# Learning-Based Ego-Vehicle Speed Estimation

Christopher Raffl
ETH Zurich
rafflc@ethz.ch

Jens Eirik Saethre
ETH Zurich
saethrej@ethz.ch

Livio Schläpfer
ETH Zurich
livios@ethz.ch

## Abstract

*Recent advances in machine learning have made deep learning techniques attractive to the automobile industry. In this work, we tackle the task of ego-vehicle speed estimation from monocular dashcam footage using a purely deep-learning based approach. Our proposed Dual-CNN-LSTM model improves on previous work by reducing the RMSE by* $38.7\%$ *on the comma2k19 dataset by exploiting optical flow between successive frames.*

## 1. Introduction

From 1971 to 2021 the weekly number of deaths from road traffic accidents in Switzerland decreased from a staggering 34 people to less than 4 [11], even though traffic on Swiss roads has significantly increased [10]. A large fraction of this decrease can be attributed to vehicles becoming safer. On the one hand, this is due to regulatory changes enforcing e.g. the use of seat belts or airbags, and, on the other hand, due to the advent of *advanced driver assistance systems (ADAS)* [5]. In the 1980s and 90s engineers used odometry to build *anti-lock braking systems (ABS)* [27] and *traction control systems (TCS)* [19] aimed at stabilizing the vehicle. In the new millennium, cameras as well as multiple sensors (e.g. RADAR, ultra-sonic, or infrared) were incorporated into vehicles, which led to the development of assistants for parking [26] and lane-keeping [12], as well as the first versions of *adaptive cruise control (ACC)* [32]. The last decade saw research interest in the area skyrocket [1, 3, 4, 34, 36] towards fully autonomous driving.

One major cause of road traffic accidents that could not be resolved by the aforementioned advances in vehicle security is over-speeding. To combat this issue, governments place speed traps at dangerous sections of roads and place heavy fines on over-speeding. These speed traps have historically been RADAR-based systems, while newer generations are based on LiDAR. The former can suffer from imprecision especially at high speeds and large distances. While the latter offers greater precision, bad weather condition can significantly hamper its performance [14, 33]. Moreover, both RADAR and LiDAR speed traps require fairly complex and expensive machinery [9] that has to be physically installed and calibrated at each location, making them less flexible.

Great progress in deep learning in recent years [21, 29, 30, 31] has made it increasingly attractive to tackle the vehicle speed estimation problem using camera footage and purely learning-based approaches. These can be carried out on already existing infrastructure such as traffic surveillance cameras and thus increase flexibility while greatly reducing cost and complexity [9].

Due to their generality, such DL-based models have been applied to mobile cameras such as dashcams placed inside the vehicle [2, 24]. This allows us to try to either estimate the speed of other vehicles or the speed of the vehicle the camera is located in, i.e. the *ego-vehicle speed*. Applications of the latter are two-fold: In real-time situations, learning-based ego-vehicle speed estimation can be used, perhaps in combination with data from additional sensors, to equip vehicles with ADAS functionality such as ACC that do not come with it built-in. In a non-real time setting, the method can be e.g. used for accident reconstruction when dashcam footage but no blackbox information is available [2].

In this work, we tackle the problem of *ego-vehicle speed estimation* by learning from monocular dashcam footage. More precisely, given an input sequence of $n$ video frames of a car driving, our models predict the ego-vehicle speed at each of the $n$ frames. To make our models as general and as applicable as possible, we do not rely on any other sensor data, even if it were available. For training we use the *comma2k19* data set (cf. section 4.1) [25]. We start by implementing a baseline model from previous work [2] and improve it by incorporating hand-crafted features such as optical flow and estimated depth information that model they way humans perceive motion. In doing so, we significantly improve the performance with respect to the baseline by up to 55%.

