Motion R-CNN: Instance-level 3D Motion Estimation with Region-based CNNs

Motion R-CNN: 3D-Bewegungsschätzung auf Instanzebene mit Regionsbasierten CNNs
Bachelor-Thesis von Simon Meister aus Erbach
Tag der Einreichung:

1. Gutachten: Prof. Stefan Roth, Ph.D.
2. Gutachten: M.Sc. Junhwa Hur

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Visual Inference Group
Fachbereich Informatik

Motion R-CNN: Instance-level 3D Motion Estimation with Region-based CNNs
Motion R-CNN: 3D-Bewegungsschätzung auf Instanzebene mit Regionsbasierten CNNs

Vorgelegte Bachelor-Thesis von Simon Meister aus Erbach

1. Gutachten: Prof. Stefan Roth, Ph.D.
2. Gutachten: M.Sc. Junhwa Hur

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 22.11.2017

(Simon Meister)

Abstract

With the advent of deep learning, it has become popular to re-purpose generic deep networks for classical computer vision problems involving pixel-wise estimation.
Following this trend, many recent end-to-end deep learning approaches to optical flow and scene flow predict complete, high resolution flow fields with a generic network for dense, pixel-wise prediction, thereby ignoring the inherent structure of the underlying motion estimation problem and any physical constraints within the scene.
We introduce a scalable end-to-end deep learning approach for dense motion estimation that respects the structure of the scene as being composed of distinct objects, thus combining the representation learning benefits and speed of end-to-end deep networks with a physically plausible scene model inspired by slanted plane energy-minimization approaches to scene flow.
Building on recent advances in region-based convolutional neural networks (R-CNNs), we integrate motion estimation with instance segmentation. Given two consecutive frames from a monocular RGB-D camera, our resulting end-to-end deep network detects objects with precise per-pixel object masks and estimates the 3D motion of each detected object between the frames. Additionally, we estimate the camera ego-motion in the same network, and compose a dense optical flow field based on instance-level and global motion predictions. We train our network on the synthetic Virtual KITTI dataset, which provides ground truth for all components of our system.

Zusammenfassung

Mit dem Aufkommen von Deep Learning ist das Umfunktionieren generischer Deep Networks ein beliebter Ansatz für klassische Probleme der Computer Vision geworden, die pixelweise Schätzung erfordern.
Viele aktuelle end-to-end Deep Learning Methoden für optischen Fluss oder Szenenfluss folgen diesem Trend und berechnen vollständige und hochauflösende Flussfelder mit generischen Netzwerken für dichte, pixelweise Schätzung, und ignorieren damit die inhärente Struktur des zugrundeliegenden Bewegungschätzungsproblems und jegliche physikalische Randbedingungen innerhalb der Szene.
Wir stellen ein skalierbares end-to-end Deep Learning Verfahren für dichte Bewegungschätzung vor, das die Struktur einer Szene als Zusammensetzung eigenständiger Objekte respektiert, und kombinieren damit die Repräsentationskraft und Geschwindigkeit von end-to-end Deep Networks mit einem physikalisch plausiblen Szenenmodell, das von slanted-plane Energieminimierungsmethoden für Szenenfluss inspiriert ist.
Hierbei bauen wir auf den aktuellen Fortschritten bei regionsbasierten Convolutional Neural Networks (R-CNNs) auf und integrieren Bewegungsschätzung mit Instanzsegmentierung. Bei Eingabe von zwei aufeinanderfolgenden Frames aus einer monokularen RGB-D Kamera erkennt unser end-to-end Deep Network Objekte mit pixelgenauen Objektmasken und schätzt die 3D-Bewegung jedes erkannten Objekts zwischen den Frames ab. Zusätzlich schätzen wir im selben Netzwerk die Eigenbewegung der Kamera, und setzen aus den instanzbasierten und globalen Bewegungsschätzungen ein dichtes optisches Flussfeld zusammen. Wir trainieren unser Netzwerk auf dem synthetischen Virtual KITTI Datensatz, der Ground Truth für alle Komponenten unseres Systems bereitstellt.

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 Technical goals . . . 2
- 1.3 Related work . . . 4
- 1.4 Outline . . . 6

2 Background **8**
- 2.1 Optical flow and scene flow . . . 8
- 2.2 CNNs for dense motion estimation . . . 8
- 2.3 SfM-Net . . . 9
- 2.4 ResNet . . . 10
- 2.5 Region-based CNNs . . . 10
- 2.6 Mask R-CNN: Training and inference . . . 14

3 Motion R-CNN **20**
- 3.1 Motion R-CNN . . . 20
- 3.2 Network design . . . 23
- 3.3 Supervision . . . 23
- 3.4 Training and inference . . . 26
- 3.5 Dense flow from 3D motion . . . 26

4 Experiments **28**
- 4.1 Implementation . . . 28
- 4.2 Datasets . . . 28
- 4.3 Virtual KITTI: Training setup . . . 31
- 4.4 Virtual KITTI: Evaluation . . . 31

5 Conclusion **35**
- 5.1 Summary . . . 35
- 5.2 Future Work . . . 35

Bibliography **39**

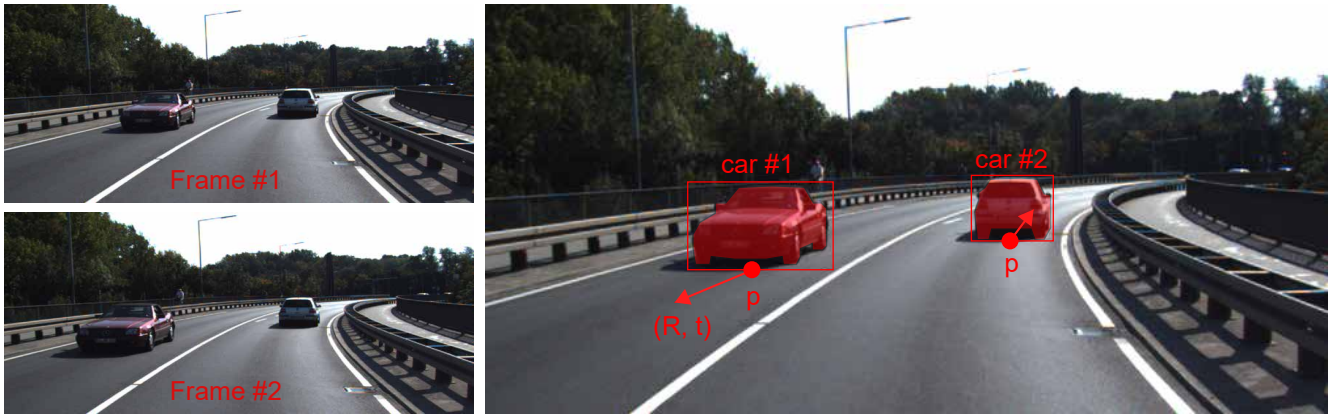

Figure 1: Given two temporally consecutive frames, our network segments the pixels of the first frame into individual objects and estimates their 3D locations as well as all 3D object motions between the frames.

1 Introduction

1.1 Motivation

For moving in the real world, it is often desirable to know which objects exists in the proximity of the moving agent, where they are located relative to the agent, and where they will be at some point in the near future. In many cases, it would be preferable to infer such information from video data, if technically feasible, as camera sensors are cheap and ubiquitous (compared to, for example, Lidar).

As an example, consider the autonomous driving problem. Here, it is crucial to not only know the position of each obstacle, but to also know if and where the obstacle is moving, and to use sensors that will not make the system too expensive for widespread use. At the same time, the autonomous driving system has to operate in real time to react quickly enough for safely controlling the vehicle.

A promising approach for 3D scene understanding in situations such as autonomous driving are deep neural networks, which have recently achieved breakthroughs in object detection, instance segmentation, and classification in still images, and are more and more often being applied to video data. A key benefit of deep networks is that they can, in principle, enable very fast inference on real time video data and generalize over many training situations to resolve ambiguities inherent in image understanding and motion estimation.

Thus, in this work, we aim to develop deep neural networks which can, given sequences of images, segment the image pixels into object instances, and estimate the location and 3D motion of each object instance relative to the camera (Figure 1).

1.2 Technical goals

Recently, SfM-Net [41] introduced an end-to-end deep learning approach for predicting dense depth and dense optical flow from monocular image sequences, based on estimating the 3D motion of individual objects and the camera. Using a standard encoder-decoder network for pixel-wise dense prediction, SfM-Net predicts a pre-determined number of binary masks ranging over the complete image, with each mask specifying the membership of the image pixels to one object. A fully-connected network branching off

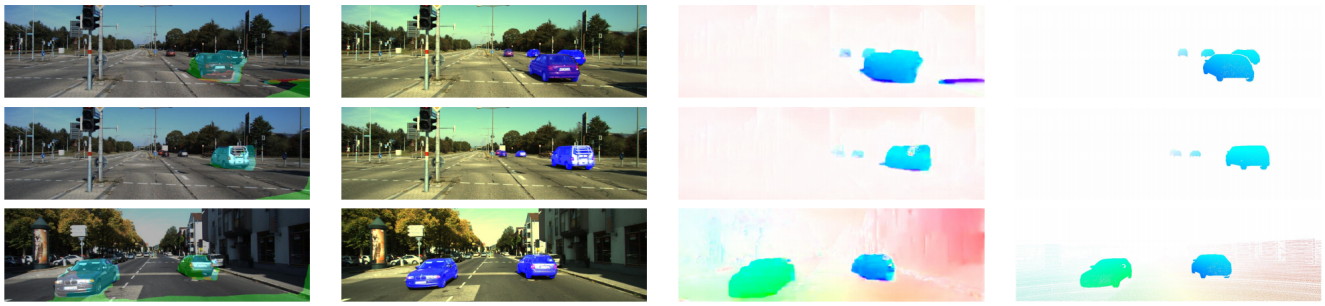

Figure 2: Results of SfM-Net [41] on KITTI [16]. From left to right, we show their instance segmentation into up to 3 independent objects, ground truth instance masks for the segmented objects, composed optical flow, and ground truth optical flow. Figure taken from [41].

the encoder then predicts a 3D motion for each object and the camera ego-motion. However, due to the fixed number of objects masks, the system can in practice only predict a small number of motions, and often fails to properly segment the pixels into the correct masks or assigns background pixels to object motions (Figure 2). Thus, due to the inflexible nature of their instance segmentation technique, their approach is very unlikely to scale to dynamic scenes with a potentially large number of diverse objects.

Still, we think that the general idea of estimating object-level motion with end-to-end deep networks instead of directly predicting a dense flow field, as is common in current end-to-end deep learning approaches to motion estimation, may significantly benefit motion estimation by structuring the problem, creating physical constraints, and reducing the dimensionality of the estimate.

In the context of still images, a scalable approach to instance segmentation based on region-based convolutional networks was recently introduced with Mask R-CNN [34]. Mask R-CNN inherits the ability to detect a large number of objects from a large number of classes at once from Faster R-CNN [17] and predicts pixel-precise segmentation masks for each detected object (Figure 3).

Inspired by the accurate segmentation results of Mask R-CNN, we thus propose *Motion R-CNN*, which combines the scalable instance segmentation capabilities of Mask R-CNN with the end-to-end instance-level 3D motion estimation approach introduced with SfM-Net. For this, we naturally integrate 3D motion prediction for individual objects into the per-RoI Mask R-CNN head in parallel to classification, bounding box refinement and mask prediction. For each RoI, we predict a single 3D rigid object motion together with the object pivot in camera space in this way. As a foundation for image matching, we extend the ResNet [24] backbone of Mask R-CNN to take two concatenated images as input, similar to FlowNetS [10]. This results in a fully integrated end-to-end network architecture for segmenting pixels into instances and estimating the motion of all detected instances without any limitations as to the number or variety of object instances (Figure 4).

Eventually, we want to extend our method to include depth prediction, yielding the first end-to-end deep network to perform 3D scene flow estimation in a principled and scalable way from the consideration of individual objects. For now, we will assume that RGB-D frames are given to break down the problem into manageable pieces.

