Activity Recognition for Locomotion and Transportation Dataset Using Deep Learning

Chan Naseeb
R & D
IBM
Munich, Germany
chan.naseeb@ibm.com

Bilal Al Saeedi
R & D
IBM
Munich, Germany
bilal.alsaeedi@de.ibm.com

ABSTRACT

Team "DL_Lock": The Sussex-Huawei Locomotion-Transportation (SHL) recognition challenge 2020 poses a unique opportunity to work on a broad, real-life dataset to classify transport-related activities in a user and location-independent manner. Since deep learning architectures have now received great attention on achieving promising results on time series classification tasks, we focused our experiments on some recent state-of-the-art deep learning architectures such as CNN, Resnet, and InceptionTime. A considerable amount of time was spent on the preprocessing pipeline, which turned out to be a critical phase that impacted most of the results. At the end and after many experiments and hyperparameter tuning, we were able to achieve a 79% F1 score on the validation dataset using InceptionTime architecture. The objective of this paper is to present the technical description of the Machine Learning processing pipeline, the algorithms used, and the results achieved during the development/training phase.

CCS CONCEPTS

• Human-centered computing → Ubiquitous and mobile computing.

KEYWORDS

Activity recognition, locomotion, transportation, time series classification, deep learning

ACM Reference Format:

Chan Naseeb and Bilal Al Saeedi. 2020. Activity Recognition for Locomotion and Transportation Dataset Using Deep Learning. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers (UbiComp/ISWC '20 Adjunct), September 12–16, 2020, Virtual Event, Mexico*, ACM, New York, NY, USA. 6 pages. https://doi.org/10.1145/3410530.3414348

UbiComp/ISWC '20 Adjunct, September 12–16, 2020, Virtual Event, Mexico

ACM ISBN 978-1-4503-8076-8/20/09…$15.00
https://doi.org/10.1145/3410530.3414348

1 INTRODUCTION

Team "DL_Lock": Recognizing human activity is a challenging as well as an interesting problem. In this regard, we wanted to recognize the modes of transportation using the inertial sensor data of a smartphone. Recognizing modes of transportation is a very useful information that can help in many applications such as transportation and parking recommendation systems. For this, we used the Sussex-Huawei Locomotion-Transportation (SHL) dataset which is a public dataset with thorough description [1, 2]. The dataset is intended to be used by researchers for activity recognition, sensor fusion, traffic analysis and many other purposes. The SHL locomotion challenge 2020 [3] provides an excellent opportunity to recognize modes of transportation in a user-independent manner with an unknown smartphone position. For this, we were given inertial sensor data of a single subject where sensors are attached in multiple body positions and expected to generalize to other subjects with unknown smartphone location. What made this challenge very interesting was the fact that besides the hardware and location differences, people also perform the same activities in a distinguish ways. Therefore, it is a hard problem to train a model for a single subject and expect it to generalize easily to other subjects specially if the activities are quite similar like the modes of transportation.

This paper describes how we approached the problem, cleaned the data, built the preprocessing pipelines, and conducted the experiments using multiple deep learning algorithms. At the end of the paper, we will discuss the final selected model and the results achieved against the validation dataset.

2 DATA PROCESSING PIPELINES

A considerable amount of time was spent on data preprocessing. A quick summary of the data used in this challenge is explained below:

- The training data consists of raw sensor data recorded from a single subject called user 1. There is a total of 7 sensor types, namely acceleration, gravity, gyroscope, linear acceleration, magnetometer, orientation, and pressure. All seven sensors have data from 3 axes (x, y, z) except for the orientation sensor, which has a fourth axis w. The sensors are attached to the user in four different locations, namely bag, hips, torso, and hand. The full dataset contains 784288 total frames, each representing 5 seconds worth of data at 100Hz sampling rate, or 500 samples per frame.