## 2. Related Work

The task of ego-vehicle speed estimation has not received a great amount of effort from the research community yet. In his work from 2016, Han proposed a method relying on projective geometry and the fact that it preserves the cross-ratio to compute the ego-vehicle speed. However, the approach requires knowledge about physical distances such as the length of the car's wheelbase [13]. In both [35] and [2], the authors use an LSTM to capture temporal dependencies between the features. While the former relies on sensor and GPS data, the latter uses CNNs to extract features automatically. In particular, Bandari et al. [2] implement a Dual-CNN architecture inspired by [18], albeit incorrectly. Qimin et al. compute a sparse optical flow between image sequences and average the obtained values to derive a speed estimate [22]. On the other hand, Rill estimates optical flow via a neural network in [24] and uses another neural network to estimate monocular depth. He then applies a filter to obtain significant locations and computes the ego-speed in the image domain. Finally, a scale factor is derived to translate the speed into the physical domain.

## 3. Methods

In this section we introduce the core concepts and building blocks that we use to tackle the task of ego-vehicle speed estimation. We start by describing how we extract features from individual frames. This is followed by methods to annotate video frames with additional data. We then explain how we estimate speed values from the extracted features. Finally, we discuss methods to exploit the temporal context between successive video frames and their associated speed value.

### 3.1. Video Frame Feature Extraction

We use convolutional neural networks (CNNs) to learn and extract complex features from individual frames. More concretely, our CNNs consist of convolutional layers followed by average pooling and ReLU activation layers. We were unfortunately restricted to using simple CNNs because we only had access to limited resources[1] and during training, almost all of the GPU's memory was consumed by the video data. We did therefore not use pre-trained CNNs such as ResNet [15], VGG [28], or Xception [7]. We describe the specifics of our CNNs in the following.

#### 3.1.1 Baseline CNN

We use a CNN model introduced in [2] as our baseline model for feature extraction. It consists of a single 2D-convolution layer with kernel size $k = 5$, stride $s = 1$, and zero-padding followed by batch normalization. In order to reduce the number of extracted feature, we add a pooling layer to the model. We will hereinafter refer to this model as *Baseline CNN*.

[1] In particular, we did not once receive a GPU node with more than 11 GB of memory on Euler.

#### 3.1.2 Improved CNN

The *Baseline CNN* consists of only a single convolutional layer. Its representational power is thus very limited and insufficient to capture the information in a frame accurately. We therefore improve the model by adding two more convolutional layers, such that our *Improved CNN* in total consists of three 2D-convolution layers with kernel size $k = 5$, stride $s = 1$, and zero-padding. Each convolutional layer is followed by an average pooling, a batch normalization, and a ReLU layer. The greater number of convolutional layers allows the model to gradually extract lower-level features, while also reducing the total number of extracted features. This has the side-effect of reducing the model's memory footprint, which effectively frees resources that can e.g. be used when taking the temporal context into account (cf. section 3.4).

#### 3.1.3 Dual-CNN

To further refine our feature extraction mechanism we build a Dual-CNN inspired by [2, 20], which was initially proposed by [18]. Instead of using just a single CNN for feature extraction, the authors of the latter propose to use two CNNs in parallel. The idea is that one CNN extracts local features using a small kernel size while the other CNN extracts global features using a larger kernel size. This combination should improve the ego-vehicle speed estimates as the global features can provide context for the local features.

We implement our Dual-CNN by following the architecture from [20] since Bandari et al. failed to properly employ the anticipated design in their work [2]. Our Dual-CNN starts with a block consisting of a single 2D-convolution layer with kernel size $k = 12$, stride $s = 1$, and zero-padding, followed by a pooling, a batch normalization, and a ReLU activation layer. This block can be considered as shared between the local and global CNNs that follow. They both consist of two 2D-convolution layers with average pooling and batch normalization. The local CNN uses a kernel size of $k = 12$, while the global CNN uses a kernel size of $K = 24$. The local and global CNNs are applied on the shared block's output and their extracted features are then concatenated. A model using a such architecture in combination with an LSTM is depicted in figure 1. Note that the first shared block was omitted for clarity.

