Vehicle Motion Detection using CNN

Yaqi Zhang* yaqiz@stanford.edu
Billy Wan* xwan@stanford.edu
Wenshun Liu* wl88@stanford.edu

Abstract

This project describes a series of vehicle motion detection experiments for front camera video recordings of moving vehicles gathered from the KITTI dataset [7]. The experiments differ from existing attempts with the leverage of complex CNN architectures and the derivation and combination of input channels that intend to capture different aspects of the raw image including flow direction and object masks. Both transfer learning and built-from-scratch models are trained and evaluated. Input channel choice, fine tuning techniques including down-sampling and hyper-parameter sweep, and other optimizations and observations are discussed. Three aspects of vehicle motion - forward velocity, acceleration, and angular velocity - are optimized using mean-square error loss, with the prediction result consolidated and visualized on top of the video frames. Overall, we achieved 3.536 validation MSE loss and 7.21 test MSE loss.

1. Introduction

Despite the abundant information collected by cameras surrounding us, little of it is processed and understood by machines. This project takes use of real-world recordings from vehicle front-camera in attempt to predict vehicle motion properties. Such motion detection tasks are very useful in analyzing videos pre-recorded by devices like dash cams, providing knowledge of at-the-time vehicle status, aiding other downstream tasks such as driver intention inference when combined with results of other computer vision tasks like object detection, and can be easily extended to other moving objects.

The input to our algorithm is different channels of information obtained from raw video frames of multiple front-camera recordings in different environment settings, including optical flow retrieved using OpenCVs dense optical flow library based on Gunnar Farnebacks algorithm [6], bounding boxes of relevant objects (such as cars and pedestrians) retrieved using Faster-RCNN [22], and the RGB channels of the images themselves. These information are then aggregated in different combinations and fed into a Convolutional Neural Network (CNN) to output 3 most important properties of vehicle motion for each frame - forward speed, forward acceleration, and angular velocity that indicates the direction the vehicle is moving towards.

[0]All authors contributed equally to this work.

2. Related Work

While various attempts have been made to detect vehicle speed from images and videos, most are based on image processing techniques and have their focus on still cameras recordings. The relation between vehicle speed and motion blur in a single image taken by a still camera has been studied extensively [15, 16], but this method falls short in terms of accuracy when the speed is low and motion blur is minimal. The CVS method [20] is a similar approach, where the combination of saturation and value method is used for foreground extraction and vehicle speed detection on images taken by a stationary camera mounted on a freeway. Other vehicle speed estimation systems using roadside traffic management cameras on traffic scenes [5, 19, 23, 25], as well as those requiring reference points [18] have also been developed, but they are all limited to still camera images from a third perspective of traffic scenes. Furthermore, none of the above methods involve neural networks, which have immense potential due to its expressiveness and the sheer amount of data available.

The input channels of our models are made possible by the existence of various object and optical flow detection techniques -

Object detection Region-based CNN (R-CNN) was a breakthrough for object detection tasks, with over 30% improvement in mean average precision (mAP) [9], making the combination of region proposal generation for object localization and deep CNN for classification the state-of-the-art object detection method. Numerous improvements have since been made to the original method for runtime optimization, such as SPP-net [10] and Fast-RCNN [8]. Faster-RCNN [22] further improved region proposal generation by leveraging the power of neural networks

with its introduction of Regional Proposal Network (RPN). Recently, Ashraf et al [3] proposed different enhancements with shallower networks and upsampled input images. On the other end of the spectrum, You Only Look Once (YOLO) [21] and SSD [17] methods aim to tackle object detection by combining the tasks of generating region proposals and classifying them into one network. They are computationally much less expensive but also have lower accuracies than the R-CNN family of methods, and are thus more suitable for real-time applications on embedded devices. For our purposes, accuracy is the most important metric, so we will use the Faster-RCNN network.

Optical flow detection It has been used extensively for autonomous driving tasks such as obstacle detection [14]

3. Dataset and Features

We use the KITTI [7] raw data recordings as our dataset, which consist of color stereo sequences in various environment settings (such as city or residential areas) recorded at 10 Hz with a resolution of 1242×375. An example image of a video frame is shown in Figure 1. A total of 3470 frames are obtained and used from 24 recordings sampled at the rate of 33%, where frames from the largest recording (659 frames) is reserved for testing, and the others are randomly assigned to either training or validation set with a rough $70\% - 30\%$ split. Each frame of the dataset is labeled with the truth data of the interested motion properties. Roughly speaking, we have forward speed $\in [0, 20]$ m/s, forward acceleration $\in [-\frac{\pi}{2}, \frac{\pi}{2}]$ m/s^2, and angular velocity $\in [-1, 1]$ rad/s.

