HAL
open science

The Devil Is in the Details: An Efficient Convolutional Neural Network for Transport Mode Detection

Hugues Moreau, Andrea Vassilev, Liming Chen

To cite this version:

Hugues Moreau, Andrea Vassilev, Liming Chen. The Devil Is in the Details: An Efficient Convolutional Neural Network for Transport Mode Detection. IEEE Transactions on Intelligent Transportation Systems, inPress, 10.1109/tits.2021.3110949 . hal-03346165

HAL Id: hal-03346165

https://hal.science/hal-03346165v1

Submitted on 16 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Published version is available:
https://doi.org/10.1109/TITS.2021.3110949

The Devil Is in the Details: An Efficient Convolutional Neural Network for Transport Mode Detection

Hugues Moreau, Andrea Vassilev, and Liming Chen, *Senior IEEE member*

Abstract—Transport mode detection is a classification problem aiming to design an algorithm that can infer the transport mode of a user given multimodal signals (GPS and/or inertial sensors). It has many applications, such as carbon footprint tracking, mobility behaviour analysis, or real-time door-to-door smart planning. Most current approaches rely on a classification step using Machine Learning techniques, and, like in many other classification problems, deep learning approaches usually achieve better results than traditional machine learning ones using handcrafted features. Deep models, however, have a notable downside: they are usually heavy, both in terms of memory space and processing cost. We show that a small, optimized model can perform as well as a current deep model. During our experiments on the GeoLife and SHL 2018 datasets, we obtain models with tens of thousands of parameters, that is, 10 to 1,000 times less parameters and operations than networks from the state of the art, which still reach a comparable performance. We also show, using the aforementioned datasets, that the current preprocessing used to deal with signals of different lengths is suboptimal, and we provide better replacements. Finally, we introduce a way to use signals with different lengths with the lighter Convolutional neural networks, without using the heavier Recurrent Neural Networks.

Index Terms—Transport mode detection, GPS, CNN, Deep learning

1 Introduction

Transport mode detection is a family of classification problems in which an algorithm has to predict the transport mode of a given user, using several signals, whether it is GPS or inertial sensors (or both). The exact list of possible transport modes may vary depending on the application, but most applications include at least the most common ones: Walk, Bike, or Car, for instance. The applications are numerous: real-time trajectory planner (one that avoids traffic jams to drivers, for instance); automatic carbon footprint estimator; or mobility behaviour analysis. The signals are collected from an embedded device (either the sensors of a mobile phone, or a dedicated device), and processed by a Transport Mode Detection Algorithm, in order to determine the transport mode of the owner of the device. This algorithm has multiple steps that often include cleaning, preprocessing, and classification in itself. The latter operation is the most diverse from one algorithm to the other. All algorithms use Machine Learning *i.e.*, methods that use a certain amount of labeled data to learn how to predict the output transport mode, before making predictions on unseen samples. But the approaches can be classified into two families: the 'traditional' Machine Learning approach, relying on handcrafted features (as in [1], [2], [3], [4], [5], [6]), and the Deep Learning one, in which the classification is learnt using neural networks (see [7], [8], [9], [10]). The traditional Machine Learning method involves using handcrafted features which are assumed to be relevant enough to solve the problem, while the Deep Learning approach replaces these handcrafted features by features that are automatically learnt by a deep neural network. This approach generally yields better performance than the traditional one, but at a cost: the memory and computation requirements are dramatically increased, thus limiting the development of embedded systems. Yet, embedded applications are a great field of application for neural networks [11].

To understand the kind of constraints embedded applications bring, let us take an example with a practical use case. Let us consider a carbon footprint automatic estimator on a smartphone. The end user would download an application, and record the beginning and end of their trips. The application automatically estimates the transport mode of the user to compute an estimation of the greenhouse gas the trip emitted. Compared to a client-server application, an embedded classifier allows to guarantee privacy to the users, and can function despite uncertain wireless data coverage. However, if this embedded classifier is not resource-efficient, the application will end up draining the battery of the end users, who might choose not to use it. Hence, resource efficiency is key for embedded applications.

When dealing with embedded devices, most neural networks are trained offline (on a computer) on prerecorded data, and sent to embedded devices for inference. This method relies on the fact that the training is the most computationally-intensive step. However, running only the inference on embedded devices does not solve all problems: a bigger model translates into longer inference times (which sometimes prevent from using the device in real-time applications), and increased energy consumption, *i.e.* reduced battery autonomy. If some works manage to em-

- *H. Moreau and A. Vassilev are with the Universite Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France ; E-mail: {hugues.moreau, andrea.vassilev}@cea.fr*
- *L. Chen is with the Computer science and mathematics department at cole Centrale de Lyon, France. E-mail: liming.chen@ec-lyon.fr*

Manuscript received Sept, 2020; revised June, 2021.

© 2021 IEEE

bed neural networks intelligently, (see [12], for instance), having a heavy network with many parameters is always a hurdle to efficient embedding. From this point of view, traditional Machine Learning methods, with their reduced requirements, are better suited to be used with embedded systems.

We will show that it is possible to get the best of both worlds. By using Convolutional Neural Networks instead of the heavier Recurrent Neural Networks we create a deep model that has a small number of parameters and a good classification performance, on two realistic datasets. Also, we show how to expand to CNNs one of the principal quality of RNNs: the ability to use segments of any lengths.

Our contributions are the following:

- We reduce the size of a network by a factor of 4 by changing its pooling layer
- We demonstrate how to use global pooling methods to allow a convolutional neural network to use segments of different lengths
- We show that the current padding method used to get to segments with equal lengths, the zero-padding, impairs significantly the learning process, and we introduce padding by wrapping, which improves the performance

The rest of the paper is organized as follows: section 2 states the problem and reviews related works. Section 3 describes the two improvements we propose, before section 4 explains how each of these choices are justified experimentally.

2 Problem Statement and Related Work

We first introduce some preliminary definitions (2.1) before the statement of the transport mode detection problem (2.2). Then we review the related work (2.3).

2.1 Preliminary definitions

We will focus on Transport Mode Detection, a problem aiming to guess the transport mode from incoming temporal data. The database consists of a series of *trajectories* (each trajectory being a series of measurements points, such as $(lat, long)$ for GPS signals, or Acc_x, Acc_y, Acc_z (for accelerometer data). Each trajectory has to be divided into *trips*: series of points that are likely recorded in one go (we chose GPS points such that two consecutive points are distant by less than 20 minutes, as in [6]). Each trip is made of one or several *triplegs*: series of points sharing a single transport mode. This setting corresponds to a more simple version of the problem, where we know triplegs to have only one mode. In real-life applications, we could consider applying segmentation algorithms like in [1], [6], [13].

The triplegs might still have different lengths, but a model sometimes requires all inputs to have the same shape. When this is the case, we cut triplegs into fixed-shape *segments*. When we use a model that can deal with arbitrary-sized inputs, segments are equal to triplegs. Thus, a model only classifies segments.

