Multi-task Learning for Transportation Mode Classification and Driver Identification

Esosa Orumwese
Data Science and Analytics
University of Exeter
Exeter, United Kingdom
730XXXXXX

***Abstract*—This study presents a multi-task deep learning approach for simultaneous transportation mode classification and driver identification using smartphone sensor data. The proposed model integrates a bidirectional LSTM for transport mode classification and a ResNet50-GRU hybrid for driver identification, leveraging accelerometer and gyroscope data. Experiments demonstrate the model's effectiveness, achieving F1 scores exceeding 0.75 for both tasks across all classes. The results suggest that multi-task learning not only consolidates the performance of individual tasks but also provides a robust, scalable solution for real-world applications such as usage-based insurance. The findings are discussed in the context of the model's potential for broader adoption in the transportation and insurance industries.**

***Index Terms*—multi-task learning, transportation mode classification, driver identification, deep learning, smartphone sensors, usage-based insurance.**

I. Introduction

Monitoring driving behaviour has an increasing application in various fields. One of such fields, as is the focus of this research, is the car insurance market. Using schemes like Usage-Based Insurance (UBI) or Pay-As-You-Drive, insurers attempt to match the premium offered to each driver according to their driving habits.

Different mechanisms have been adopted in order to acquire the driver's driving data. Most have focused on telematic systems [1], [2] which need to fixed in the users cars, in order to log different sensing variables and driving events. Despite being effective, these solutions have faced several challenges. Firstly these boxes incur an overhead cost for setup and also maintenance on the part of the insurance companies. Secondly most drivers are wary of being monitored seeing as they have no control over the devices [3].

These concerns shifted the attention of study to the use of smartphones for driver behaviour monitoring as they come equipped with sensors such as accelerometers, magnetometers, GPS, etc. Smartphones address the concerns noted above as there is no cost attached for each user and the drivers are familiar with their phones and have control over the data used [3].

When it comes to monitoring driver behaviour using smartphones, the first problem lies in knowing when exactly to log data and this is where my research lies. Given that most individuals carry with their phones on them as they go about their day, and it would too much of a hassle to manually turn on the data logger whenever the user is driving, we would need to know when exactly to log sensor data. This means figuring out, based on the sensor data received, when the user is in a car as opposed to other modes of transportation and when in a car, when the user is the driver.

This leads us to the problems covered in this research; transport mode classification and driver identification. The objective of travel mode classification is to discover users' modes of travel (walking, driving, cycling, bus, etc) by analyzing their mobility data [4]. While the driver identification is to recognize a driver within a given population also based on their unique mobility data [5].

Previous studies have applied various techniques like random forests [6], multi-task deep learning [4], [7], support vector machines [8], [9], and neural networks [6], [10] to classify transportation modes, driver identification has predominantly relied on neural networks trained on acceleration data [5], [11], [12]. These approaches, however, and the current body of work in this area, treat these issues separately with no solution to solve these problems simultaneously.

In this research, we will be consolidating already existing methodologies in academia which have proven effective in solving each task separately using data from the accelerometer and gyroscope. For the transport mode classification branch, we will be using the deep bidirectional long short-term (biLSTM) neural network proposed in [10]. To classify data into six transport modes, they extracted sequences made up of 128 vectors with 50% overlap of each of the triaxial measurements for the accelerometer and the gyroscope. These extracted features were then fed into a biLSTM network with 2 hidden layers and trained with a cross entropy loss function.

For the driver identification branch, we will be using the ResNet50-GRU model proposed in [5]. In their work, they suggested transforming the triaxial accelerometer and gyroscope measurements into longitudinal acceleration, traversal acceleration and angular velocity before mapping those 1D signals into 2D images using three simple convolution neural network trained on each of the transformed signals. Using the extracted feature maps, a pretrained ResNet-50 model, which had its last layer replaced with gated recurrent unit layer and a SoftMax layer, was trained to identify each driver.

The remainder of this study is organized as follow. In

Section II, the background is presented. Section III motivates this research project by stating the aims and objectives. Section IV establishes the methodology chosen for this project, the neural network architectures, evaluation metrics as well as experimentation configuration. Section V presents the experimental result and discusses it. Section VI concludes this study.

II. Background

A. User-Based Insurance (UBI)

The advent of big data has significantly impacted the insurance industry, particularly in the development of Usage-Based Insurance models. S. Arumugam and R. Bhargavi conducted a detailed survey on the Manage-How-You-Drive (MHYD) categories, aiming to refine how insurance companies assess driving risk [14]. Their study categorizes MHYD into driving pattern monitoring, distracted monitoring, fatigue monitoring, and drowsiness monitoring, with a focus predominantly on the first category. They propose a three-phase methodology to identify aggressive drivers and calculate personalized premiums, focusing on risk prevention.

In their contribution to driver behavior profiling using smartphones, G. Castignani, T. Derrmann, R. Frank, and T. Engel explore how smartphone sensors—like accelerometers, gravity sensors, and magnetometers—can detect risky driving maneuvers [3]. Their platform, SenseFleet, not only counts risky events but also incorporates contextual information such as weather conditions and time of day.The result of SenseFLeet was found to the equivalent to individual drivers' subjective feedback in around 90% of the cases within ± 1 neighboring driving clusters.

B. Transportation Mode Classification

Advancements in transportation mode classification are significantly influenced by the innovative use of smartphone sensors. M.A. Shafique and E. Hato utilized smartphone accelerometers and gyroscopes to extract nine features for classification; resultant acceleration, standard deviation, skewness, kurtosis, maximum resultant acceleration, average result acceleration and maximum average resultant acceleration all derived by using a moving window on the resultant acceleration [6]. Pitch and roll were obtained from orientation sensor with the GPS data reserved for validation. Implementing a Random Forest to classify the data among six modes, and varying variables such as data collection frequency, moving window size and proportion of training data they achieved an overall accuracy of 99.6%.

Further addressing the limitations associated with sensor data, particularly in urban settings where GPS reliability is compromised, H. Zhao, C. Hou, H. Alrobassy, and X. Zeng proposed a novel approach using a deep learning framework. Their method utilized a deep bidirectional long-term short-term memory (Bi-LSTM) network that records signals using the accelerometer and the gyroscope sensor rather than GPS [10]. This approach not only enhanced the detection of various transportation states but also outperformed other variants of recurrent neural networks (RNNs), with the Bi-LSTM architecture providing superior retention and utilization of information across different states. This method achieved a transportation activity classification accuracy of up to 92.8%.

Exploring the capabilities of multi-task learning, X. Song, C. Markos, and J.J.Q. Yu developed MultiMix, a framework that optimizes the use of labeled, unlabeled, and synthetic data [4]. This framework is distinctive as it simultaneously trains a labeled data classifier, an unlabeled data classifier, and an autoencoder, integrating multiple learning tasks to enhance the efficiency and effectiveness of travel mode identification. This method demonstrated exceptional performance, especially when compared to previous models, achieving an accuracy of 66.2% with just 1% of the labeled data and an impressive 84.8% when all available labels were used. Their findings were further substantiated by an ablation study, which highlighted the critical nature of each component in the model, as removing any one of them led to decreased performance.