The decision to sample the dataset comes from the trade-off between disk space constrain and estimated performance gain from consecutive frames. The disk space constrain comes from the fact that generated optical flow for each frame is cached to speed up training and remove duplicated work, and thus occupies considerable amount of disk space. Consecutive images are expected to contribute little to the training process as they are overly similar to each other in terms of color distribution, optical flow movement, and surrounding object presence, and adds additional overhead in the computation time. As a result, we have decided to sample every three images as the raw input.

Dense optical flows, object bounding boxes, and image RGB channels are then obtained from the raw input with methods detailed in Section 4. The computed optical flows are of the same dimension as the raw image and have two channels representing the horizontal and vertical directions of the flow vector. For the generated object bounding boxes, a separate binary mask is created for each detected object class with 1 for pixels locating inside the bounding boxes and 0 otherwise. The addition of such masks for classes like

Figure 1. Sample KITTI image.

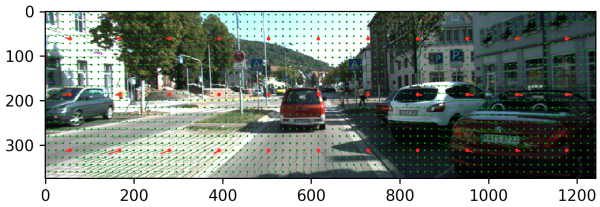

Figure 2. Averaged optical flow with 3 x 11 segments.

vehicle and pedestrian will likely improve prediction result because their relative speed to the subject vehicle can vary. For example, the relative speed of a moving car in front of the camera should be much smaller compare to the ones parked on the roadside. Lastly, RGB channels are created to help the model identify regions of the image that share similar colors (such as the ground).

Such channels of information have the potential issue of being extremely noisy, as values are gathered at pixel level. To cope with this issue, we experimented with downsampling the input by slicing the channels multiple numbers of segments, where for optical flow and RBG, values with each segment are simply averaged, and for each of the binary object masks, each segment is assigned to 1 if more pixels are part of the bounding box, and 0 otherwise. Figure 2 shows an example of the averaged optical flow (red arrows) from the original noisy inputs (green arrows). Detailed analysis on this downsample experiment is further discussed in Section 5.

The channels of inputs are then combined to feed into the CNN model of interest. As shown in Figure 3, 5 channel combinations are experimented, with the result discussed in Section 5. Optical flow is included in most modes of experiment, as it is expected to be the most indicative when it comes to motion detection. Different combinations of object mask and RGB channel are overlayed on top to experiment the effect of different supporting information. We also experimented with the input of pure RGB channel.

4. Methods

Our optical flow computation leverages the OpenCV [4] implementation of Gunnar Farnebäck's two-frame motion estimation algorithm based on polynomial expansion

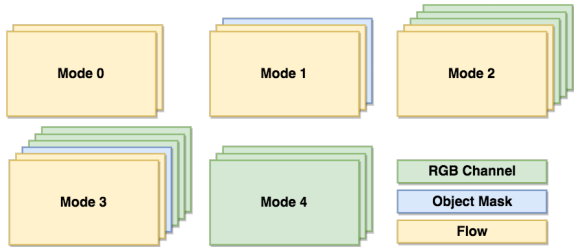

Figure 3. Input channel combination.

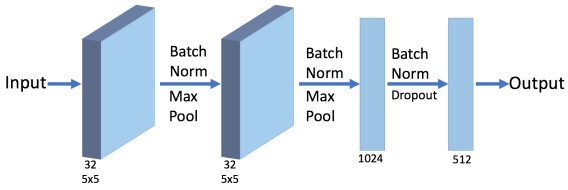

Figure 4. Architecture of 2-layer baseline CNN.

[6], where quadratic polynomials are used to approximate neighborhoods of consecutive frames and estimate the displacement fields between them. Dense optical flow vectors of each pixel represent the direction and magnitude of the pixel's movement from one frame to the next, and are thus expected to correlate tightly with the speed of the camera mounted on the vehicle.

