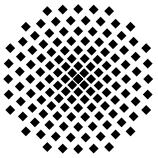


University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Master thesis

Unsupervised Monocular Scene Flow Estimation and Prediciton

Author: Nicolas Riess

Date of work begin: 01.12.2021
Date of submission: 31.05.2022

Supervisor: Dr. rer. nat. Alexander Braun
Dr.-Ing. Christoph Weinrich
M. Sc. George Eskandar
Prof. Dr.-Ing. Bin Yang

Keywords: Deep learning, Computer vision, Transformer, Scene flow estimation, Scene flow prediction

Abstract - Capturing the scene flow, which represents the displacement of an object's surface points between two points in time, can be supportive in numerous applications such as autonomous driving. For economic reasons, it is desirable to accomplish this task with only one camera, which is why monocular scene flow estimation is of high relevance. Since ground truth is difficult to obtain, we aim to improve an unsupervised monocular scene flow estimator. Therefore, a baseline based on [1] is created. In this work, we propose a new architecture leveraging transformers for scene flow estimation and prediction. In [1], computing a correlation between features and their warped version using an estimated scene flow is suggested. Contrary to the correlation, the Swin transformer can process multiple input features at once and can efficiently process long time series in parallel. Therefore, the correlation is replaced by a Swin transformer.
Moreover, the scene flow estimator is extended to additionally perform scene flow prediction to enable farsightedness. Thereby, the strength of the Swin transformer for processing multiple past frames is taken advantage of.

Contents

1. Introduction

1.1. Motivation

Enabling autonomous driving has numerous possible applications. One application is the transportation of goods and passengers. The global revenue in the ride-hailing and taxi segment will amount to approximately 260.19 billion Euro in 2022 [2] which shows the enormous economic potential. In addition, privately-owned autonomous vehicles can help physically disabled, frail elderly people as well as children to become more mobile. This self-sufficiency is particularly important in the life of physically handicapped people because they often depend on the help of nursing staff or relatives in everyday life.
According to Eurostat, 77.4% of the freight transport based on tonne-kilometers in the European Union uses roads in the year 2020 [3]. A turnover of 324.46 billion Euro were generated for freight transport in 2021 in the European Union alone according to the German traffic newspaper, Deutsche Verkehrs-Zeitung [4], which also brings the potential for big savings by replacing truck drivers with autonomous systems. Additionally, repeated truck driver shortages cause high economic damage and exacerbate supply chain issues as described by Insurance Newsnet [5]. All of these industry problems could be mitigated by the introduction of more autonomous trucks.
Furthermore, autonomous delivery robots could meet the growing demand for shipping driven by growing e-commerce. Autonomously driving robots can be used as service robots in private or in the healthcare sector or to complete tasks in environments that are dangerous or hostile to humans, such as in radioactive areas or on other planets. To conclude, the use cases of autonomous driving are numerous and they show that autonomous driving can increase the standard of living of people worldwide.

In autonomous driving, it is necessary to detect and track objects to react to the behavior of other road users and changes in the driving environment. This can be done based on camera images. Visual Object detection consists of localizing the object in the image and assigning it to a specific category. First, informative regions, which are often bounding boxes, are selected. Bounding boxes are ideally the smallest possible rectangles that completely enclose the individual objects to be detected. Subsequently, features are extracted and lastly the objects are classified [6].
For improved classification, the scene flow, which is the three-dimensional motion field of points in the world [7], can also be estimated parallel to the feature extraction and treated as an additional feature. Using the scene flow as an additional feature for classification could be advantageous because the scene flow contains spatial information such as depth, and temporal information. Furthermore, the way an object moves provides information about what kind of object it could be. For example, tree trunks do not move and cars do not move sideways. That way, classes could be ruled out or at least considered unlikely. A plurality of objects is usually detected in the process of object detection. That is why it is often followed by

multiple target tracking which includes the determination of the object's states. In order to estimate the current and future states, a Kalman filter is typically used [8]. At this point, the scene flow could act as an additional input for the Kalman filter.

Ensuring the safety of all road users is one of the biggest hurdles of autonomous driving. A key aspect of ensuring safety is to predict the future relative locations of other road users to the ego-vehicle and objects in the surroundings to forecast critical traffic situations which can be achieved by driving with foresight. Farsightedness makes it more likely to expect the intent of a person to cross a street. For example, an approaching autonomous vehicle expects that the person wants to cross the street if the person looks towards the opposite sidewalk and starts walking towards it. Consequently, the vehicle brakes and can thereby prevent an accident. An even higher degree of farsightedness is required when children, for whom a sudden change in direction and speed of movement is particularly likely, are involved in road traffic.
In general, foresighted anticipatory driving also has many advantages besides increased safety such as a reduction in energy consumption and increased driving comfort. Some of the numerous examples of this will now be given. Energy can be saved by not accelerating further when the cars in front start to slow down or when approaching a busy roundabout. If a traffic light has been green for a long time, it may indicate that it is about to turn red. This allows for early, smooth braking rather than jerky braking, which increases energy consumption, driving comfort as well as safety because rear-end collisions can be avoided in this way. In addition, slow braking reduces the stress on mechanical components of the vehicle and tire abrasion, resulting in less tire wear and less impact on the environment. Furthermore, sudden vertical movements of vehicles in front indicate a deterioration in the road condition and the necessity of the reduction of the own speed. Avoiding the experience of strong self-acceleration in the vertical direction makes driving more comfortable and reduces stress on mechanical components.
As can be seen from these examples, anticipatory driving is closely linked to predicting the relative locations of other road users to the ego-vehicle and objects in the surroundings. The relative velocity of other road users gives an indication of their future location. Since it can be computed based on the scene flow like [9] showed in the case of a vehicle, the scene flow can be seen as a key variable for autonomous driving. However, there are situations in which it is not sufficient to consider the relative speed in order to predict the future position of an object or road user. This is particularly the case for road users experiencing non-zero acceleration. Apart from inferring the future position on the basis of a relative speed derived from an estimated scene flow of objects from past frames, the future position could also be directly predicted using the prediction of a future scene flow. In general, scene flow leads to an improved understanding of a scene and is a relevant quantity for autonomous driving, object recognition, object tracking, action recognition, and scene segmentation [10].

In order to estimate the current and predicted future scene flow, an accurate perception of the environment is important. The most accurate measurement is done by fusing a whole range of different sensor data, whereby the different strengths of the sensor types are not only taken into account but explicitly exploited. For example, distances can be reliably measured with radar (radio detection and ranging) sensors even in unfavorable weather conditions. Lidar (light detection and ranging) sensors, on the other hand, provide more accurate distances with higher resolution than radar. However, they are more expensive and the precision of their measurements suffers greatly under unfavorable weather conditions such as fog or

snow. Compared to other sensors, ultrasonic sensors are best suited for measuring particularly short distances. Therefore, their measurements are often used to make parking easier for the driver. Although they do not provide explicit depth information, cameras are often used because unlike other sensors, they provide information about the colors of the objects in the environment. Ultimately, the scene flow estimators often rely on a stream of data from multiple different sensors, whereby the data stream from any sensor should be robust to temporal interruptions to achieve a safe behavior.

However, there is potential for an increase in robustness and there are a lot of possible causes for sensor failures or impairment possibly leading to poor or no scene flow estimates at all. For instance, in the case of a stereo camera set-up, it is likely that one camera fails because of dirt, certain weather conditions, or mechanical stress. Consequently, it is favorable to have an algorithm, which only requires the data of one remaining functional camera to estimate the scene flow with still sufficient accuracy, in such situations. The usage of only one camera will be referred to as monocular. The difficulty here is that the depth cannot be unambiguously reconstructed from monocular images in a dynamic scene. With images from a single camera, it is still theoretically possible to drive autonomously, since humans are also able to drive using only one eye. Humans do this by considering context information and by being able to estimate from experience how large objects are and thus how far away objects are in the respective situation. It is therefore a long-term goal to enable autonomous driving based on monocular images. Additionally, enabling monocular scene flow estimation comes with further advantages like a broader field of application on devices like smartphones. Moreover, monocular scene flow estimation can be applied to robots. If these robots do not take on any safety-critical tasks, there is no need for redundant sensors. Consequently, they are cheaper to build, weigh less, and have more assembly space left.

1.2. Contribution and Outline

The aim of this thesis is to create a scene flow estimator that offers a better basis for predicting the scene flow than the previously existing scene flow estimators. In order to be able to reliably predict the future, it is often necessary to know and process a history of the past. In the case of visual input, it is necessary to consider numerous past frames. The greater the number of known frames, the higher the order of time derivatives that can be theoretically determined based on those frames. For example, the average speed based on two measured distances to a fixed reference point at different times can be determined but not the average acceleration. A third measurement is required for this. Being able to approximate higher-order time derivatives could be advantageous for scene flow prediction. Since transformers are particularly good at computationally effective processing of long time series [11], they will be integrated into the scene flow estimator of the paper with the title "Self-Supervised Monocular Scene Flow Estimation" written by Junhwa Hur and Stefan Roth from the TU Darmstadt [1]. The main contributions of this thesis are the

- improvement of an existing scene flow architecture in terms of percentage of scene flow outliers by computing a feature pyramid based on two subsequent frames instead of computing two feature pyramids for each frame separately
- incorporation and modification of the transformer into the existing architecture such

that it produces scene flow estimates with comparable quality to the original framework

- evaluation of the influence of several architecture modifications and parameters of the transformer on the quality of estimated scene flow
- introduction and optimization of a novel scene flow predictor that leverages the strengths of the transformer
- improvement of the accuracy of the predicted scene flow due to the integration of a transformer .

By successively modifying the architecture of the scene flow estimator and especially the architecture of the integrated transformer, it will be shown which adjustments have a positive influence on the estimation accuracy and how these can be explained.

Chapter 2, motivates which of the state-of-the-art monocular scene flow estimators are used as a baseline. Since artificial neural networks are often used for monocular scene flow estimation, the basics of the artificial neural networks and in particular the transformer neural networks will be dealt with in chapter 3. The basics of scene flow estimation, including the geometric and mathematical relationships, are also covered in chapter 4. After presenting commonly used datasets in chapter 5, an evaluation metric will be introduced in section 6.1. An overview of the most important experiments is given in Fig. 1.1. First a baseline will be created in section 6.2, the effect of concatenating the inputs will be shown in section 6.3, a transformer will be integrated and optimized in the remaining sections of chapter 6. The optimized method using a transformer will be extended to additionally perform scene flow prediction in chapter 7. Thereby, the scene flow predictor will be optimized by varying its architecture and it will be demonstrated that the integration of the transformer contributes to an improvement of the predicted scene flow. In this work, a distinction is made between estimation and prediction. The term estimation always refers to the computation of quantities that have the same timestamp as the images on which they are based and which are already known during inference. In contrast, the term prediction will be used to refer to quantities with a timestamp that is further in the future than the images known during the inference. Finally, the results will be summarized and an outlook on future developments will be given in chapter 8. An ablation study performed on the original baseline can be found in the appendix. Additionally, a detailed evaluation of the scene flow estimator and predictor based on several different traffic scenes is part of the appendix.

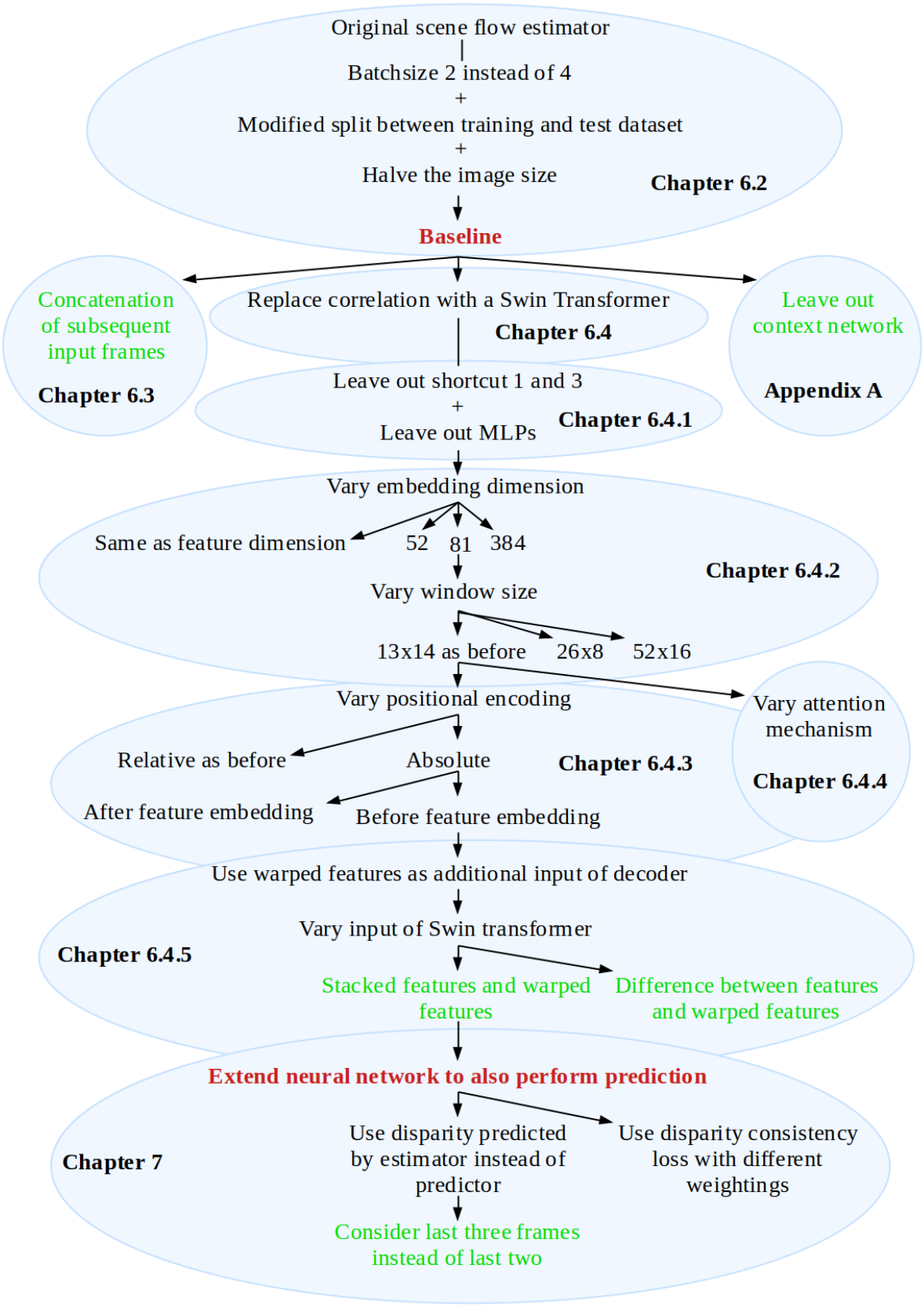


Figure 1.1.: Overview of the most important experiments
The versions which are particularly powerful according to different criteria presented later are marked in green

2. State-of-the-art unsupervised monocular scene flow estimator and predictor

State-of-the-art unsupervised monocular scene flow estimators usually consist of convolutional neural networks because of their computational efficiency and speed compared to other methods such as variational method which will be described in chapter 3. For autonomous driving data sets that contain real street images such as the KITTI dataset described in chapter 5 are particularly suitable because they contain domain-specific data. Another advantage of this dataset is that a benchmark [12] exists that compares and ranks several scene flow estimators according to standardized error measures. An overview of these approaches is given in Fig. 2.1. In the following, some approaches of this ranking are introduced and finally, an approach is sought that is the most suitable for this work. A prerequisite for the baseline is that code is available that can be directly adopted and built on. Another prerequisite is that there exists a paper that explains what was done in the code and why it was done. Among the approaches that meet these requirements, those that use the ground truth depths or stereo images contained in the data set during the inference are not considered.

The best unsupervised monocular scene flow estimator Mono-SF [13] in terms of scene flow outliers, a metric explained in details in section 6.1, combines an energy-based formulation (refer to chapter 3) with a convolutional neural network. In Mono-SF a convolutional neural network is first trained to predict a probabilistic depth using labeled depth ground truth. Subsequently, the scene is decomposed in rigid bodies as well as piecewise planar surface elements by minimizing an energy functional. Thereby, the 6D motion of the rigid bodies is estimated. This approach produces only 23.08% scene flow outliers. However, the runtime is $41s$ using an RTX 2080 Ti graphic card which makes it unsuitable for tasks like autonomous driving in which the traffic situation can change entirely in just a few seconds.

The unsupervised monocular scene flow estimator with the second smallest percentage of scene flow outliers is called MonoComb [14]. In this approach, the depth and optical flow are predicted separately with different off-the-shelf depth and optical flow networks. By warping one image to the other, the scene flow, which has 28.14% scene flow outliers, is estimated. The process of warping and the meaning of the optical flow and scene flow will be defined in the sections 4.1 and 4.2. The runtime of Monocomb is 0.58s using an RTX 2080 Ti graphic card.

Another scene flow estimator [1] generates a feature pyramid for each of two subsequent camera frames with a convolutional network and loops through the levels of these pyramids. In each level a scene flow is estimated based on the estimated scene flow of the last level. By adding the new estimated residual scene flow to the scene flow of the last level, the estimation is gradually refined. The detailed framework will be discussed in more detail in section 6.2.

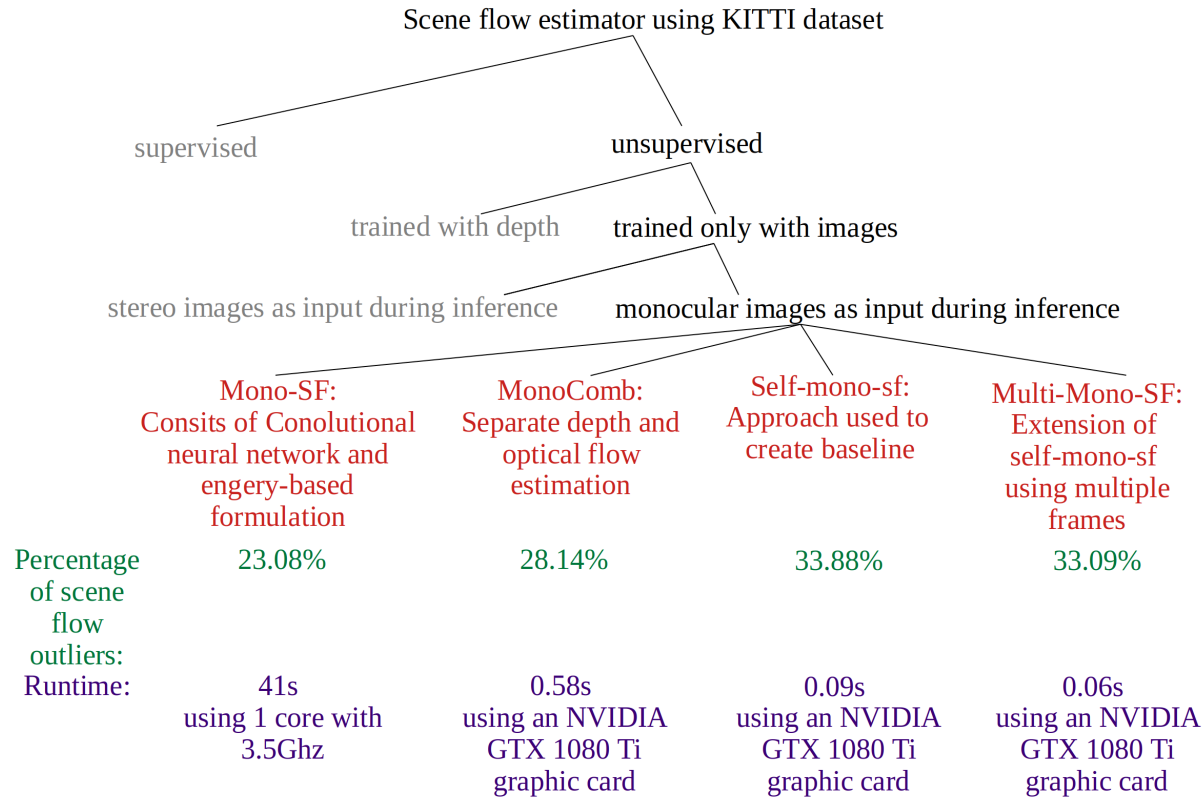


Figure 2.1.: Classification of scene flow estimators and overview of discussed state-of-the-art unsupervised monocular scene flow estimator

Although this estimator produces 33.88% scene flow outliers, which is more than Mono-SF and MonoComb generated, it has a runtime of only 0.09s using an NVIDIA GTX 1080 Ti graphic card.

The authors of the paper "Self-Supervised Monocular Scene Flow Estimation" [1] also extended their framework in such a way that it considers three subsequent frames [15]. Additionally, they employ convolutional LSTMs. The resulting framework is 0.03s faster and slightly more precise with regard to the percentage of scene flow outliers, which is 33.09%, than the previous version. However, [1] is used as a baseline in this work because it is more general, less sophisticated, and less complex. Therefore, it is more easily adaptable. More information about state-of-the-art methods can be found in sections 4.4 and 4.5.

In this work, scene flow prediction refers to a scene flow between the point cloud of a current and a future frame as stated in section 1.2. Numerous papers, such as [16], can be found that carry out scene flow prediction according to their definition. However, those papers only estimate the scene flow between the point cloud of two current frames instead of performing scene flow prediction according to our definition.

3. Artificial neural networks

Traditionally, variational methods are employed for computer vision tasks such as scene flow estimation [17,18]. In these methods, energy functionals, which depend on the scene flow, are set up and minimized. The minimum of these functionals can be found by solving the Euler-Lagrange equations. To do this, they are usually discretized and solved iteratively with the Gauß-Seidel or Jacobi Method. Thereby, several thousand iterations are often required [19] frequently making variational methods too slow for real-time applications.
In contrast with neural networks, optimization takes place during the training phase. Therefore, neural networks are often several orders of magnitudes faster than variational methods during inference making neural networks suitable for autonomous driving. An example of this is Flownet, a convolutional neural network, and its successors, which approach state-of-the-art variational methods in terms of the accuracy of optical flow estimation [20]. Optical flow is a quantity related to the scene flow and will be described in detail in section 4.1. Furthermore, the convolutional neural networks allow the extraction of more abstract and multiscale image features than variational methods [21]. These features can be used for different tasks during or after the training, which leads to more flexibility compared to the variational methods.

After first introducing the fundamental structure of an artificial neural network and the way it is trained, convolutional neural networks are presented. Due to their effectiveness in capturing long-term dependencies, transformer neural networks were first used in domains such as natural language processing in which time series or sequential data is processed. Soon, transformers proved to be beneficial for computer vision tasks. Therefore, transformer neural networks and their adaptation to computer vision tasks are discussed.

3.1. Fundamental structure

Artificial neural networks, which are inspired by the way in which the human brain processes information [22], lately have received a lot of attention because of their ability to approximately learn complex mappings between a high dimensional input and output space. In a mathematical sense, such mappings are referred to as functions. Compared to other nonlinear modeling techniques, neural networks can fit arbitrarily complex nonlinear models which do not have to be specified in advance [23]. In addition, theorems such as the universal approximation theorems [24, 25] imply that neural networks with suitable weights can approximate a wide variety of functions. This fact makes neural networks an attractive tool that can be used across numerous domains, including computer vision.

A neural network can be represented as function $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\Theta})$ that maps the input $\mathbf{x}$ to the output $\mathbf{y}$ using the parameters $\boldsymbol{\Theta}$. A feedforward neural network consisting of an input layer,

an output layer, and an arbitrary number of hidden layers is often called a multilayer perceptron (MLP). These layers in turn each consist of at least one neuron. If every neuron of one layer is connected to every neuron of the neighboring layers, the neural network is called dense. Now the way in which the individual neurons are connected to each other is explained in more detail. In the following, the assumption is made that $\mathbf{x} \in \mathbb{R}^d$ is one dimensional, although it can also have multiple dimensions. A single neuron of the layer L computes a weighted sum of its inputs x^i_{L-1} using the weights w^i_L, adds a bias $\mathbf{b}_L$ and passes the result $\mathbf{a}_L$ through an activation function Φ. The equation

$$\mathbf{x}_L = \Phi(\mathbf{a}_L) = \Phi\left(\sum_i w^i_L x^i_{L-1} + \mathbf{b}_L\right) \tag{3.1}$$

reflects this procedure. The union set of the weights w^i_L and biases $\mathbf{b}_L$ equals the set of parameters $\boldsymbol{\Theta}$. In order to be able to approximate generic continuous nonlinear functions, there has to be at least one hidden layer with a nonlinear activation function [26]. Some of the most popular activation functions are the Hyperbolic Tangent (tanh)

$$\mathbf{x}_L = \tanh(\mathbf{a}_L) \quad , \tag{3.2}$$

the Sigmoid

$$\mathbf{x}_L = \frac{1}{1 + \mathrm{e}^{-\mathbf{a}_L}} \quad , \tag{3.3}$$

the Rectified Linear Unit(ReLU)

$$\mathbf{x}_L = \max(\mathbf{0}, \mathbf{a}_L) \quad , \tag{3.4}$$

and its modifications like the LeakyReLU

$$\mathbf{x}_L = \max(\alpha \cdot \mathbf{a}_L, \mathbf{a}_L),\ \alpha \in (0, 1) \quad . \tag{3.5}$$

In the equations (3.4) and (3.5), the maximum is calculated component by component. The sigmoid activation function is often used in the output layer of a classifier because its outputs are in the range of 0 and 1 and can therefore be interpreted as probabilities for predicted classes. Like the sigmoid activation function, the tanh can suffer from vanishing gradients due to its saturation. ReLU partially solves this problem because its derivative is not zero or close to zero for positive inputs. In contrast to ReLU, leakyRELU retains the negative part of feature information. The extent to which this information is retained can be set by the parameter alpha. A bigger value of alpha increases the degree of consideration of the negative part of the information whereas a smaller value leads to a sparser network [27].

3.2. Training

The goal of the training is to improve the performance of the neural network. Before the actual training begins, the weights have to be initialized. They should not all have the same value in order to avoid the neurons making the same computations and thus effectively reducing the number of neurons. Hence, an asymmetric initialization is desirable which can be

achieved through a random initialization. Additionally, the initial weights should not be too large or too small to avoid big values for the output y and resulting in exploding or vanishing gradients during backpropagation. For instance, weights can be initialized with values sampled from a normal distribution

$$w_L^{i,j} \sim \mathcal{N}(0,1) \quad . \tag{3.6}$$

Another strategy is to adapt the standard deviation

$$\sigma \sim \sqrt{\frac{1}{n_{in}}} \tag{3.7}$$

according to the number of input neurons n_{in} which leads to a constant activation flow in the forward pass meaning that the input and output of a layer both have the same variance. Similarly, a constant activation flow in the backpropagation can be achieved by adapting the standard deviation according to the number of output layers n_{out}. The Glorot initialization [28] with the variance

$$\sigma \sim \sqrt{\frac{1}{n_{in} + n_{out}}} \tag{3.8}$$

is a compromise between those goals.

After initialization, the input is mapped to the output which is known as forward pass. Subsequently, the computed output is evaluated. This evaluation is performed with a loss function comparing the desired output with the output of the neural network. The loss function has to be defined in such a way that it takes small values for a desirable output. Therefore, the aim is to find the global minimum of the loss. This can be done by computing the gradient of the loss with respect to the individual parameters $\mathbf{\Theta}$ by using the chain rule and by adjusting the individual parameters considering the gradients. Gradient descent does this, for instance. Considering the exemplary objective function J which is a more general term for a loss function the update rule of batch gradient descent is

$$\mathbf{\Theta}_{k+1} = \mathbf{\Theta}_k - \eta \frac{\partial}{\partial \mathbf{\Theta}_k} J(\mathbf{\Theta}_k, \mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)}) \quad . \tag{3.9}$$

In equation (3.9), $\mathbf{x}^{(1:N)}$ represents all the input samples, $\mathbf{y}^{(1:N)}$ all corresponding desired output samples, N the number of training samples, η the learning rate and k the epoch. Since the computing effort of this update rule scales with the number of training samples, the computation is slow on large datasets. This is not the case for stochastic gradient descent (SGD) which updates weights for all samples $(x^{(n)}, y^{(n)})$ with $n \in [1, N]$ in a random order individually during one epoch according to the update rule

$$\mathbf{\Theta}_{k,n+1} = \mathbf{\Theta}_{k,n} - \eta \frac{\partial}{\partial \mathbf{\Theta}_{k,n}} J(\mathbf{\Theta}_{k,n}, x^{(n)}, y^{(n)}) \quad . \tag{3.10}$$

However, the downside of this approach is that the updates of the parameters are relatively noisy compared to batch gradient descent. Minibatch SGD considers a subset of the dataset in each update of the weights and can therefore be seen as a compromise between the two extremes of considering the whole dataset or only a single sample [29]. Another widely used optimizer is called Adaptive Momentum Estimator (Adam). In contrast to SGD which has a constant learning rate, ADAM computes adaptive learning rates using first and second-order momentum. Further details are given in [30].

3.3. Convolutional neural networks

Dense neural networks have several disadvantages. One of the major disadvantages comes with the processing of large input data such as images leading to high computational complexity, a large network, and the associated high memory requirement. Besides these drawbacks, a large dense neural network tends to overfit, making it unsuitable for image processing. Convolutional neural networks (CNN) are applied to tackle these issues [31].

3.3.1. Convolutional Layer

Similar to dense neural networks, CNNs consist of convolutional layers. The advantage of such a layer is that it is able to efficiently extract local features of its input. Considering the first layer $L = 1$, the input could be an image having the width W_1, the height H_1 and a channel depth of $D_1 = 3$ representing the RGB-values of the image. The inputs $\mathbf{x}_{L-1} \in \mathbb{R}^{H_{L-1} \times W_{L-1} \times D_{L-1}}$ of the subsequent layers are often referred to as features.

