the amount of valid positions (in percent) for the same sequences presented in Table 6.1. Looking at the sequences where ProFlow-MBF-c-s and ProFlow-MBF-c-d

performed particularly bad, we can see that the amount of valid positions is significantly smaller than for the other sequences. This is particularly true for the second backward flow  $w_{t \rightarrow t-2}$ . For instance, for *ambush\_4* the amount of positions from which the network can be trained decreases from 59% in the first backward flow to 48% in the second backward flow. We can see that this effect is further amplified by the conjunctive validity flag where both backward flows (as well as the forward flow) have to be valid. Therefore, with an input history of 4, the network is trained on significantly less flow values than ProFlowS which explains the bad performance of the conjunctive validity flag for these sequences. For the other sequences, this problem persists but is far less profound. E.g. for *alley\_2*, the amount of valid positions only decreases by 2%, leaving enough flow values for training.

Let us now look at the two models that use the disjunctive validity flag, ProFlow-MBF-d-s and ProFlow-MBF-d-d. Both generally outperform ProFlowS on most sequences. The amount of positions considered during training for these models is given in the last column of Table 6.2. Compared to ProFlowS, there are even more positions the network can learn from. Note that, as mentioned before, this also includes positions where one of both backward flows is considered invalid, which could have potentially exacerbated the training process. The good performance shows that this is not the case, however. The additional information gained from the second backward flow evidently outweighs the noise introduced through partially invalid backward flows.

One possible explanation is that the network is able to recognize invalid flows via its validity flag. That is, since each backward flow is appended by its validity map, if one of validity flags is zero and the other is one, it is possible that the network is able to extract that partial validity as a feature of the dataset. Another potential explanation is that even if one backward flow is considered invalid after the filtering step, the quality of said flow might still be fairly good due to both the other backward flow as well as the respective forward flow being valid. More precisely, suppose both  $w_{t \rightarrow t+1}$  and  $w_{t \rightarrow t-1}$  are valid and suppose  $w_{t \rightarrow t-2}$  is not valid. Here, it might be the case that the bi-directional consistency check of  $w_{t \rightarrow t-2}$  only slightly violates the occlusion threshold, thus still leading to a reasonable training sample.

Overall, we can conclude that the disjunctive validity flag generally provides significantly better results than its conjunctive counterpart. It might be beneficial, however, to explicitly regulate the quality of the invalid flows that are used for training when we employ this method. We will look at a model which does exactly that in the next section. First we will evaluate the two different combination steps, however.

#### Disjunctive versus ProFlowS Combination Step

While we have determined the better validity flag for training, we still have to look at the different combination steps. For the two models that use the conjunctive validity flag, the difference between the disjunctive combination step and the ProFlowS combination step is almost non-existent. For the other two models (using the disjunctive validity flag), however, the ProFlowS combination step is slightly better with a total AEE of 2.11 versus the disjunctive combination step with an AEE of 2.15.

One possible reason could be that the network is not able to reliably predict a forward flow for positions where one backward flow is invalid. That is, inpainting potentially performs better for such positions than the network prediction. Since there are far more such partially invalid positions in the disjunctive combination setting, the overall flow estimation slightly deteriorates.



The alternative and more likely explanation, however, is that the performance gain of the ProFlowS combination setting is due to inpainting generally performing better on Sintel than the network prediction with ProFlowS (see Section 4.2). More precisely, since the estimation of the baseline is better than the estimation of ProFlowS, having less positions using the estimations and more positions evaluated via inpainting leads to better overall performance.

This claim is additionally supported by the performance difference of the two validity flag methods. Since the significantly better results of the disjunctive validity flag suggests that the network is able to discern and learn from partially invalid backward flows, it should also be capable of providing a decent prediction for such positions. Thus, if the accuracy of the estimation was better than the accuracy of inpainting, the disjunctive combination step should provide a greater potential for performance increase as more positions are estimated than with the ProFlowS combination setting.

Overall, we can say that the choice of the validity flag is more crucial than the combination step. Both combination methods provide better estimations than ProFlowS when paired with the disjunctive validity flag for training. This proves that temporal information contained in the second backward flow can be leveraged for a better flow estimation. In the following section we will investigate whether it is possible to further improve the flow estimation by building on the insights we have gained in this section. To do that, an additional model is introduced that explicitly handles the quality of invalid flows that are used for training when the disjunctive validity flag is used.

### 6.1.3 Extrapolated Validity Flag Model

We have learned that the disjunctive validity flag is the crucial part for an increased estimation accuracy with a larger input history. As this includes partially invalid backward flows in the training process, however, we are going to investigate if explicitly regulating those flows can further improve the estimation. More precisely, while the disjunctive validity flag is used during training in this section, invalid backward flows are discarded and instead extrapolated from the corresponding valid backward flow. To do that, we need a model that describes the motion over multiple time steps which then allows us to extrapolate from one time step to the next.

#### Motion Model

Since the motion is generally small between two consecutive frames, it is reasonable to assume that it can be approximated linearly in many cases. Figure 6.2 shows the intuition for this approach. The first backward flow  $w_{t \rightarrow t-1}$  is denoted with red arrows, while the second backward flow  $w_{t \rightarrow t-2}$  is denoted with green arrows. As we can see, most of the green arrows are roughly double the size of the red arrows while pointing in a very close direction which indicates a linear motion between the two frames. Following that, we can extrapolate one backward flow from the other linearly.

Suppose  $w_{t \rightarrow t-1}$  is valid but  $w_{t \rightarrow t-2}$  is not, then  $w_{t \rightarrow t-2}$  is set to  $2 \cdot w_{t \rightarrow t-1}$ , i.e. the second backward flow is linearly extrapolated from the first backward flow. Since the time step is twice as big, the flow is doubled as well. Similarly, if  $w_{t \rightarrow t-2}$  is valid but  $w_{t \rightarrow t-1}$  is not, then  $w_{t \rightarrow t-1}$  is set to  $0.5 \cdot w_{t \rightarrow t-2}$ . While this assumption does not hold at all pixel locations, this method could provide a decent approximation for many of the sequences.



**Figure 6.2:** Baseline backward flows ( $h = 4$ ) for frame 5 of the *alley\_1* sequence. The first backward flow ( $w_{t \rightarrow t-1}$ ) is denoted with green arrows while the second backward flow ( $w_{t \rightarrow t-2}$ ) is denoted with red arrows. Refer to Appendix A.2 for a more in-depth example.

Sequence	ProFlowS	ProFlow-MBF-d-s	ProFlow-MBF-e-s
	AEE	AEE	AEE
alley_2	0.17	<b>0.17</b>	0.18
ambush_2	6.48	<b>5.92</b>	6.21
ambush_4	11.72	<b>10.94</b>	11.74
ambush_7	0.57	<b>0.53</b>	0.57
bandage_2	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
market_2	0.48	<b>0.47</b>	<b>0.47</b>
market_5	8.42	<b>8.23</b>	9.17
temple_3	4.90	4.75	5.43
all	2.22	<b>2.11</b>	2.29

**Table 6.3:** Results of the extrapolating model for selected sequences of the Sintel clean pass and an input history of  $h = 4$ . For a breakdown of each sequence refer to Appendix A.2.

## Results

Let us now investigate whether the extrapolated flow values contain better information than the filtered backward flow. Note that the ProFlowS combination step is used in the here presented model as it performed the best in the previous section. The new model is denoted as ProFlow-MBF-e-s and compared to ProFlow-MBF-d-s as well as the standard three-frame ProFlowS. Thus, we have two models that use the same positions for training and employ the same combination step after predicting. The only difference is some of the backward flow values at positions used for training are gained via extrapolation instead of taken from the filtered baseline.

Table 6.3 shows the results of the new model on the Sintel clean pass. We can see that ProFlow-MBF-e-s performs significantly worse than ProFlow-MBF-d-s on most sequences, suggesting that the invalid flow values of  $w_{t \rightarrow t-2}$  are more accurate than the extrapolated values. In fact, ProFlow-MBF-e-s performs even worse than ProFlowS. If we look at individual sequences, we can see that this is particularly true for sequences with complex motion patterns such as *ambush\_2* and *ambush\_4*. This is not surprising, considering these sequences contain large and non-rigid motion. It is very likely that the linear motion model we assumed is a particularly bad fit for these sequences.

Generally, only for very few sequences does ProFlow-MBF-e-s perform similar to ProFlow-MBF-d-s and, in fact, it never outperforms the latter. Therefore, we can conclude that this method of extrapolating backward flow values from adjacent time steps does not yield good results in general.

#### 6.1.4 Summary and Larger Input Histories

We have seen that both models with the disjunctive validity flag and an input history of 4 perform better than ProFlowS on Sintel clean, proving the usefulness of temporal coherence in the image sequence. However, we have yet to see the performance of this approach on larger input histories. Additionally, we are going to look at the performance on the Sintel final pass as well as KITTI 2015. Because the ProFlow-MBF-d-s model performed the best on the clean pass of Sintel, it is used here on the other benchmarks as well.

The results are listed in Table 6.4 for input histories between 3 and 5 using the ProFlow-MBF-d-s model. Note that for  $h = 3$  the results are those of ProFlowS and, thus, coincide with those in Table 4.1. As we can see, this approach generally deteriorates for an input history of 5. In fact, only Sintel clean shows considerable improvement with four frames compared to the three-frame ProFlowS setting. For KITTI 2015, on the other hand, the accuracy significantly drops even with only four input frames. One possibility is that the second backward flow  $w_{t \rightarrow t-2}$  is very sparse, leading to many positions considered by the loss function during the training process that contain invalid flow values. This likely makes it harder for the network to learn from the backward flows and to interpret the respective validity flag accurately. Presumably, this is also true for the final pass of Sintel as the baseline is less accurate here due to motion blur, severe illumination changes and atmospheric effects which were not present in the clean pass.

Similarly, for larger input histories, the amount of positions with partially invalid backward flows generally increases. Additionally, there are now several positions where two of the three backward flows are invalid (typically  $w_{t \rightarrow t-2}$  and  $w_{t \rightarrow t-3}$ ). With the conjunctive validity flag, these positions are still considered by the loss function during training, introducing more and larger outliers to the training data. It is, thus, even more important that the network is able to accurately identify these outliers based on their respective validity flag being zero. Evidently, it is not able to do so sufficiently. Therefore, it makes sense to explore alternative ways of preprocessing the backward flows in an attempt at assisting the network in identifying these features. Sections 5.3 and 5.4 introduced two such models. Their performance is discussed in the following sections.

Overall we have seen that the second backward flow can contain complementary information to the first backward flow, leading to an improved estimation for  $h = 4$  on Sintel clean. This proves that it is possible to leverage temporal information in order to increase the estimation accuracy.