### 3.2. Video Frame Annotations

So far, we have discussed approaches to extract image features through CNNs. However, this approach is com-

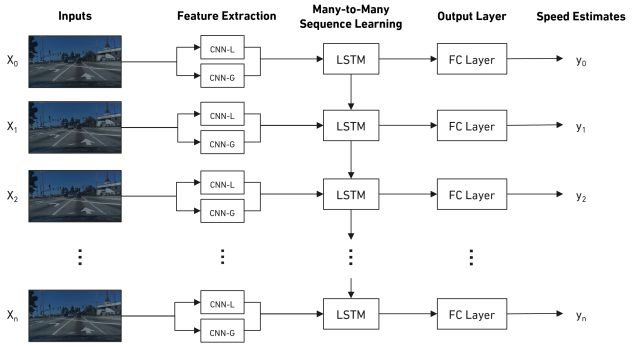


Figure 1. Architecture of a Dual-CNN-LSTM model. CNN-L and CNN-G extract local and global features, respectively.

pletely oblivious to the task at hand. We thus want to incorporate prior knowledge of the task into our models that are based on how we as humans perceive speed very intuitively: we judge an object's speed by how fast it moves relative to its distance to us. In the following, we describe two ways to annotate our input frames with such information.

#### 3.2.1 Optical Flow

Optical flow captures relative motion between an observer and a scene over consecutive frames [17]. Given the current and previous frame as input, we can estimate the magnitude and the angle of the optical flow for each pixel. An approximation of the optical flow can be efficiently computed [8] and we use OpenCV [6] on every pair of successive input frames to do so. For each image pair, OpenCV computes the optical flow per-pixel, i.e. it returns the magnitude and angle of the optical flow for every pixel. This information can thus be appended to the current input frame as two additional channels.

We expect the optical flow to be a valuable source of information as it allows the feature extraction model to directly capture the movements of objects in the input frame. Through this annotation, we hope that our models learn to distinguish between optical flows that are useful for ego-vehicle speed estimation (e.g. trees next to the road or road surface markings) and irrelevant optical flows (e.g. overtaking cars).

#### 3.2.2 Depth Estimation

Optical flow provides information about the apparent motion of pixels between consecutive frames. In order to estimate the absolute movement of the vehicle, the optical flow must be further augmented to include the distance between the objects represented by these pixels. This is due to the fact that objects closer to the camera appear to be moving faster than more distant objects with the same absolute speed. To that end, we add another type of frame annotation, namely a depth estimate. Since we rely solely on video sequences captured from a monocular camera, we cannot compute the depth of our input frames. Instead, we have to estimate it. We use a pre-trained, state-of-the-art machine learning model called MiDaS [23] to perform depth estimation. There are three different versions that vary in size and inference speed. We use the smallest model due to its reasonable memory footprint and since it allows us to process up to 30 frames per second. The computed depth estimate is added to the input frame as an additional channel and is also taken into account when extracting the frame features with the feature extraction model.

We expect that annotating input frames with depth information in conjunction with optical flow annotations should improve the performance of our models significantly. This is due to the fact that, given accurate optical flow and depth information, an estimate of the ego-vehicle speed can be computed directly, as was done by Rill in [24]. On its own, depth information could be useful in determining regions of interests in the input frame, i.e. closer regions are more important than farther ones.

### 3.3. Estimating Speeds from Frame Features

Given the frame features from the feature extraction model, we estimate the ego-vehicle speed by applying multiple non-linear regression layers. We flatten the extracted features that are forwarded by the CNN and apply two fully connected (FC) layers, where the last layer outputs one ego-vehicle speed estimate per video frame.

### 3.4. Incorporating Temporal Context

The approach described in section 3.3 predicts speed values for each frame individually, thereby completely ignoring the temporal context of successive input frames and their corresponding speed values.[2] Video sequences in the *comma2k19* dataset are shot at 20 fps. The time difference between two successive input frames is thus only 50 milliseconds. In time spans this short, it is very unlikely if not impossible for two successive speed values to differ significantly. Therefore, our models should not only consider the narrow context of a single frame but exploit the combined temporal behaviour across frames in a video sequence. In the following, we describe two common techniques used to incorporate temporal dependencies into machine learning models that allow us to enlarge the temporal context to more than one neighbouring frame.

[2]Note that our annotation approach does incorporate temporal dependencies into the model by computing the optical flow across two frames, however, only to a very limited extent.