By passing a feature $\mathbf{x}_{L-1}$ into the L-th layer of the convolutional neural network, the kernel tensor $\mathbf{F}_L \in \mathbb{R}^{K_L \times K_L \times D_{L-1} \times D_L}$ and the bias vector $\mathbf{b}_L \in \mathbb{R}^{D_L \times 1}$ are used to compute the activation tensor [32]

$$A_L^{m,n,o} = \sum_{d=1}^{D_{L-1}} \sum_{i=1}^{K_L} \sum_{j=1}^{K_L} F_L^{i,j,d,o} x_{L-1}^{m+i-1,n+j-1,d} + b_L^o \quad \in \mathbb{R}^{H_L \times W_L \times D_L} \tag{3.11}$$

with the reduced output size dimensions

$$\begin{aligned} 1 \leq m \leq H_L &= H_{L-1} - K_L + 1 \\ 1 \leq n \leq W_L &= W_{L-1} - K_L + 1 \\ 1 \leq o \leq D_L \quad &. \end{aligned} \tag{3.12}$$

Fig. 3.1, in which the kernel moves across the image and performs element-wise multiplications with it, illustrates the convolution with a channel depth of $D_1 = 1$. It is important to note that in the deep learning literature and in this thesis the summation over the product between the kernel and input tensor components described in equation (3.11) is called convolution, although it is in a mathematical sense a cross correlation between the kernel and the input tensor. Similar to dense neural networks, an activation function Φ_L is applied element-wise on the activation tensor to compute the output of the layer x_L.

Since the same kernel tensor, often having a small spatial dimension K_L of 3, 5 or 7 and being applied on all spatial locations of an input feature, the convolutional layer has a correspondingly small number of parameters. Consequently, the parameters require less memory [32].

3.3.2. Modified Convolutions

There are cases in which it is required that the output size matches the input size. This might be important when wanting to apply the activation function for each location with respect to the input [33]. However, the input size only equals the output size for a spatial kernel size of $K_L = 1$. Hence, the borders are often padded with zeros. Other padding strategies

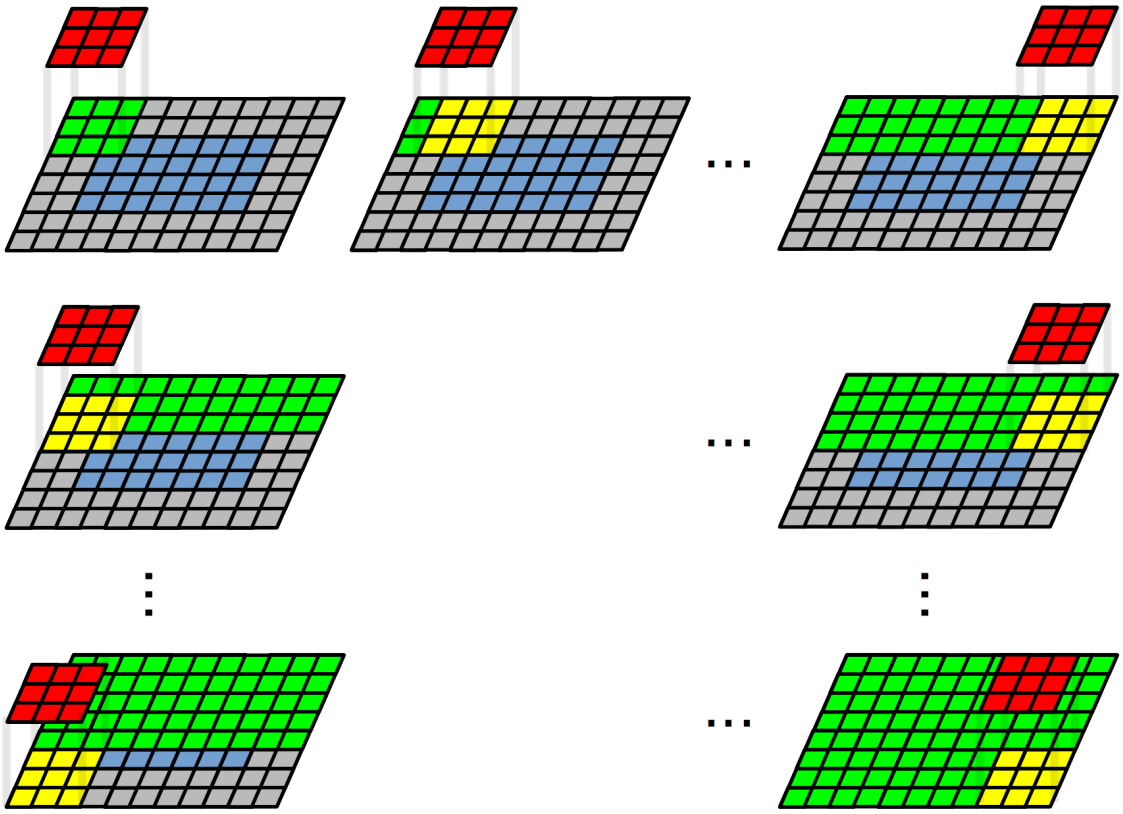


Figure 3.1.: Illustration of standard convolutional operation with a kernel represented in red, an image in blue and padding in grey

such as the reflection of the borders are also possible. Assuming an uneven spatial kernel size K_L, padding the input at each side with $P = \frac{K_L-1}{2}$ zeros and executing the convolution subsequently, preserves the spatial feature size. Fig. 3.2 visualizes the effect of padding along with other modifications of the convolution.

So far, the kernel was moved $S = 1$ positions horizontally and vertically each time after performing element-wise computation. The kernel can also be moved with a constant stride $S \geq 1$. By choosing a larger stride, the spatial dimension of the output can be decreased further, and additionally, the receptive field of the CNN can be increased. The receptive field is the size of the region of the input considered while computing a feature [34]. It should be sufficiently large in order to be able to recognize large objects or capture large displacements of objects. A relatively large receptive field contributes to the high accuracy of CNNs compared to classic methods in image recognition, [32].

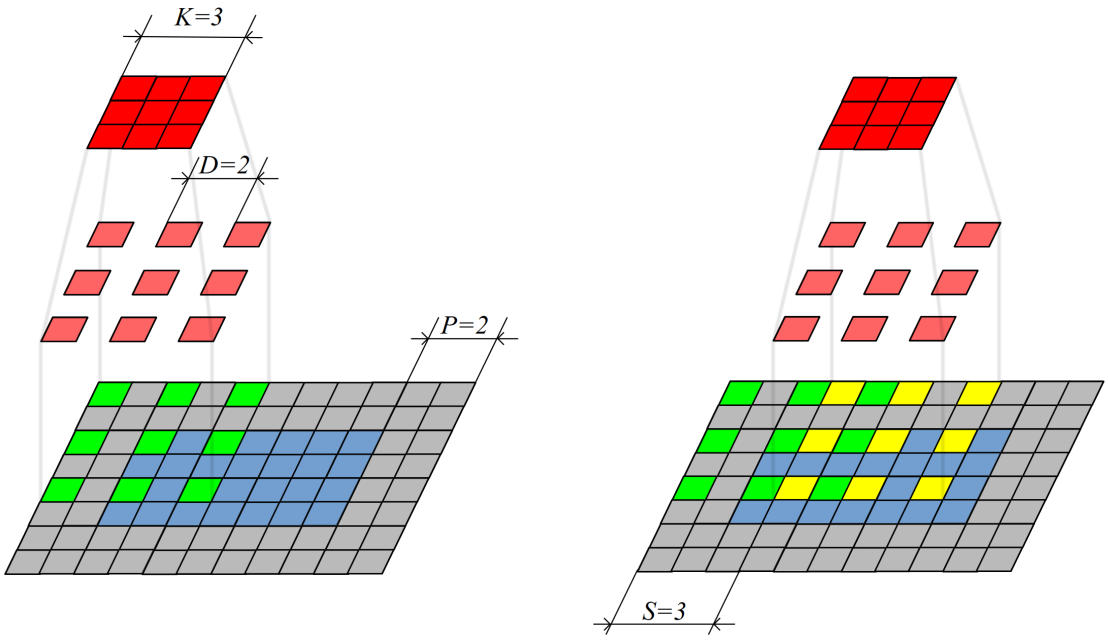
Another concept of convolutions is dilation D. Similar to the stride, the dilation increases the receptive field, except that it downsamples the input instead of the output [35].

Using these three modifications, the resolution of the output can be controlled as the following equation shows

$$W_L = \left\lfloor \frac{W_{L-1} + 2P - (K_L - 1)D - 1}{S} \right\rfloor + 1 \quad . \tag{3.13}$$

The width W_L and height H_L of the output features of layer L are equally affected by the

padding, stride and dilation [36].



(a) Convolution with kernel size $K = 3$, dilation distande $D = 2$, padding $P = 2$

(b) Convolution with stride $S = 3$

Figure 3.2.: Illustration of modified convolutional operation with a kernel represented in red and an image in blue

3.3.3. Deconvolution

The extraction of features using convolutional layers described above usually leads to a reduction of spatial resolution. However, in some computer vision tasks such as semantic segmentation, depth estimation, or scene flow estimation it might be desired to have an output with the same resolution as the input. In order to increase the resolution, the features can be gradually upsampled with deconvolutional layers. This is done by inserting zeros between the input features and then applying a convolutional kernel containing learnable parameters [37].

3.4. Transformer neural network

A Transformer neural network has first been presented in the paper "Attention is all you need" in the domain of natural language processing [38]. Four years later, the concept was transferred and adapted to computer vision tasks, resulting in the Vision Transformer (ViT) [39]. After first introducing the basic concept of the transformer, the ViT is presented.

3.4.1. Basic concept

Time series and sequential data $x(0), x(1), ...x(\tau)$ can be processed in different ways. One approach is to use recurrent neural networks (RNN) which belong to the class of artificial neural networks. They are characterized by cyclic connections. The RNN processes them sequentially by passing their input samples according to their order. Thereby, they generate a sequence of hidden states $h(0), h(1), ...h(\tau)$ each depending on the previous hidden state. The hidden states ,therefore, serve as memory including information about the past input samples. However, when long time series are fed into an RNN, the chain rule is applied frequently during backpropagation through time. This leads to a high number of recursively computed derivatives and thus increases the risk of an exploding or vanishing gradient. This issue can be alleviated by reducing the number of layers, but this has the disadvantage that the module capacity is also reduced. Besides the batch size can be reduced. However, this leads to a noisy gradient. These phenomena make the RNNs hard to train.
Long Short-Term Memory (LSTM) Networks provide a relief by introducing a forget gate, enabling a smoother gradient flow and the preservation of long-term dependencies. However, the disadvantage of sequential processing of the input samples remains slowing down the computation. Secondly, both the RNN and the LSTM neural network assume that the hidden state $h(t)$ solely depends on the previous hidden state $h(t-1)$ and the previous input $x(t-1)$. This assumption, also known as Markov property, leads to a greatly reduced influence of early samples.

Transformers, on the other hand, process the time series as a whole. Fig. 3.3 illustrates the architecture of the original transformer neural network [38]. First, the positional encoding is added to the input embedding. The input embedding can be thought of as learned vectors each representing one input sample. The positional encoding is either fixed or learned and contains information about the order of the input samples.
The sum of the input embedding and positional encoding is then fed to the multi-head attention block depicted in Fig. 3.3 and passed through one linear layer per head. These linear layers create the value vectors $\mathbf{V}$, key vectors $\mathbf{K}$ and query vectors $\mathbf{Q}$. The query vector is comparable to the previous hidden state $h(t-1)$ in an RNN. The keys represent features of all other input samples [40]. Subsequently, the value, key, and query vectors are split according to the number of heads h. The multi-head attention module applies a mechanism called self-attention h times. There are different ways to compute self-attention. In this thesis, however, the focus is on the scaled dot-product attention illustrated in Fig. 3.3. This self-attention mechanism computes the degree of association between the different positions of an input sequence by first computing the dot-product between the query and key vectors. A high entry of the resulting matrix can be interpreted as a stronger relation between the two corresponding input samples. Dividing the entries of the resulting matrix by the square root of the length of the key vectors d before passing them into the softmax function leads to better results for large values of d. A possible explanation for this improvement is that large values of d, lead to a large argument of the softmax pushing the output of the softmax into regions with vanishingly small gradients. Finally, the value vectors are weighted by the output of the softmax and summed up. After executing this procedure h times in parallel, the results of the heads are concatenated and passed through a linear layer. The original transformer neural network uses the multi-head attention block along with residual connections and normalization layers.

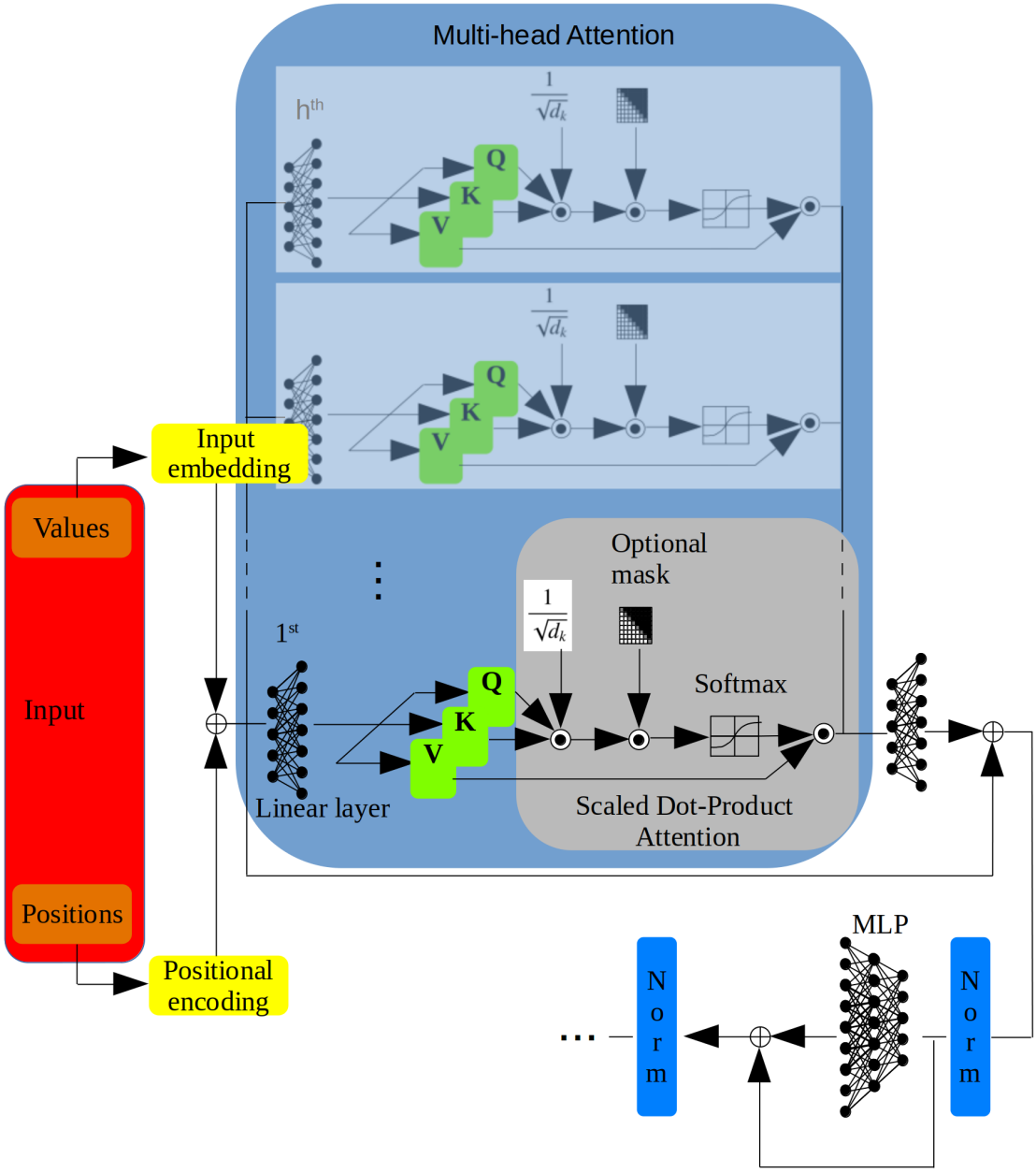


Figure 3.3.: Architecture of transformer neural network

3.4.2. Application of transformer neural networks in computer vision

Being familiar with the concept of transformer neural networks, the application of transformers in computer vision will be discussed. The ViT serves as a basic example. Its structure can be seen in Fig. 3.4. In contrast to the original transformer neural network that received words as inputs, the ViT receives a sequence of image patches. These patches are passed through a convolutional layer and flattened. Thereby, an input embedding is obtained. In order to obtain the positional embedding, each patch is assigned to a single number and passed through a convolutional layer. Similar to the original transformer, the sum of the patch and positional embedding is fed into the encoder [39]. A goal of the creators of the ViT was to design a transformer that is as similar as possible to the original transformer. This leads to some diffi-

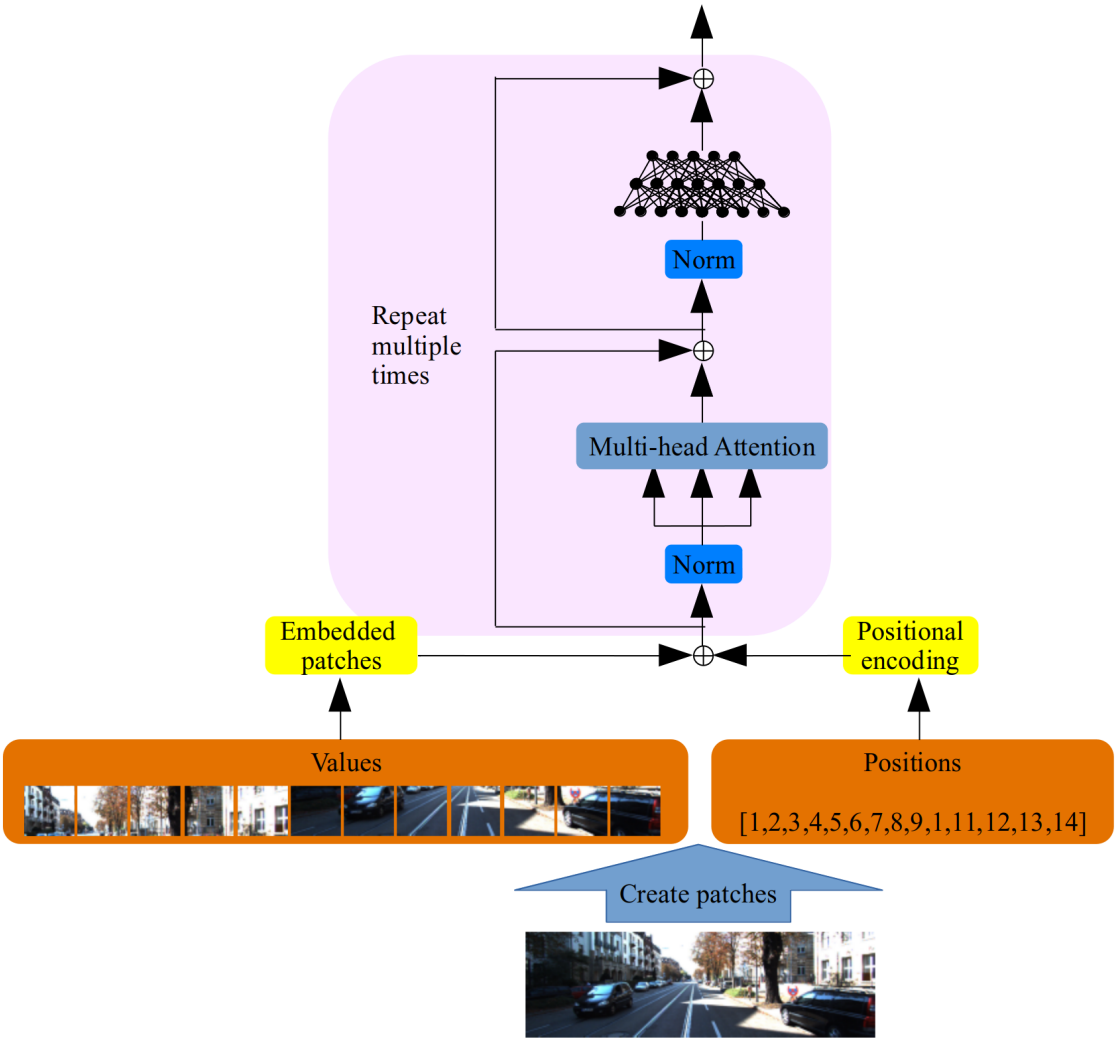
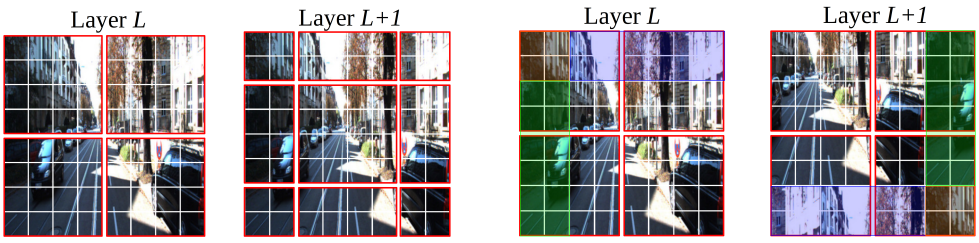


Figure 3.4.: Model architecture of Vision Transformer



(a) Shift of red framed windows between two successive layers L and $L + 1$ produces additional smaller windows

(b) Cyclic shift of window to avoid numerous small windows and thus enhance computational complexity

Figure 3.5.: Shifting of windows

culties because the inputs of natural language processing and computer vision tasks are quite different. In contrast to the number of words in a sentence, objects in images and the images themselves can drastically vary in size and also resolution. Furthermore, the computing effort of the ViT increases quadratically with the image width in the case of a square input image.

Additionally, the computing effort of the transformer scales strongly with the image size.

The designer of the shifted windows (Swin) transformer tried to tackle these issues [41] by enhancing the ViT. The paper presenting the Swin transformer divides the image into windows, and in turn, divides them into patches. The self- attention is not computed over all patches of the images as it was done in the ViT, but only on the patches inside each window. This increases computational efficiency. The application of self-attention across windows is still enabled by shifting the windows by half their size, as illustrated in Fig. 3.5a. This procedure makes the model more powerful. However, shifting the windows leads to an increased number of windows because new smaller windows emerge at the borders of the image. Padding these windows to their original size and masking the padded values afterwards is an obvious solution. However, this approach increases the number of windows especially if the original number of windows was already high. Applying a shifting strategy like in Fig. 3.5b that involves moving patches from the left and top of the image onto the other side, does not change the number of windows and is therefore preferable. Before shifting the windows back to their original positions, masking is applied to avoid big influence between far away patches.

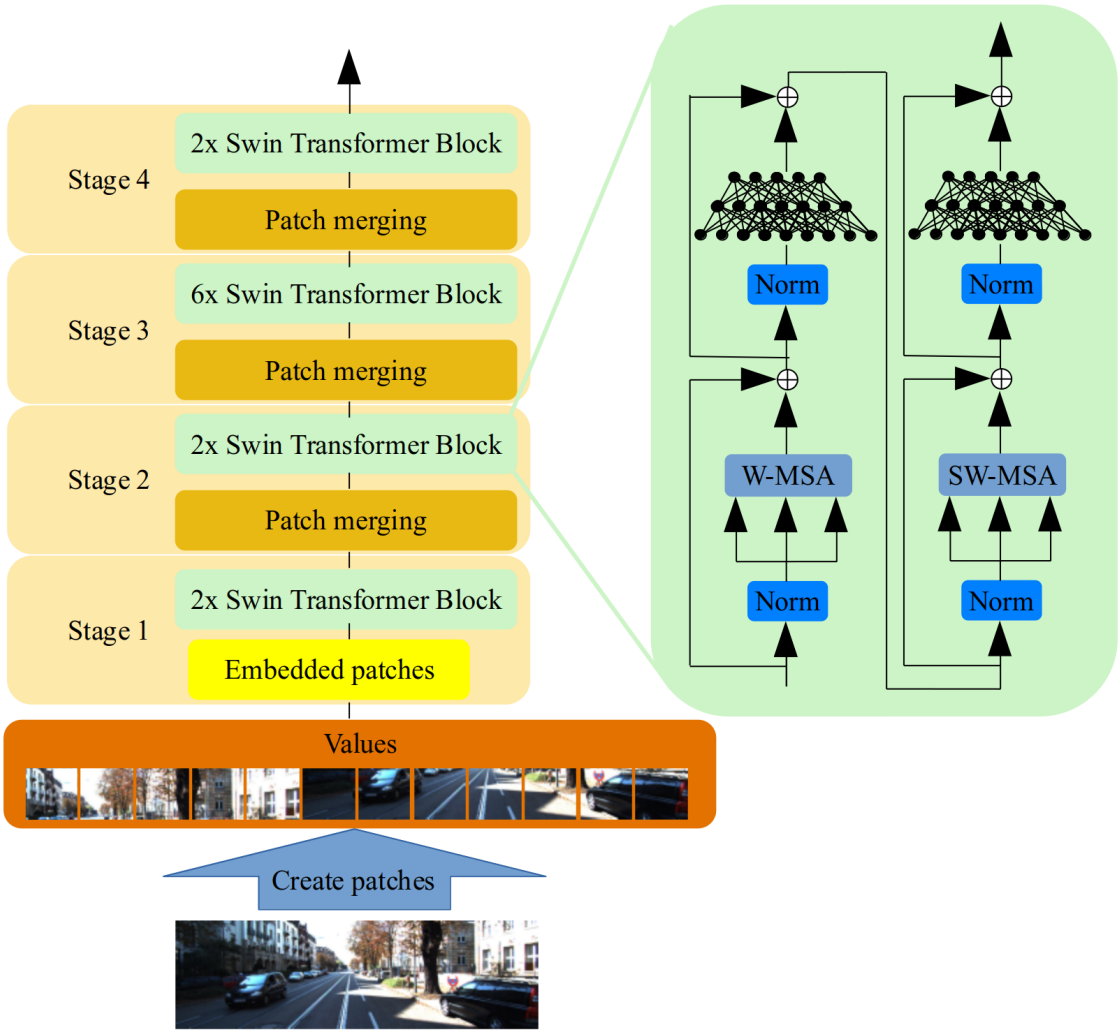


Figure 3.6.: Overview of Swin transformer

Now that the basic mechanisms of the Swin transformer are known, its global architecture shown in Fig. 3.6 is presented. Replacing the positional encoding by a relative position bias B used for the computation of the attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} + \mathbf{B}\right)\mathbf{V} \tag{3.14}$$

leads to improved results on image segmentation. A possible explanation is that the relative position bias induces translational invariance. Consequently, only the patch embeddings enter the Swin transformer block. It is built like the block which is marked in gray on the right side of Fig. 3.4 except that multi-head self-attention (MSA) is performed once on the original window (W-MSA) and then on a shifted window (SW-MSA). These two successive Swin transformer blocks are illustrated on the right side of Fig. 3.6. When passing the data through the different stages, the individual patches of the image are gradually merged as can be seen in Fig. 3.7. This hierarchy allows the Swin transformer to process high level as well as low-level features and makes it resilient to changes in the scaling of the features.

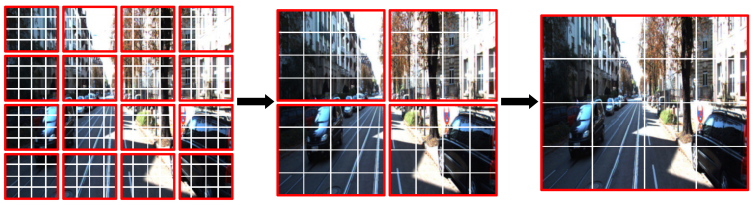


Figure 3.7.: Illustration of successive merging of patches

4. Unsupervised scene flow estimation from monocular camera images

In this work, neural networks are used to estimate scene flow. Neural networks are a subset of methods of machine learning methods that, in turn, can be grouped into supervised and unsupervised learning. In general, supervised learning is a method of machine learning where the machine learning algorithm is presented with a data set for which the target variable is already known. Therefore, in this use case, scene flow estimation is called supervised if ground truth scene flow or ground truth depth is used during training. In this work, no ground truth is used during the training, which is why unsupervised scene flow estimation is involved and why the design of the loss function is particularly challenging.

In the following, the structure of the chapter is briefly summarized and motivated. In section 4.1, scene flow will be introduced in detail along with related quantities such as the optical flow. The training of unsupervised scene flow estimators is often guided by a loss that evaluates how well the scene flow is estimated. Therefore a warped image, which is a reconstructed image based on an image from another perspective or time using one or more estimated quantities such as depth, disparity, optical flow, or scene flow, is compared with a target image. Warping (refer to section 4.2) is done such that the warped image has the same perspective and time as an, other already known image called target image. However, before a scene flow is available to the loss for warping, it must be estimated based on images from a camera. Since the scene flow is extracted out of point clouds and we only have images available as a basis for estimating the scene flow, the relation between the point clouds and the pixels of the images will be presented in section 4.3. The relation is established by using geometric transformations between a pixel of the image and a point of the point cloud. After knowing the mathematical relations between the individual quantities, suitable and commonly used losses will be discussed in section 4.4. Since not all pixels of the warped image should be considered when computing the loss due to occlusions, masking strategies will finally be presented in section 4.5.