Model	Input history (h)	Sintel Clean	Sintel Final	KITTI 2015	
		AEE	AEE	AEE	BP
ProFlowS	3	2.22	4.00	5.49	18.00
ProFlow-MBF-d-s	4	2.11	4.02	5.57	18.55
ProFlow-MBF-d-s	5	2.14	4.08	5.76	18.92

**Table 6.4:** Results of ProFlow-MBF-d-s on all major benchmarks with different input histories.

## 6.2 Flow Differences: Evaluation

Based on the two variants for the flow increment’s validity flag described in Section 5.3.2, we have two major models to examine for the flow difference approach. The first model uses the disjunctive method described in Equation 5.12. The second model sets the increment validity to that of the backward flow with the larger time step as detailed in Equation 5.14. We will refer to the former as ProFlow-DIFFa and to the latter as ProFlow-DIFFb in order to not confuse these methods with the combined validity flag of the loss function (cf. Section 5.2.1).

Regarding said loss, since the disjunctive validity flag variant proved to be significantly better than the conjunctive alternative, both models use the disjunctive approach. We will thus append the suffix *-d* to both models. Finally, the ProFlowS combination step proved to be slightly better than its disjunctive counterpart, which is why we will mostly focus on the former for both models.

### 6.2.1 Results

Table 6.5 shows the results of both models on the Sintel clean pass for an input history of 4 compared to ProFlowS. Note that the last row additionally gives a brief impression on the performance of the disjunctive combination step (cf. Section 5.2.1).

All models show improvement over ProFlowS, with ProFlow-DIFFb-d-s and ProFlow-DIFFb-d-d giving the best results. Therefore, simply setting the validity flag of  $w'_{t-1 \rightarrow t-2}$  to that of  $w_{t \rightarrow t-2}$  seems to be the better approach. In fact, ProFlow-DIFFb-d-s (2.09) is even slightly better than ProFlow-MBF-d-s (2.11) in the previous section. Likewise, ProFlow-DIFFb-d-d using the disjunctive combination step is slightly better than ProFlow-MBF-d-d, albeit still worse than the ProFlowS combination step in ProFlow-DIFFb-d-s.

Following these results, Table 6.6 supplements the errors of ProFlow-DIFFb-d-s for larger input histories on all benchmarks. Similarly to ProFlow-MBF-d-s (Section 6.1.4), the best results were achieved on the Sintel clean pass with an input history of  $h = 4$ . While the performance on Sintel clean decreases with even larger input histories, the model shows the best results for  $h = 5$  on the final pass. On KITTI 2015, the error slightly increases with more input frames.

Model	Sintel Clean	
	Input history (h)	AEE
ProFlowS	3	2.23
ProFlow-DIFFa-d-s	4	2.22
ProFlow-DIFFb-d-s	4	2.09
ProFlow-DIFFb-d-d	4	2.12

**Table 6.5:** Results of the flow difference methods on Sintel clean for an input history of  $h = 4$  compared to ProFlowS ( $h = 3$ ).

Model	Input history (h)	Sintel Clean		KITTI 2015	
		AEE	Sintel Final AEE	AEE	BP
ProFlowS	3	2.22	4.00	5.49	18.00
ProFlow-DIFFb-d-s	4	2.09	4.02	5.50	18.08
ProFlow-DIFFb-d-s	5	2.18	3.96	5.55	18.59

**Table 6.6:** Results of ProFlow-DIFFb-d-s compared to ProFlowS on both Sintel passes and KITTI 2015 for different input histories.

### 6.2.2 Discussion

As shown in Table 6.6, the accuracy is at least as good as that of the first approach or better (cf. Table 6.4). This suggests that the network is just as capable of learning from flow increments as from multiple backward flows. In fact, for an input history of 4 the prediction slightly improves compared to simply stacking multiple backward flows, implying that the learning process can benefit from this approach.

Looking at the clean pass of Sintel, however, an input history of 5 leads to worse results. One possible explanation could be that for this particular dataset, the disjunctive choice for the difference validity method (Equation 5.12) fails for the third backward flow (i.e.  $c'_{t-2 \rightarrow t-3}$  is misleading in some cases). Note, however, that the performance for  $h = 5$  mostly suffers for sequences with complex motion patterns such as *ambush\_2* or *ambush\_4*, suggesting that this is only the case for sequences with overall few valid positions. In fact, for some sequences (such as *ambush\_6*) the AEE slightly decreases for  $h = 5$ . For more details refer to Appendix A.3. Furthermore, on Sintel final, the performance is actually the best with an input history of 5, implying that both this approach and the disjunctive difference validity can work even for larger input histories and complex motions.

Additionally, note that the performance is significantly better on KITTI than with the first approach. While the accuracy still slightly deteriorates on larger input histories, both the average endpoint error and the bad pixel error are significantly better than its ProFlow-MBF-d-s counterpart.

In conclusion we can say that using flow increments instead of multiple backward flows can be beneficial. Generally, the flow estimation with four frames is significantly more accurate than with the three-frame ProFlowS alternative. More importantly, however, the results on Sintel final for  $h = 5$  prove that even the third backward flow  $w_{t \rightarrow t-3}$  can still contain additional and valuable complementary information.

Model	Input history (h)	Sintel Clean	Sintel Final
		AEE	AEE
ProFlowS	3	2.22	4.00
ProFlow-WARP-d-s	4	2.32	4.06

**Table 6.7:** Results of ProFlow-WARP-d-s for  $h = 4$  on both Sintel passes compared to ProFlowS. For more details, refer to Appendix A.4.

Overall, while this once again proves the usefulness of temporal information in the input sequence, the performance gain compared to the first approach is fairly small and could be situational.

### 6.3 Multiple Warped Backward Flows: Evaluation

In Section 5.4 an approach that uses warped backward flows was introduced. Section 5.4.2 discussed different possibilities for the validity flag of the warped flow. We are now going to look at the best performing variant and then discuss and justify these results. Note that all conducted experiments on this approach employed the disjunctive validity flag method as part of the loss function, since it proved to be significantly better than its conjunctive counterpart (see Section 6.1.2). Similarly, the ProFlowS combination step was used in all runs. Thus, we denote the general model in this section as *ProFlow-WARP-d-s*.

#### 6.3.1 Results

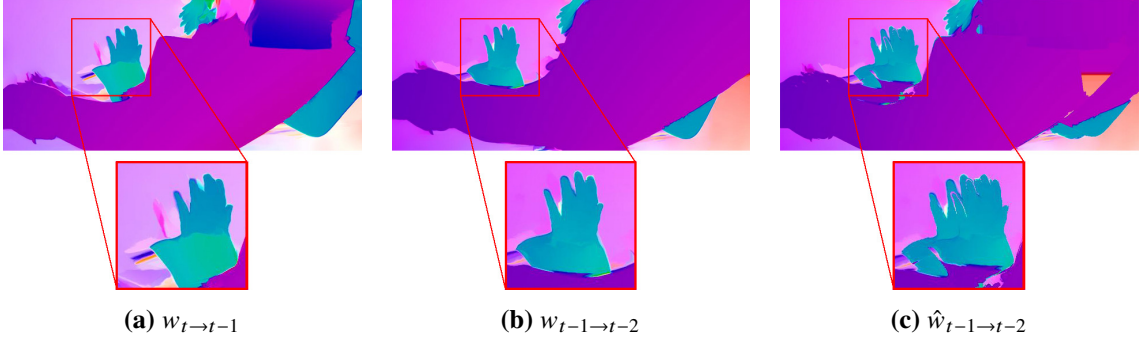
Table 6.7 shows the best results achieved with ProFlow-WARP-d-s on both Sintel passes for an input history of  $h = 4$ . Interestingly, these results were achieved by ignoring the validity flag of  $w_{t \rightarrow t-1}$  during warping (cf. Section 5.4.2). That is, the validity flag of  $\hat{w}_{t-1 \rightarrow t-2}$  solely depends on  $w_{t-1 \rightarrow t-2}$  (unless  $w_{t \rightarrow t-1}$  leaves the image boundaries).

For positions where  $w_{t \rightarrow t-1}(\mathbf{x})$  leaves the image boundaries, two methods with which we can still provide a flow value for  $\hat{w}_{t-1 \rightarrow t-2}$  have been suggested in the previous chapter: We can either assume linear flow and thus extrapolate from  $w_{t \rightarrow t-1}(\mathbf{x})$  or we can take the flow value of  $w_{t-1 \rightarrow t-2}$  at the image boundary in the direction given by  $w_{t \rightarrow t-1}(\mathbf{x})$ . In all conducted experiments, the latter achieved significantly better results and is, thus, the method used for ProFlow-WARP-d-s. Possible reasons are explored in the following sections.

Overall, however, the estimations are still less accurate than with ProFlowS. We are now going to discuss why this might be the case.

#### 6.3.2 Discussion

In order to get an intuitive understanding of why ProFlow-WARP-d-s generally performs worse than ProFlowS, let us first take a look at a warped flow. Figure 6.3 shows the difference between backward flows and the resulting warped flow. The first major problem we can see is that the warping



**Figure 6.3:** Result of the warping operation on the Sintel clean pass.

operation introduces artifacts in the form of ‘doubled’ image parts. This is especially apparent around the hand on the left side of the image. Here, the warped flow shows the finger motion twice (highlighted part).

The other visible problem arises at image boundaries. In the warped flow of Figure 6.3 (right), we can see a constant color value for some areas near the image boundary, in particular in the top right corner of the image, resulting in hard and unrealistic flow edges. This is most likely due to  $w_{t \rightarrow t-1}$  leaving the image boundaries for each pixel in the area. The warped flow values are then set to that of the outer-most pixel in  $w_{t-1 \rightarrow t-2}$ , resulting in large areas exhibiting the same flow.

We will first discuss the artifacts emerging from the warping operation and then look into the problems at image boundaries.

#### Warping Artifacts

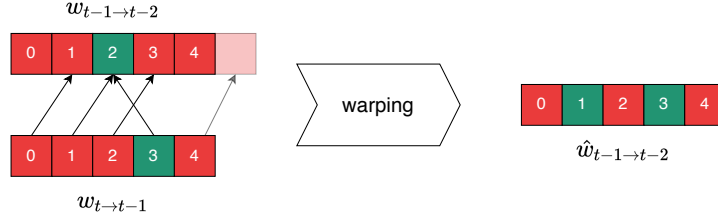
The reason for the artifacts around the hand area is the bad performance of the warping operation around occlusions. In the above figure,  $w_{t \rightarrow t-1}$  and  $w_{t-1 \rightarrow t-2}$  show a movement of the hand from the right to the left. The background, however, shows a movement from the left to the right. As a result, we have positions in  $w_{t \rightarrow t-1}$  from the hand as well as from the background that both evaluate to the position of the hand in  $w_{t-1 \rightarrow t-2}$ .

