Reproduction of Driver Identification Based on Vehicle Telematics Data using LSTM-Recurrent Neural Network by Girma et. al.

Group 48

Authors:
Sander Boers - 4670299 - s.h.boers@student.tudelft.nl

Mathijs van Geerenstein - 4598660 - m.r.vangeerenstein@student.tudelft.nl

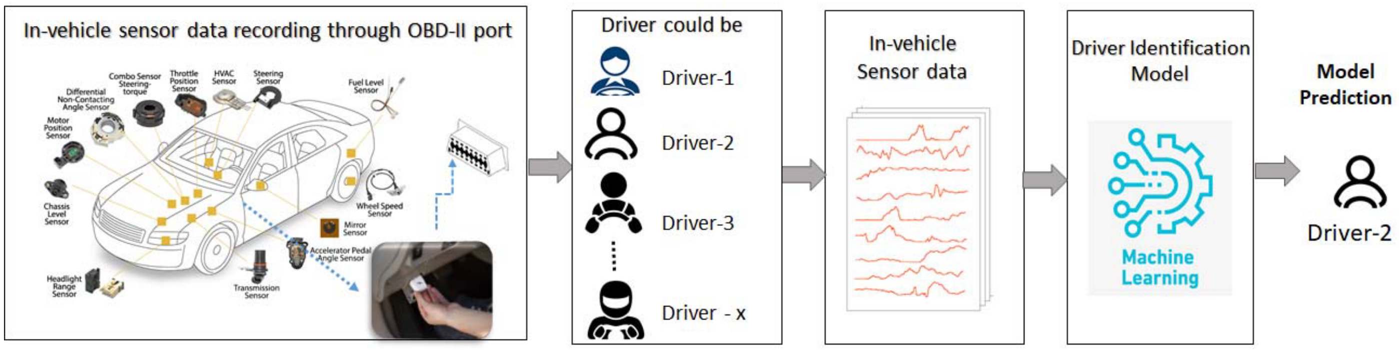
Tom Weinans - 4445449 - t.h.weinans@student.tudelft.nl

Abstract

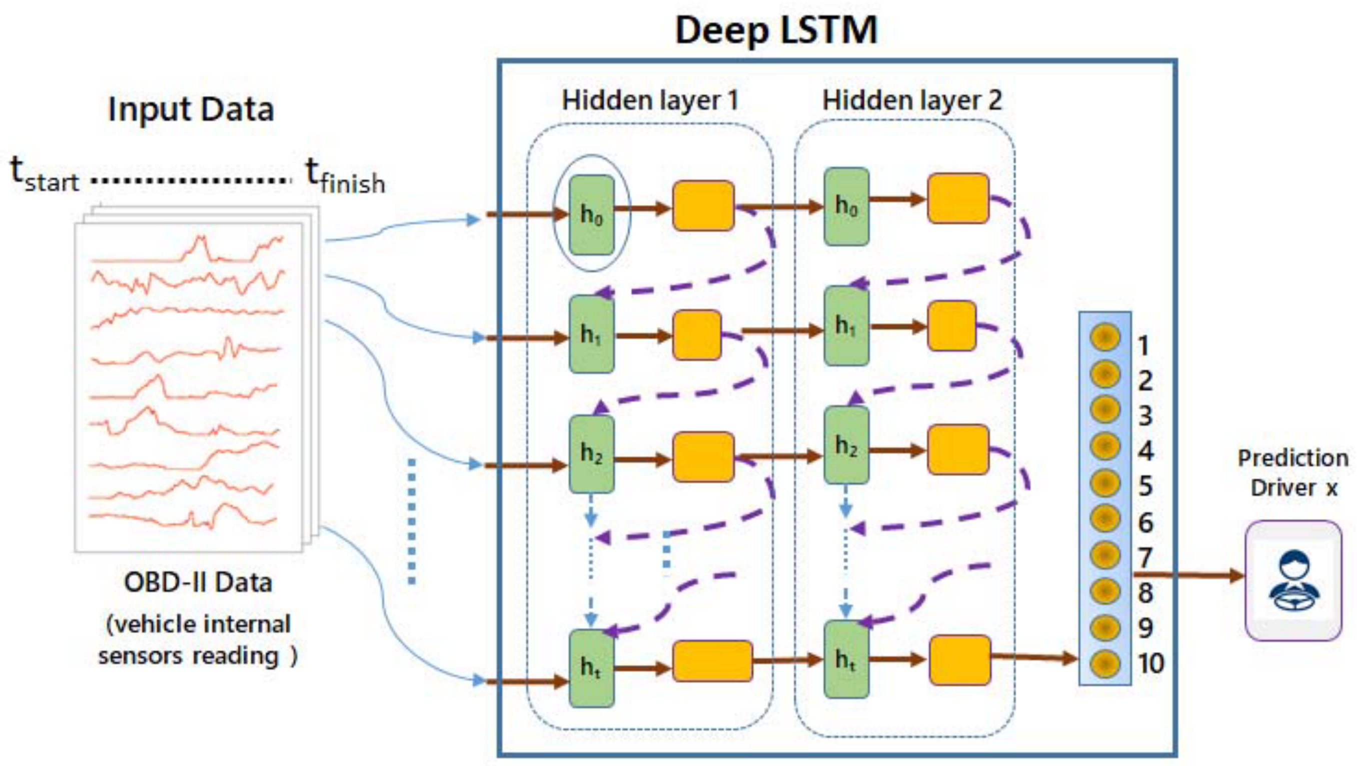
The aim of this blogpost is to describe the reproduction of the LSTM-Recurrent Neural Network proposed by Abenezer Girma et. al. [1]. This deep learning model is able to identify drivers by their driving behavior based on vehicle telematics data. Specifically, the proposed method is robust to noise and anomalies in this time-series data. In our attempt to reproduce the results, we show similar trends to Abenezer Girma et.al., but do not fully reproduce their observations.

