Project Report: Unsupervised Motion Segmentation

Jonas Dieker & Christian Hiebl

16/03/2022

Contents

Abstract

This report outlines the outcomes of the final project of the practical course 'Learning For Self-Driving Cars and Intelligent Systems' held at TU Munich's Department of Informatics. The goal of this project was to perform motion segmentation from data captured of a dynamic scene from a moving ego vehicle. As a first step this problem was approached using a supervised deep learning model, utilizing an encoder-decoder architecture. This was then followed by an expansion to make the model unsupervised in order to not require ground truth data which is difficult and expensive to collect. This unsupervised architecture consists of additional networks on top of the supervised network which predict depth, poses and optical flow. With the intention to isolate the effects of different proposed losses we opted to simulate these additional networks by substituting them with ground truth data at first.

Due to the lack of large, high quality datasets we created our own using CARLA [1] and will release this together with the code with which one can generate more sequences. The exact structure of the dataset can be seen in Section 6.1.

Our supervised architecture achieved an IoU of 0.749 on KITTI and 0.722 on our CARLA dataset. Unfortunately, due to the time constraints of this project and multiple problems encountered which are discussed in this report, we have at this current point in time not been able to successfully train the unsupervised model.

1 Motivation

Motion segmentation is the task of classifying each pixel in a frame as either moving or non-moving, examples of which can be seen in Figure 1 and 2. Through this segmentation, the resulting binary mask can be utilized to mask a scene to accurately predict ego motion from a pair of frames as well as to be used for reconstructing the 3D environment. Besides these tasks motion segmentation can be a good prior for object tracking as well as a variety of other tasks.

Up until recently, the common solution for accurate 3D reconstructions was to train a semantic segmentation network to filter out all "movable" objects from predefined classes, i.e. pedestrians, cars, bicycles etc. In this case all potentially moving objects are removed from further processing. In many situations, however, a large number of these potentially moving objects are in fact stationary and thus useful information would be unnecessarily filtered out. Some examples of lost information include vehicles that are parked, vehicles stopped at a red light, or persons sitting. Instead of this it would be advantageous to learn which pixels in a scene move and which do not. This yields a significant performance increase for downstream tasks.

Figure 1: Motion Segmentation Sample From KITTI

Figure 2: Motion Segmentation Sample From CARLA

2 Existing Datasets

The first dataset that we used was KITTI MOD which was created for the paper MODNet [2]. This dataset was automatically generated by projecting the 3D bounding boxes to 2D, associating boxes across frames, computing the velocity vectors and comparing them against the ego vehicle's motion. From this, the moving and non-moving bounding boxes were detected. In total 1950 frames were automatically annotated in this way. Here, the vehicles were not segmented pixel perfectly because of this bounding box annotation approach.

In the FuseModNet paper from 2019 [3], the motion segmentation for the KITTI dataset was extended to Extended KITTI MOD. The authors use the same approach as in MOD Net but in addition refine the output masks by running Mask-RCNN on the segmentation masks to then achieve an improved segmentation ground truth.

In comparison to the Extended KITTI MOD dataset, the Video Agnostic KITTI dataset [4] also includes segmentation of pedestrians besides the vehicle segmentations already present in the other datasets. In 'supervised/dataset/evaluation/sample_load_KITTI.ipynb' we explored the dataset which uses COCO style annotations in a .json file. The category annotations are 'moving' and 'static'. Even though the moving class images have instances for pedestrians and vehicles, these IDs do not contain information about the semantic class. Thus, it is not directly possible to extract the vehicle segmentation without using some kind of Mask R-CNN pre-processing with which we could filter out the pedestrian class. Our reason for filtering out the pedestrian class in the first place is that (1) in this way we can expand Extended KITTI MOD to include more training samples and (2) we wanted to facilitate the network's training by using only rigid motion objects.

Even though motion segmentation datasets exist, such as the ones discussed above, there are many problems with them. First, the ground truth annotation provided is inaccurate for the Extended KITTI MOD dataset, as can be seen in 3. Secondly, there is in general too little data which is crucial for the network's generalization. In KITTI MOD, there are only 1950 images which is extended to more than 10,000 in the other datasets. However, many of these images contain a completely static ground truth motion mask, which is necessary for the network to learn that not all cars are moving. However, the large number of fully static scenes is sub-optimal for training. Additionally, the ratio of dynamic-static pixels per image is presented and discussed in more detail in Section 5.2. These reasons led us to the conclusion to create our own dataset in CARLA for which we have control over the quality of the data and annotations, where static cars will be placed and we can generate a lot more training samples if need be, ad hoc.

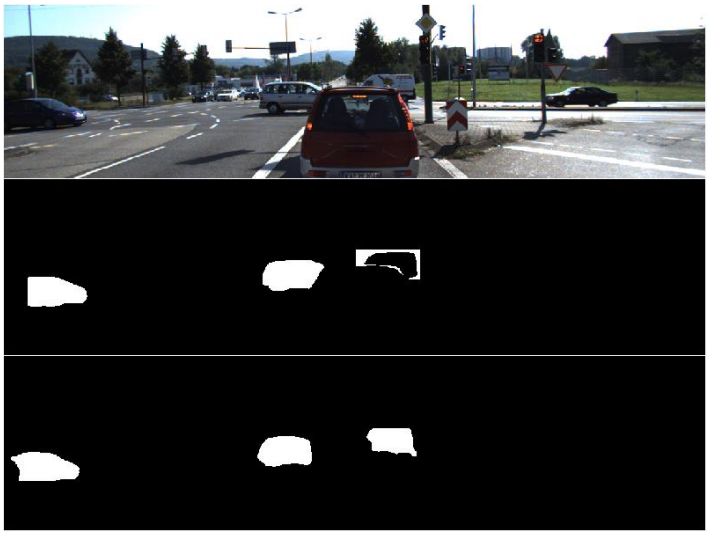


Figure 3: The first image shows a sample RGB frame of the Extended KITTI MOD dataset. The second image depicts an incorrect ground truth annotation with inverse motion segmentation for the white van in the intersection and a cropped segmentation for the car on the left border of the frame. The third image shows the prediction of the supervised motion segmentation network, which does not crop the car on the left and correctly segments the moving white van.