More precisely, there is a position  $\mathbf{x}_f$  (e.g. at the left-most finger of image  $I_t$ ) and a position  $\mathbf{x}_b$  (e.g. a pixel that is part of the background in  $I_t$ ), such that

$$\mathbf{x}_f + w_{t \rightarrow t-1}(\mathbf{x}_f) \approx \mathbf{x}_b + w_{t \rightarrow t-1}(\mathbf{x}_b), \quad (6.1)$$

and, therefore,

$$\begin{aligned} \hat{w}_{t-1 \rightarrow t-2}(\mathbf{x}_f) &= w_{t-1 \rightarrow t-2}(\mathbf{x}_f + w_{t \rightarrow t-1}(\mathbf{x}_f)) \\ &\approx w_{t-1 \rightarrow t-2}(\mathbf{x}_b + w_{t \rightarrow t-1}(\mathbf{x}_b)) \\ &= \hat{w}_{t-1 \rightarrow t-2}(\mathbf{x}_b). \end{aligned} \quad (6.2)$$



**Figure 6.4:** 1-dimensional illustration of the artifacts caused by the warping operation. Red pixels denote a flow of  $w(\mathbf{x}_{\text{red}}) = 1$ . Green pixels denote a flow of  $w(\mathbf{x}_{\text{green}}) = -1$ . For the sake of simplicity, reflecting boundary conditions are assumed.

Figure 6.4 also illustrates this effect for a small 1-dimensional flow field: Red pixels represent a motion to the right and green pixels a motion to the left. For the sake of simplicity, we assume a motion of exactly 1 pixel at each position per time step. Therefore, there is no sub-pixel precision and we do not need to interpolate flow values. Notice how both the red pixel at position 1 and the green pixel at position 3 of  $w_{t \rightarrow t-1}$  evaluate to the same pixel of  $w_{t-1 \rightarrow t-2}$  (at position 2), resulting in a ‘doubled’ flow value.

Note that, typically, the validity flag of these positions in  $w_{t \rightarrow t-1}$  should be zero due to the bi-directional consistency check failing here. However, the problem arises when we allow the validity of  $\hat{w}_{t-1 \rightarrow t-2}$  to solely depend on the validity of the second backward flow, because there is nothing that prevents the respective position in  $w_{t-1 \rightarrow t-2}$  from potentially being valid. Therefore, if we use the disjunctive validity flag method for training, both positions are considered by the loss function which leads to training on highly inaccurate flow values.

In Section 5.4.2 we introduced the option where  $\hat{c}_{t-1 \rightarrow t-2}$  is always set to zero if  $c_{t \rightarrow t-1}$  is zero. While this would help prevent the above problem, we lose the benefit of the disjunctive validity flag method. In fact, in the conducted experiments this method performed even worse.

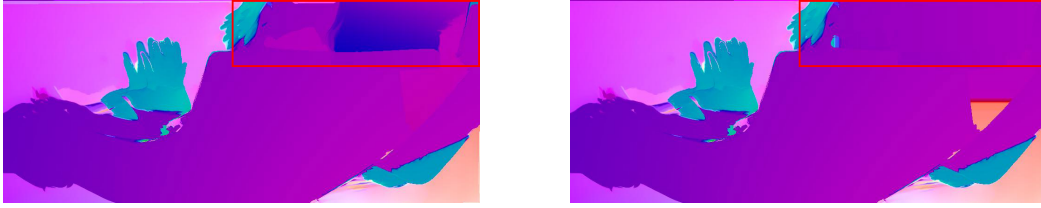
### Image Boundaries

Section 5.4.2 suggested two methods of dealing with positions where  $w_{t \rightarrow t-1}$  leaves the image boundaries. Figure 6.5 shows the resulting flow fields of both methods. For the flow on the left, linear motion was assumed (i.e.  $\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x}) = w_{t \rightarrow t-1}(\mathbf{x})$ ). For the flow on the right, the flow value is set to that at the image boundary in the direction given by  $w_{t \rightarrow t-1}(\mathbf{x})$  (also see Section 5.4.2).

Both images show inaccurate flow values around the image boundary in the marked region. More precisely, the warped flow on the left shows exactly the same motion as in the previous time step (see  $w_{t \rightarrow t-1}$  in Figure 6.3), while the warped flow on the right exhibits unrealistic constant motion in that area, resulting from the clamping at the image boundary. At best, both methods are a rough approximation of the real motion.

Note that in both cases  $\hat{c}_{t-1 \rightarrow t-2}(\mathbf{x})$  is always set to zero. Typically,  $c_{t \rightarrow t-1}(\mathbf{x})$  is also zero at such positions since the bi-directional consistency check fails if  $w_{t \rightarrow t-1}(\mathbf{x})$  leaves the image boundaries. Thus, even with the disjunctive validity flag variant, the loss function does not explicitly learn from





**Figure 6.5:** Impact of the different methods for handling flows in  $\hat{w}_{t-1 \rightarrow t-2}$  at positions where  $w_{t \rightarrow t-1}$  leaves the image boundaries. Left: linear motion assumption. Right: Clamping of  $\mathbf{x} + w_{t \rightarrow t+1}(\mathbf{x})$  at the image boundaries.

these positions. However, due to the convolution filters in the network such a position can still enter the learning process if a neighboring pixel is valid. Therefore, the different variants can have different effects on the overall estimation.

In fact, from these two options, it turns out that the method on the left performs considerably worse than the method on the right. This suggests that (a) these positions still affect the learning process and (b) that the linearity we assumed for the first method does not adequately describe the underlying motion. This coincides with the findings of the extrapolating model in Section 6.1.3.

Both the warping artifacts around occlusions and the problems at image boundaries are the reason that the warped previous backward flow  $\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x})$  was summed up with the first backward flow  $w_{t \rightarrow t-1}(\mathbf{x})$ . Since the above artifacts are only present in  $\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x})$ , the affected positions are partly rectified by the respective positions in  $w_{t \rightarrow t-1}(\mathbf{x})$ . Therefore, this achieves better results than by using  $\hat{w}_{t-1 \rightarrow t-2}(\mathbf{x})$ .

#### Other Difficulties

An additional, general, phenomenon with warping flows (or vectors) via component-wise bilinear interpolation is that magnitudes (i.e. velocities) are not preserved. More precisely, assume two flows  $w_1 = (1, 0)^T$  and  $w_2 = (0, 1)^T$ . Here,  $\|w_1\| = \|w_2\| = 1$ . A linear interpolation of both flows at the center yields  $\hat{w} = (0.5, 0.5)^T$ . Here,  $\|\hat{w}\| \approx 0.7$ , indicating a significantly slower motion. While this is not wrong per se, it is potentially less representative of the actual motion. In most cases, however, this effect is negligible, since neighboring positions are typically locally coherent and, therefore, rarely result in such extreme cases.

Finally, note that the above problems get exacerbated with each additional input frame (i.e. larger input histories). For instance, with  $h = 5$  we need to warp  $w_{t-2 \rightarrow t-3}$  with  $\hat{w}_{t-1 \rightarrow t-2}$  which already contains the artifacts and other problems emerging from the first warping. We are then propagating these inaccuracies to the next time step while, at the same time, introducing more of these artifacts. Thus, with each additional warped flow needed for a larger input history, the contained errors also grow increasingly large.

Model	Input history (h)	Sintel Clean	Sintel Final	KITTI 2015	
		AEE	AEE	AEE	BP
ProFlowS	3	2.23	4.00	5.49	18.00
Separate CNNs	4	1.87	3.67	5.74	17.73
Separate CNNs	5	1.90	3.70	6.15	17.88

**Table 6.8:** Results with separate CNNs on both Sintel passes and KITTI 2015. For a breakdown on individual sequences refer to Appendix A.5.

### 6.3.3 Conclusion

Following the results of this section, we can conclude that the warping strategy of this approach introduces too many errors to the dataset which impairs the learning process. While there might be other warping strategies that could potentially help rectify some of the discussed problems, we are not going to explore this approach further. The previous two approaches have already shown that temporal information of the sequence can be leveraged. Therefore, it makes more sense for us to focus on further improving the architecture of the prediction step instead.

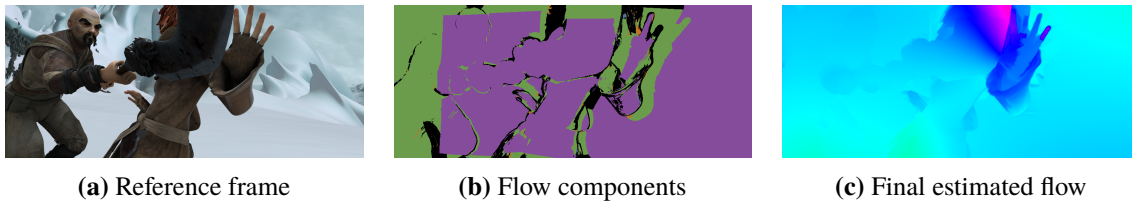
## 6.4 Separate CNNs: Evaluation

Here we are going to look at the performance of a prediction step consisting of separate CNNs as presented in Section 5.5. This approach differs from the previous approaches in that it explicitly handles the combination of information from different time steps.

### 6.4.1 Results

Table 6.8 shows the results of this approach with different input histories on all major benchmarks. We can see that the estimation significantly improves for both Sintel passes and an input history of  $h = 4$ . In the case of Sintel clean, the AEE decreases from 2.23 to 1.87 which is an improvement of over 16% and a competitive estimation despite the relatively poor performance of ProFlowS. On the final pass we have an improvement of almost 10%, lowering the AEE from 4.00 to 3.67. Looking at KITTI 2015, however, we can see a slight increase of the AEE from 5.49 with ProFlowS to 5.74 with four frames. Note that the BP error, on the other hand, decreases for KITTI 2015 which is the primary measurement on all KITTI benchmarks.

Looking at the results with an input history of 5, we can see that the estimation accuracy decreases slightly compared to  $h = 4$  for all benchmarks. For both Sintel passes, the estimation is still better than ProFlowS, however. Similarly, the BP error for KITTI with  $h = 5$  is still slightly smaller than with ProFlowS. We will discuss and further evaluate these results in the following section to get a better understanding of this approach.



