Driver Identification and Personal Attribute Inference from In-Vehicle Network Log

TDK Thesis

written by: Mina Remeli
consulent: Dr Gergely Ács

2018

Contents

Kivonat

Napjainkban már nem csak a telefonjaink, hanem az autóink is hatalmas mennyiségű adatot gyűjtenek rólunk. Ezt különféle szenzorok teszik lehetővé, amik többek között sebességet, kormányszöget, gáz-, fék- és kuplung-pozíciót is mérhetnek. Ezeket az adatokat az autók kommunikációs hálózatából (CAN buszból) az ODBII interfészen keresztül lehet elérni. Autók hálózati logjaiként is szoktak rá hivatkozni a szakirodalomban. Várható hogy az autonóm járművekkel a láthatáron az ilyen adatok gyűjtése óriási hangsúlyt fog kapni a közeljövőben.

Egy felettébb érdekes, de még kiaknázatlan használati területe a CAN hálózati logoknak a vezető azonosítás. Csupán néhány publikáció jelent meg az elmúlt pár évben ami kifejezetten vezető azonosítással foglalkozik CAN hálózati logokból. Egyik oldalról az ilyen identitást felderítő adatokban a privát adatok veszélyezettségét látták [1]. Másik oldalról meg a vezető azonosítás potenciálját látták abban hogy növeljék a vezetés élvezhetőségét [2]. Csakhogy az előbb említett kutatások erősen korlátozottak: először is csak kevés vezetőt tudnak megkülönböztetni, amiknek a száma 2 és 5 között terjed. Másodszor, az összes vezető ugyanazon az útvonalon megy végig. Ezek a feltételek erősen korlátozzák ezen modellek alkalmazhatóságát.

Amellett hogy vezető azonosítást végzek, arra törekszem hogy személyes adatok predikciójára is kiterjesszem a kutatásomat. Itt személyes adatok alatt nemet, kort, vezetői tapasztalatot, stb. értek. Szeretném kiemelni, hogy a személyes adatok predikciójára CAN hálózati logokból még nem volt példa eddigi kutatásokban. A motiváció emögött az ilyen jellegű adatok kamatoztathatósága – McKinzey&Co, egy menedzsment-tanácsadó cég becslése szerint az autókból nyert adatok értéke 2030-ig akár 750 milliárd dollárt fog érni[1]. A cégek már tudatosan készülnek az autókból nyert adatok piacára, ami majd várhatóan meghaladja még az autók piacát is[2]!

A dolgozatomban mutatni fogok egy neurális hálókra épülő gépi tanulás modellt a vezetők azonosítására. Az ehhez felhasznált, CAN buszon lehallgatott adatokat különböző szenzoroktól kapjuk, amik figyelik a fék, gáz, illetve kuplung pozícióját, az autó sebességet, stb. Azért esett a választásom neurális hálókra mert általánosságban jól teljesítenek különféle idősorok klasszifikálásában, más classifierekkel ellentétben [3]. Több mint 30, különböző útvonalon haladó vezető CAN hálózati logjai állnak e célból rendelkezésre. Az ezekre épülő modell pontosságát empirikus módon kiértékelem. Legvégül a kutatásom potenciális alkalmazási területeit mutatom be, illetve felvetek néhány jövőbeli kutatási irányt.

[1]https://www.wsj.com/articles/what-your-car-knows-about-you-1534564861
[2]https://www.forbes.com/sites/petercohan/2017/09/29/this-startup-is-helping-daimler-and-bmw-compete-with-google-for-10-trillion-market/#250ab5121342

Abstract

Nowadays it is not just our phones, but also our cars that collect a tremendous amount of data about us. This is done through recordings of different in-vehicle sensors measuring the speed, gas, brake and clutch pedal position among others. These data can be captured through the ODBII interface of the vehicle's CAN Bus, hence also referred to as CAN Network Log in the literature. With the upcoming autonomous cars, such data collection is likely to be overwhelming in the near future even more.

One interesting, but still quite unexplored usage of CAN Network Log is identifying drivers. There are only a few prior works which have addressed driver identification using CAN Network Logs exclusively. On one hand, it has been well-recognized that such data can expose a driver's identity [1] thereby raising serious privacy concerns. On the other hand, driver identification provides a novel way of enhancing driver experience [2]. However, the aforementioned works have strong limitations: first, they distinguish only a few drivers, whose number ranges between 2 and 5. Second, all drivers are supposed to drive on the same route. These assumptions significantly restrict the applicability of the proposed models.

In addition to driver identification, I aim to extend my research onto the inference of different personal attributes, such as gender, age, or driver experience. The inference of drivers' personal attributes from CAN Network Log has not been explored by any prior works in the literature. The motivation behind this is the lucrativity of such information – McKinzey&Co, a management consulting firm, estimates that "data from connected cars will be worth up to $750 billion by 2030"[1]. Companies are already preparing for the market for data from cars, which is expected to be even bigger than the market for the cars themselves[2]!

In my work, I present a neural network based machine learning model for identifying drivers based on the measurements of various in-vehicle sensors installed at the odometer, the brake- and gaspedal, and the clutch. These data are captured on the vehicle's CAN bus. The reason behind using neural networks is their superior overall classification accuracy of various time-series data over other classifiers [3]. I empirically evaluate the accuracy of the proposed model on the CAN Network Logs of more than 30 drivers, where each driver follows different routes within the same city. Finally, I conclude my work with describing potential applications of my proposal as well as some future research directions.

1 Introduction

1.1 Motivation

There are a number of systems built in our cars for various purposes - to regulate emission, to optimize engine performance or to log and detect malfunctions. The data required for these comes from in-vehicle sensors. This data is converted and processed by ECUs (Electronic Control Unit), which are essentially small computers inside our cars. The ECUs communicate with each other through the so-called CAN-bus.

The logging of such time-series produced by in-vehicle sensors is very common in our cars nowadays. For example, so-called "insurance dongles" can collect and send in-vehicle sensor data to insurance companies. Other (future) ideas include selling such data to mapping firms looking to provide more accurate traffic information, or selling consumption-related and other statistics to operators of larger vehicle fleets[1]. Be that as it may, the full amount of information that can be extracted from this kind of data is not yet fully understood by companies, nor by the subjects.

Driver identification is a very exciting way of using such data. Why not use this information to personalize your driving experience? Imagine getting into your car after your wife/husband/friend drove it, only to have your seat position, radio station and rear-view mirror adjusted back to your preferences just after getting out of your driveway?[2] Or otherwise, you can use driver identification to notify you any time someone else is driving your car.

Another interesting usage of CAN bus data (that has not been explored by anyone else so far to the my best of my knowledge) is the inference of gender, age, driver experience, etc. through such data. Car manufacturers are very much aware of cars entering the digital space and plan to make the most of such marketable data. With the possession of information related to age, gender etc., in-car advertisements (for example suggestions like which gas station to visit, or where to shop) can be much more personalized. It is no wonder therefore that people expect the market of such data to be even bigger than the already-existing market of cars[2].

However, there is a much more serious consequence to this that I want to strongly emphasize. My study demonstrates that companies collecting CAN logs of cars are indeed capable of identifying drivers and infer their various personal attributes. This has worrisome privacy implications especially if these companies wish to sell CAN logs to 3rd parties. Although drivers are expected to opt-in to such data sharing, it is still unclear whether they are completely informed about the wealth of personal information that their CAN logs potentially reveal to 3rd parties. Therefore, my study does not only raise the flag to car drivers, but also to companies collecting such data; the feasibility to identify (and/or profile) a driver means that CAN logs are indeed personal data and, as such, is subject to the European General Data Protection Regulation (GDPR) as of 25 May 2018. Therefore it is a fundamental duty of companies handling such data to adequately inform drivers and protect their personal data.

Failing to do so does not only ruin customer trust but can also result in a fine up to €20 million[3].

[3]https://www.gdpreu.org/compliance/fines-and-penalties/

1.2 Problem Definition

In this study I demonstrate that personal attributes (e.g., age, gender, experience) can in fact be extracted only from in-vehicle network logs. I have divided the problem into two stages - the first stage is to build a classifier model that identifies drivers. One of the reasons behind this is that there have been some (but altogether not too many) studies in this particular field that I could use as reference. The second reason is that it solves a very similar problem to the personal attribute inference (in fact, it can be viewed as the same problem if identity is regarded as a kind of personal attribute). Naturally, the second stage is to apply the model created in the first stage to personal attributes. So the hypothesis is that whatever model works for driver identification, should work for personal attributes as well.