#### 3.4.1 Temporal Smoothing with 1D Convolution

Given that the true ego-vehicle speeds measured at consecutive video frames evolve smoothly, the predicted ego-vehicle speeds should exhibit a smooth evolution as well. In order to leverage this insight, we can apply a 1D-convolution layer to the speed estimates that are produced by the final fully-connected layers. In our experiments (cf. section 4.3), we use a kernel size of $k = 5$. We expect the 1D-convolution to act as a weighted moving average that helps to eliminate potential outliers and, as a result, reduce the model's prediction error.

#### 3.4.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks [16] are a well-established method to exploit temporal dynamic behavior. In the past years, LSTMs have been widely used in various sequence learning tasks such as natural language processing (NLP) or time series classification. In order to take context across video frames into account, the LSTM unit maintains a cell state over time and controls the flow of information using gates. In our application, the gates allow the LSTM to selectively update the cell state based on the extracted frame features of the current video frame, and to selectively forget irrelevant information from past video frames. In our experiments (cf. section 4.3), the LSTM maintains a cell state of 64 features. We use it in a many-to-many setting such that there is one input and one output for each frame. These outputs of the LSTM are then forwarded to the regression model introduced in section 3.3 to obtain the final speed estimates.

We expect models using an LSTM for temporal context to perform better than models using temporal smoothing via 1D-convolution for three mains reasons. First, an LSTM has a much higher representational power than a single 1D-convolution layer. Moreover, the LSTM is applied directly on the extracted frame features, which allow it to capture much more complex temporal dependencies. Finally, the LSTM has a higher receptive field than our 1D-convolution with kernel size $k = 5$.

## 4. Experiments

In this section we first introduce the used dataset, necessary dataset modifications, and describe the employed training procedure. Then, we cover the experiments we conduct to iteratively enhance our performance through a comparison of different architectures and frame augmentations. The corresponding results are depicted in Table 1 and further discussed in section 5.

Figure 2. Nine samples from the comma2k19 dataset. There are day and night scenes, both in good and bad weather conditions.

### 4.1. The comma2k19 dataset

In this work, we use the *comma2k19* dataset [25] to train and evaluate our DL-based model architectures to estimate ego-vehicle speeds. The dataset consists of more than 33 hours of dashcam footage recorded on highways in the San Francisco Bay area. It offers a rich variety of external settings, including sequences with different weather conditions ranging from sunshine to heavy rain. There are sequences recorded at both day and night, and traffic ranges from rush hour to completely empty streets (cf. Figure 2). The speeds at which the recording vehicle travels also varies greatly between 0 and 35 m/s (0 to 126 km/h). Next to the video recording, additional measurements from internal sensors and GNSS data are provided. These can be used to extract the ground-truth speed labels at each video frame.

#### 4.1.1 Dataset modifications

The dataset contains 2019 video sequences that are each one minute long. They were recorded at 20fps with a resolution of 1164 x 874 pixels and are provided in `.hvec`-format. In a first step, we had to convert all video sequences to `.mp4`-format for them to be compatible with our PyTorch machine learning pipeline. We crop the initial video frames to remove static elements such as the front of the car's cockpit and further compress them, resulting in frames of size 290 x 118 pixels.

### 4.2. End-to-End ML Pipeline

Along this project, we built a custom end-to-end ML pipeline using Pytorch. It loads sequences of dashcam footage and outputs an ego-vehicle speed estimate for every dashcam video frame. The specific building blocks, frame annotation and training parameters for each model can be managed with the help of configuration files.

| # | Feature Extraction Model | Temporal Context | Frame Augmentation | RMSE [m/s] |
|---|---|---|---|---|
| 1 | Baseline CNN [2] | LSTM | - | 5.09 |
| 2 | Dual-CNN [18] | LSTM | - | 4.38 |
| 3 | Improved CNN | Temporal Smoothing | - | 4.50 |
| 4 | Improved CNN | Temporal Smoothing | Optical Flow + Depth Estimate + Grayscale | 4.09 |
| 5 | Improved CNN | Temporal Smoothing | Optical Flow + Grayscale | 3.48 |
| 6 | Improved CNN | Temporal Smoothing | Optical Flow | >20 |
| 7 | Improved CNN | LSTM | Optical Flow + Depth Estimate + Grayscale | 3.70 |
| 8 | Improved CNN | LSTM | Optical Flow + Grayscale | 3.27 |
| 9 | Dual-CNN [18] | LSTM | Optical Flow + Depth Estimate + Grayscale | 3.44 |
| 10 | Dual-CNN [18] | LSTM | Optical Flow + Grayscale | 3.12 |