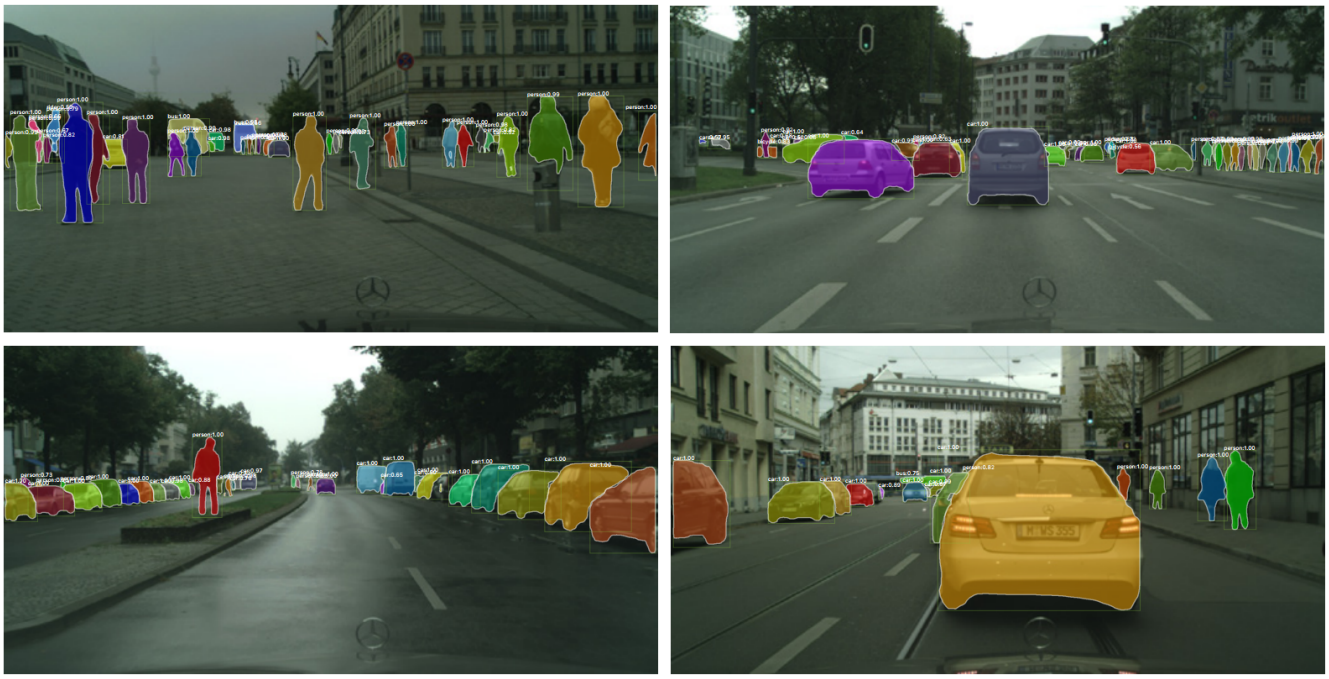

Figure 3: Instance segmentation results of Mask R-CNN ResNet-50-FPN [34] on Cityscapes [22]. Figure taken from [34].

1.3 Related work

In the following, we will refer to systems which use deep networks for all optimization and do not perform time-critical side computation at inference time (e.g. numerical optimization) as *end-to-end* deep learning systems.

Deep networks in optical flow estimation

End-to-end deep networks for optical flow were recently introduced based on encoder-decoder networks or CNN pyramids [10, 36, 40], which pose optical flow as generic (and homogeneous) pixel-wise estimation problem without making any assumptions about the regularity and structure of the estimated flow. Specifically, such methods ignore that the optical flow varies across an image depending on the semantics of each region or pixel, which include whether a pixel belongs to the background, to which object instance it belongs if it is not background, and the class of the object it belongs to. Often, failure cases of these methods include motion boundaries or regions with little texture, where semantics become very important. Extensions of these approaches to scene flow estimate dense flow and dense depth with similarly generic networks [27] and similar limitations.

Other works make use of semantic segmentation to structure the optical flow estimation problem and introduce reasoning at the object level [21, 25, 28, 42], but still require expensive energy minimization for each new input, as CNNs are only used for some of the components and numerical optimization is central to their inference.

In contrast, we tackle motion estimation at the instance-level with end-to-end deep networks and derive optical flow from the individual object motions.

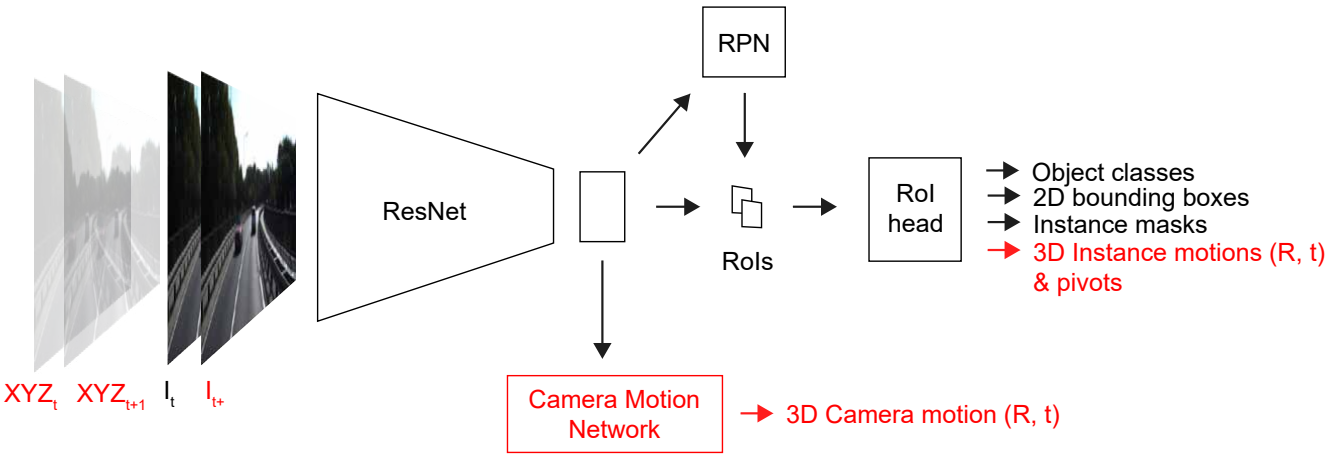

Figure 4: Overview of our network based on Mask R-CNN [34]. For each region of interest (RoI), we predict the 3D instance motion in parallel to the class, bounding box and mask. Additionally, we branch off a small network from the bottleneck for predicting the 3D camera ego-motion. Novel components in addition to Mask R-CNN are shown in red.

Slanted plane methods for 3D scene flow

The slanted plane model for scene flow [6, 20] models a 3D scene as being composed of planar segments. Pixels are assigned to one of the planar segments, each of which undergoes a independent 3D rigid motion. This model simplifies the motion estimation problem significantly by reducing the dimensionality of the estimate, and thus can lead to more accurate results than the direct estimation of a homogenous motion field. In contrast to [6, 20], the Object Scene Flow method [16] assigns each slanted plane to one rigidly moving object instance, thus reducing the number of independently moving segments by allowing multiple segments to share the motion of the object they belong to. In all of these methods, pixel assignment and motion estimation are formulated as energy-minimization problem which is optimized for each input data point, without the use of (deep) learning.

In a more recent approach termed Instance Scene Flow [31], a CNN is used to compute 2D bounding boxes and instance masks for all objects in the scene, which are then combined with depth obtained from a non-learned stereo algorithm to be used as pre-computed inputs to a slanted plane scene flow model based on [16]. Most likely due to their use of deep learning for instance segmentation and for some other components, this approach outperforms the previous related scene flow methods on relevant public benchmarks [4, 16]. Still, the method uses a energy-minimization formulation for the scene flow estimation itself and takes minutes to make a prediction.

Interestingly, the slanted plane methods achieve the current state-of-the-art in scene flow *and* optical flow estimation on the challenging KITTI benchmarks [4, 16], outperforming end-to-end deep networks like [27, 10, 36]. However, the end-to-end deep networks are significantly faster than their energy-minimization counterparts, generally taking a fraction of a second instead of minutes for prediction and can often be made to run in realtime. These concerns restrict the applicability of the current slanted plane models in practical settings, which often require estimations to be done in real time (or close to real time) and for which an end-to-end approach based on learning would be preferable.

Also, by analogy, in other contexts, the move towards end-to-end deep learning has often lead to significant benefits in terms of accuracy and speed. As an example, consider the evolution of region-

based convolutional networks, which started out as prohibitively slow with a CNN as a single component and became very fast and much more accurate over the course of their development into end-to-end deep networks.

Thus, in the context of motion estimation, one may expect end-to-end deep learning to not only bring large improvements in speed, but also in accuracy, especially considering the inherent ambiguity of motion estimation, and the ability of deep networks to learn to handle ambiguity from a large variety of training examples.

However, we think that the current end-to-end deep learning approaches to motion estimation are likely limited by a lack of spatial structure and regularity in their estimates as explained above, which stems from the generic nature of the employed networks. To this end, we aim to combine the modelling benefits of rigid scene decompositions with the promise of end-to-end deep learning.

End-to-end deep networks for 3D rigid motion estimation

End-to-end deep learning for predicting rigid 3D object motions was first introduced with SE3-Nets [32], which take raw 3D point clouds as input and produce a segmentation of the points into objects together with 3D motions for each object. Bringing this idea to the context of image sequences, SfM-Net [41] takes two consecutive frames and estimates a segmentation of pixels into objects together with their 3D motions between the frames. In addition, SfM-Net predicts dense depth and camera ego-motion to obtain full 3D scene flow with end-to-end deep learning. For supervision, SfM-Net penalizes the dense optical flow composed from all 3D motions and the depth estimate with a brightness constancy proxy loss.

Like SfM-Net, we aim to estimate 3D motion and instance segmentation jointly with end-to-end deep learning. Unlike SfM-Net, we build on a scalable object detection and instance segmentation approach with R-CNNs, which provide us with a strong baseline for these tasks.

End-to-end deep networks for camera pose estimation

Deep networks have been used for estimating the 6-DOF camera pose from a single RGB frame [15, 37], or for estimating depth and camera ego-motion from monocular video [30]. These works are related to ours in that we also need to output various rotations and translations from a deep network, and thus need to solve similar regression problems, and may be able to use similar parametrizations and losses.

1.4 Outline

First, in section 2, we introduce preliminaries and building blocks from earlier works that serve as a foundation for our networks and losses. Most importantly, we re-view the ResNet CNN (2.4) that will serve as CNN backbone as well as the developments in region-based CNNs which we build on (2.5), specifically Mask R-CNN and the Feature Pyramid Network (FPN) [39]. In section 3, we describe our technical contribution, starting with our motion estimation model and modifications to the Mask R-CNN backbone and head networks (3.1), followed by our losses and supervision methods for training the extended region-based CNN (3.3), and finally the postprocessings we use to derive dense flow from our 3D motion estimates (3.5). Then, in section 4, we introduce the Virtual KITTI dataset we use for training our networks as well as all preprocessings we perform (4.2), give details of our experimental setup (4.3),

and finally describe the experimental results on Virtual KITTI (4.4). Finally, in section 5, we summarize our work and describe future developments, including depth prediction, training on real world data, and exploiting frames over longer time intervals.

2 Background

In this section, we will give a more detailed description of previous works we directly build on and other prerequisites.

2.1 Optical flow and scene flow

Let $I_t, I_{t+1} : P \to \mathbb{R}^3$ be two temporally consecutive frames from a sequence of images. The optical flow $\mathbf{w} = (u, v)^T$ from I_t to I_{t+1} maps pixel coordinates in the first frame I_t to pixel coordinates of the visually corresponding pixel in the second frame I_{t+1}, and can be interpreted as the (apparent) movement of brightness patterns between the two frames. Optical flow can be regarded as two-dimensional motion estimation.

Scene flow is the generalization of optical flow to three-dimensional space and additionally requires estimating depth for each pixel. Generally, stereo input is used for scene flow to estimate disparity-based depth, however monocular depth estimation with deep networks is also becoming popular [26, 30]. In this preliminary work, we will assume per-pixel depth to be given.

2.2 CNNs for dense motion estimation

Output	Layer Operations	Output Dimensions
	input images I_t and I_{t+1}	H × W × 6
	Encoder	
	7 × 7 conv, 64, stride 2	$\frac{1}{2}$ H × $\frac{1}{2}$ W × 64
	5 × 5 conv, 128, stride 2	$\frac{1}{4}$ H × $\frac{1}{4}$ W × 128
	5 × 5 conv, 256, stride 2	$\frac{1}{8}$ H × $\frac{1}{8}$ W × 256
	3 × 3 conv, 256	$\frac{1}{8}$ H × $\frac{1}{8}$ W × 256
	3 × 3 conv, 512, stride 2	$\frac{1}{16}$ H × $\frac{1}{16}$ W × 512
	3 × 3 conv, 512	$\frac{1}{16}$ H × $\frac{1}{16}$ W × 512
	3 × 3 conv, 512, stride 2	$\frac{1}{32}$ H × $\frac{1}{32}$ W × 512
	3 × 3 conv, 512,	$\frac{1}{32}$ H × $\frac{1}{32}$ W × 512
	3 × 3 conv, 1024, stride 2	$\frac{1}{64}$ H × $\frac{1}{64}$ W × 1024
	Refinement	
	5 × 5 deconv, 512, stride 2	$\frac{1}{32}$ H × $\frac{1}{32}$ W × 512
	...	
flow	× 2 bilinear upsample	H × W × 2

Table 1: Overview of the FlowNetS [10] architecture. Transpose convolutions (deconvolutions) are used for refinement.

Deep convolutional neural network (CNN) architectures [5, 19, 24] became widely popular through numerous successes in classification and recognition tasks. The general structure of a CNN consists of a convolutional encoder, which learns a spatially compressed, wide (in the number of channels) representation of the input image, and a fully-connected prediction network on top of the encoder.

The compressed representations learned by CNNs of these categories do not, however, allow for prediction of high-resolution output, as spatial detail is lost through sequential application of pooling or strides. Thus, networks for dense, high-resolution, prediction introduce a convolutional decoder on top of the representation encoder, performing upsampling of the compressed features and resulting in a encoder-decoder pyramid. In most cases, skip connections from the encoder part are used to combine high-resolution detail with abstract, expressive features coming from the bottleneck (the last layer of the encoder). The most popular deep networks of this kind for end-to-end optical flow prediction are variants of the FlowNet family [10, 36], which was recently extended to scene flow estimation [27]. Table 1 gives an overview of the classical FlowNetS architecture for optical flow prediction.