In this paper I will present a CNN (Convolutional Neural Network) model. CNN is a perfect candidate for our purpose because it excels at capturing local reoccuring features such as accelerating, decelerating when braking, or changing gear. To build this model I had the following sensor-produced time series that measure clutch, gas and brake pedal position, rpm (revolutions per minute) and speed. It is quite common for most cars today to log measurements such as these.

Driver identification using CAN bus data is a subclass of the time series classification problems. Thus far driver identification was only experimented with using classical time series classification algorithms, where one had to invent and extract features manually. I chose to work with CNNs instead because there were not any similar efforts yet in this area. Using neural networks for time series classification has only recently become the focal point of studies. Another reason for choosing CNNs is that they can spare you the hard and cumbersome work of extracting features manually. This gives us a practical end-to-end model to work with.

Note that personal attribute prediction based on CAN bus data is so far unprecedented, and could potentially lead to lucrative use-cases in the car industry.

1.3 Challenges

In what follows, I outline a few obstacles which I faced in my research and make the above problems challenging to solve in practice.

- **Data augmentation** One of the challenging aspects of this work was that I had to deal with a relatively small amount of data. With SVMs or Random Forests this might not have been a promblem at all, whereas neural networks are known to work better on larger datasets.

  The usage of the *sliding window* technique has proven to be a successful method for increasing the amount of data [3] (and also increasing the model performance in general [4]). The *sliding window* technique was first used by Cui et al. on time-series to increase the amount of data in the model.

  This technique is successful at augmenting data because the data one obtains from it allows some samples to overlap. This means that one single datapoint can occur in multiple samples. This in turn creates an additional challenge when creating training/testing/validation sets that are not allowed to overlap at all.

- **Number of drivers** Driver identification thus far has mostly presented Random Forest based solutions, which worked only on a small number of drivers (2-5) from the whole set of drivers (1-vs-few) [2] [1], whereas my model classifies on all the 33 drivers present in the dataset (1-vs-all).

- **Different routes** Another source of additional complexity was that the drivers did not take the same route as opposed to others' research. However, the model still performed very well despite the circumstances.

- **Fitting into memory** After augmenting the data, I had some difficulties finding efficient ways of loading that much data into memory all at once. However since this is quite a common problem in the machine learning community, I managed to find solutions that would feed the data into the model while training/testing batch-by-batch.

- **Granularity and sampling frequency** Also the measured data was very fine-grained, which introduced a possibility of it being noisy. In addition to that every sensor had a different sampling frequency, which also posed a problem. All of these added to the problem's complexity.

1.4 Main Results

My main results are as follows:

- I achieve a **66%** mean accuracy in distinguishing one driver from the rest (compared to the 50% baseline). I achieve a **15%** accuracy in distinguishing all drivers from one another (compared to the 3.03% baseline)

- I achieve a **71%** accuracy in inferring gender (compared to the 50% baseline), **56%** accuracy in age inference (compared to the 25% baseline) and a **54%** accuracy in experience inference (compared to the 33% baseline).

These are significant improvements over the state-of-the-art solutions which achieved comparable results only in more restricted scenarios (e.g., when each driver follows the same route, and the model is trained to distinguish only a pair of drivers). Another notable achievement is that these predictions were made using only 10 second long driving samples. Moreover, to the best of my knowledge, no one has studied the feasibility of personal attribute inference from CAN logs so far.

1.5 Organization

My paper is structured as follows. Section 2 introduces some related work. Section 3 contains background information on TSC (Time Series Classification), CNNs (Convolutional Neural Networks) and Can bus data. Then I shall elaborate on how the data was collected and preprocessed in Section 4. This is followed by the description of my proposed model in Section 5. In Section 6 I evaluate my results. I conclude this work with Section 7.

2 Related Work

Driver re-identification Driver re-identification is a relatively recent subject of research, and there are not many related works in the literature yet. There have been various approaches, ranging from identifying a driver from data that comes from driving simulations, through CAN bus data or even through introducing a 4-dimensional representation of one's driving skills.

Miyajima et al. [5] have done experiments on two different test settings. The first experiment setting focused on 12 subjects that have done routes in a driving simulation where they collected data on the driver's vehicle velocity, distance from the vehicle in front, and the velocity from the vehicle in front. In this setting they achieved a 81% identification rate. The second setting was done using an actual vehicle, and included 30 test subjects with data collected on driver's vehicle velocity, force on gaspedal and force on brake pedal. All subjects took the same route. Here they achieved a 73% identification rate. They compared two methods for driver identification (Gaussian Mixture Model and Helly model), and the conclusion in both test settings was that the GMM performed better.

The next article that I want to describe is focused on the possible privacy breach that the extraction of such information entails (Enev et al., [1]). There they experimented with several classifiers, such as SVMs, Random Forest, Naive Bayes and KNN on a dataset of in-vehicle sensor measurements produced by 15 drivers. All of the drivers followed the same route. Instead of doing a *1vsALL* classification (by *1vsALL* I mean a binary classification where one has to distinguish 1 subject from all the others) they chose to do a Q-Weighted classification which essentially trains a set of pairwise (binary) classifiers (one for each pair of subjects). The drawback of this method that while it achieves high accuracy, in the worst case scenario it still has to build and compare $n * (n-1)/2 \approx n^2$ binary classifiers (where n is the number of drivers)[6]. They also used the *sliding window* technique to increase the amount of their data (with overlap).

Another related work suggests that all the information needed to identify a driver can be obtained from the measurements of a single turn (Hallac et al.[2]). This confirms the intuition that someone's driving can be quite accurately distinguished based on how they slow down, switch gears and accelerate. In their work they presented two, three, four and five-driver classifications with varying models like logistic regression, SVM and Random Forest. All of the drivers in the dataset followed the same route, so they selected the 12 most frequent turns and built classifiers based only on the 8-10 seconds describing that particular turn.

However, both researches done by Enev et al. [1] and Hallac et al. [2] achieved their best results when using a Random Forest classifier. The drawback of the algorithms that were tested by them is the need for manual feature engineering, which requires a considerable amount of expertise in the field and a deep knowledge of the data.

Finally, Fugiglando et al. [7] proposed the concept of Driving DNA, which would serve as a representation of one's driving style. The proposed dimensions of the DNA consist of data partially collected from the CAN bus, namely frontal acceleration (for measuring braking), lateral acceleration (for measuring turning), rpm (for measuring fuel efficiency) and the combined information of speed, rain and road speed limit (for measuring speeding). The advantage of their method is the very low computational complexity.

All of the above mentioned works that have done driver re-identification have done experiments where subjects had to follow the same route. In contrast to that, I present comparable results without imposing such restrictions.

Time Series Classification (TSC) with a deep learning approach There has only recently been some research on time series classification with the help of neural networks. I shall like to mention a few. One of the very first approaches was the usage of Multi-scale Convolutional Neural Networks (MCNNs) on univariate timeseries by Cui et al. [3]. This approach was tested and evaluated on the UCR dataset alongside 14 'classical' TSC algorithms. However their architecture requires some additional preprocessing of the input, in contrast to the end-to-end model I propose.

Another approach is a neural network-based model called MC-DCNN (short for Multi-Channels Deep Convolutional Neural Networks) for multivariate time-series proposed by Zheng et al. [8]. Their proposed architecture is very similar to what I propose, when I use the inputs of multiple sensors (differing only in the number of convolutions and the hyperparameters). They propose to split up the multivariate time series into univariate ones, and then apply feature extraction on the univariate time series. Finally the features are fed into a Multi Layer Perceptron (MLP). Their model was evaluated on two datasets (not included in the UCR).

Wang et al. [9] tested three neural networks (ResNet[10], FCN and MLP) on the UCR datasets - their proposed models were evaluated against many TSC algorithms (including MCNN) and they found that even though the FCN is superior with the lowest test error rate - the T-test showed that there is not much of a difference between the top 5 performing models (COTE, MCNN, BOSS, FCN and ResNet).