C. Driver Identification

Innovations in driver identification have progressively centered on utilizing smartphone sensors to minimize invasiveness while maximizing accuracy. Reflecting on these advancements, S.H. Sánchez, R.F. Pozo, and L.A.H. Gómez explored a novel approach where the driver's smartphone, in a free position within the vehicle, is the sole sensor source [11]. Utilizing triaxial accelerometer signals and gyroscope signals to derive longitudinal, transversal, and angular velocity signals [13], they employed deep neural networks like ResNet-50 and Recurrent Neural Networks (RNN) to transform these time-series data into image formats for processing. Their methodology achieved accuracies of 69.92% and 90.31% at top-1 and top-5 levels respectively across a group of 25 drivers, showcasing superior performance over traditional methods that rely on additional signals like GPS or vehicle-integrated systems.

Advancing this approach, the same researchers improved the performance of their previous methodology by integrating a ResNet-50 architecture with a gated recurrent unit (GRU) network comprising two stacked layers followed by a classifier block, consisting of a Fully Connected layer and a Softmax layer [5]. This architecture was further refined by evaluating two image transformation techniques from accelerometer signals: spectrogram representations and direct inputs into a convolutional neural network (CNN) for a 2D feature map extraction. The latter technique marked a significant improvement in the system's accuracy, achieving 71.89% and 92.02% at top-1 and top-5 driver identification levels respectively, demonstrating the effectiveness of deep learning in refining feature extraction and classification processes.

Expanding the scope of sensor utilization for driver identification, J. Sasan, G. Castignani, and T. Engel took a different route by employing GPS data from smartphones [12]. Their methodology paralleled techniques used in natural language processing, particularly word embedding, to interpret each location data-point within a trip contextually. This approach allowed for the construction of a complex set of continuous, categorical, and sequential features to depict entire trips. By employing a deep learning framework consisting of embedding and recurrent neural network layers, they achieved a nuanced analysis of driver patterns. This method significantly outperformed traditional models such as LightGBM and Hidden Markov Models (HMM), with an overall error-rate of 1.9%, 3.7%, 5.71%, 9.57%, 13.5% for groups of 5, 10, 20, 50, 100 drivers, demonstrating its scalability and robustness.

D. Deep Learning

Deep learning, unlike traditional machine learning methods, doesn't depend on the engineer's domain expertise to design and build a feature extractor for raw input data. It belongs to a class of methods called representation learning, which allows a machine to automatically find the necessary representations for detection or classification from raw data. Deep learning identifies complex patterns in large datasets using the backpropagation algorithm to adjust its internal parameters, determining how to compute each layer's representation from the previous layer.

In this work, we focus on Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs are designed to process data in multiple arrays, like a color image with three 2D arrays of pixel intensities for each color channel. Many data types come in multiple arrays: 1D for signals and sequences, 2D for images or audio spectrograms, and 3D for video or volumetric images.

RNNs handle tasks involving sequential inputs like speech and language. They process input sequences one element at a time, maintaining a state vector in their hidden units that contains information about all previous elements. Thanks to advancements in architecture and training methods, RNNs excel at predicting the next character in a text or the next word in a sequence. They are also used for complex tasks such as natural language processing and speech recognition. However, training regular RNNs can be challenging due to exploding or vanishing gradients, which occur when gradients grow or shrink at each time step [14].

E. Transfer Learning and ResNet-50

Training networks from scratch is complex and requires a large amount of data, according to the depth of the network. This is where transfer learning comes in. It enables one to make use the weights of pre-trained models in solving their given problem. For this work, we made use of the pretrained ResNet-50 model in our driver identification branch as done in [5]. Its name comes from the Deep Residual Network and specifically ResNet-50 refers to the fact that it presents 50 residual layers.

Deep Residual Networks, introduced by Microsoft Research during the 2015 ImageNet and COCO competitions, have significantly advanced image classification, object detection, and semantic segmentation. These networks were developed to address the challenges of training deep neural networks, as adding more layers to an already deep model often leads to higher training errors. ResNet simplifies training and optimization by using residual learning frameworks with shortcut connections.

F. LSTMs, BiLSTMs and GRUs

As mentioned earlier, it has been observed that training RNNs to capture long-term dependencies prove difficult due to exploding or vanishing gradients during backpropagation. This can be solved via gradient clipping or through the design of a more sophisticated activation function than the usual activation. This led to the introduction of the long short-term memory unit [15] and the gated recurrent unit [16].

Long Short Term Memory units (LSTMs) enhance RNNs by enabling them to maintain and utilize long-term dependencies in input data. This allows LSTMs to manage memory effectively by reading, writing, and erasing data as needed. These memory cells are called "gated" cells because gates decide whether to store or discard information. During training, LSTMs learn which information to keep or discard, making them effective for tasks involving long-term dependencies. An LSTM model has three types of gates: forget, input, and output [17].

The Bi-directional Long Short-Term Memory (BiLSTM) is an extension of the standard LSTM model, designed to process input sequences using two LSTM layers. In this approach, one LSTM layer processes the input sequence in a forward direction, while a second LSTM layer processes the input sequence in reverse. This bidirectional processing enhances the model's ability to capture long-term dependencies and improves overall accuracy by providing context from both past and future states of the sequence [19].

Gated Recurrent Units (GRUs), proposed by Cho et al. [20], offer a simplified architecture compared to LSTMs. GRUs merge the forget and input gates into a single gate called the update gate and combine the cell state and hidden state, resulting in a more streamlined model with fewer parameters. This reduction in parameters leads to faster convergence during training without sacrificing accuracy, making GRUs a popular choice in practical applications.

G. Multitask Learning

Multitask Learning (MTL) aims to learn multiple related tasks simultaneously, sharing learned features to improve accuracy and performance. MTL has been applied across various machine learning fields, including vision, speech, natural language processing, and reinforcement learning. The primary benefits of MTL are reduced parameters compared to training separate models for each task and the ability to uncover common structures, enhancing single-task performance with less data per task [21].

Despite its advantages, MTL presents challenges, particularly in determining 'what to share' and 'how to share' among tasks. Effective information sharing and appropriate weighting strategies are crucial to avoid conflicting gradients and optimize the total empirical loss without compromising the learning of an individual task [22]. Common loss weighting methods include uniform combination, dynamic weight average (DWA), and uncertainty-based weighting [22]. These methods help balance the contributions of each task to the overall model training. For this paper, a uniform combination of losses was applied.

Various architectures can be used to implement multitask learning in deep learning. Typically, these approaches learn a general-purpose representation shared among tasks while keeping separate output layers for each specific task [22]. This report focuses on a slightly different architecture due to the unique data requirements for both branches. In this model, only the middle layers are shared, while the lower and upper layers remain separate. During backpropagation of the uniformly weighted loss, the separate upper branches contribute gradients that influence the shared middle layers. These shared gradients then propagate back to the separate lower layers, ensuring that the lower layers indirectly benefit from the combined information of both tasks, despite the upper branches being separate.

