DEPARTMENT OF CIVIL, ENVIRONMENTAL AND GEOMATIC ENGINEERING

Semester Project Report

Data-driven identificiation and classification of rail surface defectse

Aiyu Liu

Supervised by: Cyprien Hoelzl, Prof. Eleni Chatzi

Tuesday 11th February, 2020

Acknowledgements

Contents

Chapter 1

Introduction

Railway companies need to continuously and sufficiently maintain the train tracks and optimally detect defects in order to have a more punctual and more effective train system. However, the current system is expensive, time consuming and ineffective. That is, maintenance agents need to walk along tracks and check them for defects. For visualisation purposes, there is roughly 5200 km of rails in Switzerland which needs to be inspected by 40 experienced inspectors.

In order to cope with this issue, Swiss Federal Railways (SBB) has specifically built two new special diagnostic vehicles (SDV) designed for defect identification among other purposes. For this, two accelerometers have been installed at the front and back of the vehicle to collect the signal responses from the wheel and the train track

insert picture?, mention boogey?

A defect in train tracks can be seen as a discontinuity. As a train passes over this discontinuity, it will result in a perturbation that can be detected by sensors. It is our main assumption that each type of defect will have a specific signature that will allow its identification and classification. This is similar to the idea presented in [1] about human activity recognition.

By succesfully identifying and classifying the defects, we take one step further towards reducing delays and making trains more punctual and reliable. The first step in this process consists of identification and classification, while the second step consists of future defect prediction.

1.1 Objective

As the title implies, the objective of this project is to identify and classify rail surface defects.

apply machine learning techniques on the problem

1.2 Defects

Evidently, a defect can be seen as a deviation from the standard train track. For the exact defect type, SBB has self-constructed a database for the individual defect definitions . Here is a few examples:

is this a recognized system?

Generally, a defect is separated into two overarching types: range- and point-defects. I.e. a defect that is detected at a single point versus a defect that is detected at varying lengths – e.g. .

insert pictures

insert example

For this project, we have solely focused on the point defects for analysis, as this simplifies the problem statement. . A point defect is perceived as a sharp signal response, whereas a range-defect is perceived over a greater time period. We thus disregard range-defects such that we do not have to deal with the extra, associated factors.

show signal types?

See list of defect types in appendix?

1.3 Data

The data has been collected and provided by SBB. Using their SDV, SBB has made trips back and forth to different cities in Switzerland in order to collect various data including but not limited to accelerometer data. After getting the data from SBB, it then goes through a processing pipeline (designed by Cyprien), after which the data can be manipulated with `python` dataframes (from `pd.DataFrame`). The accelerometer captures the accelerations at the XYZ-axes (along with the timestamps at each recording), of which we are only concerned with the Y-axis for the vertical pertubations.

make a table of the equipment and sample frequencies

Furhtermore, the locations of the defects have to be retrieved from SBB's database. which were retrieved by Cyprien.

We need to define terminlogy of these: defects, ins joints = in the following we will use defect as an umbrella term for these entities.

1.4 Code

Which data was worked on, put in tables, switches, ins joints and defects

The code is written purely in python. The code can be found on github: https://github.com/Aiyualive/SemesterProject2.0.

which track entities are we actually analysing

Chapter 2

Design and Implementation

For the process of defect classification we employed the following pipeline:

In the following, I would like to give an overview of how these was implemented

insert pipeline picture

2.1 Shift of GPS timestamps

The SDV has its GPS sensor installed at a specified location on the vehicle body. However, what we need to achieve is the position (covered distance) at each accelerometer at either sides of the GPS. Since the GPS sensor is sampled at a lower frequency compared to the accelerometers, we first need to get the corresponding positions for each accelerometer sample. This is done by interpolation using the timestamps of the accelerometers and GPS.

Depending on the direction of the vehicle we then subtract/add the offset between the accelerometers and the GPS sensor with regard to the position of these sensors on the vehicle body.

show a few entity signals and their features, appendix for more signals?

insert drawing of how it is calculated?

2.2 Peak windows

Retrieving the signal response around the defect location forms a crucial aspect in the overarching pipeline. The goal of this step is to, around each defect, create a "window" containing accelerometer accelerations of a specified time length – wherein Within the highest acceleration recording around is found in the center. As a result, all of these windows would be uniform in the sense that they are all centered according to the highest recording of a defect. It is then assumed that each window forms the signature of each track entity.

Since we are assuming that each track entity is identified by a well-formed peak, we first need to find this peak within a reasonable offset from the defect location, after which we center around that within another reasonable offset.

In the code, this is done by defining two parameters: `find_peak_offset = 1` and `window_offset = 0.5`. I.e. given a defect timestamp, we search for the highest acceleration recording that has occured 1 second after and 1 second before the defect timestamp. Once the peak has been found, we then center it in a 1 second window (0.5 sec on each side).

insert a drawing of how this works?