Recently, there has been an overall evaluation of all the neural-network based TSC methods in the work of Fawaz et al. [11]. They compared and tested DNN classifiers not only on the UCR datasets (that mostly contain univariate time series) but also on 12 multivariate time series datasets. They found that the best performing models (in both in univariate and mutlivariate settings) were the ResNet (Residual Network) and FCN (Fully Convolutional Neural Network). Nevertheless, I chose not to use architectures such as ResNet or FCN, because they have an extensive amount of parameters, which works better on larger data.

3 Background

3.1 Time Series Classification (TSC)

Time series classification is a widely researched area in medicine, music and the industry in general. A time series classification problem can be defined as follows: given a set of classes Y ($y_i \in Y$), and training data T (where $T_i = \{t_1, t_2, ...t_l\}$ for time series of length l ($T_i \in T$). The goal is to find a classifying function f, where $f(T_i) = y_i$.

There are mainly two methods for time series classification - distance-based (mostly with k-NN classifiers) and feature based classifications. The distance based ones use similarity measures such as Euclidean distance or Dynamic Time Warping (DTW). One of the most popular and successful approaches coming from this class of classifiers was the 1-NN DTW [12] - it can really be considered as one of the "baseline" TSC methods.

A simple example of feature based TSC is to represent time-series with features like mean, variance, standard deviation, etc. Today we typically use more complex features as well, such as frequency-domain information provided by Discrete Wavelet Transform (DWT) or Discrete Fourier Transform (DFT). Another feature based TSC relies on extracting global/local patterns (words) and by representing the original time series with a histogram containing these "words" (bag-of-words model).

Another popular approach is to use ensembles of classifiers (like COTE [13]) that combine multiple classifiers to get more accurate results (which does bring good results but is high complexity and very resource-intensive).

These are only a few examples of the many time-series classification algorithms[4] that have surfaced throughout the last few years. In order to create a benchmark for evaluating all of these algorithms, they can be evaluated (and tested against other classifiers) on the UCR datasets [14] (47 datasets from the University of California, Riverside - their number has since been expanded to 85).

Many of these classifiers have been put to the test by Bagnall et al. [15] on the aforementioned 85 datasets. The conclusion of the paper was that while COTE outperformed even DTW (even though DTW is in general hard to beat) by 8% on average, it can be said that there is no "absolute winner" algorithm. A good solution to a specific problem is tailored to it. (For example even a high-scoring algorithm like COTE still leaves a lot to be desired in terms of computational complexity.)

3.2 Convolutional Neural Networks (CNNs)

History The concept of neural networks reaches back all the way to the 1950-60's. Back then researches were interested in making computational models of artificial neurons that try to mimic the behavior of the human brain. However, back then the computational capacity and other issues (like vanishing/exploding gradient) delayed the implementation of these concepts, and so the interest in further research died down for some time.

How CNNs work CNNs are deep neural network structures that have become increasingly popular over the past few years. They were introduced in the early 1990's by LeCun [16] who used this architecture to recognize handwritten digits. They had a sudden surge in popularity after the Krizhevsky et al. [17] architecture won the ImageNet contest in 2012 with an error rate of 16.4% compared to the second place result of 26.1%.[5] Since then CNNs have become the go-to architecture for image classification. Another field where CNNs are widely used is natural language processing or face detection.

[4]A more complete list of TSC algorithms: http://www.timeseriesclassification.com/algorithm.php

[5]http://www.image-net.org/challenges/LSVRC/2012/results.html

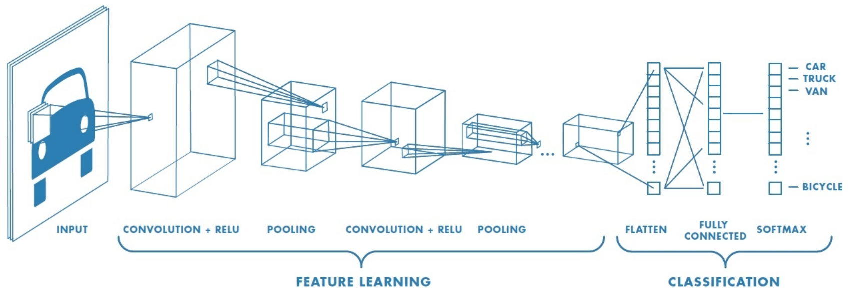


Figure 1. CNN architecture overview on an image classification example. Source: `https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html`

The CNN architecture consists of two main parts: the first part is responsible for extracting features, and the other one does the classification. The feature extracting part in essence produces a set of (learnable) filters that can recognize patterns in a translation-invariant way (so for example in case of images it can easily detect the curve of a smile in the upper left corner of the picture just as well as in the center or anywhere else).

An illustrative overview of the general architecture can be seen in Figure 1. The components of the feature learning part are the following: (1) *convolution* where the filters are created - here we can define how many filters we want, how big they should be, and what stride we should use when going over the image. This is usually followed by an activation function (for example ReLu) that introduces non-linearity into the model (and also makes the model more robust to the problem of vanishing/exploding gradient). The next part is (2) *pooling*, where we "compress" the input by averaging or taking the maximum value.

The reason why CNNs became so popular is mainly because of how scalable they are - because the filters share the same weights. This way, considerably less weights need to be updated on every training iteration. Another appealing aspect is that features are learned automatically.

Intuition behind using CNNs CNNs have proven over time to be a scalable and reliable architecture for image classification. The reason why they work with time-series as well, is because of the locality of the traits that potentially identify a driver (the short moments when we accelerate, decelerate, or change gear). CNNs excel just at that - finding traits that are unique to each class that has to be identified. My multi-channel CNN architecture is thus able to find a set of filters for each sensor that are best at capturing those local traits.

3.3 CAN bus data

Logging and processing information from various in-vehicle sensors has been present for a long time now. They can be tracked back as early as the 1970's when the american Congress passed the Clean Air Act, requiring car manufacturers to implement emission control systems in their vehicles. Since then there has been a lot of progress, leading to modern diagnostic systems and dashboards displaying information such as speed, consumption, temperature and so on.

There are several subsystems installed in our cars responsible for making A-D conversions, preprocessing the sensor-collected data and possibly sending out signals to inform other subsystems. These are called ECUs (Electronic Control Unit). The way they communicate and transfer data to one another is with the help of communication standards. There is one standard that is predominantly used nowadays, and that is the CAN (Controller Area-Networking). All nodes (such as ECUs) on the CAN bus send out information several times a second, which consists of a header+data part, where the header is simply an ID referring to the kind of data that is being transferred.

To show an example - let's say that we capture some data on the CAN bus with an ID of 1. Suppose we know that the second byte of that data contains the speed of the vehicle. If we capture all the subsequent data packets with an ID 1 and extract the second byte from it, then we get a time-series that shows us how the speed of the vehicle changed in the time observed.

4 Data

4.1 Collected Data

Collection process The driving data is obtained through the OBD-II (On-Board Diagnostic Systems) port of an 2018 Opel Astra. Each of the drivers were asked to drive around in Budapest for 20-30 minutes. There was no route they had to strictly follow, in contrast to other studies [1][2], which makes the data more general and comparable to a real-life example. (However this also introduces additional complexity in the classification process.) There were only 3 requirements the data collection process had to meet: (1) a minimum of 20-30 minutes of driving data from each driver, (2) no data was to be recorded while driving uphill or downhill and (3) no data was to be recorded in very heavy traffic. Figure 2 shows how the raw data looks from the sensors.

The credit for collecting these data goes to CrySyS (Laboratory of Cryptography and System Security)[6] at the Budapest University of Technology and Economics (BME).

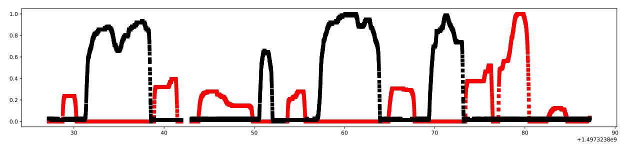
Personal attributes There are altogether 33 drivers' data in the dataset, 28 males and 5 females. Their ages range from 20-69 (11 people were between the age of 20-25, 8 between 25-30, 7 between 30-40, and 7 above 40). I also ranked their driving experience (Low, Average and High), based on how many kilometers they drive in a year on average (There were 12 with Low experience (<7000 km per year), 11 with Average experience (8-14000 km per year), and 10 with High experience (>14000 km per year)). This information can be found in Table 1 as well.

Sensors In my work I use the measurements of 5 sensors. In the sequel I will refer to the 5 sensors' data as *brake*, *clutch*, *gaspedal*, *rpm* and *speed*.