**Figure 6.6:** Impact of the baseline and each backward flow on the final estimation. 6.6b shows which prediction is used at what position for  $h = 5$ . Violet pixels: baseline forward flow. Green pixels: prediction from the first CNN (i.e. from  $w_{t \rightarrow t-1}$ ). Yellow pixels: prediction from the second CNN (i.e. from  $w_{t \rightarrow t-2}$ ). Blue pixels: prediction from the third CNN (i.e. from  $w_{t \rightarrow t-3}$ ). Black pixels: inpainting.

### 6.4.2 Discussion

To get a better intuition of how this approach behaves for different input histories, we are first going to look at the way each network’s prediction impacts the final flow estimation. Following that, we will investigate the performance of each of the network predictions, which will then let us explain the overall performance of the here presented approach.

#### Final Flow Components

Figure 6.6b shows which parts of the final flow stem from which CNN for an input history of  $h = 5$ . Violet pixels denote baseline forward flow, black pixels denote inpainting, and the other colors each represent one CNN prediction.

Clearly, with each additional backward flow, the amount of predictions from the respective network grow significantly smaller. While there are plenty of positions where the prediction of the first CNN is taken (green pixels), only very few positions remain for the third CNN (blue pixels). Thus, the potential performance gain decreases rapidly for each additional input frame.

#### Performance of each Prediction

During the evaluation of the previous approach we have seen that the second backward flow ( $w_{t \rightarrow t-2}$ ) contains considerably less valid positions than the first backward flow ( $w_{t \rightarrow t-1}$ ). For reasons of comprehensibility, the total amount of valid positions for both backward flows is listed once more in Table 6.9 with an additional column showing the valid positions for the third backward flow ( $w_{t \rightarrow t-3}$ ). Unsurprisingly, there are even fewer valid positions in  $w_{t \rightarrow t-3}$ . Since each network is trained solely on the respective backward flow and the loss function only considers positions where both forward and backward flow are valid, each additional network is trained on significantly less flow data. Thus, it is likely that the third CNN is unable to sufficiently learn from the training data, leading to a bad network estimation  $w_{t \rightarrow t+1}^{e,3}$ . However, since only few positions are affected by the third CNN (i.e.  $w_{t \rightarrow t+1}^{e,3}$  is used for very few pixels in the final estimation), the total AEE increases only slightly for an input history of 5.

	$\mathbf{c}_{t \rightarrow t+1} \wedge \mathbf{c}_{t \rightarrow t-1}$	$\mathbf{c}_{t \rightarrow t+1} \wedge \mathbf{c}_{t \rightarrow t-2}$	$\mathbf{c}_{t \rightarrow t+1} \wedge \mathbf{c}_{t \rightarrow t-3}$
valid positions (%)	86	82	78

**Table 6.9:** Amount of valid samples (in percent) per backward flow. Note that for the loss function to consider a position, the forward flow has to be valid as well.

used backward flow	AEE
$w_{t \rightarrow t-1}$	4.09
$w_{t \rightarrow t-2}$	4.41
$w_{t \rightarrow t-3}$	4.62

**Table 6.10:** Average endpoint error on Sintel clean for each of the flow predictions when trained on a different backward flow. Note that this error is computed only for the regions where the prediction of the respective CNN is carried over to the final flow estimation.

To further investigate this assumption, the different CNNs are examined individually in the following. More precisely, we evaluate each prediction separately by computing the average endpoint error only for positions where it contributes to the final flow. Therefore, we get an error measure for each CNN which, in turn, tells us the quality of the backward flow used during training.

The results for each prediction is given in Table 6.10. As we assumed, the first CNN (using  $w_{t \rightarrow t-1}$ ) gives the best predictions, while the third CNN (using  $w_{t \rightarrow t-3}$ ) gives the worst flow estimations. Thus, at some point, the error of a particular network prediction is larger than that of inpainting, leading to an overall decrease of accuracy. From the results in Table 6.8 we can see that, in our experiments, this turning point is at  $h = 5$  for both Sintel passes. Note that Table 6.10 also proves that the first backward flow contains the most valuable information.

### 6.4.3 Conclusion

The bad performance of the third CNN explains why the AEE of this approach deteriorates with an input history of  $h = 5$ . For  $h = 4$ , on the other hand, it performs particularly well on both Sintel passes. Most likely, this is owed to the following three things: Firstly, for Sintel, the second backward flow provides enough valid positions after filtering, so that the second CNN can be sufficiently trained on it. Secondly, by separating the backward flows, no invalid flow values are introduced to the training process through the disjunctive validity flag (see Section 5.2). Lastly, the here employed combination step guarantees that as few positions as possible have to be filled-in via inpainting. In contrast to the disjunctive combination step in the previous approach, however, the networks do not have to predict a forward flow from partially invalid backward flows, but instead from a single backward flow that is always valid. Thus, the here presented separate approach outperforms all previous approaches for  $h = 4$ .

Overall, we can conclude that separating the backward flows shows that  $w_{t \rightarrow t-2}$  not only contains complementary information to  $w_{t \rightarrow t-1}$  but also enough data to sufficiently train a network on it. Generally, the here presented approach performs significantly better than combining multiple backward flows into a single input and letting the network handle the extraction of temporal features on its own.

However, there are two big caveats of the separate approach: Firstly, since ProFlow is trained during the estimation process, having, e.g., three separate CNNs that need to be trained, also means a three-fold increase in computational cost for the prediction stage of the pipeline. Secondly, this approach is highly limited by the amount of valid positions in each backward flow. More precisely, while  $w_{t \rightarrow t-3}$  might still contain valuable and complementary temporal information (as seen in Section 6.2), the amount of valid data is just not enough to sufficiently train a network solely on that backward flow. In the following section, we are going to look at the Sparse Convolutional Layer introduced in Section 5.6 which might potentially help reduce the latter problem.

## 6.5 Sparse Convolution: Evaluation

Section 5.6 introduced a general approach for dealing with sparsity in the input data. Since this approach already affects the standard three-frame ProFlow setting, we are first going to look at the results for an input history of 3 and how they were achieved. Following that we are going to show preliminary results with some of the modifications introduced in Sections 5.2 to 5.5.

### 6.5.1 Results With Three Frames

Until now we have trained all networks with the Adam optimizer and standard learning rate and parameters as suggested in [KB15]. For the sparse network in this section, this does not yield decent results, however. Therefore, the learning rate had to be adjusted. More precisely, the network is trained with an exponentially decaying learning rate in addition to the Adam optimizer. The initial learning rate is set to 0.03 and decays every 25 steps (epochs) with a rate of 0.9. The network was then trained over 200 epochs.

Table 6.11 shows the results of this network, denoted as ProFlow-Sparse, on both Sintel passes compared to ProFlowS and the baseline. Note that the best results were achieved with the validity map of the backward flow as the new observation mask. That is, the corresponding forward flow validity was not incorporated into the observation value. Most likely, this is due to the network having to predict flow values for positions where the backward flow is valid but the forward flow is not. Since the same architecture is used during prediction, it makes no sense to require the forward flow to be valid as well. Otherwise the convolutions would ignore positions where we actually need the prediction for.

Evidently, the sparse network achieves the best results and, in particular, significantly outperforms the baseline. However, note that these results were only achieved with an optimized learning rate as described above. This makes a direct comparison to ProFlowS difficult as the latter uses standard parameters. However, it is evident that the Sparse network not only performs well on depth-map estimations but also on optical flow tasks. In the following sections, we will examine ProFlow-Sparse in combination with multi-frame modifications.

Model	Sintel Clean	Sintel Final
	AEE	AEE
baseline	1.94	3.78
ProFlowS	2.23	4.00
ProFlow-Sparse	1.82	3.61

**Table 6.11:** Results of ProFlow-Sparse compared to ProFlowS (standard three-frame setting) and the baseline.

### 6.5.2 Results With Multiple Backward Flows

Following the results of the previous approaches, we are going to look at some combinations of the introduced multi-frame modifications with the presented architecture. First, let us look at using multiple backward flows with Sparse Convolution Layers. Since ProFlow-MBF-d-s performed the best in Section 6.1, we are going to use the same modifications here. More precisely, the loss function considers all positions where any backward flow and the corresponding forward flow are valid (disjunctive method), and for the positions where the prediction is taken, the ProFlowS combination step is used.

Since we use the disjunctive validity flag in the loss function, it makes sense to combine the validity maps of each backward flow into the new observation mask accordingly. Therefore, a convolution only considers positions where at least one of the backward flows is valid (cf. Equation 5.22). Again, the best results were achieved when the corresponding forward validity was not included in the observation value.

#### Results

Table 6.12 shows the results on both Sintel passes compared to the three-frame results of ProFlow-Sparse presented in Table 6.11. We can see that the performance for four frames is generally similar to the performance with three frames but deteriorates slightly. On Sintel clean, the AEE increases by 0.03 (or by 1.6%). On Sintel final the AEE increases by 0.07 (or by 1.9%). For a breakdown on individual sequences refer to Appendix A.6.

#### Discussion

Evidently, we do not get better estimations with larger input histories when combined with the multiple backward flow approach. One problem with the Sparse Convolution network as presented here is that we combine the validity maps of each backward flow into a single observation mask. Therefore, we lose information of partial sparsity in the input data.

More precisely, since we use the disjunctive method for combining the validity flags, if at least one of the backward flows is valid, the position is set to 1 in the observation mask. Therefore, the Sparse network can not differentiate between a position where only one backward flow is valid and a position where both are valid. This likely leads to less optimal training which explains the slightly worse performance on larger input histories.

Model	Input history (h)	Sintel Clean	Sintel Final
		AEE	AEE
ProFlow-Sparse	3	1.82	3.61
ProFlow-Sparse with MBF	4	1.85	3.68

**Table 6.12:** Results of the Sparse network combined with the multiple backward flow model ProFlow-MBF-d-s (see Section 6.1). Note that the results are compared to ProFlow-Sparse and not to ProFlowS.

One possibility to counteract this problem, is to still append the individual validity flag to each backward flow—in addition to the combined observation mask. Furthermore, in the context of a Sparse Convolution with multiple validity flags, it could potentially make sense to use a weighted observation mask instead of a binary mask. That is, depending on how many of the validity flags are valid at a given position, the observation value would then be set to

$$o_{i,j} = \frac{\sum_{k=1}^{h-2} c_{i,j}^k}{h-2}, \quad (6.3)$$

where  $h-2$  is the number of backward flows and  $c_{i,j}^k$  is the validity of the  $k$ -th backward flow at position  $(i, j)$ . Thus, we would get a value between 0 and 1, indicating how many of the respective flow values are valid. A Sparse Convolution would then be weighted accordingly which could potentially improve the learning process.

