A machine learning approach to predict a vehicles velocity using dashcam video

Florian Wolf
Department of Mathematics and Statistics, University Konstanz
Konstanz, Germany
florian.2.wolf@uni-konstanz.de

Franz Herbst
Department of Physics, University Konstanz
Konstanz, Germany
franz.herbst@uni-konstanz.de

Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

***Index Terms*—deep learning, computer vision, visual odometry, dense optical flow, siamese network**

I. Introduction

Autonomous driving is considered to have a key role in the future of human mobility. Recently the topic has therefore gained great attention in research, economy and politics [1]. In this project we approached this topic in a related but more simplified setup and tried out several machine learning techniques and network architectures.

A. Aim of the project

In this project we tried to predict the velocity of a car based on a dashcam video. This was inspired by the *coma_ai* speed challenge[1] published in 2018. To achieve this goal we used optical flow analysis to preprocess the data and neuronal networks with different structures (classical and siamese) to predict the velocity. In order to measure the quality of the predictions we used their mean squared errors of the measured velocity.

[1]https://github.com/commaai/speedchallenge

II. Data collection, analysis and preprocessing

To start the project we used the database given in the *coma_ai* speed challenge project, which consists of a 17 minute training video (20400 frames) and the corresponding car velocity, as well as a 9 minute test video without labels to validate the model on an unknown data set.

A. Data collection

Due to our limited computational power we mainly used this data set to study different training techniques and model architectures. Still we could not expect good generalization for this rather small pool of training data. We therefore developed a technique to acquire more data that needed minimal resources. We used the open source smartphone apps *open camera* and *open street maps* to cast a video while driving and map the velocity using GPS at the same time. So we were able to train our final model with 1 h 40 min of video data.

B. Data analysis

In order to evaluate our raw data with the model we analysed our dataset concerning the recorded images and driving scenarios. The original frames have a size of $640 \times 380 \times 3$ pixels (RGB). To reduce the computation time, we cut the borders of the frames to exclude parts not needed for detection and sampled them down to half of their pixel size.

To visualize the training process in different driving scenarios, we plotted the velocity in dependence of the frame number (Fig. 2). After additional evaluation of the video we were able to classify three different driving scenarios: *highway driving* with high and relatively steady velocities, *stop and go* with low and fluctuating velocities as well as *city driving* with very abrupt speed changes between medium and very low velocities. Every driving scenario is represented with about one third of the training data.

We would then evaluate the predictions of each model $p(x_i)$ to the input data x_i using the mean squared error

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(p(x_i) - y_i)^2 \tag{1}$$

to real value of the velocity y_i. To classify the resulting value we on one hand used the velocity plot and the following rough

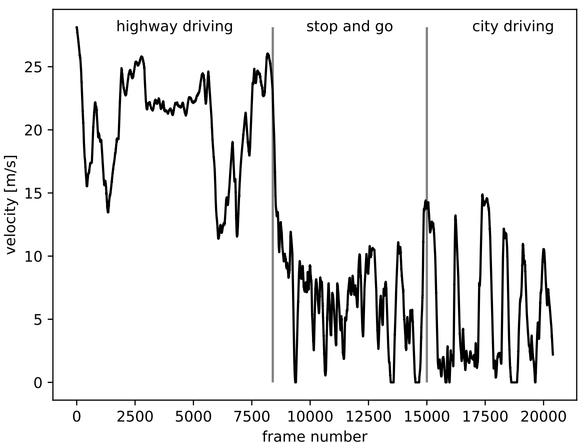

Fig. 1: Distribution of the velocities of all frames in the training video, including the three categories.

classifications[2]: $\sqrt{\mathcal{L}} \gtrsim 16$ *no fitting*, $16 \gtrsim \sqrt{\mathcal{L}} \gtrsim 10$ *average velocity fitted*, $10 \gtrsim \sqrt{\mathcal{L}} \gtrsim 5$ *scenarios detected*, $5 \gtrsim \sqrt{\mathcal{L}} \gtrsim 1$ *variances within scenarios detected*, $1 \gtrsim \sqrt{\mathcal{L}}$ *perfect fitting*.

For the training process we used a splitting of 80% training data and 20% validation data. Initially we did a hard splitting of the entire data, but this neglects the distribution of different driving scenarios in the video. Therefore we went on first splitting the data into driving scenarios with then assigning training data and validation data on each.