- The validation dataset is similar to the training dataset; however, it was recorded from other subjects, namely users 2 and 3. Consistently as the training dataset, the data comes from four different locations. There is a total of 115156 frames, each containing 500 samples.
- During experiments, we have generated an exploration dataset of 30% of both the training and validation sets. Once promising algorithms were found, the experiments were conducted on the full dataset. All the models were trained on only the training set and evaluated against only the validation set.
- In the end, we found the final model architecture that showed promising results and proved to be able to generalize well in a user and location-independent manner. This final selected model was given some part of the validation samples (50% of the given validation dataset) to increase the chances of getting good results in the test set since the model would have seen some samples from users 2 and 3. Finally, the model was evaluated on a held subset from the validation dataset (the other 50% of the given validation dataset) using a k-fold cross-validation technique to reduce overfitting. Doing this has improved the model performance by 5 % since having more samples from other subjects is always beneficial. More details are shown in **Table 1**.

In the following sections, we will explain the pipelines used to preprocess the data before applying any algorithms.

2.1 Data Preparation

We have performed the following activities to prepare the data for any further processing:

- Data were resampled from 100Hz to 50Hz. With 100Hz, the models were more susceptible to overfitting. We experimented with multiple sampling rates, such as 25H and 75Hz. We achieved the best results with a 50Hz sampling rate or 250 samples per frame.
- Data were normalized and standardized either manually using sklearn MinMax scaler [4] or automatically inside the models using batch normalization technique [5].
- An Imputation transformer was used for completing missing values by replacing missing values with the mean along each column. Replacing missing values with zeros also achieved similar improvements to the model stability.

A summary of the data used for training and validation is shown in **Table 1**. Each frame contains 250 samples as explained above.

Table 1: Data used for training and validation.

Training dataset	841866 frames (full training dataset plus 50% of the validation dataset)
Validation dataset	57578 frames (50% of the validation set)
Exploration training dataset	235286 frames (30% of the training set)
Exploration validation dataset	34546 frames (30% of the validation dataset)

2.2 Feature Engineering

For better performance, feature engineering is an important step for activity recognition tasks. Although some deep learning architectures can automatically extract useful features from raw data. We were able to achieve better results by calculating more features and calibrating the existing ones. For example, we have manually calculated the following eight extra features from the existing raw sensor data:

- Five features representing the magnitude ($m = \sqrt{x^2 + y^2 + z^2}$) of the three axes of the acceleration, gravity, gyroscope, linear acceleration, and magnetometer.
- Three features, namely pitch, roll, and yaw calculated from the orientation sensor, represent the Euler angles [6]. The Euler angles are better features because they have more real-world meaning compared to the quaternions. They were calculated from the orientation data quaternions $[q_w, q_x, q_y, q_z]$ as shown below.

$$pitch = arctan\left(\frac{2(q_w q_x + q_y q_z)}{1 - 2(q_x q_x + q_y q_y)}\right)$$
$$roll = arcsin\left(2(q_w q_y - q_z q_x)\right)$$
$$yaw = arctan\left(\frac{2(q_w q_z + q_x q_y)}{1 - 2(q_y q_y + q_z q_z)}\right)$$

Additionally, we used the tsfel library [7] to auto-generate some statistical, spectral, and temporal features. Using this library was too slow using the available computing power shown in **Table 2**; therefore, we tested this only on a small exploration dataset, which showed promising results. For the given reason, these features were not used to produce the final model. A very important step was to re-orient the acceleration, magnetometer, and linear acceleration data from the body frame to the North-East-Down (NED) coordinates [8]. It helped in dealing with the location independent requirement and showed a big performance increase.

Table 2: Available computing power.