### 6.5.3 Results With Separate CNNs

Since the Sparse Convolution Layers supposedly work well with particularly sparse data, it makes sense to test them on the separate CNNs approach introduced in Section 5.5. In Section 6.4 we have seen clear improvements for an input history of  $h = 4$ . However, with an even larger input history, the performance suffers, possibly due to the backward flow  $w_{t \rightarrow t-3}$  used for training the third CNN being very sparse. Therefore, we will examine if the here presented Sparse network can help improve the estimation for larger input histories.

Previously, each CNN was based on the ProFlowS architecture presented in Section 4.1. Applying Sparse Convolutions to this method results in multiple CNNs based on the architecture described in Section 5.6. As before, each CNN is trained on a separate backward flow appended by an observation mask which is simply the respective validity map of that backward flow. Thus, we do not need to merge multiple validity flags into a new observation value. Finally, note that each CNN is trained with the same parameters as described above (cf. Section 6.5.1).

Model	Sintel Clean	
	Input history (h)	AEE
ProFlow-Sparse	3	1.82
ProFlow-Sparse with separate CNNs	4	1.85
ProFlow-Sparse with separate CNNs	5	1.89

**Table 6.13:** Results of separate Sparse CNNs on Sintel clean for different input histories. Note that we compare the results to ProFlow-Sparse and not to ProFlowS.

## Results

Table 6.13 shows the results on Sintel clean for different input histories. While the performance difference is only marginal, we can see that increasing the input history results in slightly worse performance. The Sparse network with the standard three-frame setting yields the lowest average endpoint error.

However, note that for some sequences, the estimation accuracy increases with an input history of 4. For more details on this refer to Appendix A.6.

## Discussion

Generally, the observation mask in a Sparse Convolution has a very similar purpose as our particular loss function (cf. Section 4.1.3). The loss function explicitly ignores positions where either the forward flow or the corresponding backward flow is invalid. Similarly, the Sparse Convolution Layer ignores positions where the backward flow is invalid. Clearly, those two methods overlap in the positions that are affected. However, the Sparse network additionally makes sure that no invalid neighboring value enters a convolution. This means that the effective positions the network is trained on are further reduced. While, in theory, it makes sense to ignore neighboring values that are invalid, it might still be beneficial for the network to indirectly learn from such positions, particularly because we later want a prediction on the same dataset.

Additionally note that we used the same learning rate for each network. However, this learning rate was optimized on the first backward flow, i.e. for ProFlow-Sparse. Since each additional backward flow also provides less positions the network can be trained on (cf. Table 6.2), we may need to adjust the training rate accordingly. This is even more so the case than in the standard separate CNNs approach (Section 5.5), because the positions the network is trained on are further reduced due to the observation mask ignoring invalid positions during convolutions. Therefore, the second and third CNN are particularly prone to overfitting, which could be the reason for the slightly worse results on larger input histories.

### 6.5.4 Conclusion

In the presented results, the Sparse Convolution architecture does not perform better on larger input histories than on the standard three-frame setting. Some potential reasons that are specific to the ProFlowS multi-frame setting in the context of Sparse Convolutions were discussed above.



Model	Input history (h)	Sintel Clean	Sintel Final
		AEE	AEE
ProFlowS	3	2.22	4.00
ProFlow-MBF-d-s	4	2.11	4.02
ProFlow-MBF-d-s	5	2.14	4.08
ProFlow-DIFFb-d-s	4	2.09	4.02
ProFlow-DIFFb-d-s	5	2.18	3.96
ProFlow-WARP-d-s	4	2.32	4.06
Separate CNNs	4	<b>1.87</b>	<b>3.67</b>
Separate CNNs	5	1.90	3.70

**Table 6.14:** Overview of the best results in each approach for both Sintel passes.

However, it is important to note that these results merely serve as a general overview of Sparse Convolutions in the context of multi-frame optical flow predictions. Considering the fairly good performance on the Sintel datasets in general, it can be said that Sparse Convolutions can be useful for flow predictions. Therefore, it may be worth to further investigate in this direction, particularly with the additional adjustments suggested in Section 6.5.2.

## 6.6 Summary

Table 6.14 gives a summary over the respectively best performing models for each approach discussed in Sections 6.1 to 6.4. The results are presented for both Sintel passes and different input histories of up to  $h = 5$ .

Note that the last approach (Section 5.6) does not explicitly address multi-frame predictions and the respective models were trained with very different parameters. Additionally, the results presented in Section 6.5 only serve as a first insight into Sparse Convolutions in the context of multi-frame optical flow predictions. For these reasons, the approach is omitted here.

Except for the approach involving a warping operation (ProFlow-WARP-d-s), all multi-frame modifications yield better results on Sintel clean than ProFlowS. Using flow differences to train a single network instead of multiple backward flows seems to be beneficial as it results in lower average endpoint errors on both Sintel clean and Sintel final. Alternatively, training multiple CNNs separately—each on a different backward flow—provides the best results out of all approaches. For both Sintel passes, the separate CNNs approach achieves the lowest errors on an input history of 4. For Sintel clean, the AEE drops from 2.22 in ProFlowS to 1.87. On Sintel final, the AEE drops from 4.00 to 3.67.

Generally, on almost all approaches, the prediction deteriorates slightly for an input history of  $h = 5$ , suggesting that even larger input histories will likely lead to less optimal results. However, it might be possible to adjust the approaches accordingly, for instance, with adaptive learning rates or with the help of Sparse Convolutions and a weighted observation mask (see Section 6.5.2). Overall, we can conclude that optical flow prediction can definitely benefit from multi-frame approaches, at least with reasonable input histories.



## 7 Conclusion and Future Outlook

In the previous chapter we have seen that leveraging temporal information can significantly improve optical flow predictions. Using an adaption of the self-supervised ProFlow architecture [MB18], the average endpoint error was reduced by over 15% on the Sintel clean pass when four frames were used for prediction compared to the standard three-frame setting. Similarly, on Sintel final, we were able to increase the estimation accuracy by almost 10% when more than three frames were used.

In order to examine different multi-frame approaches on the ProFlow architecture, a new and simplified network architecture was presented in Chapter 4, called ProFlowS. With the help of this slightly modified ProFlow pipeline, four different multi-frame approaches were introduced in Chapter 5 and evaluated in Chapter 6.

Two major challenges presented itself when multi-frame modifications are applied to the ProFlow architecture. These are (a) the manner in which valid image positions are selected for training the network and (b) the method of combining predictions with baseline flow estimations. The overall best performing approach uses a separate CNN architecture that solves both challenges by making the extraction of temporal information explicit. More precisely, in this approach each network is trained on a separate backward flow and the predictions are later joined explicitly by prioritizing them according to each network's presumed performance.

However, it is also possible to let the network handle the extraction of temporal information itself. Here, the best performing method uses flow differences as input data, each appended by a separate validity map. Section 6.2 shows that the network is then able to learn temporal information contained in the sequence despite partial sparsity of the input data. While the results are not quite as good as with the separate approach, this model still performs significantly better with larger input histories, demonstrating that temporal information is not only beneficial for optical flow estimations but can also be learned directly by the network.

In addition to evaluating these direct multi-frame modifications, we have also investigated how to handle sparsity in the input data. This is particularly important with larger input histories due to the amount of invalid positions increasing for backward flows that span over larger time steps. This problem was tackled with so-called Sparse Convolution Layers which were first introduced by Uhrig *et al.* [USS+17] in the context of depth-maps. These layers explicitly handle sparsity in the input data with the help of a binary observation map. Applying this method to the three-frame ProFlowS architecture yields good results, suggesting that Sparse Convolutions can also be used for optical flow predictions.

While we have also seen first results with Sparse Convolution Layers in addition to multi-frame modifications in Section 6.5, further experimentation with that architecture is left for future research. The application of a weighted observation map instead of a binary one could particularly be beneficial for multi-frame flow predictions. Similarly, combining different multi-frame modifications into a more comprehensive prediction model could be worth investigating. For instance, in the case of

using separate CNNs, we have seen that a network solely trained on the third backward flow does not provide additional useful flow predictions. Here, one possible solution could be to train this specific network on more than one backward flow or, alternatively, employ an approximating motion model to artificially create more training data. One such method assuming a simple linear motion model was investigated in Section 6.1.3. Results showed, however, that this basic approximation does not sufficiently describe the underlying motion patterns.

Some open questions remain regarding the manner in which to train a network on multiple frames in general. The self-supervised ProFlow architecture uses a baseline approach to acquire the necessary training data. Here, this baseline can be extended to span over multiple time steps which, however, results in very sparse input data. Therefore, most multi-frame approaches on this architecture generally deteriorate with an input history of 5 or more. Thus, it remains to see how to more explicitly leverage the contained information in these highly sparse backward flows. Sparse Convolution Layers could present a promising step in the right direction.

For fully supervised models, on the other hand, the existing training datasets only contain pair-wise ground truth flows. To ensure the learned models are spatially variant, coordinates have to be aligned with the reference frame when more than one flow is used for training. One possible option is to use a warping operation. This, however, introduces artifacts as demonstrated in Section 6.3.

On a final note, despite the clear benefits of capitalizing on the temporal information contained in image sequences, multi-frame approaches are still very underrepresented in modern learning-based optical flow research. The results in this work present an opportunity for further exploration of the underlying concepts.

## Bibliography

- [ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng. “TensorFlow: A system for large-scale machine learning”. In: *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2016, pp. 265–283 (cit. on pp. 31, 35).
- [AP16] A. Ahmadi, I. Patras. “Unsupervised convolutional neural networks for motion estimation”. In: *Proc. IEEE international conference on image processing (ICIP)*. 2016, pp. 1629–1633 (cit. on p. 24).
- [BA91] M. J. Black, P. Anandan. “Robust dynamic motion estimation over time.” In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 91. 1991, pp. 296–203 (cit. on pp. 7, 12, 25).
- [BBPW04] T. Brox, A. Bruhn, N. Papenberg, J. Weickert. “High accuracy optical flow estimation based on a theory for warping”. In: *Proc. European Conference on Computer Vision (ECCV)*. 2004, pp. 25–36 (cit. on pp. 7, 12).
- [BKC17] V. Badrinarayanan, A. Kendall, R. Cipolla. “SegNet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.12 (2017), pp. 2481–2495 (cit. on p. 15).
- [BM10] T. Brox, J. Malik. “Large displacement optical flow: descriptor matching in variational motion estimation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 33.3 (2010), pp. 500–513 (cit. on p. 21).
- [BSL+11] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, R. Szeliski. “A database and evaluation methodology for optical flow”. In: *International Journal of Computer Vision (IJCV)* 92.1 (2011), pp. 1–31 (cit. on p. 19).
- [BW05] A. Bruhn, J. Weickert. “Towards ultimate motion estimation: Combining highest accuracy with real-time performance”. In: *Proc. IEEE International Conference on Computer Vision (ICCV)*. Vol. 1. 2005, pp. 749–755 (cit. on p. 12).
- [BWSB12] 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)*. 2012, pp. 611–625 (cit. on p. 19).
- [DCSF+15] E. L. Denton, S. Chintala, A. Szlam, R. Fergus, *et al.* “Deep generative image models using a Laplacian pyramid of adversarial networks”. In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2015, pp. 1486–1494 (cit. on p. 22).

