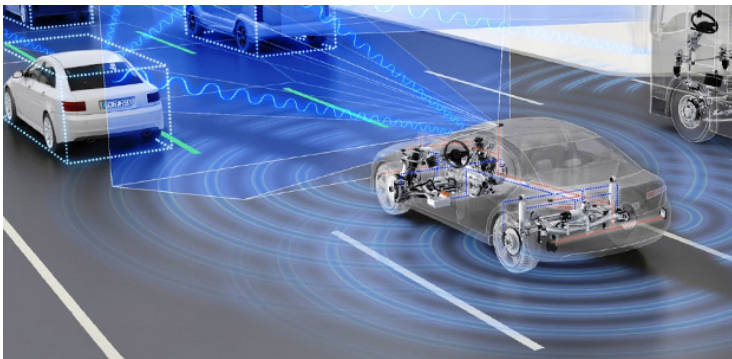


shipout/lastpageshipout/foregroundshipout/background  
shipout/lastpage



<https://www.kfz-mag.de/bild/title/0/171180autonomes-fahren.jpg>

# Predicting a vehicles velocity using dashcam footage

## A deep learning approach

Florian Wolf, Department of Mathematics and Statistics  
Franz Herbst, Department of Physics

Machine Learning using Matlab  
Universität Konstanz

March 10, 2021

# Table of contents

- 1 Motivation and initial Dataset
- 2 Analysis of the dataset
- 3 Preprocessing using optical flow
- 4 Method selection and architecture
- 5 Different optimization strategies
  - Change components of the initial architecture
  - Siamese approach
  - Problems and possible solutions
- 6 Current and further work
  - Acquire more data
  - Additional noise
- 7 Summary and demo

# The “comma ai speed challenge”<sup>1</sup>

## Motivation

- Autonomous driving is currently one of the most prominent problems in machine learning
- But quite hard to set up on a desktop pc
- Predicting a vehicles velocity from video footage is a related, but also a much more simplified task

## Initial Dataset:

- Training video with 20400 frames (20 fps)
- Data file with velocity of the car at each frame
- Test video with 10798 frames (20 fps)

## Evaluation:

- The mean squared error (MSE) is used to measure performance

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

---

<sup>1</sup><https://github.com/commaai/speedchallenge>

# Analysis of the dataset

## Video data:

- Frame size of (640, 480, 3) pixels
- Cut off last 60 pixels, to remove black frame inside the car
- Sample down the frame to half its size, to reduce computation time



Original frame



Cut off the last 60 pixels, downsampled

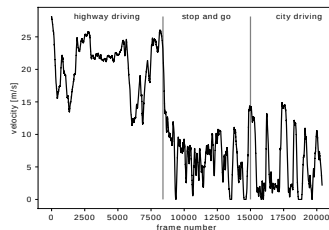
# Analysis of the dataset

## Splitting of the dataset

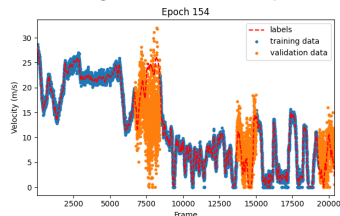
- Initial splitting: hard cut off after 80% of the frames
- Situational splitting: divide dataset into blocks of different driving scenarios, splitting with 80% test and 20% validation data on each

## Evaluation:

- variance  $\sqrt{\mathcal{L}} \gtrsim 16$ : no fitting
- $10 \lesssim \sqrt{\mathcal{L}} \lesssim 16$ : average velocity fitted
- $5 \lesssim \sqrt{\mathcal{L}} \lesssim 10$ : qualitative detection
- $1 \lesssim \sqrt{\mathcal{L}} \lesssim 5$ : quantitative detection
- $\sqrt{\mathcal{L}} \lesssim 1$ : perfect detection



driving situations in v-t-plot



example performance on training set

training:  $\sqrt{\mathcal{L}} = 0.4$ , test:  $\sqrt{\mathcal{L}} = 6.3$

# Optical flow using “Farneback pyramid method” [Farneback2003]

- Image sequence  $(f_t)_t$  with  $f_t : \Omega \rightarrow \mathbb{R}^3$ , for all  $t$
- Goal: find Global (dense) flow field  $V : \Omega \rightarrow \mathbb{R}^2, \omega \mapsto (V_x(\omega), V_y(\omega))$ , which solves the optical flow equation