3 CARLA Dataset Generation

In order to use CARLA to its fullest potential, we had to locally build CARLA 0.9.13 from source. We initially tried using the CARLA install provided on the server but came to the conclusion that we would be restricted by the possibilities of the map as well as the missing additional sensors that CARLA 0.9.13 provides. Building the packaged version of 0.9.13 locally was a relatively short and simple task. This would however not enable us to modify the world maps since these would need to be compiled before being used in the CARLA server. Therefore we ended up installing CARLA 0.9.13 from source. Even though CARLA is officially only supported for Ubuntu 18.04, we managed to get it to run locally on Ubuntu 20.04. After troubleshooting on CARLA's Github forum and Stackoverflow solutions we finally successfully installed it with a different clang compiler, changing the boost version from 172.0 to 169.0 to be used for the source build as well as adding the "CPLUS_INCLUDE_PATH" to use the conda specific install.

Despite of how tedious and strenuous this process was, in the end it enabled us to modify the CARLA maps. Through the Unreal Engine Editor, we placed existing CARLA car models inside the Town03 world map. Around 150 cars were strategically placed on the map in order for the ego vehicle to often encounter vehicles on the side of the road, in driveways or in parking lots. Through Unreal Engine's texture appearance editor we could also quickly change the parked vehicles' colors in order to increase the variety of visual input, since we are limited to around 25 existing vehicle CAD models. Additionally the cars on the sidewalk were placed far enough on the walkway to not impede the autopilot's route on the lane directly neighboring the parked car.

We decided to embed the actors on the world map since (1) we didn't need to directly modify the parked cars from the client script (2) we could save computational efficiency by only spawning the ego vehicle and NPCs on the road instead of spawning also all the parked vehicles and (3) we could assign a specific semantic label to the parked cars to simplify our motion segmentation ground truth generation. A sample image of the semantic segmentation for the parked cars/cars on the road is seen below.

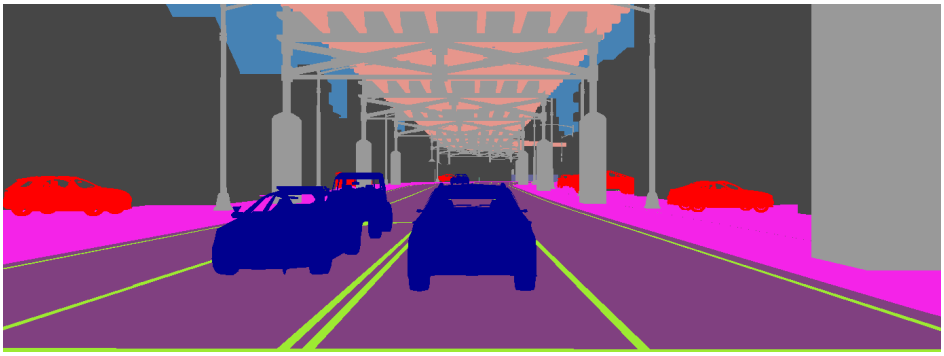


Figure 4: Semantic Segmentation with Parked Cars as Red Labels and Movable Cars on the Road as Blue Labels

Now that the static cars and potentially moving cars on the road were segmented differently, we still needed to check each actor's velocity to obtain the final motion segmentation mask. In situations where cars are stopped at a red light, waited at the roundabout or at a stop sign, the label of these vehicle's pixels needs to be static. Besides our generated motion segmentation, in CARLA 0.9.13 we were able to record ground truth optical flow which is important for our unsupervised network. Below is a sample composition of the sensor recordings.