Introduction

Over the years car technology has improved a lot, but the cars security systems did not evolve that much. The amount of car thefts has not dropped over the years. More specifically, the relative part of “digital” car thefts (where no brute force is used, but where the car is unlocked and started by hacking it) has increased. To fight this development, car alarms could operate by detecting who is actually driving the car. If the driving style does not match that of the owner, a car alarm can still identify a thief. The paper from Abenezer Girma et. al.[1] describes a deep learning model, which is able to identify drivers based on vehicle telematics. Specifically, it proposes a model structure that is especially robust to noise and other data anomalies, as is common with car sensors. The paper compares the achieved results on driver identification with more popular models. This blogpost will describe our attempt to reproduce the results achieved in the paper.



The proposed method is a deep LSTM model. Long-Short Term Memory (LSTM) models are a class of Recurrent Neural Networks (RNNs) that are able to learn the order dependence in sequence prediction problems. This approach is applicable on driver identification, because there is sequential input data available to do the prediction. The input data includes various internal vehicle sensor readings for a certain time window. The model is a classifier which predicts the driver as output. A top-level overview of the approach is shown in the figure below.



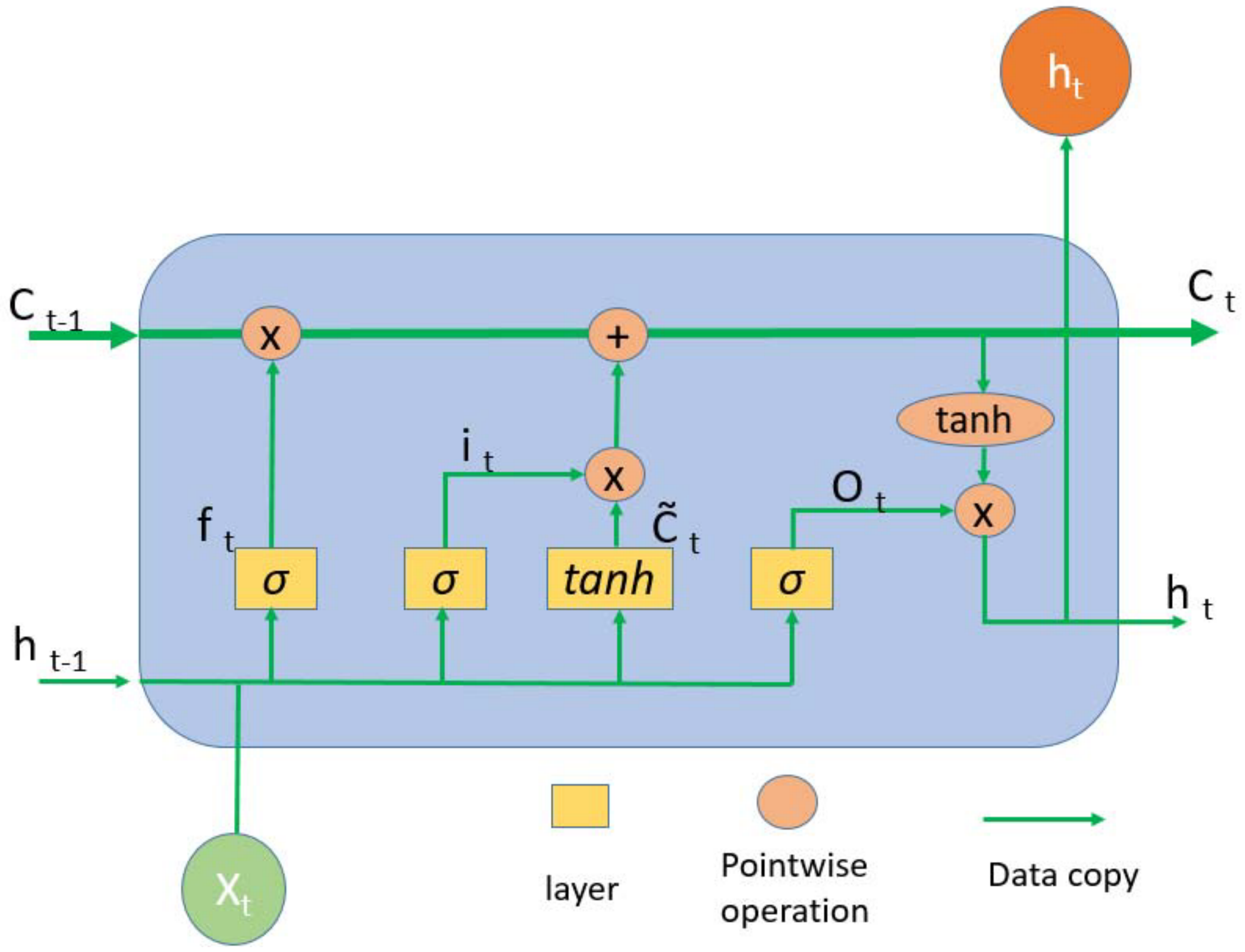
Methods

In this problem of driver identification, we make use of OBD-II data, which is sequential sensor data collected during a driving trip. Each input to the model is a snippet of time-series ODB-II data, which includes a variety of up to 50 sensor data points per time instance. For robustness in the driver prediction, we want a model that has little sensitivity to noise (white gaussian noise) and sensor anomalies (broken sensor, bad connection, etc).