Note that the network itself is a rather generic autoencoder and is specialized for optical flow only through being trained with supervision from dense optical flow ground truth. Potentially, the same network could also be used for semantic segmentation if the number of final and intermediate output channels was adapted from two to the number of classes. Still, FlowNetS demonstrates that a generic deep encoder-decoder CNN can learn to perform image matching arguably well, given just two consecutive frames as input and a large enough receptive field at the outputs to cover the displacements. Note that the maximum displacement that can be correctly estimated depends on the number of strided 2D convolutions (and the stride they use) and pooling operations in the encoder. Recently, other, similarly generic, encoder-decoder CNNs have been applied to optical flow prediction as well [43].

2.3 SfM-Net

Table 2 shows the SfM-Net architecture [41] we described in the introduction. Motions and full-image masks for a fixed number $N_{motions}$ of independent objects are predicted in addition to a depth map, and a unsupervised re-projection loss based on image brightness differences penalizes the predictions.

Output	Layer Operations	Output Dimensions
	Conv-Deconv	
	Motion Network	
	input images I_t and I_{t+1}	H × W × 6
	Conv-Deconv	H × W × 32
masks	1 ×1 conv, $N_{motions}$	H × W × $N_{motions}$
FC	From bottleneck: $\left[\text{fully connected}, 512\right] \times 2$	1 × 512
object motions	fully connected, $N_{motions} \cdot 9$	H × W × $N_{motions} \cdot 9$
camera motion	From FC: × 2	H × W × 6
	Structure Network	
	input image I_t	H × W × 3
	Conv-Deconv	H × W × 32
depth	1 ×1 conv, 1	H × W × 1

Table 2: SfM-Net [41] architecture. Here, Conv-Deconv is a simple fully-convolutional encoder-decoder network, where convolutions and deconvolutions with stride 2 are used for downsampling and upsampling, respectively. The stride at the bottleneck with respect to the input image is 32. The Conv-Deconv weights for the structure and motion networks are not shared, and $N_{motions} = 3$.

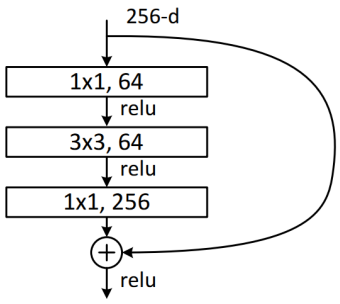

Figure 5: ResNet [24] "bottleneck" convolutional block introduced to reduce computational complexity in deeper network variants, shown here with 256 input and output channels. Figure taken from [24].

2.4 ResNet

ResNet (Residual Network) [24] was initially introduced as a CNN for image classification, but became popular as basic building block of many deep network architectures for a variety of different tasks. Figure 5 shows the fundamental building block of ResNet. The additive *residual unit* enables the training of very deep networks without the gradients becoming too small as the distance from the output layer increases.

In Table 3, we show the ResNet variant that will serve as the basic CNN backbone of our networks, and is also used in many other region-based convolutional networks. The initial image data is always passed through the ResNet backbone as a first step to bootstrap the complete deep network. Note that for the Mask R-CNN architectures we describe below, the architecture shown is equivalent to the standard ResNet-50 backbone.

We additionally introduce one small extension that will be useful for our Motion R-CNN network. In ResNet-50, the C_5 bottleneck has a stride of 32 with respect to the input image resolution. In FlowNetS [10], their bottleneck stride is 64. For accurately estimating motions corresponding to larger pixel displacements, a larger stride may be important. Thus, we add an additional C_6 block to be used in the Motion R-CNN ResNet variants to increase the bottleneck stride to 64, following FlowNetS.

2.5 Region-based CNNs

We now give an overview of region-based convolutional networks, which are currently by far the most popular deep networks for object detection, and have recently also been applied to instance segmentation.

R-CNN

The very first region-based convolutional networks (R-CNNs) [7] used a non-learned algorithm external to a standard encoder CNN for computing *region proposals* in the shape of 2D bounding boxes, which represent regions that may contain an object. For each of the region proposals, the input image is cropped using the region bounding box and the crop is passed through the CNN, which performs classification of the object (or non-object, if the region shows background).

Output	Layer Operations	Output Dimensions
	input image	$H \times W \times C$
	ResNet	
C_1	7×7 conv, 64, stride 2	$\frac{1}{2} H \times \frac{1}{2} W \times 64$
	3×3 max pool, stride 2	$\frac{1}{4} H \times \frac{1}{4} W \times 64$
C_2	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix}_b \times 3$	$\frac{1}{4} H \times \frac{1}{4} W \times 256$
C_3	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix}_{b/2} \times 4$	$\frac{1}{8} H \times \frac{1}{8} W \times 512$
C_4	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix}_{b/2} \times 6$	$\frac{1}{16} H \times \frac{1}{16} W \times 1024$
C_5	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix}_{b/2} \times 3$	$\frac{1}{32} H \times \frac{1}{32} W \times 2048$
C_6	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix}_{b/2} \times 2$	$\frac{1}{64} H \times \frac{1}{64} W \times 2048$

Table 3: Backbone architecture based on ResNet-50 [24]. Operations enclosed in a $[\cdot]_b$ block make up a single ResNet "bottleneck" block (see Figure 5). If the block is denoted as $[\cdot]_b/2$, the first convolution operation in the block has a stride of 2. Note that the stride is only applied to the first block, but not to repeated blocks. Batch normalization [13] is used after every residual unit.

Fast R-CNN

The original R-CNN involves computing one forward pass of the CNN for each of the region proposals, which is costly, as there generally is a large number of proposals. Fast R-CNN [11] significantly reduces computation by performing only a single forward pass with the whole image as input to the CNN (compared to the sequential input of crops in the case of R-CNN). Then, fixed size ($H \times W$) feature maps are extracted from the compressed feature map of the image, each corresponding to one of the proposal bounding boxes. The extracted per-RoI (region of interest) feature maps are collected into a batch and passed into a small Fast R-CNN *head* network, which performs classification and prediction of refined boxes for all regions in one forward pass. Feature extraction is performed using *RoI pooling*. In RoI pooling, the RoI bounding box window over the backbone features is divided into a $H \times W$ grid of cells. For each cell, the values of the underlying feature map are max-pooled to yield the output value at the cell. Thus, given region proposals, all computation is reduced to a single pass through the complete network, speeding up the system by two orders of magnitude at inference time and one order of magnitude at training time.

Faster R-CNN

After streamlining the CNN components, Fast R-CNN is limited by the speed of the region proposal algorithm, which has to be run prior to the network passes and makes up a large portion of the total processing time. The Faster R-CNN object detection system [17] unifies the generation of region proposals and subsequent box refinement and classification into a single deep network, leading to faster training and test-time processing when compared to Fast R-CNN and again, improved accuracy. This unified network operates in two stages. In the *first stage*, one forward pass is performed on the *backbone* network, which is a deep feature encoder CNN with the original image as input. Next, the output features from the backbone are passed into a small, fully-convolutional *Region Proposal Network* network (RPN), which predicts objectness scores and regresses bounding boxes at each of its output positions. At any of the $h \times w$ output positions of the RPN, N_a bounding boxes with their *objectness* scores are predicted as offsets relative to a fixed set of N_a *anchors* with different aspect ratios and scales. Thus, there are $N_a \times h \times w$ reference anchors in total. In Faster R-CNN, $N_a = 9$, with 3 scales, corresponding to anchor boxes of areas of $\{128^2, 256^2, 512^2\}$ pixels, and 3 aspect ratios, $\{1:2, 1:1, 2:1\}$. For the ResNet Faster R-CNN backbone, we generally have a stride of 16 with respect to the input image at the RPN output (Table 4).

For each RPN prediction at a given position, the objectness score tells us how likely it is to correspond to a detection. The region proposals can then be obtained as the N highest scoring RPN predictions.

Then, the *second stage* corresponds to the original Fast R-CNN head, performing classification and bounding box refinement for each of the region proposals, which are now obtained from the RPN instead of being pre-computed by an external algorithm. As in Fast R-CNN, RoI pooling is used to extract one fixed size feature map for each of the region proposals, and the refined bounding boxes are predicted separately for each object class.

Table 4 includes an overview of the Faster R-CNN ResNet architecture (for Faster R-CNN, the mask head is ignored).

Mask R-CNN

Faster R-CNN and the earlier systems detect and classify objects at bounding box granularity. However, it can be helpful to know class and object (instance) membership of individual pixels, which generally involves computing a binary image mask for each object instance specifying which pixels belong to that object. This problem is called *instance segmentation*. Mask R-CNN [34] extends Faster R-CNN to instance segmentation by predicting fixed resolution instance masks within the bounding boxes of each detected object, which are, at test-time, bilinearly resized to fit inside the respective bounding boxes. For this, Mask R-CNN simply extends the Faster R-CNN head with multiple convolutions, which compute a pixel-precise binary mask for each instance. Note that the per-class masks *logits* (raw network outputs) are put through a sigmoid layer, and thus there is no comptetition between classes in the mask prediction branch.

Additionally, an important technical aspect of Mask R-CNN is the replacement of RoI pooling with bilinear sampling for extracting the RoI features, which is much more precise. In the original RoI pooling adopted from Fast R-CNN, the bins for max-pooling are not aligned with the actual pixel boundaries of the bounding boxes, and thus some detail is lost.

The basic Mask R-CNN ResNet architecture is shown in Table 4.

Output	Layer Operations	Output Dimensions
	input image	$H \times W \times C$
C_4	ResNet {up to C_4} (Table 3)	$\frac{1}{16} H \times \frac{1}{16} W \times 1024$
	Region Proposal Network (RPN)	
R_0	From C_4: 1×1 conv, 512	$\frac{1}{16} H \times \frac{1}{16} W \times 512$
	1×1 conv, 4	$\frac{1}{16} H \times \frac{1}{16} W \times N_a \cdot 4$
	flatten	$A \times 4$
$boxes_{RPN}$	decode bounding boxes (Eq. 9)	$A \times 4$
	From R_0: 1×1 conv, 2	$\frac{1}{16} H \times \frac{1}{16} W \times N_a \cdot 2$
	flatten	$A \times 2$
$scores_{RPN}$	softmax	$A \times 2$
ROI_{RPN}	sample $boxes_{RPN}$ and $scores_{RPN}$	$N_{RoI} \times 6$
	RoI Head	
	From C_4 with ROI_{RPN}: RoI extraction	$N_{RoI} \times 7 \times 7 \times 1024$
R_1	ResNet {C_5 without stride} (Table 3)	$N_{RoI} \times 7 \times 7 \times 2048$
ave	average pool	$N_{RoI} \times 2048$
	From ave: fully connected, $N_{cls} \cdot 4$	$N_{RoI} \times N_{cls} \cdot 4$
boxes	decode bounding boxes (Eq. 9)	$N_{RoI} \times N_{cls} \cdot 4$
	From ave: fully connected, N_{cls}	$N_{RoI} \times N_{cls}$
classes	softmax, N_{cls}	$N_{RoI} \times N_{cls} + 1$
	RoI Head: Masks	
M_0	From R_1: 2×2 deconv, 256, stride 2	$N_{RoI} \times 14 \times 14 \times 256$
	1×1 conv, N_{cls}	$N_{RoI} \times 14 \times 14 \times N_{cls}$
masks	sigmoid	$N_{RoI} \times 28 \times 28 \times N_{cls}$

Table 4: Mask R-CNN [34] ResNet-50 [24] architecture. Note that this is equivalent to the Faster R-CNN ResNet-50 architecture if the mask head is left out. In Mask R-CNN, bilinear sampling is used for RoI extraction, whereas Faster R-CNN uses RoI pooling.

Feature Pyramid Networks

In Faster R-CNN, a single feature map is used as the source of all RoI features during RoI extraction, independent of the size of the bounding box of any specific RoI. However, for small objects, the C_4 (see Table 3) features might have lost too much spatial information to allow properly predicting the exact bounding box and a high resolution mask. As a solution to this, the Feature Pyramid Network (FPN) [39] enables features of an appropriate scale to be used for RoI extraction, depending on the size of the bounding box of the RoI. For this, a pyramid of feature maps is created on top of the ResNet [24] encoder by combining bilinearly upsampled feature maps coming from the bottleneck with lateral skip connections from the encoder (Figure 6). For each consecutive upsampling block, the lateral skip connections are taken from the encoder block with the same output resolution as the upsampled features coming from the bottleneck.