Our object detection process makes use of the Faster-RCNN [22] algorithm implemented in Tensorflow [2], which is composed of a deep fully convolutional RPN that proposes regions of candidate objects, followed by a Fast-RCNN [8] with VGG-16 [24] detector that performs image classification on top and outputs class softmax probabilities and per-class bounding box offsets. Attention mechanisms are used to point the detector to the appropriate proposed regions. Based on the object detection results, we construct binary object masks for the class car.

Given our input data, we implement three different CNN architectures to output the predicted forward speed, acceleration, and angular velocity. Figure 4 shows the baseline 2-layer CNN architecture, which consists of 2x conv-relu-batch_norm-max_pooling layers, 2x affine-relu-batch_norm-dropout layers.

Figure 5 shows the architecture of AlexNet used for this project. This is the same architecture as originally proposed in [13] and later revised in [12]. The architecture is chosen because it is deeper than but not radically different from our baseline. We also experiment with transfer learning for AlexNet, where we initialize the middle convolutional layers with pre-trained weights [1].

Figure 6 shows the architecture of 17-layer ResNet used for this project. The core concept of ResNet is that deep neural networks should at least achieve the same performance as shallower networks, so adding the input of each residual block to the output allows the network to fall back on its shallower counterpart. Batch normalization is performed between every pair of adjacent convolutional layers, and down-sampling is performed at the first convolutional layer of the last three residual blocks by a convolution operation with stride 2.

A final affine layer with output size 3 is appended to all three models to output the three motion predictions. Given that this task is a regression problem instead of classification, we use the standard mean-squared error (MSE) loss as the metric to evaluate the model's predictions. Intuitively, the MSE loss measures on average how close the predictions are to the ground truth labels. Given ground truths y and predictions $\hat{y}$ for a batch size of N, the MSE loss L can be computed as

$$L = \frac{1}{N}\sum_{1}^{N}(y - \hat{y})^2. \tag{1}$$

By default, the losses of each predicted motion property are added together as a total loss and optimized accordingly. Considering that forward acceleration and angular velocity are of smaller magnitude than forward speed, they may contribute to the total loss unevenly and the model will thus prioritize optimizing speed, we also experiment with keeping separate losses for each property and optimize them individually.

The Adam [11] optimizer is used for all 3 architectures with an exponential learning rate decay schedule at a rate of 0.95 every 100 steps. In other words, the learning rate α at any given step t is

$$\alpha(t) = \alpha_0 e^{\frac{t}{100}}. \tag{2}$$

In this project, we first compare the results of different input preprocessing, including the five input channel combinations and the amount of vertical and horizontal segmentations in calculating averaged flow. We then compare the different CNN architectures - baseline, AlexNet, ResNet, and AlexNet with transfer learning. The hyperparameters of our model include the initial learning rate α_0, dropout rate for baseline CNN and AlexNet, and batch size, which we fine tune in our experiments.

5. Experiment Results & Discussion

In this section, we present sets of experiments we performed to improve and evaluate our model. In Section 5.1, we present evaluation on preprocessing to our input data, including effectiveness of down-sample data and using different combination of channel type as explained in Section 3. We also present our result on hyper-parameter tunning in Section 5.3. Due to huge space of our hyper-parameters

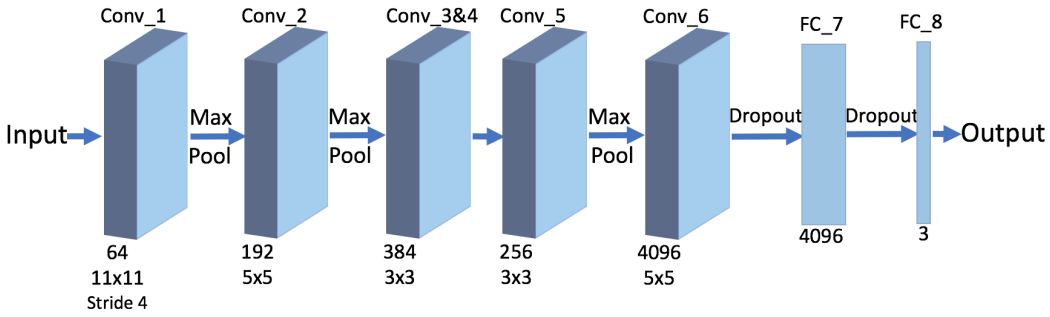

Figure 5. Architecture of AlexNet CNN.