III. Aims & Objectives

A. Aim

The primary goal of this project is to develop a multi-task deep learning model that can simultaneously perform transportation mode classification and driver identification using only sensor data from smartphones (specifically accelerometer and gyroscope triaxial data with aid from the rotation vector sensor) with the aim of achieving an F1 score of ¿0.75, which signifies a good model, for all classes of both tasks.

B. Scope

This project focuses more on the analytical capabilities of the model than on its application in real-world scenarios. Specifically, it does not aim to classify driver behavior explicitly nor does it include the full software implementation of the proposed methodology into an application. In this work, we assume that when the user is in a car, he is driving i.e. we do not differentiate between a passenger and a driver.

C. Objectives

1) Sub-trip Classification Accuracy: Enhance the model's ability to distinguish whether the smartphone user is driving a car or using another mode of transport during various subtrips within a single journey. This objective targets the nuanced detection of user status in mixed-transport scenarios.

2) Robustness to Change in Phone Orientation: Build a model that remains effective despite multiple positions the smartphone is placed in during travel. This design aims to account of the change in phone orientation by making use of the rotation vector sensor to normalize all signals received.

3) Efficiency and Prediction Time: Develop a methodology that not only achieves high accuracy but also operates with low prediction time and energy efficiency. These features are crucial for the feasibility of deploying such models in energy-constrained devices like smartphones by making sure it can maintain a high accuracy with low prediction given a low data sampling rate.

IV. Experiment Design & Methods

The primary objective of this experiment is to develop a robust multi-task learning model capable of simultaneously performing transport mode classification and driver identification using features extracted from the triaxial accelerometer and gyroscope data.

A. Dataset Description

This project made use of a subset of the University of Sussex-Huawei Locomotion-Transportation (SHL) Dataset [23], [24], a comprehensive collection of locomotion and transportation modes recorded by mobile users in real-life settings across the UK. Over a period of seven months in 2017, three participants captured 750 hours of labeled data encompassing eight transportation modes including car, bus, train, subway, walk, run, bike, and standing still, using four Huawei Mate 9 smartphones each placed at different locations; hand, hip, bag and torso as shown in Figure 1.

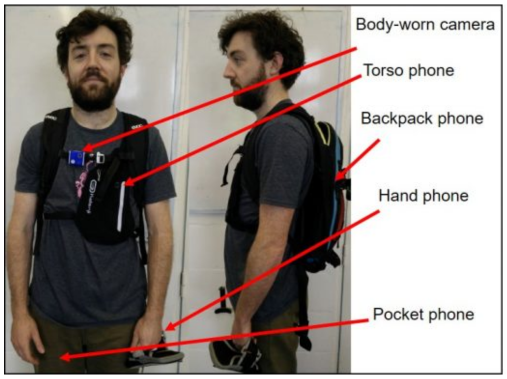

Fig. 1: Sussex-Huawei Locomotion-Transportation (SHL) Dataset phone placements [23].

Through a specialized android app, Datalogger [25], the data from each phone at the 4 different positions are collected, synchronized and logged. The eventual dataset contains data from all sensors of the 4 synchronized phones. Given that the SHL dataset is released in batches for an annual competition, with only select parts made publicly available each year, only the preview dataset was used for this project [26]. This preview subset, contains data for all users at all four phone locations but for just three recording days corresponding to 227 hours of data. Of all the available sensors, the only sensors of interest to this project are the; accelerometer sensor, gyroscope sensor, and rotation vector sensor.

The accelerometer and gyroscope sensors are hardware-based and make use of the sensor coordinate system while the rotation vector sensor is software-based and makes use of a reference coordinate system that is relative to the world's frame of reference [27]-[29]. The sensor coordinate system is defined relative to the device's screen when the device is held in its default orientation as seen in Figure 2. When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. It is important to note that the sensor's coordinate system never changes as the device moves.

As for the rotation vector sensor, its coordinate system defined relative to the Earth as seen in Figure 2. The X-axis is tangential to the ground at the device's current location and points approximately East. The Y-axis is tangential to the ground at the device's current location and points toward the geomagnetic North Pole. And the Z-axis is perpendicular to the ground plane and points toward the sky.

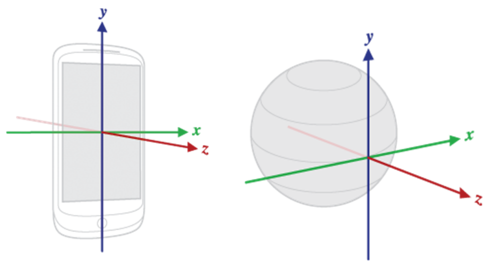

Fig. 2: Sensor coordinate system [28] (left), World coordinate system [29] (right).

The acceleration sensor measures the acceleration applied to the device in m/s^2, including the force of gravity, with respect to the device's X, Y and Z axis [30]. The gyroscope measures the rate of rotation in rad/s around a device's X, Y, and Z axis and assumes rotation as positive in the clockwise direction [31]. The rotation vector sensor represents the device's orientation as a combination of an angle and an axis of rotation, providing a vector whose direction is the axis of rotation and whose magnitude is equal to the sine of half the angle of rotation. The components of this rotation vector are expressed as a quaternion, which encodes 3D rotations mathematically and this is the representation used in this dataset. In this representation, the elements of the rotation vector correspond to the last three components of a unit quaternion: $\cos(\frac{\theta}{2})$, $x{\cdot}\sin(\frac{\theta}{2})$, $y{\cdot}\sin(\frac{\theta}{2})$, and $z{\cdot}\sin(\frac{\theta}{2})$ [29], where θ is the rotation angle and x, y, z are the components of the unit vector representing the axis of rotation.

B. Dataset Processing

Due to the large sampling rate, 100Hz, of the data, we decided to down sample it to 5Hz by only selecting every 20th row. This enabled us to not only achieve our objective of building an effective model on a low data sampling rate but to greatly reduce the amount of training data from roughly 81,938,560 relevant records to 4,096,928 records. Seeing as the data for each user at each day represented the total trip throughout that day i.e. the user could have potentially used different modes of transportation throughout the day, we had to identify the different journey segments for each transport mode each day. This was done so to capture the uniqueness of each trip for a better model prediction.

There are countless methods for journey segmentation in research, however, we decided to implement a simple definition for our journey. In this project, a journey is defined as the segment of data, for a given transport mode for a given day, where there is no change in transport mode. A typical trip for a user in a day could look like; Walking, Driving, Walking, Driving, Cycling, Bus, Walking, Bus, and Walking. However, this journey segmentation yielded around 5 journey segments (although capturing a significant amount of data), and as our driver identification model involved looking at the data for each driving journey, we needed to be able to create "false" segments which we referred to as sub-segments. This was done so as to provide our model with enough journey segments to learn the user's driving pattern from.