2.2 Transport mode detection as a classification problem

Our goal is to use some labeled data in order to learn to assign a transport mode to unknown data. Using Machine Learning vocabulary, this is a classification problem: we use a certain amount of labelled samples to train a classifier that will predict the transport mode used during the recording of an unseen segment:

$$class(segment_i) \in \{walk,\ bike,\ bus,\ car\,\&\,taxi,\ train,\ subway\}$$

Some additional preprocessing (like computing speed and acceleration) is applied so that the data can be fed to a machine learning model. In order to train and test properly the model, the dataset will be split into train, validation and test sets. A fraction of the data with the associated labels will be used to train the machine learning model to predict the class of an unseen segment. This is the *train set*. We usually have a wide choice when selecting preprocessing functions. In addition, machine learning models generally involve several hyperparameters, *e.g.*, number of filters, or number of layers, that need to be chosen. These hyperparameters are optimized and chosen in evaluating the variants of the machine learning models on previously unseen segments, the *validation set*. Once we have chosen every possible parameter using the validation set, we use the last part of the dataset (*test set*) to evaluate the generalisation skill of the learned model against the state of the art. The following algorithm shows how we use the three datasets:

Require: Three datasets X_{train}, X_{val}, X_{test}, for training, validation, and testing; a list of hyperparameters sets L_h to evaluate.
  $best_score \leftarrow 0$
  $best_hyperparameters \leftarrow \emptyset$
  for h in L_h **do**
    Create a deep learning model with hyperparameters h
    Train the model on X_{train}
    Evaluate the model on X_{val}, measure the F1-score $score_{val}$
    if $best_score < score_{val}$ **then**
      $best_score \leftarrow score_{val}$
      $best_hyperparameters \leftarrow h$
    end if
  end for
  Create a deep learning model with hyperparameters $best_hyperparameters$
  Train the model on X_{train}
  Evaluate the model on X_{test}, measure the F1-score $score_{test}$
  return $score_{test}$ for comparison with the state-of-the-art

2.3 Related work

Current research on the subject features two different approaches: traditional machine-learning approaches using handcrafted features and deep learning-based approaches. However, both approaches share a common structure as follows:

- *data cleaning*: When dealing with signals that are are known for being noisy (such as GPS signals [14]),

most research works use Kalman filters [3], Savitzky-Golay filters [15], or outlier detection thanks to clustering techniques [3] to clean the data. Some approaches also remove the trajectory from the dataset if its speed or acceleration is above a realistic threshold given the class of the trajectory (for instance, the maximum speed and acceleration for the 'bus' class are set to 120 km/h and 2 m/s^2 respectively in [15])

- *point-level feature computation*. When dealing with GPS data, researchers often convert the *(lat, long)* into more significant values. Those features are computed at each timestamp. Speed and acceleration are used universally. Other point-level features include distance [6] [2], jerk (the time derivative of acceleration [3]), or delta-bearing (the angle difference at each time step) [3].
- *segmentation* in single-mode segments. This step is often skipped, most research works prefer using the ground truths to be sure to work on segments containing one transport mode. Those who segment using the data typically rely on walk detection (a small walk is usually necessary between two modes, [6]) or the PELT algorithm [13], [16].
- For classical Machine-learning approaches, *trajectory-level feature computation* takes place to get back to a fixed-size feature vector, usable by machine learning models. This treatment often involves computing the mean, standard deviation, minimum, and maximum of each point-level feature [2], [3], [17]. Other used operations are the computation of percentiles [3], [17], frequency energy bands [5], or autocorrelation coefficients [17]. Some trajectory-level features do not rely on a specific point-level feature, such as the stop rate and direction change rate [6], or the closeness to train and bus stations (obtained from other sources, such as Open Street Map [18], or Baidou Map [1]).
- *classification*. For classical Machine-learning approaches, the state of the art is Random Forests [3], [19] and SVM [20], [21]. Other classifiers include NaiveBayes [5], MultiLayer Perceptrons [5], [19], KNN [17], HMM [22], or even rules [23] and fuzzy rules [24].

The 'traditional' Machine Learning approaches use handcrafted features (*e.g.* average speed, maximum acceleration), before using a Machine Learning algorithm (SVM, Random Forest, *etc.*). These approaches are the most simple (computation-wise), but they are also less performing than pure deep learning approaches.

Within deep learning-based approaches, there is a great diversity of neural networks: Multi Layer Perceptron [1], [10], Convolutional Neural Networks [13], [15], [25], [26], [27], or LSTMs [7], [8] (a specific kind of Recurrent Neural Networks). Some ([1], [10], [25]) use autoencoders to extract features from trajectories but, curiously, only one [13] makes use of the unlabeled data in the dataset. This fact is rather surprising, given that unlabeled GPS data is relatively cheap to obtain, contrary to labeled data.

The neural networks that seem to be the best (we will see in section 4.6 that comparisons are not straightforward) are the LSTMs, followed by CNNs. However, Recurrent Neural Networks, (including LSTM) rely on matrix multiplications to compute the features for the next layer. This operation is costly in terms of memory, as the weight matrices usually have many parameters. To give an example, the LSTM from [7] uses 8 million weights, and 3.2 billion floating-point operations for a single inference, which represents heavy requirements. Contrary to Recurrent Neural Networks, Convolutional Neural Networks rely on convolution operations, which can be extremely cheap, while still retaining good performance levels.

3 Proposed improvements

We present the two main improvements we use to be able to obtain smaller networks with better performance: the choice of an effective pooling layer for convolutional neural networks (section3.1), and the padding of shorter segments (section 3.2).

3.1 Pooling operation

Our convolution layers work with 2-dimensional feature maps (with a *channel* axis and a *time* axis), while the fully-connected layers use 1-dimensional feature vectors. In order to get to a 1-dimensional vector, the most common is to use a flatten operation, a block that simply reshapes the feature map into a vector. However, this operation has a major downside: if the input feature map has T temporal steps and C channels, the next fully-connected will accept features with $T*C$ scalars. As the shape of a fully connected layer cannot change during the training (nor can the number of channels of the previous convolutional layer), the number of time steps T must remain fixed during the training. This means a convolutional network that uses a flatten cannot deal with arbitrary-shaped inputs. By replacing the flatten step with a *global pooling* operation [28], we obtain a vector which is as long as the input tensor had feature maps, no matter how many time points the input had (see fig. 1). This allows to reduce the number of parameters of the network (as the vector is shorter, the next fully-connected layer is much smaller), and to use segments that have arbitrary lengths.

Several variants of global pooling are possible. If $X_{t,c}$ is the two-dimensional matrix at the end of the last convolution layer (t being the index along the 'time' dimension, while c is the index along the 'channel' dimension), there are several ways to use global pooling. One could use an average over time ($Y_c = \frac{1}{T}\sum_{t=0}^{T-1} X_{t,c}$) or a maximum ($Y_c = max_{t\in 0..T-1}\ (X_{t,c})$), but a more general option would be to use the generalized mean [29]:

$Y_c = (\frac{1}{T}\sum_{t=0}^{T-1} (X_{t,c})^{\alpha_c})^{1/\alpha_c}$.