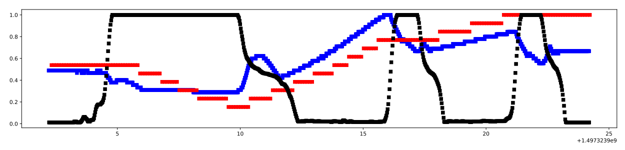
Brake, *clutch* and *gaspedal* measure the position of the respective pedals. As for the rest: *rpm* records the revolutions per minute and *speed* measures

[6]https://www.crysys.hu/

the vehicle's speed. The sensor's measured value ranges (measured_min and measured_max) are depicted in Table 2.



(a) Brake pedal position (black) vs. gaspedal position (red).



(b) Clutch pedal position (black) vs. rpm (blue) vs. speed (red).

Figure 2. Raw sensor data.

Attribute	*Value*	*Population*
Gender	Male	28
	Female	5
Age	[20-25]	11
	[25-30]	8
	[30-40]	7
	[40-70]	7
Experience	Low	12
	Average	11
	High	10

Table 1. Personal attributes.

	min	**max**	**mean**	**standard deviation**
speed	0	3513	891.77	727.21
gaspedal	0	254	18.55	31.24
clutch	0	255	97.73	106.07
brake	0	161	22.8	30.72
rpm	0	80	22.2	9.09

Table 2. Highest and lowest values measured on all sensors, their mean and standard deviation.

Data frequency and their alignment The obtained time series from each sensor were unfortunately unequal in length, because their sampling frequency was not the same. This of course means that the i'th sample from *sensor1* is not measured at the same time as the i'th sample from *sensor2*. This poses a problem for us, since we need aligned data in order to make a decision. To achieve the same length over all of the time series I have down-sampled the data collected by sensors that work with a higher sampling frequency. The sensor with the shortest sampling frequency was *speed*, namely it was 20Hz. For example suppose that another sensor had a sampling frequency that was 5 times bigger. I took that sensor's recordings, split it up into groups of 5, and then took the average of those groups to get the same amount of measurements as the *speed* sensor. Hence, each transformed time-series has identical length, and I work these transformed data henceforth.

Anonymization and consent The drivers' identities were not disclosed, instead of names they were labeled with unique pseudo-identifiers. The subjects of this study have all given their consent to us to log and work with their driving data. They were also informed on how their data was going to be used in this study.

However, because my research suggests that CAN data does potentially contain personal information about a driver, I cannot disclose their data. That would require their further consent according to the GDPR.

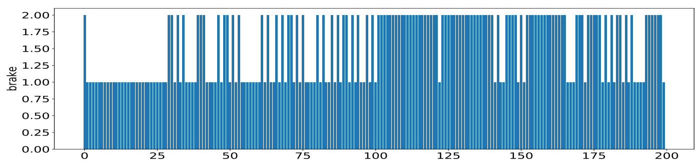
4.2 Preprocessing

Creating samples First, I drop the 0 values when loading the data, because those values have no information for our purpose[7] (theoretically the CNN is able to learn if a value holds no information, but this seemed as a reasonable choice since it speeds up the process of learning). As mentioned above, I align each and every sensor's data to one another in the described fashion. Then, I had to slice up the approximately 20 minute long time series into samples that the classifier could work with. The sample length I used was about 10 seconds long ($10 * T_{sample} = 10 * (1/f_{sample}) = 200$ elements in a sample). Intuitively one might think that the distinguishing part in everyone's driving is the part where they accelerate, which is about 2-3 seconds long. However, when driving in a city one must realize that there are a lot of situations where one must stand still (in morning traffic, in front of a stoplight, or while letting an old lady cross the street). Hence, I let the sample size be a bit longer, and also because other research has achieved considerable results on similar sample sizes [2].

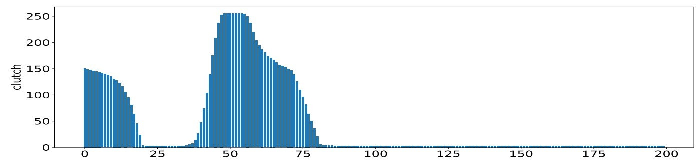
I created the samples using the *sliding window* technique. It takes a $T = \{t_1, t_2, ..., t_l\}$ time series, and creates many smaller time series ($S_i, S_i \subset T$) of length w (window size): $S_i = \{t_{s*k}, ..., t_{s*k+w}\}, k = 0, ..., l - w + 1$ where s is the stride. This also increases the amount of data since we have set s to be smaller than w (this creates overlap between the S_i samples). My window size was $w = 200$ while the stride was $s = 3$. A window-sized slice equals one sample in our setting.

It must be mentioned that the *sliding window* technique introduces redundancy in the dataset because of the overlaps between two windows (if the stride is smaller than the window size). One can argue that this might introduce overfitting. To battle this, I have used state-of-the art techniques to help our model generalize better, which I will elaborate on in Section 5.1.

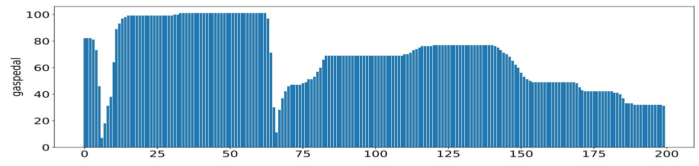
[7] I have experimented with datasets containing 0's as well. It did not affect the model accuracy in any significant way.



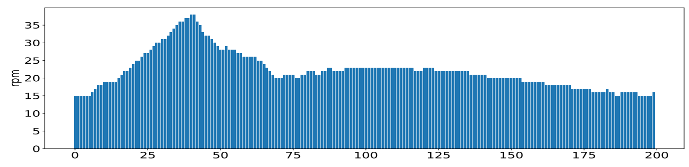
(a) brake



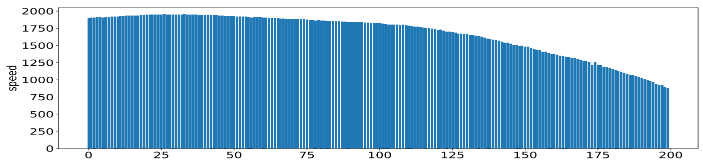
(b) clutch



(c) gaspedal



(d) rpm



(e) speed

Figure 3. A sample from each sensor.

Train/test split To create the train/test split first I have split the dataset into two parts: dataset1 and dataset2. While doing this, I paid attention to split the dataset in a way so that they come from the same distribution. (For further explanation on how, see next subparagraph.)

The training samples are randomly sampled from dataset1, while the testing samples are randomly sampled from dataset2. This ensures that the distribution of the training and the testing datasets is the same *on average*. I also made sure that the training and testing datasets are *balanced*. A balanced dataset contains the same amount of positive samples as negative samples in binary classification

- in multiclass classification it means that there is the same number of targets from each class.

Creating dataset1 and dataset2 In order to have two datasets that come from the same distribution as the original one, I have done the following: let us call the recordings of a sensor for one driver at one occasion a *driving session*. After creating the 200 long samples from a particular *driving session* (that corresponds to 10 seconds worth of driving data), I took the created samples, and split them into two seperate datasets (while dropping some samples from the center to ensure that there is absolutely no overlap between the 2 created datasets). The result is two datasets that contain the same distribution of drivers as the original one. More precisely: the probability of observing a driver a in the training set must be the same as the probability of observing a driver a in the testing set (this should hold true for each driver a).

All machine learning problems are approached with the hypothesis that the training data comes from the same distribution as the testing data. However, this is not always true, because in practice we have no guarantee that the data in production will come from the same distribution as the data we trained on. And since a model can only learn from the training dataset, it will likely fail no matter what on data that comes from a different distribution. This shifting phenomenon is called *covariate shift* (when input distributions are shifted) or *dataset shift* (when the joint distribution of input and output is shifted). To battle this, one can either try to get data from the same distribution as the testing dataset (if possible). Otherwise, one can train a number of models with varying distributions and create an ensemble from them (multi-dataset models).

Figure 3 shows how a sample from each sensor might look like.

5 Model

5.1 Model structure

Overview The model has a fairly simple architecture when it has one input, as can be seen in Figure 4. The presented model is a classical CNN and consists mainly of two parts: a feature extracting part and a MLP (Multilayer Perceptron) for the actual classification based on these features. This kind of architecture is prevalently used for image classification, face and edge detection[18].