Processor	Intel Core i7-7800X, 3.50GHz, 3504 MHz, 6 Cores
Physical Memory (RAM)	32.0 GB
Disk	Samsung SSD 970 EVO 500GB
GPU	NVIDIA GeForce RTX 2080 Ti

2.3 Feature Selection

We performed feature selection by running a random search against multiple configurations to understand the usefulness of certain features combinations. By doing this, we found the following observations:

- Orientation, pressure, and gravity raw sensor data were not so useful but rather caused overfitting.
- Acceleration and linear acceleration data were the most influencing features in the results.
- The extra eight manually calculated features contributed positively to the model performance; however, it was not a big improvement. With these features, we noticed an increase of approximately 3-5 % in the total F1 score against the exploration dataset.

In the end, we decided to use only acceleration, linear acceleration, gyroscope, and magnetometer raw sensor data to produce the final model. The reason why we skipped the manual features was also in favor of using deeper deep learning architectures that produced better performance. We tried to use as minimum features as possible to be able to fit these deeper architectures into the available memory and speed up the training

process to iterate faster. The final selected features are shown in **Table 3**.

Table 3: List of selected features.

Feature	Type	Selected?
linear acceleration (x, y, z)	Raw	Yes
Gyroscope (x, y, z)	Raw	Yes
Magnetometer (x, y, z)	Raw	Yes
Acceleration (x, y, z)	Raw	Yes
Orientation (x, y, z, w)	Raw	No
Gravity (x, y, z)	Raw	No
Pressure	Raw	No
Yaw	Calculated	No
Roll	Calculated	No
Pitch	Calculated	No
Linear acceleration magnitude	Calculated	No
Acceleration magnitude	Calculated	No
Gravity magnitude	Calculated	No
Gyroscope magnitude	Calculated	No
Magnetometer magnitude	Calculated	No

We have also performed the following experiments, but they did not impact the final submitted model:

- We identified and removed any transition frames. Transition frames are any frame that contains more than one label. We did not notice any performance increase after the removal. It is because the number of transition frames were insignificant, around 2300 frames, and had no impact on the results.
- We tried identifying and removing any outliers. We identified this by analyzing outliers within every location and every class. For example, we analyzed the data coming from the bag location and removed the outliers within every activity. We assumed that outliers could happen because people can sometimes perform the same activity in the wrong way. The outliers were detected using a Z-score outlier technique [9]. However, we did not see any major improvements after the removal of the outliers.
- Under the assumption that time-series data usually contains a high degree of correlation, we tried reducing any data correlation between the features. It turned out also not to help increase performance.
- We tried selecting the top features using tree-based algorithms, which can reduce the noise in the data by selecting only the important features. We achieved this by selecting various top features; however, we noticed a performance drop due to this technique.
- We tried multiple auto feature engineering techniques using open source libraries such as tsfel [7], tsfresh [10], and sktime [11]. It was difficult to run these techniques for the full dataset due to the limited access to enough compute power to get the results in a timely manner. Therefore, we tried this on a very small exploration dataset. However, it is difficult to assess the usefulness of adding these extra features without evaluating it on a larger part of the data.

3 ALGORITHM SELECTION

In the beginning, we explored some traditional machine learning algorithms on a small subset of the dataset such as random forest, KNN, a bag of decision trees, SVM, gradient boosting, xboost, and Naïve Bayes. However, due to the following reasons, we focused our efforts on deep learning models:

- With the available compute power, it was difficult to experiment with the full dataset using traditional machine learning algorithms since it was so slow.
- Extracting manual features is very important to get meaningful results with these classifiers, which requires time and domain experience.

Therefore, we run experiments manually in various deep learning architectures that are popular for solving time series classification problems such as CNN, LSTM, ConvLSTM, and CNN-LSTM architectures. The models were manually built using Keras, and a random search was performed against their hyper-parameters. A multi-headed version of all these models was also tested. A multi-headed model consists of multiple heads. Each model head reads the input time steps using a kernel of different sizes. For example, a three-headed model can have three kernel sizes of 3, 5, 11, allowing the model to read and interpret the sequence data at three distinct resolutions. The prediction from all heads are then merged within the model, and then interpreted by a fully connected layer before making a prediction.