This expression uses one parameter $\alpha_c > 0$ for each channel c, which are learnt by gradient descent like any of the other weights. When $\alpha_c = 1$, the expression is equal to the arithmetic average. When $\alpha_c \to +\infty$, the generalized mean converges towards the maximum of the $(X_{t,c})_{t\in[1..T]}$. In order to avoid numerical instability, a small term[1] was added to the input tensor X. In practice, the values of α_c are initialized following $\mathcal{N}(5,1)$ because we do not want α_c to be negative before the learning even begins (if $\alpha_c < 0$,

1. the lowest value we could use was 5.10^{-5}, which seems quite high

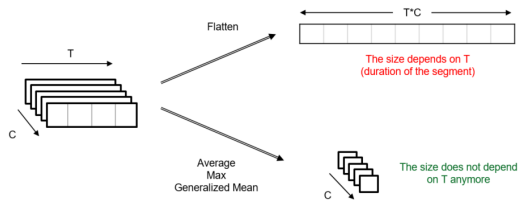

Fig. 1: Output sizes of different operations replacing the flatten step. We notice the size of the output of the flatten step depends on both the size of its input along the time dimension (T) and along the channel dimension (C)

the expression is equivalent to using a generalization of the harmonic mean, which is close to 0 when one of the features $X_{c,t}$ is close to zero, which makes the associated channel useless). The value of all the α_c seem to converge between 0 and 5 during the training process.

3.2 Padding

Padding is the action of filling a short segment with well-chosen values so that the segment reaches a desired shape. This action can be required in two occasions:

- When the model requires a fixed-size input, we cut the triplegs that are too long, which inevitably generates segments that are shorter than the limit. We pad those segments so that they have the same lengths as the others.
- When the model can deal with segments of arbitrary shape, one problem arises: to accelerate the training, the common practice is to parallelize the computation and to submit to prediction a set of 64 or 128 samples (a batch). This requires to put all segments into a single tensor, which is impossible when the segments have different lengths. One could simulate the batch computation by sending each segment one after the other and computing the weight update once after a certain number of segments are processed, but doing so would increase greatly the training times. This is why we pad all the short segments so that they reach the length of the longest segment in the batch.

There are several ways to pad short segments. One could pad using zero-values (like in [13], [15]), but this disturbs the learning process (see section 4.5). Instead, we pad using the data from the input segment itself (see fig. 2). We tried padding with a reflection of the segment (adding a reversed copy of the segment after the original), or simply repeating the segment until the maximum length is reached.

We show in section 4.5 that padding with zero-values (like in [13], [15]) is detrimental to the learning process. This is why we chose to pad by wrapping the segment around or by using a symmetric of the segment.

4 Experiments

We begin by giving details about the databases we used (sections 4.1, 4.2). For each dataset, we present the database

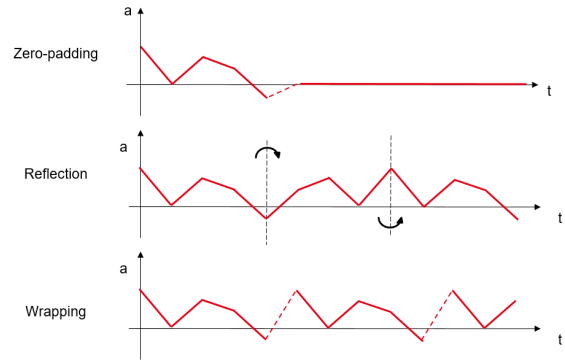

Fig. 2: The different kinds of padding.
Zero-padding simply adds zeros until the maximum length is reached, Reflection reverses a copy of the segment and adds it at the end, while wrapping simply duplicates the segment until the maximum length is reached

in itself, along with the preprocessing steps, and the baseline architectures we chose to use. In sections 4.4 and 4.5, we will alter these architectures by changing the pooling and padding (respectively), in order to justify the choices presented back in section 3, before we providing the results of our approach against the state of the art (section 4.6).

4.1 The GeoLife database

The GeoLife database [20], [30], [31] was collected between 2007 and 2012. The GPS signals of 180 participants living in five different cities of China were recorded during their commutes, in order to study their behaviours when travelling. Unfortunately, the need for labeled data did not appear immediately, and only one tenth of the trajectories of the database is labeled (in the subsequent, we will only refer to the labeled data, unless otherwise specified). The dataset is an ensemble of trajectories, each trajectory being a series of $(latitude, longitude, timestamp)$ points. Each labeled trajectory has one or more transport modes, and each change between modes has an associated timestamp, so that each point can be attributed a label. The transport modes (classes) in the dataset are: *walk, bike, car, taxi, bus, train, subway*. An overview of the dataset is available in table 1. We follow the recommendations of the GeoLife user guide [20], merging together the classes 'taxi' and 'car'. One important thing to note is that the data points are not sampled at the same rate: some trajectories have a sampling rate of 1 or 2 Hz, while some others can go down to 0.02 Hz.

As we explained in section 2.2, we split the triplegs into three sets, training, validation, and testing. For the GeoLife dataset, 64% of the triplegs go in the train set, 16% in the validation set, and 20% in the test set. With this dataset, we chose the hyperparameters using Random Search [33]. We fix a possible range of hyperparameters using usual values found in the literature, and train several models, each model using a series of hyperparameters at random. We then selected the hyperparameter which improved the performance on average, by looking at the median and quartiles of the validation F1 (results not shown). We also use the validation set for early stopping in each of the

dataset	GeoLife [20]	SHL 2018 [32]
number of users	69	1
total duration	5,000 h	216 h
total trajectories length	116,000 km	1,600 km
average interval between two data points	7s	0.01s
median interval between two data points	2s	0.01s
Total number of data point in the database	2.5×10^6	9.6×10^7
sensors used	GPS	Accelerometer
classes	Walk, Bike, Bus, Car & Taxi, Subway, Train	Still, Walk, Bike, Run, Bus, Car, Subway, Train

TABLE 1: An overview of the labeled data in the GeoLife and SHL 2018 datasets

	walk	bike	car & taxi	bus	subway	train
total	4517	1731	1459	2129	632	200
train	2890	1108	934	1362	405	128
validation	723	277	233	341	101	32
test	904	346	292	426	126	40

(a)

	still	walk	run	bike	car	bus	subway	train
total	2302	2190	686	2101	2475	2083	2520	1953
train	1899	1630	506	1716	2141	1663	1973	1472
validation	403	560	180	385	334	420	547	481

(b)

TABLE 2: The number of triplegs in each subset the GeoLife (a) and SHL 2018 (b) datasets. Note that the GeoLife dataset is more balanced than the SHL dataset

training process, during the random search or to produce the results in the present work.

4.1.1 Preprocessing

We begin by computing the speed and acceleration of each point. This way, our data is not dependent on the precise location of the trajectory. We remove the data points which acceleration or speed are deemed unrealistic given the annotated transport mode (we reused the values from [15]). We considered adding the bearing [15], but it turned out using this feature did not increase the performance of our model. As the sampling rate is very irregular, we interpolate linearly our data points (T = 2s), so that a difference between two consecutive points always has the same meaning.

We do not apply any cleaning or filtering, for we found these to be unnecessary.