C. *Preprocessing*

Each of the provided frames has a size of $(640, 480, 3)$ pixels. Due to computational limitations, we decided to cut off the last 60 pixels from the lower border, to remove a black frame inside the car, which did not have any effect on the optical flow. Furthermore, we cut the frame size in half and calculated the optical flow using the "Farneback pyramid method" [2] with the following parameters

$$\begin{aligned} \text{pyramid levels} &:= 3 \\ \text{pyramid scaling} &:= 0.5 \\ \text{window size} &:= 6 \\ \text{pixel neighborhood size} &:= 5 \\ \text{SD of the gaussian filter} &:= 1.1 \end{aligned}$$

We choose three pyramid levels, because we wanted the calculations to be more accurate. To decrease the training duration, we halved the size of the optical flow frames again, resulting in a resolution of $(160, 105, 3)$ pixels per frame. As we used a window size of six pixels, a comparison between the original optical flow and the down sampled one lead to the result, that we do not loose a lot of information.

[2] These rules do only apply on datasets with an equal distribution of all driving scenarios

The optical flow calculation returns the magnitude and the angle of the flow vectors, which we transformed into polar coordinates. To get an RGB image representing the optical flow of two consecutive frames, we normalized the magnitudes and put them into the third channel of the frame. The values of the second channel were all set to the value 255. We then multiplied the angle with the factor $180/(2\pi)$ and set this value for the first channel.

We wanted to see, if the model performs better using the dashcam frames as additional material. Therefore, we did the same down sampling with the frames.

III. Method selection and architecture

The prediction of the vehicles speed is a non-linear regression task , so the choice of a neural network is reasonable. Recent architectures we discussed in the lecture (ResNet, GoogLeNet, etc.) have shown that using multiple stacked convolution layers combined with stacked dense layers, perform well on image classification tasks. Therefore the choice of a convolutional neural network is justified.

A. *Initial Network*

As an initial architecture we decided to give the model of [3] a try. As the group used it for self-driving cars, the model has enough complexity to handle a task like ours and with the amount of layers, we had a lot of possibilities to fine-tune and improve the model.

HERE SHOULD BE AN IMAGE OF THE NETWORKS ARCHITECTURE

In a first approach we tried the raw model, gaining a MSE of around 18-20 on testing and under 3 on training. One can clearly see, that the model really overfits the train dataset.

B. *Siamese approach*

In order to improve our results we expanded our model to also use the original image: Siamese approach

Initial splitting blocks of 100 frames size with 80% training data, 20% test data; 20 epochs (train error: 0.411, test error: 2.70)

New splitting: each driving scenario 80% training data, 20% test data: (train error: 0.23, test error: 29.7)

IV. Tuning of the model

To speed up the training we used batch normalization layers [4] and we tested different activation functions, to improve the performance of the model.

As proposed in the lecture, we used

$$\mathrm{ReLu} : \mathbb{R} \to \mathbb{R}_0^+, x \mapsto \max\{0, x\}$$

as the initial activation function. Using the ReLU function and 15 epochs for training, we achieved a MSE of around 15 on the testing set. We ran the code multiple times, to ensure this result holds. This result was not really promising, so we wanted to decrease the error by modifying the model even more.

As we still had a lot of issues with overfits, we decided to include a dropout layer, according to [5]. We tried different numbers and positions for dropout layers, but using one layer with a dropout probability of $p = 0.5$ after the third convolutional layer seemed to work best.

To solve the problem of dead neurons[3] of the ReLU function, we tried the leakyReLU function

$$\mathrm{leakyReLU}: \mathbb{R} \to \mathbb{R}, x \mapsto \begin{cases} x, x \geq 0 \\ c \cdot x, x < 0 \end{cases}$$

with a hyperparameter $c = 0.01$. Using the leakyReLU function, we achieved a MSE of around 12 on the testing set and under 3 on the training set. All of the models were trained with only eight epochs, as we still had problems with overfits.

We identified three possible reasons for our poor results:

(i) Too complex model, as the paper used it for autonomous driving or we put too little information into the model.
(ii) Problems with different brightnesses in the frames (lack of generality), which leads to unstable calculations in the optical flow, as the optical flow is quite sensitive to brightness. (Explain how in the train part the sky is quite dark and in the city (end) the sky is bright)
(iii) Too naive/ambiguous splitting of the data into train and testing set, as both datasets seem to represent totally different scenarios in the road traffic.

We came up with the following approaches to solve these problems

(i) Simplify the model by using Pooling (we will try average and max pooling), to get more compression and we tried on the other hand to feed in more information into the model, by using a linear combination of the optical flow and the raw frames itself, and we tried using a siamese network, to put simultaneously the of and raw frames into the convolutional layers.
(ii) We wanted to try adding some additional noise into the frames before calculating the optical flow, to make the calculation more robust against brightness changes. As intentionally adding noise to a frame is quite atypical in computer vision, this idea looked quite interesting.
(iii) Use another splitting. To get a better ratio between highway and city driving scenarios, we decided to split the data into blocks of 100 frames and take the first 80 for training and the last 20 for testing. Therefore our model should have seen some city driving.