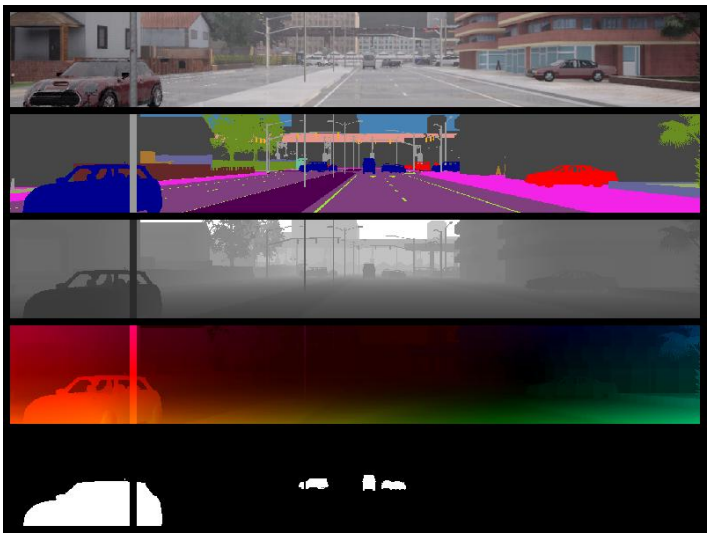
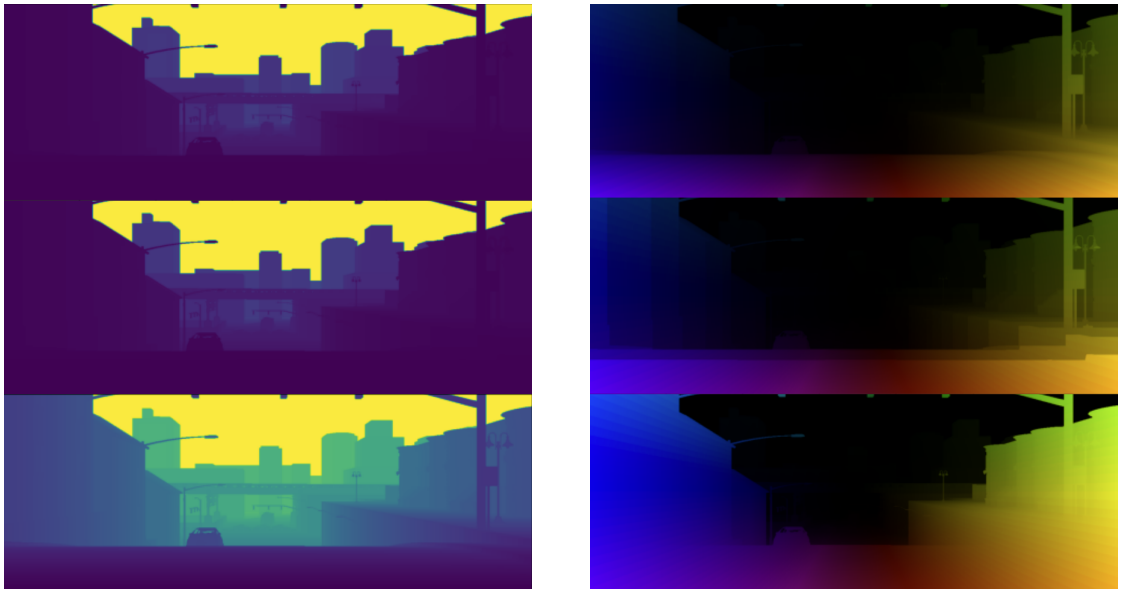


Figure 5: Sample Sensor Recordings from CARLA

In order to obtain the best possible results, we saved the depth images with CARLA's color converter's "raw" setting and not with the other possible conversions, namely "depth" or "logarithm depth" conversions. The depth images with the different depth saving variants are seen below in Figure 6a.

Using these depth maps and provided poses, the static flow can be computed, detailed in Section 5.3. The visualizations of the resulting static flow maps with the different depth saving approaches are shown below in Figure 6b. The most ideal option is the "raw" saving which only shows slight discretization lines in the z-axis (into the paper). The second visualization is with the "depth" conversion. This conversion option is the worst of the three since it has great discretization jumps in the depth which appear to be after every five meters. The "log-depth" has the same small discretization lines, which are visible in all three saving techniques. For closer distances, the "log depth" saving has the best resolution but for farther away objects, the "raw" saving technique is still better which is why we decided to record and use the "raw" saving approach.



(a) Depth Maps (b) Optical Flow

Figure 6: From top to bottom: "Raw" Sensor Saving, "Depth" Conversion Sensor Saving, "Log Depth" Conversion Sensor Saving

4 Supervised Network

In order to compare performance to the unsupervised motion segmentation by setting a baseline, we started with a rather simplistic architecture to be directly trained with ground truth motion segmentation masks. For this we used the autoencoder-style network U-Net [5] which receives a stacked pair of RGB images as input and outputs the binary motion segmentation mask prediction, shown in Figure 7.

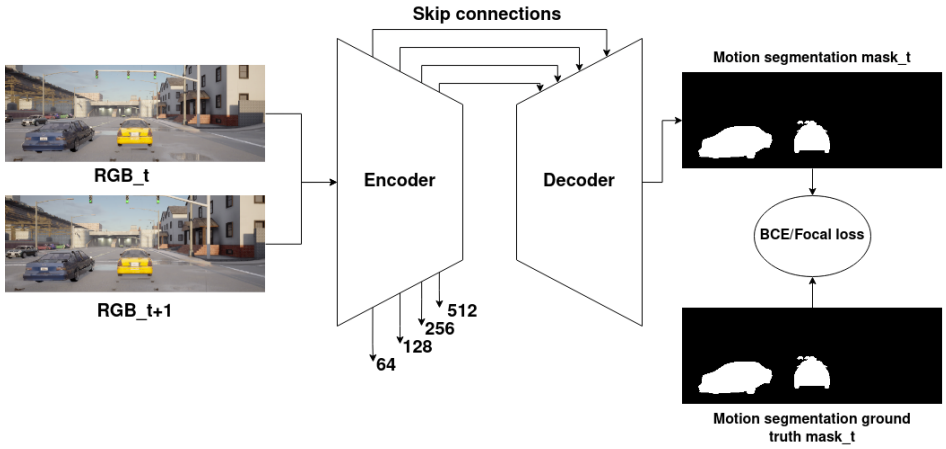


Figure 7: Supervised Network Architecture Structure

In Figure 8 we show the number of ground truth images which fall into specific bins for dynamic-pixel to static-pixel ratios. It is apparent that static pixels heavily outweigh dynamic pixels, meaning that there exists a large class imbalance in the ground truths.