4.1. Scene Flow

In the following, the scene flow is defined and different ways of visualizing it are presented in order to make the scene flow intuitively understandable. These visualizations will be used to check the plausibility of the scene flow in subsequent chapters.
Autonomous driving requires precise recording of the environment and its changes over time so that the vehicle can react accordingly. For this purpose, a three-dimensional representation of the environment is generated by sampling points described with three coordinates on the surfaces of the objects. Together these points called scene points form a point cloud. In

order to obtain them, information about the respective depths and distances to the objects is required. The distance can either be measured directly using sensors such as radar or lidar, or the depth can be reconstructed from one or more camera images. With this approach, point clouds can be generated at specific time intervals.
As described above temporal change in the environment described by point clouds is of interest. Consequently, the scene flow was introduced by Vedula *et al.* in 1999 [7] that describes this relationship. He called this variable scene flow and described it a three-dimensional motion field of surface points in the world. The scene flow shows the three-dimensional displacement vector of each surface point between two frames [42] and is closely related to the optical flow. In contrast to the scene flow, the optical flow is two-dimensional, and represents the pixel motion between two subsequent frames [43]. In other words, the optical flow describes the displacement of a pixel between two images that were taken at different points in time whereby the two considered pixels refer to the same scene point. In fact, it is the projection of the scene flow on the image plane of the camera [44]. The advantage of the scene flow over the optical flow is that it contains more information.

Now, we aim to visualize the scene flow. For this, images from the KITTI dataset are used, which is described in more detail in chapter 5. Briefly summarized, this dataset includes, among other data, images from two color cameras mounted on the roof of a moving vehicle. Fig. 4.1 illustrates the position of the two cameras which the KITTI dataset was created with. In Fig. 4.2, two consecutive camera images $I_{t-1,l}$ and $I_{t,l}$ from the KITTI dataset are shown

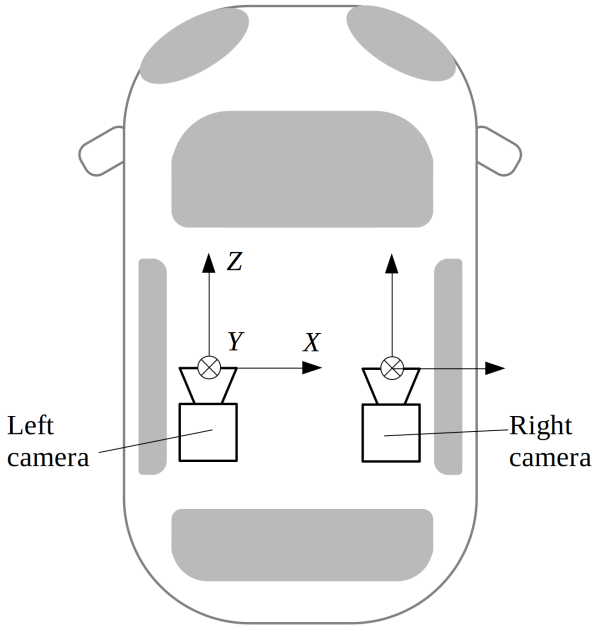


Figure 4.1.: Location of the color cameras used for the generation of the KITTI dataset including the orientation of their coordinate systems

at the top left and bottom left. The index t always describes the time at which the picture was taken and the index l that the picture was taken from a camera located on the left side of the car. These images show that the ego-vehicle, on which the camera is mounted, is slowly driving up to a red traffic light in front of an intersection. All the vehicles shown in the images are in motion. The center-left image of Fig. 4.2 shows the magnitude of an estimated

forward scene flow. The forward scene flow describes the displacement of the point cloud $\mathbf{M}_{t-1}$ at time $t-1$ to the point cloud $\mathbf{M}_t$ of time t. Therefore, the forward scene flow is marked with the index fw.
The magnitude of the scene flow clearly indicates that the car on the right, going straight ahead, has the highest speed. The white car in the middle is about to turn and is therefore already a little slower. The car on the left, however, hardly moves between the successive frames because it is currently driving in a left curve. Considering the magnitude of the scene flow, the left car hardly differs from its surroundings. The consideration of the x, y, and z components of the scene flow, which refer to the coordinate system of Fig. 4.1, in the right part of Fig. 4.2 allows deeper insights. Now the movement of the left vehicle to the left can

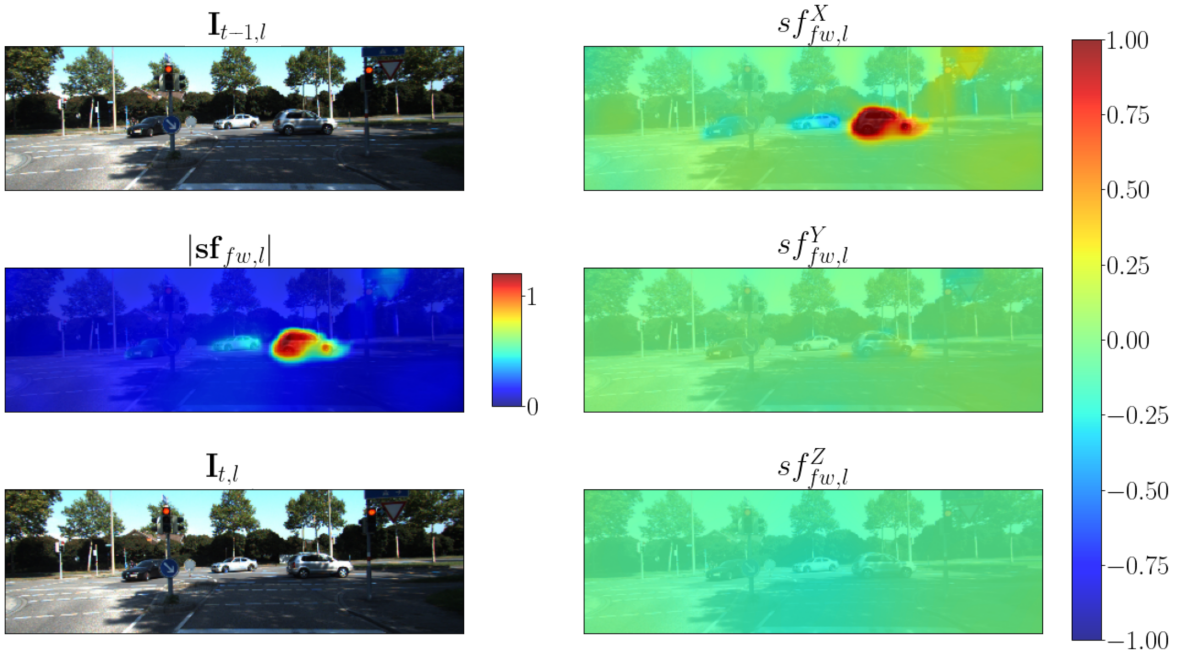


Figure 4.2.: Visualization of two subsequent images $\mathbf{I}_{t-1,l}$ and $\mathbf{I}_{t,l}$, the corresponding estimated magnitude of the forward scene flow $|\mathbf{sf}_{fw}|$ and its components sf^X_{fw}, sf^Y_{fw} and sf^Z_{fw}

be seen by looking at $sf^X_{fw,l}$. As expected, no road user moves vertically. The illustration of the scene flow component $sf^Z_{fw,l}$ is slightly bluish and mainly reflects the forward movement of the ego-vehicle. So far the representation of the scene flow is similar to that of the optical flow, where a single image is often sufficient and where the directions and magnitudes of the optical flow are often coded with colors.

In contrast to the optical flow, the estimated scene flow $\mathbf{sf}_{fw}$ can be visualized in one image together with the point cloud at time $t-1$ or t. The point cloud is usually already available as an intermediate result and does not have to be additionally computed. Since the individual points of the point cloud at time t come from the pixels of the images $I_{t-1,l}$ and $I_{t,l}$, they can be colored with their RGB values as can be seen in Fig. 4.3. How the scene points can be reconstructed from the RGB values of the pixels will be explained later in the 4.3 chapter. Fig. 4.3 shows the point cloud at time t with the subset of corresponding forward scene flows $\mathbf{sf}_{fw}$ depicted as pink lines. The majority of the pink lines start at the scene points and point towards the observer because the observer approaches the intersection and is, therefore,

closer to the scene points at time t. In addition, the relative direction of movement and relative speed of the vehicles can be easily deduced from the orientation and length of the pink lines. In this work, the scene flow is estimated with images from only one camera. Since the scene flow describes the relation between point clouds, depth is also estimated. Both the scene flow and the depth are used for transformations of the camera images between different perspectives and points in time. The interim results of these transformations are point clouds which, together with the scene flow as plotted in Fig. 4.3, provide information about the quality of the scene flow and depth estimations due to their plausibility. More examples for scene flows along with their visualization, interpretation, and evaluation are provided in section B.

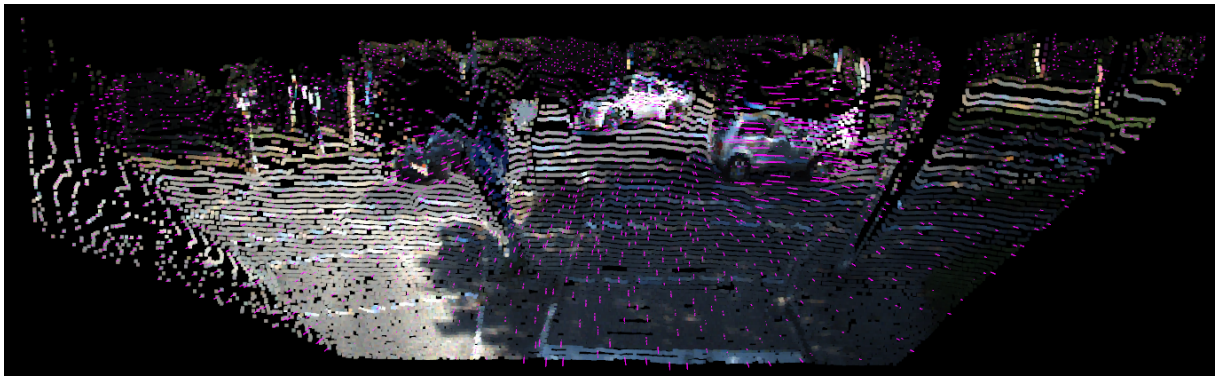


Figure 4.3.: Generated colorized point cloud $\mathbf{M}_{t,l}$ and forward scene flow $\mathbf{sf}_{fw,l}$ shown by pink lines

4.2. Warping

When the scene flow is estimated using a neural network, a loss is needed to guide the network during training and to evaluate the accuracy of the estimated scene flow. In general, this loss essentially compares a reconstructed quantity such as an image or a point cloud, created using an estimated quantity such as a scene flow, optical flow, or disparity, with a corresponding target quantity which is known during training. The disparity d refers to the displacement between a conjugate pair which consists of two image points from different camera images that result from the projection of the same scene point [45]. Its role in estimating scene flow will be discussed in section 4.4 and its relation to the depth in section 4.3. The comparison described above is used in the loss to evaluate the quality of the reconstruction and thus also of the estimated quantity. These reconstructions are carried out by warping an image or a point cloud. Warping essentially means shifting, for example, an image or a point cloud with, for instance, a scene flow, an optical flow, or a disparity, and is discussed in detail in the following.

First, warping an image from the perspective of the right camera r to the perspective of the left one l with time index t and pixel coordinates i and j is considered. From now on, the reconstructed quantities are always denoted with a tilde. The reconstructed left image

$$\tilde{I}_{t,l}(i,j) = I_{t,r}(i - d(i,j), j) \tag{4.1}$$

equals the right image which is shifted by the disparity. The reconstruction of the right image based on the left one works analogously. Similarly, the future frames

$$\tilde{I}_{t,v}(\mathbf{p}) = I_{t-1,v}(\mathbf{p} - \mathbf{f}_{fw}(\mathbf{p})), \quad v \in \{l, r\} \tag{4.2}$$

using the abbreviation $(i, j) = \mathbf{p}$ and the forward optical flow $\mathbf{f}_{fw}$, which shows the displacement of pixels forward in time, are reconstructed. Analogously, $\tilde{I}_{t-1,v}$ can be reconstructed using the backward optical flow $\mathbf{f}_{bw}$ instead of $\mathbf{f}_{fw}$ and by substituting the minus sign with a plus sign. The reconstruction of future or past point clouds with the scene flow $\mathbf{sf} = \begin{pmatrix} sf^X & sf^Y & sf^Z \end{pmatrix}^T$ is similar and is shown exemplary in the following equation

$$\tilde{\mathbf{M}}_{t,v} = \mathbf{M}_{t-1,v} + \mathbf{sf}_{fw}, \quad v \in \{l, r\} \quad . \tag{4.3}$$

This equation can be rearranged to $\mathbf{M}_{t-1,v}$. The resulting equation describes the shifting of the point cloud from Fig. 4.3 along the pink lines that represent the scene flow.

Usually, the resulting output pixels do not have integer coordinates which makes resampling of the output pixels necessary. It can be done by assigning the unmodified value of the output pixel to the closest input pixel position. This approach is computationally inexpensive. Big changes in the pixel locations through warping entail the necessity for dropping or duplicating pixel values. For instance, when two warped pixels are located in the same original pixel cell, the one furthest away from its center is dropped. Conversely, the values of warped pixels are duplicated to fill neighbouring input cells in which now a warped pixel is located. These effects lead to blocky artifacts in the resampled image.
Another method of resampling, producing much sharper and far less blurry images, is called cubic convolution. It considers a 4×4 block of surrounding input cells. However, this method involves much more computation than the nearest neighbour method making it relatively slow.
By using bilinear interpolation, a good compromise can be made between the computational effort and the quality of the generated image. Therefore, it is used widely. Bilinear interpolation is illustrated in Fig. 4.4. To deduce its formula, the names of the pixel value of the warped image being adjacent to the original pixel location (i, j) is introduced. We call the neighbouring input pixel location on the

- upper left side $\tilde{I}(\tilde{i}_{left}, \tilde{j}_{upper})$
- upper right side $\tilde{I}(\tilde{i}_{right}, \tilde{j}_{upper})$
- lower left side $\tilde{I}(\tilde{i}_{left}, \tilde{j}_{lower})$
- lower right side $\tilde{I}(\tilde{i}_{right}, \tilde{j}_{lower})$.

The pixel location after warping is referred to as $(\tilde{i}, \tilde{j})$. First, interpolation between the two upper points is executed to obtain the point

$$\tilde{I}(i, \tilde{j}_{upper}) = \frac{i - \tilde{i}_{left}}{\tilde{i}_{right} - \tilde{i}_{left}} \tilde{I}(\tilde{i}_{left}, \tilde{j}_{upper}) + \frac{i - \tilde{i}_{right}}{\tilde{i}_{right} - \tilde{i}_{left}} \tilde{I}(\tilde{i}_{right}, \tilde{j}_{upper}) \tag{4.4}$$

in the middle between $I(\tilde{i}_{left}, \tilde{j}_{upper})$ and $I(\tilde{i}_{right}, \tilde{j}_{upper})$. Analogously, an interpolation is done between the lower points to obtain [46]

$$\tilde{I}(i, \tilde{j}_{lower}) = \frac{i - \tilde{i}_{left}}{\tilde{i}_{right} - \tilde{i}_{left}} \tilde{I}(\tilde{i}_{left}, \tilde{j}_{lower}) + \frac{i - \tilde{i}_{right}}{\tilde{i}_{right} - \tilde{i}_{left}} \tilde{I}(\tilde{i}_{right}, \tilde{j}_{lower}) \quad . \tag{4.5}$$

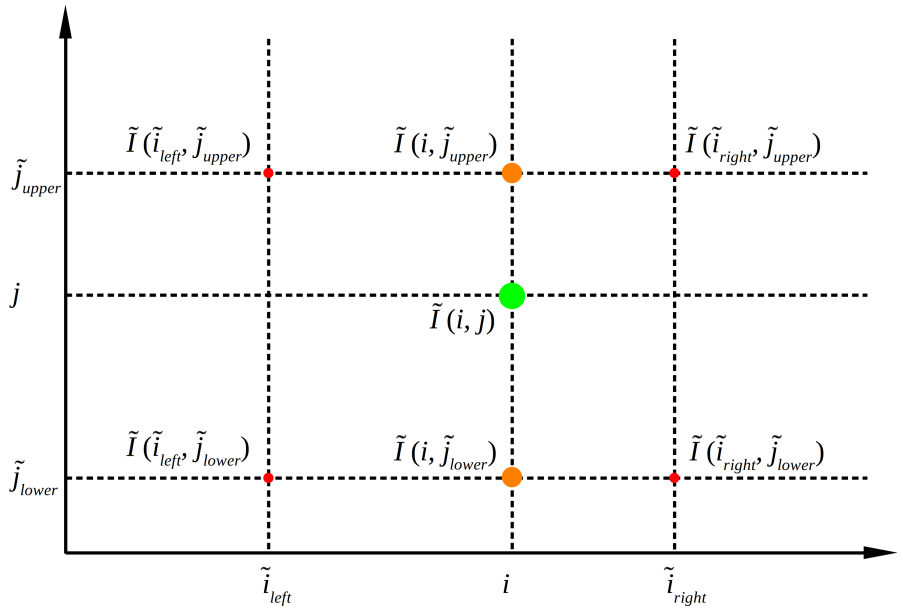


Figure 4.4.: Bilinear interpolation

Finally, the interpolation in j-direction yields

$$\tilde{I}(i,j) = \frac{j - \tilde{j}_{left}}{\tilde{j}_{right} - \tilde{j}_{left}} \tilde{I}(i, \tilde{j}_{upper}) + \frac{j - \tilde{j}_{right}}{\tilde{j}_{right} - \tilde{j}_{left}} \tilde{I}(i, \tilde{j}_{lower}) \quad . \tag{4.6}$$

4.3. Geometric Background

Now it is known how point clouds can be moved with the scene flow. However, in monocular scene flow estimation, only two-dimensional images are initially available. Therefore, the generation of point clouds, for which geometric transformations are used, is relevant. The individual points of the point cloud are referred to as scene points and the individual points of the image are called image points. Before explaining the transformation between the point cloud and the image points in the simplified case of a perspective transformation, the assumptions of this transformation are discussed and its terminology is introduced now. This simplification is based on the assumption of an ideal pinhole camera, meaning that the camera has an infinitely small aperture, which is why effects such as lens distortions can be neglected [47].

The scene point in world coordinates $\mathbf{M}^W = \begin{pmatrix} X^W & Y^W & Z^W & 1 \end{pmatrix}^T$ can be described in the camera coordinate system by multiplying the camera extrinsic matrix

$$\mathbf{K}_{ext} = \left(\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{T} \\ \hline \mathbf{0} & 1 \end{array}\right) \tag{4.7}$$

from the left side whereby $\mathbf{T}$ represents the translational shift and the rotation matrix $\mathbf{R}$ the orientation of the camera coordinate system regarding the world coordinate system [48]. The

projection matrix in turn

$$\mathbf{P}_f = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.8}$$

maps the resulting three-dimensional scene point to a two-dimensional image point in camera coordinates $\mathbf{m}^I = \begin{pmatrix} x & y & 1 \end{pmatrix}^T$ whereby f denotes the focal length of the camera [47]. Subsequently, the resulting image point can be transformed in homogeneous pixel coordinates $\mathbf{m}^P = \begin{pmatrix} i & j & 1 \end{pmatrix}^T$ using the camera intrinsic matrix [48]

$$\mathbf{K}_{int,\bar{f}} = \begin{pmatrix} k_i & k_i \cot(\gamma) & i_0 \\ 0 & k_j / \sin(\gamma) & j_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.9}$$

containing the intrinsic camera parameters

- k_i: column-wise pixel density, describing the number of pixels per unit of length
- k_j: row-wise pixel density
- i_0: x-coordinate of the principal point c which is the origin of the image coordinate system
- j_0: y-coordinate of the principal point
- γ: angle between the coordinate axes of the pixel coordinate system.

By performing all transformations

$$\lambda \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} = \mathbf{K}_{int,\bar{f}} \mathbf{P}_f \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{K}_{int,\bar{f}} \mathbf{P}_f \mathbf{K}_{ext} \begin{pmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{pmatrix} \tag{4.10}$$

one after the other, the relation between the scene point $\mathbf{M}^W = \begin{pmatrix} X^W & Y^W & Z^W & 1 \end{pmatrix}^T$ in world coordinates and the image point is obtained. All transformations that were discussed so far including the relevant coordinate systems are illustrated in Fig. 4.5. The scaling factor $\lambda = Z$ is equal to the depth $\hat{d}$ and results from the ambiguity between depth and scale. For instance, an object being twice as far appears half the size in the image. This is a challenge for scene flow estimation.

When estimating the scene flow, the point cloud is described relative to the camera. Therefore, the multiplication with the extrinsic camera matrix is omitted. Furthermore, we absorb the projection matrix $\mathbf{P}_f$ into the camera intrinsic matrix $\mathbf{K}_{int,\bar{f}}$, receive the modified camera intrinsic matrix $\mathbf{K}_{int} = \mathbf{K}_{int,\bar{f}} \mathbf{P}_f$ by doing so and refer to it as $\mathbf{K}$ for the sake of simplicity from now on. As a result, we obtain the equation

$$\mathbf{M}(\mathbf{p}) = \hat{d}(\mathbf{p}) \mathbf{K}^{-1} \mathbf{m}^P(\mathbf{p}) \tag{4.11}$$

for the computation of the scene points $\mathbf{M}(\mathbf{p})$ belonging to the image point at pixel position $\mathbf{p}$ using the image point $\mathbf{m}^P(\mathbf{p})$ and the depth $\hat{d}(\mathbf{p})$ corresponding to the pixel located at $\mathbf{p} = (i, j)$.

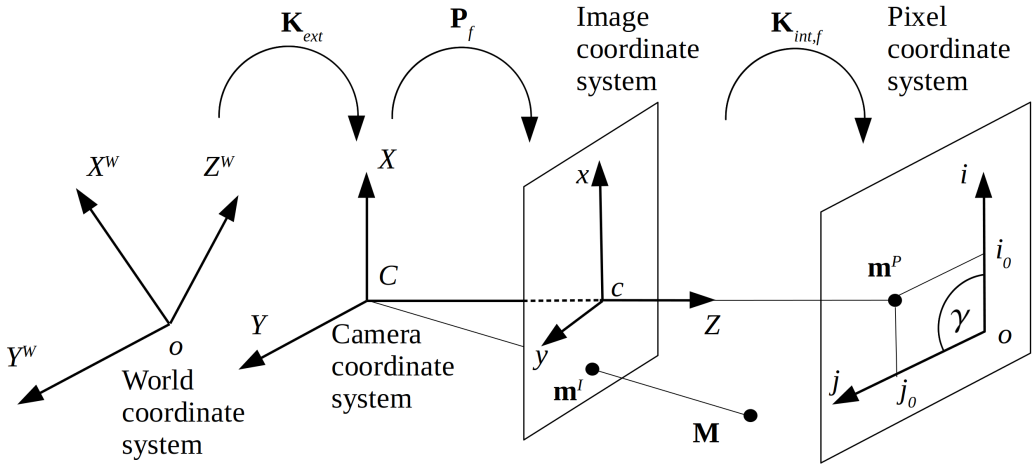


Figure 4.5.: Transformation of scene point $\mathbf{M}(\mathbf{p})$ into image point $\mathbf{m}^P(\mathbf{p})$ and relation between several coordinate systems

However, the depth is unknown. Training often relies on a stereo setup, from which it is possible to estimate the depth. Nevertheless, it is still possible to estimate the depth based on the images of only one camera during inference because the images of the other camera are only used in the loss function. Instead of directly estimating the depth, the disparity can be estimated instead based on stereo images. The disparity d, which refers to the displacement between a conjugated pair, is estimated in an intermediate step. A conjugate pair consists of two image points from different camera images that result from the projection of the same scene point [45]. When the cameras have coplanar image planes and the disparity is known, the depth

$$\hat{d} = \frac{bf}{d} \tag{4.12}$$

can be extracted from it and the baseline distance b [45]. The baseline distance is the distance between the two optical centers of the cameras. This approach was used, for instance, by [49].

4.4. Loss functions

In order to accurately estimate the scene flow, the neural network requires information about the position of a scene point in a given frame in the three-dimensional space. Using only one camera, each individual frame only contains the projection of the scene point on a two-dimensional plane. Even a human looking at one image can not distinguish between the scale and depth of an object without using additional knowledge as classification of the object and subsequent inclusion of empirical values. To overcome the problem, unsupervised monocular scene flow estimators often use stereo images during training to be able to reconstruct the depth during inference [1, 15].

The photometric reconstruction loss ensures that the synthesized image $\tilde{\mathbf{I}}$ resembles the target image $\mathbf{I}$ as closely as possible. This loss can also be used to check whether the estimated optical flow or scene flow is reasonable by reconstructing the previous or subsequent frame

using one of those quantities. The question now is which loss is best suited for evaluating the similarity of the images. The widely used mean square error (MSE)

$$MSE(I(\mathbf{p}), \tilde{I}(\mathbf{p})) = \sum_{\mathbf{p}} (I(\mathbf{p}) - \tilde{I}(\mathbf{p}))^2 \tag{4.13}$$

with the index p describing the location of the pixel can be a first choice. However, it penalizes larger errors while it is more tolerant to small errors. As a consequence, texture-less regions are hardly taken into account. Moreover, the MSE is sensitive towards outliers. To alleviate these effects, the mean absolute error (MAE)

$$MAE(I(\mathbf{p}), \tilde{I}(\mathbf{p})) = \sum_{\mathbf{p}} |I(\mathbf{p}) - \tilde{I}(\mathbf{p})| \tag{4.14}$$

in computer vision is commonly used instead. This is also because of the observation that neural networks trained with an MAE often outperform those trained with an MSE even when using the MSE for evaluation [50]. The MAE is good at preserving luminance and colour since all deviations are weighted equally. However, the remaining downside of both metrics, MAE and MSE, is that they assume that the noise does not depend on local characteristics such as luminance, contrast, and structure. These characteristics are considered in the structural similarity index measure (SSIM) [51]. Generally, it is computed based on two image signals x and y which are aligned to each other and have the same dimensionality. For example, the image signals could be image patches extracted from the same spatial location of the compared images. In our case, $\mathbf{x}$ corresponds to the target image $\mathbf{I}$ and $\mathbf{y}$ corresponds to the synthesized image $\tilde{\mathbf{I}}$. The SSIM is computed as follows

$$SSIM(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2\mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad \in [0, 1] \tag{4.15}$$

whereby

- μ_x, μ_y represent the mean value of x and y respectively
- σ_x, σ_y represent the standard deviations of x and y respectively
- σ_{xy} is the covariance between x and y
- $c_1 = (0.01L)^2$ and $c_2 = (0.03L)^2$ stabilize the measurement of similarity stable for small denominators
- L is the dynamic range of the pixel values and is equal to 255 for gray scale images with 8 bits/pixel.

Unfortunately, the SSIM does not take into account uniform biases. Consequently, the weighted sum of the MAE and the SSIM

$$L_{rec} = \alpha \frac{1 - SSIM(I, \tilde{I})}{2} + (1 - \alpha) \sum_{\mathbf{p}} |I(\mathbf{p}) - \tilde{I}(\mathbf{p})| \tag{4.16}$$

is computed. Equation (4.16) is commonly used as a photometric reconstruction loss [1, 52, 53] to combine the aforementioned advantages. The higher the SSIM, the higher the similarity between the two compared images. Since the loss is minimized, the SSIM has to be maximized. That is why the SSIM appears in the equation with a negative sign.

However, especially in textureless regions having a low image gradient, the correspondence of pixels needed to infer the disparity is difficult to determine using only a photometric reconstruction loss. Hence, an additional constraint is necessary. Namely, it is desirable that neighbouring pixels in regions with low image gradients correspond to similar depths $\hat{d}(\mathbf{p})$. Simultaneously in regions with high image gradients commonly found at object edges, we want to allow jumps in the depth. This requirement is incorporated in a smoothness loss

$$L_{smooth} = \sum_{\mathbf{p}} \left\|\partial_x \hat{d}(\mathbf{p})\right\| e^{-\|\partial_x I(\mathbf{p})\|} + \left\|\partial_y \hat{d}(\mathbf{p})\right\| e^{-\left\|\partial_y I(\mathbf{p})\right\|} \tag{4.17}$$

in which x and y denote the horizontal and vertical direction respectively [1, 52, 53]. The same requirements should also be met by the scene flow. Hence, the smoothness loss can also be directly applied on the scene flow.

The disparity can as already mentioned in section 4.3 also be directly exploited to obtain information about the depth. When the cameras have coplanar image planes, the depths of the images corresponding to the same scene points are identical. According to equation (4.12), the corresponding disparities of the left d^l and right camera d^r are consequently also identical. This requirement can be found in the left-right disparity consistency loss

$$L_{lr} = \sum_{i,j} |d^l_{i,j} - d^r_{i,j+d^l_{i,j}}| \tag{4.18}$$

from [49] which ensures similarity between the estimated disparities.

Similar to the photometric loss, the 3D point reconstruction loss

$$L_{pt} = \sum_{\mathbf{p}} \left\|\mathbf{M}(\mathbf{p}) - \tilde{\mathbf{M}}(\mathbf{p})\right\|_2 \tag{4.19}$$

penalizes deviations between a target quantity and the corresponding reconstructed quantity. The main difference is that it considers the point cloud $\mathbf{M}$ instead of the image $\mathbf{I}$. Penalizing the euclidean distance between the target scene points $\mathbf{M}(\mathbf{p})$ and the reconstructed scene points $\tilde{M}(\mathbf{p})$ significantly contributes to an increased accuracy of the disparity and therefore also of the scene flow [1].