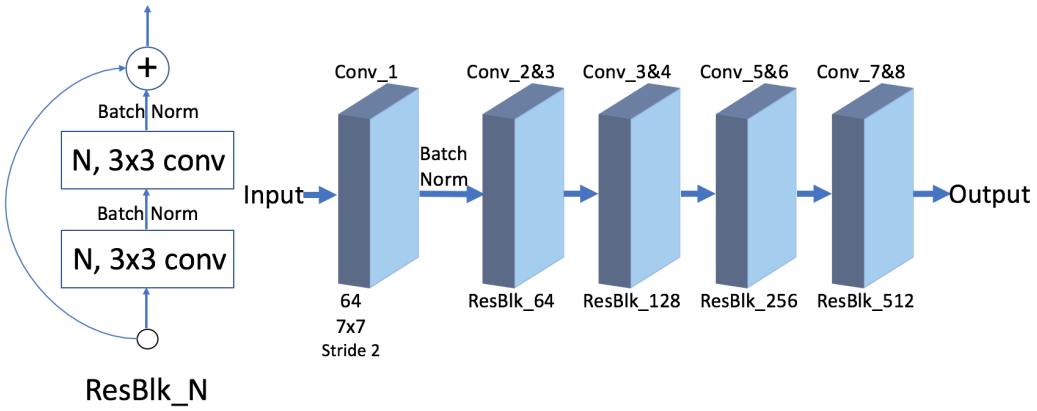

Figure 6. Architecture of ResNet-17 CNN.

and long duration to train a single epoch, we only used simple cross-validation with 16 videos for training, 7 videos for validation,and 1 video for testing. Furthermore, we presented our study on impact of sampling approach of validation set on hyper-parameter tuning. Finally, we present our best results among all models and hyper parameter and evaluate our results both qualitatively and quantitatively.

5.1. Input Comparison

5.1.1 Input Down-sampling

In order to reduce noise in input optical flow, we introduce down-sampling in our input as described Section 3. In input down-sample, number of vertical and horizontal segmentations are hyper-parameters, where the more slices are taken in each direction, the more noisy the input will be. For example, a 50×100 segmentation indicates the input tensor is segmented into 50×100 grids and each grids produces a averaged value for each channel of the input. The most extreme case where every pixel is a grid is equivalent to the original image. Figure 7 shows result MSE loss for different input down-sample segmentations. For baseline, the best validation result is achieved with segmentation at 100×300. As expected, when imaged is segmented into very few blocks, the important information associated flow vector is lost, and hence increase validation loss. There are some variation in segmentation between 100×300 to the image original size, which is lightly introduced by noise. For ResNet, since the model contains much more parameters than baseline, down-sample input helps alleviate overfitting and improve model runtime very significantly. Due to memory, disk, and run-time constrain, we can only fit a 17 layer RedNet as supposed to the standard 50 or 100 layer. As a result, the laster layer of convolution in ResNet is very large and introduce lots of parameters in the first fully connected layer. With input down-sampling, we are able to fit ResNet with batch size equals to 32 in GPU memory, while the original image can fits only batch size of 2, which has really noisy updates in stochastic gradient descent. For AlexNet since output of the last convolution layer is already very small, input down-sample will cause negative dimension in max-pooling.

5.1.2 Input Channel Types

In this section we present evaluation of 5 combinations of input channels types described in 3: [flow, flow+objmask,

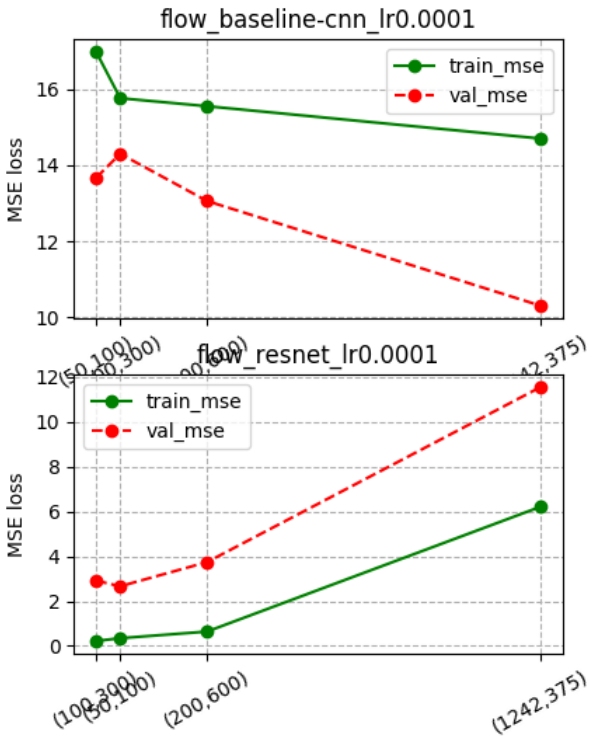

Figure 7. MSE Loss vs. Input Down-sampling.