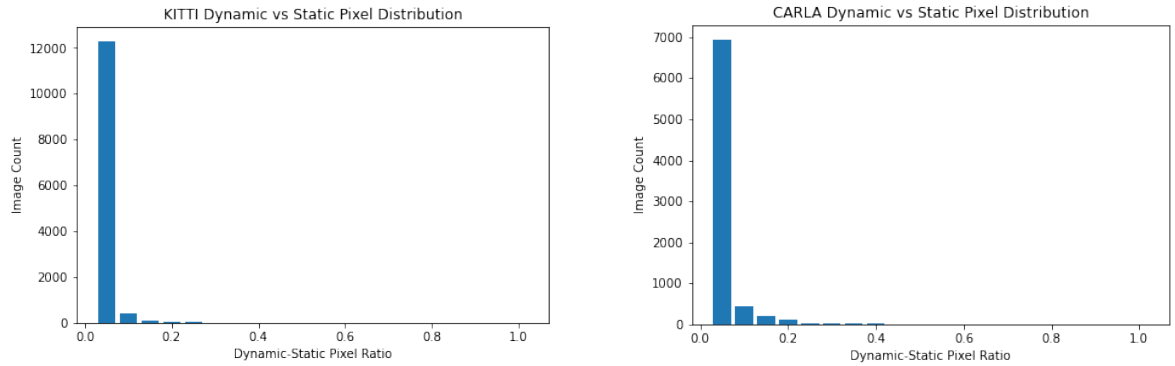


Figure 8: Dataset Distributions According to Dynamic-Static Pixel Ratio

Due to these results, we decided to switch from binary cross entropy (BCE) loss to Focal Loss [6] formulation which assigns a lower loss to the easy to classify pixels and a greater loss to difficult to classify pixels. After applying grid search we came to the conclusion that the standard parameters used in the Focal Loss paper were ideal with $\gamma = 2$ and $\alpha = 0.25$. The behaviour of focal loss with varying γ is shown in Figure 9.

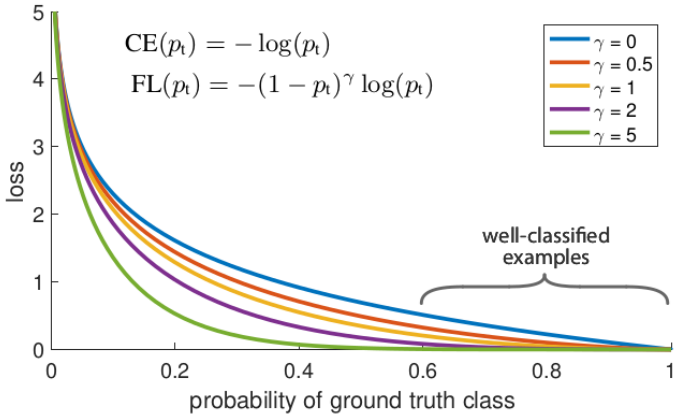


Figure 9: Focal Loss [6]

5 Unsupervised Network

The task of motion segmentation is a complex problem requiring dense, continuous-valued ground truths which are either impossible or very expensive to collect. For example, there is no sensor that can directly measure optical flow. As a result it makes logical sense to let a model generate its own ground truth by letting it solve multiple coupled sub-problems. In Competitive Collaboration (CC) [7] the authors learn single view depth, camera motion, optical flow and motion segmentation jointly.

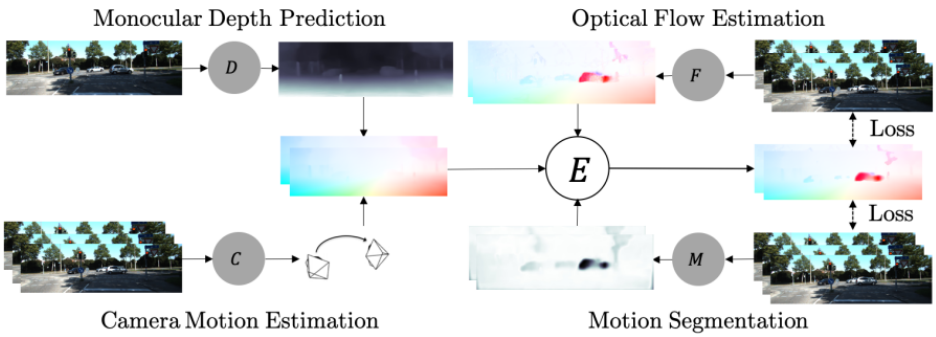


Figure 10: Competitive Collaboration Schematic [7]

The network consists of three parts, two competitors R=(D,C) and F and a moderator M. The moderator M assigns what it believes to be static pixels to R and dynamic pixels to F. In the first step R and F are trained on those pixels. In the second step R and F collaborate to train M.

The moderator's ability to determine whether a pixel is static or dynamic is learned by a specific BCE loss called the consensus loss. For this loss the prediction term is the probability of a pixel being static and the class ground truth is the result of two indicator functions. Meanwhile, a second loss term E_M biases the pixels to be classified as static.

5.1 Proposed Unsupervised Losses

For our purposes the consensus loss is of the most interest as it constrains masks to segment moving objects. For each pixel a BCE loss H is taken, where m is the probability of a pixel being static and the first term in H is the self-generated class label for the pixel. As can be seen in Equation 1, the class label depends on two indicator functions. The first indicator function depends on the photometric error based on the static scene flow being smaller than the photometric error based on the dynamic flow and the second indicator function depends on the relative size of both flows.

$$E_C = \sum_{\Omega} H(\mathbb{I}_{\rho_R < \rho_F} \vee \mathbb{I}_{\|\nu(e,d)-u\|<\lambda_c}, m) \tag{1}$$

where $\nu(e,d)$ is the static scene flow and u is the dynamic scene flow.