4.1.2 Data Preparation and Splitting

Etemad [3] showed that the way the segments are split between the different sets (training, validation, test) can have a huge influence on performance: when a tripleg is split into several segments and the segments of a single trajectory can go in both the training and the test set, the trained model will be likely to have seen parts of all trajectories in the dataset, which will cause it to overfit.

In his experiments, Etemad found the F1 score can be vary by 20 percentage points between a training scenario in which the users are correctly split (71 %), compared to a scenario in which the fragments of a given trajectory can go in different sets (91 %). This result is not surprising, as mobility trajectories have a high degree of regularity [34], [35], [36]. Ideally, we should train a prediction model using the trajectories from a set of users and test the generalization skills of the trained model on those of unseen users. This means that we need first to assign the trajectories of each user to one set among training, validation or test (as in [3], [10], [13], [15]). This corresponds to the most realistic setting, where an algorithm predicts the transport mode of unseen users.

But in practice, with the GeoLife dataset, this method leads to extremely imbalanced sets, as users have different habits when it comes to transportation. In some cases (depending on the seed used to initialize the splitting process), splitting the dataset by users can even produce validation or test sets that completely lack one class. To show this, we realized 200 separations with different seeds initializing the random process, and looked at the effect it would have on the final distribution (results not shown). If the median is centered around the correct distribution, the quartiles show a high variance. In the validation and test set, the third quartile is at least twice higher than the first quartile. This distribution variance is a problem for comparison, because the performance of imbalanced models will strongly depend on the distribution, even when using measures like the F1 score.

This is why we only split the sets by tripleg: we first separate triplegs between train (64% of the trajectories), validation (16%), and test (20%), before segmenting the triplegs into segments. This method is less realistic than splitting the sets by users, but it allows to produce sets with similar distributions consistently. The following pseudo-code explains how we split by triplegs:

Require: A list L_{users} of users, each user being a list of trips
  Create a list L_{trips} of trips by merging all users
  Randomly split the list L_{trips} into three lists $\{L_{trips}^{train}, L_{trips}^{val}, L_{trips}^{test}\}$
  for s in $\{train, val, test\}$ **do**
    $X_s \leftarrow \emptyset$
    for each trip t in L_{trips}^{s} **do**
      Split the trip t into triplegs and,
      Add the triplegs to the set:
      $X_s \leftarrow X_s \bigcup t_{split}$
    end for
  end for
  return the three tripleg sets $X_{train}, X_{val}, X_{test}$

4.1.3 Baseline architecture

The GeoLife CNN uses the $(speed, acc)$ features to classify a segment. Its architecture is inspired by ResNet [38] and is given in fig. 3. As explained in the section 2, the use of convolutions allow to obtain a particularly efficient architecture compared to the recurrent models, especially in terms of memory footprint (see table 6).

To train our neural network, we use a weighted version of the cross-entropy loss: the loss of every segment is given

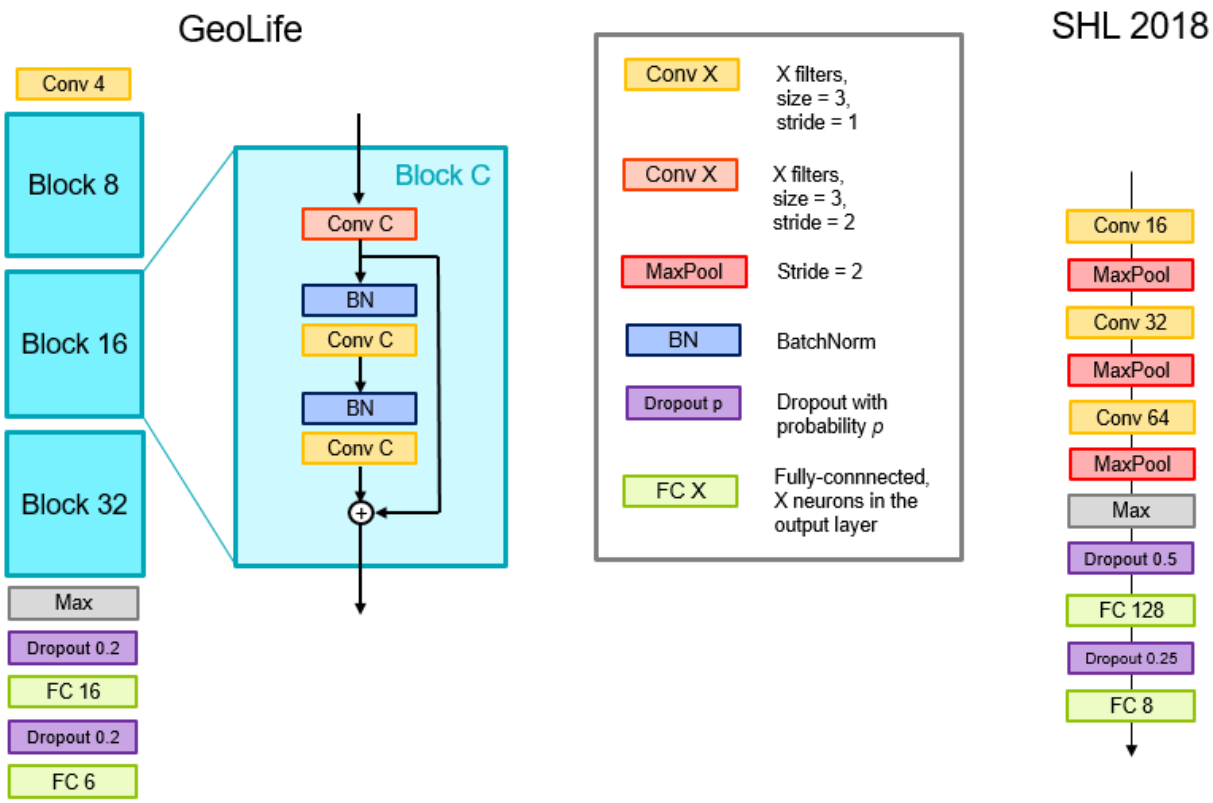

Fig. 3: The baseline architectures of both networks

Parameter	value (GeoLife)	value (SHL)
learning rate	10^{-2}	10^{-3}
regularization parameter	3.10^{-3}	10^{-3}
batch size	128	64
non-linearity	ReLU	ReLU
optimizer	Adadelta [37]	Adam
max number of epochs	2000	50
patience	100	/

TABLE 3: The chosen hyperparameters for the training of both models

a weight that is inversely proportional to the number of elements in this segment's class on the training set (which is proportional to the proportion on the whole dataset, see section 4.1.2). The goal of this procedure is to compensate for the class imbalance in the dataset. We also apply early stopping: we stop the training process when the loss on the validation set did not increase for more than a fixed number of epochs (chosen to be 100, see table 3), and we keep the model which minimized the validation loss for testing. Usually, this minimum loss is reached between epoch 100 and 600.