For my work I have used the Keras framework (v. 2.2.4) for python with tensorflow-gpu (v. 1.11) in the backend. The models were trained and tested on a machine with an Intel i7-7500 CPU, 8 GB RAM, and an NVIDIA Geforce 940Mx (2 GB). My work can be accessed on a public github repository: `https://github.com/minaremeli/tdk_2018`.

Hyperparameter and model structure heuristics The hyperparameters and model structure were selected by rules of thumb that are commonly used for building CNNs. Some of the general rules I tried to abide by was to first experiment with simpler models, gradually increasing its complexity if needed. Another very common rule is to first make the model overfit, and then use regularizing strategies like L_2 or dropout. Another CNN-specific rule says to use one or two, but at most 3 convolutions in the feature extracting part. A common heuristic for selecting the number of nodes in the dense layers is for the number to be a power of 2. There are no "common" heuristics yet for selecting hyperparameters in the convolutional layers in a time-series application, because this field is fairly young still.

Model architecture The general architecture of the model depends on how many channels of data we want to feed into it. A *"channel"* refers to a stream of data that comes from one of the introduced sensors. For a general overview of the model architecture please see Figure 4 with 1 input channel and Figure 5 with 4 input channels. The main difference between the two settings is that in case of a multi-channel model each channel has its own feature extracting part (its own local convolutions) - that way when the channels are concatenated later on, one can extract only the most meaningful information from each sensor using dense layers.

The first layer is a normalization layer. It shifts the input values to a 0-mean 1-variance. The feature extracting part consists of 3 consecutive convolution-activation-pooling layers. For the convolution layers I used a filter size of 8, 16 and 32 respectively, a kernel size of 5 and a local stride of 1 over all three of them. The activation function I used after the convolutions is the ReLU[19] (Rectified Linear Unit). The pooling layers had a pooling size of 2 (they halved the input size by going over it with a 2x1 window that selected the largest of the underlying values).

In the end I have three dense layers - a dense layer with 128 nodes followed by a dropout with a rate of 0.2, and finished off with two more dense layers - one with 64 nodes and the last one with either one node (if doing binary classification, like *1vsALL*) or 33 nodes (for multi-class classification).

Input As described in Section 4.2, our inputs are window-sized samples. I will show two settings I used: one is where I used only one sensor's data for classification, and another where I used multiple. When working with one sensor I had an input of size (200x1), whereas when I had multiple sensors I had an input of (200x1) N times, where N is the number of sensors (and thus channels).

Regularization techniques Pooling layers were used between the convolutional layers to increase generalization. Another method I used for this is dropout layers on one of the hidden layers. The dropout technique[8] essentially deactivates a set of random neurons in the hidden layer it is applied to. This is done again and again for each mini-batch, so in the end one gets a fairly accurate and generalized model by "combining" different models that have a different set of weights and neurons. This is the basic intuition behind dropout.

[8] https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

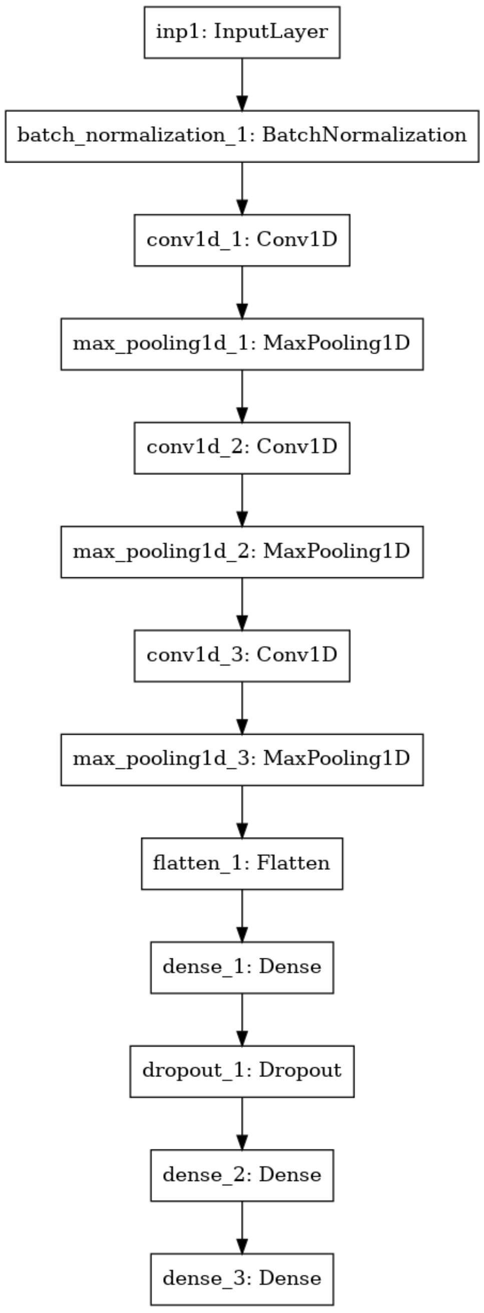


Figure 4. Model architecture with 1 channel.

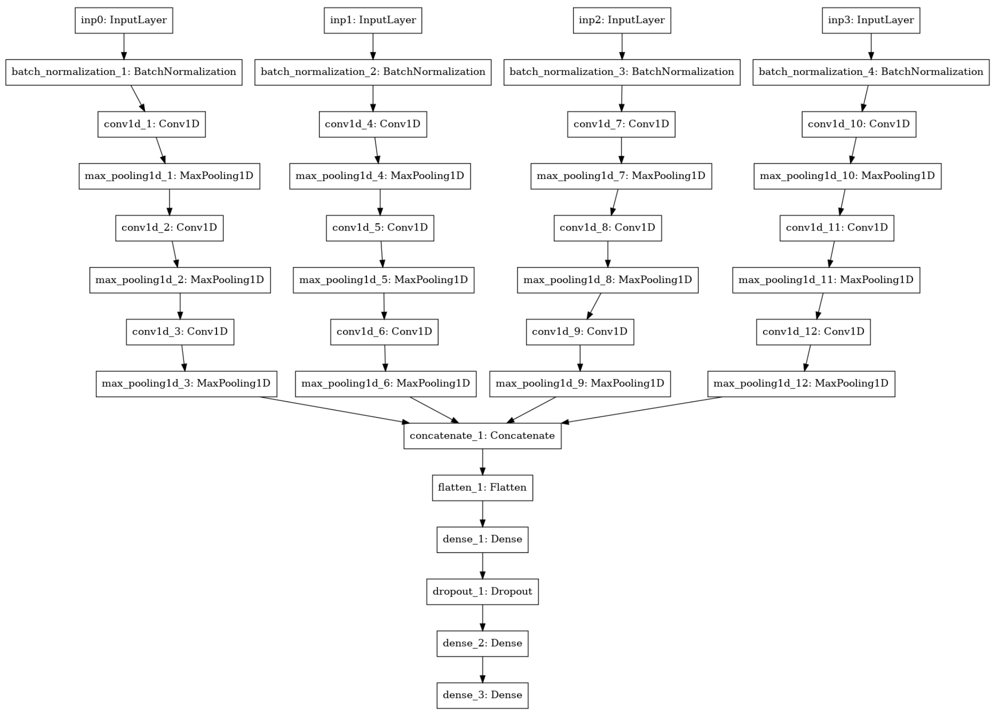

inp0: InputLayer
inp1: InputLayer
inp2: InputLayer
inp3: InputLayer
batch_normalization_1: BatchNormalization
batch_normalization_2: BatchNormalization
batch_normalization_3: BatchNormalization
batch_normalization_4: BatchNormalization
conv1d_1: Conv1D
conv1d_4: Conv1D
conv1d_7: Conv1D
conv1d_10: Conv1D
max_pooling1d_1: MaxPooling1D
max_pooling1d_4: MaxPooling1D
max_pooling1d_7: MaxPooling1D
max_pooling1d_10: MaxPooling1D
conv1d_2: Conv1D
conv1d_5: Conv1D
conv1d_8: Conv1D
conv1d_11: Conv1D
max_pooling1d_2: MaxPooling1D
max_pooling1d_5: MaxPooling1D
max_pooling1d_8: MaxPooling1D
max_pooling1d_11: MaxPooling1D
conv1d_3: Conv1D
conv1d_6: Conv1D
conv1d_9: Conv1D
conv1d_12: Conv1D
max_pooling1d_3: MaxPooling1D
max_pooling1d_6: MaxPooling1D
max_pooling1d_9: MaxPooling1D
max_pooling1d_12: MaxPooling1D
concatenate_1: Concatenate
flatten_1: Flatten
dense_1: Dense
dropout_1: Dropout
dense_2: Dense
dense_3: Dense

Figure 5. Model architecture with 4 channels.