If the correspondence of scene points between subsequent point clouds is unknown, the 3D point cloud alignment loss [52] can be employed. It also penalizes the difference between two point clouds. One goal of [52] is to find the real ego-motion $\mathbf{T}_t$ which describes the transformation between two point clouds corresponding to subsequent frames in the following way $\mathbf{M}_{t-1} = \mathbf{T}_t \mathbf{M}_t$. Using an estimated depth, the point cloud $\tilde{\mathbf{M}}_{\mathbf{t-1}}$ is calculated. Subsequently, the optimal transformation matrix is obtained by using the iterative closest point method. This algorithm basically outputs the optimal transformation between the point cloud of the current and the previous time step

$$\mathbf{T}'_{opt,t} = \underset{\mathbf{T}'_t}{\operatorname{argmax}} \frac{1}{2} \sum_{\mathbf{p}} \left\|\mathbf{T}'_t \tilde{\mathbf{M}}_{t-1}(\mathbf{p}) - \mathbf{M}_{t-1}(\mathbf{p})\right\|^2 \quad . \tag{4.20}$$

Since we want the reconstructed point cloud to match the original, the transformation matrix should be the unit matrix and the residual

$$\mathbf{r}'_t = \tilde{\mathbf{M}}_{t-1}(\mathbf{p}) - {\mathbf{T}'_{opt,t}}^{-1} \mathbf{M}_{t-1}(\mathbf{p}) \tag{4.21}$$

should be equal to zero. This goal is expressed in the 3D point cloud alignment loss

$$L_{3d} = \left\| \mathbf{T}'_{opt,t} - \mathbf{I} \right\|_1 + \|\mathbf{r}_t\|_1 \quad . \tag{4.22}$$

The drawback of this method, however, is that the influence between the depth error and the ego-motion error is not taken into account.

The Chamfer loss

$$L_c(\mathbf{M}, \tilde{\mathbf{M}}) = \min_{\tilde{\mathbf{p}}} \sum_{\mathbf{p}} \left\| \mathbf{M}(\mathbf{p}) - \tilde{\mathbf{M}}(\tilde{\mathbf{p}}) \right\|_2^2 + \min_{\mathbf{p}} \sum_{\tilde{\mathbf{p}}} \left\| \mathbf{M}(\mathbf{p}) - \tilde{\mathbf{M}}(\tilde{\mathbf{p}}) \right\|_2^2 \tag{4.23}$$

is an alternative that penalizes the distance between the closest points of two subsequent point clouds. It was employed by [54]. Additionally, [54] also applies Laplacian regularization

$$L_L = \sum_{\mathbf{p}} \left\| \boldsymbol{\delta}(\mathbf{M}(\mathbf{p})) - \boldsymbol{\delta}(\tilde{\mathbf{M}}(\mathbf{p})) \right\|_2^2 \tag{4.24}$$

to ensure a similar local shape characteristic. The Laplacian coordinate vector

$$\boldsymbol{\delta}(\mathbf{M}(\mathbf{p})) = \frac{1}{|N(\mathbf{p})|} \sum_{\mathbf{M}(\mathbf{p}') \in N(\mathbf{M}(\mathbf{p}))} (\boldsymbol{\delta}(\mathbf{M}(\mathbf{p}')) - \boldsymbol{\delta}(\mathbf{M}(\mathbf{p}))) \tag{4.25}$$

corresponds to the sum of differences between a scene point and its neighbouring scene points $N(\mathbf{M}(\mathbf{p}))$ and characterizes the local shape of the surface.

Furthermore, other deep learning frameworks like generative adversarial networks (GANs) can also be utilized such as in the paper with the title "GANVO: Unsupervised Deep Monocular Visual Odometry and Depth Estimation with Generative Adversarial Networks" [55]. In this paper a generator G reconstructs a frame using an estimated depth and ego-motion given a current, past and subsequent frame $\{\mathbf{I}_{t-2}, \mathbf{I}_{t-1}, \mathbf{I}_t\}$. Subsequently, a discriminator D tries to distinguish the reconstructed frame from the real image. The authors applied a generator loss consisting of MAEs and a discriminator loss

$$L_d = \mathbb{E}_{\mathbf{I_{t-1}} \sim p(\mathbf{I}_{t-1})} \left[\log(D(\mathbf{I}_{t-1})) \right] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left[\log(1 - D(G(\mathbf{z})) \right] \tag{4.26}$$

whereby z represents a latent variable and $p()$ a probability distribution function.

Finally, most scene flow estimators are not GANs and, in terms of architecture, they only have one encoder and one decoder. Additionally, the majority of scene flow estimators use a photometric reconstruction loss and a smoothness loss along with some modifications [1, 15, 56].

4.5. Masking strategies

So far, all pixels were considered for the computation of the losses. However, it is reasonable to neglect or completely ignore outliers resulting from mismatched pixel values. Mismatching can occur due to occlusions or non-lambertian surfaces which do not reflect the light energy in all directions such as a mirror or a window. In a static scene, occlusion can occur simply by taking images from different perspectives. When the images are not only taken

from different perspectives, but also at different times in a dynamic scene and with existing ego-motion, the issue of occlusion exacerbates. Additionally, a large area of the image moves out of the field of view due to ego-motion. That increases the need for a mask. This is not an easy task because as stated above, knowing the occluded areas is important for scene flow estimation. However, occlusion is in turn a consequence of the movement and thus of the scene flow leading to a chicken-and-egg problem [57].

A common strategy to tackle this issue is to consider the scene flow as correct. This was done in [1] where different occlusion masks for the disparity and the scene flow $\mathbf{o}_{sf}$ are applied. In order to compute the left occlusion mask $\mathbf{o}_{sf,l}$, the right disparity is warped forward according to [58] resulting in a disocclusion mask which is the inverse of the occlusion mask. Analogously, the right occlusion mask for the scene flow and disparity is calculated.

Another strategy is to deduce the occlusion mask from the consistency of predicted quantities as the optical flow [59]. If pixels between two subsequent frames are mismatched, the optical flow is also erroneous. This makes it more likely that the cycle consistency between the forward and backward optical flow is violated meaning that the forward optical flow is not equal to the backward optical flow anymore. This indication of an occlusion can be exploited to compute the pixel-wise occlusion in forward direction

$$\mathbf{o}_{fw}(\mathbf{p}) = \begin{cases} 0, & \text{if } \left|\mathbf{f}_{fw}(\mathbf{p}) + \mathbf{f}_{bw}(\mathbf{p} + \mathbf{f}_{fw}(\mathbf{p}))\right|^2 < \alpha_1 \left|\mathbf{f}_{fw}(\mathbf{p})\right|^2 + \left|\mathbf{f}_{bw}(\mathbf{p} + \mathbf{f}_{fw}(\mathbf{p}))\right|^2 + \alpha_2 \\ 1, & \text{otherwise} \end{cases} \tag{4.27}$$

and analogously in backward direction. The value 1 indicates an occlusion and α_1 and α_2 are two constant values.

Alternatively, [60] matches the features considering a learned occlusion map $\mathbf{o}_r(\mathbf{p})$. The cost for matching features can be determined by computing the correlation between the features of the image $\mathbf{x}$ and the warped, masked features

$$\begin{aligned} \text{corr}(\mathbf{x}_l(\mathbf{p}), \tilde{\mathbf{x}}_l(\mathbf{p})) = \mathbf{x}_l(\mathbf{p})^T \cdot \tilde{\mathbf{x}}_l(\mathbf{p})^T = \mathbf{x}_l(\mathbf{p})^T \cdot \mathbf{x}_{r,masked}(\mathbf{p} + \mathbf{d})^T, \\ \mathbf{d} \in \left\{(d_0, d_1)^T : d_0, d_1 \in [-d_{max}, d_{max}]\right\} \end{aligned} \tag{4.28}$$

with

$$\mathbf{x}_{r,masked}(\mathbf{p}) = \mathbf{x}_r(\mathbf{p}) \cdot \mathbf{o}_r(\mathbf{p}) \quad , \mathbf{o}_r(\mathbf{p}) \in [0, 1] \quad . \tag{4.29}$$

Warping is done using the disparity or the optical flow and the displacement of the pixel values is bounded by d_{max}.

5. Datasets

In this section, the choice of a suitable dataset is briefly discussed and the used dataset is presented in detail. An overview of the different data sets mentioned here can be seen in Fig. 5.1. The datasets Monkaa, FlyingThings3d and Driving datasets all contain synthetically generated stereo videos in two versions, namely the clean pass and the final pass. In contrast to the former pass, the latter includes effects such as depth-of-field blur, motion blur, sunlight glare, and gamma curve manipulation. Additionally, depth maps, object segmentation maps, material segmentation maps as well as intrinsic and extrinsic camera parameters are available for each frame. Disparities and optical flow maps, which can be used as ground truth for scene flow estimation, are derived and publicly available quantities.

The dataset Monkaa originates from an animated short film and videos displaying fictional characters in a fictional world. Unlike the Monkaa dataset, FlyingThings3D contains nonfictional three-dimensional objects flying along random trajectories. With a similar artificiality, the Driving dataset shows videos of traffic and the surrounding landscape of a moving car from the perspective of the driver [61]. However, the disadvantage of these datasets is that they are synthetic and thus do not accurately mimic reality.

In contrast to the previous datasets, the KITTI Raw Dataset contains real-world videos. It was particularly created to promote the development of computer vision and robotic algorithms targeted to autonomous driving [62]. Consequently, it is widely used for computer vision algorithms. Due to these reasons, training and testing are done using the KITTI Raw Dataset in this thesis. The dataset contains

- synchronized and rectified colour stereo images with an image size of about 1230×370 which were taken by camera 2 and 3
- greyscale stereo images with similar characteristics taken by camera 0 and 1
- 3D GPS/IMU data
- Calibration data of the Camera, the Camera to GPS/IMU, and the camera to the Velodyne sensor
- 100,000 3D points per frame created using a Velodyne sensor.

Each of the four cameras generated 47940 images. In this thesis, only the stereo colour images and the calibration data are used. Another dataset from KITTI especially designed for scene flow estimation is called KITTI Scene Flow 2015. It is widely used as a benchmark and is divided into a training and testing folder, each containing 200 scenes. Each scene is represented by four colour images representing subsequent frames of each, the left and right camera. Additionally, the training folder contains ground truth namely, disparity maps for two subsequent frames, and the corresponding optical flows. Furthermore, an occlusion map is given for each disparity and optical flow map. The exact creation of these occlusion maps is explained in details in [63]. Since only the training folder of the KITTI Scene Flow 2015

dataset contains ground truth for the disparity and optical flow, it is used as a second test set in this thesis. To the best of my knowledge, currently, there exists no publicly available real-world dataset containing scene flow ground truth. Therefore, the accuracy of the estimated scene flow has to be derived based on the ground truth depth and disparity as described in section 6.1. Only those scene flow values are taken into account for this evaluation that were not masked out in the KITTI Scene Flow 2015 dataset either in the disparity in frame $t - 1$, in frame t or in the optical flow.

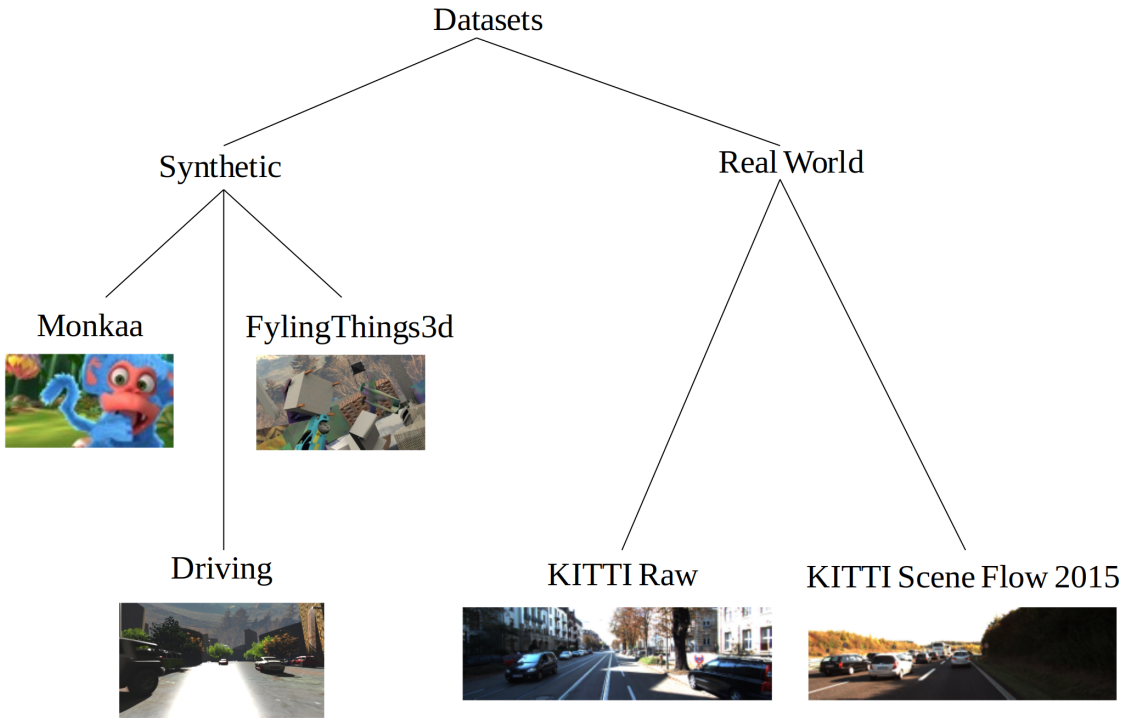


Figure 5.1.: Overview of different datasets

6. Methodology, results and their interpretation

In this chapter, we first introduce an evaluation metric in section 6.1 that will be applied to the test set KITTI Scene Flow 2015. A baseline for the scene flow estimation will then be introduced in section 6.2. Based on this, an improved version of the scene flow estimator will be presented. Originally, the two input frames were processed independently by a single encoder one after the other. By processing the concatenated two input frames together in the encoder, temporal relations between the input frames can already be learned. Motivated by the fact that the Swin transformer, can efficiently process long time series, we leverage the advantages of the powerful transformer architecture and adapt it into our baseline. We believe that processing multiple past frames is a key step to make more accurate predictions of the future scene flow. Therefore, the inclusion of the Swin transformer serves as a better basis for an additional prediction of the scene flow prediction besides its estimation. The Swin transformer and the associated architecture will be modified until results comparable to the baseline are achieved. Finally, the resulting scene flow estimator will be modified in such a way that it can also be used for scene flow prediction and it is then further optimized.

6.1. Evaluation metric for testing

Since scene flow ground truth is not available in the KITTI Scene Flow 2015 dataset, another method is necessary to evaluate the estimated scene flow during testing. The following method evaluation is adopted from [63] and was used by all methods listed in the ranking of scene flow estimators using the KITTI Scene Flow 2015 dataset [12].

The evaluation of the scene flow is done by evaluating the disparity of the first and second input frame and the optical flow between those frames. These evaluations are equivalent since the scene flow can be derived from the aforementioned quantities as visualized exemplary in Fig. 6.1. With equation (4.11) the scene point of the first frame $\mathbf{M}_{t-1}(\mathbf{p})$ can be computed using the camera intrinsic matrix and corresponding disparity. This is the first step and is therefore marked accordingly in Fig. 6.1. Secondly, the equation

$$\tilde{I}_{t,v}(\mathbf{p}) = I_{t-1,v}(\mathbf{p} - \mathbf{f}_{fw}(\mathbf{p})), \quad v \in \{l, r\} \tag{4.2}$$

outputs the image points of the second frame using the optical flow. Equation (4.11) can be applied again and this time the scene point of the second frame $\mathbf{M}_t(\mathbf{p})$ is obtained. According to equation (4.3), the difference between the two point clouds corresponds to the scene flow. Therefore, the accuracy of the scene flow is a solid approximation being solely directly related to the accuracy of the optical flow and disparities of the subsequent frames given the assumptions made while deriving the referenced equations.

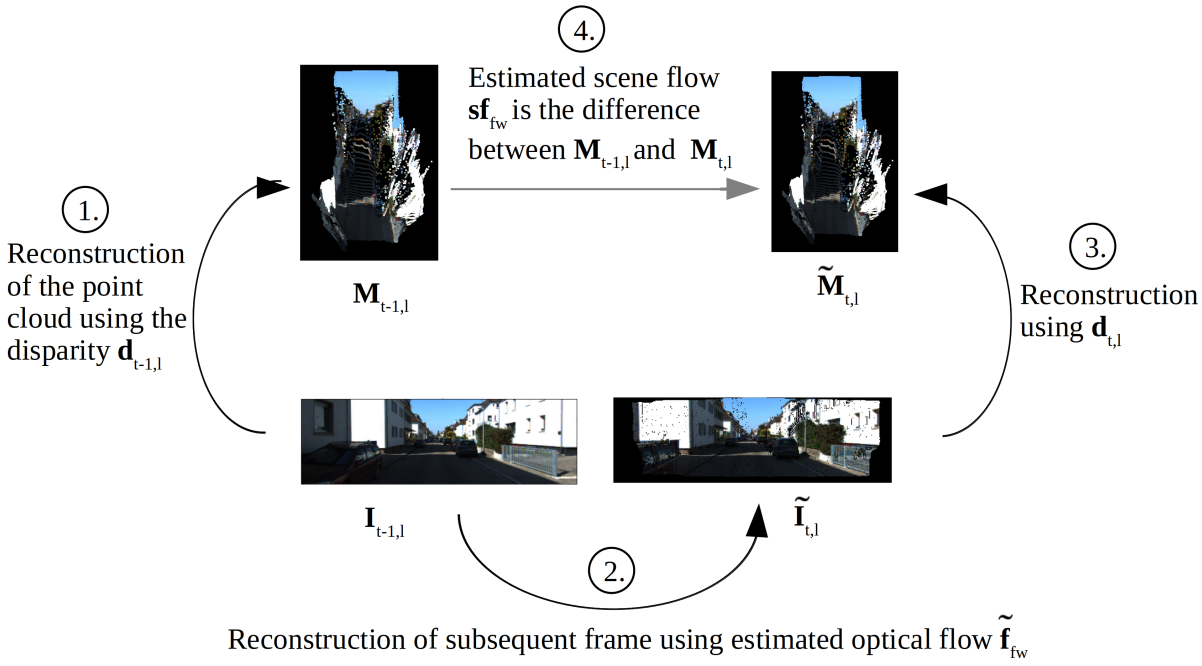


Figure 6.1.: Visualization of connection between scene flow, optical flow and disparities

Based on the accuracies, the estimations can be classified into outliers and non-outliers. From now on the optical flow is not called an outlier if it satisfies both of the two following conditions. The first condition

$$\|\mathbf{f}_{fw,gt}(\mathbf{p}) - \tilde{\mathbf{f}}_{fw}(\mathbf{p})\| < 3 \quad , \tag{6.1}$$

compares the estimated scene flow $\tilde{\mathbf{f}}_{fw}$ with the ground truth scene flow $\mathbf{f}_{fw,gt}$. It states that the estimated optical flow should have an endpoint error which is smaller than 3 pixels. The second equation

$$\frac{\|\mathbf{f}_{fw,gt}(\mathbf{p}) - \tilde{\mathbf{f}}_{fw}(\mathbf{p})\|}{\|\mathbf{f}_{fw,gt}(\mathbf{p})\|} \geq 0.05 \tag{6.2}$$

states that the endpoint error is smaller than 5% [63]. Analogously, the disparities are classified into outliers and no outliers. As explained in the beginning of this section, the evaluation of the optical flow and disparities is equivalent to the evaluation of the scene flow. Consequently, the scene flow is stated as an outlier when either one of the disparities or the optical flow is an outlier. The percentage of scene flow outliers between two subsequent frames is indicated with the metric SF.

However, these metrics do not exactly reflect the quality of the estimated disparities, the estimated scene and optical flow. For example, in the first case all deviations of the disparity can be just below the threshold values defined above and in the second case all slightly above the threshold. In both scenarios, there are no optical flow outliers. Consequently, when using the outlier definitions presented in this section, there are 0% disparity and scene flow outliers in the first scenario and 100% disparity and scene flow outliers in the second scenario. Although the predicted disparities and optical flows were similarly good, they result

in completely different metrics that give a wrong picture of the actual estimation quality. Nevertheless, the percentage of scene flow outliers is used as evaluation metric in this work. The reason for this is that it allows a comparison to other approaches listed in the ranking [12] and that it can be used in addition to another evaluation metric presented later. The two scenarios reflect extreme cases, which are possible but should rarely occur due to the averaging over numerous evaluated images and pixels.

6.2. Baseline

In this thesis, the scene flow estimator of [1] is used to create a baseline. Its whole architecture except the context network is schematically shown in Fig. 6.3 and is discussed in the following. This scene flow estimator uses a stereo pair of subsequent images during training $\{\mathbf{I}_{t-1,l}, \mathbf{I}_{t,l}, \mathbf{I}_{t-1,r}, \mathbf{I}_{t,r}\}$. The images of the right camera are only used in the loss function in order to infer the depth. The architecture and the forward pass of the neural network are now discussed in detail.
During the forward pass, the two subsequent frames of the left camera are passed individually into the same encoder which consists of convolutional layers. All intermediate and last output of these layers, which are referred to as features $\mathbf{x}$ are saved. Together these features form a feature pyramid which is depicted in Fig. 6.2. The argument L in the round brackets refers to the pyramid level. For the sake of simplicity, this argument is omitted if it can be concluded from the context which level is meant. Even if this argument is not present from now on, $\mathbf{x}$ always refers to the features of a certain pyramid level.

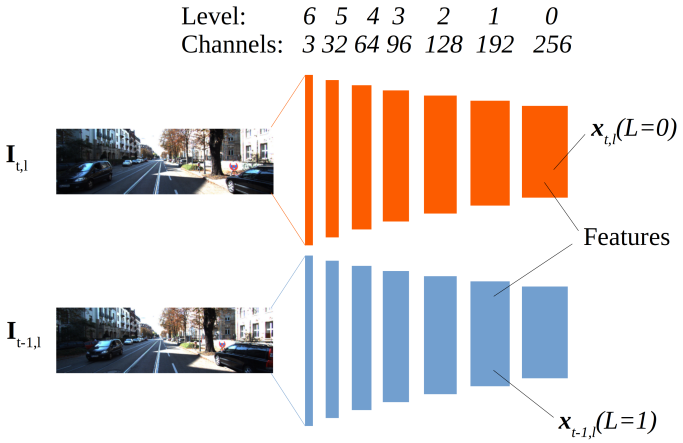


Figure 6.2.: Feature pyramid generated by encoder

Now, the second part of the forward pass is briefly summarized before the details are clarified. The feature pyramid depicted in Fig. 6.2 is looped through whereby a scene flow is estimated at each level. The estimate of the scene flow of the previous level is used to warp the features from one point in time to the other point in time. Then, this warped image is used to estimate a residual scene flow that adjusts the previous level's scene flow. Now, this process is considered in detail.

Therefore, we start at the level with the lowest spatial resolution of features $L = 0$. For $L = 0$ there is still no estimate for the optical flow, which is why no warping is carried out yet as

can be seen in Fig. 6.3. Warping takes place only for $L \geq 1$. Since warping the image is not yet possible, an initialization for the warped image is required. For this initialization the features $\tilde{\mathbf{x}}_{t-1,l}$ are set to be equal to $\mathbf{x}_{t,l}$ for level 0.
Afterwards, it is desirable to estimate in which direction and to which extent the current estimate of the optical flow should be adjusted such that the warped features $\tilde{\mathbf{x}}_{t-1,l}$ match the original features $\mathbf{x}_{t,l}$. For this, the warped features are each shifted by a maximum of s pixels in every possible direction. In the scene flow estimator of [1], $s = 4$ was chosen. A possible reason for this is that $s = 4$ represents a good compromise between the required memory and the computation speed on the one hand and the percentage of scene flow outliers on the other hand. By computing the correlation, it can be determined how bad the matching between these features and the original features is. According to [64], the correlation can be interpreted as a cost for associating the features of one frame with the warped frame. This matching cost is not only computed between the corresponding pixels of the features and the warped features. Additionally, it is also computed between the features at the pixel index p and the warped features with a slightly shifted pixel index. The correlation

$$\text{corr}(L, \mathbf{p}_1, \mathbf{p}_2) = \frac{1}{N}(\mathbf{x}_{t-1,l}(L, \mathbf{p}_1)^T, \tilde{\mathbf{x}}_{t-1,l}(L, \mathbf{p}_2)) \tag{6.3}$$

is computed between the two features whereby p_1 and p_2 denote the pixel positions fulfilling $|\mathbf{p}_1 - \mathbf{p}_2|_\infty \leq 4$. It evaluates how poor the features match the warped features. Additionally, the correlation indicates in which way the estimation of the optical flow should be changed so that the warped features better match the original ones. For instance, a low correlation between a feature at position $\mathbf{p}_1$ and $\mathbf{p}_2 = \mathbf{p}_1 + (3, 0)$ compared to the correlations with other relations between $\mathbf{p}_1$ and $\mathbf{p}_2$, indicates that the optical flow in horizontal direction should have been 3 pixels smaller.
Then, the correlation tensor and unwarped features are stacked in channel dimension and passed into the decoder which consists of convolutional layers and is illustrated with a green rectangle with rounded corners in Fig. 6.3. After calculating an intermediate result $\mathbf{x}_{t-1,l,out}$, the decoder splits into two separate convolutional layers. One layer only determines the forward residual scene flow and the other only the disparity. The disparity and the scene flow are passed into the context network which has shown to lead to a significantly better accuracy of the scene flow as well as of the disparity in the paper of Junhwa Hur and Stefan Roth [1]. The predicted scene flow and disparity are upsampled such that their spatial dimension matches the spatial dimension of the subsequent level.
All operations which were done so far are executed for each level. Additionally, for each level the same operations are applied to the features $\tilde{\mathbf{x}}_{t-1,l}$ and $\tilde{\mathbf{x}}_{t,l}$ with interchanged roles. The meanings of the quantities illustrated in Fig. 6.3 for the case with the interchanged roles are denoted in gray inside of brackets. The backward scene flow is projected to the optical flow, which is used to warp the features of the next level backward in time similar to equation (4.2) except that the warping takes place backward in time instead of forward. The warped features are bilinearly interpolated so that the positions of the pixel values of the warped feature match those of the original one. Again, the correlation between the warped and original features is computed. Now, the scene flow, the disparity, the output of the correlation, and the features $\tilde{\mathbf{x}}_{t-1,l}$ of level 1 are stacked and fed into the decoder. At this point in time, the features, the matching costs for the optical flow, the optical flow itself, and the disparity are known. Consequently, the decoder has all the necessary information to adjust its estimates. The newly estimated residual scene flow is added to the upsampled scene flow of the last

level in order to refine it. This coarse to fine approach allows the network first to give a rough estimate of the scene flow for large image regions. It also allows the network to focus on local differences in the scene flow later. This approach is advantageous since the scene flow generally tends to be more similar to close pixels than to distant pixels.

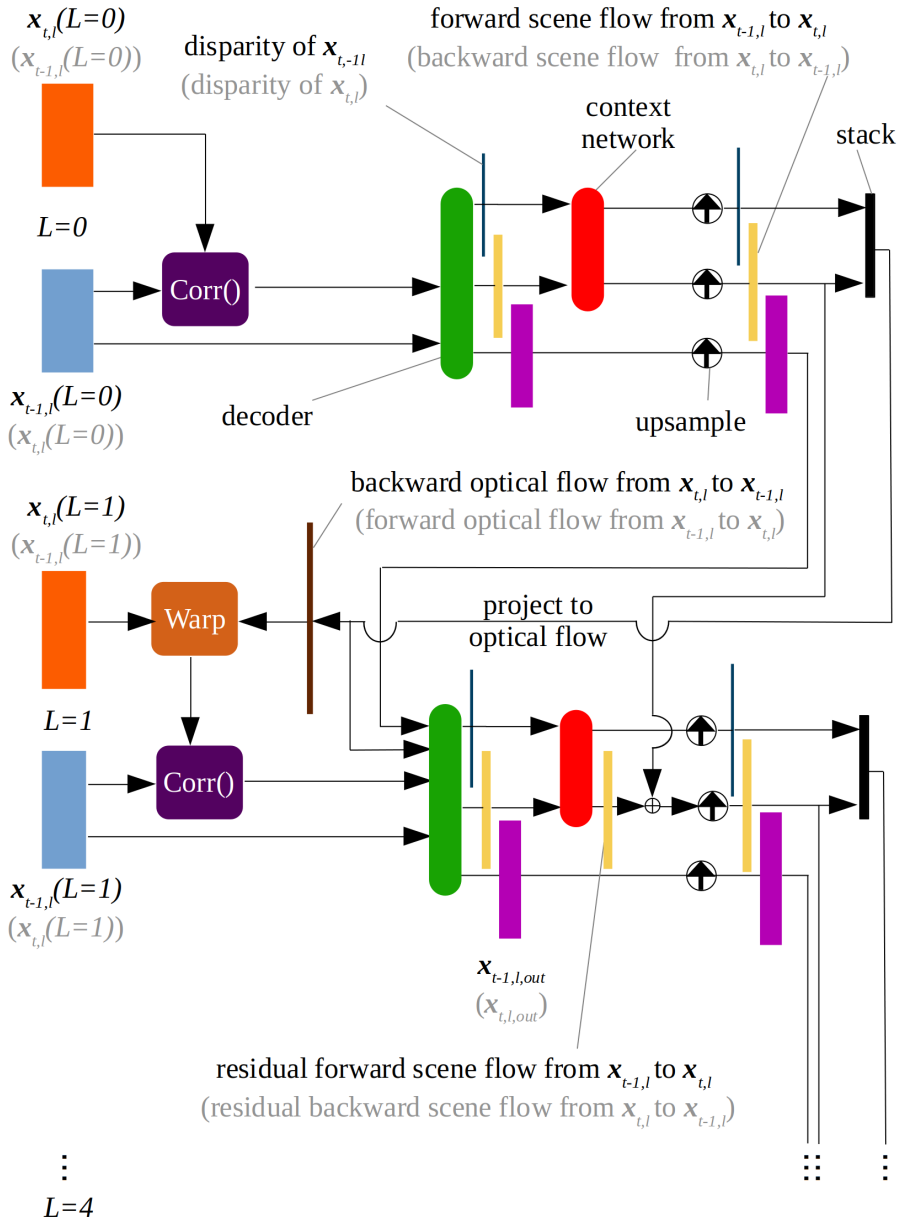


Figure 6.3.: Architecture of unsupervised monocular scene flow estimator

The training is guided by a weighted sum of a disparity L_d and scene flow loss L_{sf}

$$L_{pres} = L_d + \lambda_{sf} L_{sf} \quad . \tag{6.4}$$

Both the disparity and the scene flow loss consist of a photometric reconstruction loss and a smoothness loss which are described in detail in section 4.4. Additionally, a 3D point reconstruction loss is employed as part of the scene flow loss. All losses are masked by taking the inverse of the disocclusion mask as described in 4.5. The test loss is computed in the same way as the training loss.