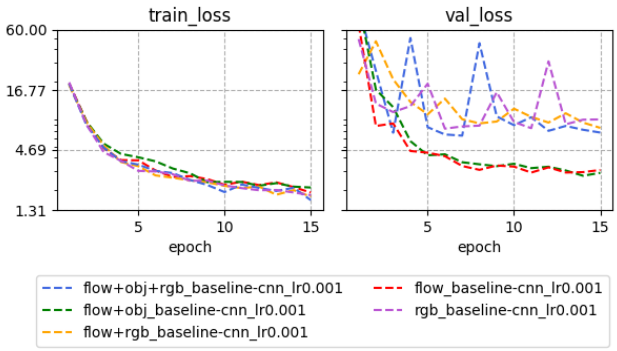

Figure 8. Input channel type comparison.

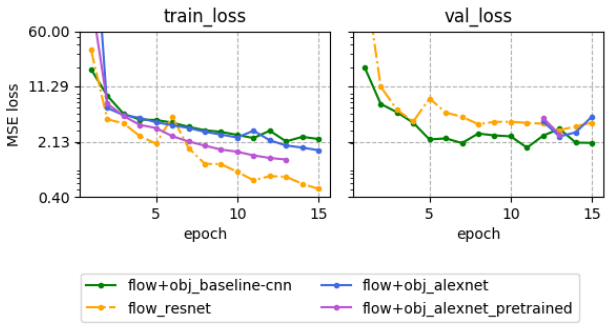

Figure 9. Comparison of different models.

flow+rgb, flow+objmask+rgb, rgb](Figure 3). The final training MSE of all modes are similar, but mode 0 (flow) and 1 (flow+objmask) can achieve much lower validation MSE, with mode 1 having the lower of the two. The results validate our assumption that optical flow are closely related to the speed of the camera. The relatively poor performance of modes 2-4 can be explained by the fact that it is hard to tell the speed of the camera solely from a single frame image, and rather the three motion properties we are trying to predict pertain much more closely to the difference between two adjacent images.

5.2. Model Comparison

Figure 9 shows the results of using different models for speed prediction. For certain models, we only performed validation in last few iterations to speed up the training process. Overall all, ResNet-17 is able of overfit the training data the most as it has the most parameters. The next best model for training set is AlexNet with partially transfered weights. We only transfered the convolution layers except the first layer because our input dimension does match the original model. The next best model is AlexNet trained from scratch followed by baseline is the worst model in training set. For validation however, our best result is given by the baseline mode. This is major due to the baseline model is much faster to train, and as a result we could search the hyper-parameter space much more extensively. AlexNet-pretrained and AlexNet are the second best and ResNet performs less well on the validation set as they have much larger hyper parameter space and takes much longer to train.

5.3. Hyperparameter Tunning

5.3.1 Learning Rate

Figure 10 shows the results of initial learning rate tuning on images downsampled to 100x300. As expected, both the baseline CNN and ResNet-17 do not perform well when the initial learning rate is too large, i.e. greater than 10^{-2}. The AlexNet and pre-trained AlexNet model (not shown) exhibit similar behavior to the baseline CNN and ResNet-17, respectively, so we choose 0.0001 for ResNet-17 and pre-trained AlexNet, and 0.001 for the baseline and AlexNet models, with the observation that the best learning rate for ResNet is lower than that of the other models.

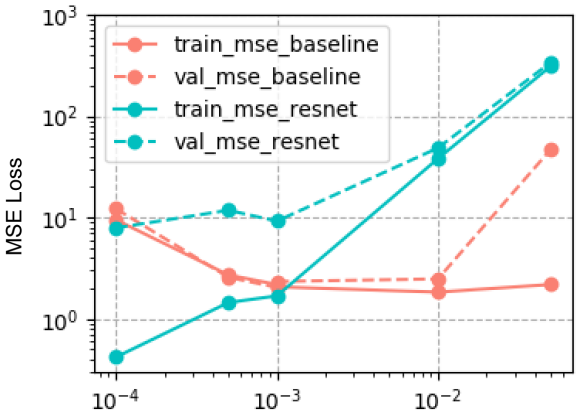

Figure 10. Initial learning rate tuning results.

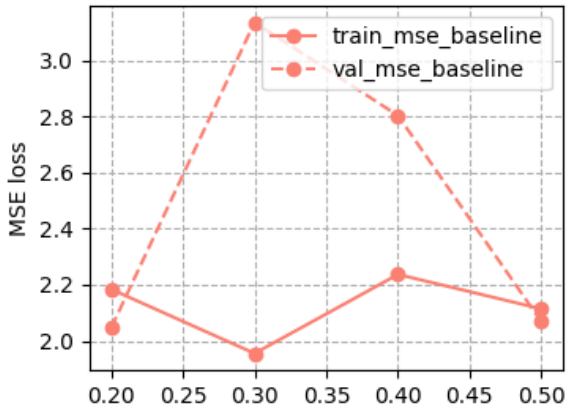

Figure 11. Dropout rate tuning results.

5.3.2 Dropout

Figure 11 shows the results of dropout rate tuning on images down-sampled to 100x300. Overall, dropout rate from 0.2 to 0.5 does not have as significant an effect on the train and validation MSE's as the learning rate. This can be explained by the fact that all the input images are already averaged and down-sampled, which essentially adds a regularization effect on the model. A dropout value of 0.2, the value that achieve the best validation MSE in this experiment, is chosen for the remaining experiments with this model.

5.3.3 Batch Size

Figure 12 illustrates the results of batch size (b) turning and the effect it has on train and validation results. For $b = 16$ and $b = 32$, the figure shows the consolidated result of a decent number of experiments that have been conducted. For

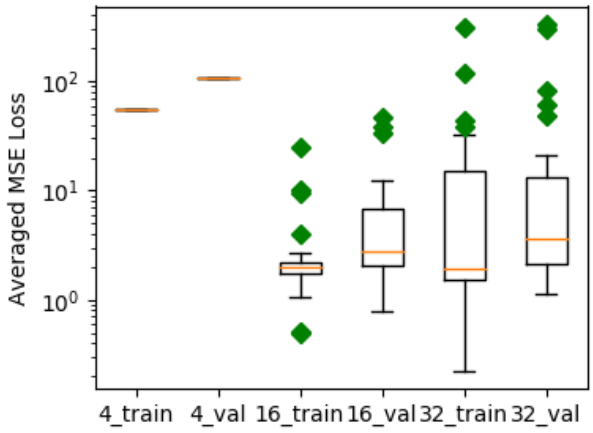

Figure 12. Loss distribution with different batch size.

$b = 4$, only a single point is included, as it is the biggest batch size that can fit under our GPU's memory constraint for running ResNet without any down-sampling of the data. From this figure we can see that, the models do not perform well when batch size is too small (as in the case of $b = 4$). This is because gradients computed from limited frames of data could be overly noisy and unstable. We can also observe that average performance is roughly the same for $b = 16$ and $b = 32$, with sightly smaller mean training MSE and slightly higher validation MSE for the latter (We do see that results for $b = 32$ are more spread, which could be due to the fact that more of our initial experiments were performance on $b = 32$, where the hyper-parameter choice in general is less optimal). Based on this result, we take $b = 16$ as the batch size as result of its less memory requirement, faster computation time, and the observed similarity in performance.

5.4. Optimization Method

All models described in Section 4 optimize on the *summation* of the individual losses, while the absolute difference in angular velocity in radius is in general much smaller than the other two. As a result, the models could be subjective to a skewed update as the loss of angular velocity is not visible (a consequence of this behavior is described in Section 5.7). One approach is to normalize three losses before summing them up. However, that would require making assumption on the maximum value of three losses, which cannot be generalize to arbitrary video. In attempt to rectify this issue, an experiment is made to update gradient of weights with respect to three losses in three individual steps, hoping that in this way equal attention will be given to each of the properties of interest. The result of this experiment was very unsatisfying, with blown-up MSE for both training and testing. We believe this is due to over-stepping when three

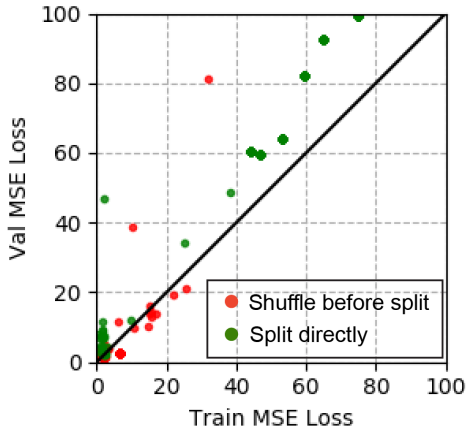

Figure 13. Validation Loss vs train loss with different validation sampling approach.

updates are made for each batch of data. Although the three losses are highly correlated, their loss space might be very noisy and three individual updates might not necessary step correct direction of gradient descent. Other potential methods worth exploring are discussed in Section 6.

5.5. Validation Sample Scheme

An interesting experiment we run is on the different sampling schemes of the training and validation datasets. After consolidating all video frames, one approach is to shuffle the entire dataset randomly, and then split according to the preset ratios. Another approach is to directly split the dataset without shuffling. Figure 13 shows a comparison of the train and validation MSE's of these two approaches. The majority of red dots are below the $y = x$ line, indicating that training loss is overwhelmingly larger than validation loss, while the inverse case is true for the green dots. The results can be explained by the higher chance of validation data coming from the same video as the training data and thus being more similar if the dataset is shuffled before splitting. This is thus an incorrect sampling approach, and we choose the second approach instead for all of our experiments.

5.6. Quantitative Evaluation

Table 1 shows the train, validation, and test MSE of our best model, with the baseline architecture trained on images averaged and downsampled to 100x300 and optical flow and object detection masks as input. It is expected that the test MSE is higher but similar to the train and validation MSE's due to their innate differences. Table 2 further breaks down the test MSE into the MSE of each motion property. A test MSE of 6.832 for forward speed translates to an average difference of 2.61 m/s, or a roughly 13% error.

Table 1. Best overall train, validation & test MSE

Train	Validation	Test
4.596	3.536	7.21

Table 2. Test MSE for each motion property

Forward Speed	Forward Acceleration	Angular Velocity
6.832	0.366	0.012

5.7. Qualitative Evaluation

Based on the predicted values, a final state is computed that consolidates the vehicle's motion in four categories: Still, Forward, Turning Left, and Turning Right. Forward velocity fv determines the vehicle's "stillness", where vehicle is predicted as moving "Forward" when $fv > 2m/s$. For any moving vehicle, its steeling angle is further categorized based on angular velocity av, where vehicle is predicted as "Turning Left" when $av > 2°$, and "Turning Right" when $av < -2°$.

Choosing the appropriate thresholds for the consolidated results is a rather subjective process. The KITTI dataset [7] does not provide such kinds of truth data, and as a result, we visualize the predicted values on the video frames for intuitive understanding of the results, and pick the thresholds based on observations on the validation set.

The visualized results roughly match the loss we see from the quantitative results. Figure 14 and 15 show two example visualizations on the testing frames, where the red boxes represent the input object mask and textbox underneath the frames show side-by-side comparison of the properties of interest, as well as the final consolidated status. Several observations are made from these visualizations:

Uneven Error Distribution Among all three predictions, angular velocity appear to be much more error-prone compare to the others. This observation could be caused by the fact that we currently train a single model to predict all interested properties and angular velocity's contribution to the overall loss is comparatively small due to the small magnitude in radius.

Skewed Status Prediction Figure 15 is an example of the model falsely believing that the vehicle is making a left turn. In fact, we have observed the model is in general skewed towards believing that the vehicle is making a left turn. This observation could be due to the fact that all video recordings in the dataset are from right-hand roads, where pixels on the right are expected to have higher optical flows, and they are closer to the vehicle and move faster. Coincidentally, this behavior is similar to that of a left-turning vehicle, where objects on the right-hand side are expected to have more dramatic optical flow change. In fact, we do observe bet-