4.2 The SHL 2018 database

The SHL 2018 dataset consists in a series of 16,310 consecutive annotated recordings of embedded sensors. Each recording is 60-seconds long, and contains data from 7 sensors, and 20 channels: three axes (x, y, z) for the accelerometer, magnetometer, gravity, linear acceleration (acceleration minus gravity), and gyrometer; one orientation quaternion (x, y, z, w), and a recording of the barometric pressure. Each signal was recorded at $100Hz$, so that one sample to classify is a set of 20 vectors of size $60 \times 100 = 6000$ points. There are 8 classes available: Still, Walk, Run, Bike, Car, Bus, Train, Subway.

The aim here is to check the general nature of our method using a different dataset, in particular the finding that a well-chosen padding and global average pooling help improve the performance and reduce the memory footprint of the networks.

4.2.1 *Preprocessing*

The SHL dataset is particular, because the triplegs of the dataset all have the same length. Padding is unnecessary when all segments are the same lengths. Hence, we choose to work with a degraded dataset: for each segment, we randomly keep between 10 and 100% of the data (uniform probability), and we pad the rest by wrapping the segments (this choice is justified experimentally in section 3.2). When we evaluate a result, we generate a new set (thereby choosing a new proportion for padding for each segment) each time we train a new network (5 times in total). This is why the standard deviations in tables 4 and 5 are so high: they account for network performance and dataset variation.

As we do not aim to overcome the state of the art with this set, we focus on the single bet sensor: the accelerom-

eter [32], [39]. We compute the norm of the accelerometer $Acc_{norm} = \sqrt{Acc_x^2 + Acc_y^2 + Acc_z^2}$ as our input signal, this will be our only preprocessing (apart from the degradation of the dataset mentioned earlier). As the accelerometer signal is not particularly noisy, we do not apply any cleaning. We do not even remove the trajectories with unlikely values, because the creators of the dataset already double-checked their annotations with videos they recorded.

In order to split the training set from the validation set, we make sure that we split the data not to have fragments of a single trip go into both the train and the val set [40]. The segments are sorted chronologically, and we take the first 20% of the segments for the validation set and the last 80% for the training set (doing so ensure both sets are approximately balanced, as the end of the dataset is severely imbalanced). We use the validation set for comparisons, but, contrary to the algorithm in section 2.2 showed, but we do not use the test set for this dataset.

4.2.2 Baseline architecture

Our baseline architecture, along with its parameters, come from [39]. The CNN uses a segment with input size $6,000 \times 1$ (as all our segments were padded back to get 6,000 points), to return a prediction among the 8 classes (Still, Walk, Run, Bike, Car, Bus, Train, Subway). Figure 3 shows the architecture, while table 3 shows the selected hyperparameters. Contrary to the GeoLife model, we only train the SHL model for 50 epochs, after which we evaluate it and return the validation score (similarly to [39]).

4.3 Experimental setup

In order to measure the performance of our models, we use the F1-score. This measure derives from the accuracy, except that it penalizes heavily the models which have imbalanced predictions. In the case of Transport Mode Detection, this is almost always the case. To make sure of this, one can look at the distributions in the SHL and GeoLife datasets (tables 2a and 2b, respectively).

To evaluate the complexity of each models, we display their number of parameter, along with the number of operations for a single inference (forward pass). As an illustration, we did include training and evaluation times, even though these results depend heavily of the device. For instance, GPUs are more optimized for heavily parallel operations, such as convolutions. The number of parameters and operations, however, is independent of the implementation, and provides an objective measure for comparison. We trained the models on a server with a Nvidia tesla V100 GPU (32 Gb of memory), cuda version was 10.2, and a 40-core Intel Xeon Gold 6230 CPU @ 2.10GHz with 190 Gb of RAM. The evaluation times were measured on a CPU, a 4-core Intel i7-7820 @ 2.90 GHz with 32 Gb of RAM. However, those running times are not absolute measures: some devices are better optimized for different types of operations. The only objective measurements are the number of parameters and operations, which do not depend on the device

For each result (F1-score, training times), we repeat the training and devaluation process 5 times, changing the seed each time. We display the average $\pm$ standard deviation.

4.4 Pooling operation

We compared three alternatives to the Flatten, namely Maximum, Mean, and Generalized Mean. As table 4 shows, on the GeoLife dataset, only the flatten step is worse than the rest, in terms of performance, computational requirements, and training and testing time. On the SHL dataset, the global max-pooling is significantly better than both the flatten step and the global average. Surprisingly, the Average pooling has a worse performance than the flatten step. The max-pooling, however, is better performance-wise and complexity-wise. Finally, it should be noted that, for both datasets, the general mean is not significantly higher than the rest, even though this operation encompasses the simple average and maximum poolings. We did not help the network by providing it with a more general expression.

As for the running times, table 4 shows us that the use of alternatives to the flatten step make the training times *longer*. The convergence is slower with these architectures, which cancels the speed gain of these architectures. As most applications rely on training the model offline (on a computer), the training time of a model is not a major concern. More interesting is the inference time: we can see that the global pooling operations are up to $X\%$ faster than the flatten step they replace.

4.5 Padding

Using the baselines architectures (fig. 3), we compared the three kinds of padding shown in fig. 2:

- *Zero-padding*, where zeros are added to the shorter segments until they reach the correct length, as in [13], [15].
- *Wrapping*, where segments are padded using their own data
- *Reflection*, which consists in padding the segments using a time-reversed version of the segment itself.

As table 5 shows, padding with zeros is particularly detrimental to the performance of our model. However, one can wonder which padding is better between wrapping and reflection. Wrapping creates discontinuities in the data, but this is the only artifact it introduces: otherwise, wrapping only uses data from the segment itself. On the other hand, reflection removes some of the meaning of the data (*i.e,* and acceleration becomes a breaking), but it does not introduce any discontinuity.

The GeoLife model has a particularity: during the random search, we contemplated adding a cleaning step using median or Savitzky-Golay filters, and it turns out these filters did not improve the performance (results not shown). This means the GeoLife network is naturally robust to noise in the data, which is why it is not affected by the discontinuities brought by wrapping the segments around. We hypothesize the same reasoning applies for the SHL network. In this case, the direction of time matters much mode than the discontinuities we brought by using wrapping.

One could wonder why zero-padding is worse than the rest. Two mutually exclusive hypotheses could be formulated to explain this phenomenon:

- A long series of zeros is interpreted as being meaningful by the model, and disturbs its *predictions*.