Later on, we used an open-source library called mcfly [12], which is a library aimed at facilitating the use of deep learning for time series classifications tasks. The library was used to perform a random search over the hyperparameter spaces for some deep learning classifiers, as will be explained later. The following are some common facts in all the experimented models:

- We used a LeCun Uniform Weight initialization.
- L2 regularization on all convolutional and dense layers is selected.
- A categorical cross-entropy loss is utilized.
- F1 score was the performance measure to select the best model.

Using mcfly, we ran more than 290 experiments based on CNN, InceptionTime, and Resnet architectures. The summary is shown in **Table 4**. The summary of the achieved performance in the training and validation datasets across all the experiments is shown in **Figure 1**.

Table 4: Number of experiments per model type.

InceptionTime	260 experiments
ResNets	20 experiments
CNN	10 experiments

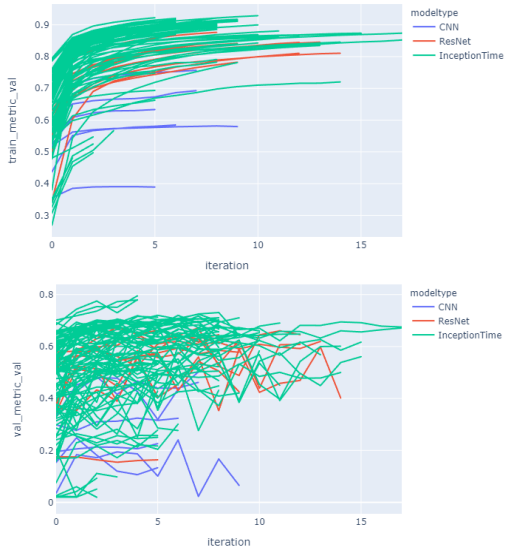

Figure 1: Performance achieved on all the experiments against the training and validation datasets.

3.1 CNN Experiments

We used a convolutional neural network that has been shown to produce state-of-the-art results with little to no feature engineering on complex activity recognition tasks, rather than using feature learning on raw data. This model's hyperparameters include 1) number of Conv layers, 2) the number of filters in each Conv layer, and 3) the number of neurons in the hidden Dense layer. The results were not so promising due to the difficulties in the datasets. Having a subject and location independent model is a challenging task since we train on only a single subject. It makes it hard for the model to generalize. The results are shown in **Figure 2**.

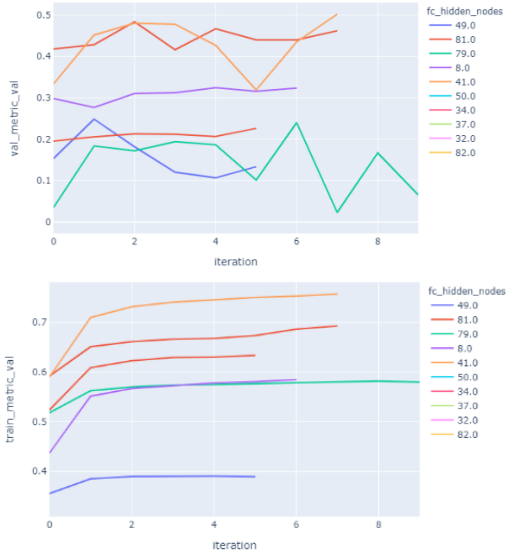

Figure 2: Performance achieved on the CNN models against the training and validation datasets. Each model with different number of nodes for the fully connected layers.

3.2 ResNets Experiments

Residual Networks (ResNets) [13] has been very successful in object detection and other vision-related tasks. ResNets has alleviated the problem of training very deep neural networks by introducing a linear short-cut connection as part of a residual block. The residual connection increases the model's accuracy by solving the vanishing or exploding gradients problem that happens in the very deep layers of a neural network. We evaluated the use of multiple variants of ResNets by changing network depth, the number of filters, and kernel sizes. The network depth represents the total number of residual blocks that are connected, followed by a final softmax layer having a number of neurons equal to the number of activities to be classified. Every residual block consists of a set of convolutions with a configurable number of filters and kernel size. We ran around 20 experiments on the full training and validation datasets. The best model of this type achieved an F1 validation accuracy of 67 %, as can be seen in **Figure 3**.