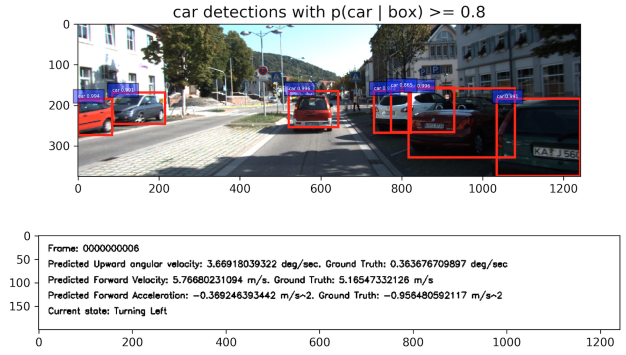

Figure 14. Visualized prediction result 1

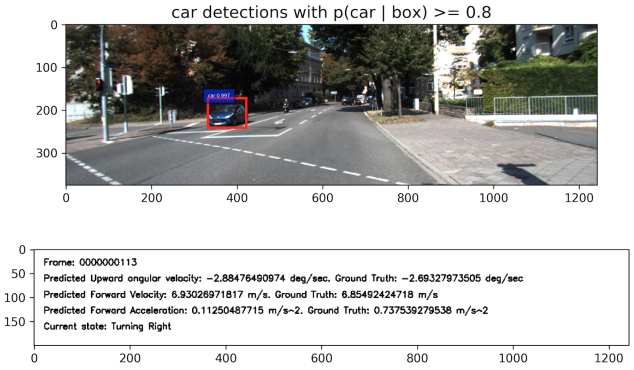

Figure 15. Visualized prediction result 2

ter results when the object masks are included in the input channels, and expect to see more apparent differences with expanded dataset and more occurrences of turning.

6. Conclusion & Future Work

In this project, we propose a framework to predict vehicle motion status using videos recorded by vehicle-mounted front cameras. We extract multiple types of information from each image, apply preprocessing on the images, and perform experiments on different combinations of input types and CNN architectures. After hyperparameter tuning, our model is able to achieve great performance on all three motion properties, and make reasonable inferences about the status of the vehicle. The video visualization further confirms and complements our quantitative results.

Given more time and resources, we can still improve some aspects of our project. The entire dataset contain only 24 videos, which is not enough for our methods to generalize well to mass application. It may also prove better to train a separate model for each motion property instead of adding losses together or training the same model with three objectives, a plan not executed due to time constraints. Finally, combined with other computer vision tasks such as sign and road detection, more meaningful inferences about vehicle status and driver intention can also be made.

References

[1] Alexnet implementation + weights in tensorflow. `http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/`. Accessed: 2017-06-11.

[2] Faster-rcnn_tf. `https://github.com/smallcorgi/Faster-RCNN_TF`. Accessed: 2017-06-11.

[3] K. Ashraf, B. Wu, F. N. Iandola, M. W. Moskewicz, and K. Keutzer. Shallow networks for high-accuracy road object-detection. *CoRR*, abs/1606.01561, 2016.

[4] G. Bradski. *Dr. Dobb's Journal of Software Tools*.

[5] D. J. Dailey, F. W. Cathey, and S. Pumrin. An algorithm to estimate mean traffic speed using uncalibrated cameras. *IEEE Transactions on Intelligent Transportation Systems*, 1(2):98–107, 2000.

[6] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag.

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

[8] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[9] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[14] W. Kruger, W. Enkelmann, and S. Rossle. Real-time estimation and tracking of optical flow vectors for obstacle detection. In *Intelligent Vehicles' 95 Symposium., Proceedings of the*, pages 304–309. IEEE, 1995.

[15] H. Lin. Vehicle speed detection and identification from a single motion blurred image. In *7th IEEE Workshop on Applications of Computer Vision / IEEE Workshop on Motion and Video Computing (WACV/MOTION 2005), 5-7 January 2005, Breckenridge, CO, USA*, pages 461–467, 2005.

[16] H. Lin, K. Li, and C. Chang. Vehicle speed detection from a single motion blurred image. *Image Vision Comput.*, 26(10):1327–1337, 2008.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, pages 21–37, 2016.

[18] C. Pornpanomchai and K. Kongkittisan. Vehicle speed detection system. In *Signal and Image Processing Applications (ICSIPA), 2009 IEEE International Conference on*, pages 135–139. IEEE, 2009.

[19] S. Pumrin and D. Dailey. Roadside camera motion detection for automated speed measurement. In *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on*, pages 147–151. IEEE, 2002.

[20] A. G. Rad, A. Dehghani, and M. R. Karim. Vehicle speed detection in video image sequences using cvs method. *International Journal of Physical Sciences*, 5(17):2555–2563, 2010.

[21] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[22] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[23] T. N. Schoepflin and D. J. Dailey. Dynamic camera calibration of roadside traffic management cameras for vehicle speed estimation. *IEEE Transactions on Intelligent Transportation Systems*, 4(2):90–98, 2003.

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[25] A. R. YG, S. Kumar, H. Amaresh, and H. Chirag. Real-time speed estimation of vehicles from uncalibrated view-independent traffic cameras. In *TENCON 2015-2015 IEEE Region 10 Conference*, pages 1–6. IEEE, 2015.