LSTM Model

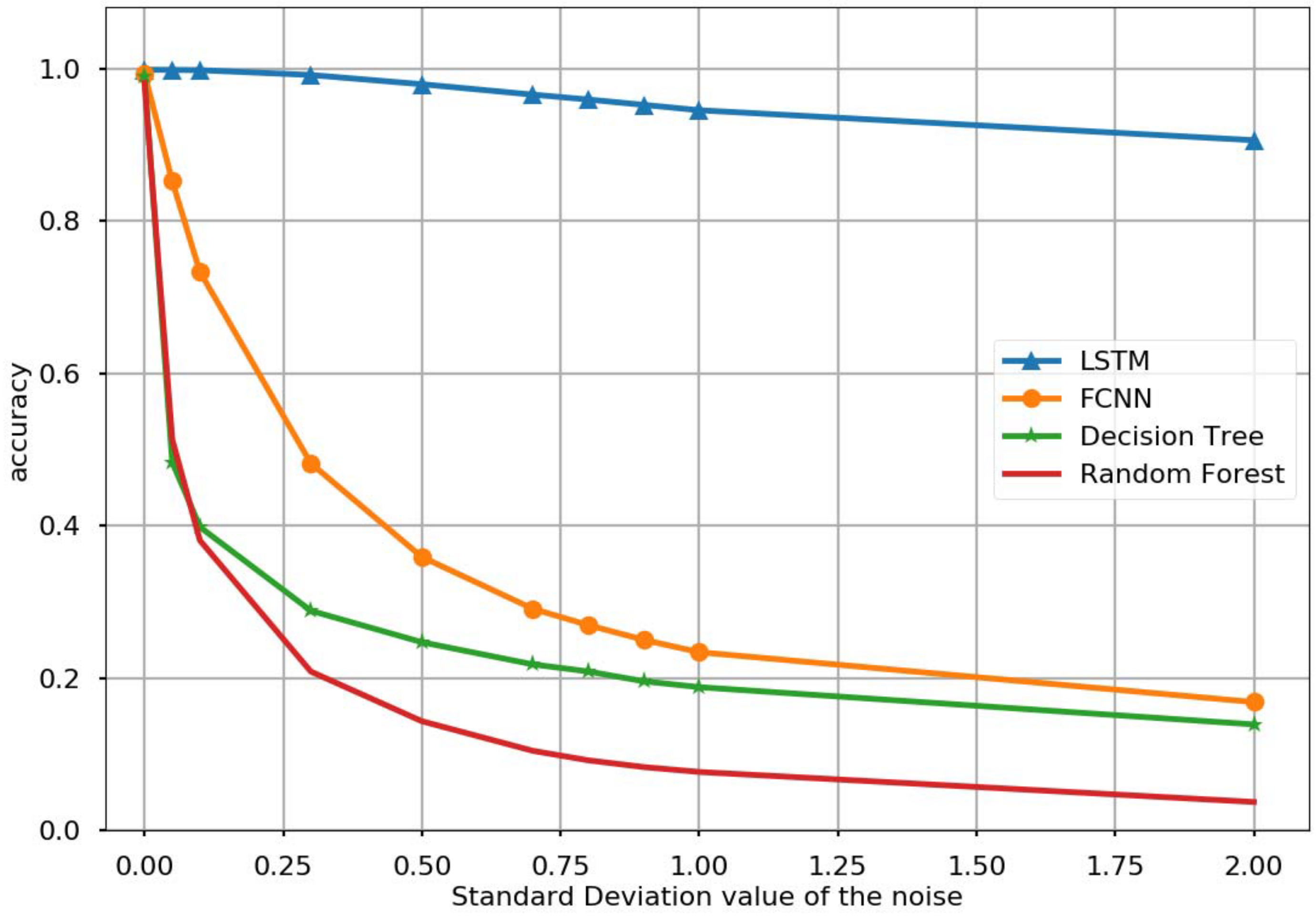
Recurrent Neural Networks (RNNs) are one of the most successful techniques to do time-series classification tasks such as speech recognition. RNNs take sequential input data, and give a certain output based on a decision made by the internal state. The internal state extracts features and can capture temporal dynamics of time-series data. The problem with RNNs, is that it can suffer from the “vanishing gradient” problem whenever the number of internal (hidden) states increases. This results in the RNN being unable to learn.

The proposed method uses a Long-Short Term Memory (LSTM) network, which does not suffer from this problem. Because of a memory block in the hidden layer, the model can capture long-term dependencies. Because of this memorizing ability, LSTM has shown great performance on time-series tasks. A visual representation of an LSTM memory block is shown in the figure below.