Instead of a single RPN head with anchors at 3 scales and 3 aspect ratios, the FPN variant has one RPN head after each of the pyramid levels P_2 ... P_6 (see Table 5). At each output position of the resulting RPN pyramid, bounding boxes are predicted with respect to 3 anchor aspect ratios $\{1:2, 1:1, 2:1\}$

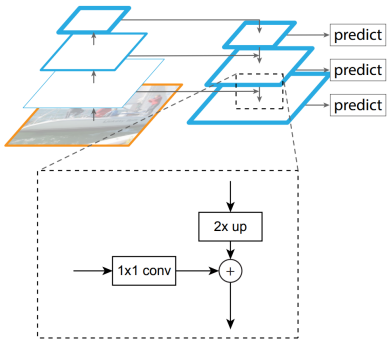

Figure 6: FPN block from [39]. Lower resolution features coming from the bottleneck are bilinearly upsampled and added with higher resolution skip connections from the encoder. Figure taken from [39].

and a single scale ($N_a = 3$). For P_2, P_3, P_4, P_5, P_6, the scale corresponds to anchor bounding boxes of areas $32^2, 64^2, 128^2, 256^2, 512^2$, respectively. Note that there is no need for multiple anchor scales per anchor position anymore, as the RPN heads themselves correspond to different scales. Now, in the RPN, higher resolution feature maps can be used for regressing smaller bounding boxes. For example, boxes of area close to 32^2 are predicted using P_2, which has a stride of 4 with respect to the input image. Most importantly, the RoI features can now be extracted from the pyramid level P_j appropriate for a RoI bounding box with size $h \times w$, where

$$j = 2 + j_a, \tag{1}$$

$$j_a = \text{clip}\left(\left[\log_2\left(\frac{\sqrt{w \cdot h}}{s_0}\right)\right], 0, 4\right) \tag{2}$$

is the index (from small anchor to large anchor) of the corresponding anchor box, and

$$s_0 = 256 \cdot 0.125 \tag{3}$$

is the scale of the smallest anchor boxes. This formula is slightly different from the one used in the FPN paper, as we want to assign the bounding boxes which are at the same scale as some anchor to the exact same pyramid level from which the RPN of this anchor is computed. Now, for example, the smallest boxes are cropped from P_2, which is the highest resolution feature map.

The Mask R-CNN ResNet-FPN variant is shown in Table 5.

2.6 Mask R-CNN: Training and inference

Loss definitions

For regression, we define the smooth-ℓ_1 regression loss as

$$\ell_{reg}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \tag{4}$$

which provides a certain robustness to outliers and will be used frequently in the following chapters. For vector or tuple arguments, the sum of the componentwise scalar losses is computed. For classification with mutually exclusive classes, we define the categorical (softmax) cross-entropy loss,

$$\ell_{cls}(c, c^*) = -\log(c_{c^*}), \tag{5}$$

where $c^* \in \{0, \mathrm{C}\}$ is a ground truth label, c is the output vector from a softmax layer, $c_{c^*} \in (0, 1)$ is the output probability for class c^*, and C is the number of classes. Note that for the object category classifier, $\mathrm{C} = \mathrm{N}_{cls} + 1$, as N_{cls} does not include the background class. Finally, for multi-label classification, we define the binary (sigmoid) cross-entropy loss,

$$\ell_{cls*}(y, y^*) = -y^* \cdot \log(y) - (1 - y^*) \cdot \log(1 - y), \tag{6}$$

where $y^* \in \{0, 1\}$ is a ground truth label and $y \in (0, 1)$ is the output of a sigmoid layer. Note that for the mask loss that will be introduced below, ℓ_{cls*} is the sum of the ℓ_{cls*}-losses for all 2D positions over the mask.

Bounding box regression

All bounding boxes predicted by the RoI head or RPN are estimated as offsets with respect to a reference bounding box. In the case of the RPN, the reference bounding box is one of the anchors, and refined bounding boxes from the RoI head are predicted relative to the RPN output bounding boxes. Let (x, y, w, h) be the top left coordinates, width, and height of the bounding box to be predicted. Likewise, let (x^*, y^*, w^*, h^*) be the ground truth bounding box and let (x_r, y_r, w_r, h_r) be the reference bounding box. The ground truth *box encoding* b_e^* is then defined as

$$b_e^* = (b_x^*, b_y^*, b_w^*, b_h^*), \tag{7}$$

where

$$b_x^* = \frac{x^* - x_r}{w_r}, \qquad b_y^* = \frac{y^* - y_r}{h_r},$$

$$b_w^* = \log\left(\frac{w^*}{w_r}\right), \qquad b_h^* = \log\left(\frac{h^*}{h_r}\right),$$

which represents the regression target for the bounding box outputs of the network.

Thus, for bounding box regression, the network predicts the box encoding b_e,

$$b_e = (b_x, b_y, b_w, b_h), \tag{8}$$

where

$$b_x = \frac{x - x_r}{w_r}, \qquad b_y = \frac{y - y_r}{h_r},$$

$$b_w = \log\left(\frac{w}{w_r}\right), \qquad b_h = \log\left(\frac{h}{h_r}\right).$$

At test time, to convert from a predicted box encoding b_e to the predicted bounding box b, the definitions above can be inverted,

$$b = (x, y, w, h), \tag{9}$$

where

$$x = b_x \cdot w_r + x_r, \qquad y = b_y \cdot b_r + y_r,$$

$$w = \exp(b_w) \cdot w_r, \qquad h = \exp(b_h) \cdot h_r,$$

and thus the bounding box is obtained as the reference bounding box adjusted by the predicted relative offsets and scales encoded in b_e.

Supervision of the RPN

A positive RPN proposal is defined as one with a IoU of at least 0.7 with a ground truth bounding box. For training the RPN, $\mathrm{N}_{RPN} = 256$ positive and negative examples are randomly sampled from the set of all RPN proposals, with at most 50% positive examples (if there are less positive examples, more negative examples are used instead). For examples selected in this way, a regression loss is computed between predicted and ground truth bounding box encoding, and a classification loss is computed for the predicted objectness scores. Specifically, let $s_i^* = 1$ if proposal i is positive and $s_i^* = 0$ if it is negative, let s_i be the predicted objectness score and b_i, b_i^* the predicted and ground truth bounding box encodings. Then, the RPN loss is computed as

$$L_{RPN} = L_{obj} + L_{box}^{RPN}, \tag{10}$$

where

$$L_{obj} = \frac{1}{\mathrm{N}_{RPN}} \sum_{i=1}^{\mathrm{N}_{RPN}} \ell_{cls}(s_i, s_i^*), \tag{11}$$

$$L_{box}^{RPN} = \frac{1}{\mathrm{N}_{RPN}^{pos}} \sum_{i=1}^{\mathrm{N}_{RPN}} s_i^* \cdot \ell_{reg}(b_i^* - b_i), \tag{12}$$

and

$$\mathrm{N}_{RPN}^{pos} = \sum_{i=1}^{\mathrm{N}_{RPN}} s_i^* \tag{13}$$

is the number of positive examples. Note that the bounding box loss is only active for positive examples, and that the classification loss is computed between the classes {object, non-object}.

Supervision of the Mask R-CNN RoI head

For selecting RoIs to train the RoI head network, a foreground example is defined as one with a IoU of at least 0.5 with a ground truth bounding box, and a background example is defined as one with a maximum IoU in $[0.1, 0.5)$. A total of 64 (without FPN) or 512 (with FPN) RoIs are sampled, with at most 25% foreground examples. Now, let c_i^* be the ground truth object class, where $c_i^* = 0$ for background examples and $c_i^* \in \{1, ..., \mathrm{N}_{cls}\}$ for foreground examples, and let c_i be the RoI class prediction. Then, for any foreground RoI, let b_i^* be the ground truth bounding box encoding and b_i the predicted refined RoI box encoding for class c_i^*. Additionally, for any foreground RoI, let m_i be the predicted $m \times m$ mask for class c_i^* and m_i^* the $m \times m$ mask target with values in $\{0, 1\}$, where the mask target is cropped and resized from the binary ground truth mask using the RPN proposal bounding box. In our implementation, we use nearest neighbour resizing for resizing the cropped mask targets. Note that values in m_i and c_i are already normalized probabilities from sigmoid and softmax layers, respectively.

Then, the ROI loss is computed as

$$L_{RoI} = L_{cls} + L_{box} + L_{mask}, \tag{14}$$

where

$$L_{cls} = \frac{1}{\mathrm{N}_{RoI}} \sum_{i=1}^{\mathrm{N}_{RoI}} \ell_{cls}(c_i, c_i^*), \tag{15}$$

is the average (categorical) cross-entropy classification loss,

$$L_{box} = \frac{1}{\mathrm{N}_{RoI}^{fg}} \sum_{i=1}^{\mathrm{N}_{RoI}} [c_i^* \geq 1] \cdot \ell_{reg}(b_i^* - b_i) \tag{16}$$

is the average smooth-ℓ_1 bounding box regression loss,

$$L_{mask} = \frac{1}{\mathrm{N}_{RoI}^{fg}} \sum_{i=1}^{\mathrm{N}_{RoI}} [c_i^* \geq 1] \cdot \ell_{cls*}(m_i, m_i^*) \tag{17}$$

is the average (binary) cross-entropy mask loss,

$$\mathrm{N}_{RoI}^{fg} = \sum_{i=1}^{\mathrm{N}_{RoI}} [c_i^* \geq 1] \tag{18}$$

is the number of foreground examples, and

$$[c_i^* \geq 1] = \begin{cases} 1 & c_i^* \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

is the Iverson bracket indicator function. Thus, the bounding box and mask losses are only enabled for the foreground RoIs. Note that the bounding box and mask predictions for all classes other than c_i^* are not penalized.

Inference

During inference, the 300 (ResNet) or 1000 (ResNet-FPN) highest scoring region proposals from the RPN are selected. The corresponding features are extracted from the backbone, as during training, by using the RPN bounding boxes, and passed through the RoI bounding box refinement and classification heads (but not through the mask head). After this, non-maximum supression (NMS) is applied to predicted RoIs for which the predicted class is not the background class, with a maximum IoU of 0.7 of the refined boxes. Finally, the mask head is applied to the 100 highest scoring (after NMS) refined boxes, after extracting the corresponding features again. Thus, during inference, the features for the mask head are extracted using the refined bounding boxes for the predicted class, instead of the RPN bounding boxes. This is important for not introducing any misalignment, as the instance masks are to be created inside of the final, more precise, refined detection bounding boxes. Furthermore, note that bounding box and mask predictions for all classes but the predicted class (the highest scoring class) are discarded, and thus the output bounding box and mask correspond to the highest scoring class.

Output	Layer Operations	Output Dimensions
	input image	$H \times W \times C$
C_5	ResNet {up to C_5} (Table 3)	$\frac{1}{32} H \times \frac{1}{32} W \times 1024$
	Feature Pyramid Network (FPN)	
P_5	From C_5: 1×1 conv, 256	$\frac{1}{32} H \times \frac{1}{32} W \times 256$
P_4	$\left[\text{skip from } C_4\right]_p$	$\frac{1}{16} H \times \frac{1}{16} W \times 256$
P_3	$\left[\text{skip from } C_3\right]_p$	$\frac{1}{8} H \times \frac{1}{8} W \times 256$
P_2	$\left[\text{skip from } C_2\right]_p$	$\frac{1}{4} H \times \frac{1}{4} W \times 256$
P_6	From P_5: 2×2 subsample	$\frac{1}{64} H \times \frac{1}{64} W \times 256$
	Region Proposal Network (RPN)	
	$\forall i \in \{2...6\}$	
	From P_i: 1×1 conv, 512	$\frac{1}{2^i} H \times \frac{1}{2^i} W \times 512$
	1×1 conv, 6	$\frac{1}{2^i} H \times \frac{1}{2^i} W \times N_a \cdot 6$
RPN_i	flatten	$A_i \times 6$
	From $\{RPN_2 \ldots RPN_6\}$: concatenate	$A \times 6$
	decode bounding boxes (Eq. 9)	$A \times 6$
ROI_{RPN}	sample bounding boxes & scores	$N_{RoI} \times 6$
	RoI Head	
R_2	From $\{P_2 \ldots P_6\}$ with ROI_{RPN}: RoI extraction (Eq. 3)	$N_{RoI} \times 14 \times 14 \times 256$
	2×2 max pool	$N_{RoI} \times 7 \times 7 \times 256$
F_1	$\left[\text{fully connected}, 1024\right] \times 2$	$N_{RoI} \times 1024$
	From F_1: fully connected, $N_{cls} \cdot 4$	$N_{RoI} \times N_{cls} \cdot 4$
boxes	decode bounding boxes (Eq. 9)	$N_{RoI} \times N_{cls} \cdot 4$
	From F_1: fully connected, N_{cls}	$N_{RoI} \times N_{cls}$
classes	softmax, N_{cls}	$N_{RoI} \times N_{cls} + 1$
	RoI Head: Masks	
M_1	From R_2: $\left[3 \times 3 \text{ conv}\right] \times 4$, 256	$N_{RoI} \times 14 \times 14 \times 256$
	2×2 deconv, 256, stride 2	$N_{RoI} \times 28 \times 28 \times 256$
	1×1 conv, N_{cls}	$N_{RoI} \times 28 \times 28 \times N_{cls}$
masks	sigmoid	$N_{RoI} \times 28 \times 28 \times N_{cls}$

Table 5: Mask R-CNN [34] ResNet-50-FPN [24] architecture. Operations enclosed in a $[\cdot]_p$ block make up a single FPN block (see Figure 6).