In the following, we deal with the adjustment of the training setup to obtain a suitable baseline. Since we aim to improve unsupervised monocular scene flow estimation, no semi-supervised fine-tuning was done. Leaving out the fine-tuning increases the number of scene flow outliers from 33.88% to 49.54% according to [1]. In order to be able to run the trainings on the available hardware, simplifications of the training setup as compared to [1] had to be implemented. Firstly, the batch size was reduced from four to two in order to be able to run the trainings on the available graphic cards. The results of the training of this and the following modifications are shown in table 6.1.

modifications	Training		Validation		Test	Runtime in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
batch size 2 instead of 4	/	**5.5**	/	**5.8**	**46.9**	0.18
batch size 2 instead of 4 with modified split between training and test dataset	/	**5.5**	/	**5.8**	47.4	0.18
batch size 2 instead of 4 with modified split between training and test dataset and images resized to half of their size	21.2	5.8	24.7	6.2	55.0	**0.04**

Table 6.1.: Self-produced results of scene flow estimator of the paper [1] without fine-tuning while SF is the abbrevation for precentage of scene flow outliers

Before going into further adjustments to the training setup, the results of the current setup and the way it is evaluated in table 6.1 is discussed. Table 6.1 contains the training, test loss based on a portion of the KITTI raw dataset, the percentage of scene flow outliers of the KITTI Scene Flow 2015 datast, and the runtime of the forward pass during inference for the approach of [1]. The portion of masked pixels, which were not considered for the calculation of the losses, of the first two experiments listed in table 6.1 is not collected during the experiment and could not be generated later due to the timeframe of this work. The experiment in the third row will represent the baseline for the rest of this chapter. The scene flow outliers were only computed for those pixels for which ground truth depth and optical flow are available. This is the case for 80.2% of all pixels. This number is constant for all experiments in this thesis since the available ground truth remains the same.
In the paper [1], the neural network was trained and validated with the KITTI Raw dataset. The test was performed with the KITTI Scene Flow 2015 dataset. The standard approach is to train until the validation loss increases and save the parameters of the network at the epoch with the smallest validation loss. Then, the network based on these saved parameters using test metrics is evaluated. Contrary, to the standard approach, all experiments in this work are trained for 62 epochs for three reasons.
First, the scene flow estimator of Junhwa Hur and Stefan Roth [1] was also trained for 62 epochs suggesting that this value has already been optimized.
Second, the test, as well as the training loss, do not increase anymore in all experiments of this work when approaching 62 epochs so it can be assumed that no overfitting occurs. Since

the test loss does not change significantly at all when approaching 62 epochs, no prospect of significant improvements in test loss or scene flow outliers from longer training is expected. The training-, test loss and scene flow outliers are almost constant for epochs close to 62. This behavior can be seen in Fig. 6.4 which illustrates these losses and the percentage of scene flow outliers over the epochs. These values originate from running the model configuration of the baseline. The observation of constant losses and percentages of scene flow outliers for epochs close to 62 holds true for all modifications of the scene flow estimator presented in this work.
Third, the fixed number of maximum epochs shortens the execution time required for the experiments. Given a constant maximum number of epochs, the duration of the entire training can be approximately calculated if the duration of an epoch is known. Consequently, planning is simplified. The short execution time and facilitated planning allow more experiments to be performed and evaluated in the limited time of this thesis. This is why, from now on the experiments will be evaluated directly after 62 epochs.
In this work, we aim to use the test loss as a second test metric. This leads to a more accurate overall rating of the neural network compared to only using SF as a test metric considering its imperfection as an evaluation metric described in section 6.1. Therefore, the test loss is always considered after 62 epochs like the other test metric.
The runtime of the scene flow estimator of [1] was determined using an Nvidia Geforce GTX 1080 Ti in [1] and is given as $0.09s$. In contrast to [1] all runtimes measured in this work, are determined using an Nvidia Geforce GTX 1080. Furthermore, an Intel Core i7-6900K CPU is used and 64GB RAM are available. This data about the hardware is not specified in [1]. Using our hardware, the model of [1] requires $0.18s$ which doubles the runtime given in the paper. This may be due to the less performant GPU and other differences in the used hardware compared to [1].
In this work, the evaluation of the quality of the scene flow estimation is essentially based on two key performance indicators (KPI), the test loss and the percentage of scene flow outliers. Both KPIs are not totally perfect metrics for the evaluation of the prediction quality. The percentage of scene flow outliers is imperfect since it is based on only one fixed threshold as explained in details in section 6.1. The test loss is also not a perfect evaluation metric because slightly different percentages of the image pixels are masked out due to occlusion resulting from different scene flow estimations in each experiment. As a result, the comparison of the validation losses results in the comparison of the accuracies of scene flow in different areas. Since both KPIs have their weaknesses when it comes to their usage as evaluation metric, both KPIs are considered together during evaluation. Thereby, the percentage of masked area during the computation of the test losses is taken into account. It is assumed that looking at both KPIs together leads to a more accurate evaluation overall and better reflects the real quality of the estimates.
Training with batch size 2 only generated 46.9% scene flow outliers, which is less than the 49.5% specified in the paper generated with batch size 4. The reason for this could be the increased noise occurring at a smaller batch size [65]. More noise can be helpful during optimization since local minima or saddle points can be escaped more easily [66].

Now, the choice of the split between training and test set is discussed. The authors of [1] used stereo camera frames for the test set that were taken temporally between the frames of the training data set. However, because the consecutive frames of the KITTI raw dataset look very similar, training is carried out with images that are similar to images in the test dataset

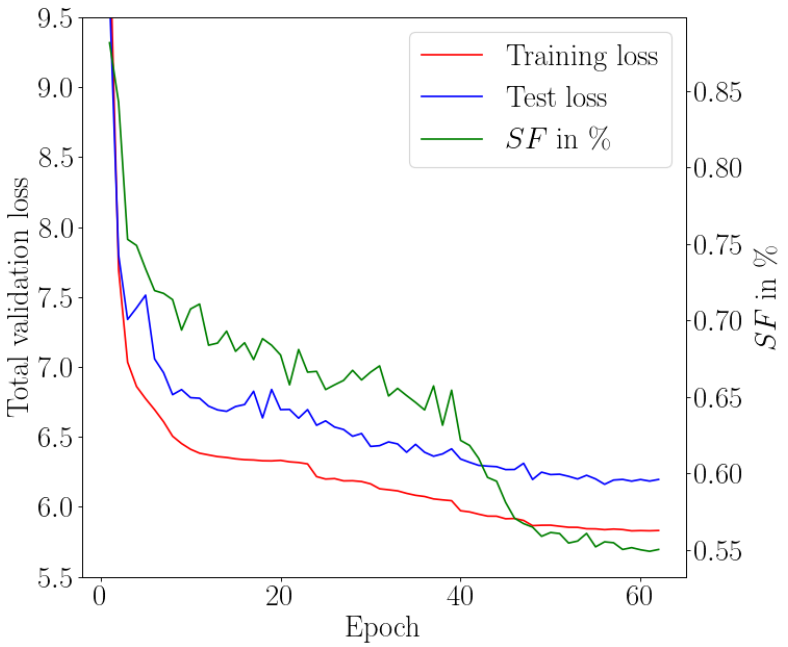


Figure 6.4.: Training-, test loss and percentage of scene flow outliers over epochs

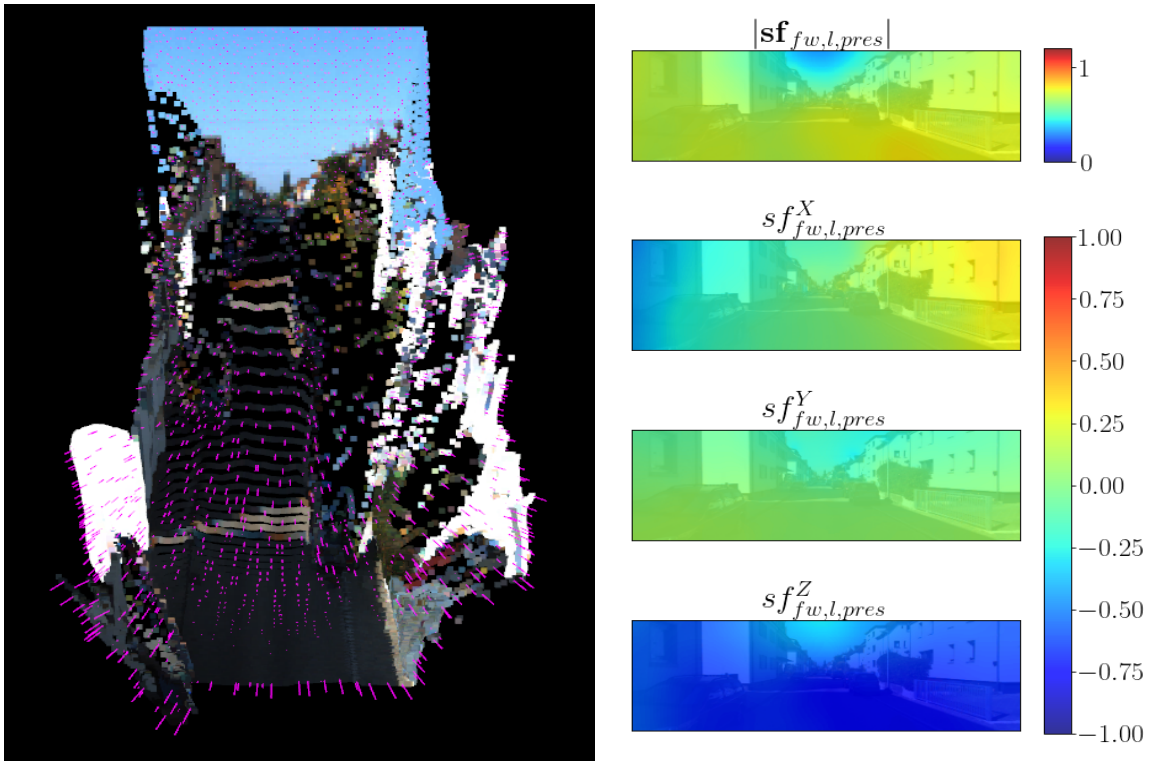
of the KITTI raw dataset. As a consequence, it is assumed that the test loss is not a totally reliable evaluation metric. Additionally, the frames contained in the split of [1] in the test set, which is part of the KITTI raw dataset, are often not directly consecutive meaning that the time between the input frames $\{\mathbf{I}_{t-1,l}, \mathbf{I}_{t,l}\}$ vary greatly. This would make the testing of the scene flow prediction in chapter 7 much more difficult because it requires frames that have a reasonably constant temporal distance from each other. With scene flow prediction, we try to predict the scene flow between $\mathbf{I}_{t,l}$ and $\mathbf{I}_{t+1,l}$. This will be done based on the pictures $\mathbf{I}_{t-1,l}$ and $\mathbf{I}_{t,l}$. If these three frames are all arbitrarily far apart, the scene flow predictor has no way of finding out how far in the future the frame $\mathbf{I}_{t+1,l}$ is apart from frame $\mathbf{I}_{t,l}$ during inference. However, if the time interval between the images in the training and the images in the test dataset is approximately constant, the scene flow predictor has a chance to learn the time interval between the input images and use the same interval to predict the scene flow. Therefore, the split between training and test loss will be changed by using the first 25548 stereo images of the KITTI Raw dataset for the training dataset and the remaining 1543 for the test dataset. The percentage of scene flow outliers increases by the new split. This may be due to the fact that the model is worse overall because the model was trained with data that varies less with the new split compared to the old one. Nevertheless, the new split will be kept since we believe that it increases the relevance of the validation loss.

In [1], all images were changed to the size 832×256 before training and testing. To speed up development and run multiple trainings on one GPU the size was further reduced to 416×128. Another advantage is that it frees up space on the GPU for expansions of the scene flow estimator in this work. As expected, this measure comes at the expense of estimation accuracy reflected by higher losses and a higher percentage of scene flow outliers as can be seen in table 6.1.

The results of all modifications done so far are now checked for plausibility. Therefore, the scene flow and point cloud is visualized as in section 4.1 in Fig. 6.7 for two subsequent input

(a) $\mathbf{I}_{t-1,l}$ (b) $\mathbf{I}_{t,l}$

Figure 6.5.: Subsequent frames from the part of the KITTI raw dataset used for testing with which the point cloud and scene flow of Fig. 6.6 is generated



(a) Generated point cloud $\mathbf{M}_{t,l}$ and forward scene flow $\mathbf{sf}_{fw}$ represented as pink lines

(b) Visualization of magnitude and components of the forward scene flow of the left camera which is depicted with its coordinate system in Fig. 4.1

Figure 6.6.: Scene flow and point cloud of images shown in Fig. 6.5 generated by the scene flow estimator from [1] using the new split between training and test data set and input images with the size of 416×128 for training

images of the test set belonging to the KITTI Raw dataset shown in Fig. 6.5. Despite using batch size two instead of four the modification of the training and the resizing of the images, a point cloud is reconstructed during the testing, which is very similar to the original image from the perspective of the car as illustrated in Fig. 6.6a.
Furthermore, the magnitude and components of the scene flow depicted in Fig. 6.6b are reasonable for the most part. The magnitude for the scene flow should theoretically be zero for the whole sky. However, the sky should not be given too much attention when evaluating the quality of the scene flow estimation. One reason is that the estimated depths are clipped

to 80m. Consequently, the resulting depth is incorrect which leads to erroneous scene flow estimates. Another reason is that the color of the sky is not perfectly uniform. As a result, the gradient in the image is not zero and the smoothness loss allows jumps in it. Additionally, the magnitude of the scene flow should be same for all regions excluding the sky. However, this is not quite the case in the lower right portion of the street, where the magnitude is slightly larger.
The x-component of the scene flow sf_{fw}^X should be close to zero everywhere since the ego-vehicle only drives straight ahead and does not move to the left or right. This is mostly true except for the scene flows on the far left and far right of the image. These areas do not appear in the loss due to occlusion by the ego-vehicle's forward motion and therefore the scene flow is not optimized in these regions. The same problem occurs in a weakened form in the vertical direction. Here the scene flow sf_{fw}^Y should be zero everywhere. However, the values deviate from zero at the top and bottom of the images more than in the rest of the images.
Finally, the motion along the Z-axis, which points to the front, is considered. Due to the forward motion of the ego-vehicle, sf_{fw}^Z should be highly negative except for the sky where $sf_{fw}^Z = 0$ should hold. Although sf_{fw}^Z is smaller for the sky than in the rest of the image, its absolute value is still clearly greater than zero. There are multiple possible reasons for this. For example, the color of the sky is not uniform. As a result, the smoothness loss allows larger jumps in the scene flow. The quantitative evaluation of this experiment is shown in table 6.1. Although these scene flow estimates are not totally correct for all regions, the results are mostly plausible, which is why the scene flow estimator trained with the new split and a batch size of two on images with the size 416×128 is used as a baseline.

6.3. Concatenation of inputs

The first reduction of the scene flow outliers came with the concatenation of the input of the neural network along the feature dimension as illustrated in Fig. 6.7.

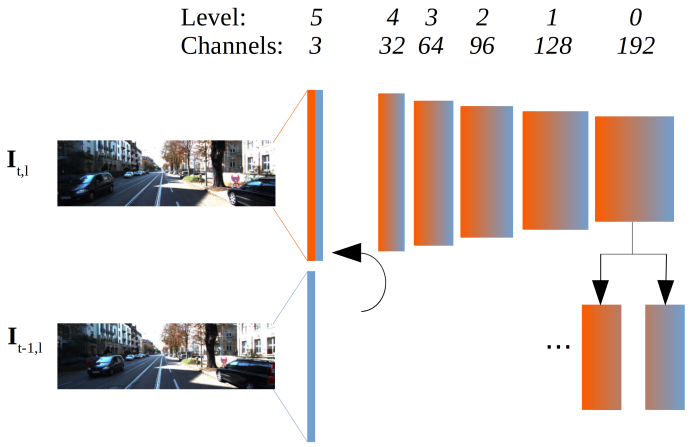


Figure 6.7.: Modified feature pyramid

When looping through the feature pyramid like in Fig. 6.3, the features are split along the feature dimension into two tensors of equal size. After splitting them, the features are treated as features from image $t-1$ and t respectively. The underlying idea of this approach is to

simultaneously learn the features of the images as well as the temporal connections between the images. However, this modification alone leads to poor results as can be seen in table 6.2. The reason for the poor metrics resulting from concatenating the input images lies in the incorrect interpretation of the correlation. The correlation between the warped and original features can be interpreted as matching costs [64]. However, this interpretation is wrong when the two inputs of the correlation each contain information about the warped and original features.

Therefore, the correlation has been omitted. Instead of the correlation, the warped image directly enters the decoder. Table 6.2 shows that concatenating the inputs and leaving out the computation of the correlation decreases the percentage of outliers of the test set significantly from 54.8% to 50.2%. The warping still seems to work, although the inputs to the warping operation do not separately receive the information from features of $t-1$ and t. A possible explanation for this is that the features which are interpreted as features of time $t-1$ contain information about features of time t that cannot be learned well by warping, such as occluded areas and vice versa. Thus the warping with the optical flow is supported by the additional information of the other time step. Thereby, temporal information could already be learned in the encoder and included in the feature pyramid.

modifications	Training		Test			Runtime in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
/	21.1	**5.6**	24.6	**6.0**	54.8	**0.04**
concatenation of input as depicted in Fig. 6.7	21.1	9.4	22.4	10.6	90.2	0.05
concatenation of input as depicted in Fig. 6.7 and no application of correlation	21.4	5.7	25.0	6.1	**50.2**	0.05

Table 6.2.: Results of baseline and some modifications of it after 62 epochs

6.4. Replacing the Correlation with a Swin Transformer

In this section, we examine the replacement of the correlation by a transformer. First, this replacement is motivated. As stated in chapter 1.2, a fundamental goal of this work is to create better conditions for creating a scene flow predictor. To predict the future scene flow with monocular input data as accurately as possible, it is helpful to look at several past images. For example, a speed can only be determined unambiguously with measurements at two specific points in time. However, the acceleration and higher orders of the position-time derivatives cannot be determined unequivocally with two measurements. The more time-shifted measurements take place and the more measurements flow into a mathematical model for calculating current physical quantities, the more accurate this model tends to be. In our use case, this means that several past images and their features would need to be included in the model. This would involve large amounts of data which, in the case of parallel processing, would have to be stored simultaneously on the graphic card during training.

In the case of sequential processing, the neural network would take significantly more time than is currently the case and would therefore not be suitable for real-time applications anymore.
Additionally, the flood of data would make it difficult for the scene flow predictor to concentrate on the essential parts of the features. It would be desirable if the importance of the data could be evaluated. This is exactly one of the strengths of the self-attention mechanism, which is adopted by transformers. The transformer learns the relation between features and weights their importance.
In addition, transformers can process long sequences efficiently as described in section 3.4.1. This can be particularly useful when processing long sequences of features derived from images from the past. In doing so, features could also be discarded that receive a comparatively low weighting from the transformer in order to reduce the amount of data to be processed.
Another advantage of the transformer compared to the correlation operation is that transformers can be extended to process multiple input frames. The correlation can only ever be formed between two images. Transformers, on the other hand, can handle multiple input images. As a result, a longer time series of past images can be better taken into account for the prediction of the scene flow with a single transformer. The correlation would have to be calculated several times each considering individual pairs of frames. The best way to do this would be to use consecutive pairs of frames. However, this approach would neglect the connection between pairs of frames that are further apart in time.
The correlation was initially introduced to reflect the matching costs for associating the warped and the original features. However, these costs do not take into account occlusion and the associated errors in the warped image, which inevitably occur even when the image was warped with a true optical flow resulting from a true scene flow. In comparison to the correlation, the attention-mechanism as described in chapter 3.4 models the association between the warped and non-warped features. These associations should help the decoder to consider certain entries of the non-warped and warped features more than others. It is conceivable that the transformer provides the decoder with information about occluded areas in the warped and non-warped image which would allow the decoder to concentrate better on the other areas of the image when predicting the scene flow. This is possible because the transformer contains learnable parameters in contrast to the correlation. Due to these reasons, we aim to replace the correlation with a transformer. To be more precise, a Swin transformer [41] is used as a replacement due to its suitability in computer vision applications, which was explained in more detail in section 3.4.

Now a whole series of adjustments of the Swin transformer is necessary to use it instead of the correlation. In the original Swin transformer, the input consisted of features from a single image. Now that two features have to be compared with each other and with themselves, the inputs are first concatenated along the feature dimension. In contrast to correlation, this has the advantage that not only the connection between the elements of the warped and unwarped features is calculated, but also between the elements of the unwarped features among themselves and between the elements of the warped features among themselves.
The original Swin transformer was applied on unmodified input images. In contrast to those, the features to which the Swin transformer is applied to have a much lower spatial resolution. This is why patches are chosen to have a size of one. Another reason is that a reduction of spatial resolution is not desired. Furthermore, as we do not use the Swin transformer as a backbone all patch merging is left out and a 2x Swin Transformer Block as depicted in

Fig. 3.6 is used.
Now we address the choice of the windows which were introduced in chapter 3.4.2. In the original Swin transformer of paper "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows" [41], the attentions for square windows were determined because the input images were also square. However, the images of the KITTI data set are not square, which is why non-square windows are also chosen. It is assumed that windows that are as big as possible positively affect the quality of the scene flow estimation since that is the way to consider a high number of neighbouring patches. Therefore, we first choose the same window size 13×4 at all levels. Another reason for this choice is that the features of the pyramid level $L = 0$ have the lowest spatial dimension 13×4. The number of heads in the attention is set to 1 because the attention should be computed between all patches of a window and not between subgroups of patches. The whole architecture of this first version is depicted in Fig. 6.8. The dimensions of the computed quantities for pyramid level $L = 1$ are shown as an example.

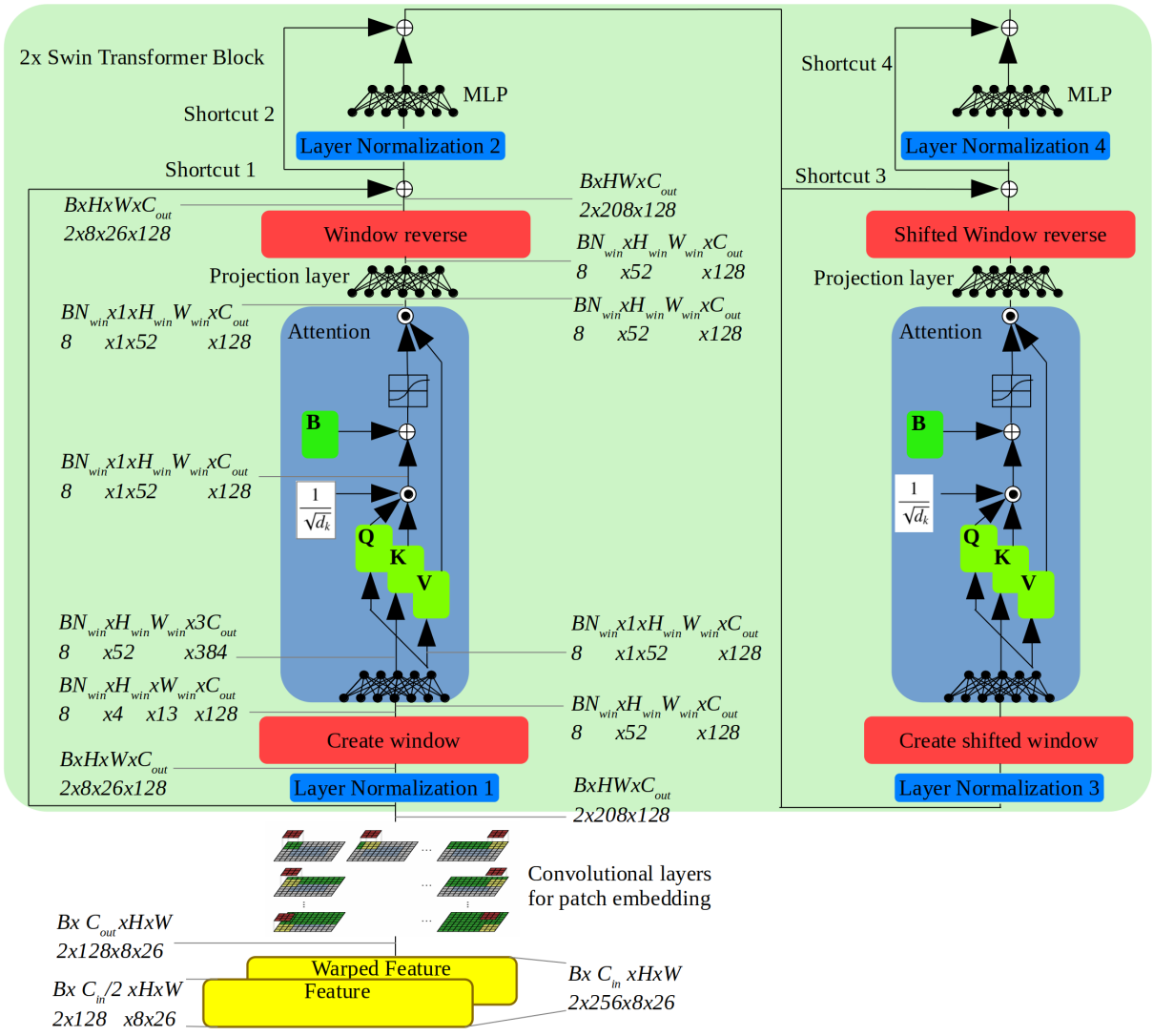


Figure 6.8.: Architecture of the fist implementation of the Swin transformer in this thesis with indications of the dimensions of the intermediate values of pyramid level $L = 1$

6.4.1. Ablation study of Swin transformer

It is striking that in the previous architecture, after the calculation of the actual attention, a few more operations were carried out. These include the multilayer perceptrons, which is named MLP in the Fig. 6.8, the shortcuts, and the computation of the attention on the shifted windows. We conduct an extensive ablation study on the transformer architecture to examine the effect of these components and their interactions on the performance. The results of those experiments are denoted in table 6.3.

modification	Training		Test			Runtime in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
/	21.2	6.9	24.0	7.4	68.9	0.09
shortcut 1 and 3 are missing	21.4	**6.8**	24.3	7.3	65.2	0.08
shortcut 1 and 3 and MLPs are missing	21.6	6.9	24.9	**7.2**	**59.4**	**0.06**
shortcut 1 and 3 and MLPs are missing and that no windows are shifted	21.3	**6.8**	24.2	7.4	70.3	**0.06**

Table 6.3.: Results of ablation study of Swin transformer depicted in Fig. 6.8 after 62 epochs

Simply omitting the shortcut 1 and 3 leads to less scene flow outliers compared to the architecture of Fig. 6.8. Also, the test loss decreases slightly by leaving out the shortcuts 1 and 3. However, this may also be due to more masking. An explanation for the reduced percentage of scene flow outliers could be that the values of the entries of the shortcut 1 and 3 are much higher than the values they are added to. This would mean that the actual attention contributes too little to the final result of the Swin transformer.
This assumption is checked by examining the intermediate values of the Swin transformer with the architecture shown in Fig. 6.8, which was trained up to epoch 62, during the computation of the forward scene flow in the forward pass. Fig. 6.9 basically illustrates the values of shortcut 1 in the first, the values of shortcut 3 in the third column, and the values to which they are added in the second and fourth column. For a better overview, the positions at which the values were extracted are marked in yellow above the individual columns in a schematic sketch of the Swin transformer. The first two columns refer to the values of the transformer before the windows were shifted. There it can be seen that shortcut 1 is not always significantly larger or smaller for all levels than the result after reversing the windows. However, by looking at the third and fourth columns, it can be seen that the values of the shortcuts 3 are significantly larger for almost all levels than the intermediate result after reversing the windows. This observation confirms the initial assumption that the shortcuts 1 and 3 overshadow the values they are added to.

Ultimately, the goal is to calculate the attention. However, the MLPs reduce the influence of the attention. Therefore, they are also omitted which further decreases the number of scene flow outliers.