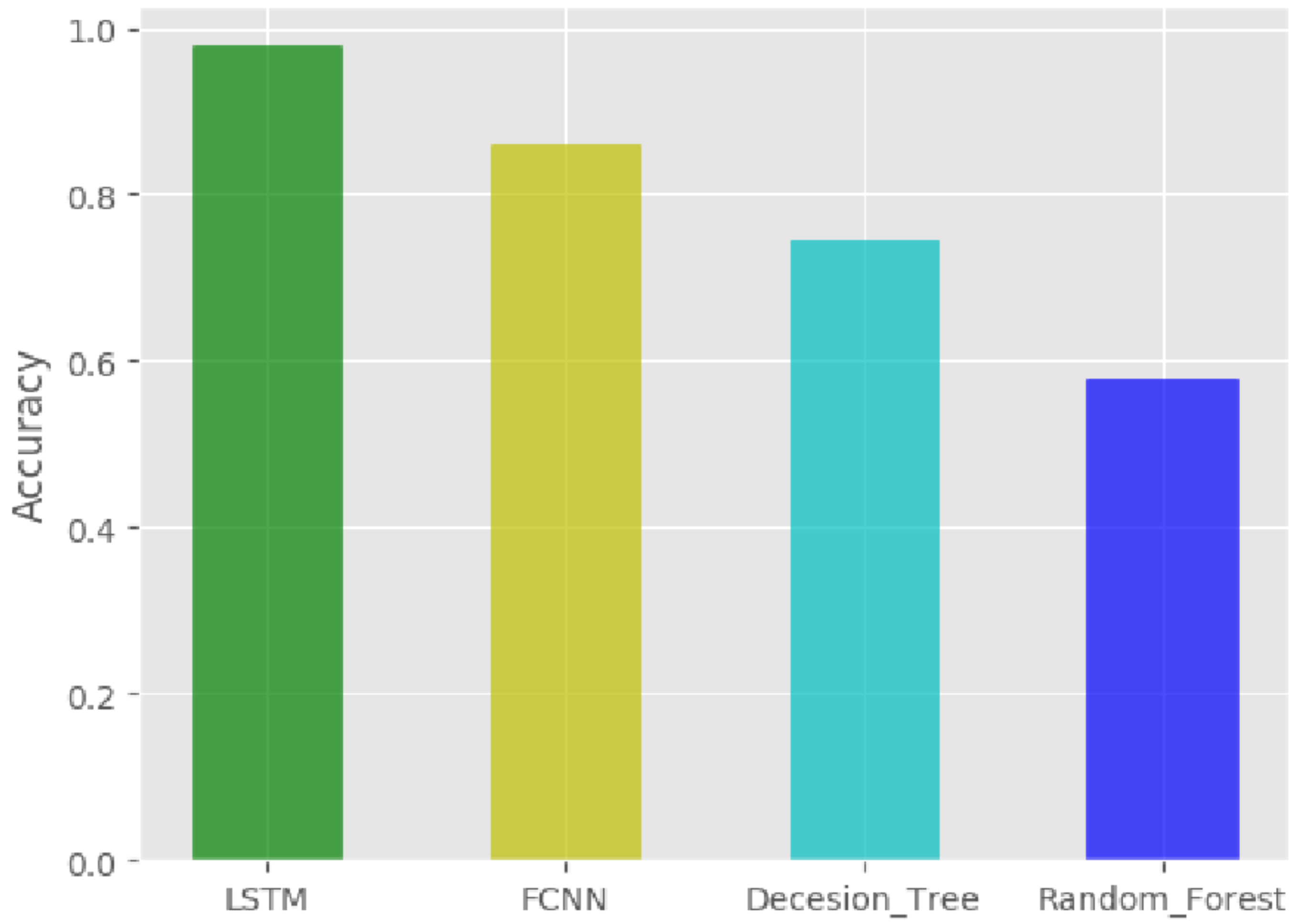


Evaluation

To evaluate the proposed method, the LSTM network is compared with three other methods, namely a fully connected neural network (FCNN), a decision tree model and a random forest model. All models are trained on the same training data. To test the robustness, each model is tested several times, each with more artificial noise added to the test data. Without any noise added to the test data, all models perform similarly in the driver classification task. However, when artificial noise is added, the performance of the FCNN, decision tree and random forest drops steeply, while the LSTM model does not. The results achieved by the paper are shown in the figure below.



Additionally, the proposed method is evaluated by training on noisy sensor data, and then testing on noisy sensor data. Again, the LSTM model shows superior performance over the other three methods it is compared with. These results are shown in the figure below.



Results

This section shows our attempts in reproducing the comparison of the proposed LSTM model with three other methods: a fully connected neural network (FCNN), a decision tree and a random forest.