	Pooling	Validation F1-score	number of parameters	operations (FLOPs)	training time (min)	epochs to convergence	inference time (ms)
GeoLife	Flatten (segments of 1,024 points)	$77.0 \pm 1.6\%$	7.6×10^4	9.4×10^4	7.8 ± 0.7	113 ± 17	1.92 ± 0.13
	Generalized Mean	$80.9 \pm 1.0\%$	1.1×10^4	3.3×10^4	43.9 ± 7.3	538 ± 114	1.94 ± 0.04
	Average	$80.2 \pm 1.3\%$	1.1×10^4	3.3×10^4	78.7 ± 34.6	1161 ± 571	1.81 ± 0.04
	Maximum	$80.3 \pm 1.6\%$	1.1×10^4	3.3×10^4	16.8 ± 2.7	262 ± 66	1.85 ± 0.06
SHL	Flatten (segments of 6,000 points)	$65.2 \pm 3.2\%$	6.1×10^6	4.2×10^7	1.20 ± 0.00	50	4.74 ± 0.02
	Generalized Mean	$67.5 \pm 1.7\%$	1.7×10^4	3.0×10^7	1.15 ± 0.09	50	3.60 ± 0.43
	Average	$59.8 \pm 2.6\%$	1.7×10^4	3.0×10^7	0.97 ± 0.00	50	2.60 ± 0.02
	Maximum	$68.7 \pm 1.1\%$	1.7×10^4	3.0×10^7	1.03 ± 0.06	50	2.68 ± 0.04

TABLE 4: The effectiveness of each kind of pooling, in terms of performance, computational resources, and training and inference time. For each result, we display the average and the standard deviation, over 5 runs

Padding	GeoLife	SHL
Zero	$77.7 \pm 1.5\%$	$64.2 \pm 3.5\%$
Reflection	$80.2 \pm 1.6\%$	$63.9 \pm 1.9\%$
Wrapping	$80.3 \pm 1.6\%$	$68.7 \pm 1.1\%$

TABLE 5: The validation F1-score of each type of padding. Zero-padding is particularly detrimental to the model performance

- The model notices the long series of zeros in the learning process, and, upon seeing they are uncorrelated with the segment's classes, somehow learns to ignore the end of a segment during the training process, which cause it to miss relevant information

To know which one is true, we compute the accuracy for segments with different lengths (using the fact that the shorter the segment, the more zeros it will be padded with). If the former hypothesis was true, we would see the performance to be correlated negatively with the number of zeros. The performance would be correlated positively with the length of a segment, resulting in a decreasing curve in fig. 4.

Here, the behaviour depends on the dataset: the GeoLife curve is quite irregular, which means that the model learnt to ignore zeros at the end of segments. For the SHL model however, the performance is clearly negatively correlated with segment length: the model does not know that zeros are meaningless, and tries to interpret them, leading to higher errors when zeros are more present.

4.6 Comparison with the state of the art

We show the improvements one can achieve on the GeoLife dataset (we do not compare the SHL model against the state of the art because it relies on using a degraded dataset). When evaluating a model, a close attention must be paid to all the classes every publication worked with: in all our previous experiments, we strictly followed the recommendations of the GeoLife user guide [20], which advises to merge together the classes 'taxi' and 'car', for the number of segments in the former is deemed too small.

To know how a change in the categories employed would affect the results, one could look at the confusion matrix (fig. 5). For instance, if two modes are frequently confused with each other, merging them would cause the model's F1 to increase. This applies to the merger of {car & taxi} with bus and, to a lesser extent, to the merger of the classes train with subway. A similar reasoning could be made about the removing of the two rail modes (leaving only four modes, {walk, bike, car, bus}, as [7] do): not only are these modes no longer confused with each other, but they are not even confused with car. As one could expect, the number of classes employed correlates negatively to a model's performance.

Noticeably, [10] choose not to merge any classes, while [26] add another difficulty: not only they do not merge transport modes, but they also try to predict whether the trajectory was "fast" or "slow" (above a threshold depending on the transport mode *eg* 25 m/s for the car). A trajectory is correctly classified only if both the transport mode (7 classes) and the speed category (2 classes per transport mode) are correct. This separation makes sense in their setting, because the representation they use for a trajectory (binary heatmaps) prevents the speed information from being directly transmitted to the deep learning model. Finally, it should be noted that [13] does work with unsegmented trajectories, which makes the challenge harder.

The number of classes vastly impact a model's performance and the problem is harder with more classes. Nevertheless, to enable comparison with the state of the art despite the class difference, we retrain five models on each class combination, and evaluate them on the test set (in this section, the validation set is only used to find the loss minimum for early stopping). However, the model we chose to train still keeps the hyperparameters and the architecture we found using the 6-class problem (see section 4.1.3). The results are in table 6. We can see that our approach creates a network which is both small and effective, for our model manages to compete with the state of the art, while reducing the number of parameters by a factor 4 to 1,000, and the number of Floating-point operations (FLOPs) by a factor 10 to 100,000.

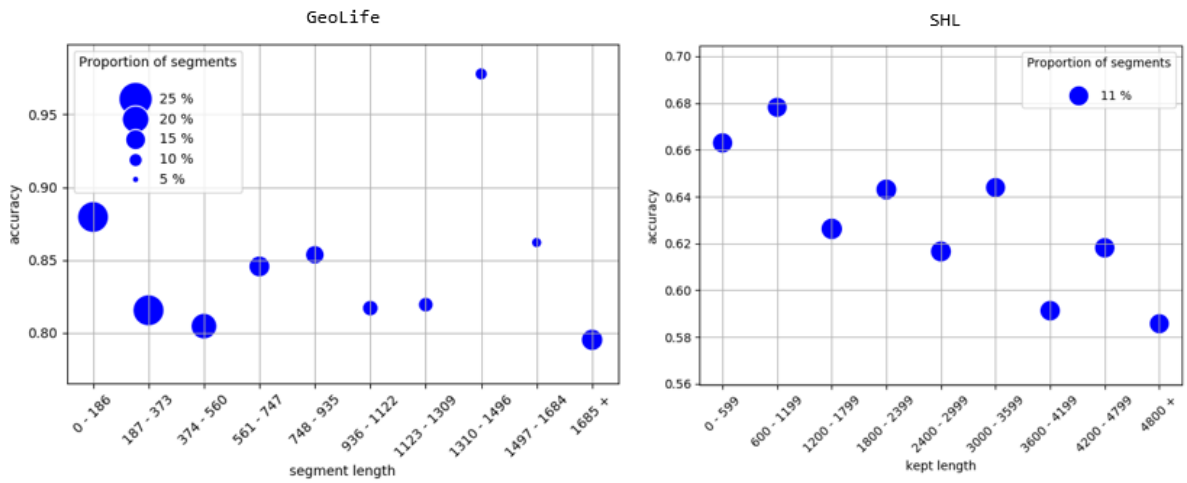

Fig. 4: The validation accuracy versus the size of each segment (the shorter the segment, the more zeros it will be padded with). Adding zeros is not particularly detrimental to the classification performance, which means the network learnt to ignore the zeros, missing potentially relevant information. Intervals bins obtained using equidistant separations between 0 and the 90-th percentile

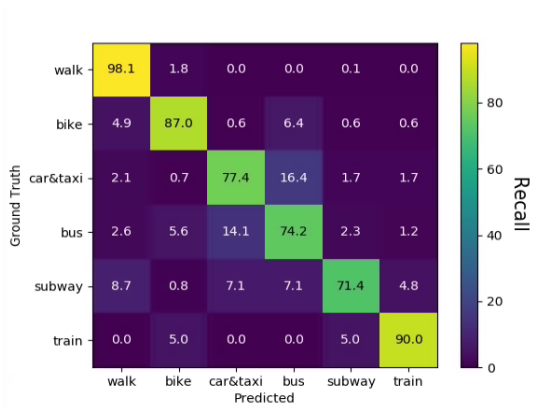

Fig. 5: The confusion matrix of the GeoLife model, on the test set. We can see merging together the "car & taxi" and "bus" classes will improve the performance considerably.