3 Motion R-CNN

3.1 Motion R-CNN

Building on Mask R-CNN [34], we estimate per-object motion by predicting the 3D motion of each detected object. For this, we extend Mask R-CNN in two straightforward ways. First, we modify the backbone network and provide it with two frames in order to enable image matching between the consecutive frames. Second, we extend the Mask R-CNN RoI head to predict a 3D motion and pivot for each region proposal. Tables 6 and 7 show our Motion R-CNN networks based on Mask R-CNN ResNet and Mask R-CNN ResNet-FPN, respectively.

Output	Layer Operations	Output Dimensions
	input images I_t, I_{t+1}, and (optional) XYZ_t, XYZ_{t+1}	$\text{H} \times \text{W} \times \text{C}$
C_4	ResNet {up to C_4} (Table 3)	$\frac{1}{16}\,\text{H} \times \frac{1}{16}\,\text{W} \times 1024$
	Region Proposal Network (RPN) (Table 4)	
	Camera Motion Network	
	From C_4: ResNet {C_5} (Table 3)	$\frac{1}{32}\,\text{H} \times \frac{1}{32}\,\text{W} \times 2048$
	ResNet {C_6} (Table 3)	$\frac{1}{64}\,\text{H} \times \frac{1}{64}\,\text{W} \times 2048$
	bilinear resize, 7×7	$7 \times 7 \times 512$
	flatten	$1 \times 7 \cdot 7 \cdot 512$
T_0	$\left[\text{fully connected}, 1024\right] \times 2$	1×1024
R_{cam}	From T_0: fully connected, 3	1×3
t_{cam}	From T_0: fully connected, 3	1×3
	From T_0: fully connected, 2	1×2
o_{cam}	softmax, 2	1×2
	RoI Head & RoI Head: Masks (Table 4)	
	RoI Head: Motions	
T_1	From ave: $\left[\text{fully connected}, 1024\right] \times 2$	$\text{N}_{RoI} \times 1024$
$\forall k : R_k$	From T_1: fully connected, 3	$\text{N}_{RoI} \times 3$
$\forall k : t_k$	From T_1: fully connected, 3	$\text{N}_{RoI} \times 3$
$\forall k : p_k$	From T_1: fully connected, 3	$\text{N}_{RoI} \times 3$
	From T_1: fully connected, 2	$\text{N}_{RoI} \times 2$
$\forall k : o_k$	softmax, 2	$\text{N}_{RoI} \times 2$

Table 6: Motion R-CNN ResNet architecture based on the Mask R-CNN ResNet architecture (Table 4). We use ReLU activations after all hidden layers and additonally dropout with $p = 0.5$ after all fully-connected hidden layers.

Motion R-CNN backbone

Like Faster R-CNN and Mask R-CNN, we use a ResNet variant [24] as backbone network to compute feature maps from input imagery.

Inspired by FlowNetS [10], we make one modification to the ResNet backbone to enable image matching, laying the foundation for our motion estimation. Instead of taking a single image as input

Output	Layer Operations	Output Dimensions
	input images I_t, I_{t+1}, and (optional) XYZ_t, XYZ_{t+1}	$\text{H} \times \text{W} \times \text{C}$
C_6	ResNet (Table 3)	$\frac{1}{64}\,\text{H} \times \frac{1}{64}\,\text{W} \times 2048$
	RPN & FPN (Table 5)	
	Camera Motion Network	
	From C_6: 1×1 conv, 512	$\frac{1}{64}\,\text{H} \times \frac{1}{64}\,\text{W} \times 512$
	bilinear resize, 7×7	$7 \times 7 \times 512$
	flatten	$1 \times 7 \cdot 7 \cdot 512$
T_2	$\left[\text{fully connected}, 1024\right] \times 2$	1×1024
R_{cam}	From T_2: fully connected, 3	1×3
t_{cam}	From T_2: fully connected, 3	1×3
	From T_2: fully connected, 2	1×2
o_{cam}	softmax, 2	1×2
	RoI Head & RoI Head: Masks (Table 5)	
	RoI Head: Motions	
T_3	From F_1: $\left[\text{fully connected}, 1024\right] \times 2$	$\text{N}_{RoI} \times 1024$
$\forall k : R_k$	From T_3: fully connected, 3	$\text{N}_{RoI} \times 3$
$\forall k : t_k$	From T_3: fully connected, 3	$\text{N}_{RoI} \times 3$
$\forall k : p_k$	From T_3: fully connected, 3	$\text{N}_{RoI} \times 3$
	From T_2: fully connected, 2	$\text{N}_{RoI} \times 2$
$\forall k : o_k$	softmax, 2	$\text{N}_{RoI} \times 2$

Table 7: Motion R-CNN ResNet-FPN architecture based on the Mask R-CNN ResNet-FPN architecture (Table 5). To obtain a larger bottleneck stride, we compute the feature pyramid starting with C_6 instead of C_5, and thus, the subsampling from P_5 to P_6 is omitted. The modifications are analogous to our Motion R-CNN ResNet, but we still show the architecture for completeness. Again, we use ReLU activations after all hidden layers and additonally dropout with $p = 0.5$ after all fully-connected hidden layers.

to the backbone, we depth-concatenate two temporally consecutive frames I_t and I_{t+1}, yielding a input image map with six channels. Additionally, we also experiment with concatenating the camera space XYZ coordinates for each frame, XYZ_t and XYZ_{t+1}, into the input as well. We do not introduce a separate network for computing region proposals and use our modified backbone network as both RPN and for extracting the RoI features.

Technically, our feature encoder network will have to learn image matching representations similar to those learned by the FlowNet encoder, but the output will be computed in the object-centric framework of a region-based convolutional network head with a 3D parametrization. Thus, in contrast to the dense FlowNet decoder, the estimated dense image matching information from the encoder is integrated for specific objects via RoI extraction and subsequently processed by the RoI head for each object.

Per-RoI motion prediction

We use a 3D rigid motion parametrization similar to the one used in SE3-Nets and SfM-Net [32, 41]. For the k-th object proposal, we predict the rigid transformation $\{R_k, t_k\} \in \mathbf{SE}(3)$ [1] of the object between the two frames I_t and I_{t+1}, as well as the object pivot $p_k \in \mathbb{R}^3$ at time t. We parametrize R_k using an Euler angle representation,

$$R_k = R_k^z(\gamma) \cdot R_k^x(\alpha) \cdot R_k^y(\beta), \tag{20}$$

where

$$R_k^x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}, \tag{21}$$

$$R_k^y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix}, \tag{22}$$

$$R_k^z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{23}$$

and α, β, γ are the rotation angles in radians about the x, y, z-axis, respectively.

We then extend the Mask R-CNN head by adding a small fully-connected network for motion prediction in addition to the fully-connected layers for refined boxes and classes and the convolutional network for the masks. Like for refined boxes and masks, we make one separate motion prediction for each class. Each instance motion is predicted as a set of nine values, $\sin(\alpha)$, $\sin(\beta)$, $\sin(\gamma)$, t_k and p_k, where $\sin(\alpha)$, $\sin(\beta)$ and $\sin(\gamma)$ are clipped to $[-1, 1]$. Here, we assume that motions between frames are relatively small and that objects rotate at most 90 degrees in either direction along any axis, which is in general a safe assumption for image sequences from videos, and enables us to obtain unique cosine values from the predicted sine values. All predictions are made in camera space, and translation and pivot predictions are in meters. We additionally predict softmax scores o_k for classifying the objects into still and moving objects. As a postprocessing, for any object instance k with predicted moving flag $o_k = 0$, we set $\sin(\alpha) = \sin(\beta) = \sin(\gamma) = 0$ and $t_k = (0, 0, 0)^T$, and thus predict an identity motion.

Camera motion prediction

In addition to the object transformations, we optionally predict the camera motion $\{R_{cam}, t_{cam}\} \in \mathbf{SE}(3)$ between the two frames I_t and I_{t+1}. For this, we branch off a small fully-connected network from the bottleneck output of the backbone. We again represent R_{cam} using a Euler angle representation and

[1] $\mathbf{SE}(3)$ refers to the Special Euclidean Group representing 3D rotations and translations: $\{R, t | R \in \mathbf{SO}(3), t \in \mathbb{R}^3\}$

predict $\sin(\alpha)$, $\sin(\beta)$, $\sin(\gamma)$ and t_{cam} in the same way as for the individual objects. Again, we predict a softmax score o_{cam} for differentiating between a still and moving camera.

3.2 Network design

Camera motion network

In our ResNet variant without FPN (Table 6), the underlying ResNet backbone is only computed up to the C_4 block, as otherwise the feature resolution prior to RoI extraction would be reduced too much. Therefore, in our variant without FPN, we first pass the C_4 features through C_5 and C_6 blocks (with weights independent from the C_5 block used in the RoI head in this variant) to increase the bottleneck stride prior to the camera motion network to 64. In our ResNet-FPN variant (Table 7), the backbone makes use of all blocks through C_6, and we can simply branch off our camera motion network from the C_6 bottleneck. Then, in both, the ResNet and ResNet-FPN variant, we apply one additional convolution to the C_6 features to reduce the number of inputs to the following fully-connected layers, and thus keep the number of weights reasonably small. Instead of averaging, we use bilinear resizing to bring the convolutional features to a fixed size without losing all spatial information, flatten them, and finally apply multiple fully-connected layers to predict the camera motion parameters.

RoI motion head network

In both of our network variants (Tables 6 and 7), we compute the fully-connected network for motion prediction from the flattened RoI features, which are also the basis for classification and bounding box refinement. Note that the features (extracted from the upsampled FPN stage appropriate to the RoI bounding box scales) passed to our ResNet-FPN RoI head went through the C_6 bottleneck, which has a stride of 64 with respect to the original image. In contrast, the bottleneck for the features passed to our ResNet RoI head is C_4 (with a stride of 16). Thus, the ResNet-FPN variant can in principle estimate object motions based on larger displacements than the ResNet variant. Additionally, as smaller bounding boxes use higher resolution features, the motions and pivots of (especially smaller) objects can in principle be more accurately estimated with the FPN variant.

3.3 Supervision

Per-RoI instance motion supervision with 3D instance motion ground truth

The most straightforward way to supervise the object motions is by using ground truth motions computed from ground truth object poses, which is in general only practical when training on synthetic datasets. Given the k-th foreground RoI (as defined for Mask R-CNN) with ground class c_k^*, let R_k, t_k, p_k, o_k be the predicted motion for class c_k^* as parametrized above, and $R_k^*, t_k^*, p_k^*, o_k^*$ the ground truth motion for the matched ground truth example. Similar to the camera pose regression loss in [37], we use a variant of the ℓ_1-loss to penalize the differences between ground truth and predicted rotation, translation (and pivot, in our case). We found that the smooth ℓ_1-loss performs better in our case than the standard ℓ_1-loss. We thus compute the RoI motion loss as

$$L_{motion} = \frac{1}{\mathrm{N}_{RoI}^{fg}} \sum_{k}^{\mathrm{N}_{RoI}} (l_R^k + l_t^k) \cdot o_k^* + l_p^k + l_o^k, \tag{24}$$

where

$$l_R^k = \ell_{reg}(R_k^* - R_k), \tag{25}$$

$$l_t^k = \ell_{reg}(t_k^* - t_k), \tag{26}$$

and

$$l_p^k = \ell_{reg}(p_k^* - p_k) \tag{27}$$

are the smooth-ℓ_1 losses for the predicted rotation, translation and pivot, respectively and

$$l_o^k = \ell_{cls}(o_k, o_k^*) \tag{28}$$

is the (categorical) cross-entropy loss for the predicted classification into moving and non-moving objects.

Note that we do not penalize the rotation and translation for objects with $o_k^* = 0$, which do not move between t and $t+1$. We found that the network may not reliably predict exact identity motions for still objects, which is numerically more difficult to optimize than performing classification between moving and non-moving objects and discarding the regression for the non-moving ones. Also, analogously to masks and bounding boxes, the estimates for classes other than c_k^* are not penalized.

Now, our modified RoI loss is

$$L_{RoI} = L_{cls} + L_{box} + L_{mask} + L_{motion}. \tag{29}$$

Camera motion supervision

We supervise the camera motion with ground truth analogously to the object motions, with the only difference being that we only have a rotation and translation, but no pivot loss for the camera motion. If the ground truth shows that the camera is not moving, we again do not penalize rotation and translation. In this case, the camera motion loss is reduced to the classification loss.

Per-RoI instance motion supervision *without* 3D instance motion ground truth

A more general way to supervise the object motions is a re-projection loss similar to the unsupervised loss in SfM-Net [41], which we can apply to coordinates within the object bounding boxes, and which does not require ground truth 3D object motions.