- [DFI+15] 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: *Proc. IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2758–2766 (cit. on pp. 12, 15, 20–22).
- [GBG12] V. Grabe, H. H. Bülthoff, P. R. Giordano. “On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow”. In: *Proc. IEEE International Conference on Robotics and Automation (ICRA)*. 2012, pp. 491–497 (cit. on p. 12).
- [Gib50] J. J. Gibson. *The Perception of the Visual World*. Riverside Press, 1950 (cit. on p. 11).
- [GLU12] A. Geiger, P. Lenz, R. Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on pp. 12, 19).
- [GRA13] R. Garg, A. Roussos, L. Agapito. “A variational approach to video registration with subspace constraints”. In: *International Journal of Computer Vision (IJCV)* 104.3 (2013), pp. 286–314 (cit. on p. 25).
- [HAGM15] B. Hariharan, P. Arbeláez, R. Girshick, J. Malik. “Hypercolumns for object segmentation and fine-grained localization”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 447–456 (cit. on p. 15).
- [HGDG17] K. He, G. Gkioxari, P. Dollár, R. Girshick. “Mask R-CNN”. In: *Proc. IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2961–2969 (cit. on p. 15).
- [HKH86] B. Horn, B. Klaus, P. Horn. *Robot vision*. MIT Electrical Engineering and Computer Science Series. MIT Press, 1986 (cit. on p. 11).
- [HLS17] Y. Hu, Y. Li, R. Song. “Robust interpolation of correspondences for large displacement optical flow”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 481–489 (cit. on pp. 28, 31).
- [HR19] J. Hur, S. Roth. “Iterative residual refinement for joint optical flow and occlusion estimation”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5754–5763 (cit. on p. 24).
- [HS81] B. K. Horn, B. G. Schunck. “Determining optical flow”. In: *Artificial Intelligence (AIJ)* 17 (1981), pp. 185–203 (cit. on pp. 7, 12).
- [HSL16] Y. Hu, R. Song, Y. Li. “Efficient coarse-to-fine PatchMatch for large displacement optical flow”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 5704–5712 (cit. on p. 28).
- [HTC18] T.-W. Hui, X. Tang, C. Change Loy. “LiteFlowNet: A lightweight convolutional neural network for optical flow estimation”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8981–8989 (cit. on p. 24).
- [IMS+17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox. “FlowNet 2.0: Evolution of optical flow estimation with deep networks”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2462–2470 (cit. on pp. 21, 23).

- [ISI16] S. Iizuka, E. Simo-Serra, H. Ishikawa. “Let there be color! Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification”. In: *ACM Transactions on Graphics (ToG)* 35.4 (2016), pp. 1–11 (cit. on p. 15).
- [ISKB18] E. Ilg, T. Saikia, M. Keuper, T. Brox. “Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation”. In: *Proc. European Conference on Computer Vision (ECCV)*. 2018, pp. 614–630 (cit. on p. 24).
- [JHD16] J. Y. Jason, A. W. Harley, K. G. Derpanis. “Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness”. In: *Proc. European Conference on Computer Vision (ECCV)*. 2016, pp. 3–10 (cit. on p. 24).
- [KB15] D. P. Kingma, J. Ba. “Adam: A method for stochastic optimization”. In: *Proc. International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. 13, 31, 35, 69).
- [KSH12] A. Krizhevsky, I. Sutskever, G. E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105 (cit. on pp. 12, 15, 20).
- [KT15] R. Kennedy, C. J. Taylor. “Optical flow with geometric occlusion estimation and fusion of multiple frames”. In: *Proc. International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*. 2015, pp. 364–377 (cit. on p. 25).
- [LLKX19] P. Liu, M. Lyu, I. King, J. Xu. “SelFlow: Self-supervised learning of optical flow”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4571–4580 (cit. on pp. 24–27).
- [LRS+18] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, B. Catanzaro. “Image inpainting for irregular holes using partial convolutions”. In: *Proc. European Conference on Computer Vision (ECCV)*. 2018, pp. 85–100 (cit. on p. 15).
- [LSD15] J. Long, E. Shelhamer, T. Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440 (cit. on p. 15).
- [MB18] D. Maurer, A. Bruhn. “Proflow: Learning to predict optical flow”. In: *arXiv preprint arXiv:1806.00800* (2018) (cit. on pp. 3, 7, 19, 25, 28–31, 75).
- [MG15] M. Menze, A. Geiger. “Object scene flow for autonomous vehicles”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3061–3070 (cit. on pp. 12, 19).
- [MHG15] M. Menze, C. Heipke, A. Geiger. “Discrete optimization for optical flow”. In: *Proc. German Conference on Pattern Recognition (GCPR)*. 2015, pp. 16–28 (cit. on p. 31).
- [MHR18] S. Meister, J. Hur, S. Roth. “UnFlow: Unsupervised learning of optical flow with a bidirectional census loss”. In: *Proc. AAAI Conference on Artificial Intelligence*. 2018 (cit. on p. 24).
- [MIH+16] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, T. Brox. “A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4040–4048 (cit. on pp. 20, 23).

- [MSB17] D. Maurer, M. Stoll, A. Bruhn. “Order-adaptive and illumination-aware variational optical flow refinement”. In: *Proc. British Machine Vision Conference (BMVC)*. 2017 (cit. on pp. 28, 31).
- [Ng03] T. W. Ng. “The optical mouse as a two-dimensional displacement sensor”. In: *Sensors and Actuators A: Physical* 107.1 (2003), pp. 21–25 (cit. on pp. 7, 12).
- [NH10] V. Nair, G. E. Hinton. “Rectified linear units improve restricted Boltzmann machines”. In: *Proc. International Conference on Machine Learning (ICML)*. 2010, pp. 807–814 (cit. on pp. 14, 31).
- [NŠM18] M. Neoral, J. Šochman, J. Matas. “Continual occlusion and optical flow estimation”. In: *Proc. Asian Conference on Computer Vision (ACCV)*. 2018, pp. 159–174 (cit. on pp. 24, 26).
- [OKK16] A. v. d. Oord, N. Kalchbrenner, K. Kavukcuoglu. “Pixel recurrent neural networks”. In: *Proc. International Conference on Machine Learning (ICML)*. Vol. 48. 2016, pp. 1747–1756 (cit. on p. 15).
- [RB17] A. Ranjan, M. J. Black. “Optical flow estimation using a spatial pyramid network”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4161–4170 (cit. on pp. 7, 22, 23).
- [RDGF16] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. “You only look once: Unified, real-time object detection”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788 (cit. on p. 15).
- [RGS+19] Z. Ren, O. Gallo, D. Sun, M.-H. Yang, E. Sudderth, J. Kautz. “A fusion approach for multi-frame optical flow estimation”. In: *Proc. IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2019, pp. 2077–2086 (cit. on p. 25).
- [RHGS15] S. Ren, K. He, R. Girshick, J. Sun. “Faster R-CNN: Towards real-time object detection with region proposal networks”. In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2015, pp. 91–99 (cit. on p. 15).
- [RSL09] H. Romero, S. Salazar, R. Lozano. “Real-time stabilization of an eight-rotor UAV using optical flow”. In: *IEEE Transactions on Robotics (T-RO)* 25.4 (2009), pp. 809–817 (cit. on p. 12).
- [RWHS15] J. Revaud, P. Weinzaepfel, Z. Harchaoui, C. Schmid. “Epicflow: Edge-preserving interpolation of correspondences for optical flow”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1164–1172 (cit. on p. 28).
- [RYN+17] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, H. Zha. “Unsupervised deep learning for optical flow estimation”. In: *AAAI Conference on Artificial Intelligence*. 2017, pp. 1495–1501 (cit. on p. 24).
- [SB95] S. M. Smith, J. M. Brady. “ASSET-2: Real-time motion segmentation and shape tracking”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 17.8 (1995), pp. 814–820 (cit. on pp. 7, 12).
- [SEZ+14] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun. “Overfeat: Integrated recognition, localization and detection using convolutional networks”. In: *Proc. International Conference on Learning Representations (ICLR)*. 2014 (cit. on p. 15).



- [SRB14] D. Sun, S. Roth, M. J. Black. “A quantitative analysis of current practices in optical flow estimation and the principles behind them”. In: *International Journal of Computer Vision (IJCV)* 106.2 (2014), pp. 115–137 (cit. on p. 22).
- [SS07] A. Salgado, J. Sánchez. “Temporal constraints in large optical flow estimation”. In: *Proc. International Conference on Computer Aided Systems Theory (EUROCAST)*. 2007, pp. 709–716 (cit. on p. 25).
- [SSB10] D. Sun, E. B. Sudderth, M. J. Black. “Layered image motion with explicit occlusions, temporal consistency, and depth ordering”. In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2010, pp. 2226–2234 (cit. on p. 25).
- [SWS+13] D. Sun, J. Wulff, E. B. Sudderth, H. Pfister, M. J. Black. “A fully-connected layered model of foreground and background flow”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 2451–2458 (cit. on p. 25).
- [SYLK18a] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. “Models matter, so does training: An empirical study of CNNs for optical flow estimation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 42.6 (2018), pp. 1408–1423 (cit. on p. 24).
- [SYLK18b] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. “PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8934–8943 (cit. on pp. 7, 23).
- [Tek95] A. M. Tekalp. *Digital video processing*. Prentice Hall Press, 1995 (cit. on pp. 7, 12).
- [TJN98] A. J. Tabatabai, R. S. Jasinschi, T. Naveen. “Motion estimation methods for video compression—a review”. In: *Journal of the Franklin Institute* 335.8 (1998), pp. 1411–1441 (cit. on p. 12).
- [UGVT88] S. Uras, F. Girosi, A. Verri, V. Torre. “A computational approach to motion perception”. In: *Biological Cybernetics* 60.2 (1988), pp. 79–87 (cit. on p. 12).
- [USS+17] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, A. Geiger. “Sparsity invariant CNNs”. In: *Proc. IEEE International Conference on 3D Vision (3DV)*. 2017, pp. 11–20 (cit. on pp. 8, 49–51, 75).
- [VBVZ11] S. Volz, A. Bruhn, L. Valgaerts, H. Zimmer. “Modeling temporal coherence for optical flow”. In: *Proc. IEEE International Conference on Computer Vision (ICCV)*. 2011, pp. 1116–1123 (cit. on pp. 7, 25).
- [WKSL11] H. Wang, A. Kläser, C. Schmid, C.-L. Liu. “Action recognition by dense trajectories”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2011, pp. 3169–3176 (cit. on p. 7).
- [WYY+18] Y. Wang, Y. Yang, Z. Yang, L. Zhao, P. Wang, W. Xu. “Occlusion aware unsupervised learning of optical flow”. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 4884–4893 (cit. on p. 24).
- [YK16] F. Yu, V. Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *Proc. International Conference on Learning Representations (ICLR)*. 2016 (cit. on p. 24).
- [YR19] G. Yang, D. Ramanan. “Volumetric correspondence networks for optical flow”. In: *Proc. Advances in Neural Information Processing Systems (NIPS)*. 2019, pp. 793–803 (cit. on p. 24).