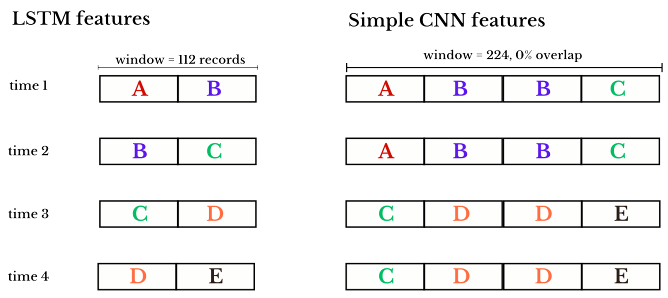

Fig. 3: Windowed sub-segments which we used as features for BiLSTM and for our CNN-GRU network.

To create these sub-segments, we borrowed a window method which accounted for data sampling rate from [10]. Their method using a sliding fixed window of 2.56secs and 50% overlap for each segment as this would be able to sufficiently capture the temporal relationship in each segment fully whilst reducing noise. However, this applied on our 5Hz data would have resulted in a window covering only 12 data points which isn't sufficient. In order to capture more information, we went with a window of 112 data points (22.4secs) with 50% overlap. This window size was chosen because it allowed us to easily create features for the transport classification model (this needed 112 data points per sequence) and the driver identification model which need 224 data points to be fed into the simple CNN. It also allowed for an easy data synchronization of the multitask learning model when both branches, although requiring different data formats looked at the same data-time chunk, when it was joined.

C. Feature Extraction & Representation

1) Transport Classification: For transport mode classification, we decided to implement the BiLSTM model suggested in [10] and as a result we also used the same feature representation suggested but with a new window length. The LSTM was chosen due to its ability to account for temporal dependence between inputs in a sequence and in order to increase the available information to the model, the bi-directional LSTM was chosen. A sample input vector or sequence looked a specific window length, in our case 112 data points, for each feature vector i.e. the triaxial accelerometer and gyroscope readings. The target variable was the transport mode. This resulted in 72,256 features.

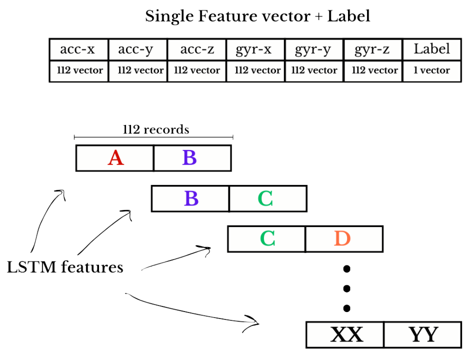

Fig. 4: BiLSTM features gotten from our sub-segments.

2) Driver Identification: For driver identification, we went with the ResNet-50 pretrained model where the last few layers were replaced with a GRU unit with 2 hidden layers and fully connected layer as suggested in [5]. Unlike the feature extraction for the BiLSTM model, this was a bit complicated as the triaxial data for both the accelerometer and the gyroscope needed to be transformed to just 3 signals which could then be mapped into 2D images of size 224x224 with 3 channels representing the signals, so that it can be fed as input to the ResNet-50 model. A CNN-RNN combination was used because as proven in their work, the CNN acted as an excellent feature extractor with the pretrained ResNet-50 CNN being robust to noisy data and the RNN was excellent as modelling dynamic and temporal information. The GRU was chosen here due to its powerful and efficient recurrent dynamic modelling capabilities.

1) Transforming the data:
As discussed above, feature extraction was done in two stages. The first was converting the triaxial accelerometer data into longitudinal and transversal acceleration, and the triaxial gyroscope data into angular velocity. This is an important step because the data from the accelerometer and the gyroscope were gotten with respect to the device's coordinate system which does not take the rotation or placement of the smartphone into account. In order to cancel out this variability, we need to transform the data to match the Earth's coordinate system using data from the rotation vector sensor which gives us corresponding quaternions for each datapoint. Using the methodology described in [32], we derived the Euler angles (roll, pitch and yaw) from the quaternions using equations (1)-(3).

$$\varphi = \arctan\left(\frac{2(q_1q_0 + q_2q_3)}{q_0^2 - q_1^2 - q_2^2 + q_3^2}\right) \tag{1}$$

$$\theta = -\arcsin(2(q_1q_3 - q_2q_0)) \tag{2}$$

$$\psi = \arctan\left(\frac{2(q_3q_0 + q_1q_2)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \tag{3}$$

Also, as suggested in [33], for each sub-segment of 112 data points, we split it into halves, resulting in 2 non-overlapping windows of 11.2 seconds and took the median of the Euler angles so as to account for noise. Using the median Euler angles and for each split sub-window, we derived the rotation matrix of ZYX notation [32]. The x and y values of the acceleration data in each sub-window was multiplied by the rotation matrices, equation (5) resulting in the longitudinal and transverse acceleration, equation (4). The angular velocity was gotten by deriving the square root of the sum of the squared triaxial gyroscope measurements, equation (6).

$$\begin{pmatrix} a_{long.} \\ a_{trans.} \\ a_{vert.} \end{pmatrix} = R \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \tag{4}$$

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{5}$$

where;

$$\begin{aligned}
&R_{11} = \cos\theta\cos\psi;\ \ R_{12} = \cos\theta\sin\psi;\ \ R_{13} = -\sin\theta \\
&R_{21} = \sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi \\
&R_{22} = \sin\varphi\sin\theta\sin\psi + \cos\varphi\cos\psi \\
&R_{23} = \sin\varphi\cos\theta \\
&R_{31} = \cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi \\
&R_{32} = \cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi \\
&R_{33} = \cos\varphi\cos\theta
\end{aligned}$$

$$v_{ang.} = \sqrt{g_x^2 + g_y^2 + g_z^2} \tag{6}$$

2) Mapping 1D signals to 2D images:
The next stage of the feature extraction was converting these signals into 224x224 images and this was done using a simple CNN. The basic idea was that any convolutional layer in a CNN architecture can be considered as a feature map extractor where the feature map is dependent on the purpose for which the model was trained. As such, a model was trained for driver identification for each signal. This simple CNN was made up of 2 Convolutional layers and a fully connected layer. The first convolutional layer performed 1D convolutions on the raw longitudinal and transversal accelerations and angular velocity signals which was fed in as windows of 224 datapoints with 50% overlap. The second convolutional layer performed 2D convolutions on the resulting 224 feature maps with vectors of size 224.

Before training these models, the dataset was split in training and test sets. To evaluate the performance of these three models, we performed driver identification on the test set using late fusion of their outputs (i.e. averaging the SoftMax outputs) resulting in a driver identification accuracy of 67.56%. After this, the three signals for the entire dataset was converted into windows of 224 data points with 50% overlap and fed into its respective model.

For each entry and for each signal, the feature map was collected and joined together to form an image of shape 224x224 with 3 channels representing the three signals.