The photometric error and flow difference indicator functions, however, do not take into account the consistency of the 3D structure of the scene. We therefore propose to additionally evaluate the geometric errors. Using the flows we can find pixel correspondences in a pair of images and use the depth maps to project the two sets of pixels into 3D to calculate the geometric error based on static scene flow g_R and on dynamic flow g_F. This is shown in Equation 2.

$$E_C = \sum_{\Omega} H(\mathbb{I}_{\rho_R < \rho_F} \vee \mathbb{I}_{\|\nu(e,d)-u\|<\lambda_c} \vee \mathbb{I}_{g_R < g_F}, m) \tag{2}$$

Furthermore, other variations of this loss might also prove to yield increased performance. We therefore also propose another formulation of the loss in Equation 3. Instead of comparing the two geometric errors g_R and g_F we now compare each of the geometric errors to some hyper-parameter λ_{geo}.

$$E_C = \sum_{\Omega} H(\mathbb{I}_{\rho_R < \rho_F} \vee \mathbb{I}_{\|\nu(e,d)-u\|<\lambda_c} \vee \mathbb{I}_{\|p-\tilde{p}\|<\lambda_{geo}}, m) \tag{3}$$

where $p = \pi^{-1}(p')$ and $\tilde{p} = \pi^{-1}(w_c(p', \nu(e,d)))$ in the static scene flow case or $\tilde{p} = \pi^{-1}(w_f(p', u))$ in the dynamic flow case.

A final extension to all loss variations mentioned above, would be to use Focal Loss instead of BCE for H. As will be discussed later in the report, focal loss yields some performance increase over simply using BCE.

5.2 Proposed Unsupervised Network Architecture

With the goal to test different loss hypotheses it was decided to at first simulate the networks R and F. This enables us to test different losses independently of the performance of the aforementioned networks. We started with the loss formulation in 2 since this alleviates setting an additional hyper-parameter which could potentially greatly influence our network's performance. The proposed architecture, including the simulated networks, is visualized in Figure 11.

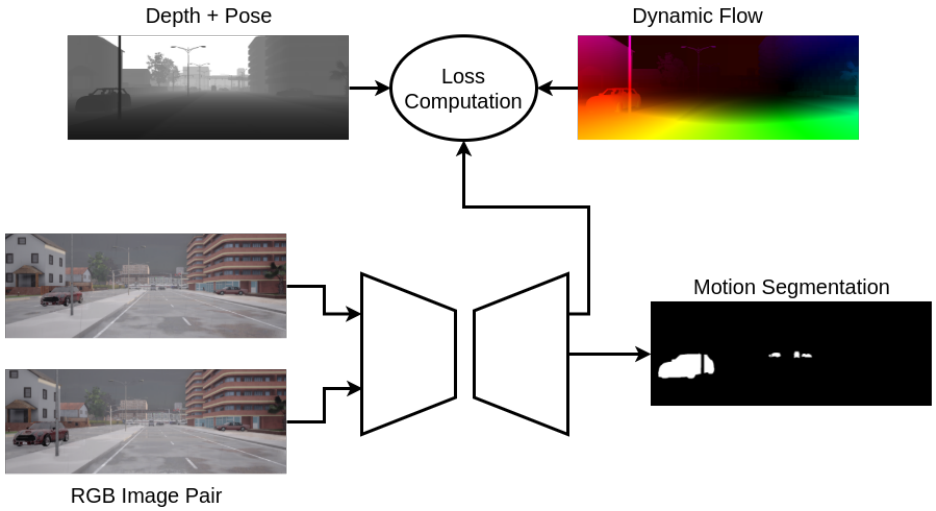


Figure 11: Unsupervised Architecture

One source of error that persists even when simulating sub-networks with ground truth data as described above, is that there are always potential occlusions. In Figure 12 the lamp post covers part of the background in both images. When computing the static scene flow between these two images, which use depth maps and poses of each scene, incorrect flows are calculated for the occluded pixels which in actuality have no correspondences. This leads to errors not only in the static scene flow, but also to errors in the photometric and geometric errors because they directly depend on the static scene flow.

A potential solution to this problem is to explicitly model the occlusions and then use the occlusion masks to mask out pixels which are responsible for these incorrect flow vectors. One option would be to use occlusion aware optical flow [8] which does exactly what was outlined above.

Figure 12: No Pixel Correspondences for Occluded Background Areas

5.3 Data Pre-Processing

The static scene flow required for our proposed network cannot be generated directly by sensors that CARLA provides. Instead this is an extra processing step performed pre-training. The flow is computed using the depth map and the poses which can be directly generated using CARLA. The generation of the backward static scene flow involves re-projecting points in the second image into 3D, transforming the 3D points according to the pose difference and projecting the 3D points into the first image.

To match the dynamic flow recorded directly in CARLA we needed the backward flow. Since the recorded poses are in ego-frame we need an additional transformation. The first step is to transform the points from the second sensor-frame to ego-frame. Then we apply the transformation from ego-to-ego and finally we can transform back from ego-frame into the first sensor-frame. This then gives the final transformation between the re-projected sets of points, shown in Equation 4.

$$_{sensor1}T_{sensor2} \;=\; _{ego}T^{-1}_{sensor}\; _{ego1}T_{ego2}\; _{ego}T_{sensor} \tag{4}$$

where the transformation from sensor to ego is defined as:

$$_{ego}T_{sensor} \;=\; RotY(90)RotZ(90) \tag{5}$$

When comparing Figure 13 and Figure 14, it can be seen that the static scene flow is being generated correctly since the flows visually seem to be identical for the static scene. One thing which can be observed however, is that the dynamic flow contains some structured artifacts. This is likely a bug in CARLA's optical flow sensor since this same pattern appears in all optical flow sensor recorded images. We unsuccessfully attempted to find the exact cause for this problem in the CARLA repository.