- [ZF14] M. D. Zeiler, R. Fergus. “Visualizing and understanding convolutional networks”. In: *Proc. European Conference on Computer Vision (ECCV)*. 2014, pp. 818–833 (cit. on p. 21).
- [ZIE16] R. Zhang, P. Isola, A. A. Efros. “Colorful image colorization”. In: *Proc. European Conference on Computer Vision (ECCV)*. 2016, pp. 649–666 (cit. on p. 15).
- [ZLB17] S. Zhao, X. Li, O. E. F. Bourahla. “Deep optical flow estimation via multi-scale correspondence structure learning”. In: *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*. 2017, pp. 3490–3496 (cit. on p. 24).
- [ZTF11] M. D. Zeiler, G. W. Taylor, R. Fergus. “Adaptive deconvolutional networks for mid and high level feature learning”. In: *Proc. IEEE International Conference on Computer Vision (ICCV)*. 2011, pp. 2018–2025 (cit. on p. 21).

# A Additional Evaluation Data

In this appendix additional data that complement the evaluations of Chapter 6 are presented. In particular, supplementary and more detailed results on the benchmark datasets for each introduced model are given in the following sections. The order of presentation follows the same order as the models were introduced. We will first look at some generally relevant data, however, before model specific results are discussed.

## A.1 Validity Flags

In order to get a better understanding of the implications of the combined validity flags introduced in Section 5.2.1, a detailed overview of the respective positions considered by the loss functions during training is presented here. Table A.1 shows the amount of valid positions (in percent) for each sequence of the Sintel clean pass. This supplements the excerpt presented in Section 6.1.2.

Generally, as discussed in Section 6.1.2 the disjunctive validity flag contains significantly more valid positions than its conjunctive counterpart which is particularly true for sequences that contain complex motion patterns such as most of the *ambush* sequences.

Furthermore, Table A.1 gives us an indication of the amount of additional (new) flow data contained in the second backward flow. More precisely, note that for most sequences the conjunctive validity flag results in a similar amount of valid positions as  $c_{t \rightarrow t+1} \wedge c_{t \rightarrow t-2}$ . Likewise, the disjunctive validity flag results in a similar amount of valid positions as  $c_{t \rightarrow t+1} \vee c_{t \rightarrow t-1}$ . It follows that the second backward flow can not add many additional positions to the learning process. This seems typically to be the case, suggesting that the first backward flow is generally more important for training the network since the second backward flow evidently contains less valid positions.

This does not mean, however, that the second backward flow does not complement the information of the first backward flow. It merely implies that the second backward flow by itself contains less useful information than the first.

## A.2 Multiple Backward Flow Models

The results of each model discussed in Section 6.1 regarding the approach introduced in Section 5.2 build the foundation for all subsequent approaches, in particular with respect to the loss function and the combination step. Thus, in order to get a better overview of the general performance of each technique, Table A.2 shows the results for all models on each sequence of the Sintel clean pass.

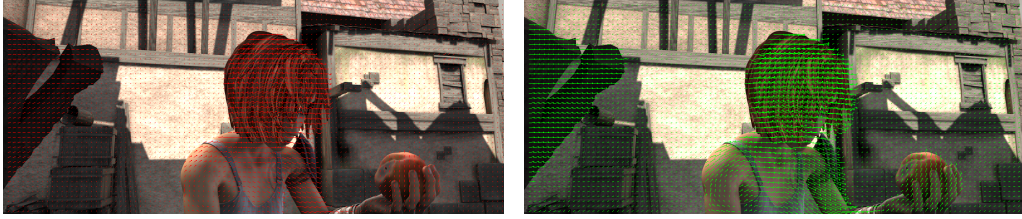
## A Additional Evaluation Data

Sequence	$c_{t \rightarrow t+1} \wedge c_{t \rightarrow t-1}$ valid positions (%)	$c_{t \rightarrow t+1} \wedge c_{t \rightarrow t-2}$ valid positions (%)	Conjunctive Flag valid positions (%)	Disjunctive Flag valid positions (%)
alley_1	96	94	94	97
alley_2	93	91	90	93
ambush_2	60	48	47	61
ambush_4	60	49	47	62
ambush_5	74	66	65	74
ambush_6	64	52	51	65
ambush_7	94	92	92	94
bamboo_1	96	94	93	97
bamboo_2	94	91	91	94
bandage_1	95	92	92	95
bandage_2	96	93	93	96
cave_2	65	54	52	67
cave_4	78	70	68	79
market_2	92	89	88	93
market_5	63	52	51	65
market_6	79	71	70	80
mountain_1	95	94	94	95
shaman_2	99	98	98	99
shaman_3	97	96	96	97
sleeping_1	98	98	98	98
sleeping_2	98	98	98	98
temple_2	87	82	81	87
temple_3	67	58	57	68
all	86	82	81	87

**Table A.1:** Amount of valid positions for each backward flow as well as the two combined validity flag methods for  $h = 4$ . The amounts are given for each sequence of the Sintel clean pass. Note that since the corresponding forward flow  $w_{t \rightarrow t+1}$  serves as the label in training, it always has to be valid for a particular pixel to be potentially considered by the loss function.

The two models employing the conjunctive validity flag method perform by far the worst on all sequences. Contrarily, the two disjunctive models both perform significantly better than ProFlowS. Therefore, it is clear that in order to improve the prediction with multiple frames, we need to consider all positions that are at least partially valid.

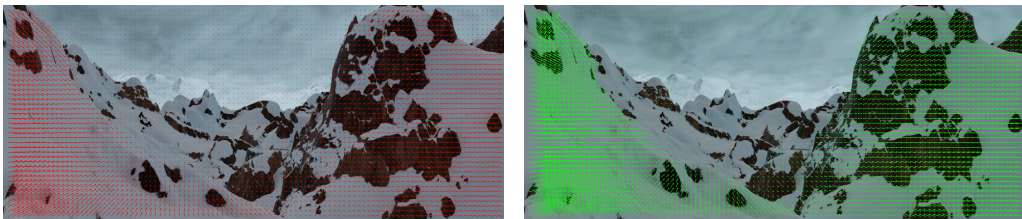
Additionally, Figure A.1 highlights the intuition for the extrapolation method in ProFlow-MBF-e-s. Red arrows denote the first backward flow and green arrows denote the second backward flow. Note that the red arrows are drawn above the green arrows in Figures A.1b and A.1d. Generally, the green arrows have a very similar angle as the red arrows with roughly twice the magnitude, suggesting linearity in the motion. However, while our linear motion model seems like a decent approximation for the displayed sequences, the results in Table A.2 show that it does not sufficiently describe the underlying motion model even for sequences with less complex motion patterns. We can see why when we look at the head in A.1b, for instance. Here, the red and green arrows indicate non-linear motion, explaining the bad performance of ProFlow-MBF-e-s even on ‘simple’ sequences.



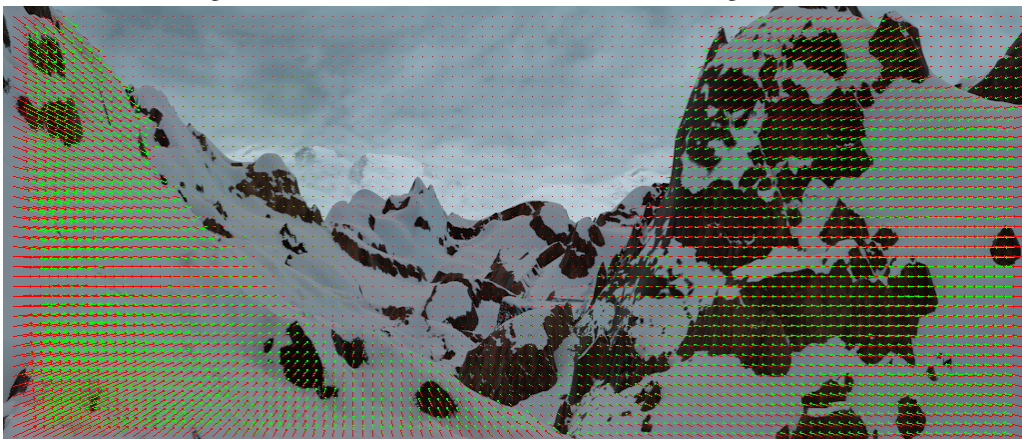
(a) Left: Visualization of the first backward flow (red arrows).  
Right: Visualization of the second backward flow (green arrows).



(b) The first backward flow (red arrows) overlapping the second backward flow (green arrows).



(c) Left: Visualization of the first backward flow (red arrows).  
Right: Visualization of the second backward flow (green arrows).



(d) The first backward flow (red arrows) overlapping the second backward flow (green arrows).

**Figure A.1:** Visualization of the baseline backward flows for  $h = 4$ . Red arrows denote the first backward flow  $w_{t \rightarrow t-1}$ . Green arrows denote the second backward flow  $w_{t \rightarrow t-2}$ . (a-b): frame 5 of the *alley\_1* sequence. (c-d): frame 5 of the *mountain\_1* sequence.