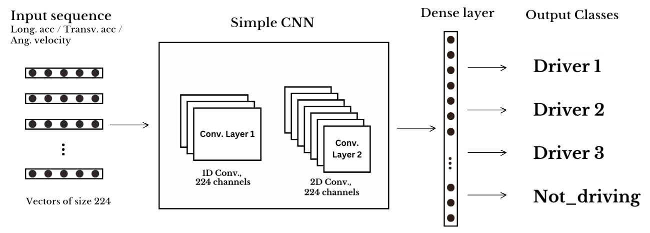

Fig. 5: Simple CNN Architecture.

D. MultiTask Learning: Dataset Synchronization

The Multitask model needed to look at the same time frame when training on both branches (this is referred to here as dataset synchronization) but we were faced with two problems. Firstly the driving data model was initially meant to be trained on the driving data subset and secondly, the ResNet50-GRU model looked at 224 data points with 50% overlap while the BiLSTM model only looked at 112 data points. This means for every two sets of training data for the ResNet50-GRU model, its time respective feature for the BiLSTM model will be 3 features.

Solving this was easy. For the first problem, its solution was found in introducing a dummy variable which would represent non-driving data and carrying out the preprocessing for the other modes of transport. For the second problem, we simply duplicated the data points so as to achieve data synchronization as shown in Figure 3. This way we could shuffle our data during training and test data split and also during training the models and both branches would be at the same set of data points but in different representations.

E. Model Architectures

1) Transport Mode Classification—BiLSTM: Both tasks were solved separately as single tasks to allow for comparison with the multitask model which we expected to see a significant performance increase. As discussed in subsection above, for solving the transport mode classification problem, we implemented a Bi-LSTM model with one hidden layer made up of 300 hidden units. The output from the BiLSTM was passed through a ReLU activation function before it got to a fully connected layer which gave the SoftMax probabilities for the 8 classes. There was no need for a dropout due to the simplicity of the model's architecture. The model was trained using the AdamW optimizer with weight decay and an exponential learning rate scheduling technique. The Cross Entropy loss function was used with the normalized class weights given due to the imbalance in the class size.

2) Driver Identification—ResNet50-GRU: For the driver identification single task model, we made use of the ResNet50-GRU model as described earlier. We modified the original ResNet50 architecture by replacing its final classification block with a GRU layer followed by a fully connected layer. As a

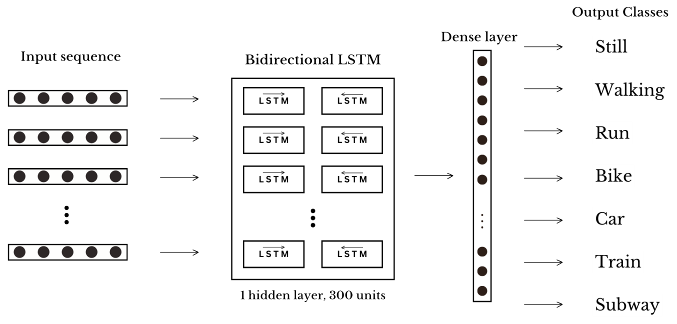

Fig. 6: BiLSTM architecture for transport model classification.

result, the input into the GRU is 2048 feature maps of size 7x7. To account for the input shape of GRU (batch size, sequence length, input size), each 7x7 feature map was flattened to represent the sequence length which had an input size of 2048. This effectively means that each row in the sequence of length 49, looked as the same pixel position for 2048 feature maps. The output of the GRU was connected to a fully connected layer which gave the SoftMax probabilities for the 4 classes (i.e. the 3 users and the dummy class).

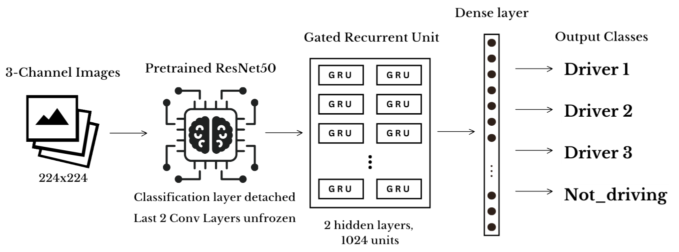

Fig. 7: ResNet50-GRU architecture for driver identification.

Additionally, we fine-tuned the model by unfreezing the last two convolutional layers of ResNet50, enabling them to be updated during training to better capture the specific features relevant to our task. The GRU was made of 2 hidden layers each with 1024 hidden units. The model was trained using the Adam optimizer with weight decay and an exponential learning rate scheduling technique. The Cross Entropy loss function was used with the class weights given due to the imbalance in the class size. To account for the large amount of overfitting in the model, dropout was implemented before the GRU layer and in the GRU, as well as data augmentation. The data augmentation implemented involved randomly performing a series of PyTorch transforms to the images including horizontal flips, color jitter, resize and crop, gaussian blur, random affine, and rotation. However we ensured that the size was always resized to match the required 224x224 shape and this greatly reduced overfitting in the model.

3) Multi-Task Learning Model: As seen in Figure 8, the MTL model was made up of both single task models with two shared single fully connected layer before branching out for predictions of both tasks. The model was trained using the AdamW optimizer with weight decay and the 'reduce learning rate on plateau' scheduling technique. Separate cross entropy loss functions were used for the transport loss ($Loss_{tr}$) and driver loss ($Loss_{drv}$), each with the respective normalized class weights. To arrive at the total loss which would be used to calculate the gradients, a uniform weighting was implemented as shown in equation (7), where α and β was set as 1.

$$Total\ Loss = \alpha \cdot Loss_{tr} + \beta \cdot Loss_{drv} \tag{7}$$

To account for the large amount of overfitting in the model as a result of the ResNet50-GRU branch, dropout was implemented at the shared fully connected layers, as well as data augmentation of the image data. The data augmentation implemented was identical to the one used in the single task driver identification model to allow for a fair comparison.

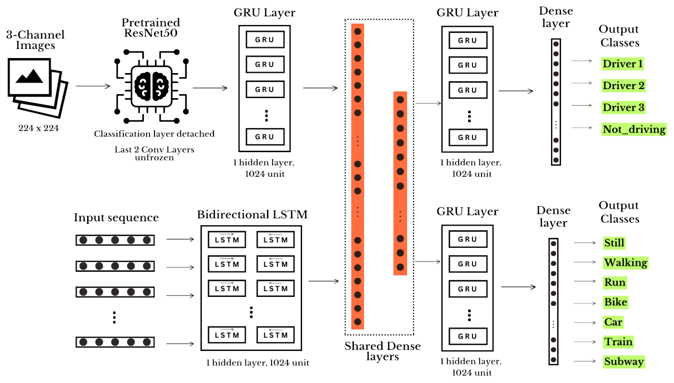

Fig. 8: Multitask Model architecture

F. Evaluation Metrics

In evaluating the performance of our models, we employed several key metrics: Accuracy, Precision, Recall, and F1 Score. These metrics provide a comprehensive assessment of the model's degree of generalization when tested on the test set.

a) Accuracy: This measures the proportion of correctly predicted instances among the total instances. It is a fundamental metric that gives an overall sense of model performance. The formula for accuracy is:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Due to the severe class imbalance in this dataset, accuracy would not be used to assess the model's performance although it was used during training. Instead we would use the remaining metrics.

b) Precision: This is the ratio of true positive predictions to the total predicted positives, provides insight into the quality of positive predictions made by the model:

$$\text{Precision} = \frac{TP}{TP + FP}$$

c) Recall: This measures the ratio of true positive predictions to the total actual positives, indicating the model's ability to capture positive instances:

$$\text{Recall} = \frac{TP}{TP + FN}$$

d) F1 Score: This is the harmonic mean of Precision and Recall, balances the trade-off between the two metrics. It is particularly useful in scenarios with imbalanced datasets:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where: TP, TN, FP, FN represent true positive, true negative, false positive, and false negative respectively.