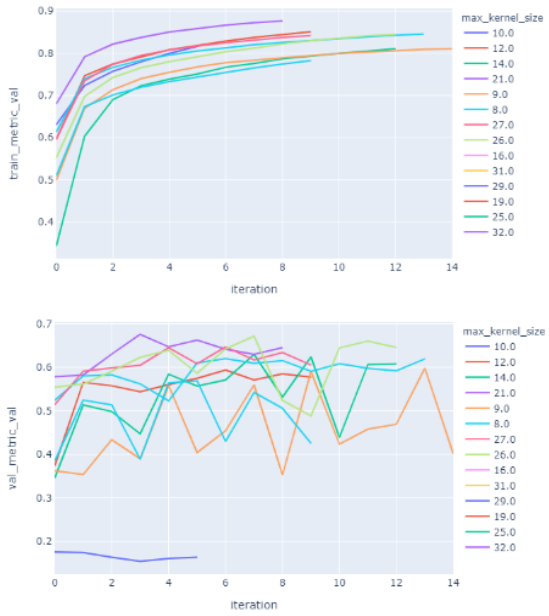

Figure 3: Performance achieved on the ResNets models against the training and validation datasets. Each model with different filter kernel size.

The maximum network depth we were able to achieve with available hardware was 18 layers. We observed an increase in the accuracy with the increase of the network depth, as seen in **Figure 4**.The figure shows a subset of the ResNet experiments and we can see that deeper networks architectures are performing better.

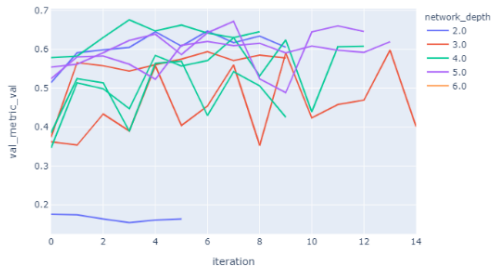

Figure 4: The average performance per network depth using ResNets models.

3.3 InceptionTime Experiments

InceptionTime is a recent model architecture specialized in time-series classification tasks [14]. It is an ensemble of a set of inception networks where each is created by cascading various Inception modules [15]. Inception modules consist of multiple convolutional filters with various lengths that are applied simultaneously to extract features from the input time series data. Every inception network also contains two different residual blocks, and every residual block consists of three inception modules. Every residual block's input has a residual skip connection to the next block's input to mitigate the vanishing or exploding gradients problem that occurs in very deep neural networks. Most of the experiments were conducted using this architecture since it showed promising results from the beginning. We have performed around 260 experiments by playing with the network depth, the number of filters, and the kernel sizes. A summary of a subset of the experiments using InceptionTime is summarized in **Figure 5**.

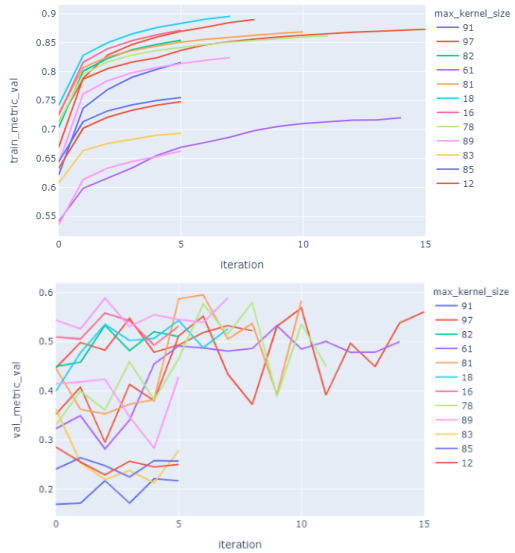

Figure 5: Performance achieved on the InceptionTime models against the training and validation datasets. Each model with different filter kernel size

We tried multiple network depths and observed that deeper layers could improve accuracy, as shown in **Figure 6**. It is shown that the best performance was achieved on layer 26, which is the deepest layer we were able to test using the available compute power.