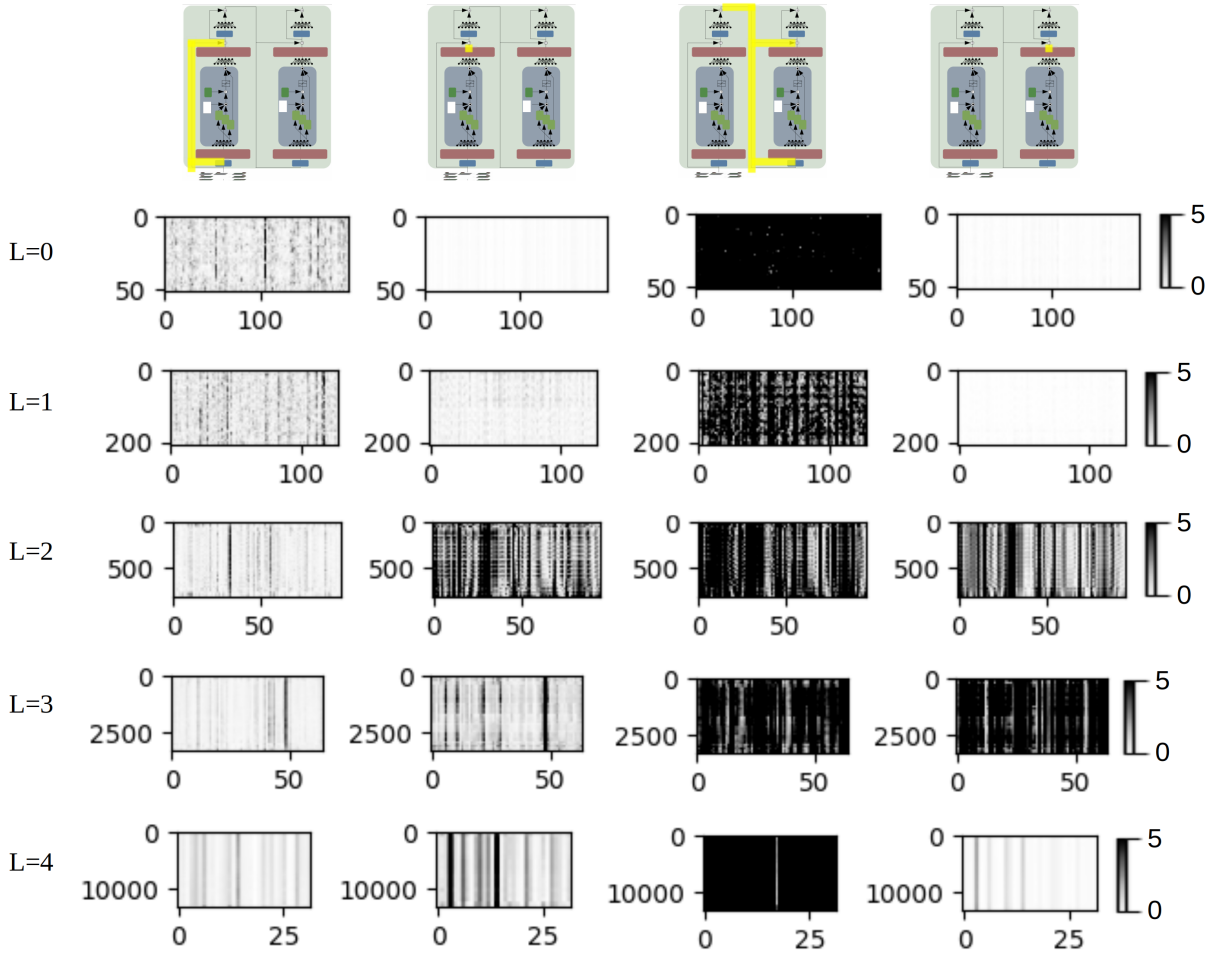


Figure 6.9.: Visualization of the values

- after the feature embedding in the 1^{st} column
- after reversing the windows in the 2^{nd} column
- added after reversing the shifted window 3^{rd} column
- after reversing the shifted window 4^{th} column

for all levels $0 \leq L \leq 4$ of the Swin tranformer with the architecture of Fig. 6.8 during the computation of the forward scene flow which was trained for 62 epochs
The signal path marked in yellow in the image at the top of each column shows where the values of the respective column was extracted from the Swin transformer depicted in Fig. 6.8
Meaning of x-axes: Channel dimension of value being less or equal to C_{out}
Meaning of y-axes: Spatial position of value being less or equal to HW

Finally, it is checked whether shifting the windows as described in the paper presenting the Swin transformer [41] is actually beneficial. An argument for shifting the windows is that it allows for attention across windows. However, with the application presented here, doubts may arise as to whether this is really necessary because the windows at the lowest pyramid levels relate to already very large windows in the image. As a result, the relevant neighbouring areas could already be fully taken into account. Neighbouring areas are considered to be

relevant if those areas depend on each other, for example, because they belong to the same object that has the same scene flow.
However, these doubts have not been confirmed. In particular, omitting the entire calculation of the attention of the shifted windows leads to more scene flow outliers and a higher test loss. In this experiment, the entire right block of Fig. 6.8 has been omitted. A reason for the poorer results may be that the area of the original image that is related to a window of the pyramid levels $L \geq 1$ is too small. In the following, the changes to the architecture that produced the best results are adopted.

6.4.2. Hyperparameter optimization of Swin transformer

In this section, the impact of the dimension of the feature embedding and the window size on the quality of the estimated scene flow is analyzed.
The embedding dimension of the features is the channel dimension after the convolution operation depicted in Fig. 6.8. It is called C_{out} and has the value 128 in the example of Fig. 6.8. So far, the dimension of the feature embedding of the stacked features and warped features was chosen to match the channel dimension of the features in each level. Consequently, the lower levels have a higher embedding dimension than the higher levels. As a result, the proportion of the output of the Swin transformer at the input of the decoder is significantly larger for small pyramid levels L than for large pyramid levels. However, the relevance of the output of the Swin transformer for small pyramid levels is much smaller than for larger ones because the Swin transformer processes the warped features, which are based on a very imprecise scene flow for small pyramid levels compared to large pyramid levels. Therefore, a constant embedding dimension could lead to better results. With a constant embedding dimension, the decoder can first focus on the given features of time t and on the scene flow from the last level until the scene flow is within a reasonable range before taking the derived and more abstract output of the Swin transformer more into account. In contrast to our approach, the authors of [1] chose a constant channel dimension of the correlation across all pyramid levels. A reasonable size for the embedding dimension could be 52 because it corresponds to the number of patches in a window. Even though the percentage of scene flow outliers increases, the test loss decreases by using a constant embedding dimension of 52 instead of the non-constant feature dimension as can be seen in table 6.4.
There are various reasons why the two KPIs, the test loss and the percentage of the scene flow outlier, are not similarly affected by changes in the neural network. With the test loss depending on the predicted scene flow, different areas are always masked out, which means that these relate to different regions. In addition, the portion of the image that is masked out varies which also varies the relevance of the test loss. In contrast to the test loss, the percentage of scene flow outliers always refers to the same regions in the image because the same regions are always masked out. However, the percentage of scene flow outliers depends on the definition of an outlier, and this, in turn, depends on defined threshold values resulting in several weaknesses described in detail in section 6.1. Therefore, it is necessary to be aware that these metrics are not perfect and that the bigger picture should always be considered. This can be done by looking at the test loss with the percentage of the area masked out together with the percentage of the scene flow outliers. The increased number of scene flow outliers of embedding dimension 52 could mean that the overall embedding dimension is too small. Since the embedding dimension 52 is smaller than the original embedding dimension

for $0 \leq L \leq 3$, the information is compressed. This compression may be too high.
In the paper [1], 81 channels were chosen for the channel dimension of the correlation and this dimension was optimized together with the other hyperparameters such as the number of channels of the features. Therefore, the influence of embedding dimension 81 is also examined. Increasing the embedding dimension from 52 to 81 significantly reduces the percentage of scene flow outliers. Nevertheless, the percentage of scene flow outliers is still greater using the embedding dimension of 81 compared to the non-constant embedding dimension. A further increase in the embedding dimension to 384 causes poorer KPIs. This may be because the significantly higher number of degrees of freedom than the number of positions to be coded, makes optimization harder. In addition, with such a high embedding dimension, the decoder could pay too much attention to the output of the Swin transformer compared to the other inputs of the decoder. The proportion of test loss reduction is greater than the increase in scene flow outliers comparing the Swin transformers with embedding dimension 81 to the Swin transformers with non-constant embedding dimension. Therefore, the results of the Swin transformer with embedding dimension 81 are seen as an improvement. In order to improve performance in this way, however, the runtime has to be increased from $0.06s$ to $0.08s$.

	Training		Test			Runtime in s
embedding dimension	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
192, 128, 96, 64, 32 for level 0 to 4 respectively	21.6	6.9	24.9	7.2	**59.4**	**0.06**
52 for all levels	21.8	**6.4**	24.7	**6.8**	65.7	**0.06**
81 for all levels	21.6	6.5	24.3	6.9	61.5	0.08
384 for all levels	21.9	7.6	24.8	8.1	65.1	0.10

Table 6.4.: Results of Swin transformer that is depicted in Fig. 6.8 wihtout shortcut 1 or 3 or MLPs after 62 epochs for different embedding dimensions

Now, in a further series of tests, based on the variant of the neural network with Swin transformer with the best result so far, the window size is varied. The results are summarized in table 6.5. Theoretically, the best result should be achieved when the window size corresponds to the spatial dimension of the features. Then, the attention could be computed on the whole feature and thereby learn relations even between distant features. However, this is not feasible because the GPU memory would not be sufficient. For $L = 0$ the window size cannot be increased further because the window size already matches the spatial dimensions of the features. But, this is not the case for the subsequent levels. Therefore, the width and height of the window are doubled to 26×8 for all levels $L > 0$. This results in a significantly reduced number of scene flow outliers.
It seems to be reasonable to increase the embedding dimension with the enlarged windows because the windows each contain more elements. A 26×8 window contains 208 elements. Therefore, the embedding dimension is adapted accordingly. Since the test loss and the percentage of scene flow outliers have hardly changed at the end of the training as can be seen in Fig. 6.10, it can be ruled out that the network should have been trained for several epochs longer because of additional weights resulting from the increased embedding dimension. The

test loss resulting from window size 26×8 and embedding dimension 208 is greater the test loss resulting from embedding dimension 81. One reason for this could be that the decoder deals with the result of the Swin transformer in an exaggeratedly intensive manner and at the same time neglects the other inputs, such as the previous estimate of the scene flow.
If the window size is further doubled to 52×16 for $L > 1$, then more scene flow outliers are observed again. Windows that are too large seem to lead to learning of dependencies between features that are far apart and no longer have anything to do with one another.

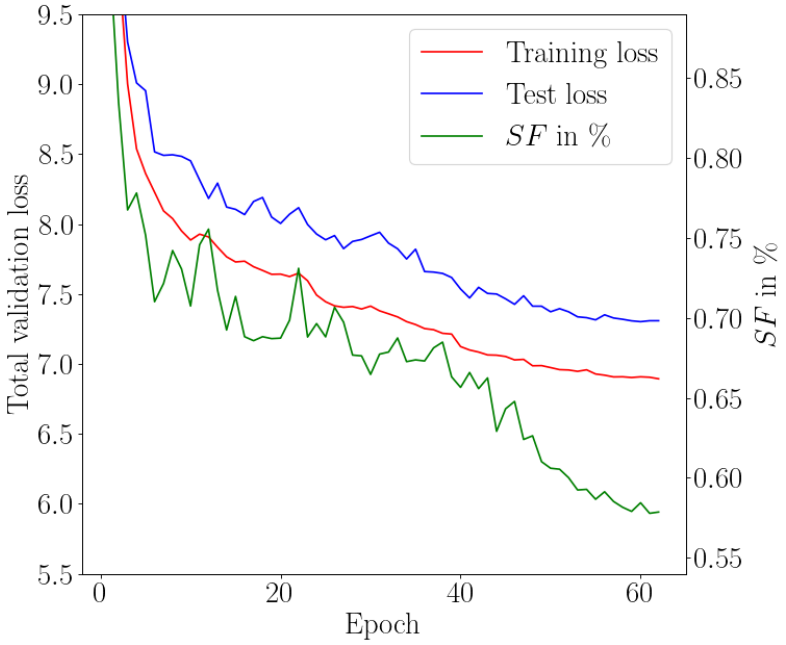


Figure 6.10.: Training-, test loss and percentage of scene flow outliers over epochs

modification	Training		Test			Run-time in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
window size 13×4	21.6	**6.5**	24.3	**6.9**	61.5	**0.08**
window size 26×8	21.4	6.6	24.1	7.0	**57.0**	**0.08**
window size 26×8 and embedding dimension 208 instead of 81	20.9	6.9	24.0	7.3	57.6	0.09
window size 52×16	20.8	6.6	23.6	7.2	65.9	**0.08**

Table 6.5.: Results of Swin transformer that is depicted in Fig. 6.8 wihtout shortcut 1 or 3 or MLPs after 62 epochs for different embedding dimensions and window sizes

The network with window size 26×8 uses $7.257GB$ for training, which is significantly more than the network with window size 13×4 occupying $4.228GB$. Therefore despite the superiority of the network with window size 26×8, we stick to the network with a window size of 13×4 to leave space on the graphics card for future network expansions in this

master's thesis and to be able to subject them to a fair comparison with previously presented networks.

6.4.3. Variation of positional encoding of the Swin transformer

The paper presenting the Swin transformer [41] found that the relative position bias ameliorates the overall result compared to no relative position bias or compared to an absolute position embedding bias. This statement is checked for its validity in our specific application. The results of the experiments conducted for this purpose are given in table 6.6.

modification	Training		Test			Run-time in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
relative positional embedding	21.6	6.5	24.3	6.9	61.5	0.08
absolute positional embedding after feature embedding	21.8	6.7	23.8	7.5	73.4	0.10
absolute positional embedding before feature embedding	21.4	**6.4**	24.2	**6.8**	60.1	**0.07**

Table 6.6.: Results of Swin transformer that is depicted in Fig. 6.8 wihtout shortcut 1 or 3 or MLPs and with an embedding dimensions of 81 and window size 13×4 after 62 epochs for different positional embeddings

For the first experiment, the pre-programmed option to enable absolute positional embedding is enabled and relative positional embedding is disabled. What is striking about the pre-programmed version of the absolute positional embedding is that, contrary to the ViT [39], it is applied to the embedded features and not to the input image. With the pre-programmed version of absolute positional embedding, the quality of the result decreases as described in the paper presenting the Swin transformer. [41].
Now the absolute positional embedding is carried out before the feature embedding as it is the case in the ViT except for a task-specific modification. Namely, the same positional encoding is learned for the warped and non-warped features. This modification would not have made sense if the positional encoding had only taken place after the feature embedding because the channels of the warped and non-warped features are no longer clearly separated after the feature embedding. Executing the positional encoding before the feature embedding has the clear advantage that it gives the opportunity to directly tell the neural network that the warped and non-warped features refer to the same positions. As a result, this connection does not have to be learned first and errors in learning this connection are prevented from the outset. This way of positional encoding reduces the test loss as well as the scene flow outliers. The architecture of the resulting Swin transformer together with the adaptations adopted so far is shown in Fig. 6.11.

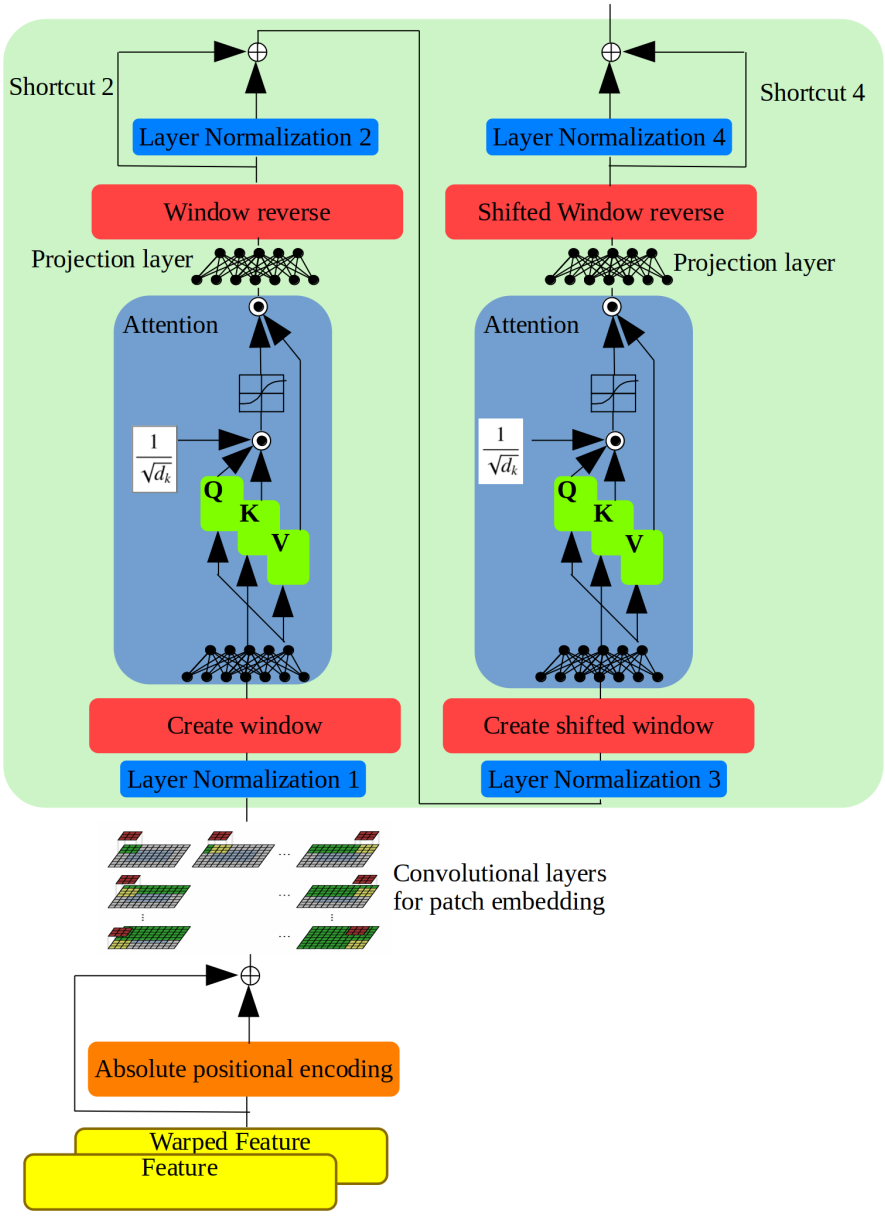


Figure 6.11.: Swin transformer including all modifications so far

6.4.4. Variation of attention mechanism

Parallel to the experiments listed in table 6.6, it is checked whether the stacking of the features and the warped features and the subsequent application of the self-attention mechanism is actually the optimal way to process these features and their warped version.
Alternatively, the cross-attention [67] between the image and the warped image could be computed. The main difference between self-attention and cross-attention is the way the value, key, and query are generated. While one input image is used to determine the query, key, and value for the computation of the self-attention, one input image is used to determine the query and another to determine the key and value for the computation of the cross-attention. However, using a cross-attention instead of the self-attention mechanism leads to clearly poorer KPIs as can be seen in table 6.7. It is possible that the optimum lies some-

where in between self-attention and cross-attention. The benefit of cross-attention is that it inherently processes two inputs, which is the case here. Furthermore, we are primarily interested in the relation between the features and the warped features. However, a pixel-wise comparison would not be particularly meaningful because a pixel that belongs to a specific location on an object can take completely different values in different images. The reasons for this are, among other things, different lighting and the associated differences in shading and reflection behavior. In our case, the cross-attention is not applied to the original image, in which the attention would be calculated pixel by pixel by multiplying query and key, but to the image features. Nonetheless, these individual features relate to small regions in the image.
To counter the comparisons between small image regions, the value that weights the attention is now calculated based on the stacked features and warped features. In this way, we expect more inclusion of context information from the neighborhood because the warping is based on an imperfect optical flow, and channels from neighboring features are therefore also included in the calculation of the value. This measure significantly reduces the test loss compared to the previous cross-correlation calculation method. However, the evaluation of the test loss should be treated with caution, since a larger proportion of the pixels are masked than in the other experiments in this series of experiments. By calculating the value of the cross-correlation with the features and warped features, the percentage of scene flow outliers decreases significantly. However, it is still significantly higher compared to the approach with the computation of self-attention. In summary, the stacking of the warped features and the features combined with the subsequent calculation of the self-attention is important because in this way the relationships between the channels of the individual features are also learned.

modification	Training		Test			Runtime in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
stacked features and warped features are used for the generation of query, key and value	21.6	**6.5**	24.3	**6.9**	**61.5**	**0.08**
use features for the generation of the key and value and use warped features for the generation of the query	22.1	6.8	23.8	7.6	82.4	**0.08**
use features for the generation of the key, warped features for the query and both stacked for the value	21.8	**6.5**	24.8	**6.9**	71.2	0.09

Table 6.7.: Results of Swin transformer that is depicted in Fig. 6.8 wihtout shortcut 1 or 3 or MLPs, with an embedding dimensions of 81, window size 13×4 and relative positional encoding after 62 epochs for different ways of computing the attention

6.4.5. Further modifications

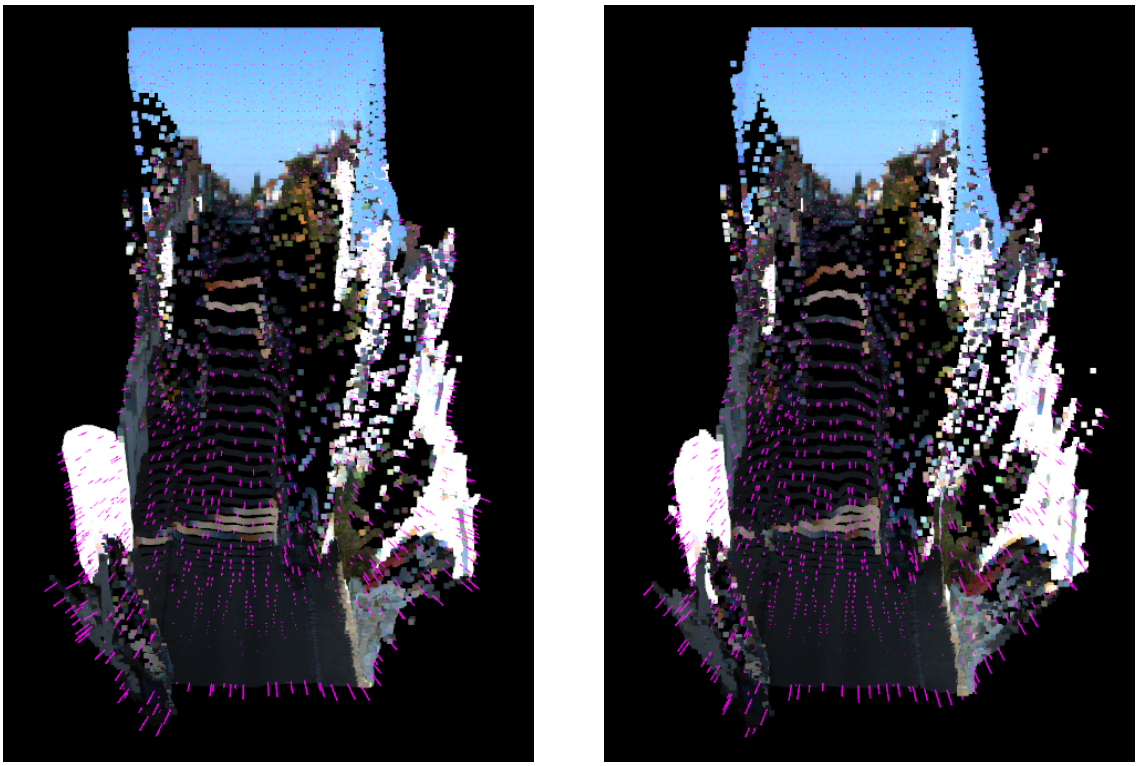
The decoder gets among other values the features and the output of the Swin transformer, which reflects the importance of the relationship between each feature, as input. However, it might be difficult for the decoder to interpret the output of the Swin transformer if the warped features are unknown to the decoder.
Therefore, it is examined if it is helpful to also pass the warped features directly into the decoder. As can be seen in table 6.8 this measure substantially reduces the test loss as well as the percentage of scene flow outliers. Although the test loss is greater than the test loss of the neural network using the correlation, the percentage of scene flow outliers is smaller by 4.2 percentage points. It is possible that the Swin transformer learns information about areas in the image, which are masked out by the training and test loss, and in this way supports the decoder in making more accurate scene flow estimates in these regions. That would explain the comparatively high test loss with a simultaneously low number of scene flow outliers. The results from table 6.8 show that the results obtained with Swin transformers are at least as good as with correlation. However, it must be taken into account that the Swin transformer increases the runtime from $0.04s$ to $0.1s$.

modification	Training		Test			Runtime in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
/	21.4	6.4	24.2	6.8	60.1	0.07
use warped features as additional input of the decoder	22.1	6.2	25.0	6.4	**50.6**	0.10
use warped features as additional input of the decoder and use the difference between the features and the warped features as input of the Swin transformer	22.1	5.8	25.0	6.1	53.6	0.09
correlation instead of Swin transformer	21.1	**5.6**	24.6	**6.0**	54.8	**0.04**
correlation instead of Swin transformer and use warped features as additional input to the decoder	21.7	6.4	24.5	6.7	64.1	0.06

Table 6.8.: Results of Swin transformer that is depicted in Fig. 6.8 without shortcut 1 or 3 or MLPs, with an embedding dimensions of 81, window size 13×4 and absolute positional encoding before patch embedding after 62 epochs

Now, the combination of the usage of the correlation operation and the transfer of the warped features to the decoder is tested. This combination worsens all KPIs. This is probably due to the fact that the correlation already contains information about the adjustment direction of the scene flow and that the warped features, which have become superfluous for the decoder,

distract from the other inputs of the decoder.
As an alternative to the stacked features and warped features, the difference between the features and the warped features could also be calculated directly. This means that there is only one input for the Swin transformer and that the self-attention mechanism can be used as it was originally intended. This can reduce the test loss at the expense of the percentage of scene flow outliers. If a longer time series of features is available, however, pairwise differences have to be calculated with this version, which raises the following question. Which differences between which features should be calculated? This would create an additional degree of freedom that could be exploited in future work. By simply stacking the features, the Swin transformer is easily expandable for processing time series. Therefore, we stick with this version of the Swin transformer.



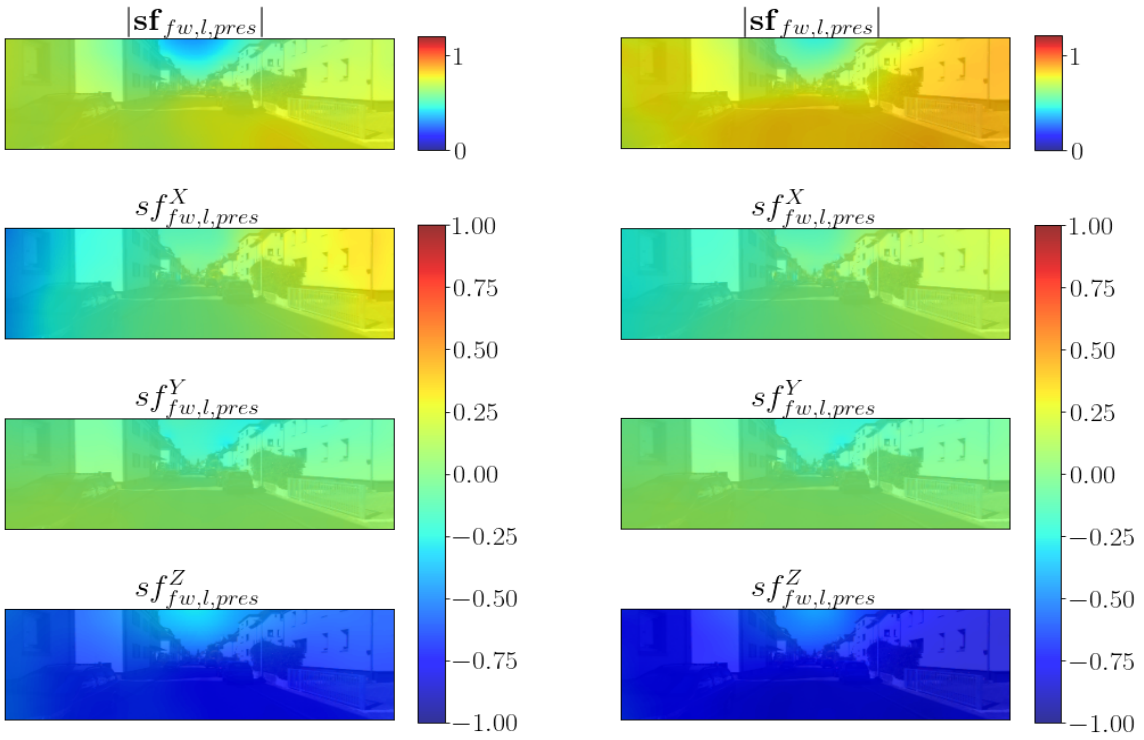
(a) Baseline using correlation

(b) Modified version using Swin transformer

Figure 6.12.: Visualization of the point clouds $\mathbf{M}_{t,l}$ and forward scene flow $\mathbf{sf}_{fw}$ from $\mathbf{I}_{t-1,l}$ to $\mathbf{I}_{t,l}$ generated by the baseline and the modified version using a Swin transformer as depicted in Fig. 6.5 and passing the warped features to the decoder as additional input

Finally, the results of the Swin transformer with the stacked warped and non-warped features are visually compared to those of the baseline with the correlation and the plausibility of the output using the Swin transformer is checked. Therefore, the two subsequent images of Fig. 6.5 are considered. The estimated point clouds corresponding to $\mathbf{I}_{t,l}$ and the forward scene flow from $\mathbf{I}_{t-1,l}$ to $\mathbf{I}_{t,l}$ are illustrated in Fig. 6.12. On the whole, the point clouds differ only marginally and most importantly both scene flows as well as the corresponding point clouds are plausible.
In a component-wise comparison of both generated scene flows in Fig. 6.13, it is striking that

the scene flow $sf^Z_{fw,l,pres}$ is significantly more negative when replacing the correlation with a Swin transformer. However, the scene flow $sf^X_{fw,l,pres}$ is significantly closer to zero on the far left and far right of the image. This may be because the Swin transformer learns information about occluded areas.