Table 1. Experiment results. We decompose models into three parts: (i) feature extraction, (ii) temporal context, and (iii) frame augmentation techniques. Every row in the table constitutes an individual model for which we report the root mean squared error (RMSE) in m/s. Vertical lines delineate sets of experiments.

#### 4.2.1 Training and Inference

During training and testing, we randomly sample 60 consecutive frames from every 1 minute video segment (1200 frames) together with the corresponding vehicle speed from the CAN data which act as the ground truth labels. We use a 60:20:20 split for training, validation and testing data, respectively. At training, our models are optimized on the *mean squared error (MSE)* using Adam with a fixed learning rate. For our experiments, we train our models for up to 6 hours on GPUs that were made available to us on the Euler cluster of ETH Zürich.

### 4.3. Experimental Results

In the following, we provide a high-level overview of the experiments we conduct. Each experiment builds on the insights gained from the previous one, allowing us to identify elements that increase the performance and thus iteratively increase the prediction capability. The corresponding results are shown in Table 1.

**Baseline performance.** We first evaluate the performance of the identified previous work. To that end, we test the previously describe baseline CNN in combination with a LSTM and our implementation of the Dual-CNN-LSTM (cf. Table 1 lines 1 and 2).

**Benefits of frame augementations.** In a second experiment, we investigate whether frames augmented with annotations are enhancing the performance of our ego-vehicle speed estimate. We try several combinations thereof with the goal of understanding their effects on another and to check if our expectations outlined in Section 3.2 are valid. For this experiment we use our improved CNN with temporal smoothing (cf. Table 1 lines 3-6).

**Adding advanced temporal context.** Leveraging the insights we gain from the previous ablation study, we keep the best combination of frame augmentations. In the next step, we experiment with advanced temporal context and compare our temporal smoothing to a LSTM (cf. Table 1 lines 7+8).

**Improved frame feature extraction.** Lastly, we replace our improved CNN with the Dual-CNN and compare their performances. Due to its bigger representational power we expect this step to result in an improved frame feature extraction (cf. Table 1 lines 7+8).

## 5. Discussion

In this section, we discuss the results of the experiments outlined in section 4.3. The results are outlined in table 1.

### 5.1. Baseline performance

The goal of this experiment is to establish a baseline that we can use to track our progress. The Dual-CNN-LSTM (cf. figure 1) model achieves a RMSE of 4.38 m/s and thus decreases the error by 13.9% compared to the baseline CNN-LSTM. This result is in line with our expectations since the Dual-CNN has a bigger representational power. Furthermore, we assume that the Dual-CNN is able learn about external conditions and incorporate this knowledge into its speed estimation.

### 5.2. Benefits of Frame Augmentations

Before testing different architectures, we first want to gain some insights into the benefits gained by different frame augmentations. As expected, both optical flow and depth estimation increase model performance. This makes sense especially for the optical flow augmentation, since it introduces prior knowledge about temporal dependencies

between consecutive frames that our model can't learn by itself.

However, the model performs better if the depth estimation is not included (4.09 vs 3.48). Contrary to our expectation, this is the case for every architecture. Considering that depth information could provide valuable insights into how far an object actually moves, we expected that this frame augmentation would further enhance the capabilities of our models. We tried to use the largest and best of the three models in MiDaS, but the performance did not improve. One possible explanation is that a CNN, due to its additive nature, cannot capture the rather complex and perhaps non-linear context between estimated depth and optical flow. It thus seems to suffice to augment a grayscaled input frame with optical flow information for our model to extract valuable features.

The last of the four models (#6) we run in this experiment is merely to show that optical flow only becomes a powerful frame augmentation technique if it is combined with a grayscaled version of the frame. Using only optical flow information of each frame results in terrible performance. This is in line with our expectations since our model cannot make much sense of the relative motion of pixels without further context.