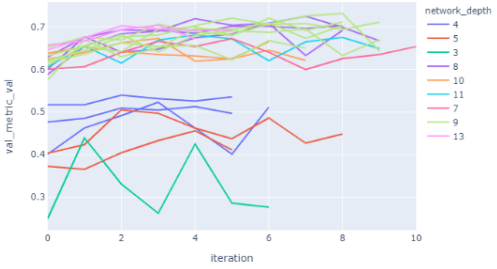

Figure 6: The average performance per network depth using InceptionTime models.

3.4 Final Selected Model

The final model used to predict the test dataset is based on InceptionTime and has the parameters in **Table 5**. **Figure 7** shows an overview of the complete process including data preprocessing, training, and testing phases. The model architecture of the final model is also shown in **Figure 7**. The model consists of a total of 3 inception blocks or ensembles each contains 3 inception modules and a final dense layer with a softmax activation. For more details about the InceptionTime architecture concepts, please refer to section 3.3. It is shown from the results that the residual skip connections helped in achieving better accuracy with deeper layers.

Table 5: The final model hyperparameters.

Learning rate	0.00057
Regularization rate	0.0135
Network depth	10
Filter number	33
Kernel size	97

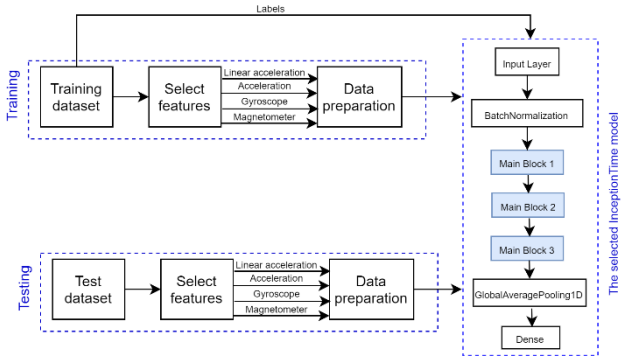

Figure 7: An overview of the complete process including data preprocessing, training, and testing phases.

We were able to achieve the results shown in **Table 6** against the validation dataset. The results also showed that the model extracted features aren't enough to distinguish between similar activities such as recognizing a user being in a train or being in a subway, as shown in **Figure 8**.

Table 6: The final model results against validation dataset.

Accuracy	0.7935
Precision	0.8304
Recall	0.7615
F1 score	0.7939

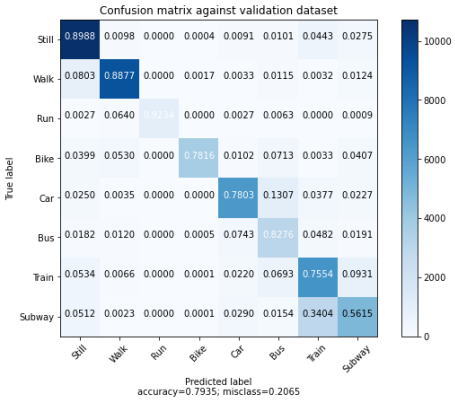

Figure 8: The confusion matrix of the final model according to the validation dataset results.