For the 4-class problem, even if the measures differ, the performance of our approach seems lower than the reported performance of the corresponding publication, but the required resources are also much lower. Besides, the corresponding work ([7]) does not say whether segments from the same trajectory can go in same or different sets. For the 5-class problem, our approach is better or equal to existing Convolutional neural networks performance-wise. If the residual CNN is worse than the LSTM implemented in [8], this publication does not precise how the train, validation, and test set were split. On the 6-class problem, our residual CNN is better than the Random Forest presented in [3]. However, even if the author does not leave enough to precisely estimate the memory and computation requirements of their approach, it is likely that our CNN has worse requirements, despite these requirements being quite low for a typical CNN. On the 7-class problem, the residual CNN is better than the autoencoder from [10], both performance and constraint-wise. We do not look at the 14-class problem because, with our encoding (we compute the speed explicitely), it is equivalent to the 7-class problem.

Memory and computation costs

Two measures matter when it comes to comparing the efficiency of different neural networks: the number of parameters (which indicates how much memory will be needed to store the network), and the number of operations required for a single inference[2] (forward pass). Looking at the time needed for an inference (as in [8]) is a good measure to evaluate if a network can be used in a client-server architecture, but it is device-dependant. Contrary to computing times, the number of parameters and operations can be computed from the architecture of a network, and allow for clear comparison (table 6).

We computed the number of operations using the code from [41]. When the input shape may vary (as in [7], [8] or the present work), we used the median length of a segment in the dataset: 200 points on the original dataset (for [7], [8]), 500 points on the interpolated one (with our proposed model). *Note*: for [7], we chose the network with a hidden layer of 100 elements instead of 300, even if it is slightly worse (94.5 % AUC instead of 94.6 %). As we aim to find a good balance between performance (F1 score, AUC, or accuracy) and efficiency, we estimated selecting a network with three times more weights for a .1 percentage point gain was unfair.

5 Conclusion

Using experiments on two realistic datasets, we showed that using adequate pooling operations could drastically reduce the number of parameters of a network without impairing its performance, which allowed us to obtain networks with the same level as the state of the art, but with 10 to 1,000 times less parameters. Also, we showed the global

2. As neural networks are typically trained offline before being used for predictions, we did not look at training costs in this part

model	metric	reported score	classes	are trajectories properly split ?	are trajectories already segmented ?	Number of parameters	Number of operations (FLOPs)
LSTM + embedding [7]	AUC	94.5 %	4	NM	yes	1.1×10^6 $(100 \times p)$	4.2×10^8 $(10,000 \times o)$
Proposed	F1	$87.1 \pm 1.1\%$	4	yes	yes	1.1×10^4 (p)	3.3×10^4 (o)
AE + CNN [13]	F1	76.4 %	5	NA	no	4.1×10^4 $(4 \times p)$	6.4×10^6 $(100 \times o)$
CNN ensemble (7 models) [15]	F1	84.0 %	5	yes	yes	$7 \times 2.6 \times 10^6$ $(1,000 \times p)$	$7 \times 1.7 \times 10^7$ $(1,000 \times o)$
LSTM + Wavelet features [8]	F1	91.9 %	5	NM	yes	8.1×10^6 $(1,000 \times p)$	7.3×10^9 $(100,000 \times o)$
Proposed	F1	$83.9 \pm 1.1\%$	5	yes	yes	1.1×10^4 (p)	3.3×10^4 (o)
Random Forests [3]	F1	71 %	6	yes	yes	50 trees	/
Proposed	F1	$81.8 \pm 0.9\%$	6	yes	yes	1.1×10^4 (p)	3.3×10^4 (o)
AE + Logistic Regression [10]	accuracy	67.9 %	7	yes	yes	2.7×10^5 $(10 \times p)$	5.2×10^5 $(10 \times o)$
Proposed	F1	$74.1 \pm 0.7\%$	7	yes	yes	1.1×10^4 (p)	3.3×10^4 (o)
CNN (DenseNet + Attention) [26]	F1	72.0 %	14	NM	yes	1.3×10^5 $(10 \times p)$	7.2×10^7 $(1,000 \times o)$

TABLE 6: Final results, on the GeoLife test set. We display a performance metric (as provided by the cited works), the estimated size (the number of weights) of each model, and the estimated number of operations required for one classification forward pass. With each of these values, we also display the ratio with the number of parameters (p) or operations (o) of our model. NM: not mentioned. NA: not applicable (trajectories are not segmented into triplegs and can have several transport modes)

pooling operation allows to get rid of one of the limits of Convolutional Neural Networks: the limitation of input size. This limit required to use an operation (the padding operation) which impaired the learning process when used as implemented in the literature. By padding with data from the segment instead of zeros, we could increase the performance.

We consider three directions for future work: we could decide to work with full, unsegmented, trajectories, in order to do both the segmentation and the classification at once. Alternatively we could try to solve the open-set problem. Currently, our network tries to classify every segment it sees, even if these segments does not belong in any class from the training set. This means segments from unknown classes (whether they belong to classes we removed from the GeoLife dataset, *e.g.*, boat, motorbike, plane, or to completely novel transport modes which might appear in the future). We could consider adding an extra class, named 'unknown' or 'other' so that our model does not try to assign a defined class to novel segments. A last direction for improvement is to use the promising attention-based architectures (*e.g.*, transformer). These models have achieved impressive results on natural language processing and even computer vision, at the cost of a massive increase in complexity. Finding a way to make them usable with a reasonable amount of parameters or operations might be the next hurdle for Transport Mode Detection.

References

[1] X. Zhu, J. Li, Z. Liu, S. Wang, and F. Yang, "Learning Transportation Annotated Mobility Profiles from GPS Data for Context-Aware Mobile Services," in *2016 IEEE International Conference on Services Computing (SCC)*, Jun. 2016, pp. 475–482.

[2] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W.-Y. Ma, "Understanding transportation modes based on GPS data for web applications," *ACM Transactions on the Web*, vol. 4, no. 1, pp. 1–36, Jan. 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1658373.1658374

[3] M. Etemad, "Transportation Modes Classification Using Feature Engineering," *arXiv:1807.10876 [cs, stat]*, Jul. 2018. [Online]. Available: http://arxiv.org/abs/1807.10876

[4] S. Dodge, R. Weibel, and E. Forootan, "Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects," *Computers, Environment and Urban Systems*, vol. 33, no. 6, pp. 419–434, Nov. 2009. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0198971509000556

[5] H. Menp, A. Lobov, and J. L. Martinez Lastra, "Travel mode estimation for multi-modal journey planner," *Transportation Research Part C: Emerging Technologies*, vol. 82, pp. 273–289, Sep. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X17301808

[6] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceeding of the 17th international conference on World Wide Web - WWW '08*. Beijing, China: ACM Press, 2008, p. 247. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1367497.1367532

[7] H. Liu and I. Lee, "End-to-end trajectory transportation mode classification using Bi-LSTM recurrent neural network," in *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, Nov. 2017, pp. 1–5.