G. Experiment Configuration & Resources

The model training was done on a Lenovo Legion 5 15ACH6H, equipped with an Nvidia GeForce RTX 3060 GPU, AMD Ryzen 7 5800H processor, and 16GB of RAM. For software, the model was implemented using Python with the PyTorch [34] library as well as utilizing libraries like NumPy [35], Pandas [36] and Scikit-learn [37]. Before arriving at the near-optimal hyperparameters for training the models, we made use of the Python library, Ray Tune [38], for hyperparameter tuning which is compatible with PyTorch. Using Random Search, we started at a wide range of values before working my way to a more reasonable range of values based on the results gotten using just 10% of the data. The hyperparameters used in training are seeing in Table I. Initial training was done with the seed set at 42 with further training done on the remaining 4 seeds shown in the table.

Parameters	BiLSTM	ResNet-GRU	MTL Model
Loss	CrossEntropy	CrossEntropy	CrossEntropy (2)
Optimizer	AdamW	Adam	AdamW
Scheduler	ExponentialLR	ExponentialLR	ReduceLROnPlateau
Batch Size	64	32	128
Epochs	40	30	50
Hidden units	300	1024	1024
No. of Layers	1	2	1
Learning Rate	0.00769697	0.00132603	0.00053923
Weight decay	0.0756331	0.00111036	0.04224803
alpha, beta	N.A., N.A.	N.A., N.A.	1, 1
Pr(dropout)	0.4946*	0.7	0.6409

TABLE I: Table showing hyperparameters used in training the models at 5 different random seeds; 42, 172, 47, 117, and 192. *Pr(dropout) for BiLSTM is essentially 0 since it has only one layer.

V. Result & Discussion

In this section, we would discuss the results of the models by comparing their performance on the single tasks and also as an ensemble method. Throughout this section, we refer to the single task model for transport mode classification as BiLSTM model, the single task model for driver identification as ResNet50-GRU model and the multitask learning model as MTL model.

A. Learning Rate

From Figure 9, we can see that the single task model took a far less time in training with the BiLSTM model taking roughly 1.1min/iteration (including training and validation) and the ResNet50-GRU model taking 5.2min/iteration. Although the MTL model had a far longer training time, marked at 6.54min/iteration, we can see that it is roughly equivalent to the sum of the training times for both single tasks although training was done simultaneously.

Before discussing the issues encountered during training, it is important to highlight the positive fact that the loss gotten at the end of training was relatively low for both kinds of models especially the MTL model where the loss was the sum of the losses for both tasks. On the negative side of things, all models suffered from overfitting initially and as a result, we had to implement hyperparameter tuning in order to experiment which set of adjustment works. We eventually ended up with these set of hyperparameters as seen in Table I.

However, we were able to reduce the overfitting in the BiLSTM model, the ResNet50-GRU model, after the 10th epoch, stopped learning as seen in the plateau. This could probably be caused by the information in the feature maps not being too discrimitative or could be as a result of the model having to discriminate between driving data and non-driving data. Nevertheless, the MTL model appears to have solved it as seen in the plot. However, as with the BiLSTM model, it also suffers from overfitting.

B. Precision and Recall

For a closer look at the predictive capabilities of both kinds of models, we decided to compare each single task with its respective branch in the MTL model to see if there is a performance increase. In Figure 10, we can already notice the impressive improvement when comparing the single task models and the multitask.

In the BiLSTM, we can see that the model struggles a bit with classes, 'Train', 'Bus', and 'Subway' and this is possibly due to the fact that these modes of transport seldom have significant variations in maneuvers in the x, y, and z directions. However, it shows a good prediction for the 'Car' and 'Still' class whilst having near perfection in predicting the 'Run', 'Walking' and 'Bike'. In the transport branch of the MTL model, we can see that the model learnt to predict the classes which it struggled with in the single task model way better. Focusing on the 'Car' class, we can see it has a balanced precision/recall of around 0.9/0.9.

In the ResNet50 model, we can see effect of the plateau shown in its learning curve. The model wasn't learning and as a result, it has difficulty differentiating the drivers. This is most likely due to the imbalance in the 'Not_driving' dummy class and as such, even though class weights were used during training, it still focused more on learning to predict this class. However, in the MTL mode, we can see a significant improvement in the predictive performance as it is able to properly differentiate, with good precision and recall, between the 3 drivers as well as the dummy class.

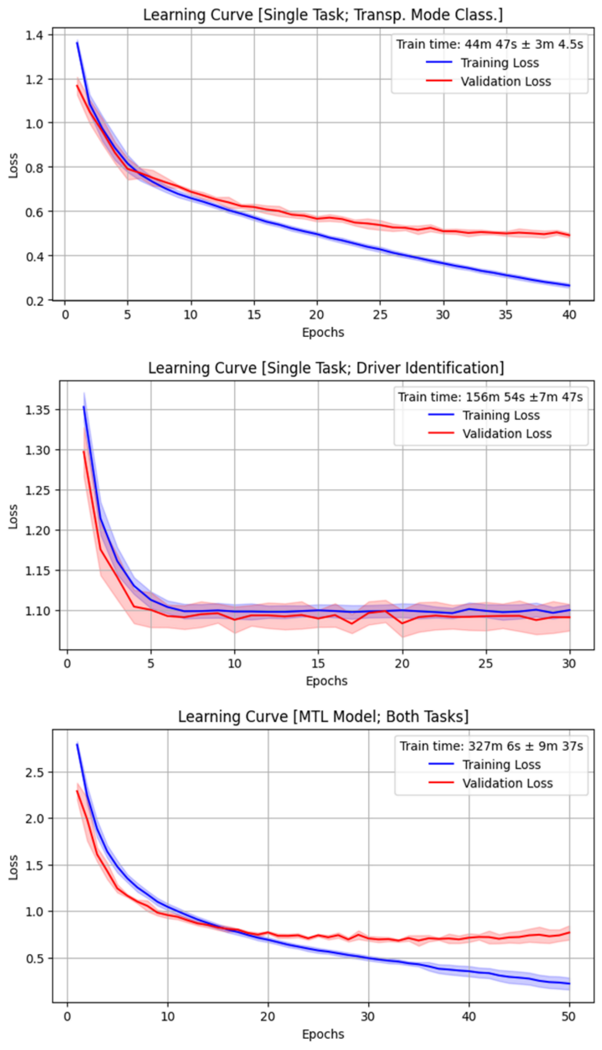

Fig. 9: Learning rates for all 3 models at all 5 seeds. Confidence band shows 95% confidence.