In this case, for any RoI, we generate a uniform $m \times m$ grid of 2D points inside the RPN proposal bounding box with the same resolution as the predicted mask. Note that the predicted mask we use here was binarized at a threshold of 0.5. We use the same bounding box to crop the corresponding region from the dense, full-image depth map and bilinearly resize the depth crop to the same resolution as the mask and point grid. Next, we create a grid of 3D points (point cloud) from the grid of 2D points and depth crop. To this point cloud, we apply the object motion predicted for the RoI, masked by the

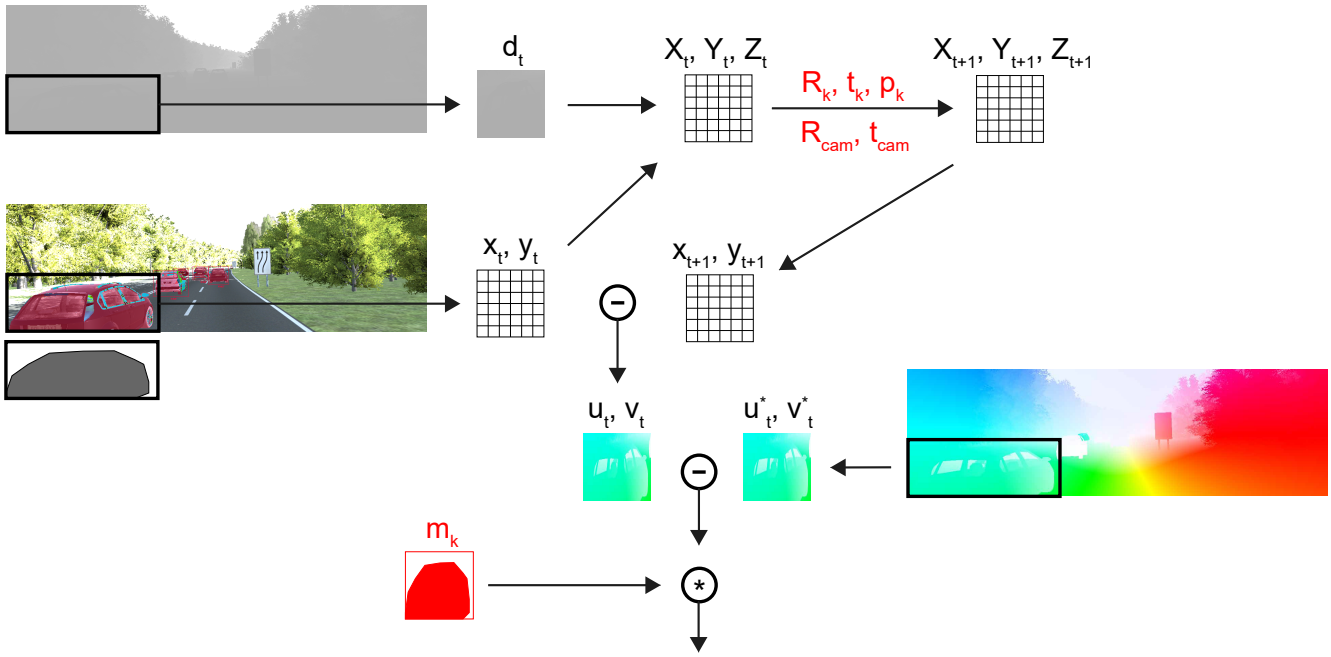

Figure 7: Overview of the alternative, flow-based loss for instance motion supervision without 3D instance motion ground truth. In contrast to SfM-Net [41], where a single optical flow field is composed and penalized to supervise the motion prediction, our loss considers the motion of all objects in isolation and composes a batch of flow windows for all RoIs. Network predictions are shown in red.

predicted mask. Then, we apply the camera motion to the 3D points, project them back to 2D and finally compute the optical flow at each point as the difference of the initial and re-projected 2D grids. Note that we batch this computation over all RoIs, so that we only perform it once per forward pass. Figure 7 illustrates the approach.

The mathematical details for the 3D transformations and mappings between 2D and 3D are analogous to the dense, full-image flow composition in the following subsection, so we will not duplicate them here. The only differences are that there is no sum over objects during the point transformation based on instance motion, as we consider the single object corresponding to an RoI in isolation, and that the masks are not resized to the full image resolution, as the depth crop and the grid of 2D points are at the same resolution as the predicted $m \times m$ mask.

For each RoI, we can now compute L_{RoI} and thus supervise the object motion by penalizing the $m \times m$ optical flow grid. If there is optical flow ground truth available, we can use the RoI bounding box to crop and resize a region from the ground truth optical flow to match the RoI's optical flow grid and penalize the difference between the flow grids with a (smooth) ℓ_1-loss.

However, we could also use the re-projection loss without optical flow ground truth to train the motion prediction in an unsupervised manner, similar to [41]. In this case, we could use the bounding box to crop and bilinearly resize the corresponding region from the first image I_t and bilinearly sample the corresponding region from the second image I_{t+1}, using the 2D point grid displaced with the predicted flow grid (which is often called *backward warping*). Then, we could penalize the difference between the resulting image crops, for example, with a census loss [3, 44]. For more details on differentiable bilinear sampling for deep learning, we refer the reader to [14].

When compared to supervision with 3D instance motion ground truth, a re-projection loss could benefit motion regression by removing any loss balancing issues between the rotation, translation and pivot losses [37], which could make it interesting even when 3D motion ground truth is available.

3.4 Training and inference

Training

We train the Motion R-CNN RPN and RoI heads in the exact same way as described for Mask R-CNN. We additionally compute the camera and instance motion losses and concatenate the additional frame (and, optionally, XYZ coordinates) into the network input, but otherwise do not modify the training procedure and sample proposals and RoIs in the exact same way.

Inference

During inference, we proceed analogously to Mask R-CNN. In the same way as the RoI mask head, at test time, we compute the RoI motion head from the features extracted with refined bounding boxes.

Again, as for masks and bounding boxes in Mask R-CNN, the predicted output object motion is the predicted object motion for the highest scoring class.

3.5 Dense flow from 3D motion

As a postprocessing, we compose the dense optical flow between I_t and I_{t+1} from the outputs of our Motion R-CNN network. Given the depth map d_t for frame I_t, we first create a 3D point cloud in camera space at time t, where

$$P_t = \begin{pmatrix} X_t \\ Y_t \\ Z_t \end{pmatrix} = \frac{d_t}{f} \begin{pmatrix} x_t - c_0 \\ y_t - c_1 \\ f \end{pmatrix}, \tag{30}$$

is the 3D coordinate at t corresponding to the point with pixel coordinates x_t, y_t, which range over all coordinates in I_t, and (c_0, c_1, f) are the camera intrinsics. For now, the depth map is always assumed to come from ground truth.

Given k detections with predicted motions as above, we transform all points within the bounding box and mask of a detected object according to the predicted motion of the object.

For this, we first define the *full image* mask M_k for object k, which can be computed from the predicted box mask m_k (for the predicted class) by bilinearly resizing it to the width and height of the predicted bounding box and then copying the values of the resized mask into a full (image) resolution mask initialized with zeros, starting at the top-left coordinate of the predicted bounding box. Again we binarize masks at a threshold of 0.5.

Then, given the predicted motions (R_k, t_k) and pivots p_k for all objects,

$$P'_{t+1} = P_t + \sum_{1}^{\mathrm{N}} M_k \left\{ R_k \cdot (P_t - p_k) + p_k + t_k - P_t \right\}, \tag{31}$$

where N is the number of detections. The motion predictions are understood to have already taken into account the classification into moving and still objects, and we thus have, as described above, identity motions for all objects with $o_k = 0$.

Next, we transform points given the camera transformation $\{R_{cam}, t_{cam}\} \in \mathbf{SE}(3)$,

$$\begin{pmatrix} X_{t+1} \\ Y_{t+1} \\ Z_{t+1} \end{pmatrix} = P_{t+1} = R_{cam} \cdot P'_{t+1} + t_{cam}. \tag{32}$$

Finally, we project the transformed 3D points at time $t+1$ to 2D pixel coordinates again,

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \frac{f}{Z_{t+1}} \begin{pmatrix} X_{t+1} \\ Y_{t+1} \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}. \tag{33}$$

We now obtain the optical flow between I_t and I_{t+1} at each point as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x_{t+1} - x_t \\ y_{t+1} - y_t \end{pmatrix}. \tag{34}$$

4 Experiments

4.1 Implementation

Our networks and loss functions are implemented using built-in TensorFlow functions [9], enabling us to use automatic differentiation for all gradient computations. To make our code easy to extend and flexible, we build on the TensorFlow Object detection API [35], which provides a Faster R-CNN baseline implementation. On top of this, we implemented Mask R-CNN and the Feature Pyramid Network (FPN) as well as the Motion R-CNN extensions for motion estimation and related evaluations and postprocessings. In addition, we generated all ground truth for Motion R-CNN in the form of TFRecords from the raw Virtual KITTI data to enable fast loading during training. Note that for RoI extraction and bilinear crop and resize operations, we use the `tf.crop_and_resize` TensorFlow function with interpolation set to bilinear.

4.2 Datasets

Virtual KITTI

The synthetic Virtual KITTI dataset [23] is a re-creation of the KITTI driving scenario [4, 16], rendered from virtual 3D street scenes. The dataset is made up of a total of 2126 frames from five different monocular sequences recorded from a camera mounted on a virtual car. Each sequence is rendered with varying lighting and weather conditions and from different viewing angles, resulting in a total of 10 variants per sequence. In addition to the RGB frames, a variety of ground truth is supplied. For each frame, we are given a dense depth and optical flow map and the camera extrinsics matrix. There are two annotated object classes, cars, and vans ($\mathrm{N}_{cls} = 2$). For all cars and vans in each frame, we are given 2D and 3D object bounding boxes, instance masks, 3D poses, and various other labels.

This makes the Virtual KITTI dataset ideally suited for developing our joint instance segmentation and motion estimation system, as it allows us to test different components in isolation and progress to more and more complete predictions up to supervising the full system on a single dataset.

For our experiments, we use the *clone* sequences, which are rendered in a way that most closely resembles the original KITTI dataset. We sample 100 examples to be used as validation set. From the remaining 2026 examples, we remove a small number of examples without object instances and use the resulting data as training set.

Motion ground truth from 3D poses and camera extrinsics

We will now describe how we use the ground truth poses and camera matrices from Virtual KITTI to compute instance and camera motion ground truth. For two consecutive frames I_t and I_{t+1}, let $[R_t^{ex}|t_t^{ex}]$ and $[R_{t+1}^{ex}|t_{t+1}^{ex}]$ be the camera extrinsics at the two frames. We compute the ground truth camera motion $\{R_{cam}^*, t_{cam}^*\} \in \mathbf{SE}(3)$ as

$$R_{cam}^* = R_{t+1}^{ex} \cdot \mathrm{inv}(R_t^{ex}), \tag{35}$$

$$t^*_{cam} = t^{ex}_{t+1} - R^*_{cam} \cdot t^{ex}_t. \tag{36}$$

Additionally, we define $o^*_{cam} \in \{0, 1\}$,

$$o^*_{cam} = \begin{cases} 1 & \text{if the camera pose changes between } t \text{ and } t+1 \\ 0 & \text{otherwise,} \end{cases} \tag{37}$$

which specifies whether the camera is moving in between the frames.

For any object k visible in both frames, let (R^k_t, t^k_t) and (R^k_{t+1}, t^k_{t+1}) be its orientation and position in camera space at I_t and I_{t+1}, respectively. Note that the pose at t is given with respect to the camera at t and the pose at $t+1$ is given with respect to the camera at $t+1$.

We define the ground truth pivot $p^*_k \in \mathbb{R}^3$ as

$$p^*_k = t^k_t \tag{38}$$

and compute the ground truth object motion $\{R^*_k, t^*_k\} \in \mathbf{SE}(3)$ as

$$R^*_k = R^k_{t+1} \cdot R^*_{cam} \cdot \text{inv}(R^k_t), \tag{39}$$

$$t^*_k = \text{inv}(R^*_{cam}) \cdot t^k_{t+1} + t^*_{cam^{-1}} - R^*_k \cdot t^k_t, \tag{40}$$

where

$$t^*_{cam^{-1}} = t^{ex}_t - inv(R^*_{cam}) \cdot t^{ex}_{t+1}. \tag{41}$$

As for the camera, we define $o^*_k \in \{0, 1\}$,

$$o^*_k = \begin{cases} 1 & \text{if the position of object i changes between } t \text{ and } t+1 \\ 0 & \text{otherwise,} \end{cases} \tag{42}$$

which specifies whether an object is moving in between the frames.

Evaluation metrics with motion ground truth

To evaluate the 3D instance and camera motions on the Virtual KITTI validation set, we introduce a few error metrics. Given a foreground detection k with an IoU of at least 0.5 with a ground truth example, let R_k, t_k, p_k, o_k be the predicted (and postprocessed) motion for the predicted class c_k and

$R_k^*, t_k^*, p_k^*, o_k^*$ the motion ground truth for the best matching example. Then, assuming there are N such detections,

$$E_R = \frac{1}{N} \sum_k \arccos\left(\min\left\{ 1, \max\left\{ -1, \frac{\mathrm{tr}(\mathrm{inv}(R_k^*) \cdot R_k) - 1}{2} \right\} \right\} \right) \tag{43}$$

measures the mean angle of the error rotation between predicted and ground truth rotation,

$$E_t = \frac{1}{N} \sum_k \left\| \mathrm{inv}(R_k) \cdot (t_k^* - t_k) \right\|_2, \tag{44}$$

is the mean Euclidean distance between predicted and ground truth translation, and

$$E_p = \frac{1}{N} \sum_k \left\| p_k^* - p_k \right\|_2 \tag{45}$$

is the mean Euclidean distance between predicted and ground truth pivot.