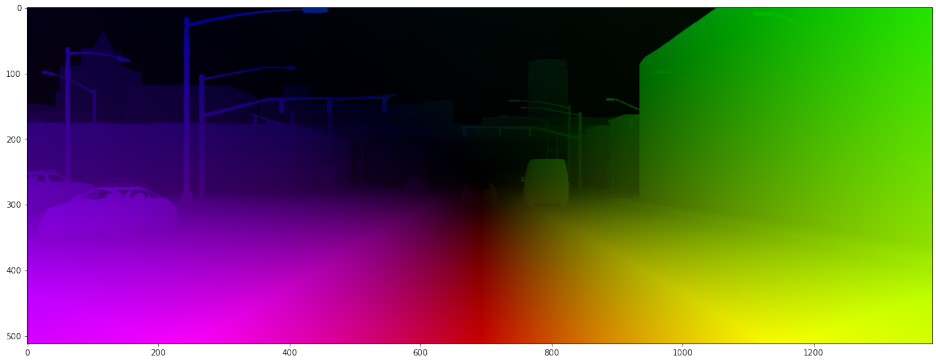


Figure 13: Static Scene Flow Render

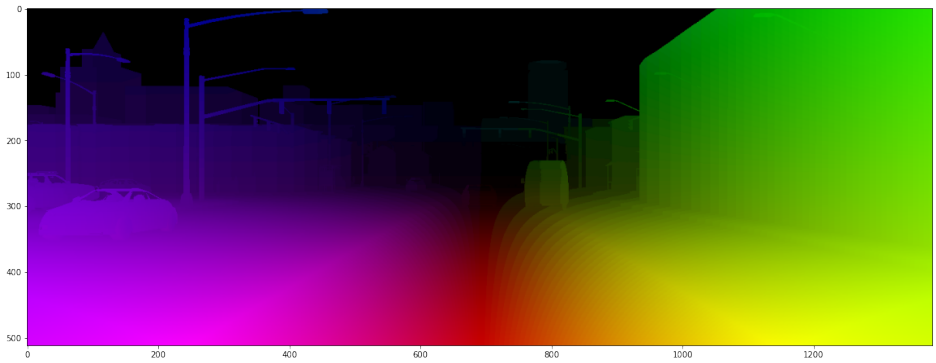


Figure 14: Dynamic Flow Render

In order to visualize the difference between the two flows, we took the L2 norm between the flow vectors and cut-off all vectors smaller than length 1, this is shown in Figure 15 (left). We also plot the motion segmentation ground truth in the same figure on the right to compare against. The most striking difference between the plots are the structured artifacts mentioned before which are in the right half of the left plot. Interestingly, the closest car (on the left) is almost perfectly segmented while the two other moving cars, one directly behind the closest car and one in the center of the image, can not be seen.

Figure 15: Plotting Flow Difference vs Motion Segmentation Ground Truth

The left plot is essentially the second indicator function in our proposed loss 2. The fact that the more distant cars are not shown in the optical flow differences justifies the additional indicator functions because from the flow difference alone we cannot reason about distant moving objects in the scene.

6 Results and Discussion

6.1 Final CARLA Dataset

We generated a dataset which contains all the ground truth data required to do motion segmentation for both supervised and unsupervised approaches. The dataset is located at */storage/remote/atcremers40/motion_seg/datasets/CARLA* and the directory structure is as shown below:

```
/
├── depth
│   ├── 0000
│   │   ├── 0000.png
│   │   ├── 0001.png
│   │   └── ...
│   ├── 0001
│   └── ...
├── images
├── motion_segmentation
├── opt_flow
├── semantic_segmentation
├── static_flow
└── transformations
```

One currently still persisting problem is that occasionally cars are not rendered properly in CARLA. We suspect this is due to the performance of the computer we locally installed and ran CARLA from. We attempted to make our data generation more efficient by only checking for velocity of cars controlled by the traffic manager as well as only the actors in a manually specified radius of 75m from the ego vehicle. Additionally introducing a new semantic class for static cars and by simply running the simulator at a lower FPS solved this problem only partially. An example of one of these rendering issues is shown in Figure 16 where a car in the center of the image has no rendered roof.

Figure 16: CARLA Rendering Issues with Car in Center

Additionally, since our ground truth is not manually labeled or generated by some learned network, the annotations for the motion segmentation are very precise. An example of such an annotation is in Figure 17 with a cropped version in Figure 18. Even with the naked human eye it would be difficult to detect the motion of objects at this great distance. Since the annotation is pixel perfect, it becomes very difficult for our network to segment all moving instances since it needs to pay attention to every single pixel in the scene, unlike in KITTI where the dynamic pixels are more clustered. It also begs the question whether segmenting moving cars at such a great distance is important. As will be discussed in Section 6.2.1, different metrics were compared and one new metric was proposed in order to present this behavior in a more meaningful way. However, it was decided to use regular IoU, as it is still a powerful indicator of performance while also being widely accepted in the computer vision community.

Figure 17: Very precise ground truth annotation by CARLA

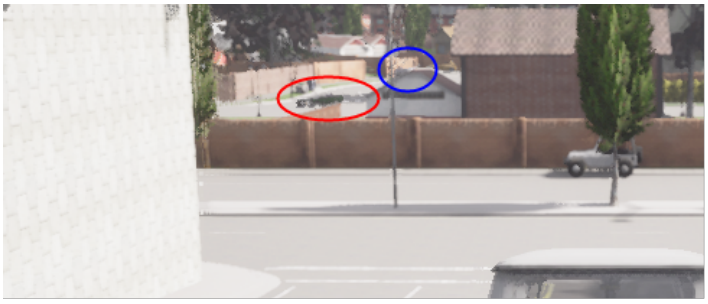


Figure 18: Cropped version of Figure 17 - The vehicles circled in red and blue are the small ground truth annotations. The car circled in blue is just disappearing behind the house's roof and barely visible even with the crop.

6.2 Supervised Network

As mentioned in Section 2 and apparent from Figure 8 there is a great class imbalance between dynamic and static pixels in our data. In order to improve the network's performance we changed our loss from BCE to Focal Loss. The results are presented in Table 1.