5.2 Classifiers

I describe next the different classifiers that I use in my study.

5.2.1 Driver Identification

- **1vsALL** In this experimental setting I create binary classifiers that could distinguish one driver (the subject) from all the others. The experiment is repeated for each driver in the dataset (everyone takes a turn at being the subject).

  There are two stages to this experiment. The first stage of the experiment trains models with the input of one sensor (this is done for each of the 5 sensors: *brake*, *clutch*, *gaspedal*, *rpm*, *speed*).

  The second stage of the experiment trains the models with the sensors that provided the best results in my previous experiment. Thus with these best performing sensors I show a multi-channel CNN classifier. Hence, I have 5 classifiers per driver, and 33*5 classifiers altogether in this experiment. Because the experiment took such a long time, I decided to train the models on 2 epochs only. All the models had about 1700 samples to train on.

- **ALLvsALL** In the ALLvsALL setting I show multiclass classifications, where the model has to distinguish all of the 33 drivers from one another. This model was trained for 2 epochs, on 27000 samples of training data, on the measurements of *clutch*, *gaspedal*, *rpm* and *speed* seperately. The reason why I trained on 2 epochs here as well is very similar as last time (the duration of building 33 classifiers on so many training samples takes quite a long time). In one experiment I also combine these four sensors as input. I omitted the measurements of *brake*, because in the previous experiment the models that used its data did not perform very well (its accuracy was near random guess).

5.2.2 Personal Attribute Inference

In this section I build 3 different classifiers for the inference of age, gender, and driving experience, respectively. The first classifier is a binary classifier for guessing gender, the second and third one are multi-class classifiers for guessing the subject's age (4 classes: [20-25], [25-30], [30-40], [40-70]) and experience (3 classes: Low, Average and High). Here I will experiment with both the multi-channel model, and the 1-channel models respectively. For further information on the personal attributes, see Table 1.

For gender inference I had about 24000 samples for training data, for age inference 65000 and for experience inference 80000. The models were trained for 4 epochs.

6 Evaluation

6.1 Evaluation Metrics

To evaluate my model I have done an extensive amount of experiments, building more than 200 classifiers in total. The main metric I used across all of my experiments is the binary accuracy in case of binary classification, and categorical accuracy. Binary and categorical accuracy both calculate the true positive rate ($TPR = \frac{TP}{(TP+FP)}$) of the classifications.

Another metric I used (on the personal attribute inference) is the ROC (Receiver Operator Characteristic) curve. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR), and is often used for selecting an optimal model in machine learning.

6.2 1vsALL

Results on the first stage of this experiment can be seen in Table 3. On average the best performing model was the one that got its data from the *gaspedal* sensor, performing on average with 66%. The model built on the measurements of *clutch* yielded the overall highest classification accuracy (92%). However the models built on the data of *clutch* are not consistent in accuracy, having the largest variance. One can also see that the models built on information from *brake* make only near-random guesses. For this reason, I henceforth omit this sensor from the experiments that use the inputs of multiple sensors at once. Results on the first stage of this experiment can be observed in Table 3.

In the second stage of my experiment I combine the 4 best performing sensors in a single multi-input model, namely of *clutch*, *gaspedal*, *rpm* and *speed*. The results can be seen in Table 4. This multi-input model achieves a 64% mean accuracy, which comes in as the second best performing model when distinguishing one driver from the rest. The variance is second best as well.

Driver ID	brake	clutch	gaspedal	rpm	speed
ID1	0.507212	0.519231	0.567308	0.485577	0.528846
ID12	0.444712	0.567308	0.531250	0.384615	0.540865
ID31	0.629808	0.778846	**0.812500**	0.668269	0.567308
ID24	0.572115	0.788462	0.600962	0.449519	0.540865
ID7	0.492788	0.627404	0.778846	0.456731	0.709135
ID0	0.512019	0.572115	0.634615	0.567308	0.528846
ID5	0.557692	0.526442	0.766827	0.447115	0.540865
ID13	0.572115	0.603365	0.545673	0.586538	0.461538
ID2	**0.687500**	0.865385	0.540865	0.603365	0.598558
ID26	0.487981	0.538462	0.802885	0.653846	0.514423
ID4	0.473558	0.798077	0.632212	0.502404	0.454327
ID20	0.663462	0.639423	0.692308	0.620192	0.447115
ID14	0.562500	0.670673	0.725962	0.560096	0.622596
ID22	0.560096	0.680288	0.697115	0.435096	0.610577
ID17	0.487981	**0.918269**	0.694712	0.540865	0.552885
ID3	0.427885	0.555288	0.649038	0.545673	0.569712
ID18	0.562500	0.697115	0.612981	0.406250	0.399038
ID28	0.557692	0.745192	0.670673	0.658654	0.591346
ID15	0.569712	0.560096	0.668269	**0.713942**	**0.781250**
ID30	0.514423	0.468750	0.783654	0.658654	0.492788
ID10	0.521635	0.528846	0.663462	0.514423	0.677885
ID9	0.519231	0.680288	0.581731	0.512019	0.543269
ID11	0.509615	0.569712	0.718750	0.670673	0.661058
ID16	0.454327	0.567308	0.545673	0.536058	0.468750
ID8	0.562500	0.658654	0.600962	0.629808	0.675481
ID19	0.430288	0.478365	0.750000	0.600962	0.519231
ID29	0.634615	0.718750	0.742788	0.677885	0.584135
ID23	0.588942	0.653846	0.730769	0.533654	0.598558
ID25	0.389423	0.774038	0.663462	0.605769	0.665865
ID27	0.591346	0.487981	0.632212	0.562500	0.600962
ID32	0.584135	0.540865	0.625000	0.685096	0.653846
ID21	0.475962	0.548077	0.670673	0.456731	0.569712
ID6	0.516827	0.536058	0.504808	0.480769	0.552885
Mean	0.534018	0.632212	**0.661786**	0.557911	0.57044
Variance	0.004685	0.013398	0.007017	0.008059	0.006797

Table 3. The mean accuracy rate across all predictions for each 1vsALL classifier that trained on one sensor. The highest accuracies are highlighted in this table.

Driver ID	**4 sensors**
ID17	**0.896635**
ID29	0.810096
ID4	0.495192
ID24	0.509615
ID30	0.548077
ID22	0.644231
ID0	0.536058
ID14	0.790865
ID21	0.564904
ID31	0.858173
ID12	0.507212
ID2	0.769231
ID7	0.721154
ID13	0.584135
ID20	0.689904
ID11	0.651442
ID27	0.524038
ID32	0.576923
ID23	0.644231
ID18	0.447115
ID25	0.814904
ID16	0.528846
ID19	0.699519
ID3	0.687500
ID10	0.682692
ID5	0.564904
ID26	0.629808
ID8	0.716346
ID15	0.762019
ID28	0.687500
ID9	0.646635
ID1	0.490385
ID6	0.555288
Mean	0.643502
Variance	0.013682

Table 4. The mean accuracy rate across all predictions for each 1vsALL classifier that trained on four sensors (*clutch*, *gaspedal*, *rpm* and *speed*). The highest accuracy is highlighted.

	4 sensors	**clutch**	**gaspedal**	**rpm**	**speed**
ALLvsALL	**0.153924**	0.1132	0.11686	0.059157	0.047674

Table 5. The mean accuracy rate for the ALLvsALL experiment. The highest accuracy is highlighted in this table.

6.3 ALLvsALL

In the ALLvsALL experiment the best model achieves an accuracy of **15%**, which is approximately 5 times better than a random guess! (A random guess in this case would be $1/33 = 0.0303$, which is about 3%.) The best performing model combined the same 4 inputs that were used in the previous experiment, as described in Section 6.2. One can conclude that in this case the model improved significantly when using the combined input of the 4 best-achieving sensors. Results can be seen in Table 5.