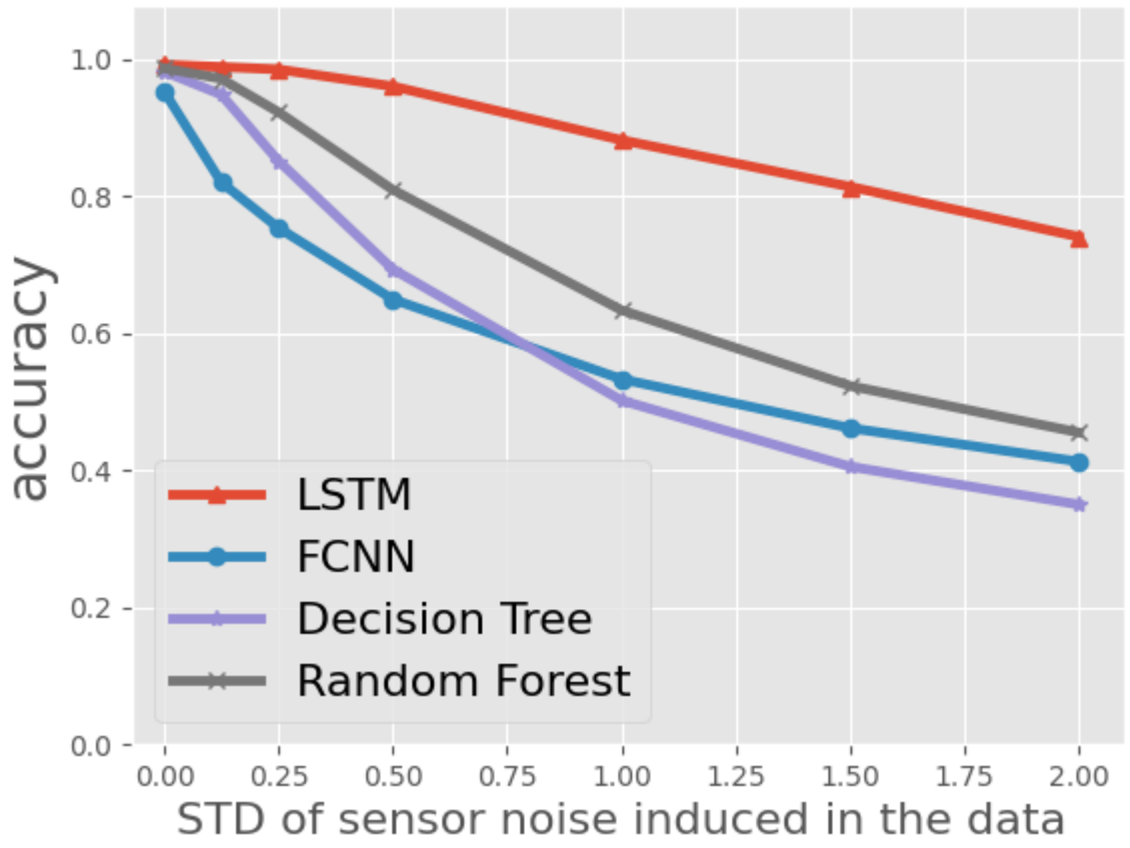
Reproduction

In this subsection an attempt was made to reproduce the figures in the paper. We have tried to do so using the code provided on the github mentioned in the paper, however the pre-trained model which is provided there did not use the (full) dataset (more on this in the discussion). Because of this we have retrained the model with parameters that fit the full dataset, the details of which are shown below.

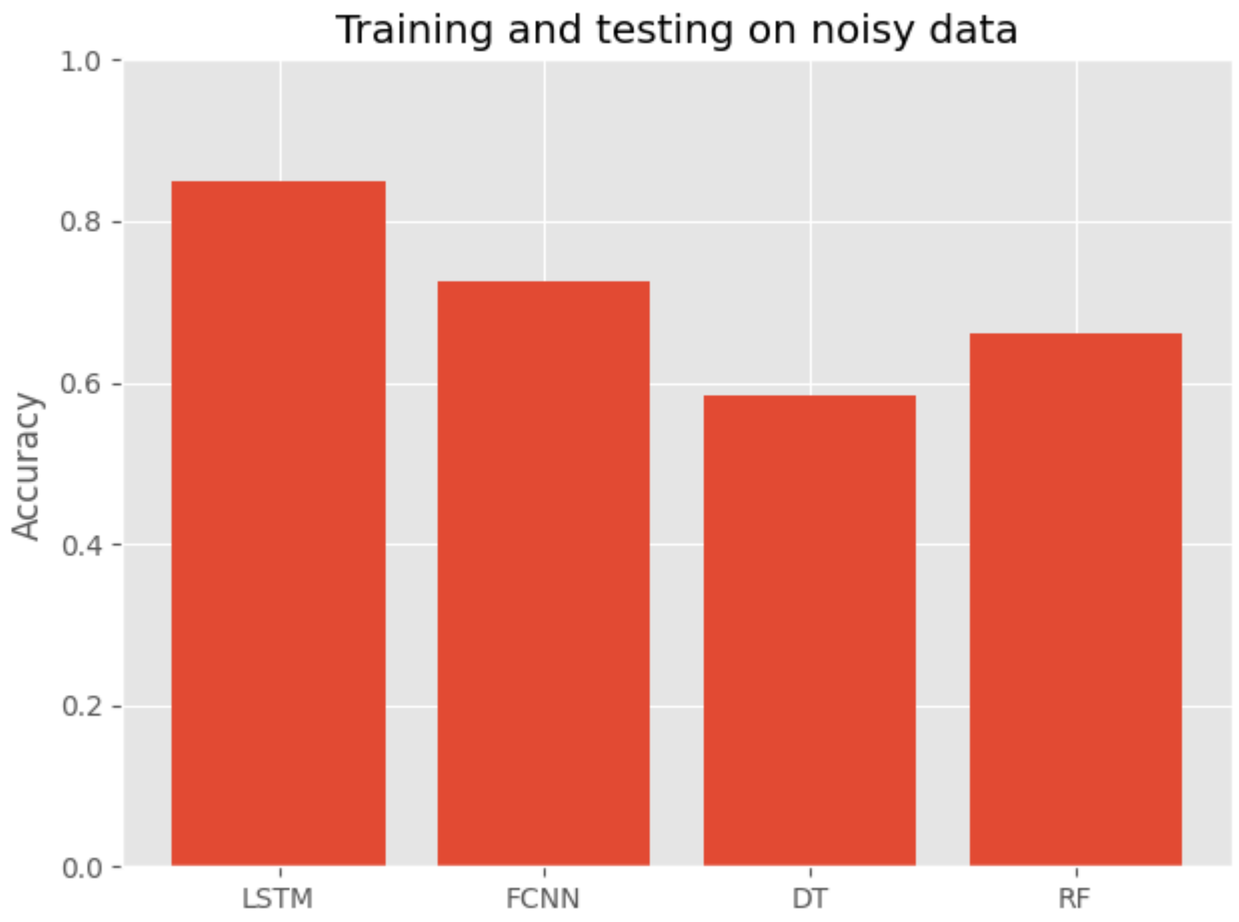
Hyperparameters	
Epochs	25
Learning Rate	1e-3
Optimizer	Adam