(a) Baseline

(b) Modified version of baseline using a Swin transformer as depicted in Fig. 6.5 and passing the warped features to the decoder as additional input

Figure 6.13.: Visualization of magnitude and components of forward the scene flow of the left camera

7. Scene flow prediction

In this chapter, the framework of the scene flow estimator is expanded in such a way that it additionally predicts the scene flow between the known current frame $\mathbf{I}_{t,l}$ and the subsequent future frame $\mathbf{I}_{t+1,l}$. Before going into the extension, it is first repeated which framework of the scene flow estimator generated the best results in terms of accuracy. This estimator is similar to the architecture of Fig. 6.3 except that both warped features $\tilde{\mathbf{x}}_{t-1,l}$ and $\tilde{\mathbf{x}}_{t,l}$ are passed directly to the decoder and the correlation is replaced with the Swin transformer depicted in Fig. 6.11.
For the prediction of the future scene flow and the future disparity, a decoder and context network identical to the previous architecture of 6.3 are added. The decoder and the context network were shown in more detail in Fig. 6.3 and are combined for clarity in Fig. 7.1. The parts of the architecture that have been modified or added compared to the architecture of the baseline are marked in yellow. The prediction decoder has the same inputs as the estimation decoder for $L = 0$. For the levels $1 \leq L \leq 4$, the prediction decoder additionally processes the disparity and scene flow generated by the estimation context network of the last level. This is done because it is assumed that the prediction decoder benefits from the estimates of the estimation decoder, which should be more reliable than the prediction because it is based on images of the corresponding timestamps.

Now the loss is adapted. The loss L_{pres} of the scene flow estimator of equation (3.1) remained unchanged and is exactly the same as the one used in [1]. This loss is now supplemented by another loss L_{fut} to additionally predict the future forward and backward scene flow between the frames $\mathbf{x}_{t,l}$ and $\mathbf{x}_{t+1,l}$. In addition to predicting the scene flows, the corresponding disparities are also predicted because it is necessary to have a prediction of the disparity of the future frame to reconstruct the corresponding point cloud that is used to predict the future scene flow. Furthermore, information from the future frame $\mathbf{I}_{t+1,l}$ that is available for calculating the losses can be exploited more extensively. For these reasons, the future loss L_{fut} has the same components as L_{pres} except that the timestamp of each estimated disparity and scene flow and the corresponding frames is increased by one.
Consequently, the disparities $d_{t+1,l}$, $d_{t,l}$, the predicted forward scene flow $sf_{t+0\rightarrow 1,l}$ and the predicted backward scene flow $sf_{t+1\rightarrow 0,l}$ are required for the computation of L_{fut}. From these quantities, the disparity $d_{t,l}$ is estimated both by the estimation part and by the prediction part of the architecture. Therefore, a disparity consistency loss

$$L_{disp,cons} = \sum_{L} \frac{1}{N(L)} \sum_{\mathbf{p}} |d_{t,l,pres} - d_{t+1,l,fut}| \quad , \tag{7.1}$$

whereby $N(L)$ represents the number of pixels of the features of the corresponding pyramid level L, is introduced. The reason for the employment of this loss is that both disparities should be the same. These modifications could be helpful for the computation of $d_{t+1,l}$ because the same channels are used once for the computation of $d_{t,l,fut}$ and once for $d_{t,l}$ as can

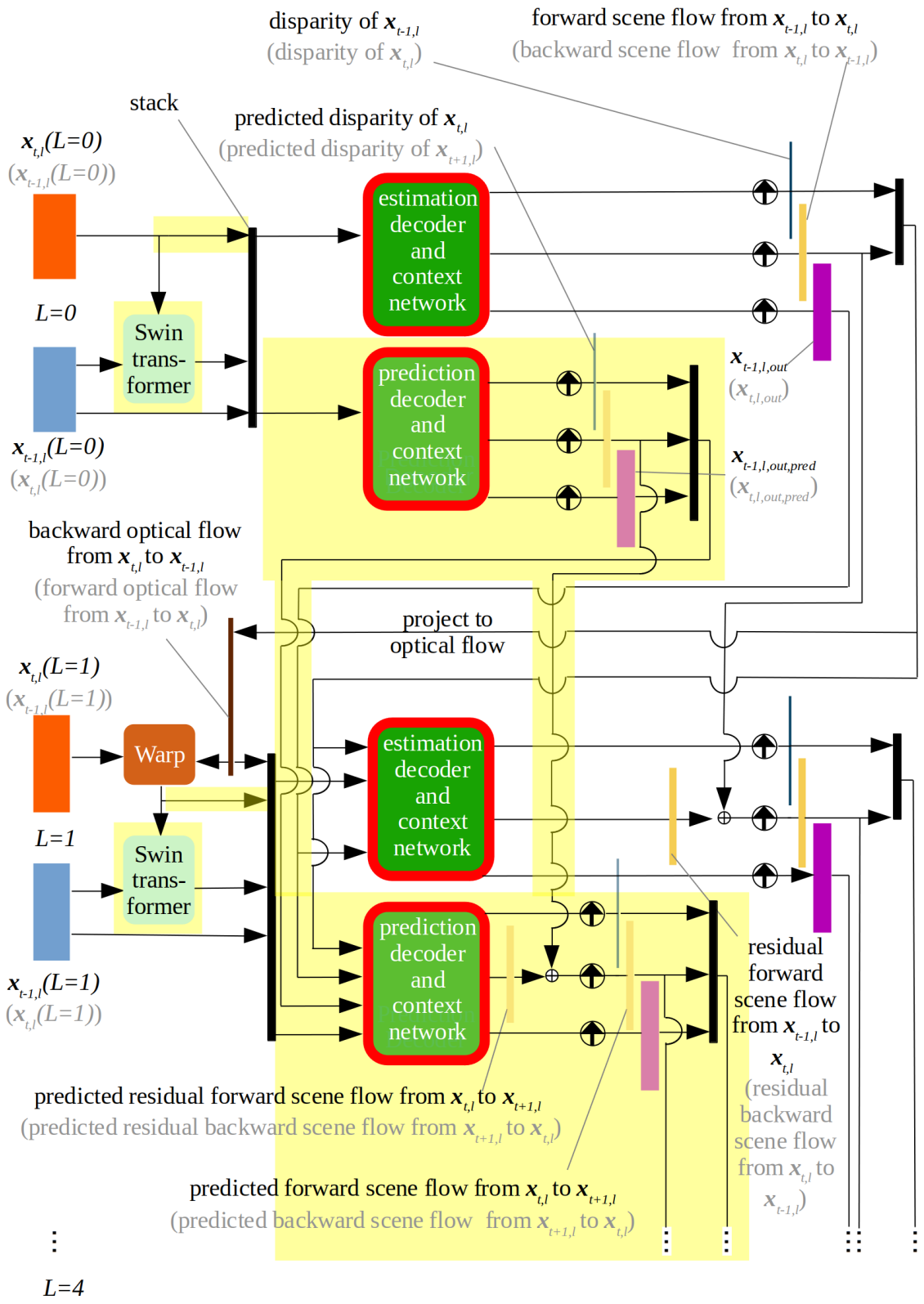


Figure 7.1.: Architecture of unsupervised monocular scene flow predictor with Swin transformer depicted in Fig. 6.11 with estimation-, prediction decoder and context network with the same structure as those depicted in Fig. 6.7

be seen in the black an white writing of Fig. 7.1. The disparity consistency loss is added to the other losses with a constant β in order to receive the total loss

$$L_{total} = L_{pres} + L_{fut} + \beta L_{disp,cons} \quad . \tag{7.2}$$

Now the question arises which metrics can be used to evaluate the predicted scene flows and disparities. The KITTI Scene Flow 2015 dataset, which was previously used as a test set, can not be used to evaluate prediction because this dataset only ever contains image pairs of different street scenes. However, there should be three consecutive images with the timestamps $t-1$, t, and $t+1$ because the first two are required for the forward pass and the last two for determining the scene flow outlier. The Kitti Raw dataset, from which the training and test dataset was formed, meets this requirement. For this reason, the test loss is mainly used as an evaluation metric of the future scene flow and disparity.
One danger that exists when learning the future and current scene flow together is that the values of the estimated scene flow $sf_{t-1\rightarrow 0,l}$ are adopted for the future scene flow $sf_{t+0\rightarrow 1,l}$ one-to-one. The possibility of this naive copying of the values should be considered while evaluating. Therefore, for a better classification of L_{fut}, a new metric $L_{fut,comp}$, which is not incorporated into the total loss used during training, is introduced. This metric contains the same loss components as L_{fut} except that the estimated scene flows and disparities are used instead of the predicted ones. Compared to L_{pres}, the metric $L_{fut,comp}$ is not calculated on the basis of the image pair $\{I_{t-1}, I_t\}$, but on the basis of the image pair $\{I_t, I_{t+1}\}$. The metric $L_{fut,comp}$ thus indicates how large L_{fut} would be if the neural network had learned to adopt the estimated scene flows and disparities one-to-one as predicted scene flows and disparities. So if $L_{fut} < L_{fut,comp}$ holds, then it can be concluded that the neural network learns something about the future. If the scene flow and disparity prediction are not better than the estimated quantities and if the predicted quantities are of higher quality compared to the adoption of the estimated quantities , $L_{pres} < L_{fut} < L_{fut,comp}$ should hold. However, it can happen that the neural network does not learn anything about the future.
Since there is an expectation for the interval in which L_{fut} is located, another metric c_{learn} normalized to this interval is introduced, which is intended to indicate how much has been learned about the future compared to the present. This metric

$$c_{learn} = \frac{L_{fut,comp} - L_{fut}}{L_{fut,comp} - L_{pres}} \tag{7.3}$$

reflects the relative prediction ability. The following cases can theoretically occur. For

- $c_{learn} < 0$: the predictions are of worse quality than they would be if estimated quantities were adopted
- $0 = c_{learn}$: the predictions are of same quality as they would be if estimated quantities were adopted
- $0 < c_{learn} < 1$: the predictions are of better quality than they would be if estimated quantities were adopted, but of worse quality than the estimations
- $c_{learn} = 1$: the predictions are of same quality as the estimations
- $1 < c_{learn}$: the predictions are of a higher quality than the estimations

holds true. Furthermore, it is initially assumed that the predicted quantities should not be better than the estimated quantities in an epoch-by-epoch comparison. The reason is that

images are available as an input of the neural network $\{I_{t-1}, I_t\}$ with the same timestamps as all estimated quantities, which is not the case for the predicted quantities for timestamp $t+1$.

The architecture of Fig. 7.1 is now tested and evaluated for different parameters β. The results are given in table 7.1. The percentage of scene flow outliers SF only evaluate $sf_{t-1\to 0}$ and $sf_{t-0\to 1}$ because ground truth depth and optical flow is only given for two subsequent pairs of frames $\mathbf{I}_{t-1,l}$ and $\mathbf{I}_{t,l}$ in the test set KITTI Scene Flow 2015. All other entries refer to the other test set, which is part of the KITTI Raw dataset. The runtime refers to the forward pass during testing. The percentage of masked pixels for the computation of $L_{fut,comp}$ equals the percentage of the masked pixels for the computation of L_{pres} because both losses are based on the same scene flows. Since $c_{learn} > 0$, it can be concluded that this scene flow predictor is already able to learn information about the future.
Table 7.1 shows the clear tendency that an increase of β leads to an increase in L_{fut}. This relation also exists in a less pronounced form between β and L_{pres}. The results for $\beta = 5$ seem to represent outliers that do not confirm the trend. In addition, the scene flow outliers, which refer to the quantities with timestamp $t-1$ and t, are each significantly higher than if only scene flow estimation is carried out. The problem could be that the disparity consistency loss does not adjust the weights of the prediction decoder and context network, as was mainly desired, but also the weights of the estimation decoder and context network. If $d_{t,l,fut}$ is less accurate as $d_{t,l,pres}$, which should be the case because $\mathbf{I}_{t+1,l}$ is not given as input, the disparity consistency loss worsens the quality of $d_{t,l,pres}$ by forcing the disparities to be similar. However, the disparity $d_{t,l,pres}$ is again used as input for the decoders of the subsequent level. Therefore, this error propagates from level to level more and more. To conclude, the negative influence on the estimation by the disparity consistency loss could be an explanation for the high percentage of scene flow outliers compared to the best network with Swin transformer only making estimations.

β	$L_{disp,cons}$	c_{learn} in %	masked pixels in %	L_{fut}	masked pixels in %	L_{pres}	$L_{fut,comp}$	SF in %	Runtime in s
1	**0.0101**	**57.8**	22.5	**6.92**	24.7	6.30	7.77	63.5	0.14
5	0.0060	39.3	22.4	7.09	24.6	**6.21**	**7.66**	62.4	0.14
20	0.0031	56.8	22.5	7.08	24.4	6.60	7.71	**62.2**	0.14
100	0.0012	52.2	22.2	7.13	22.2	6.53	7.90	63.7	0.14
500	$2 \cdot 10^{-7}$	0.0	20.0	10.30	20.0	10.30	10.30	100.0	0.14

Table 7.1.: Results of scene flow estimator and predictor that is depicted in Fig. 7.1 and results of its modifications after 62 epochs

This issue is dealt with by excluding the disparity consistency loss from the total loss, by ignoring $d_{t,l,fut}$ entirely and using $d_{t,l,pres}$ instead. The neural network with the resulting architecture is trained and evaluated. Its results are given in table 7.2. First, a comparison between the scene flow estimator and its extension, which additionally performs scene flow prediction, is done. By additionally performing scene flow prediction, the percentage of scene flow outliers decreases slightly from 50.6% to 49.6% and L_{pres} also decreases slightly from 6.44 to 6.35. The prediction of the scene flow and disparity nearly doubles the runtime from $0.08s$ to $0.14s$. However, the fact that L_{fut} is smaller than $L_{fut,comp}$ shows that

the network is capable of predicting the future. In addition, the relative prediction ability is $c_{learn} = 52\%$, which is quite remarkable.

modification	c_{learn} in %	masked pixels in %	L_{fut}	masked pixels in %	L_{pres}	$L_{fut,comp}$	SF in %	Runtime in s
no extension to perform prediction	/	/	/	25.0	6.44	/	50.6	0.08
/	52.2	22.2	6.96	24.6	6.35	7.74	**49.6**	0.14
replace Swin transformer by correlation and not use the warped features as input for the decoder	**73.2**	21.9	7.13	24.3	6.78	8.13	64.8	**0.10**
apply transformer on $\mathbf{x}_{t-2}$, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ instead of on the warped and non-warped features	65.7	22.2	**6.79**	24.5	**6.30**	**7.71**	/	0.15

Table 7.2.: Results of scene flow estimator and predictor that is depicted in Fig. 7.1 except that $d_{t,l,pres}$ is used instead of $d_{t,l,fut}$ in the loss and in the architecture and results of its modifications after 62 epochs
The column with the modifications always refers to version of the neural network described in this caption
The percentage of scene flow outliers can not be computed in the last experiment because the KITTI Scene Flow 2015 dataset only contains two subsequent images

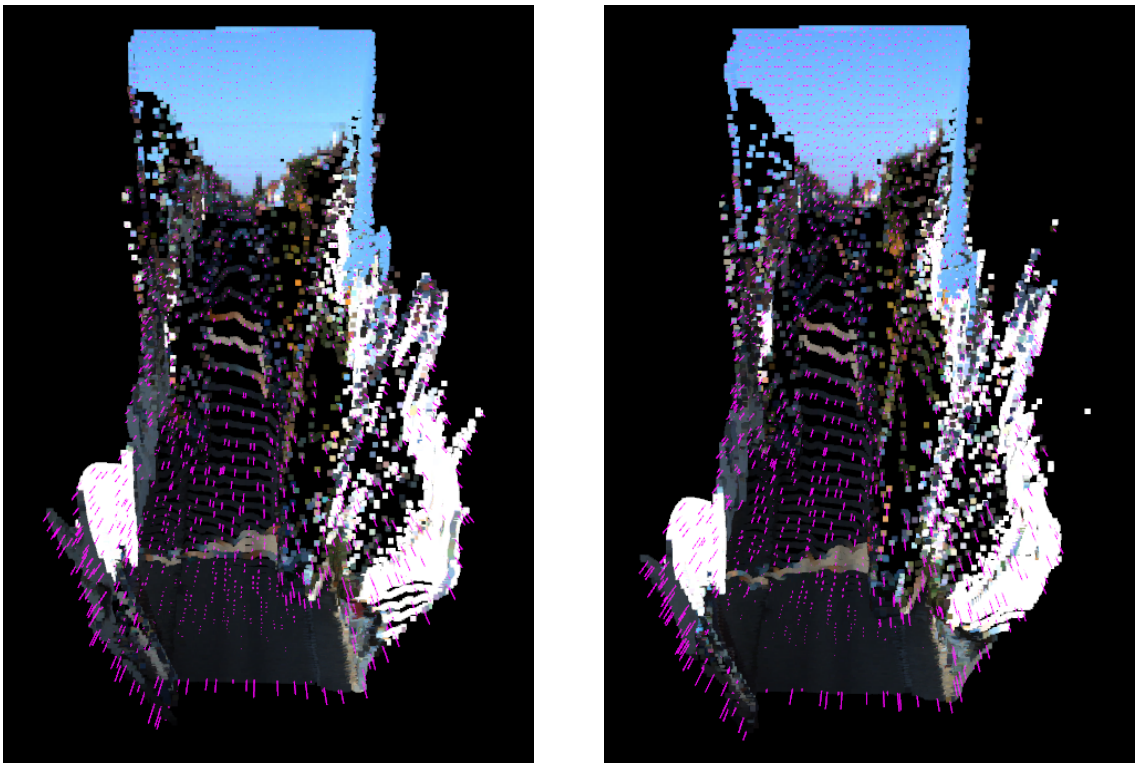
Now the contribution of the Swin transformer is analyzed. Therefore, the Swin transformer is replaced by the correlation operation. For scene flow estimation using the correlation operation, the processing of the warped features by the decoder is disadvantageous compared to the baseline as shown in table 6.8. Therefore, the warped features are not used as input for the decoder. The results of the neural network using a correlation operation and not processing the warped features in the decoder can be seen in table 7.2. This neural network is now compared with the scene flow estimator and predictor that uses a Swin transformer and that process the warped features in the decoder. By using the correlation operation, the runtime can be reduced significantly. Since $L_{fut} < L_{fut,comp}$ holds true, it is possible to learn information about the future by computing the correlation instead of using a Swin transformer. However, the test loss L_{fut} increases from 6.96 to 7.13 and L_{pres} increases drastically from 6.35 to 6.78 by computing the correlation. Since the present and future losses are worse when using the correlation operation instead of a Swin transformer, it can be concluded that the application of the Swin transformer facilitates the simultaneous computation of the present and future scene flow during a single forward pass. The comparison of the KPIs shows that the employment of the Swin transformer is beneficial for scene flow estimation as well as

prediction.

An additional past frame is now taken into account in order to achieve more accurate predictions of the future. So far, only the last two captured frames $\mathbf{I}_{t-1}$ and $\mathbf{I}_t$ were considered during inference. Now the frame $\mathbf{I}_{t-2}$ is also considered. Therefore, a feature pyramid is now also determined for the frame $\mathbf{I}_{t-2}$. So far, the Swin transformer received the stacked features $\mathbf{x}_{t-1}$ and $\tilde{\mathbf{x}}_{t-1}$ or $\mathbf{x}_t$ and $\tilde{\mathbf{x}}_t$. Extending the approach by additionally applying the Swin transformer on the stacked features $\mathbf{x}_{t-1}$ and warped features $\tilde{\mathbf{x}}_{t-2}$ can not be done readily because the current neural network does not compute the warped features $\tilde{\mathbf{x}}_{t-2}$. These warped features are not computed because no scene flow between $t-2$ and $t-1$ and no disparity at time $t-2$ is estimated. These scene flows and disparities could be estimated in the same way as those of the frames $\mathbf{I}_{t-1}$ and $\mathbf{I}_t$. However, the temporal connection between frames that are not temporally adjacent would remain unconsidered in this way. In addition, this would not exhaust the transformer's potential for efficient processing of time series because the transformer would process the features and warped features three times separately for all three timestamps. Instead we propose to use the Swin transformer on the stacked features $\mathbf{x}_{t-2}$, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$. Thereby, the transformer's strength compared to the correlation to process multiple inputs is exploited. It can be criticized that this approach is disadvantageous because the warped features are not processed by the Swin transformer. Since the features are warped using the optical flow, which is the projection of the estimated scene flow of the last level, there is less feedback of the last scene flow. Instead of our proposition, $\{\mathbf{I}_{t-2}, \mathbf{I}_{t-1}\}$ could also be processed in the same way as $\{\mathbf{I}_{t-1}, \mathbf{I}_t\}$ is processed by also estimating the corresponding disparities and scene flows. However, with our proposition, there is no reliance on the past estimated disparity at time $t-2$ or on past estimated scene flows between the frames $\mathbf{I}_{t-2}$ and $\mathbf{I}_{t-1}$ and those quantities do not have to be determined additionally which would increase runtime.
Table 7.2 shows that applying the Swin transformers on the stacked features $\mathbf{x}_{t-2}$, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ significantly improves the L_{fut} while L_{pres} slightly decreases. Overall, this small change already increases the prediction ability by 13.5 percentage points while increasing runtime only slightly. The KITTI Scene Flow 2015 dataset only ever provides ground truth for two subsequent camera frames. However, three frames are needed to execute the forward path in this case. Consequently, the percentage of scene flow outliers can not be computed.

Finally, the constructed point cloud of the scene flow estimator and predictor are visualized. Both corresponding point clouds are illustrated in Fig. 7.2 along with their scene flow. As expected the reconstruction of the future point cloud $\mathbf{M}_{t+1,l}$ is less accurate than the reconstructed point cloud $\mathbf{M}_{t,l}$. since the frames of both temporally adjacent neighbouring frames are given for the point cloud $\mathbf{M}_{t,l}$, but the future temporally adjacent frame $\mathbf{I}_{t+1}$ of $\mathbf{M}_{t+1,l}$ is not given as input. This deterioration in the quality of the reconstruction of $\mathbf{M}_{t+1,l}$ is particularly evident on the roofs of the houses further away on the left-hand side of the street.

Fig. 7.3 illustrates the magnitude of the scene flow along with its components It strikes the eye is that the scene flow component $sf^Z_{fw,l,pres}$ of the neural network that makes estimation and prediction has greater values than that of the neural network that only makes estimation. Furthermore, it can be seen that the magnitude of the scene flow of the estimator and predictor decreases over time. The change of the magnitude is due to the forward scene flow $sf^Z_{fw,l,pres}$ of the ego-vehicle. It is now checked whether the vehicle is actually slowing down. Therefore, the forward velocity of the KITTI Raw dataset corresponding to the frames $\{\mathbf{I}_{t-2}, \mathbf{I}_{t-1}, \mathbf{I}_t \mathbf{I}_{t+1}\}$ are considered in table 7.3. As can be seen in the table the ego-vehicle



(a) Point cloud $\mathbf{M}_{t,l}$ and scene flow $sf_{t-0\to1}$

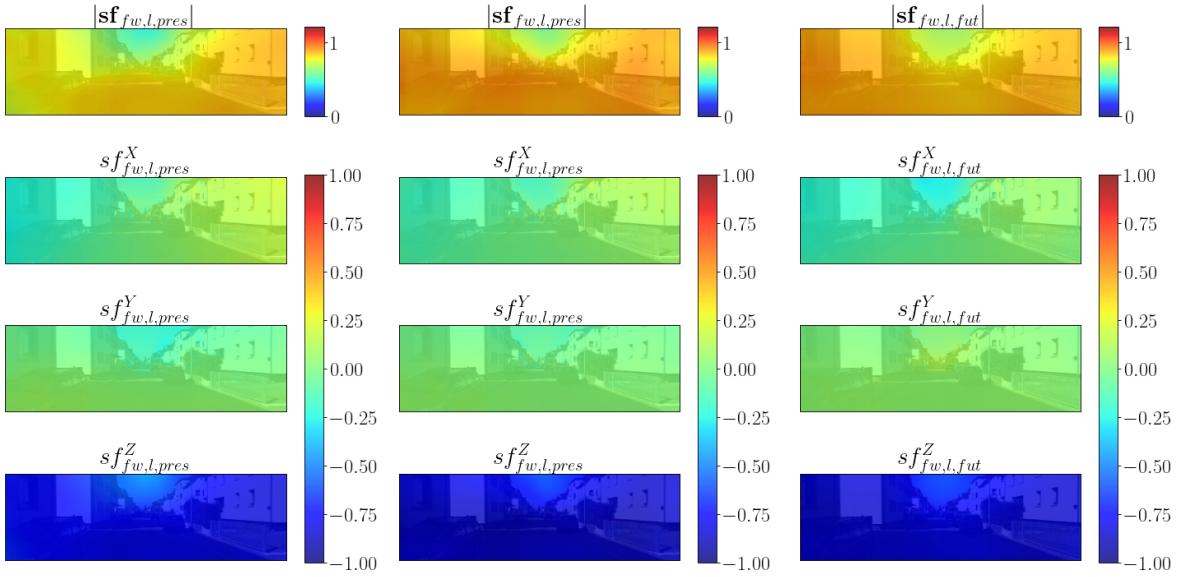
(b) Point cloud $\mathbf{M}_{t+1,l}$ and scene flow $sf_{t+0\to1}$

Figure 7.2.: Visualization of the point clouds and forward scene flow $\mathbf{sf}_{fw}$ generated by the baseline and the scene flow estimator and predictor as depicted in Fig. 7.1 except that the Swin transformer is applied on on $\mathbf{x}_{t-1}$, $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ instead of on the warped and non-warped features

slightly slows down. So the trend of the prediction is correct. As expected, the results are plausible overall.

timestamp	$t-2$	$t-1$	t	$t+1$
forward velocity in $\frac{m}{s}$	10.51	10.47	10.43	10.39

Table 7.3.: Velocities of frames of KITTI Raw dataset from drive 27 for the indices 3059 until 3062 corresponding to the timestamps $t-2$ until $t+1$ of the 10^{th} October



(a) Scene flow estimator with Swin transformer before extending it to a scene flow predictor

(b) Present scene flow between $\mathbf{M}_{t-1,l}$ and $\mathbf{M}_{t,l}$

(c) Future scene flow between $\mathbf{M}_{t,l}$ and $\mathbf{M}_{t+1,l}$

Figure 7.3.: Visualization of forward scene flows generated by the scene flow estimator and predictor as depicted in Fig. 7.1 except that the Swin transformer is applied on on $\mathbf{x}_{t-2}$, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ instead of on the warped and non-warped features

8. Conclusion and future work

In conclusion, in this work, we have examined the effect of processing the two subsequent input frames simultaneously in the same encoder on the quality of the scene flow estimator to enable learning temporal features. Thereby, the baseline was outperformed. The percentage of scene flow outliers has been reduced from 54.8% to 50.2%. Furthermore, different variations of the Swin transformers and several ways to incorporate them into a preexisting architecture were analyzed. Thereby, the scene flow estimator was optimized. As a result, a scene flow estimator with similar performance compared to the baseline was created. Contrary to the baseline, the Swin transformer can process multiple past frames whereas the correlation can only ever be formed between two frames. Consequently, the Swin transformer forms a better basis for scene flow prediction which benefits from the consideration of multiple past frames. Finally, the architecture including a Swin transformer was extended to also predict the scene flow. Lastly, we took advantage of the strengths of the scene flow estimator by inputting the last three frames instead of only the last two into our neural network. Thereby, the quality of the predicted scene flow was improved.

Stacking the input frames and processing them with one big encoder significantly reduced the percentage of scene flow outliers from 54.8% to 50.2%. The reason could be that this modified architecture already enables the encoder to learn temporal connections between the input frames as well as information about possible occlusions between the frames.
Integrating the Swin transformer into the architecture of the baseline requires a lot of adjustments. For example, the omission of shortcuts 1 and 3 and the MLPs of the Swin transformer reduced the scene flow outlier by 9.5%. These math operations obscure the actual attention computed by the Swin transformer, preventing the decoder from getting the pure attention. Instead, the decoder obtains an input that does not reflect the degree of association between the features and the importance of these associations anymore. Subsequently, a constant embedding dimension of the Swin transformer of 81 was chosen instead of keeping the dimension of the input feature as the embedding dimension. This time the test loss, which has hardly changed in the previous modifications, has even improved noticeably. Namely, it declines by about 0.3 from 7.2 to 6.9. Although the percentage of scene flow outliers increased by 2.1%, the change of the embedding dimension is interpreted as an improvement. The reason is that the relative decrease in the test loss is greater than the relative increase in scene flow outliers. The constant input embedding dimension allows the decoder to relatively take attention more into account compared to its other inputs in higher pyramid levels. The accuracy of the estimated scene flow increases for higher levels. Thus also the projected optical flow and that of the warped features become more accurate. Consequently, the relevance of the attention increases for higher levels. To conclude, a more relevant attention for higher levels can be focused on more intensively in higher levels by the decoder compared to its other inputs. The percentage of scene flow outliers can be further decreased by 4.5 percentage points by increasing the window size from 13×4 to 26×8 because the association between more features can be determined in this way. Furthermore, performing

absolute positional embedding before feature embedding instead of relative positional embedding slightly further improves both KPIs of scene flow estimation since the decoder gets additional information about which features of the stacked warped and non-warped image belong to each other. Consequently, this information must not be learned by the neural network.
Finally, the transfer of the warped features to the decoder gave the scene flow estimator with an integrated Swin transformer a breakthrough decreasing SF by 10.9 percentage points from 60.1% to 50.6% and the test loss from 6.8 to 6.4. This change makes sense because the decoder not only has information about the features and their importance in relation to itself and the warped features but also the warped features directly. Consequently, the attention can be better classified by the decoder in the overall context. This measure produces a test loss being 0.4 greater, but a percentage of scene flow outliers being 4.2 percentage points smaller compared to the baseline.
As an alternative to stacking the features and warped features, the differences between the two can also be transferred to the Swin transformer. This creates scene flow outliers being greater by 3 percentage points. However, the test loss can be further reduced from 6.4 to 6.1 compared to the method with the stacking. In comparison to the baseline, this method produces a slightly higher test loss by about 0.1, but 1.2 percentage points less scene flow outliers. In summary, two similarly good scene flow estimation methods compared to our baseline were developed using a Swin transformer considering our KPIs. The runtime has increased from $0.04s$ to $0.10s$ by replacing the correlation with the Swin transformer, which is mainly due to the choice of the embedding dimension and the fact that the warped features also have to be processed in the decoder.
We showed that predicting the future scene flow works because the neural network does not simply adopt the scene flow estimation as a prediction. Additionally, the baseline which computes a correlation was extended to perform scene flow prediction. This neural network was compared with our neural network which uses a Swin transformer. Our neural network with integrated Swin transformer made slightly better predictions according to L_{fut} which dropped 7.13 to 6.96 and significantly better estimates according to L_{pres} which dropped from 6.78 to 6.35. These results emphasize the benefits of the Swin transformer.
Furthermore, considering the features of an additional past frame in the Swin transformer increases the percentage learned about the future relative to what is known about the present c_{learn} from 52.2% to 65.7%. The test loss $L_{pres} = 6.3$ is similar to the test loss of the baseline $L_{pres} = 6.2$.