6.4 Personal Attribute Inference

The results of the personal attribute inference can be seen in Table 6. One can conclude that the models that are trained on all four sensors are almost always superior to the models that are trained only on one sensor (the only exception to the rule is the inference of experience, with only a slight difference between the two best-scoring models). Thus I achieve a very impressive improvement over the baseline in all cases. For gender inference, I get a model that peforms 1.4 times better than a random guess would, for age inference I get an even larger improvement (2.2 times better), and last but not least the experience is 1.6 times better than a random guess. In two cases out of three the model that was built on four sensors achieves far better results than the competing models.

	4 sensors	**clutch**	**gaspedal**	**rpm**	**speed**
gender	**0.706410**	0.58141	0.634776	0.549038	0.59407
age	**0.555756**	0.416305	0.485188	0.32671	0.305093
experience	0.529144	0.413597	**0.550647**	0.357857	0.322982

Table 6. The mean accuracy rate for personal attribute inference. The highest accuracies are highlighted in this table. The baselines for gender is 50%, for age it's 25% and for experience it's 33%.

In Figure 6 I plotted the ROC curve for each classifier that was built for gender inference. It can be observed here as well that the classifier built on the combined input of the 4 sensors is the best.

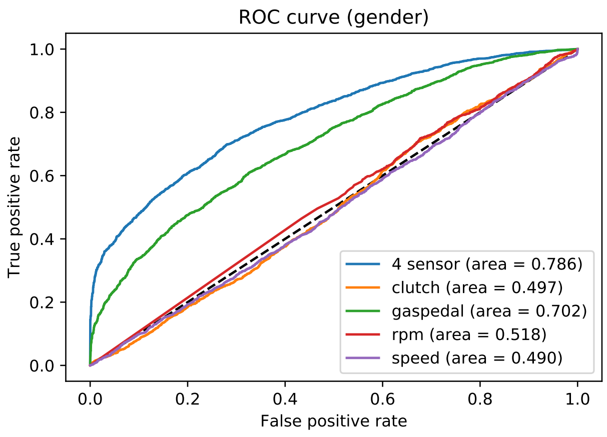


Figure 6. ROC curve for the 5 models that were built for gender inference. AUC (Area Under Curve) values are shown in the legend.

Attribute	*Value*	*Class*
Age	[20-25]	0
	[25-30]	1
	[30-40]	2
	[40-70]	3
Experience	Low	0
	Average	1
	High	2

Table 7. Personal attributes to class mapping (For better understanding of the ROC curves).

I have extended the ROC curve analysis onto age and experience inference as well. For instance, in age inference we have 4 classes. I have created 4 different ROC curves for these 4 settings (classes can be found in Table 7):

- Class 0 vs Class 1&2&3
- Class 1 vs Class 0&2&3
- Class 2 vs Class 0&1&3
- Class 3 vs Class 0&1&2

In Figure 7 we can see the ROC curves for the best classifier on age inference. It is interesting that the class that is hardest to classify is Class 2 (age [30-40]). Similarly, we can see the ROC curves of the other classifiers in Figure 8. The model that is trained on *gaspedal* exhibits the same behavior as the model that is trained on all 4 sensors - here too Class 2 is hardest to classify.

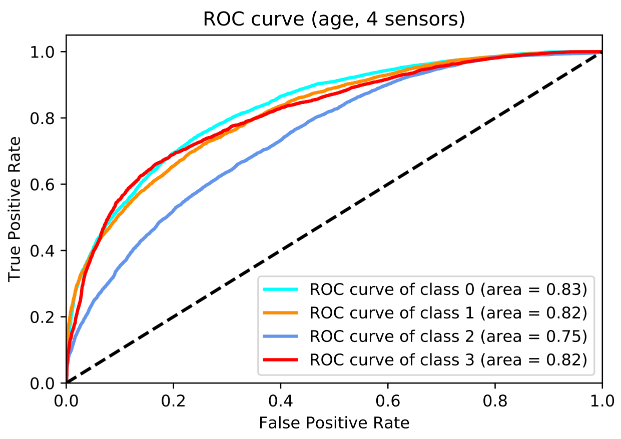


Figure 7. Age inference ROC curve for the classifier that uses all 4 sensors. AUC (Area Under Curve) values are shown in the legend.

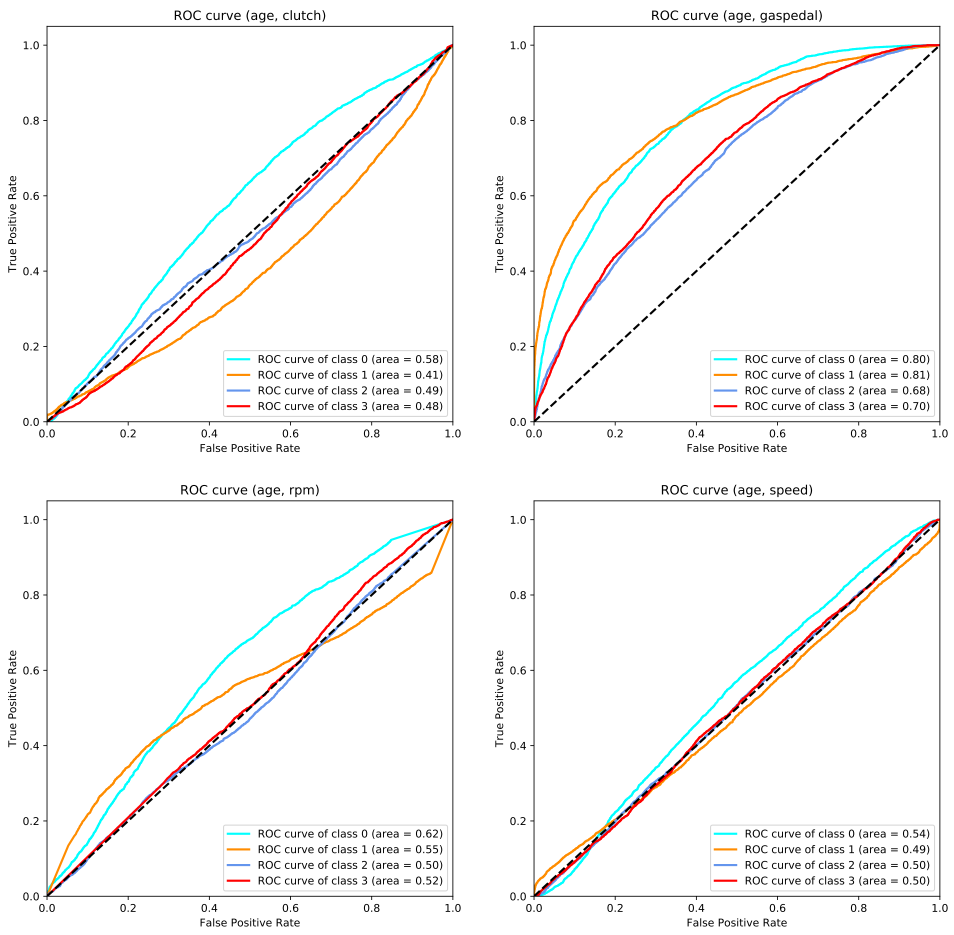


Figure 8. Age inference ROC curve for the classifier that uses all 4 sensors. AUC (Area Under Curve) values are shown in the legend.

The ROC curve for the best classifier on experience inference can be seen in Figure 9 (in this case it shows the model that is trained on the *gaspedal*). This figure shows us that a more experienced driver can be more easily classified, than an average or an inexperienced one. However, the model that is trained on all four (which does not fall behind very much in terms of accuracy) distinguishes the most and the least experienced drivers the best! This ROC curve can be seen in Figure 10.

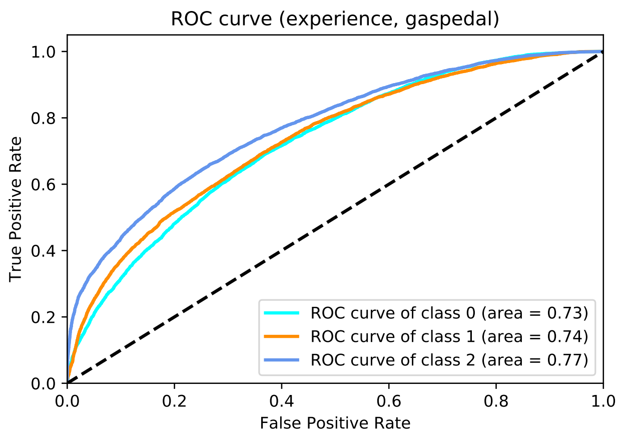


Figure 9. Experience inference ROC curve for the classifier that uses the measurement of *gaspedal*. AUC (Area Under Curve) values are shown in the legend.

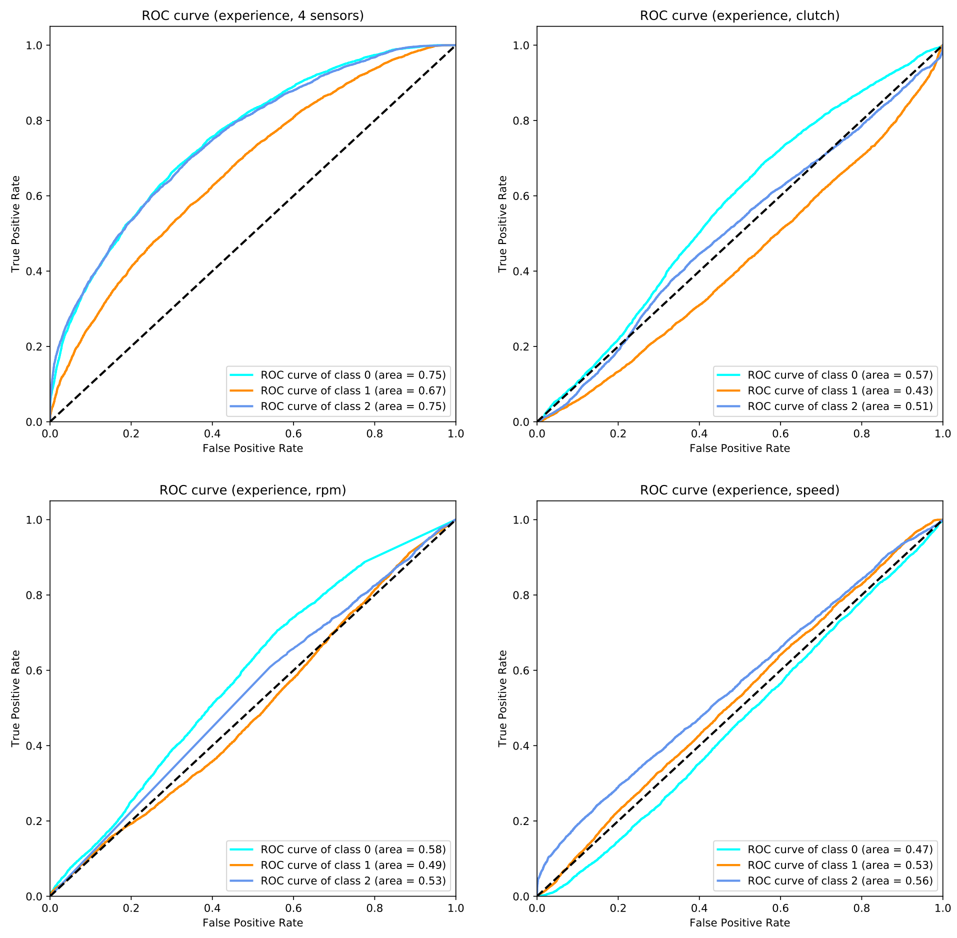


Figure 10. Experience inference ROC curve for the rest of the classifiers. AUC (Area Under Curve) values are shown in the legend.

7 Conclusion and Future Work

In my work I have presented a novel way of performing driver re-identification, which takes advantage of CNN's ability to single out local patterns that are unique to our driving. This achieved a 66% mean accuracy in distinguishing one driver from all the others, and a 15% accuracy when performing multiclass classification on all 33 drivers. Both are satisfying indicators that driver inference is in fact possible using only CAN bus data. These results were achieved despite the fact that the drivers did not take the same route (as opposed to other's work [1] [2]). They are also commendable for the fact that the predictions were made based only on 10 second long samples.

Furthermore, I have shown significant results in predicting the driver's gender (71%), age (56%) and experience (53%). This tells us that CAN bus data does not only potentially identify, but also contains personal information on that person. The GDPR imposes strict regulations regarding data which can profile a person, and thus CAN data should be handled with the utmost precaution. Therefore it is the duty of car companies to properly inform the drivers on what kind of data they are collecting and potentially sharing with 3rd parties. An alternate solution could be to anonymize the data. However, research shows that it is rather hard to anonimyze high-dimensional data such as this [20].

Future work could include building a more accurate (yet scalable) end-to-end model, tested on an even larger pool of drivers. Another challenging problem would be to make the same kind of predictions based only on raw CAN logs. It is often the case that we don't know where (on which byte) the individual sensor signals travel (car manufacturers like to keep it secret for security reasons). However this would bring about its own challenges, like how do you tell the relevant time series apart from hundreds of irrelevant ones, while keeping everything scalable? This, and many more questions wait to be answered.

References

[1] Miro Enev et al. “Automobile Driver Fingerprinting”. In: *Proceedings on Privacy Enhancing Technologies* 2016.1 (Jan. 2016), pp. 34–50. DOI: 10.1515/popets-2015-0029.

[2] David Hallac et al. “Driver identification using automobile sensor data from a single turn”. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)* (2016). DOI: 10.1109/itsc.2016.7795670.

[3] Zhicheng Cui, Wenlin Chen, and Yixin Chen. “Multi-Scale Convolutional Neural Networks for Time Series Classification”. In: (2016). arXiv: 1603.06995 [quant-ph].

[4] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. “Data Augmentation for Time Series Classification using Convolutional Neural Networks”. In: *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data* (2016).

[5] Chiyomi Miyajima et al. “Study on Driver Identification Method Using Driving Behavior Signals”. In: (2006). DOI: 10.1093/ietisy/e89d.3.1188.

[6] Sang-Hyeun Park and Johannes Fürnkranz. “Efficient Pairwise Classification”. In: *Machine Learning: ECML 2007 Lecture Notes in Computer Science* (2007), pp. 658–665. DOI: 10.1007/978-3-540-74958-5_65.

[7] Umberto Fugiglando et al. “Characterizing the ”Driver DNA” Through CAN Bus Data Analysis”. In: *CarSys ’17 Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services* (Oct. 2017), pp. 37–41. DOI: 10.1145/3131944.3133939.

[8] Yi Zheng et al. “Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks”. In: *Web-Age Information Management Lecture Notes in Computer Science* (2014), pp. 298–310. DOI: 10.1007/978-3-319-08010-9_33.

[9] Zhiguang Wang, Weizhong Yan, and Tim Oates. “Time series classification from scratch with deep neural networks: A strong baseline”. In: *2017 International Joint Conference on Neural Networks (IJCNN)* (2017). DOI: 10.1109/ijcnn.2017.7966039.

[10] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016). DOI: 10.1109/cvpr.2016.90. URL: http://dx.doi.org/10.1109/CVPR.2016.90.

[11] Hassan Ismail Fawaz et al. “Deep learning for time series classification: a review”. In: (Sept. 2018). arXiv: 1603.06995.

[12] Xiaopeng Xi et al. “Fast time series classification using numerosity reduction”. In: *Proceedings of the 23rd international conference on Machine learning - ICML 06* (2006). DOI: 10.1145/1143844.1143974.

[13] Anthony Bagnall et al. “Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (2015), pp. 2522–2535. DOI: 10.1109/TKDE.2015.2416723.

[14] Hoang Anh Dau et al. *The UCR Time Series Classification Archive.* https://www.cs.ucr.edu/~eamonn/time_series_data_2018/. Oct. 2018.

[15] Anthony Bagnall et al. "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery* 31.3 (2016), pp. 606–660. DOI: 10.1007/s10618-016-0483-9.

[16] Y. LeCun et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in Neural Information Processing Systems (NIPS 1989)*. Ed. by David Touretzky. Vol. 2. Denver, CO: Morgan Kaufman, 1990.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[18] Saining Xie and Zhuowen Tu. "Holistically-Nested Edge Detection". In: *The IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[19] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: ().

[20] Charu C. Aggarwal. "On k-Anonymity and the Curse of Dimensionality". In: *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. 2005, pp. 901–909. URL: http://www.vldb.org/archives/website/2005/program/paper/fri/p901-aggarwal.pdf.