$$\partial_x f \cdot V_x + \partial_y f \cdot V_y + \partial_t f = 0$$

- Uses a downsampling pyramid, to solve the equation for different resolutions of the image
- Parameters for the Farneback method

pyramid levels := 3

pyramid scaling := 0.5

window size := 6

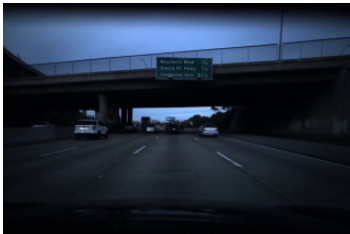
SD of the gaussian filter := 1.1

- Result: **Flow field with (160, 105, 2) pixels**

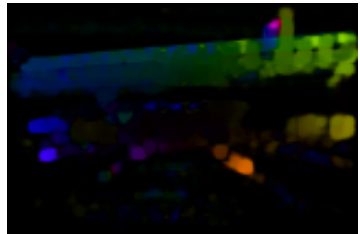


# Visualization of the flow field

- Flow field is a two-dimensional vector field
- RGB representation via
  - Transform flow field into polar coordinates  $(V_x, V_y) \mapsto (r, \varphi)$
  - Normalize magnitudes  $r$  for the third channel
  - Values of the second channel are all set to 255
  - Multiply angle  $\varphi$  by factor  $\frac{180}{2\pi}$  for the first channel
- Sample down the resolution again to speed up the training



Input frame



Corresponding flow field

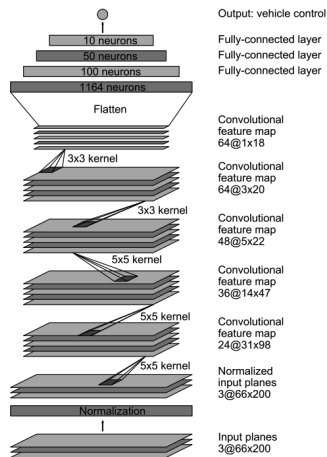
# Choosing an initial architecture

## Method selection

- Speed prediction is a non-linear regression task  $\Rightarrow$  neural network
- Task involves feature extraction  $\Rightarrow$  convolutional neural network (CNN)

## Initial architecture

- Using paper of *NVIDIA* work group [NVIDIA2016] of a CNN for self-driving cars adapted on our initial data
- Enough complexity and layers to handle the task and lots of possibilities to fine-tune it
- Initial results with the raw model:  $\mathcal{L} < 3$  on the training set and about  $\mathcal{L} \approx 19$  (initial splitting) on the test set



Original architecture of the *NVIDIA* paper [NVIDIA2016]

# Our approaches to optimize the results

## 1 Change components of the initial architecture

- Adding different pooling layers
- Use other activation functions

## 2 Change the architecture

- Expand structure to Siamese network
- Use different setups

## 3 Change the input data

- Acquire more data
- Use brightness augmentation

# Batch Normalization, Dropout layers, activation function and pooling

- Batch normalization to speed up the training [**BatchNorm2015**]
- Initial activation function:  $\text{ReLU} : \mathbb{R} \rightarrow \mathbb{R}_0^+, x \mapsto \max\{0, x\}$ , still MSE of over 15 on the testing set, less than 2 on the training set  
⇒ Overfitting problems
- Dropout layers [**Dropout2014**] to make the model more robust and reduce overfitting
- Solve problems of dead neurons using

$$\text{leakyReLU} : \mathbb{R} \rightarrow \mathbb{R}, x \mapsto \begin{cases} x, & x \geq 0 \\ c \cdot x, & x < 0 \end{cases}$$

with  $c = 0.01$ , MSE of around 11 on the testing set and less than 3 on the training set

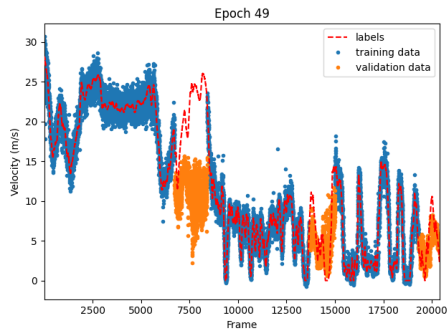
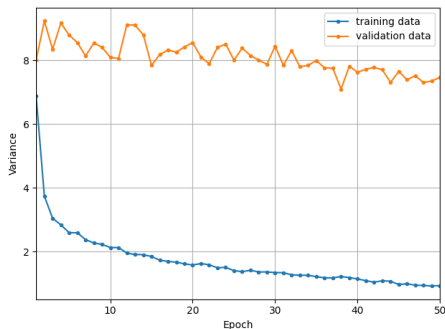
# Pooling layers (initial splitting)

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)	-	-	<b>3.22</b>	<b>9.63</b>
Average pooling	7.70	11.40	6.08	13.09

**Table:** MSE  $\mathcal{L}$  results of the network using different pooling strategies, one dropout layer, two different activation functions and the initial splitting. We trained each of the models for eight epochs.

# Training of the optimized model

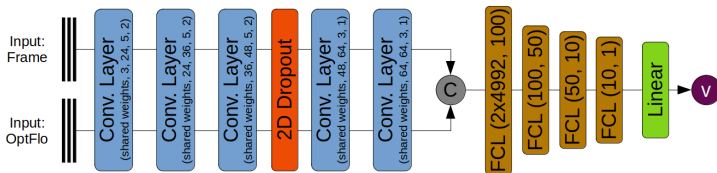
## Performance (Max Pooling):



- model converges on training data  $\sqrt{\mathcal{L}} < 1$
- test data only with qualitative fitting  $\sqrt{\mathcal{L}} = 7.4$

# Siamese Architecture: Setup

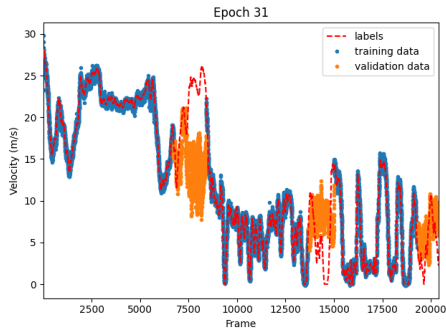
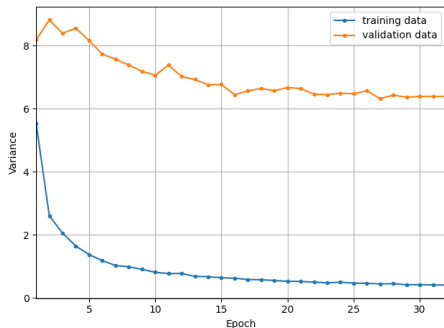
- Based on the initial architecture
- Using the same convolutional layers on raw frame and optical flow
- Put concatenated results into fully connected layers



- Use model also for two consecutive frames  $f_t$  and  $f_{t+1}$   
⇒ advantage: no previous calculation of optical flow needed

# Siamese Architecture: Performance

## Performance (Siamese network frame with optical flow):

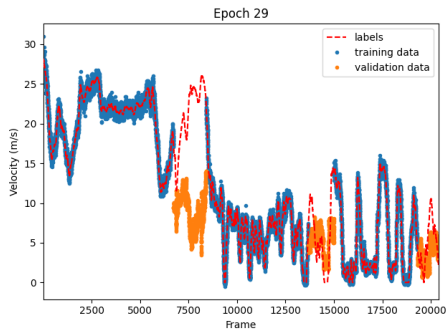
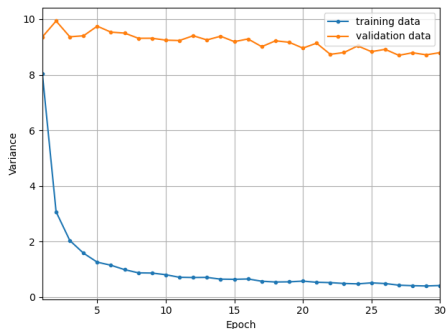


- model converges on training data  $\sqrt{\mathcal{L}} < 1$
- test data only with qualitative fitting  $\sqrt{\mathcal{L}} = 6.3$



# Siamese Architecture: Performance

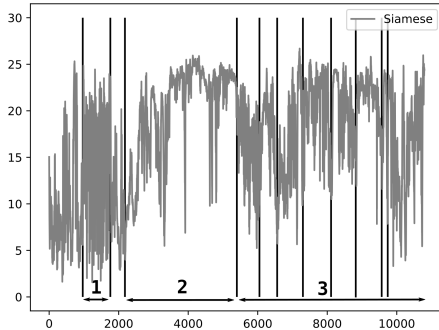
## Performance (Siamese network with two frames):



- model converges on training data  $\sqrt{\mathcal{L}} < 1$
- test data only with qualitative fitting  $\sqrt{\mathcal{L}} = 8.9$

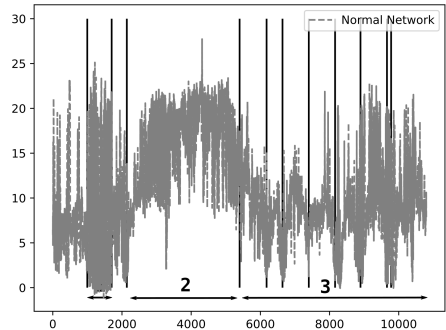
# Evaluation on the test video

## Siamese network:



- 1: longer halt at crossroads
- 2: highway driving
- 3: city driving with several stops

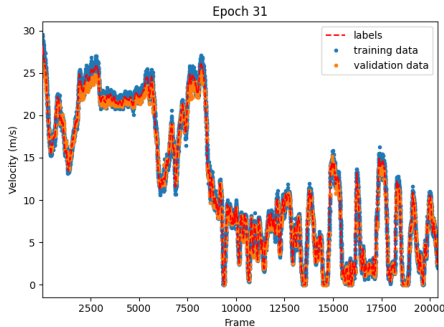
## Original (optimized) network:



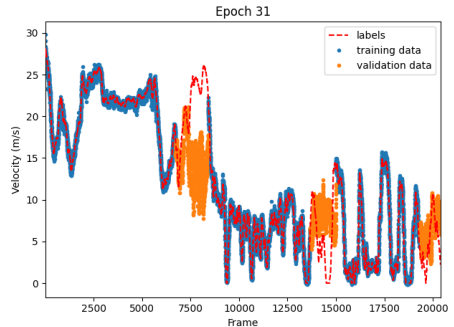
# Problems

We identified two problems

- 1 Overfitting, when validation data is not randomly distributed in the data but a of continuous blocks
- 2 Predictions are very susceptible for brightnesses/illumination changes in the frames, because of unstable calculations of the optical flow



random shuffled validation data



block validation data

# Problems and possible solutions

## Problems

- Predictions are limited to data very similar to the training data
- Predictions do not generalize well

## Most likely explanations:

- Very limited dataset for a complex model
- Network is not prepared for unseen situations
- Changes in brightness and illumination

## Solution:

- Acquire more data for various driving situations and/or use a less complex model
- Add noise to the data, more robustness in the optical flow calculation

# Acquire more data and evaluation of the test video

## Method

- Video producing and velocity detection with common apps
  - *open street maps*: GPS tracking (.gpx-files)
  - *open camera*: dashcam footage

## Advantages and Issues

- Easy method to create a lot of data (if a car is available)
- Velocity has some uncertainties (needs to be extrapolated to cover all frames; uncertainties of GPS vs. car sensors)
- Frame rate differs a little from original data (24 fps vs 20 fps)

# Contrast and brightness augmentation

- Additional noise to frames **before** calculating the flow field.
- Change the brightness and contrast of an image via

$$\text{frame}_{\text{augmented}}(i, j) = \alpha(i, j) \cdot \text{frame}(i, j) + \beta(i, j)$$

with functions  $\alpha$  (contrast:  $> 1$  increase,  $< 1$  decrease) and  $\beta$  (brightness).  
To get some noise into the frames, we used

$$\alpha \sim \mathcal{U}(0, 1) + 0.35$$

$$\beta \sim \mathcal{U}(-5, 35),$$

where  $\mathcal{U}(a, b)$  is the uniform distribution in an interval  $[a, b]$  for  $a < b$ .

# Summary and outlook

- So far good results and quite promising models  
⇒ Short demo video
- Still a lot work to do, to get a truly convincing model
- Currently working on own data gathering and brightness/contrast augmentation to improve and generalize models  
⇒ Idea: pick the so far best (two) model(s) and train them on the bigger and augmented dataset

# References



**Thank you for your attention.**