Hyperparameters	
Criterion	BinaryCrossEntropy

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=(None,NUM_FEATURES)))
model.add(tf.keras.layers.LSTM(160, input_shape=(None,NUM_FEATURES), return_sequence
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(.2))
model.add(tf.keras.layers.LSTM(120, input_shape=(NUM_FEATURES,)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(.2))
model.add(tf.keras.layers.Dense(NUM_CLASSES))
model.add(tf.keras.layers.Softmax())
lstm_model = tf.keras.models.clone_model(model)
```



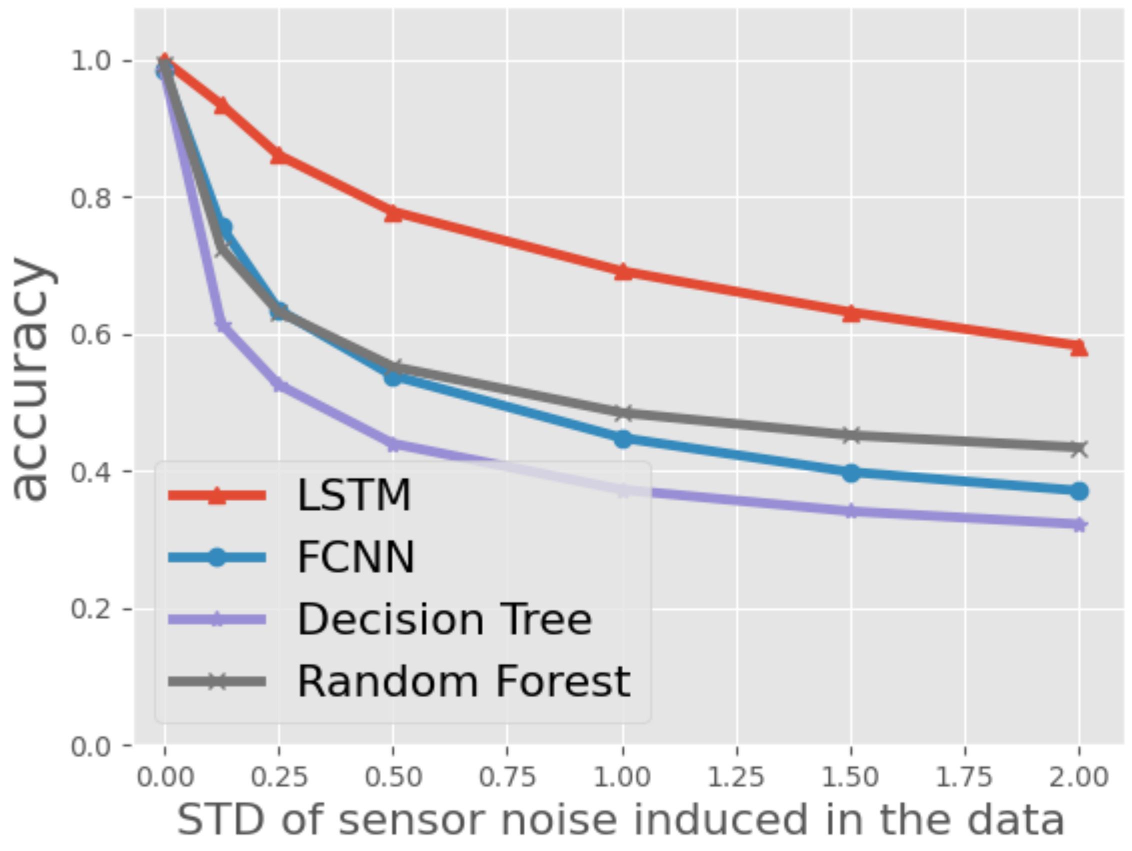
The plot above shows our result of the reproduction. As can be seen LSTM still performs best, compard to the other three. However the decay of this line is more prominent than the in the plot of the paper. The reason for this can be read in the discussion.



This barplot also shows that LSTM has the best accuracy for training and testing with noise. But the accuracy shown in the paper, is not achieved. Reasons can also be found in the discussions.

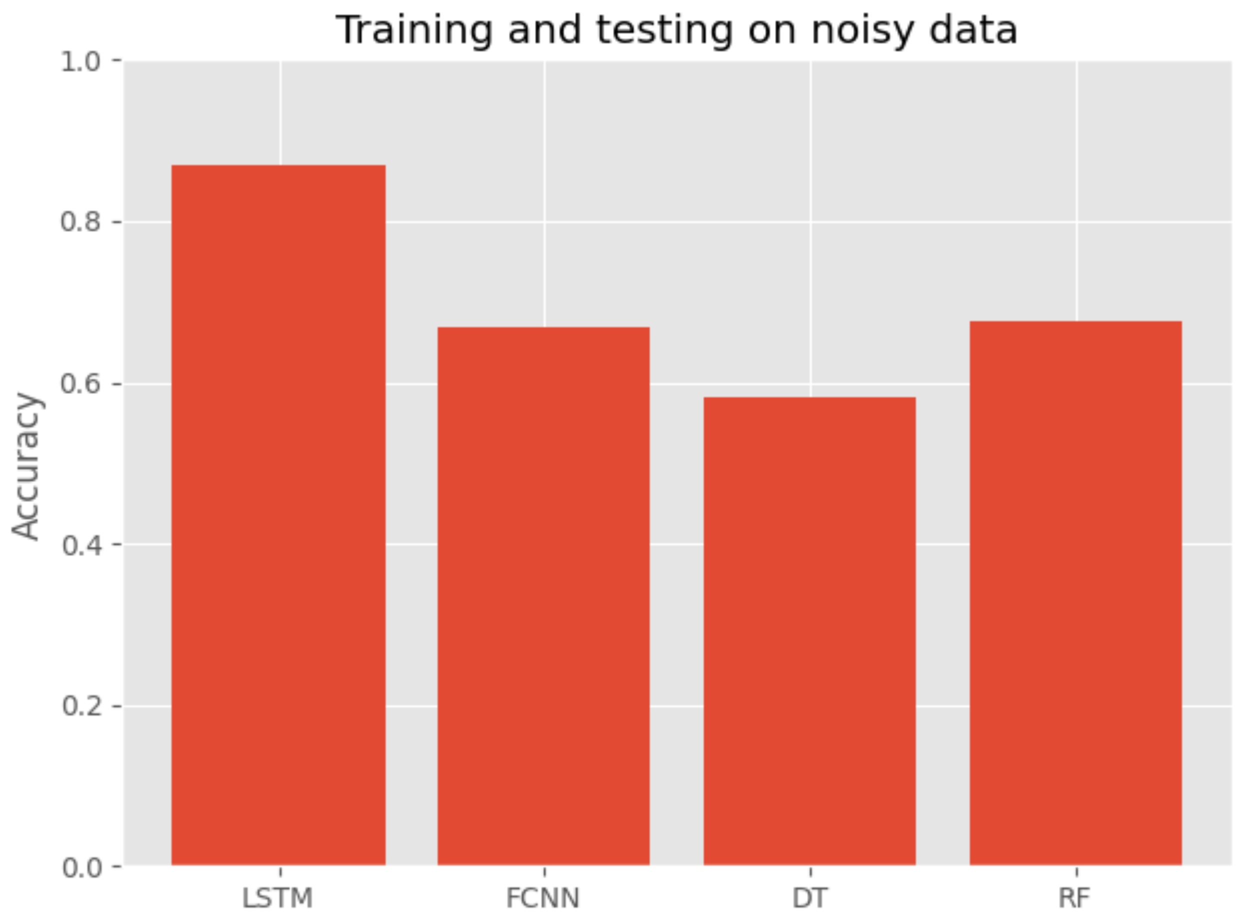
New data

In addition to the reproduced results the choice was made to also check this implementation while training on a different dataset. For this the Vehicular-trace dataset [2] is used. This dataset consists of two parts. The first part has 10 different drivers and the second part has 4 different drivers. In order to compare this new dataset with the original we have used the first part of this dataset with 10 drivers and 21 comparable features. Training the LSTM, FCNN, Decision Tree and Random Forest on this dataset yields the following result for the accuracy vs noise induced data:



Comparing the figure above with the one trained on the original dataset shows similar trends for the FCNN, Decision Tree and Random Forest. THe LSTM however is performing significantly worse with an accuracy that is less than 60% at 2*STD compared to over 70% for the original dataset.