[8] J. J. Q. Yu, "Travel Mode Identification With GPS Trajectories Using Wavelet Transform and Deep Learning," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2019.

[9] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 2261–2269. [Online]. Available: http://ieeexplore.ieee.org/document/8099726/

[10] Y. Endo, H. Toda, K. Nishida, and J. Ikedo, "Classifying spatial trajectories using representation learning," *International Journal of Data Science and Analytics*, vol. 2, no. 3-4, pp. 107–117, Dec. 2016. [Online]. Available: http://link.springer.com/10.1007/s41060-016-0014-1

[11] D. Wu, Z. Jiang, X. Xie, X. Wei, W. Yu, and R. Li, "LSTM Learning With Bayesian and Gaussian Processing for Anomaly Detection in Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5244–5253, Aug. 2020.

[12] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, and D. Wu, "Towards Real-time Cooperative Deep Inference over the Cloud and Edge End Devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 69:1–69:24, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3397315

[13] S. Dabiri, "Semi-Supervised Deep Learning Approach for Transportation Mode Identification Using GPS Trajectory Data," Ph.D. dissertation, Virginia Polytechnic Institute and State University, Virginia Polytechnic Institute and State University, 2018.

[14] "GPS.gov: GPS Accuracy." [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/

[15] S. Dabiri and K. Heaslip, "Inferring transportation modes from GPS trajectories using a convolutional neural network," *Transportation Research Part C: Emerging Technologies*, vol. 86, pp. 360–371, Jan. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X17303509

[16] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal Detection of Changepoints With a Linear Computational Cost," *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1590–1598, Dec. 2012. [Online]. Available: https://doi.org/10.1080/01621459.2012.737745

[17] Z. Xiao, Y. Wang, K. Fu, and F. Wu, "Identifying Different Transportation Modes from Trajectory Data Using Tree-Based Ensemble Classifiers," *ISPRS International Journal of Geo-Information*, vol. 6, no. 2, p. 57, Feb. 2017. [Online]. Available: http://www.mdpi.com/2220-9964/6/2/57

[18] J. Rodrguez-Echeverra, S. Gautama, and D. Ochoa, "A methodology for train trip identification in mobility campaigns based on smart-phones," in *2017 IEEE First Summer School on Smart Cities (S3C)*, Aug. 2017, pp. 141–144.

[19] M. Rezaie, "Knowledge inference from smartphone GPS data," Master's thesis, Concordia University, Apr. 2018. [Online]. Available: https://spectrum.library.concordia.ca/983733/

[20] Y. Zheng, H. Fu, X. Xie, W.-Y. Ma, and Q. Li, "GeoLife User Guide," Jan. 2019. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/User20Guide-1.2.pdf

[21] F. Asgari and S. Clemencon, "Transport Mode Detection when Fine-grained and Coarse-grained Data Meet," in *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, Sep. 2018, pp. 301–307.

[22] P. Nitsche, P. Widhalm, S. Breuss, N. Brndle, and P. Maurer, "Supporting large-scale travel surveys with smartphones A practical approach," *Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 212–221, Jun. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0968090X13002325

[23] Y. Shen, H. Dong, L. Jia, Y. Qin, F. Su, M. Wu, K. Liu, P. Li, and Z. Tian, "A Method of Traffic Travel Status Segmentation Based on Position Trajectories," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sep. 2015, pp. 2877–2882.

[24] A. Sauerlnder-Biebl, E. Brockfeld, D. Suske, and E. Melde, "Evaluation of a transport mode detection using fuzzy rules," *Transportation Research Procedia*, vol. 25, pp. 591–602, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352146517307512

[25] H. Wang, G. Liu, J. Duan, and L. Zhang, "Detecting Transportation Modes Using Deep Neural Network," *IEICE Transactions on Information and Systems*, vol. E100.D, no. 5, pp. 1132–1135, May 2017.

[26] R. Zhang, P. Xie, C. Wang, G. Liu, and S. Wan, "Classifying transportation mode and speed from trajectory data via deep multi-Scale learning," *Computer Networks*, vol. 162, p. 106861, Oct. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618314397

[27] A. Vassilev, "Reconnaissance des modes de transport par apprentissage profond partir de signaux GPS," Lille, France, 2019.

[28] X. Wang, L. Gao, P. Wang, X. Sun, and X. Liu, "Two-Stream 3-D convNet Fusion for Action Recognition in Videos With Arbitrary Size and Length," *IEEE Transactions on Multimedia*, vol. 20, no. 3, pp. 634–644, Mar. 2018.

[29] G. Tolias, R. Sicre, and H. Jgou, "Particular object retrieval with integral max-pooling of CNN activations," *arXiv:1511.05879 [cs]*, Nov. 2015, arXiv: 1511.05879. [Online]. Available: http://arxiv.org/abs/1511.05879

[30] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proceedings of the 18th international conference on World wide web - WWW '09*. Madrid, Spain: ACM Press, 2009, p. 791. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1526709.1526816

[31] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on GPS data," in *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp '08*. Seoul, Korea: ACM Press, 2008, p. 312. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1409635.1409677

[32] L. Wang, H. Gjoreskia, K. Murao, T. Okita, and D. Roggen, "Summary of the Sussex-Huawei Locomotion-Transportation Recognition Challenge," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1521–1530. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3267305.3267519

[33] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012. [Online]. Available: http://www.jmlr.org/papers/v13/bergstra12a.html

[34] M. C. Gonzlez, C. A. Hidalgo, and A.-L. Barabsi, "Understanding individual human mobility patterns," *Nature*, vol. 453, no. 7196, pp. 779–782, Jun. 2008. [Online]. Available: https://www.nature.com/articles/nature06958

[35] C. Song, Z. Qu, N. Blumm, and A.-L. Barabsi, "Limits of Predictability in Human Mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, Feb. 2010. [Online]. Available: https://science.sciencemag.org/content/327/5968/1018

[36] Y. Huang, Z. Xiao, D. Wang, H. Jiang, and D. Wu, "Exploring Individual Travel Patterns Across Private Car Trajectory Data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5036–5050, Dec. 2020.

[37] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv:1212.5701 [cs]*, Dec. 2012, arXiv: 1212.5701. [Online]. Available: http://arxiv.org/abs/1212.5701

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[39] C. Ito, X. Cao, M. Shuzo, and E. Maeda, "Application of CNN for Human Activity Recognition with FFT Spectrogram of Acceleration and Gyro Sensors," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers - UbiComp '18*. Singapore, Singapore: ACM Press, 2018, pp. 1503–1510. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3267305.3267517

[40] P. Widhalm, M. Leodolter, and N. Brndle, "Top in the Lab, Flop in the Field?: Evaluation of a Sensor-based Travel Activity Classifier with the SHL Dataset," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, ser. UbiComp '18. New York, NY, USA: ACM, 2018, pp. 1479–1487. [Online]. Available: http://doi.acm.org/10.1145/3267305.3267514

[41] V. Sovrasov, "sovrasov/flops-counter.pytorch," Jan. 2020, original-date: 2018-08-17T09:54:59Z. [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch