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

January 30, 2021

Table of content

The “comma ai speed challenge”[1]

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 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

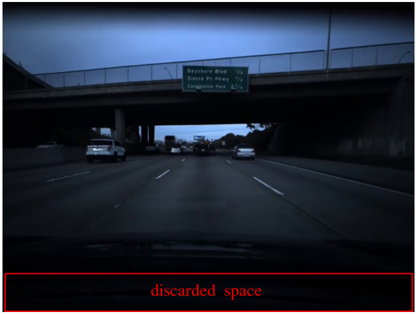
$$\mathcal{L} = \sum_i (p(x_i) - y_i)^2$$

[1]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

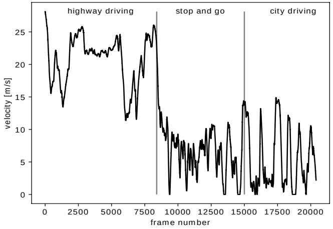
Analysis of the dataset

Situation data:

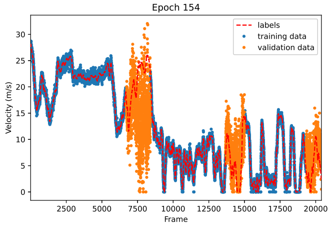
- test set with three different driving scenarios
- splitting with respect to them
  - divide dataset into different situations
  - splitting with 80% test and 20% validation data on each

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**]

- Global method to solve the optical flow equation

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

  for an image sequence $(f_t)_t$ with $f_t : \Omega \to \mathbb{R}^3$, for all t, and the (dense) flow field $V : \Omega \to \mathbb{R}^2, \omega \mapsto (V_x(\omega), V_y(\omega))$.
- Uses a downsampling pyramid, to solve the equation for different resolutions of the image
- Parameters for the Farneback method

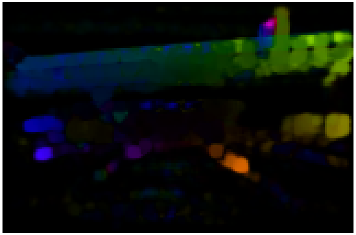
$$\begin{aligned} \text{pyramid levels} &:= 3 \\ \text{pyramid scaling} &:= 0.5 \\ \text{window size} &:= 6 \\ \text{pixel neighborhood size} &:= 5 \\ \text{SD of the gaussian filter} &:= 1.1 \end{aligned}$$

- Result: **Flow field with** $(160, 105, 3)$ **pixels**

Visualization of the flow field

- Flow field is a two-dimensional vector field
- RGB representation via
  - Transform flow field into polar coordinates $(V_x, V_y) \overset{\cong}{\mapsto} (r, \varphi)$
  - Normalize magnitudes r for the third channel
  - Values of the second channel are all set to 255
  - Multiply angle φ 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

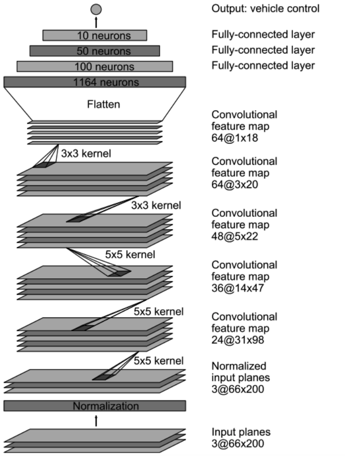
Convolutional neural network and initial architecture

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 $\mathcal{L} = 18 - 20$ on the test set
  $\Rightarrow$ Improvements needed



Original architecture of the *NVIDIA* paper [**NVIDIA2016**]

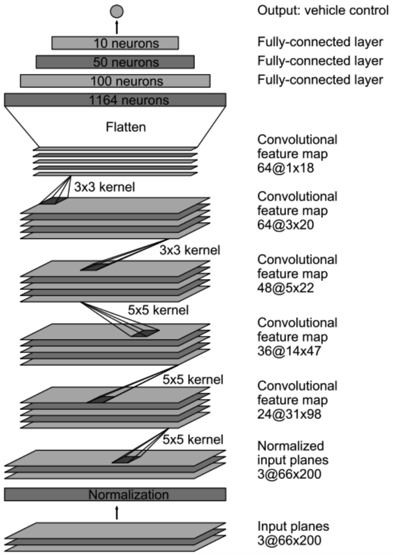


Figure: Original architecture of the NVIDIA paper [**NVIDIA2016**].

Batch Normalization, Dropout layers, activation function and pooling

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

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

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

Problems

We identified three possible problems for our poor results

1. Too complex model, as initially used for autonomous driving or insufficient amount of information put into the model
2. Brightnesses/illumination changes in the frames, therefore unstable calculations of the optical flow
3. Too ambiguous splitting, as the training and testing datasets represent totally different road traffic scenarios

Possible solutions

1. **Simplify model**: pooling layers (maximum and average pooling) to get more compression
   Siamese approach: put flow field and raw frame into the model or put two consecutive frames into the model
2. **Add additional noise**: add noise before computing the optical flow filed, to get more invariance regarding illumination changes
3. **Different splitting**: get better ratio between different scenarios, by using a splitting based on the different road traffic situations in the video

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)	-	-	**3.22**	**9.63**
Average pooling	7.70	11.40	6.08	13.09

Table: MSE 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.

Siamese approach (new splitting)

RESULTS ARE NEEDED :)
Demo videos of highway and city driving scenes under ../demos/

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 α (contrast: > 1 increase, < 1 decrease) and β (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 $a < b$.

Siamese approach for two consecutive frames

HERE SOME IDEAS AND/OR RESULTS ARE NEEDED

References