## A Additional Evaluation Data

Sequence	ProFlowS AEE	ProFlow-MBF-c-s AEE	ProFlow-MBF-c-d AEE	ProFlow-MBF-d-s AEE	ProFlow-MBF-d-d AEE	ProFlow-MBF-e-s AEE
alley_1	<b>0.14</b>	0.15	0.15	<b>0.14</b>	<b>0.14</b>	<b>0.14</b>
alley_2	<b>0.17</b>	0.19	0.19	<b>0.17</b>	<b>0.17</b>	0.18
ambush_2	6.48	9.68	9.43	<b>5.92</b>	6.03	6.21
ambush_4	11.72	22.09	21.87	<b>10.94</b>	10.97	11.74
ambush_5	4.12	5.81	5.54	3.92	<b>3.90</b>	4.15
ambush_6	8.72	8.22	8.49	<b>6.89</b>	7.65	8.54
ambush_7	0.57	0.64	0.58	<b>0.53</b>	0.55	0.57
bamboo_1	<b>0.28</b>	0.29	0.29	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bamboo_2	<b>0.65</b>	0.73	0.75	0.66	0.67	<b>0.65</b>
bandage_1	<b>0.28</b>	0.30	0.30	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bandage_2	<b>0.17</b>	0.19	0.19	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
cave_2	6.03	9.42	9.58	<b>5.74</b>	6.15	6.27
cave_4	3.20	3.77	3.95	<b>3.16</b>	3.22	3.17
market_2	0.48	0.55	0.55	<b>0.47</b>	0.48	<b>0.47</b>
market_5	8.42	12.60	12.40	<b>8.23</b>	8.25	9.17
market_6	<b>1.85</b>	2.66	2.72	1.87	1.90	1.86
mountain_1	0.17	0.19	0.19	0.18	0.18	0.18
shaman_2	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>
shaman_3	<b>0.13</b>	0.14	0.14	0.14	0.14	0.14
sleeping_1	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>
sleeping_2	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>
temple_2	2.14	2.62	2.76	<b>2.02</b>	2.03	2.20
temple_3	4.90	8.00	8.00	4.75	<b>4.72</b>	5.43
all	2.22	3.27	3.27	<b>2.11</b>	2.15	2.29

**Table A.2:** Results of the different MBF-models for all sequences of the Sintel clean pass. Except for ProFlowS (three frames), the given results are for an input history of  $h = 4$ .

### A.3 Flow Differences

Table A.3 presents an overview of the general performance of ProFlow-DIFFb-d-s (see Section 6.2) on the Sintel clean pass. The breakdown per sequence shows that ProFlow-DIFFb-d-s generally performs best with an input history of 4.

However, we can see an improvement with a larger input history for some sequences. In particular *ambush\_6* and *temple\_3* perform better on  $h = 5$ . For possible reasons that explain the performance disparity of different input histories refer to Section 6.2.

### A.4 Multiple Warped Backward Flows

Section 5.4 introduced a multi-frame approach that makes use of the warping operation. We later discussed the results of this approach on both Sintel clean and Sintel final. We have seen that the warping operation introduces artifacts into the flow field, in particular around occluded regions and image boundaries. Table A.4 shows the performance of that approach (ProFlow-WARP-d-s) for each sequence of the Sintel clean pass. We can see that ProFlow-WARP-d-s generally performs worse than ProFlowS. More precisely, the optical flow estimation particularly deteriorates for sequences with complex motion patterns such as *ambush\_4* or *market\_5*.

Sequence	ProFlowS	ProFlow-DIFFb-d-s	ProFlow-DIFFb-d-s
	h=3 AEE	h=4 AEE	h=5 AEE
alley_1	<b>0.14</b>	<b>0.14</b>	<b>0.14</b>
alley_2	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
ambush_2	6.48	<b>5.67</b>	6.07
ambush_4	11.72	<b>10.41</b>	12.48
ambush_5	4.12	<b>3.94</b>	4.05
ambush_6	8.72	7.07	<b>6.80</b>
ambush_7	0.57	0.55	<b>0.54</b>
bamboo_1	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bamboo_2	<b>0.65</b>	0.67	0.67
bandage_1	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bandage_2	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
cave_2	6.03	<b>5.76</b>	5.89
cave_4	3.20	<b>3.14</b>	3.18
market_2	0.48	<b>0.46</b>	0.47
market_5	8.42	<b>8.36</b>	8.60
market_6	1.85	<b>1.76</b>	1.84
mountain_1	<b>0.17</b>	0.19	0.18
shaman_2	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>
shaman_3	<b>0.13</b>	0.14	0.14
sleeping_1	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>
sleeping_2	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>
temple_2	2.14	1.90	<b>1.88</b>
temple_3	4.90	4.84	<b>4.72</b>
all	2.22	<b>2.09</b>	2.18

**Table A.3:** Results of ProFlow-DIFFb-d-s per sequence on the Sintel clean pass with different input histories compared to ProFlowS.

Looking at Table A.1, we can see that these sequences generally exhibit only few valid positions after the bi-directional consistency check. This, in turn, suggests that they contain numerous occlusions. Additionally, these sequences typically contain large motions which means that the flow at many positions leaves the image boundaries. Since both properties lead to more warping artifacts in ProFlow-WARP-d-s, it is no surprise that it performs significantly worse on these sequences. This further supports the reasoning given in Section 6.3.

## A.5 Separate CNNs

In Section 6.4 we have evaluated an approach that trains a separate CNN on each backward flow and later combines the predictions explicitly during the combination step of the ProFlow pipeline. This multi-frame approach generally achieved the best results on all benchmarks. For the sake of completeness, Table A.5 presents a breakdown on each sequence of the Sintel clean pass.

Sequence	ProFlowS	ProFlow-WARP-d-s
	AEE	AEE
alley_1	<b>0.14</b>	0.15
alley_2	<b>0.17</b>	0.18
ambush_2	<b>6.48</b>	6.59
ambush_4	<b>11.72</b>	13.12
ambush_5	<b>4.12</b>	4.20
ambush_6	8.72	<b>7.23</b>
ambush_7	<b>0.57</b>	0.60
bamboo_1	<b>0.28</b>	0.29
bamboo_2	<b>0.65</b>	0.74
bandage_1	<b>0.28</b>	0.29
bandage_2	<b>0.17</b>	<b>0.17</b>
cave_2	<b>6.03</b>	6.18
cave_4	<b>3.20</b>	3.38
market_2	0.48	<b>0.46</b>
market_5	8.42	9.55
market_6	<b>1.85</b>	2.07
mountain_1	<b>0.17</b>	<b>0.17</b>
shaman_2	<b>0.13</b>	<b>0.13</b>
shaman_3	<b>0.13</b>	<b>0.13</b>
sleeping_1	<b>0.07</b>	<b>0.07</b>
sleeping_2	<b>0.05</b>	<b>0.05</b>
temple_2	2.14	<b>2.03</b>
temple_3	4.90	4.82
all	<b>2.22</b>	2.32

**Table A.4:** Results of ProFlow-WARP-d-s per sequence on the Sintel clean pass for an input history of  $h = 4$  compared to ProFlowS.

Evidently, this approach performs significantly better than ProFlowS on nearly all sequences. However, this approach performs consistently better with an input history of 4 than with an input history of 5, suggesting that the prediction of the third CNN is generally worse than inpainting.

## A.6 Sparse Convolution Models

Table A.6 supplements the results presented in Tables 6.11 to 6.13. The average endpoint error is given for each sequence of the Sintel clean pass and an input history of 4, except for ProFlow-Sparse which uses the standard three-frame setting.

As discussed in Section 6.5, the three-frame setting generally performs the best with a Sparse network. However, both the multiple backward flow method and the separate CNNs approach perform better on at least some sequences.



Sequence	ProFlowS	Separate CNNs	Separate CNNs
	h=3 AEE	h=4 AEE	h=5 AEE
alley_1	<b>0.14</b>	<b>0.14</b>	<b>0.14</b>
alley_2	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
ambush_2	6.48	<b>5.94</b>	5.96
ambush_4	11.72	<b>8.88</b>	9.01
ambush_5	4.12	<b>3.70</b>	3.77
ambush_6	8.72	<b>6.50</b>	6.58
ambush_7	0.57	<b>0.52</b>	0.53
bamboo_1	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bamboo_2	<b>0.65</b>	0.66	0.67
bandage_1	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bandage_2	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
cave_2	6.03	<b>4.81</b>	4.86
cave_4	3.20	<b>3.04</b>	3.09
market_2	0.48	<b>0.44</b>	0.44
market_5	8.42	<b>7.13</b>	7.31
market_6	1.85	<b>1.60</b>	1.67
mountain_1	<b>0.17</b>	0.18	0.18
shaman_2	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>
shaman_3	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>
sleeping_1	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>
sleeping_2	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>
temple_2	2.14	<b>1.66</b>	1.69
temple_3	4.90	<b>4.17</b>	4.28
all	2.22	<b>1.87</b>	2.90

**Table A.5:** Results of the separate CNNs approach for different input histories compared to ProFlowS on all sequences of the Sintel clean pass.

Sequence	ProFlow-Sparse	Sparse with MBF	Separate Sparse CNNs
	AEE	AEE	AEE
alley_1	<b>0.14</b>	0.15	<b>0.14</b>
alley_2	<b>0.16</b>	0.17	<b>0.16</b>
ambush_2	5.63	5.78	<b>5.56</b>
ambush_4	<b>8.45</b>	9.36	8.79
ambush_5	<b>3.58</b>	3.38	3.65
ambush_6	6.15	<b>5.19</b>	6.12
ambush_7	0.47	0.52	0.51
bamboo_1	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bamboo_2	<b>0.64</b>	0.67	0.67
bandage_1	<b>0.28</b>	<b>0.28</b>	<b>0.28</b>
bandage_2	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
cave_2	4.83	4.91	<b>4.80</b>
cave_4	<b>3.02</b>	3.05	3.07
market_2	<b>0.43</b>	0.44	0.44
market_5	<b>6.92</b>	7.06	7.09
market_6	1.68	1.71	<b>1.66</b>
mountain_1	<b>0.17</b>	<b>0.17</b>	<b>0.17</b>
shaman_2	<b>0.13</b>	<b>0.13</b>	<b>0.13</b>
shaman_3	<b>0.13</b>	0.14	<b>0.13</b>
sleeping_1	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>
sleeping_2	<b>0.05</b>	<b>0.05</b>	<b>0.05</b>
temple_2	1.63	<b>1.62</b>	1.65
temple_3	3.97	4.21	4.20
all	1.81	1.85	1.85

**Table A.6:** Results of the Sparse Convolution Layers applied to ProFlowS combined with multi-frame modifications on each sequence of the Sintel clean pass. *Sparse with MBF* refers to the multiple backward flows approach (cf. Section 5.2) in combination with Sparse Convolutions. Similarly, *Separate Sparse CNNs* refers to the separate CNNs approach presented in Section 5.5 in combination with Sparse Convolutions. The results are shown for  $h = 4$  and compared to ProFlow-Sparse which uses the standard three-frame setting (i.e. an input history of  $h = 3$ ).

### **Declaration**

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

---

place, date, signature