Moreover, we define precision and recall measures for the detection of moving objects, where

$$O_{pr} = \frac{TP}{TP + FP} \tag{46}$$

is the fraction of objects which are actually moving among all objects classified as moving, and

$$O_{rc} = \frac{TP}{TP + FN} \tag{47}$$

is the fraction of objects correctly classified as moving among all objects which are actually moving. Here, we used

$$TP = \sum_k [o_k = 1 \wedge o_k^* = 1], \tag{48}$$

$$FP = \sum_k [o_k = 1 \wedge o_k^* = 0], \tag{49}$$

and

$$FN = \sum_k [o_k = 0 \wedge o_k^* = 1]. \tag{50}$$

Analogously, we define error metrics E_R^{cam} and E_t^{cam} for the predicted camera motion.

Network	Instance Motion					Camera Motion		Flow Error	
FPN	$E_R[deg]$	$E_t[m]$	$E_p[m]$	O_{pr}	O_{rc}	$E_R^{cam}[deg]$	$E_t^{cam}[m]$	AEE	Fl-all
-	(0.279)	(0.442)	-	-	-	(0.220)	(0.684)	-	-
×	0.301	0.237	3.331	0.790	0.916	0.087	0.053	11.17	24.91%
✓	0.293	0.210	1.958	0.844	0.914	0.169	0.050	8.29	45.22%

Table 8: Evaluation of different metrics on the Virtual KITTI validation set. AEE: Average Endpoint Error; Fl-all: Ratio of pixels where flow estimate is wrong by both ≥ 3 pixels and $\geq 5\%$. We compare network variants with and without FPN. All metrics are averaged over all examples in the validation set. Quantities in parentheses in the first row are the average ground truth values for the estimated quantity. For example, we compare the error in camera angle, $E_R^{cam}[deg]$, to the average rotation angle in the ground truth camera motions.

4.3 Virtual KITTI: Training setup

Training schedule

Our training schedule is similar to the Mask R-CNN Cityscapes schedule [34]. We train for a total of 192K iterations on the Virtual KITTI training set. For this, we use a single Titan X (Pascal) GPU and a batch size of 1, which results in approximately one day of training for a complete run. As optimizer, we use stochastic gradient descent (SGD) [1] with momentum set to 0.9. As learning rate we use $0.25 \cdot 10^{-2}$ for the first 144K iterations and $0.25 \cdot 10^{-3}$ for all remaining iterations.

R-CNN training parameters

For training the RPN and RoI heads and during inference, we use the exact same number of proposals and RoIs as Mask R-CNN in the ResNet and ResNet-FPN variants, respectively. All losses (the original ones and our new motion losses) are added up without additional weighting between the loss terms, as in Mask R-CNN.

Initialization

For initializing the C_1 to C_5 (see Table 3) weights, we use a pre-trained ImageNet [18] checkpoint from the official TensorFlow repository. Following the pre-existing TensorFlow implementation of Faster R-CNN, we initialize all other hidden layers with He initialization [12]. For the fully-connected camera and instance motion output layers, we use a truncated normal initializer with a standard deviation of 0.0001 and zero mean, truncated at two standard deviations. Note that a larger weight prevented the angle sine estimates from properly converging to the very small values they are in general expected to output.

4.4 Virtual KITTI: Evaluation

For our initial experiments, we concatenate both RGB frames as well as the XYZ coordinates for both frames as input to the networks. We train both, the Motion R-CNN ResNet and ResNet-FPN variants, and supervise camera and instance motions with 3D motion ground truth.

In Figure 8, we visualize instance segmentation and optical flow results on the Virtual KITTI validation set. In Figure 9, we visually justify the addition of the classifier that decides between a moving and still object. In Table 8, we compare various metrics for the Motion R-CNN ResNet and ResNet-FPN network variants on the Virtual KITTI validation set.

Camera motion

Both variants achieve a low error in predicted camera translation, relative to the average ground truth camera translation. The camera rotation angle error is still relatively high, compared to the small average ground truth camera rotation. Although both variants use the exact same network for predicting the camera motion, the FPN variant performs worse here, with the error in rotation angle twice as high. One possible explanations that should be investigated in future work is that in the FPN variant, all blocks in the backbone are shared between the camera motion branch and the feature pyramid. In the variant without FPN, the C5 and C6 blocks are only used in the camera branch, and thus only experience weight updates due to the camera motion loss. This could mean that the weight updates due to the RoI head losses are detrimental to the camera motion estimation. As a remedy, increasing the loss weighting of the camera motion loss may be helpful.

Instance motion

The object pivots are estimated with relatively high accuracy in both variants (given that the scenes are in a realistic scale), although the FPN variant is significantly more accurate, which we ascribe to the higher resolution features used in this variant.

The predicted 3D object translations and rotations still have a relatively high error, compared to the average actual (ground truth) translations and rotations, which may be due to implementation issues, a non-optimal network architecture, or problems with the current 3D motion ground truth loss (e.g., non-optimal weighting between the components of the motion loss, or between motion and classification losses). Note that the relative error is higher for rotations, which is also the case in the camera motion estimates. The FPN variant is only slightly more accurate for these predictions, which again suggests that there may still be issues with our loss design, loss weighting, or implementation, as one would expect the FPN to yield more accurate motion estimates (as is the case for the pivot estimation).

Instance segmentation

Looking at Figure 8, our instance segmentation results are in some cases still lacking the accuracy seen in the Mask R-CNN Cityscapes [34] results, which is likely due to implementation details.

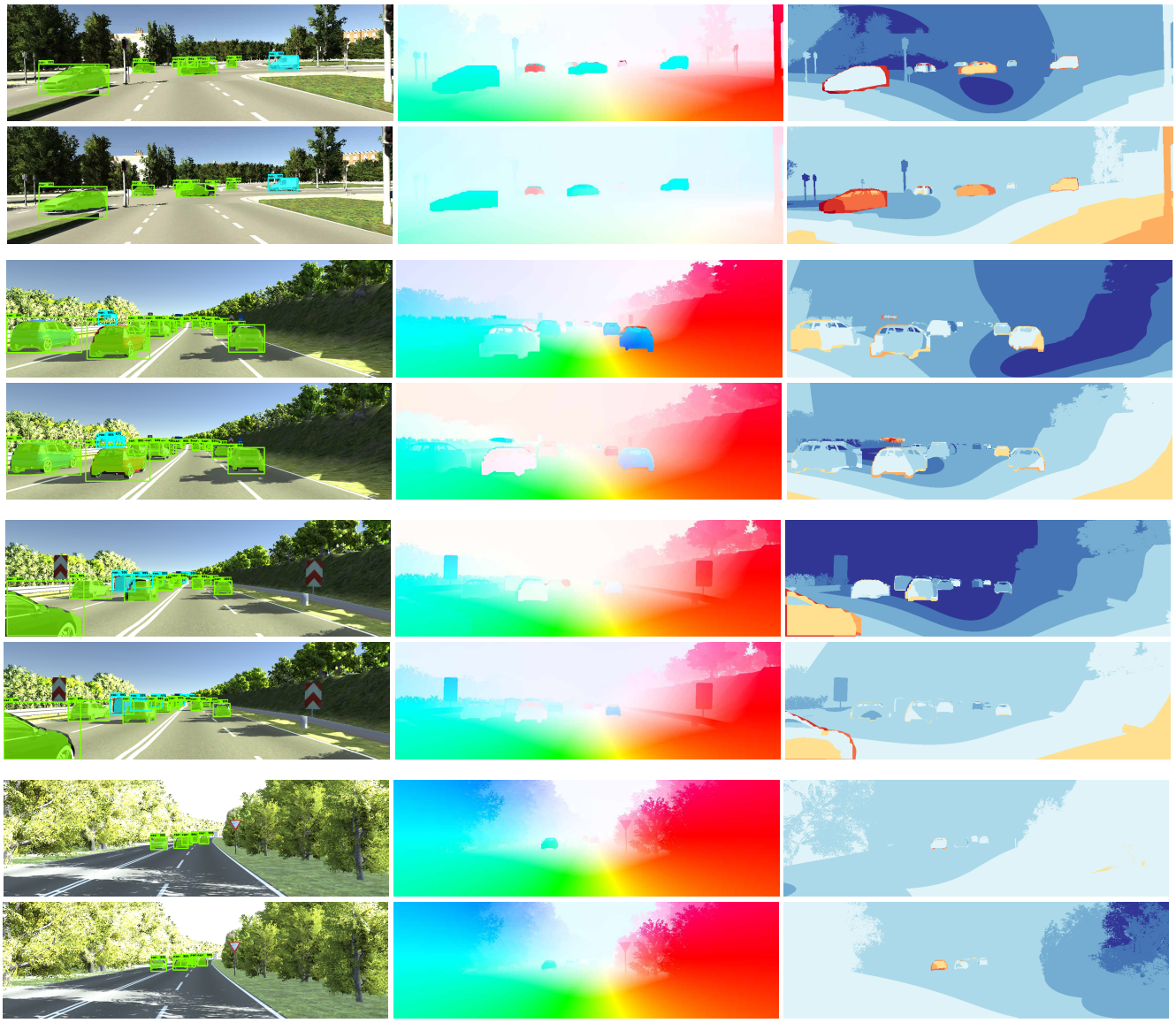

Figure 8: Visualization of results on Virtual KITTI with XYZ input, camera motion prediction and 3D motion supervision. For each example, we show the results with Motion R-CNN ResNet and ResNet-FPN in the upper and lower row, respectively. From left to right, we show the first input frame with instance segmentation results as overlay, the estimated flow, as well as the flow error map. The flow error map depicts correct estimates ($\leq$ 3 px or $\leq$ 5% error) in blue and wrong estimates in red tones.

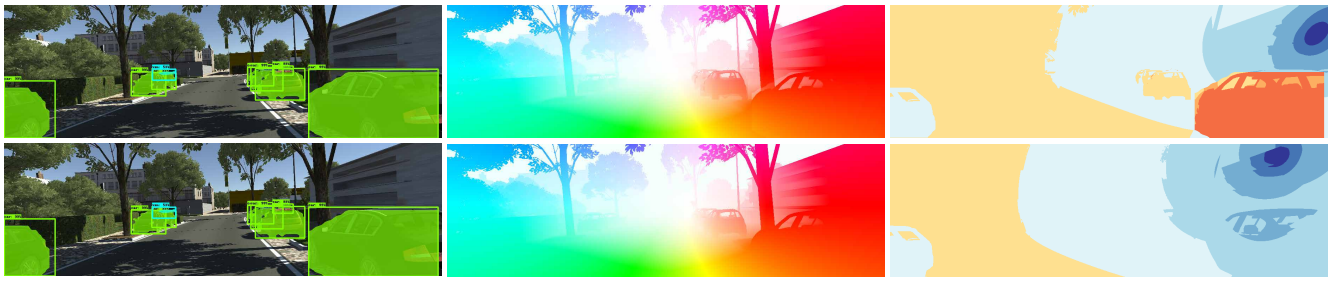

Figure 9: We visually compare a Motion R-CNN ResNet trained without (upper row) and with (lower row) classifying the objects into moving and non-moving objects. Note that in the selected example, all cars are parking, and thus the predicted motion in the first row is an error. From left to right, we show the first input frame with instance segmentation results as overlay, the estimated flow, as well as the flow error map. The flow error map depicts correct estimates (≤ 3 px or $\leq 5\%$ error) in blue and wrong estimates in red tones.

5 Conclusion

5.1 Summary

We introduced Motion R-CNN, which enables 3D object motion estimation in parallel to instance segmentation in the framework of region-based convolutional networks, given an input of two consecutive frames (and XYZ point coordinates) from a monocular camera. In addition to instance motions, our network estimates the 3D ego-motion of the camera. We combine all these estimates to obtain a dense optical flow output from our end-to-end deep network. Our model is trained on the synthetic Virtual KITTI dataset, which provides us with bounding box, instance mask, depth, and 3D motion ground truth, and evaluated on a validation set created from Virtual KITTI. During inference, our model does not add any significant computational overhead over the latest iterations of R-CNNs (Faster R-CNN, Mask R-CNN) and is therefore just as fast and interesting for real time scenarios.

Although our system gives first reasonable instance motion predictions, estimates the camera ego-motion reasonably well, and achieves high accuracy in classifying between moving and non-moving objects, the accuracy of the motion predictions is still not convincing. More work will be thus required to bring the system (closer) to competetive accuracy, which includes trying penalization with the flow loss instead of 3D motion ground truth, experimenting with the weighting between different loss terms, and improvements to the network architecture, loss design, and training process.

We thus presented a partial step towards real time 3D motion estimation based on a physically sound scene decomposition. Thanks to instance-level reasoning, in contrast to previous end-to-end deep networks for dense motion estimation, the output of our network is highly interpretable, which may also bring benefits for safety-critical applications.

5.2 Future Work

Mask R-CNN baseline

As our Mask R-CNN re-implementation is still not as accurate as reported in the original paper [34], working on the implementation details of this baseline would be a critical, direct next step. Recently, a highly accurate, third-party implementation of Mask R-CNN in TensorFlow was released, which should be studied to this end.