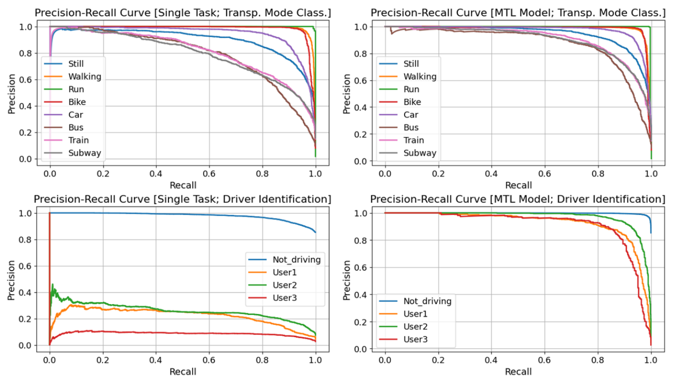

Fig. 10: Precision-Recall curve per task for both kinds of models

C. F1 Score and Recall

The F1 score which gives a balanced interpretation of precision and recall, sheds more light into what was discuess in the previous precision and recall subsection. From Figure 11, we can see that for the transport mode classification aspect improves over the prediction in the BiLSTM model. However, seeing as the BiLSTM model in Figure 9 was overfitting as well as the MTL model, we believe that improvements could be made by addressing the possible reasons for overfitting in this task. This would better improve the MTL's performance in transport mode classification.

For driver identification, we can also see that the MTL model significantly improved over the single task ResNet50-GRU model. This could possibly be because of the shared features from the transport mode classification branch which enabled the model to focus more on distinguishing between the drivers. It could be possible to see improvements in this aspect once the overfitting for the transport mode classification branch is sorted out.

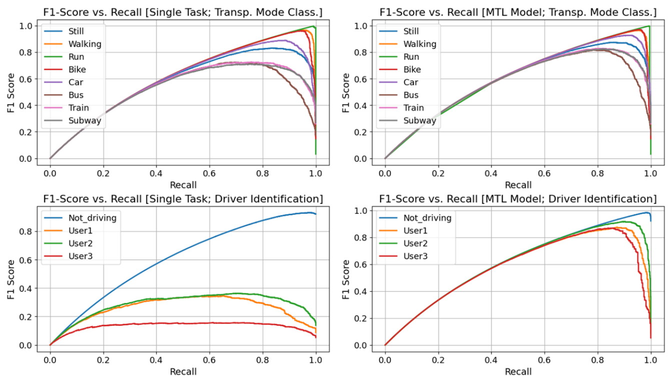

Fig. 11: F1 score-Recall curve per task for both kinds of models

D. *Precision, Recall & F1 score for Ensemble Method*

For this subsection, we decided to compare the performance of both kinds of models when taken as an ensemble method where the softmax probabilities for both tasks would be combined after prediction to form the prediction probabilities for the new combined classes. This was aimed at exploring the question, for a given set of data for any user, can we know who the driver is only when the user is in a car. This led to the combination of both classes from the two different classes as seen in the Figure 12 where the 'Mismatch' class refers to the wrong pair of predictions i.e. predicting a driver when the mode is a bus/walking or predicting a 'car' with the driver class as 'Not_driving'.

First glance at the precision-recall curve and F1-score-recall curve for the single task ensemble case in Figure 12 would give the impression of a better prediction when compared to the multi-task scenario. However that is not the case, although the single task mode does well in predicting the non-driving pairs, it does a horrible job in getting the right drivers and also has a number of mismatch pairs.

As for the Multi-task ensemble mode, although it performs better than the single task mode, its performance isn't that impressive. Looking at the F1 score for the User1 (Car), it barely gets to 0.6 before plummeting. However, one important thing to note is that it has no mismatched predictions as can be seen by the diagonal line in its precision-recall curve. This means that the model learnt to make the right pairs of predictions.

It is important to remember that the model was not trained as an ensemble and as a result these results do not reflect the

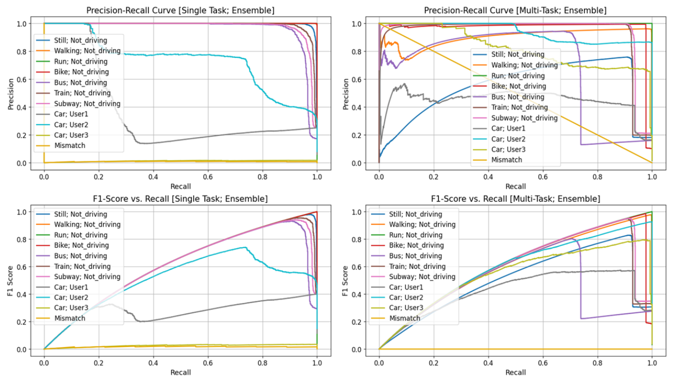

Fig. 12: Precision-Recall curve and F1 score-Recall curve for both kinds of models when each is taken as an ensemble method. Note that the 'Mismatch' class is a dummy class which represents wrong pairs of predictions and that the true label has no mismatch.

performance of the model for its intended purposes i.e. solving both tasks simultaneously.

VI. Conclusion

This study proposes a multitask learning model which aims at simultaneously solving transport mode classification and driver identification using a deep multitask learning network which combines methods from research. Although most studies have solved these problems separately, they overlooked the fact that these tasks are similar and would benefit from shared information. We also limited our sensor data to accelerometer, gyroscope and rotation vector sensor as well as training on data with a low sampling rate.

As was seen in the results, the multitask model showed a significant improvement in performance over the single task models although it took a longer time to train. However, as was also seen in the results, the models suffered from overfitting especially the transport mode classification section. Future work could investigate the possible changes either to the hyperparameters or the network architecture in order to improve prediction. Focus could also be placed on varying the window sizes used for prediction to test its effect on performance. Also, better journey segmentation techniques could be implemented than the basic method used.

VII. Declaration

A. *Declaration of Originality*

I am aware of and understand the University of Exeter's policy on plagiarism and I certify that this assignment is my own work, except where indicated by referencing, and that I have followed the good academic practices.

B. *Declaration of Ethical Concerns*

This work does not raise any ethical issues. No human or animal subjects are involved neither has personal data of human subjects been processed. Also no security or safety critical activities have been carried out.

References

[1] "Black Box Car Insurance for Learners and New Drivers — ingenie Insurance." Accessed: Aug. 02, 2024. [Online]. Available: https://www.ingenie.com/

[2] "Car Insurance — Towergate." Accessed: Aug. 02, 2024. [Online]. Available: https://www.towergateinsurance.co.uk/motor/car-insurance

[3] G. Castignani, T. Derrmann, R. Frank, and T. Engel, "Driver behavior profiling using smartphones: A low-cost platform for driver monitoring," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 91–102, Jan. 2015, doi: 10.1109/MITS.2014.2328673.