Interestingly, the *Improved CNN* (#3) using temporal smoothing already significantly outperforms the *Baseline CNN* with an LSTM (#1). Moreover, by introducing frame augmentation techniques, we even surpass the *Dual-CNN-LSTM* model (#2). This clearly highlights the importance of incorporating prior knowledge such as optical flow.

### 5.3. Incorporating Temporal Context

Our third experiments reveals that, as expected, models using LSTMs (#7 and #8) are significantly better at capturing temporal dependencies than models using temporal smoothing (#4 and #5). While the latter only counteracts outliers, LSTMs have the capability to propagate information between frames. This is beneficial even when some temporal dependencies are already encoded in the features through optical flow calculations. From this result, we infer that it is better to not only consider changes between two consecutive frames but changes within the entire series. Propagating information from the first frame through the entire series allows our model to learn additional complex features that cannot be simply encoded through frame augmentation.

We further note that using an LSTM results in a larger decrease in the prediction error when depth information is present (9.5% from #4 to #7) compared to when it is not (6.0% from #5 to #8). This might imply that the LSTM is able to extract more information from the depth estimation. However, in both cases the performance is still better if the depth estimation is omitted.

### 5.4. Improved Frame Feature Extraction

In this last experiment, we investigate the difference between using the *Improved CNN* (#7 and #8) and the *Dual-CNN* (#9 and #10), both using LSTMs for temporal context. Our experiment shows that the latter performs better and yields our best result with a RMSE of 3.12 m/s. The decrease in the RMSE between the two versions is 7.0% and 4.5% for versions with and without depth estimation, respectively. Due to the Dual-CNN's greater representational power and more sophisticated architecture, this result is to be expected. Compared to our baseline model (#1), our the Dual-CNN-LSTM with optical flow augmentation decreases the RMSE by a respectable 38.7% from 5.09 m/s to 3.12 m/s.

## 6. Conclusion

In this work, we develop a feed-forward neural network based on a Dual-CNN-LSTM architecture for the task of learning-based ego-vehicle speed estimation in iterative steps. We augment our input sequences by calculating the optical flow between two consecutive frames and feed this prior knowledge together with a grayscaled version of the frame into our model. This leads to a significant decrease of 38.7% of the RMSE compared to our baseline model. Furthermore, we perform a study where we combine building blocks into different architectures that we then compare. By doing so, we show the benefit of including prior knowledge into our models.

## 7. Contributions

In the following, we list the contributions of the individual authors. Note, however, that this project was carried out as a group and thus there were no major contributions from one individual group member where the other two were not involved. All team members equally contributed to literature research, coming up with ideas for our models, conducting experiments and writing the report. The basis including data loading and thus the major part of our pipeline was implemented by Livio. Jens Eirik and Christopher implemented the models, preprocessing, dataset modifications and set up the necessary infrastructure on Euler. Writing the final report was again a collaborative effort involving all the team members.

## References


[1] Aravind Balakrishnan, Jaeyoung Lee, Ashish Gaurav, Krzysztof Czarnecki, and Sean Sedwards. Transfer reinforcement learning for autonomous driving: From w¡span class="smallcaps smallercapital"¿ise¡/span¿m¡span class="smallcaps smallercapital"¿ove¡/span¿ to w¡span class="smallcaps smallercapital"¿ise¡span class="smallcaps smallercapital"¿s¡/span¿im¡/span¿. *ACM Trans. Model. Comput. Simul.*, 31(3), jul 2021.

[2] Hitesh Linganna Bandari and Binoy B Nair. An end to end learning based ego vehicle speed estimation system. In *2021 IEEE International Power and Renewable Energy Conference (IPRECON)*, pages 1–8, 2021.

[3] Sagar Behere and Martin Törngren. A functional architecture for autonomous driving. In *Proceedings of the First International Workshop on Automotive Software Architecture*, WASA '15, page 3–10, New York, NY, USA, 2015. Association for Computing Machinery.