	BCE	Focal Loss
KITTI	0.733	0.749
CARLA	0.676	0.722

Table 1: Quantitative Performance - Inference (IoU)

For KITTI, the network achieved an IoU gain of only 0.016 which was measurable but not substantial. In contrast, training the network with the adapted loss on CARLA yielded an IoU improvement of 0.046 which is quite significant. We think this difference in performance boost is due to the pixel perfect annotations in CARLA which even contain the tiniest unoccluded corners of moving cars which are not segmented in the KITTI ground truth. Samples of these ground truth annotations can be seen in Figure 17 as well as Figure 19.

In general, performance on CARLA is worse than on KITTI. We suspect the reason for this is that CARLA data is harder to learn due to much more detailed ground truths. Additionally, CARLA performance is more indicative of the network's actual performance on CARLA than for KITTI. This is because even though we have a good IoU score for KITTI, due to errors in the ground truth for KITTI, evaluation on real ground truths may yield a worse result.

6.2.1 Metrics

During the project we tried different metrics in order to find one that best captures the performance of our network. One problem we found with regular IoU was that the pixel perfect ground truth masks produced by CARLA can result in zero IoU in some almost entirely static scenes as shown in the lower row in Figure 19. In the Figure both predictions (center column) are qualitatively good. However, the top prediction results in an IoU score of 0.93 while the bottom row yields an IoU score of 0.0. This metric thus fails to reflect the network's performance.

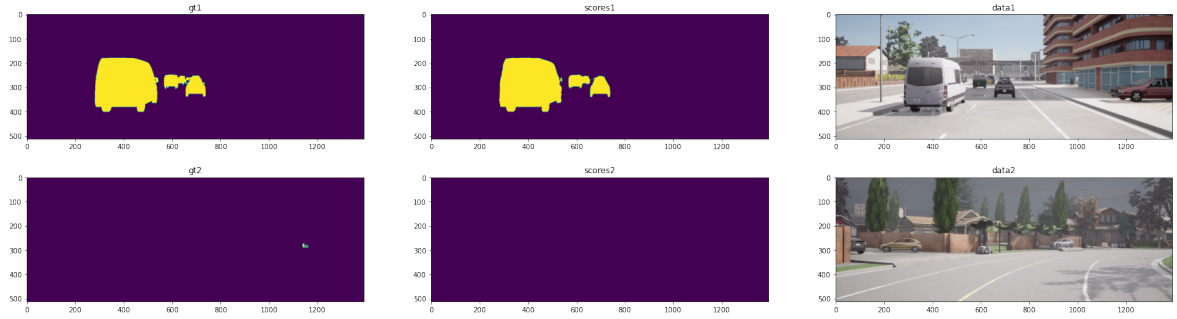


Figure 19: Example of Good Prediction but Bad IoU Score

We used one additional metric, mean IoU (mIoU) and proposed "aggregated IoU" both of which were supposed to reflect the network's performance better. In mIoU we calculate the IoU of the dynamic pixels and IoU of the static pixels and take the average over both classes. Since most frames in CARLA and KITTI have predominantly static pixels, a network that predicts the whole scene as static would already achieve an mIoU of about 0.5 which is why this metric might not be an ideal measurement after all. In the aggregated IoU we aggregate all moving and static pixels across the batch and jointly calculate the intersection and the union for the whole batch. However in our case, being limited to a batch size of 2, the probability of this problem still occuring over the whole batch was still fairly high. Therefore this loss was ultimately deemed to not be a significant improvement over regular IoU.

	IoU	aggregated IoU	mIoU
KITTI	0.749	0.730	0.871
CARLA	0.722	0.744	0.856

Table 2: Quantitative Performance of Motion Segmentation Network
IoU - IoU of Dynamic Pixel Class only
Aggregated IoU - IoU of Dynamic Pixel Class over Batch
mIoU - Average of IoU for Dynamic and Static Pixel Classes

6.2.2 Cross-Dataset Training and Inference

In order to test the generalization of our trained network we compared the variations of training and running inference on KITTI and CARLA, see Table 3.

	training KITTI	training CARLA
inference KITTI	0.749	0.078
inference CARLA	0.217	0.722

Table 3: Quantitative Performance of Motion Segmentation Network (IoU)

It is curious to see that the two off-diagonal terms differ so strongly. One would think that the domain gap for training on KITTI + testing on CARLA is about the same as training on CARLA + testing on KITTI. We suspect this is because the network trained on KITTI can generalize better with the coarse ground truth outline annotations, whereas the network trained on CARLA learns to detect objects in the distance which are not in the ground truth annotations in KITTI.

Below are qualitative results of ground truth motion segmentation annotation, motion segmentation prediction and the accompanying RGB image for the regular training and testing on the same dataset, KITTI and CARLA respectively.