4 DISCUSSION

- Feature extraction is a hard problem because of the inter-activity similarities; different activities might have similar characteristics, which makes it difficult to extract distinguishable features.
- Activities are person dependent; every person has his way of performing the same activity. It makes it difficult to generalize to multiple users if the training involves only a single subject. Besides the hardware and location differences, people also perform activities differently. It is a hard problem to train a model in a single subject and expect it to generalize.
- Having deep models with more layers helped in extracting and learning useful features and showed performance improvements. However, there is a limit on how much deeper we can try due to data and computer power limitations. The available compute power is shown in **Table 2**. For example, the maximum network depth, we were able to test using the InceptionTime architecture, was 10 before we got out of memory errors.
- Getting access to more compute power to do more experiments and iterate faster is very beneficial. Due to the size of the dataset, most experiments took a lot of time with the available hardware especially the auto feature generation using frameworks such as tsfel, tsfresh, and sktime. We believe adding the auto-generated features for the full dataset would have improved the performance.

5 CONCLUSION

The SHL locomotion challenge provides a unique chance and a very interesting dataset for locomotion activity recognition. In addition, recognizing locomotion activities in a user and location independent manner is very challenging. Our approach was to focus on some state-of-the-art deep learning models which proved to show promising results in the activity recognition and times series classification problems. We mostly experimented with CNN, ResNets and InceptionTime architectures by trying multiple hyperparameters such as the network depths, the number of neurons of dense layers, convolutional filter numbers and kernel sizes. The best model is based on the InceptionTime architecture and achieved 79% F1 score on the validation set. This model generalized well regardless of the user or the smartphone location. It was shown also that data preparation, and preprocessing is a critical part to increase the model performance. The recognition result for the testing dataset will be presented in the summary paper of the challenge [16].

REFERENCES

[1] L. Wang, H. Gjoreski, M. Ciliberto, S. Mekki, S. Valentin, and D. Roggen, "Enabling reproducible research in sensor-based transportation mode recognition with the Sussex-Huawei dataset," IEEE Access 7 (2019): 10870-10891.

[2] H. Gjoreski, M. Ciliberto, L. Wang, F.J.O. Morales, S. Mekki, S. Valentin, and D. Roggen, "The university of sussex-huawei locomotion and transportation dataset for multimodal analytics with mobile devices," IEEE Access 6 (2018): 42592-42604.

[3] "Sussex-Huawei Locomotion Challenge.," [Online]. Available: http://www.shl-dataset.org/activity-recognition-challenge-2020/.

[4] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[5] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[6] Wikipedia contributors. (2020, June 26). Euler angles. In Wikipedia, The Free Encyclopedia. Retrieved 08:14, June 29, 2020, from https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=964664972.

[7] Barandas, Marília and Folgado, Duarte, et al. "TSFEL: Time Series Feature Extraction Library." SoftwareX 11, 2020. https://doi.org/10.1016/j.softx.2020.100456.

[8] Wikipedia contributors. (2020, June 11). Local tangent plane coordinates. In Wikipedia, The Free Encyclopedia. Retrieved 08:21, June 29, 2020, from https://en.wikipedia.org/w/index.php?title=Local_tangent_plane_coordinates&oldid=961980629.

[9] Ronald E. Shiffler (1988) Maximum Z Scores and Outliers, The American Statistician, 42:1, 79-80, DOI: 10.1080/00031305.1988.10475530.

[10] Christ, Maximilian, et al. "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)." Neurocomputing 307 (2018): 72-77.

[11] Löning, Markus, et al. "sktime: A unified interface for machine learning with time series." arXiv preprint arXiv:1909.07872 (2019).

[12] Dafne van Kuppevelt, Christiaan Meijer, Vincent van Hees, Mateusz Kuzak "mcfly" 10.5281/zenodo.495345 (2017).

[13] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," arXiv: Computer Vision and Pattern Recognition, vol., no., p., 2016.

[14] Fawaz, Hassan Ismail, et al. "Inceptiontime: Finding alexnet for time series classification." arXiv preprint arXiv:1909.04939 (2019).

[15] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9.

[16] L. Wang, H. Gjoreski, M. Ciliberto, P. Lago, K. Murao, T. Okita, and D. Roggen. "Summary of the Sussex-Huawei locomotion-transportation recognition challenge 2020", Proceedings of the 2020 ACM International Joint Conference and 2020 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, 2020.