Instance motion supervision with the optical flow re-projection loss

We developed and implemented a loss for penalizing instance motions with optical flow ground truth, but could not yet train a network with it due to time constraints. The second next step will be conducting experiments with this loss.

Training on all Virtual KITTI sequences

We only trained our models on the *clone* variants of the Virtual KITTI sequences to make training faster. In the future, it would be interesting to train on all variants, as the different lighting conditions and viewing angles should lead to a more general model.

Evaluation and finetuning on KITTI 2015

Thus far, we have evaluated our model on a subset of the Virtual KITTI dataset on which we do not train, but we have yet to evaluate on a real world dataset. The best candidate to evaluate our complete model is the KITTI 2015 dataset [16], which provides depth ground truth to compose a optical flow field from our 3D motion estimates, and optical flow ground truth to evaluate the composed flow field. Note that with our current model, we can only evaluate on the *train* set of KITTI 2015, as there is no public depth ground truth for the *test* set.

As KITTI 2015 also provides instance masks for moving objects, we could in principle fine-tune on KITTI 2015 train alone. As long as we can not evaluate our method on the KITTI 2015 test set, this makes little sense, though.

Predicting depth

In this work, we focused on motion estimation when RGB-D frames with dense depth are available. However, in many applications settings, we are not provided with any depth information. In most cases, we instead want to work with raw RGB sequences from one or multiple simple cameras, from which no depth data is available. To do so, we could integrate depth prediction into our network by branching off a depth network from the backbone in parallel to the RPN (Table 9). Alternatively, we could add a specialized network for end-to-end depth regression in parallel to the region-based network (or before, to provide XYZ input to Motion R-CNN), e.g. [38]. Although single-frame monocular depth prediction with deep networks was already done to some level of success, our two-frame input should allow the network to make use of epipolar geometry for making a more reliable depth estimate, at least when the camera is moving. We could also extend our method to stereo input data easily by concatenating all of the frames into the input image. In case we choose the option of integrating the depth prediction directly into the current backbone, this would however require using a different dataset for training Motion R-CNN, as Virtual KITTI does not provide stereo images. If we would use a specialized depth network, we could use stereo data for depth prediction and still train Motion R-CNN independently on the monocular Virtual KITTI dataset, though we would loose the ability to easily train the system in an end-to-end manner.

As soon as we can predict depth, we can evaluate our model on the KITTI 2015 test, and also fine-tune on the training set as mentioned in the previous paragraph.

Training on real world data

Due to the amount of supervision required by the different components of the network and the complexity of the optimization problem, we trained Motion R-CNN on the simple synthetic Virtual KITTI dataset for now. A next step will be training on a more realistic dataset, ideally without having to rely on synthetic data at all. For this, we can first pre-train the RPN on an instance segmentation dataset like Cityscapes [22]. As soon as the RPN works reliably, we could execute alternating steps of training on, for example, Cityscapes and the KITTI 2015 stereo and optical flow datasets. On KITTI 2015 stereo and flow, we could run the instance segmentation component in testing mode and only penalize the motion losses (and depth prediction, if added), as no complete instance segmentation ground truth exists. On Cityscapes, we could continue training the instance segmentation components to improve detection and

Layer ID	Layer Operations	Output Dimensions
	input images I_t, I_{t+1}, and (optional) XYZ_t, XYZ_{t+1}	H × W × C
C_6	ResNet (Table 3)	$\frac{1}{64}$ H × $\frac{1}{64}$ W × 2048
	RPN & FPN (Table 5)	
	Depth Network	
	From P_2: 3 × 3 conv, 1024	$\frac{1}{4}$ H × $\frac{1}{4}$ W × 1024
	1 × 1 conv, 1	$\frac{1}{4}$ H × $\frac{1}{4}$ W × 1
	× 2 bilinear upsample	H × W × 1
	Camera Motion Network (Table 7)	
	RoI Head & RoI Head: Masks (Table 5)	
	RoI Head: Motions (Table 7)	

Table 9: A possible Motion R-CNN ResNet-FPN architecture with depth prediction, based on the Mask R-CNN ResNet-FPN architecture (Table 5).

masks and avoid forgetting instance segmentation. As an alternative to this training scheme, we could investigate training on a pure instance segmentation stereo dataset with unsupervised warping-based proxy losses for the motion (and depth) prediction. Unsupervised deep learning of this kind was already done to some level of success in the optical flow setting [29, 44], and was recently also applied to monocular depth networks trained on the KITTI dataset [33].

Supervising the camera motion without 3D camera motion ground truth

We already described a loss based on optical flow for supervising instance motions when we do not have 3D instance motion ground truth, or when we do not have any motion ground truth at all. However, it would also be useful to train our model without access to 3D camera motion ground truth. The 3D camera motion will be already indirectly supervised when it is used in the flow-based RoI instance motion loss. Still, to use all available information from ground truth optical flow and obtain more accurate supervision, it would likely be beneficial to add a global, flow-based camera motion loss independent of the RoI supervision. To do this, one could use a re-projection loss conceptually similar to the one for supervising instance motions with ground truth flow, but computed on the full image instead of for individual RoIs. However, to adjust for the fact that the camera motion can only be accurately supervised with flow at positions where no object motion accurs, this loss would have to be masked with the ground truth object masks. Again, we could use this flow-based loss in an unsupervised way, as long as ground truth instance masks are available. For training on a single dataset without any motion ground truth, e.g. Cityscapes, it may be critical to add this term in addition to an unsupervised loss for the instance motions.

Temporal consistency

A next step after the aforementioned ones could be to extend our network to exploit more than two temporally consecutive frames, which has previously been shown to be beneficial in the context of

classical energy-minimization-based scene flow [8]. In fact, by incorporating recurrent neural networks, e.g. LSTMs [2], into our architecture, we could enable temporally consistent motion estimation from image sequences of arbitrary length.

Bibliography

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. *Backpropagation applied to handwritten zip code recognition*. In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551.

[2] S. Hochreiter and J. Schmidhuber. *Long Short-Term Memory*. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780.

[3] F. Stein. *Efficient Computation of Optical Flow Using the Census Transform*. In: *Pattern Recognition, Proceedings of the 26th DAGM-Symposium*. Ed. by C. Rasmussen, H. Bülthoff, B. Schölkopf, and M. Giese. Vol. 3175. Lecture Notes in Computer Science. Springer, 2004, pp. 79–86.

[4] A. Geiger, P. Lenz, and R. Urtasun. *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Providence, Rhode Island, June 2012, pp. 3354–3361.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. 2012, pp. 1097–1105.

[6] C. Vogel, K. Schindler, and S. Roth. *Piecewise Rigid Scene Flow*. In: *Proceedings of the Fourteenth IEEE International Conference on Computer Vision*. Sydney, Australia, Dec. 2013, pp. 1377–1384.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Columbus, Ohio, June 2014, pp. 580–587.

[8] C. Vogel, S. Roth, and K. Schindler. *View-Consistent 3D Scene Flow Estimation over Multiple Frames*. In: *Proceedings of the 13th European Conference on Computer Vision*. Ed. by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars. Vol. 8692. Lecture Notes in Computer Science. Springer, 2014, pp. 263–278.

[9] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[10] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. *FlowNet: Learning Optical Flow with Convolutional Networks*. In: *Proceedings of the Fifteenth IEEE International Conference on Computer Vision*. Santiago, Chile, Dec. 2015, pp. 2758–2766.

[11] R. Girshick. *Fast R-CNN*. In: *Proceedings of the Fifteenth IEEE International Conference on Computer Vision*. Santiago, Chile, Dec. 2015, pp. 1440–1448.

[12] K. He, X. Zhang, S. Ren, and J. Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. In: *Proceedings of the Fifteenth IEEE International Conference on Computer Vision*. Santiago, Chile, Dec. 2015, pp. 1026–1034.

[13] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In: *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France, July 2015, pp. 448–456.

[14] M. Jadeberg, K. Zisserman, and K. Kavukcuoglu. *Spatial transformer networks*. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. 2015, pp. 2017–2025.

[15] A. Kendall, M. Grimes, and R. Cipolla. *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization*. In: *Proceedings of the Fifteenth IEEE International Conference on Computer Vision*. Santiago, Chile, Dec. 2015, pp. 2938–2946.

[16] M. Menze and A. Geiger. *Object Scene Flow for Autonomous Vehicles*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Boston, Massachusetts, June 2015, pp. 3061–3070.

[17] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. 2015, pp. 2017–2025.

[18] O. Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252.

[19] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. In: *3rd International Conference on Learning Representations*. San Diego, California, 2015.

[20] C. Vogel, K. Schindler, and S. Roth. *3D Scene Flow with a Piecewise Rigid Scene Model*. In: *International Journal of Computer Vision* 115.1 (Oct. 2015), pp. 1–28.

[21] M. Bai, W. Luo, K. Kundu, and R. Urtasun. *Exploiting Semantic Information and Deep Matching for Optical Flow*. In: *Proceedings of the 14th European Conference on Computer Vision*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. Vol. 9910. Lecture Notes in Computer Science. Springer, 2016, pp. 154–170.

[22] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Las Vegas, Nevada, June 2016, pp. 3213–3223.

[23] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. *Virtual Worlds as Proxy for Multi-Object Tracking Analysis*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Las Vegas, Nevada, June 2016, pp. 4340–4349.

[24] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Las Vegas, Nevada, June 2016, pp. 770–778.

[25] J. Hur and S. Roth. *Joint Optical Flow and Temporally Consistent Semantic Segmentation*. In: *4th Workshop on Computer Vision for Road Scene Understanding and Autonomous Driving*. Ed. by G. Hua and H. Jégou. Vol. 9913. Lecture Notes in Computer Science. jointly with ECCV 2016. Springer, 2016, pp. 163–177.

[26] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. *Deeper Depth Prediction with Fully Convolutional Residual Networks*. In: *Proceedings of the International Conference on 3D Vision*. Stanford, California, 2016, pp. 239–248.

[27] N. Mayer, E. Ilg, P. Häusser, 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 Computer Society Conference on Computer Vision and Pattern Recognition*. Las Vegas, Nevada, June 2016, pp. 4040–4048.

[28] L. Sevilla-Lara, D. Sun, V. Jampani, and M. J. Black. *Optical Flow with Semantic Segmentation and Localized Layers*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Las Vegas, Nevada, June 2016, pp. 3889–3898.

[29] J. J. Yu, A. W. Harley, and K. G. Derpanis. *Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness*. In: *1st Workshop on Brave new Ideas for Motion Representations in Videos*. Vol. 9907. Lecture Notes in Computer Science. jointly with ECCV 2016. Springer, 2016, pp. 3–10.

[30] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. *Unsupervised CNN for single view depth estimation: Geometry to the rescue*. In: *Proceedings of the 14th European Conference on Computer Vision*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. Vol. 9912. Lecture Notes in Computer Science. Springer, 2016, pp. 740–756.

[31] A. Behl, O. H. Jafari, S. K. Mustikovela, H. A. Alhaija, C. Rother, and A. Geiger. *Bounding Boxes, Segmentations and Object Coordinates: How Important is Recognition for 3D Scene Flow Estimation in Autonomous Driving Scenarios?* In: *Proceedings of the Sixteenth IEEE International Conference on Computer Vision*. Venice, Italy, Oct. 2017.

[32] A. Byravan and D. Fox. *SE3-Nets: Learning Rigid Body Motion using Deep Neural Networks*. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Singapore, Singapore, 2017, pp. 173–180.

[33] R. Garg, B. V. Kumar, G. Carneiro, and I. Reid. *Unsupervised Learning of Depth and Ego-Motion from Video*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 6612–6619.

[34] K. He, G. Gkioxari, P. Dollár, and R. Girshick. *Mask R-CNN*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017.

[35] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. *Speed/accuracy trade-offs for modern convolutional object detectors*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 3296–3297.

[36] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. *FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 1647–1655.

[37] A. Kendall and R. Cipolla. *Geometric loss functions for camera pose regression with deep learning*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 6555–6564.

[38] A. Kendall, H. Martirosyan, S. Dasgupta, P. H. R. Kennedy, A. Bachrach, and A. Bry. *End-to-End Learning of Geometry and Context for Deep Stereo Regression*. In: *Proceedings of the Sixteenth IEEE International Conference on Computer Vision*. Venice, Italy, Oct. 2017.

[39] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. *Feature Pyramid Networks for Object Detection*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 936–944.

[40] A. Ranjan and M. J. Black. *Optical Flow Estimation using a Spatial Pyramid Network*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 2720–2729.

[41] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki. *SfM-Net: Learning of Structure and Motion from Video*. In: *arXiv preprint arXiv:1704.07804* (2017).

[42] J. Wulff, L. Sevilla-Lara, and M. J. Black. *Optical Flow in Mostly Rigid Scenes*. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017, pp. 6911–6920.

[43] Y. Zhu and S. D. Newsam. *DenseNet for Dense Flow*. In: *Proceedings of the IEEE International Conference on Image Processing*. Beijing, China, Sept. 2017.

[44] S. Meister, J. Hur, and S. Roth. *UnFlow: Unsupervised Learning of Optical Flow with a Bidirectional Census Loss*. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. to appear. New Orleans, Louisiana, Feb. 2018.