[4] Sagar Behere and Martin Törngren. A functional architecture for autonomous driving. In *Proceedings of the First International Workshop on Automotive Software Architecture*, WASA '15, page 3–10, New York, NY, USA, 2015. Association for Computing Machinery.

[5] Klaus Bengler, Klaus Dietmayer, Berthold Farber, Markus Maurer, Christoph Stiller, and Hermann Winner. Three decades of driver assistance systems: Review and future perspectives. *IEEE Intelligent Transportation Systems Magazine*, 6(4):6–22, 2014.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[8] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[9] David Fernández Llorca, Antonio Hernández Martínez, and Iván García Daza. Vision-based vehicle speed estimation: A survey. *IET Intelligent Transport Systems*, 15(8):987–1005, 2021.

[10] Bundesamt für Statistik (BFS). *Leistungen des privaten Personenverkehrs auf der Strasse. Methodenbericht 2021. Zeitreihe bis 2020*. Number 20104764. Neuchâtel, Dec 2020.

[11] Bundesamt für Statistik (BFS). Strassenverkehrsunfälle, May 2022.

[12] Jens E. Gayko. *Lane Departure and Lane Keeping*, pages 689–708. Springer London, London, 2012.

[13] Inhwan Han. Car speed estimation based on cross-ratio using video data of car-mounted camera (black box). *Forensic Science International*, 269:89–96, 2016.

[14] Sinan Hasirlioglu, Alexander Kamann, Igor Doric, and Thomas Brandmeier. Test methodology for rain influence on automotive surround sensors. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2242–2247, 2016.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[17] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[18] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics (ToG)*, 35(4):1–11, 2016.

[19] Pushkin Kachroo and Masayoshi Tomizuka. Vehicle traction control and its applications. 1994.

[20] Nobuo Kawaguchi, Junto Nozaki, Takuto Yoshida, Kei Hiroi, Takuro Yonezawa, and Katsuhiko Kaji. End-to-end walking speed estimation method for smartphone pdr using dualcnn-lstm. In *IPIN (Short Papers/Work-in-Progress Papers)*, pages 463–470, 2019.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[22] Xu Qimin, Li Xu, Wu Mingming, Li Bin, and Song Xianghui. A methodology of vehicle speed estimation based on optical flow. In *Proceedings of 2014 IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 33–37, 2014.

[23] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[24] Róbert Adrian Rill. Speed estimation evaluation on the KITTI benchmark based on motion and monocular depth information. *CoRR*, abs/1907.06989, 2019.

[25] Harald Schafer, Eder Santana, Andrew Haden, and Riccardo Biasini. A commute in data: The comma2k19 dataset. *arXiv preprint arXiv:1812.05752*, 2018.

[26] Michael Seiter, Hans-Jörg Mathony, and Peter Knoll. *Parking Assist*, pages 829–864. Springer London, London, 2012.

[27] Bettina Simon. Safe Braking. Start of developing Bosch's anti-lock braking system ABS. https://www.bosch.com/stories/beginnings-of-abs/, September 2018. Accessed: 2022-06-11.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[29] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[32] Hermann Winner, Bernd Danner, and Joachim Steinle. *Adaptive Cruise Control*, pages 478–521. Vieweg+Teubner Verlag, Wiesbaden, 2012.

[33] J. Wojtanowski, M. Zygmunt, M. Kaszczuk, Z. Mierczyk, and M. Muzal. Comparison of 905 nm and 1550 nm semiconductor laser rangefinders' performance deterioration due to adverse environmental conditions. *Opto-Electronics Review*, 22(3):183–190, 2014.

[34] Jian Wu, Jianbo Jiao, Qingxiong Yang, Zheng-Jun Zha, and Xuejin Chen. Ground-aware point cloud semantic segmentation for autonomous driving. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 971–979, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Kyuhwan Yeon, Kyunghan Min, Jaewook Shin, Myoungho Sunwoo, and Manbae Han. Ego-vehicle speed prediction using a long short-term memory based recurrent neural network. *International Journal of Automotive Technology*, 20(4):713–722, Aug 2019.

[36] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. *DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems*, page 132–142. Association for Computing Machinery, New York, NY, USA, 2018.