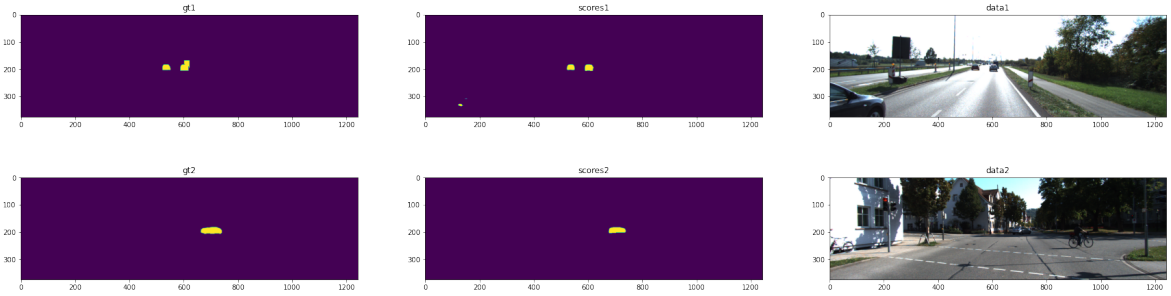


Figure 20: Qualitative Results KITTI

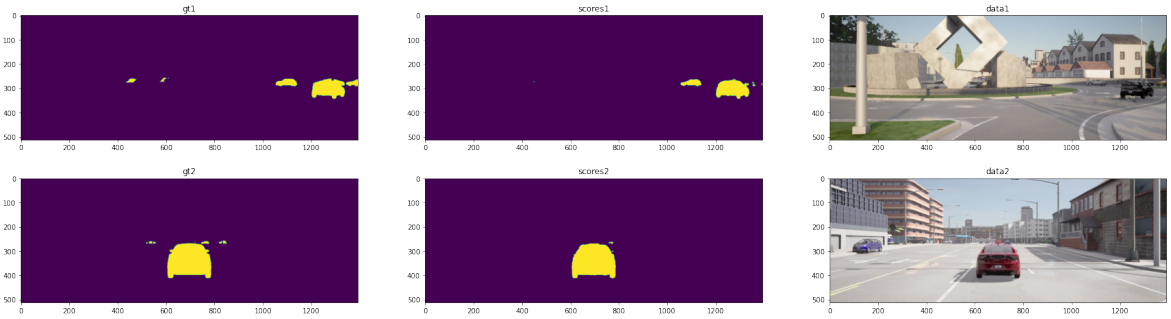


Figure 21: Qualitative Results CARLA

6.3 Unsupervised Network

At this point in time we are still having problems with learning the motion segmentation as outlined in section 5.2. Overfitting to a single data instance only yields partially consistent results after trying a number of variations of losses and hyper-parameters.

As a first attempt we used our proposed loss from Equation 2 and hyper-parameters of $\lambda_C = 0.3$ and $\lambda_M = 0.005$. We then performed grid search to optimize these hyper-parameters but overfitting was not possible. Next, we tried the original loss from Competitive Collaboration and its loss function, Equation 1 and optimized hyper-parameters in a similar way to before, again unsuccessfully. In Figure 23a we show our best attempt at overfitting using the accompanying RGB image pairs in Figure 22, where in addition to segmenting the cars, the network also segments large parts of the surrounding buildings.

Figure 22: Sample Image CARLA

(a) Unsupervised Network Prediction (b) Ground Truth

Figure 23: Motion Segmentation

Results like the one above would suggest that the E_M term needs to be weighted higher as this term increases a pixel's bias to be classified as static. However, when manually increasing the term's weighting, the predicted motion segmentation would almost instantly converge to zero.

A reason for this sub-optimal performance could be that the CARLA optical flow discretization error causes some problems in the consensus loss. Additionally, since there are many hyper-parameters to tune for in this loss formulation, it could also be that we have not found the optimal set yet.

7 Future Work

Since we are currently training our motion segmentation network with a rather simplistic U-Net architecture it could be improved by not concatenating the RGB input images but instead using shared convolutional layers and then merging the feature vectors with a fixed correlation layer, similar to the approach in FlowNet-C [9]. In this way, the network learns a good feature representation for each image in the first few layers which is then combined later in the correlation step which directly extracts correspondences. As opposed to when concatenating the RGB images, the network needs to learn the feature correspondences between the images using convolutions only.

As mentioned in Section 5.2 a possible extension to the dynamic flow would be to utilize occlusion aware optical flow [8]. We would still simulate network R, by passing in ground truths for poses and depths, but now we would learn optical flow for the F network. Importantly, this allows us to test the effects the occlusion aware optical flow has versus the normal optical flow in isolation, without introducing more variables by also learning the static scene flow.

The final step is to learn all parts of the network. At this point we are able train this on CARLA as well as KITTI, since before this was not possible as the KITTI dataset includes no optical flow or pixel-wise depth information. A performance comparison between synthetic data from CARLA and real data from KITTI can then be made.

Due to time constraints we were not able to try out these different extensions. However, for most of the extensions, for example FlowNet-C for the optical flow, there exist pre-trained networks which we could integrate into our existing training. Since the losses and general training, validation, metrics, logging parts are written, this addition would entail adding some lines of code in the training loop to pass data trough the additional networks. One additional step would be that the static scene flow would now need to be calculated online instead of being a pre-processing step as before. A potential problem might be however, that we would not have enough VRAM on even the most powerful remote machine available to us, which has 12 GB of VRAM, to run the full model, as even with the current setup we require between 8 and 9 GB of VRAM.

References

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. [Online]. Available: http://arxiv.org/abs/1711.03938

[2] M. Siam, H. Mahgoub, M. Zahran, S. Yogamani, M. Jagersand, and A. El-Sallab, "Modnet: Motion and appearance based moving object detection network for autonomous driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 2859–2864.

[3] H. Rashed, M. Ramzy, V. Vaquero, A. El Sallab, G. Sistu, and S. Yogamani, "Fusemodnet: Real-time camera and lidar based moving object detection for robust low-light autonomous driving," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[4] M. Siam, A. Kendall, and M. Jagersand, "Video class agnostic segmentation benchmark for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2825–2834.

[5] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[6] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[7] A. Ranjan, V. Jampani, K. Kim, D. Sun, J. Wulff, and M. J. Black, "Adversarial collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation," *CoRR*, vol. abs/1805.09806, 2018. [Online]. Available: http://arxiv.org/abs/1805.09806

[8] Y. Wang, Y. Yang, Z. Yang, L. Zhao, and W. Xu, "Occlusion aware unsupervised learning of optical flow," *CoRR*, vol. abs/1711.05890, 2017. [Online]. Available: http://arxiv.org/abs/1711.05890

[9] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.