The other figure that needed to be reproduced is a figure comparing the accuracy of LSTM, FCNN, Decision Tree and Random Forest trained and tested on noisy data. The result of training with the new dataset is shown below.

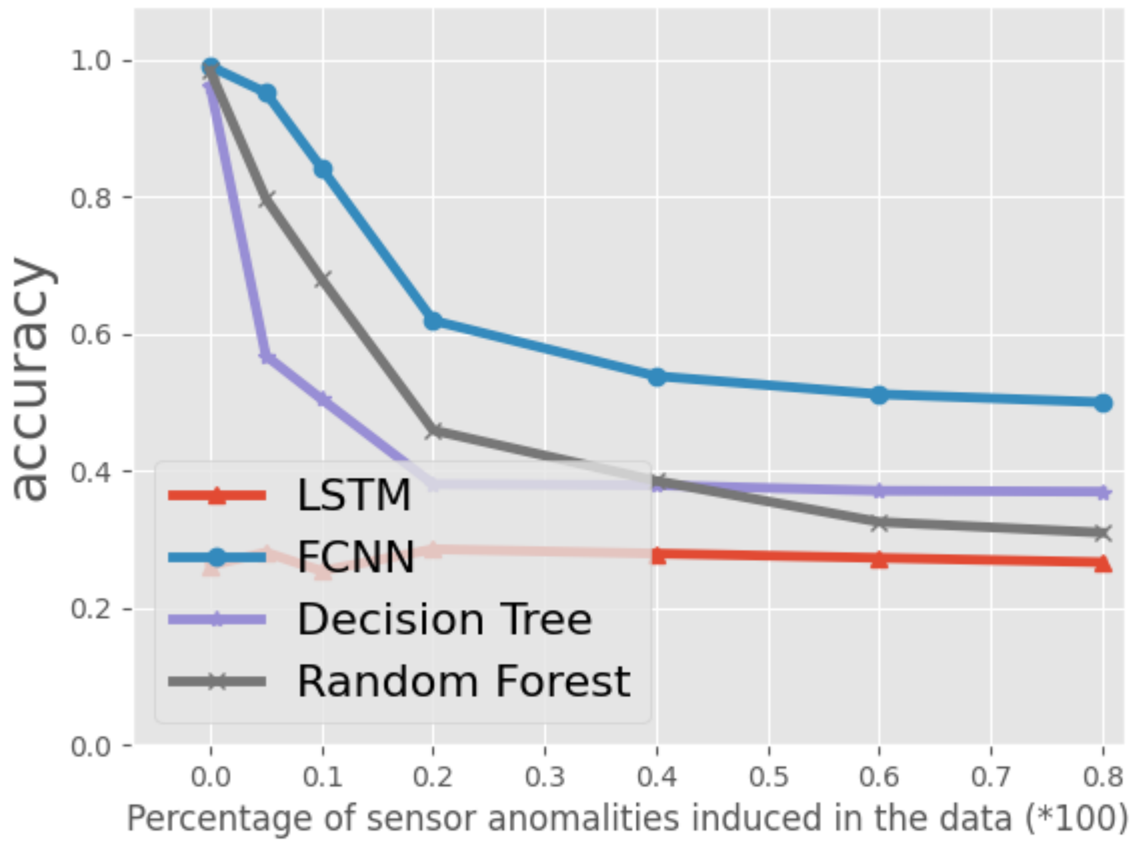


The accuracy of the four models trained on the new dataset is very similar to the ones trained on the original dataset. The LSTM and CNN are a bit lower while the DT has a slightly higher accuracy but overall the changes are minor and the ratios are very similar.

New code variant

As part of the reproducibility assignment, a new code variant has been developed. The existing code by Abenezer Girma et.al. is written with the TenserFlow framework. We wanted to see if we could achieve the same results or even better with shorter runtime, with another machine learning framework. To do this, the TensorFlow model is completely rewritten in PyTorch. The PyTorch framework is the standard in the Deep Learning course, that is why this framework has been chosen as the new code variant.

The new PyTorch model is able to train with the vehicular telematics data and can evaluate results on the testset. However, the model gets stuck in training after the first 10 epochs, not being able to decrease the loss any further. It could be that the PyTorch LSTM layers are different from the TensorFlow layers, or that we suffer from the vanishing gradient problem after all. It could also be because of a flaw in the way the training loop is done, which is a much more difficult task in PyTorch compared to TensorFlow.The resulting accuracy curve is terrible, as is to be expected (see figure below). We were not able to improve on these results within reasonable time.