2.3 Entity library

The peak windows arguably forms the central feature of the defect library. However, from domain knowledge, other features like speed also needs to be considered for our neural network. Apart from the peak windows, we have also extracted a variety of relevant features that might be useful for classifcation. These include .

list, and how was it implemented

some can be extracted directly by the original dataframe, some needs some processing.

2.4 Neural network

To create neural network architectures, we are using: `keras` along with `tensorflow`. `keras` is essentially a high-level neural networks library which runs on top of `tensorflow`. It has consistent, a simple API and provides clear and actionable feedback upon user error. Models are easily made by connecting configurable building blocks together, with few restrictions [2].

For our use case, we have created a primary `NN` class (short for neural network) along with a `ModelMaker` class. The former does everything from pre-processing the data to evaluating the used model. The latter, as the name suggests, is utlised for creating and using different models, which is useful as we can keep track of how the models have been modified and improved.

To make a classification, we first need to select the relevant features. Then we simply feed the features into an `NN` object, where the API of the `NN` class can be called for classification. The usage of the `NN` class is demonstrated below in 2.1.

Workflow of NN class	
`__init__()`	initialises a `NN` object
`prepare_data()`	pre-process data, this includes standardisation of data
`make_model()`	uses `ModelMaker` class to select a model
`fit()`	trains the model

Table 2.1: To train a model, these functions needs to be called sequentially

Other useful functions	
`predict()`	used to classify a testing set in the future
`measure_performance()`	currently only done on validation data
`plot_metrics()`	plots the metrics
`plot_confusion_matrix()`	plots the confusion matrix
`save_history()`	
`save_model()`	
`run_experiment()`	evaluates the given model for a number of times

Table 2.2: f

insert predict-classify after this?

2.5 Visualisation

Finally, after evaluating the results (results can be seen in the next section) from the neural network, we have not achieved any significant results. Arguably, the visualisations of class separability should have been handled first. However, the previous steps took the majority of the time.

Chapter 3

Evaluation

Here we will present the results and discuss the findings herein.

3.1 Model

Table 3.1: Table Title

Layer	Output Shape	Number of params
Synthetic data	2000	9

Dataset formats

http://alexlenail.me/NN-SVG/AlexNet.html

3.2 Classification results

Model / run #	Model1	ClaveVectors	MNIST-bin-digits	MNIST-all-digits
1		61.355	3212.497	2454.585
2	16.568	62.396	3346.047	2782.271
3	14.362	59.282	2908.446	2618.519
4	13.85	62.193	3043.189	2600.233
5	13.755	61.024	3309.422	2470.139
6	12.218	58.531	3192.178	2823.685
7	12.232	60.187	3287.289	2452.566
8	13.346	59.398	3026.486	2810.943
9	11.433	61.892	3127.583	2528.202
10	11.63	60.925	3018.402	2746.729

Table 3.2: Architecture of model number

insert the different defect types and

3.3 Visualisation results

insert the plots

3.4 Discussion

I tried to increase the outliers, but this was a hugely naive approach

Chapter 4

Conclusion and future work

4.1 Conclusion

4.2 Future work

- Might be interesting to also consider the XZ-axes.
- range defects
- tune the peak finding parameters
- track entity dependent/specific window offsets
- we must not set the findpeakoffset too high
- Questions, what if you want to use multiple features with 1 CNN

4.3 TODO

- get percentage of each class in the validation set
- create an average of the model ie run model for more times
- add speed as a feature
- what does each filter do? what is kernel size?
- very fast speed, overlap between switch and ins, old vs new rail, ax1 arrow 2 arrow 3 arrow 4
- 3D plots?
- change the defect library to use pandas instead?
- visualise what the network is doing using Harry's code
- use speed as a feature also
- be consistent with function naming and variable names: function names with underscore and variable names with camelcase
- save confusion matrix, and metrics (modify the plotting of this)
- citation [**?**]
- Which type of defects are we actually working with, we can see that it does good at the switches and ins joints, but no chance with the defects
- we should instead call it an entity library – make up your mind
- plot confusion matrix instead, and classify instead of predict
- do objective, and entity thingy

Bibliography

[1] Introduction to 1d convolutional neural networks in keras for time sequences. https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf. (Accessed on 02/11/2020).

[2] Tensorflow vs keras: Which one should you choose. https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/. (Accessed on 02/10/2020).

Appendix A

Implementation

Functions in NN class
prepare data

Appendix B

Results

Latent Dimension	2
Shape	8×8
Latent Points	64
Dist. Mixtures	Bernoulli
Regularisation α	0.001

RBF Dimension	2
RBF grid	4×4
RBF centres	16
Basis function	Gaussian
Widths $\boldsymbol{\sigma}$	1.0

Table B.1: The parameters for the latent model and the RBF neural network. Distribution mixtures and basis functions cannot be altered as our implementation is specific for the Bernoulli GTM version