With more time and computing power, the results obtained during this work could be surpassed by adapting the training and evaluation method in future work. For example, a larger test dataset than KITTI Scene Flow 2015 could be generated with longer image sequences instead of image pairs, which also includes real-world street scenes, ground truth depth, and ground truth optical flow. Additionally, all experiments could be repeated with the original resizing of the images to a size of 832×256 used in [1] instead of 416×128. Thereby, the accuracy of some scene flow estimators such as the one with the concatenated input frames or the one with the best Swin transformers could surpass the estimates of [1] in terms of the accuracy of the scene flow. Furthermore, the learning rate schedule which remained unchanged in this work, could be optimized for each neural network individually. The neural networks were always evaluated based on their parameters after 62 epochs of training. However, the training of the neural networks could also be run until the test loss increases significantly

and the parameters of the neural network with the smallest test loss of all epochs could be evaluated.
Future work can be devoted to fine-tuning hyperparameters such as the embedding dimension or the window size of the Swin transformer by testing more different intermediate values for each pyramid level individually. The investigations carried out in this work only concentrate on the rough optimization of the hyperparameters and on the influence of their scale on the resulting estimation because otherwise the capacities in terms of computing and time would be exceeded. When designing the Swin transformer, more adjustments can be done in comparison to the correlation so that there is still a lot of potential for further optimization of the Swin transformer.
The scene flow predictor could be extended to consider more than 3 past frames. A drawback of considering multiple past frames is that a lot of memory is required for the weights involved in the computation of the self-attention of the Swin transformer. This issue could be mitigated by using deformable attention modules. These modules compute the self-attention for a subgroup of pixels. These pixels are shifted by a learned offset from their original positions on the intersections of equidistant gridlines [68].
The scene flow predictor could also be extended predict scene flows further into the future. Instead, a network predicting a future scene flow $sf_{t+0\rightarrow\tau,l}$ with $\tau \in \{1, 2, 3...\}$ could be tested. The two most recent input frames should be $\mathbf{I}_{t-\tau,l}$ and $\mathbf{I}_{t,l}$ so that the time intervals between the inputs and the predictions are the same and that the scene flow predictor can learn how far into the future it should predict. In real-time applications such as autonomous driving, the scene flow could be repeatedly estimated at different times. The features and warped features that arose during the computation of past estimates could also be used as additional inputs of the Swin transformer for the current estimate.

A. Ablation study of original monocular scene flow estimation framework

In future applications, it is possible that a short runtime of the scene flow estimator is particularly important so that a reduction in the accuracy of the estimation can be accepted. Therefore, the omission of several parts of the monocular scene flow estimation framework of [1], the contribution of their individual components to the overall performance, and the runtime of the resulting scene flow estimator is examined.

First of all, the warping is omitted as a whole because it could lead to more direct and faster computation of the scene flow and allows to directly work on the original features in the decoder. However, the results denoted in table A.1 clearly reflect the importance of warping. The test and training losses as well as the scene flow outliers are significantly higher when the warping is left out. Moreover, the runtime does not change significantly.
The biggest difference between the scene flow estimator, which is used as a baseline, and other state-of-the-art estimators is the calculation of the correlation between the features and the warped features. Since other estimation methods also produce similarly good or better results when estimating the scene flow, it is reasonable to assume that the computation of the correlation is superfluous. Furthermore, it is possible that the correlation operation could also be learned and performed by the decoder. However, omitting the correlation worsens all KPIs. When leaving out the correlation, the optical flow is the only feedback of the last level. However, the optical flow has only two channels. These are significantly fewer channels than the channels of the other input of the decoder. Consequently, by omitting the correlation, too little attention is paid to the optical flow by the decoder. However, the main reason for keeping the correlation is to point out in which direction the estimated scene flow of the last level should be adjusted.
Finally, leaving out the context network, which is at the end of the decoder and post-processes the disparity and scene flow, could also decrease the computational effort. The test loss decreases to a greater extent than the percentage of scene flow outliers increases. Additionally, omitting the context network decreases the runtime by $0.2s$. Therefore, omitting the context network seems to be advantageous and can be left out in future work.

modification	Training		Test			Run-time in s
	masked pixels in %	L_{pres}	masked pixels in %	L_{pres}	SF in %	
/	21.1	5.6	24.6	6.0	**54.8**	0.04
no warping is performed	20.7	5.8	24.2	6.3	64.6	0.04
no correlation is computed	21.1	6.0	24.6	6.5	64.4	0.04
no context network exists	19.8	**5.2**	22.8	**5.5**	56.7	**0.02**

Table A.1.: Results of several modification of the baseline after 62 epochs

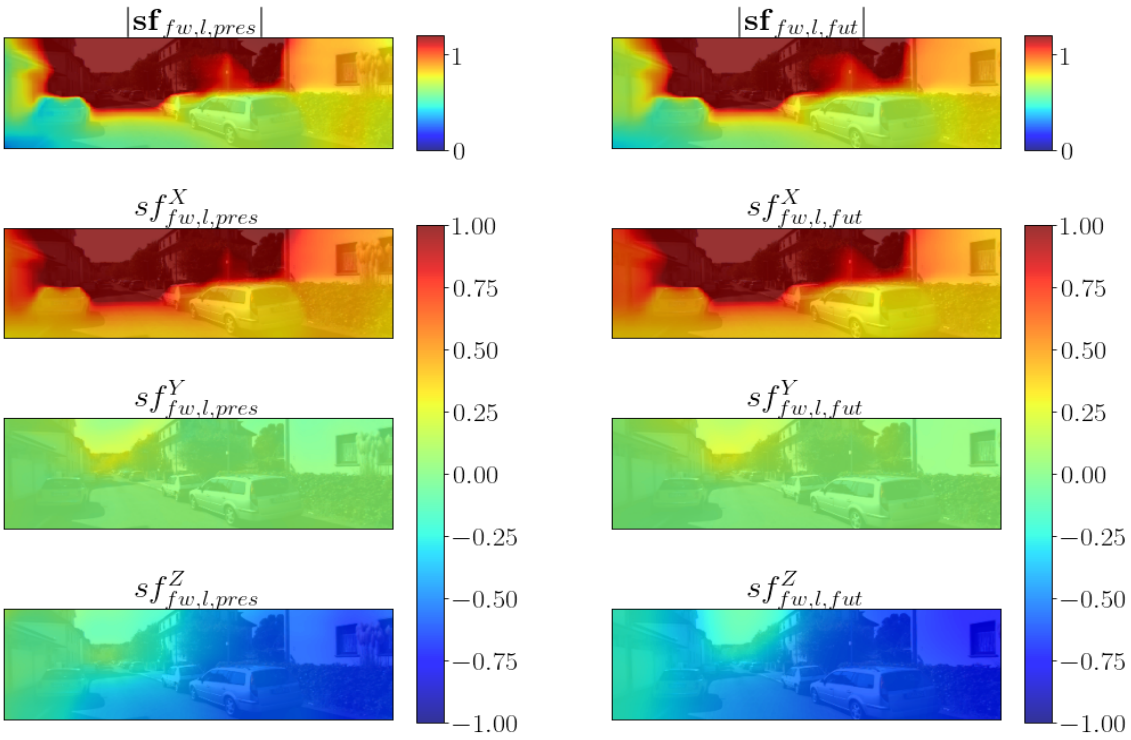
B. Plausibility check of scene flow estimates and predictions

In this chapter, the output of the scene flow estimator and predictor, which considers three subsequent frames as was presented in chapter 7, is checked for plausibility. This neural network uses a Swin transformer to process the three most recent frames whereby the warped image is directly passed to the decoder instead of being processed by the Swin transformer. The following driving situations are considered. In the first situation, the ego-vehicle is turning left. In a second scenario, another car is driving in a straight line towards the ego vehicle. The third traffic situation involves a car approaching from the right on a side street. The frames capturing these traffic situations can be seen in Fig B.1, Fig B.4 and Fig B.6 respectively. After visualizing the frames, the corresponding estimated and predicted forward scene flow between the frames is illustrated in Fig B.2, Fig B.5 and Fig B.7.

(a) $\mathbf{I}_{t-1,l}$ (b) $\mathbf{I}_{t,l}$ (c) $\mathbf{I}_{t+1,l}$

Figure B.1.: Images of the test dataset originating from the KITTI raw dataset in which the ego-vehicle turns left

In the first scenario, the ego-vehicle turns to the left. Thereby, the translational speed is low and rotation is mainly performed. Because there should not be a significant translational motion in the x-direction of the coordinate system of the ego-vehicle as shown in Fig. 4.1, the translational motion can be ignored. Due to the rotation of the ego vehicle around its y-axis, the coordinates of the scene points that are further away from the coordinate origin change more strongly. For this reason, the scene flow should be greater in the x-direction the further away it is from the ego-vehicle. This is the case when considering Fig. B.2.
The interpretation of the forward scene flow in the z-direction might be less intuitive. Therefore, the ego-vehicle is illustrated along with its coordinate system for two subsequent timestamps in Fig. B.3. The coordinate axes X' and Z' refer to the same axes as X and Z except that they correspond to the subsequent timestamp. The Y and Y' axes point downward and are not shown for the sake of simplicity Additionally, the distances between static objects and the origins of the coordinate systems in the z-direction are marked in Fig. B.3. Now, object 1 is considered. This object is located on the right side of the images of the two subsequent timestamps. It can be seen that the z-component of its distance from the origin of the coordinate system decreases as the ego-vehicle turns. A decrease in this distance results in a negative scene flow. This is the reason why the z-component of the scene flow is negative for the right half of the image. While the z-component of the distance decreases, the x-



(a) Scene flow between $\mathbf{M}_{t-1}$ and $\mathbf{M}_t$

(b) Scene flow between $\mathbf{M}_t$ and $\mathbf{M}_{t+1}$

Figure B.2.: Visualization of the magnitudes and components of the estimated and predicted forward scene flow of the left camera based on the frames visualized in Fig B.1

component increases. However, this effect is hardly visible when looking at the x-component of the scene flow. The reason is that this effect has less impact than the effect that was described at the beginning of the paragraph and that has an impact that scales with the depth. When the object is located on the left side of the image as it is the case for object 2, the z-component of the distance to the origin of the coordinate system barely changes. Consequently, the z-component of the scene flow is close to zero on the left part of the image. The left part of the road still has a negative scene flow component which is the result of the forward motion of the ego-vehicle. Overall, the predicted scene flow is similar to the estimated scene flow.

Now, the sequence depicted in Fig. B.4 is considered in which a car approaches the ego-vehicle on the opposite lane. We expect that the x- and y-component of the scene flow of the approaching car is equal to zero because the distance in the x- and y-direction of the approaching vehicle to the ego-vehicle stays constant. Only the z-component is expected to change since both vehicles move straight forward. However, the opposite is the case as can be seen in Fig. B.5. The x- and y-component of the scene flow correspond to what we would expect from the x- and y-component of the optical flow. Since, the pixel values of the approaching car move to the left side of the image, the x-component of the optical flow is negative and since the pixels also move downward, the y-component of the optical flow is negative. The fact that scene flow behaves like optic flow is an indicator that depth has not been learned well enough. This may be because the approaching car is too far away. Consequently, its disparity is small. Due to the low resolution, small disparities can only be

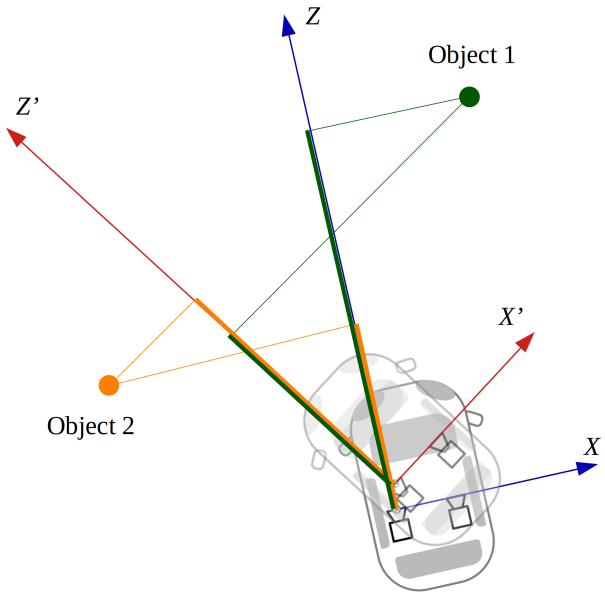
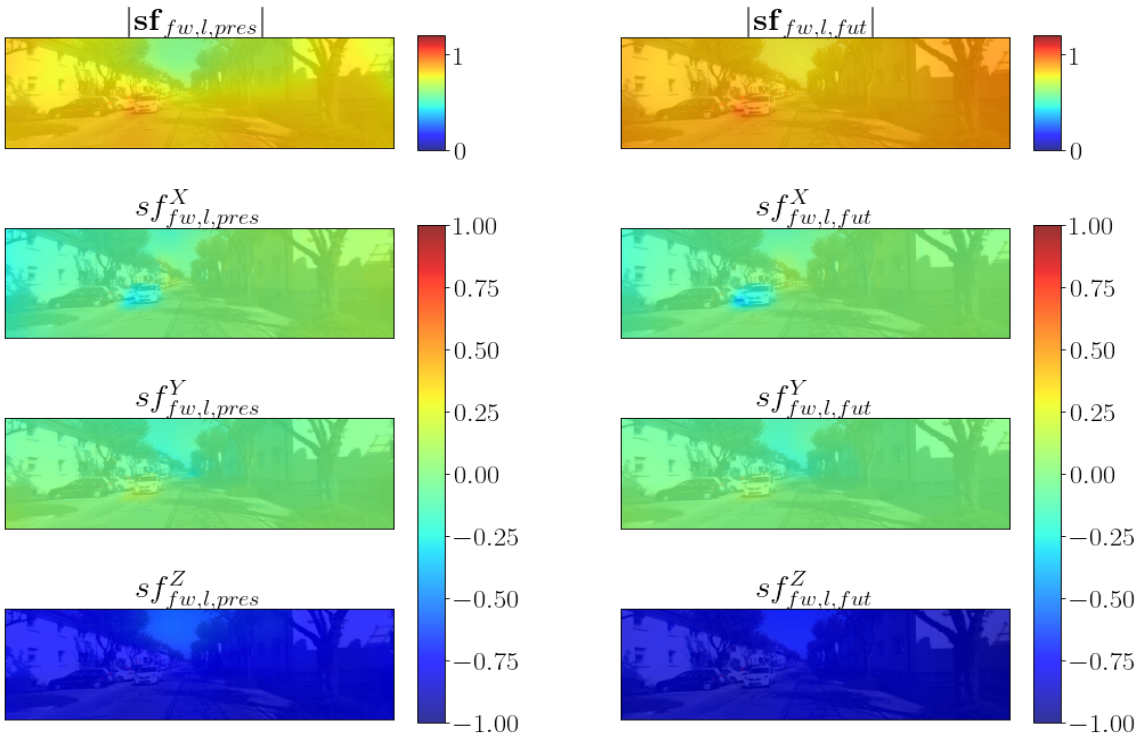


Figure B.3.: Ego-vehicle that turns to the left and distances to static objects relative to the coordinate system of the ego-vehicle

determined imprecisely. Since the depth is computed based on the disparity, the depth is also imprecise. This issue could be mitigated by using the original resizing of the images as it was done in the baseline. The tendency of the increased velocity between the frames of Fig. B.4 from $6.8\frac{m}{s}$ to $7.0\frac{m}{s}$ and $7.2\frac{m}{s}$ is predicted correctly.

(a) $\mathbf{I}_{t-1,l}$ (b) $\mathbf{I}_{t,l}$ (c) $\mathbf{I}_{t+1,l}$

Figure B.4.: Images of the test dataset originating from the KITTI raw dataset in which another car drives in a straight line towards the ego vehicle



(a) Scene flow between $\mathbf{M}_{t-1}$ and $\mathbf{M}_t$

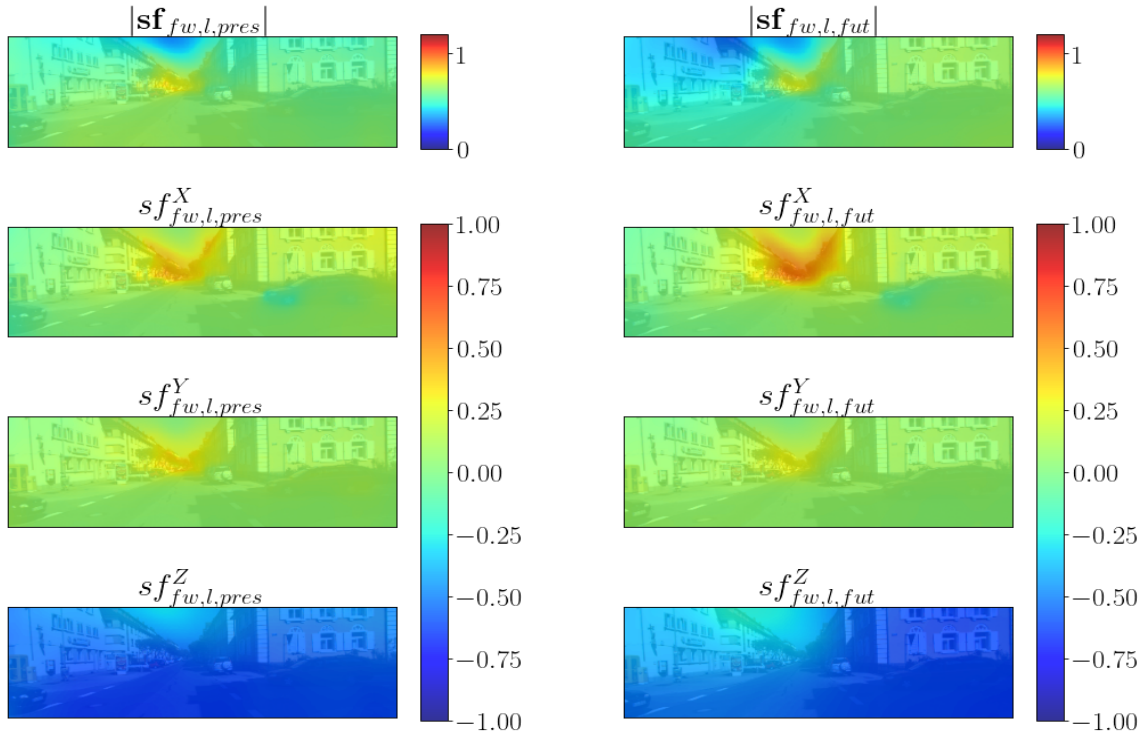
(b) Scene flow between $\mathbf{M}_t$ and $\mathbf{M}_{t+1}$

Figure B.5.: Visualization of the magnitudes and components of the estimated and predicted forward scene flow of the left camera based on the frames visualized in Fig B.4

Finally, a situation of a car approaching an intersection from the right side as depicted in Fig. B.6 is considered. A motion in negative x-direction is correctly estimated by the scene flow as can be seen in Fig. B.7. However, it can be seen again that the neural network has difficulties in predicting scene flows for distant points. For example, excessively large scene flows are predicted for the points on the horizon. Apart from the fact that the predicted scene flow in the z-direction of the scene points on the left side of the images is slightly smaller than on the right side, the prediction also makes sense.

(a) $\mathbf{I}_{t-1,l}$ (b) $\mathbf{I}_{t,l}$ (c) $\mathbf{I}_{t+1,l}$

Figure B.6.: Images of the test dataset originating from the KITTI raw dataset in which another car approaches the ego-vehicle from the right on a side street



(a) Scene flow between $\mathbf{M}_{t-1}$ and $\mathbf{M}_t$

(b) Scene flow between $\mathbf{M}_t$ and $\mathbf{M}_{t+1}$

Figure B.7.: Visualization of the magnitudes and components of the estimated and predicted forward scene flow of the left camera based on the frames visualized in Fig B.6

List of Figures

List of Tables

Bibliography

[1] J. Hur and S. Roth, "Self-supervised monocular scene flow estimation," 2020.

[2] (2022) Ridehailing taxi. [Online]. Available: https://de.statista.com/outlook/mmo/shared-mobility/gemeinsame-fahrten/ridehailing-taxi/weltweit

[3] (2020) Freight transport statistics - modal split. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Freight_transport_statistics_-_modal_split

[4] T. Bosse. (2021) Wie sich der transportmarkt in europa bis 2025 entwickelt. [Online]. Available: https://www.dvz.de/rubriken/land/strasse/detail/news/deutschland-bleibt-nummer-1.html

[5] B. Bulletin. (2022) Truck driver shortage intensifies supply chain issues. [Online]. Available: https://insurancenewsnet.com/oarticle/truck-driver-shortage-intensifies-supply-chain-issues

[6] Z.-Q. Zhao, P. Zheng, S.-t. Xu and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[7] S. Vedula, S. Baker, P. Rander, R. Collins and T. Kanade, "Three-dimensional scene flow," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 722–729 vol.2.

[8] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[9] A. Meyer and R. De Charette, "Computing ego velocity from scene flow estimation," Ph.D. dissertation, Inria Paris, 2016.

[10] M. Zhai, X. Xiang, N. Lv and X. Kong, "Optical flow and scene flow estimation: A survey," *Pattern Recognition*, vol. 114, p. 107861, 2021.

[11] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan and L. Sun, "Transformers in time series: A survey," *arXiv preprint arXiv:2202.07125*, 2022.

[12] C. S. R. U. Andreas Geiger, Philip Lenz. (2022) Scene flow evaluation 2015. [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php

[13] F. Brickwedde, S. Abraham and R. Mester, "Mono-sf: Multi-view geometry meets single-view depth for monocular scene flow estimation of dynamic traffic scenes," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2780–2790.

[14] R. Schuster, D. Stricker and C. Unger, "Monocomb: A sparse-to-dense combination approach for monocular scene flow," in *Computer Science in Cars Symposium*, 2020, pp. 1–8.

[15] J. Hur and S. Roth, "Self-supervised multi-frame monocular scene flow," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2684–2694.

[16] Z. Ren, D. Sun, J. Kautz and E. Sudderth, "Cascaded scene flow prediction using semantic segmentation," in *2017 International Conference on 3D Vision (3DV)*. IEEE, 2017, pp. 225–233.

[17] A. Wedel, T. Brox, T. Vaudrey, C. Rabe, U. Franke and D. Cremers, "Stereoscopic scene flow computation for 3d motion understanding," *International Journal of Computer Vision*, vol. 95, no. 1, pp. 29–51, 2011.

[18] F. Huguet and F. Devernay, "A variational method for scene flow estimation from stereo sequences," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–7.

[19] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger and C. Schnörr, "Real-time optic flow computation with variational methods," in *International Conference on Computer Analysis of Images and Patterns*. Springer, 2003, pp. 222–229.

[20] T.-W. Hui, X. Tang and C. C. Loy, "A lightweight optical flow cnn—revisiting data fidelity and regularization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 8, pp. 2555–2569, 2020.

[21] Z. Tu, W. Xie, D. Zhang, R. Poppe, R. C. Veltkamp, B. Li and J. Yuan, "A survey of variational and cnn-based optical flow techniques," *Signal Processing: Image Communication*, vol. 72, pp. 9–24, 2019.

[22] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[23] D. J. Livingstone, D. T. Manallack and I. V. Tetko, "Data modelling with neural networks: advantages and limitations," *Journal of computer-aided molecular design*, vol. 11, no. 2, pp. 135–142, 1997.

[24] Z. Lu, H. Pu, F. Wang, Z. Hu and L. Wang, "The expressive power of neural networks: A view from the width," *Advances in neural information processing systems*, vol. 30, 2017.

[25] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/089360809190009T

[26] B. Widrow and M. Lehr, "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, pp. 1415 – 1442, 10 1990.

[27] X. Zhang, Y. Zou and W. Shi, "Dilated convolution neural network with leakyrelu for environmental sound classification," in *2017 22nd International Conference on Digital Signal Processing (DSP)*. IEEE, 2017, pp. 1–5.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[29] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[30] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[31] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[32] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[33] M. Volpi and D. Tuia, "Dense semantic labeling of subdecimeter resolution images with convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 881–893, 2016.

[34] A. Araujo, W. D. Norris and J. Sim, "Computing receptive fields of convolutional neural networks," *Distill*, 2019.

[35] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2016.

[36] "Conv2d," https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html, accessed: 2022-03-11.

[37] Y. Yang, W. Zhang, J. Wu, W. Zhao and A. Chen, "Deconvolution-and-convolution networks," *arXiv preprint arXiv:2103.11887*, 2021.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[39] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[40] Z. Niu, G. Zhong and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021.

[41] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows. arxiv 2021," *arXiv preprint arXiv:2103.14030*.

[42] Z. Yan and X. Xiang, "Scene flow estimation: A survey," *arXiv preprint arXiv:1612.02590*, 2016.

[43] R. Arnheim, "The perception of the visual world by james j. gibson," *The Journal of Aesthetics and Art Criticism*, vol. 11, no. 2, pp. 172–173, 1952.

[44] S. Vedula, S. Baker, P. Rander, R. Collins and T. Kanade, "Three-dimensional scene flow," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2. IEEE, 1999, pp. 722–729.

[45] R. Jain, R. Kasturi and B. Schunck, *Machine Vision*, 01 1995.

[46] F. Arif and M. Akbar, "Resampling air borne sensed data using bilinear interpolation algorithm," in *IEEE International Conference on Mechatronics, 2005. ICM'05.* IEEE, 2005, pp. 62–65.

[47] Z. Zhang, *Perspective Camera*. Boston, MA: Springer US, 2014, pp. 590–592. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6_114

[48] Z. Zhang, *Camera Parameters (Intrinsic, Extrinsic)*. Boston, MA: Springer US, 2014, pp. 81–85. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6_152

[49] C. Godard, O. Mac Aodha and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 270–279.

[50] H. Zhao, O. Gallo, I. Frosio and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on computational imaging*, vol. 3, no. 1, pp. 47–57, 2016.

[51] Z. Wang, E. P. Simoncelli and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2. Ieee, 2003, pp. 1398–1402.

[52] R. Mahjourian, M. Wicke and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5667–5675.

[53] V. Guizilini, K.-H. Lee, R. Ambrus and A. Gaidon, "Learning optical flow, depth, and scene flow without real-world labels," *IEEE Robotics and Automation Letters*, 2022.

[54] W. Wu, Z. Wang, Z. Li, W. Liu and L. Fuxin, "Pointpwc-net: A coarse-to-fine network for supervised and self-supervised scene flow estimation on 3d point clouds," *arXiv preprint arXiv:1911.12408*, 2019.

[55] Y. Almalioglu, M. R. U. Saputra, P. P. de Gusmao, A. Markham and N. Trigoni, "Ganvo: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 5474–5480.

[56] L. Liu, G. Zhai, W. Ye and Y. Liu, "Unsupervised learning of scene flow estimation fusing with local rigidity." in *IJCAI*, 2019, pp. 876–882.

[57] J. Hur and S. Roth, "Mirrorflow: Exploiting symmetries in joint optical flow and occlusion estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 312–321.

[58] Y. Wang, Y. Yang, Z. Yang, L. Zhao, P. Wang and W. Xu, "Occlusion aware unsupervised learning of optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4884–4893.

[59] S. Meister, J. Hur and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," 11 2017.

[60] R. Saxena, R. Schuster, O. Wasenmuller and D. Stricker, "Pwoc-3d: Deep occlusion-aware end-to-end scene flow estimation," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 324–331.

[61] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4040–4048.

[62] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[63] M. Menze, C. Heipke and A. Geiger, "Object scene flow," *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[64] D. Sun, X. Yang, M.-Y. Liu and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8934–8943.

[65] H. Yong, J. Huang, D. Meng, X. Hua and L. Zhang, "Momentum batch normalization for deep learning with small batch size," in *European Conference on Computer Vision*. Springer, 2020, pp. 224–240.

[66] R. Ge, F. Huang, C. Jin and Y. Yuan, "Escaping from saddle points—online stochastic gradient for tensor decomposition," in *Conference on learning theory*. PMLR, 2015, pp. 797–842.

[67] H. Lin, X. Cheng, X. Wu, F. Yang, D. Shen, Z. Wang, Q. Song and W. Yuan, "Cat: Cross attention in vision transformer," *arXiv preprint arXiv:2106.05786*, 2021.

[68] Z. Xia, X. Pan, S. Song, L. E. Li and G. Huang, "Vision transformer with deformable attention," *arXiv preprint arXiv:2201.00520*, 2022.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 31.05.2022 Nicolas Riess