```
class LSTMmodel(torch.nn.Module):
   def __init__(self):
       super(LSTMmodel, self).__init__()
       self.lstm1 = nn.LSTM(hidden_size=160, input_size=NUM_FEATURES)
       self.batchNorm1 = nn.LazyBatchNorm1d()
       self.dropout1 = nn.Dropout(p=0.2)
       self.lstm2 = nn.LSTM(hidden_size=120, input_size=160)
       self.batchNorm2 = nn.BatchNorm1d(num_features=16)
       self.dropout2 = nn.Dropout(p=0.2)
       self.linear1 = nn.LazyLinear(NUM_CLASSES)
       self.softmax = nn.Softmax(dim=1)
```

Discussion

General discussion

Apart from slightly differing results as compared to the paper there are several reasons which made it difficult to reproduce their results in the first place. Most of these are a result of ambiguity and a lack of clearness in the provided code on their github or in their description in the paper. First of all their python file is built to show accuracy vs anomalies and it is unclear if they have used a similar approach for the sensor noise which we needed to produce.

Furthermore it is stated in the paper that they are using the Kia dataset, which has 10 drivers and 52 features. However, in their python file the LSTM model works with only 21 input features and 4 outputs (drivers) and they don’t specify which part of the dataset they have used to get the desired results shown in the paper.

In order to still be able to create similar results we had to build our own LSTM model with a similar structure as their pretrained model. Unfortunately it is never stated how they trained their LSTM model so loss function and number of epochs was something we had to make an educated guess about. This also means that it was unclear what they used as a loss function so we had to make a guess.

This brings us to the next part which made the replication difficult, the inconsistency in the code. The code below is a part of their python function called “LSTM_anomality”. It takes as inputs X_test_rnn & y_test_rn, does some things which it needs to do and then computes an average test loss and an average test accuracy which are linked to exactly the same variable which cannot be correct. After this they return acc_noise_test and acc_noise_test_rf_box while there is no noise added in this function and its output is also not used for a Random Forest as the rf suggests. These kinds of small naming errors make it difficult to understand what is happening in the code.

```
            score_1 = clean_model.evaluate(X_test_rnn_noise_scaled, y_test_rnn, batc
            iter_score.append(score_1[1])
            # print(score_1[1])

        dif = max(iter_score) - min(iter_score)
        score_2 = sum(iter_score)/len(iter_score)
        acc_noise_test.append(score_2)
        print('Avg Test loss:', score_2)
        print('Avg Test accuracy:', score_2)
        acc_noise_test_rf_box.append(dif)

    return acc_noise_test,acc_noise_test_rf_box
```

Results discussion

In the two figures we were reproducing we did not manage to exactly replicate the results. Starting with the figure where the LSTM and other models are trained on data without noise after which their performance is checked on data with increasing levels of artificial noise. In the paper they show that LSTM is very robust against the increasing levels of noise and keeps an accuracy of 90% while the other three rapidly drop until their accuracy is under 20%. Our result shows a similar trend where the LSTM outperforms the other three models. However, the accuracy of the LSTM drops significantly more while the other three models seem less affected compared to the paper. When we then look at the figure where the models are also trained on noisy data the trend seems to be the same as in the paper with a slight decrease in accuracy for the first free models. However, the Random Forest has a higher accuracy in our results and now outperforms the

Decision Tree.

It is difficult to exactly pinpoint the reason for these differences in the original and the reproduction but we think several factors have played a role. Apart from changing the model structure to correctly use the input features and number of outputs, we also had to write our own implementation to add the noise to the test data for the first figure and to the training data for the second figure. There was no training data normalization or test data normalization after adding the noise in the original code so that had to be implemented by us.

These reasons in combination with the previously stated difficulties like not knowing which data they used exactly resulted in that we were not able to create a one on one match with the papers results.

References

[1]. Girma, A., Yan, X., & Homaifar, A. (2019, November). Driver identification based on vehicle telematics data using lstm-recurrent neural network. In 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 894-902). IEEE. https://ieeexplore.ieee.org/abstract/document/8995202

[2] Byung Il Kwak. (2016). Hacking and countermeasure research lab. Security Driving dataset. https://ocslab.hksecurity.net/Datasets/driving-dataset.

[3] P. Rettore. (2018, January). Vehicular-trace dataset. http://www.prof.rettore.com.br/vehicular-trace/