[4] X. Song, C. Markos, and J. J. Q. Yu, "MultiMix: A Multi-Task Deep Learning Approach for Travel Mode Identification with Few GPS Data," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, Sep. 2020, pp. 1–6. doi: 10.1109/ITSC45102.2020.9294272.

[5] S. Hernández Sánchez, R. F. Pozo, and L. A. H. Gómez, "Driver Identification and Verification from Smartphone Accelerometers Using Deep Neural Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 97–109, Jan. 2022, doi: 10.1109/TITS.2020.3008210.

[6] G. Xiao, Z. Juan, and J. Gao, "Travel mode detection based on neural networks and particle swarm optimization," *Information* (Switzerland), vol. 6, no. 3, pp. 522–535, 2015, doi: 10.3390/info6030522.

[7] N. Asif, H. Zhiqiu, and W. Senzhang, "SSMDL for Transportation Mode Classification and Path Prediction with GPS Trajectories," in *Web and Big Data: 4th International Joint Conference*, APWeb-WAIM 2020, Tianjin, China, September 18-20, 2020, Proceedings, Part II 4 , Springer International Publishing, 2020, pp. 391–405. doi: 10.1007/978-3-030-60290-1_31.

[8] N. Theresa, C. Edmund, G. Jan, and G. Juergen, "Neural Networks (IJCNN), the *2010 International Joint Conference* on: date, 18-23 July 2010," in The 2010 International Joint Conference on Neural Networks (IJCNN), Spain: IEEE, Jul. 2010, pp. 1–6. doi: 10.1109/IJCNN.2010.5596549.

[9] M.-C. Yu et al., "Big Data Small Footprint: The Design of A Low-Power Classifier for Detecting Transportation Modes," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1429–1440, 2014, doi: 10.14778/2733004.2733015.

[10] H. Zhao, C. Hou, H. Alrobassy, and X. Zeng, "Recognition of Transportation State by Smartphone Sensors Using Deep Bi-LSTM Neural Network," *Journal of Computer Networks and Communications*, vol. 2019, 2019, doi: 10.1155/2019/4967261.

[11] H. S. Sara, F. P. Rubén, and A. H. G. Luis, "Deep Neural Networks for Driver Identification Using Accelerometer Signals from Smartphones," in *International Conference on Business Information Systems*, Springer, Cham, May 2019, pp. 206–220. doi: 10.1007/978-3-030-20482-2_17.

[12] S. Jafarnejad, G. Castignani, and T. Engel, "Towards a real-time driver identification mechanism based on driving sensing data," in 2017 *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–7. doi: 10.1109/ITSC.2017.8317716.

[13] S. H. Sánchez, R. F. Pozo, and L. A. Hernández Gómez, "Estimating vehicle movement direction from smartphone accelerometers using deep neural networks," *Sensors*, vol. 18, no. 8, Aug. 2018, doi: 10.3390/s18082624.

[14] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature* 2015 521:7553, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[15] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/NECO.1997.9.8.1735.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," Dec. 2014, [Online]. Available: http://arxiv.org/abs/1412.3555

[17] G. B. Zhou, J. Wu, C. L. Zhang, and Z. H. Zhou, "Minimal gated unit for recurrent neural networks," *International Journal of Automation and Computing*, vol. 13, no. 3, pp. 226–234, Jun. 2016, doi: 10.1007/s11633-016-1006-2.

[18] M. Pirani, P. Thakkar, P. Jivrani, M. H. Bohara, and D. Garg, "A Comparative Analysis of ARIMA, GRU, LSTM and BiLSTM on Financial Time Series Forecasting," *IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics*, ICDCECE 2022, doi: 10.1109/ICDCECE53908.2022.9793213.

[19] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," 1997.

[20] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," EMNLP 2014 - *2014 Conference on Empirical Methods in Natural Language Processing*, Proceedings of the Conference, pp. 1724–1734, Jun. 2014, doi: 10.3115/v1/d14-1179.

[21] R. Caruana, "Multitask Learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997, doi: 10.1023/A:1007379606734/METRICS.

[22] T. Gong et al., "A Comparison of Loss Weighting Strategies for Multi task Learning in Deep Neural Networks," IEEE Access, vol. 7, pp. 141627–141632, 2019, doi: 10.1109/ACCESS.2019.2943604.

[23] H. Gjoreski et al., "The University of Sussex-Huawei Locomotion and Transportation Dataset for Multimodal Analytics with Mobile Devices," IEEE Access, vol. 6, pp. 42592–42604, Jul. 2018, doi: 10.1109/ACCESS.2018.2858933.

[24] L. Wang, K. Murao, H. Gjoreski, T. Okita, and D. Roggen, "Summary of the Sussex-Huawei locomotion-transportation recognition challenge," in UbiComp/ISWC 2018 - *Adjunct Proceedings of the 2018 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2018 ACM International Symposium on Wearable Computers*, Association for Computing Machinery, Inc, Oct. 2018, pp. 1521–1530. doi: 10.1145/3267305.3267519.

[25] "App – Sussex-Huawei Locomotion Dataset." Accessed: Aug. 06, 2024. [Online]. Available: http://www.shl-dataset.org/app/

[26] "Download – Sussex-Huawei Locomotion Dataset." Accessed: Aug. 06, 2024. [Online]. Available: http://www.shl-dataset.org/download/#shldataset-preview

[27] "Motion sensors — Sensors and location — Android Developers." Accessed: Aug. 06, 2024. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion

[28] "Sensors Overview — Sensors and location — Android Developers." Accessed: Aug. 06, 2024. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview#sensors-coords

[29] "Motion sensors — Sensors and location — Android Developers." Accessed: Aug. 06, 2024. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion#sensors-motion-rotate

[30] "Motion sensors — Sensors and location — Android Developers." Accessed: Aug. 06, 2024. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion#sensors-motion-accel

[31] "Motion sensors — Sensors and location — Android Developers." Accessed: Aug. 06, 2024. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion#sensors-motion-gyro

[32] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, "Sensors Used in Mobile Systems," in *Wheeled Mobile Robotics*, Elsevier, 2017, pp. 207–288. doi: 10.1016/b978-0-12-804204-5.00005-6.

[33] E. I. Vlahogianni and E. N. Barmpounakis, "Driving analytics using smartphones: Algorithms, comparisons and challenges," *Transportation Research Part C: Emerging Technologies*, vol. 79, pp. 196–206, Jun. 2017, doi: 10.1016/j.trc.2017.03.014.

[34] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[35] C. R. Harris et al., "Array programming with NumPy," *Nature* 2020 585:7825, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[36] W. Mckinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51–56.

[37] F. Pedregosa Fabianpedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011, Accessed: Aug. 07, 2024. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[38] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A Research Platform for Distributed Model Selection and Training," Jul. 2018, Accessed: Aug. 07, 2024. [Online]. Available: https://arxiv.org/abs/1807.05118v1