A. Pooling layers with initial splitting

We added two generic pooling layers to the network to reduce the number of parameters of the model. One after the second convolutional layer and the second one right before the fully connected layers start[4].We tested maximum and average pooling with the following parameters

$$\begin{aligned} \text{kernel size} &:= 2 \times 2 \\ \text{stride} &:= 2 \\ \text{padding} &:= 1 \\ \text{dilatation} &:= \text{None} \end{aligned}$$

The implementation of pooling layers helped a lot, as now the loss on the train and test data seemed to decrease nearly equally. Our results with the initial splitting are shown in Table I. We gave the network with max pooling a try with 15 epochs, as the loss on the train and test set was decreasing in a pretty stable manner. Using this network, we achieved for the first time a MSE of under 10 on the test set, which is according to our assumptions in Section I-A a pretty good result — especially, as we trained the model mostly on highway scenes and tested it only in city driving scenarios.

Initial splitting, 8 epochs	ReLU		leakyReLU	
	Train	Test	Train	Test
No pooling	2.85	12.08	2.45	10.75
Max pooling	5.62	11.82	5.52	10.29
Max pooling (15 epochs)	-	-	**3.22**	**9.63**
Average pooling	7.70	11.40	6.08	13.09

TABLE I: MSE results of the network using different pooling strategies, one dropout layers, two different activation functions and the initial splitting. We trained each of the models for eight epochs.

B. New splitting and siamese network

We decided to take a closer look at the different road traffic scenarios in the video and segmented the frames into three category. The first category consists of highway driving scenes (from minute 0:00 to minute 7:30), the second one contains frames of the car being in a stop and go scenario on the highway (from minute 7:31 to minute 15:00) and the third category consists of city driving scenes (from minute 15:01 to the end). We visualized the velocity distribution and the categories in Fig. 2. We shuffled the frames of each block randomly and used again the 80/20 rule to split each of the category blocks into train and test data.

C. Augmented brightness

STILL UNDER CONSTRUCTION

V. Training of the models

we used bla bla as optimizers and bla bla as schedulers, for each model we tried different combinations and bla bla

[3]One can clearly see in the definition of the ReLU function, that neurons with a value below zero cannot participate in the learning process.

[4]Indeed, the number of parameters decreased from a total of 636.225 trainable parameters to 156.225. Therefore the number of parameters decreased by a factor of 4. We calculated these numbers using the tool "PyTorch summary"(https://github.com/sksq96/pytorch-summary).

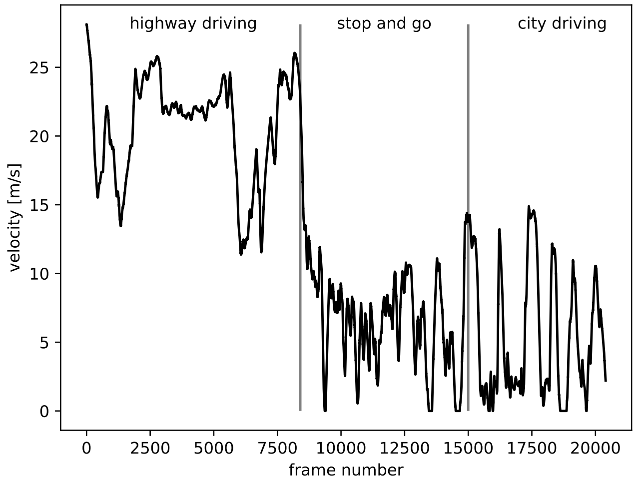

Fig. 2: Distribution of the velocities of all frames in the training video, including the three categories.

VI. New Approach using siamese network for two consecutive frames without optical flow

inspired by https://arxiv.org/pdf/1709.08429.pdf

VII. Error analysis and results

VIII. Further work

Use MUCH MUCH MUCH MUCH MUCH more data!

Create Validation Data that is not connected to training data and still covers all situations

Acknowledgment

We like to thank bla bla bla

References

[1] Markus Maurer et al. *Autonomous Driving: Technical, Legal and Social Aspects*. Berlin, Heidelberg: Springer, 2016.

[2] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Scandinavian Conference on Image Analysis* (2003), pp. 363–370.

[3] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". In: (Apr. 2016). URL: https://arxiv.org/pdf/1604.07316v1.pdf.

[4] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: (Feb. 2015). URL: https://arxiv.org/pdf/1502.03167.pdf.

[5] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.