# Feature Map Transformation for Multi-sensor Fusion in Object Detection Networks for Autonomous Driving

Enrico Schröder[1(✉)], Sascha Braun[1], Mirko Mählisch[1], Julien Vitay[2], and Fred Hamker[2]

[1] Department of Development Sensor Data Fusion and Localisation, AUDI AG, 85045 Ingolstadt, Germany
{enrico.schroeder,sascha-alexander.braun,mirko.maehlisch}@audi.de
[2] Department of Computer Sciences, Technical University Chemnitz, Straße der Nationen, Chemnitz, Germany
{julien.vitay,fred.hamker}@informatik.tu-chemnitz.de

**Abstract.** We present a general framework for fusing pre-trained object detection networks for multiple sensor modalities in autonomous cars at an intermediate stage. The key innovation is an autoencoder-inspired Transformer module which transforms perspective as well as feature activation characteristics from one sensor modality to another. Transformed feature maps can be combined with those of a modality-native feature extractor to enhance performance and reliability through a simple fusion scheme. Our approach is not limited to specific object detection network types. Compared to other methods, our framework allows fusion of pre-trained object detection networks and fuses sensor modalities at a single stage, resulting in a modular and traceable architecture. We show effectiveness of the proposed scheme by fusing camera and Lidar information to detect objects using our own as well as the KITTI dataset.

**Keywords:** Autonomous driving · Perception · Sensor fusion · Object detection · Lidar

## 1 Introduction

One of the most important tasks in automotive perception is detecting objects and obstacles like cars or pedestrians in the vehicle's surroundings. For reasons of performance and robustness, autonomous cars combine multiple sensor modalities such as camera, Lidar- and Radar sensors to exploit individual strengths of each sensor type.

Traditionally, discrete objects detected by each sensor are fused via a model-based Bayesian Framework. Recently, methods based on convolutional neural networks have been used to detect objects not only in raw images [9] but also


K. Arai and S. Kapoor (Eds.): CVC 2019, AISC 944, pp. 118–131, 2020.
https://doi.org/10.1007/978-3-030-17798-0_12

Lidar pointclouds [10,14], outperforming methods based on traditional handcrafted features. This makes it viable to explore methods which fuse sensor modalities at a lower level of abstraction directly within these object detection networks to potentially make use of additional information that has not yet been abstracted away by an underlying model. Our method achieves this by fusing feature maps of pre-trained object detection networks, while still providing some of the benefits of a traditional late-fusion scheme such as modularity and traceability.

## 2 Related Work

In contrast to traditional image classification networks, object detection networks predict position and class of *multiple objects* in the frame. Most object detection networks first feed input images through a convolutional *feature extractor* network such as ResNet [5] or VGG [11]. The resulting feature maps are then passed through a combination of convolutional and fully-connected layers (the *detector*) to generate object bounding boxes and classes. In the context of autonomous driving we are especially interested in multi-modal variations of these object detection networks, detecting objects using both camera and Lidar sensors. Research in this area has made significant progress since introduction of the 2D bird view and 3D object detection challenges to the automotive KITTI dataset [3], providing labeled camera and Lidar data.

Most approaches implement involved network architectures with cross connections between the individual sensor paths at multiple locations. **Frustum PointNets** [8] do object detection on the ego-view camera image and then, for each object proposal, generate a 3D perspective frustum which defines a region-of-interest in the 3D point cloud. This region-of-interest is then fed to a segmentation network to generate 3D objects. While this approach performs well, it relies on camera alone to detect objects in the first place and then only enhances class and depth prediction by use of the Lidar pointcloud. **Multi-View 3D Object Detection** (MV3D) [1] projects Lidar pointclouds into front view and bird view images and generates 3D object proposals from bird view. These 3D proposals are then used to generate proposals for the projected views and to extract candidate regions from the individual feature maps via region-of-interest pooling (see [4]). Feature map region proposals from the three input views (front view Lidar, bird view Lidar and camera) are then fused via concatenation and fed to an object detector which outputs 3D bounding boxes. **Aggregate View Object Detection** (AVOD) [7] is similar, but additionally fuses input data before region proposal generation. Both MV3D and AVOD, while performing well, form quite complex architectures with multiple links between the two sensor modalities at different layers of abstraction. These architectures strictly require both sensors to be operational and hence would not be able to deal with sensor failure. Additionally, the complex nature of these architectures makes it difficult to analyze system behavior.

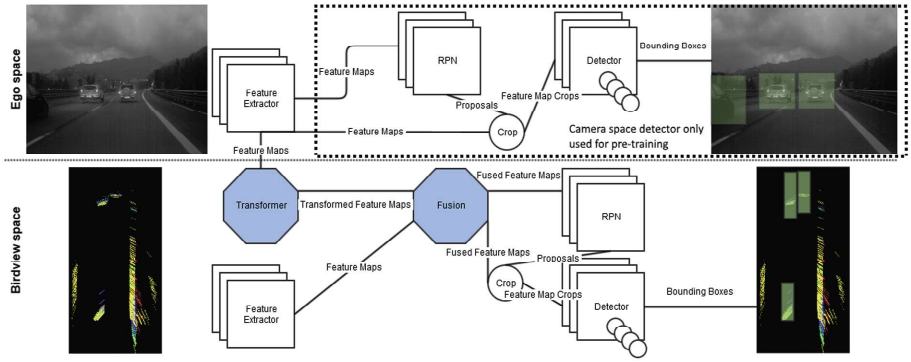



**Fig. 1.** Our proposed fusion architecture. White elements denote parts of the network belonging to an arbitrary state-of-the-art object detector (Faster-RCNN in our case). Blue elements denote our Transformer and Fusion modules mapping feature maps from ego to bird view. This allows using camera features in addition to Lidar for predicting objects' locations and classes in bird view. The camera-only detector is used only for training the feature extractor and can be removed afterwards.

A modular approach, fusing information at a single point, would be beneficial for real-world use cases. Our approach focuses on making feature map activations from different sensors compatible in locality and properties to allow for fusion. The author in [12] uses an approach that is not dissimilar to ours. The authors perspectively transform feature maps from the camera feature extractor network into the corresponding bird view space by using the camera's inverse perspective transformation matrix. Then they concatenate the two feature maps and perform region proposal generation and classification on this fused feature representation. However, the process of using a static inverse perspective transformation from camera space to bird view (and hence implicitly calculating depth information from a 2D image through inverse camera projection and ground plane assumption) is inherently lossy and produces features with strong distance-dependent distortion which are not suited well for processing via convolutional layers in a detection network. We thus improve upon this idea by learning the perspective transformation in order to generate non-distorted features.

The main component of our proposed architecture is influenced by the **Perspective Transformer** [13], originally used to learn perspective-invariant features from 2D projections of objects. They propose an encoder-decoder architecture that takes as input a 2D image of an object and generates 3D volumetric features. 2D input data is downsampled via convolutional operations, fed through a fully-connected bottleneck layer and then upsampled using transposed convolutions to a 3D volume. From this 3D volume, a 2D projection image is created. Similarity between projection of the generated 3D volume and the original 2D image is used as training objective. This forces the encoder to extract perspective invariant features of the object in order for the decoder to create meaningful 3D information from the fully-connected intermediate representation.

## 3 Fusion Architecture

In our implementation of our proposed architecture we use camera and Lidar sensors to detect objects in the autonomous vehicle's environment. Our employed architecture is shown in Fig. 1. The architecture respects that most relevant sensors can be processed efficiently in a 2D bird view representation, since all relevant sensors with exception of monocular cameras produce distance information. Thus, our framework mainly operates in 2D bird view and outputs 2D object bounding boxes. It consists of two standard object detection networks (one for each sensor modality) and the Transformer as well as a Fusion module. Fusion of the sensor modalities takes place at one single location within the object detection networks, after feature extraction and before region proposal generation.

### 3.1 Object Detectors

In our experiments, we use two Faster RCNN [9] object detection networks with Resnet feature extractors [5], one for camera ego view and one for Lidar bird view. Our proposed architecture does not have any special requirements for the object detection network (other then separation of feature extraction and detection), so it would work with any state-of-the-art object detection network. We modified the standard Faster RCNN network to output six scalars per bounding box, four for the object location and dimensions relative to the anchor with position $(y_\mathrm{a}, x_\mathrm{a})$ and dimension $h_\mathrm{a}, w_\mathrm{a}$ and two representing object orientation angle $\phi$ as imaginary and real components $t_\mathrm{im}, t_\mathrm{re}$ of a complex number $e^{i\phi}$ with unit length. Given $\phi$, $t_\mathrm{im} = sin(\phi)$ and $t_\mathrm{re} = cos(\phi)$ or given $t_\mathrm{im}, t_\mathrm{re}$, $phi = arctan_2(t_\mathrm{im}, t_\mathrm{re})$. This representation of an oriented bounding box was suggested by [10] as a simple modification to add orientation to a bounding box without modifying the regression loss of the network, as the *euclidean distance* metric can be applied to this angle representation as well. Our predicted bounding box parameters are thus:

$$t_\mathrm{y} = \frac{(y - y_\mathrm{a})}{h_\mathrm{a}} \quad t_x = \frac{(x - x_\mathrm{a})}{w_\mathrm{a}} \quad t_\mathrm{h} = log(\frac{h}{h_\mathrm{a}}) \quad t_w = log(\frac{w}{w_\mathrm{a}}) \tag{1}$$

$$t_\mathrm{im} = sin(\phi) \quad t_\mathrm{re} = cos(\phi) \tag{2}$$

Note that this box encoding scheme is used not only on the bird view Lidar network, but also for the ego camera space network.

### 3.2 Transformer

Key component of our proposed fusion architecture is the Transformer module, shown in Fig. 2. It is used to transform feature maps from camera space into bird view, creating locality in the feature maps needed for further processing via convolutional layers in the detector network. We adopt a convolutional encoder-decoder scheme as employed in [13].

The general setup of the Transformer is comparable to that of a regular autoencoder, with the difference that the system is not tasked to recreate the original input after decoding. Instead, the task is to output features which are similar in *locality* and *properties* to that of the other sensor modality. The convolutional encoder-decoder represents a function $T : \mathsf{F}_{\mathsf{ego}} \rightarrow \mathsf{F}_{\mathsf{birdview}}$ from ego (camera) feature maps $\mathsf{F}_{\mathsf{ego}}$ to bird view (Lidar) feature maps $\mathsf{F}_{\mathsf{birdview}}$. The Transformer has to perform two tasks: It has to transform *perspective* of the feature maps from ego to bird view while at the same time producing features in bird view *matching* those of the native Lidar feature extractor. We could model the perspective transformation via traditional methods (inverse perspective mapping), but this only works by ground plane assumption (every point in the image lies on the ground plane), resulting in strong distance-dependent distortions which are not suited for convolutional operations. Instead, we train the Transformer to perform both tasks in a joint manner, thereby being able to learn which features at what location in ego view are useful for detecting objects in bird view.

Input to the encoder are feature maps $\mathsf{F}_{\mathsf{ego}}$ from the camera space feature detector. These feature maps contain feature activations with locality in regard to the input camera space. A cascade of convolutions downsamples these feature maps, which are then fed through two fully-connected network layers and are upsampled through transposed convolutions to generate the output feature map $\mathsf{F}_{\mathsf{birdview}}$. We use a second fully connected layer in the bottleneck in order to get matching dimensions of $\mathsf{F}_{\mathsf{birdview}}$ for fusion of the feature maps with those from the Lidar in the next step, so the exact dimensions are tailored to our particular experiment setup. A Rectified Linear Unit (ReLU) non-linearity follows after every layer.

The main idea behind the encoder-decoder architecture is that by squeezing information through a fully-connected bottleneck layer and then upsampling this information to the desired output space, the Transformer is forced to represent information in an perspective invariant manner. The characteristics of the transformation are entirely defined by the training procedure and not by the Transformer architecture: The output space is enforced by letting the system predict objects in the desired output space. Generated output features are implicitly specified by using them with a pre-trained detector network.

### 3.3 Fusion

Transformed feature maps $\mathsf{F}_{\mathsf{ego}\rightarrow\mathsf{birdview}}$ from the prior step possess the same width, height and feature locality as those of the bird view Lidar detector $\mathsf{F}_{\mathsf{birdview}}$ (see Fig. 5 for an example of the transformed feature maps). This allows fusion with the native bird view Lidar feature map through various schemes, such as mean, weighted mean or concatenation. Following [1,7], we employ depth concatenation, i.e. concatenation along the depth axis of the feature map tensors. Since the resulting concatenated feature maps have incompatible depth for the bird view Lidar object detector, we feed them through two convolution operations (stride one, kernel $3 \times 3$) to bring the feature map depth down to the original depth of the bird view object detector.

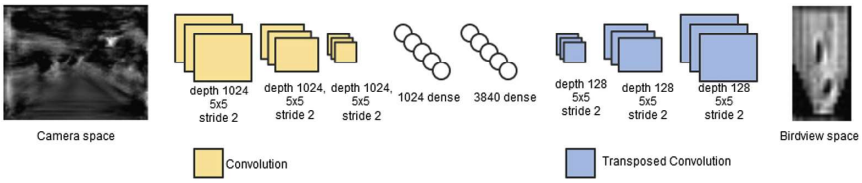



**Fig. 2.** Transformer: Encoder-decoder architecture for mapping feature maps from ego space to bird view space. Rectangular elements denote convolutional operations while circle-shaped elements are fully-connected layers. A *ReLU* non-linearity is applied after every layer.

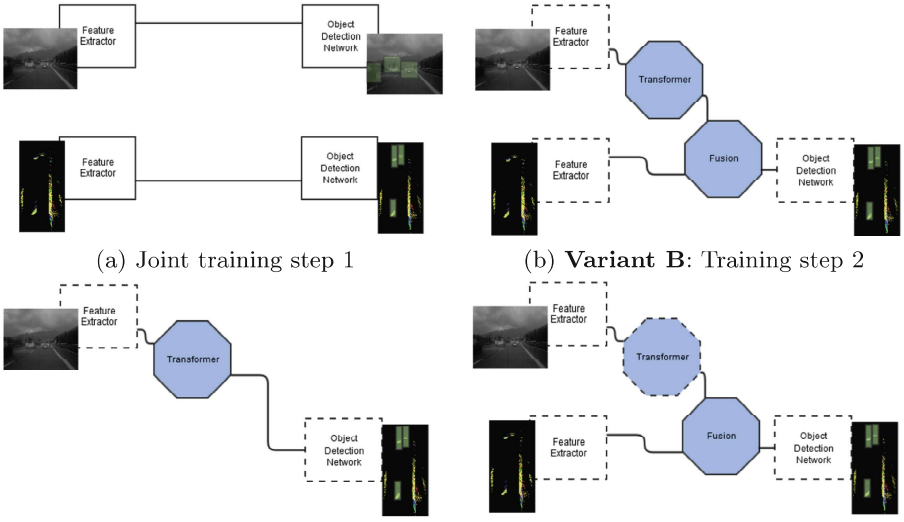
## 4 Training

The training scheme to train the fusion network is outlined below. We use Momentum-based Stochastic Gradient Descent (SGD) with a learning rate of $10^{-4}$ for optimization. Additionally we use L2-regularization, Batch Normalization and Dropout with $p = 0.5$ on the convolutional layers of Transformer and Fusion modules.

### 4.1 Training Procedure

As outlined above, characteristics of the Transformer depend entirely on the Training procedure. Therefore we investigate two slightly different variants, denoted *variant A* and *variant B*. Both variants have the first step in common (Fig. 3a): Training of the **single-modality baseline** Faster-RCNN models for camera ego view and Lidar bird view individually with random weight initialization until convergence (i.e. no more significant improvement on validation error). The next training steps are described below.

**Variant A.** Trains Transformer and Fusion modules in separate steps. First, we train the **Transformer** (Fig. 3c): For this, we take the camera feature extractor trained in the previous step, connect it to a random-initialized Transformer module and feed its output through the bird view detection network that has also been trained in the previous step. The task is to predict objects in bird view. All weight parameters except those of the Transformer are frozen, meaning that only its weight parameters are affected by the optimizer. This forces the Transformer to learn a transformation from camera ego to Lidar bird view feature maps. After training until convergence, transformed feature maps now *mimic* those of the native Lidar bird view feature extractor. We could use this network in standalone mode to predict bird view objects (and hence implicitly predict depth information) from a monocular camera sensor. For fusion (Fig. 3d), we insert a random-initialized **Fusion** module between Transformer, Lidar bird view feature extractor and detector. We now freeze all weights except those of the Fusion module and again train the entire system until convergence with detecting bird view objects as task.




(a) Joint training step 1

(b) **Variant B**: Training step 2

(c) **Variant A**: Training step 2

(d) **Variant A**: Training step 3

**Fig. 3.** Top-level network layouts for the employed training steps. Solid boxes indicate the modules which are being trained in this step. Dashed boxes indicate pre-trained, weight-fixed modules from a previous training step. Training (a) Joint training of single modality baseline networks. (b) Variant B: Training of Transformer and Fusion Module. (c) Variant A: Training of Transformer. (d) Variant A: Training of Fusion module with pre-trained Transformer.

**Variant B.** Trains Transformer and Fusion modules in a joint manner (Fig. 3b). The setup is the same as that of the last step of Variant A, however the Transformer has not been trained on its own and its weights are thus also randomly initialized. All weights except those of the Transformer and Fusion modules are frozen and the whole system is again trained until convergence with the task of detecting bird view objects.

### 4.2 Objective Function

Objective (loss) function for all training phases is the default loss function as employed in the original Faster-RCNN paper [9], minimizing error over class and object location for each of the anchor boxes $i$ that the network predicts:

$$L_{\text{det}} = \frac{1}{N_{\text{cls}}} \sum_i L_{\text{cls}}(p_{\text{i}}, p_{\text{i}}^*) + \lambda \frac{1}{N_{\text{reg}}} \sum_i L_{\text{reg}}(t_{\text{i}}, t_{\text{i}}^*) \tag{3}$$

$L_{\text{cls}}$ denotes the *classification loss* while $L_{reg}$ is the bounding box regression loss (which for us includes six regression parameters as we also include bounding box orientation as outline in Sect. 3.1). $p_i, p_i^*$ are predicted and groundtruth classes and $t_i, t_i^*$ are predicted and groundtruth bounding box parameters. For any of the training phases described above we use this prediction-centric loss function while freezing different parts of the network weights.

When training the Transformer, we additionally use a reconstruction loss $L_r$, which is the mean squared distance between transformed camera ego view feature map $\mathsf{F}_{\mathsf{ego}\rightarrow\mathsf{birdview}}$ and native Lidar bird view feature map $\mathsf{F}_{\mathsf{birdview}}$, as this helps convergence by encouraging similarity to the native Lidar feature map:

$$L_{\text{r}} = \frac{1}{N}\sum_{i}^{N}(\mathsf{F}^{\mathsf{i}}_{\mathsf{birdview}} - \mathsf{F}^{\mathsf{i}}_{\mathsf{ego}\rightarrow\mathsf{birdview}})^2 \tag{4}$$

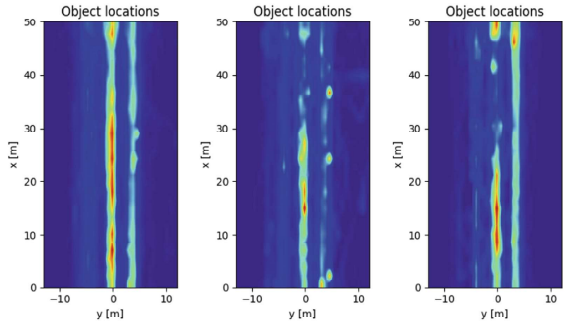
where $i$ is the index running over all entries of the feature maps. The full loss term $L$ thus becomes

$$L = L_{\text{det}} + \lambda L_{\text{r}} \tag{5}$$

with a hyperparameter $\lambda$ for weighting the reconstruction loss (set to 0.5 for our experiments).

## 5 Data

We evaluate our method on two datasets: Our own automotive dataset as well as the public KITTI dataset [3]. Our own dataset is comprised of 16 mixed driving sequences (mostly European highway and rural) with an overall length of approx. 8 h sampled at 10 hz, yielding 284690 samples. The employed sensor set consists of a monochrome camera with resolution $1280 \times 960$ and a field of view of 45° as well as a front-facing four channel Lidar with a field of view of 140°. Additionally a Velodyne HDL-64E 64 channel Lidar sensor is used as reference sensor to manually label objects in 3D space (x/y/z position, width/length/height, yaw orientation and class) in the ego vehicles own lane and the adjacent left and right lanes up to a distance of 100 m. From each of the 16 sequences, two subsequences amounting to 20% length are set aside as validation set. Furthermore we have a separate 40 min long mixed driving sequence that we only use as test set. See Table 1 for statistics on our dataset and Fig. 4 for distribution of groundtruth objects. For experiments on the KITTI set we use a custom training/validation split of approximately 7:1, where we made sure that all samples in the validation set are from different sequences as those in the training set.




(a) Training (b) Validation (c) Test

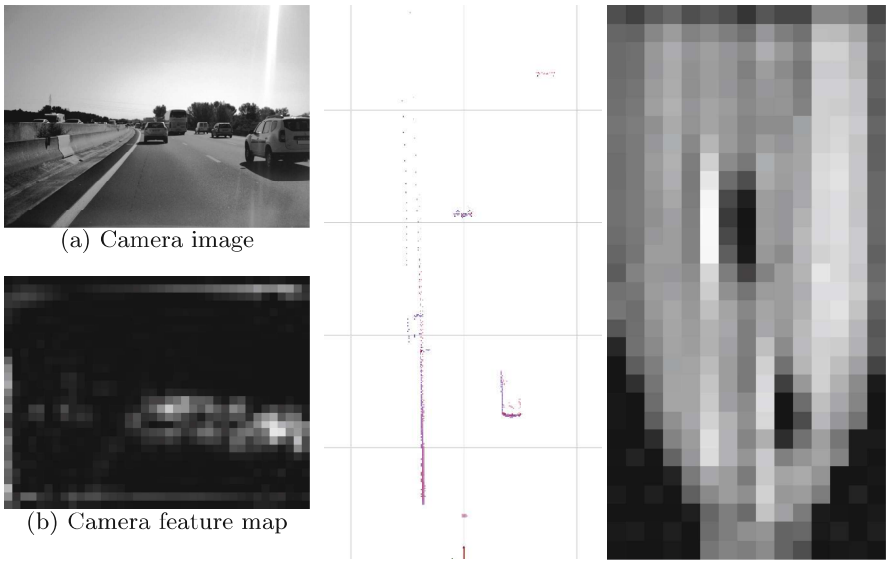
**Fig. 4.** Distribution of groundtruth object locations (bird view) in our dataset, encoded as heatmap. Hotter locations denote areas of increased occurrence of groundtruth objects. Note that the labeled area includes only the own vehicles' lane as well as the ones right next to it.

### 5.1 Preprocessing

Camera images are preprocessed through VGG-style per-channel mean substraction as in [11]. We project Lidar pointclouds to two 2D bird view grid representations as suggested by [1], one containing the normalized height of the highest Lidar point which falls into a particular grid location and one containing the normalized mean intensity of all points falling into a grid location. The two grids are then concatenated by depth, resulting in two-channel 2D input images for the Lidar object detector. We use a [50 m, 25 m] area as viewport for Lidar bird view projection, with the ego vehicle located at the bottom of the frame and facing upwards. Figures 5a and c show an example of our input camera and lidar data. For KITTI experiments we employ the same data preprocessing, however due to the higher field of view of the camera of 90° we use a viewport of [50 m, 50 m].

**Table 1.** Statistics of our employed dataset. Note that for our experiments we only use the class *car* and thus only report numbers for this class.

| | Frames | Duration | Number of instances (Car) |
|---|---|---|---|
| Training | 225745 | 6:16:14 | 110662 |
| Validation | 58945 | 1:38:14 | 31059 |
| Test | 31090 | 0:38:20 | 25809 |



(a) Camera image

(b) Camera feature map

(c) Bird view projection of Lidar pointcloud

(d) Transformed camera feature map

**Fig. 5.** (a) shows the front camera image of an example scene. (b) shows one of the 1024 feature maps output by the camera feature extractor. (c) shows our bird view Lidar projection as input to the Lidar feature extractor. (d) shows the feature map of (a) transformed from ego to bird view by the Transformer after being trained using *Variant A* of the training procedure (see Sect. 4.1). Note that activations are generated from the 2D camera input image only, so the Transformer implicitly has to learn to predict depth from the image. You can clearly identify areas in the feature maps belonging to the two cars in front of the ego vehicle.

## 6 Experiments

We use an extended version of the Object Detection API framework [6] for our implementation, which has been heavily modified to support multiple synchronous input sensor streams and predicting objects in multiple spaces. Reported metrics are Pascal VOC metrics [2] with mean average precision at 0.5 intersection over union (IOU). IOU is calculated on axis-aligned bounding boxes regardless of an object's orientation, so we also require the predicted orientation to be within $[+\frac{\pi}{16}, -\frac{\pi}{16}]$ of the groundtruth orientation to count as a true positive. We investigate both variant A and B of the training algorithm. For comparison, we also provide results for the Lidar-only detector which has been used as baseline for Transformer and Fusion module training. Our networks have been trained either entirely on our own dataset or entirely on KITTI (due to the different employed sensors in both datasets).

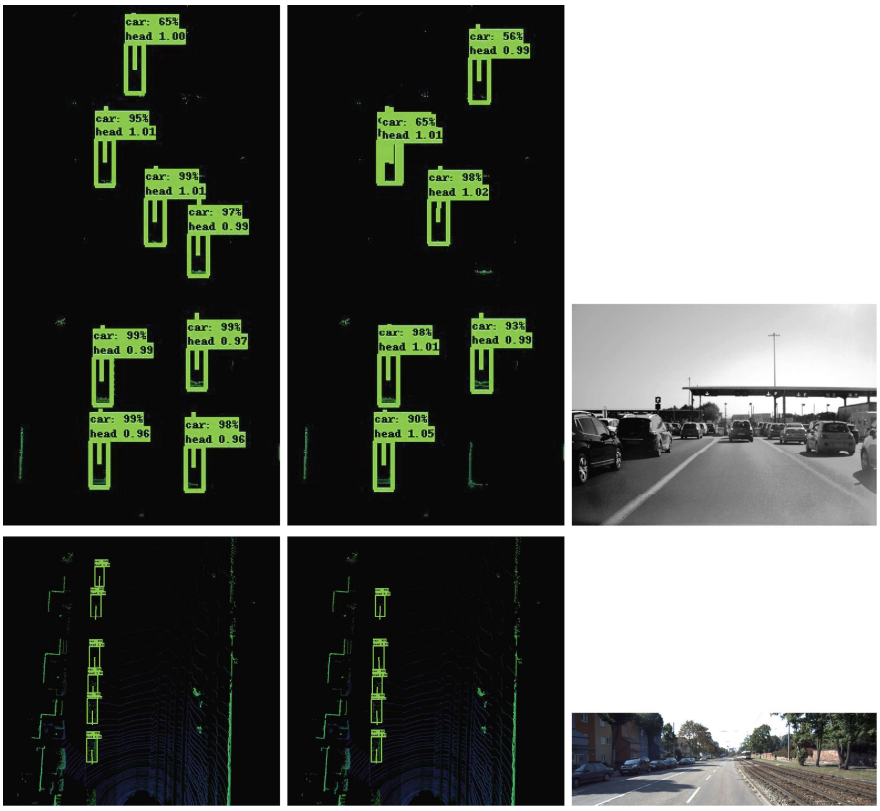
**Table 2.** Results on our validation set. We report mean average precision (mAP) at bounding box intersection-over-union (IOU) 0.5 for class *car*. Numbers labeled *Cam+Lidar* describe results with both sensor modalities enabled. Numbers labeled *Lidar* or *Cam* describe results with that particular sensor modality only (in case of Fusion, the corresponding input feature maps to the Fusion stage are set to zero).

| | Baseline | Fusion Variant A | | | Fusion Variant B | | |
|---|---|---|---|---|---|---|---|
| | Lidar | Cam+Lidar | Lidar | Cam | Cam+Lidar | Lidar | Cam |
| Own validation | 0.85 | 0.90 | - | - | **0.94** | - | - |
| Own test | 0.78 | 0.78 | 0.56 | 0.16 | **0.82** | 0.70 | 0.0 |
| KITTI validation | 0.75 | 0.76 | 0.64 | 0.01 | **0.77** | 0.68 | 0.0 |

### 6.1 Results

Table 2 shows results of the experiments while Fig. 6 contains some sample scenes with predictions of the fusion system. We see that both fusion schemes show improved classification results on our validation set. Variant B also improves performance on the test set as well. Variant A however allows to "switch off" one of the sensor paths (simulated in our experiments by setting the tensor coming from this sensors' feature extractor to zero). In case of disabling the camera, the network is still able to show good detection performance, though worse than the Lidar-only network. This shows that the network relies on a combination of features from both Lidar and camera. When disabling the Lidar, detection performance drops significantly, although near objects directly in front of the car are still detected relatively well. These are very desirable characteristics for any real-world perception system which has to deal with sensor failures or blindness of individual sensors due to weather. Variant B (joint training of Transformer and Fusion from scratch) does not result in the Transformer learning any features that are useful on their own. However, running this network as intended with both sensor modalities enabled improves detection performance compared to the Lidar-only baseline network. Qualitatively, fusion significantly reduces false-positives compared to Lidar-only (which tends to erroneously detect objects on barriers or other static objects resembling the characteristic L-shapes in the projected Lidar data).

We can confirm these results on the KITTI set as well. Variant A camera-only however does not yield proper detections even though the feature maps are interpretable like those on our own dataset. Note that the KITTI dataset is several magnitudes smaller than our dataset, while the employed Lidar sensor has a much higher resolution resulting in much denser pointcloud projection. In order for the Transformer to produce feature maps that are similar enough to the dense ones from the Lidar feature extractor, it would need a higher capacity. However, due to the limited amount of data we could not significantly increase capacity of the Transformer. Nonetheless, we see improved detection performance when using both sensor modalities together.




(a) Fusion (b) Lidar-only (d) Camera image

**Fig. 6.** Inference detection samples on our test set (top row) as well as on the KITTI validation set (bottom row). (a) and (b) show detected objects in bird view for the Fusion system and the Lidar-only system as reference (plotted over the projected Lidar pointcloud). We see that in accordance to the quantitative results the fusion system manages to detect more of the objects in the scene. Qualitatively, using both sensors in conjunction also improves false positives which the Lidar-only network tends to generate on isolated frames.

## 7 Conclusion

We propose a novel and general method for fusing pre-trained single-modality object detection networks. Key component is a Transformer module which learns transformation of feature map activations from one sensor space to another, using one pre-trained detection network for another sensor modality. Using this method, we show that we can fuse features from a camera sensor in ego space with

those of a Lidar sensor in bird view to improve detection performance compared to using only the Lidar. Our fusion method differs from the state-of-the-art in that it fuses sensor modalities at a single location, leaving the underlaying single modality networks intact. This results in a simple network architecture, increasing traceability of the entire system and allowing to deal with real-world challenges such as sensor failures or blindness.

## References


1. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3D object detection network for autonomous driving. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6526–6534 (2017). https://doi.org/10.1109/CVPR.2017.691. http://doi.ieeecomputersociety.org/10.1109/CVPR.2017.691
2. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes (VOC) challenge. Int. J. Comput. Vis. **88**(2), 303–338 (2010). https://doi.org/10.1007/s11263-009-0275-4, http://link.springer.com/10.1007/s11263-009-0275-4
3. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3354–3361. IEEE (2012). http://ieeexplore.ieee.org/abstract/document/6248074/
4. Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1440–1448 (2015). http://www.cv-foundation.org/openaccess/content_iccv_2015/html/Girshick_Fast_R-CNN_ICCV_2015_paper.html
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016). https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html
6. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors. arXiv:1611.10012 [cs] (2016)
7. Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.: Joint 3D Proposal Generation and Object Detection from View Aggregation. arXiv:1712.02294 [cs] (2017)
8. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum PointNets for 3D object detection from RGB-D data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 918–927 (2018). http://openaccess.thecvf.com/content_cvpr_2018/html/Qi_Frustum_PointNets_for_CVPR_2018_paper.html
9. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497 [cs] (2015)
10. Simon, M., Milz, S., Amende, K., Gross, H.M.: Complex-YOLO: Real-time 3D Object Detection on Point Clouds. arXiv:1803.06199 [cs] (2018)
11. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs] (2014)
12. Wang, Z., Zhan, W., Tomizuka, M.: Fusing Bird View LIDAR Point Cloud and Front View Camera Image for Deep Object Detection (2017). https://arxiv.org/abs/1711.06703

13. Yan, X., Yang, J., Yumer, E., Guo, Y., Lee, H.: Perspective transformer nets: learning single-view 3D object reconstruction without 3D supervision. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 1696–1704. Curran Associates, Inc. (2016)
14. Zhou, Y., Tuzel, O.: VoxelNet: end-to-end learning for point cloud based 3D object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4490–4499 (2018). http://openaccess.thecvf.com/content_cvpr_2018/html/Zhou_VoxelNet_End-to-End_Learning_CVPR_2018_paper.html