Terrain Identification for Time Series Data

Tsu-Hsin (Ian) Yeh (tyeh3), Chi-Jen Chien (cchien), Luis Delossantos (lgdeloss)

I. METHODOLOGY

The input files contain the **sensor reading files** from the accelerometer (x, y, z) and gyroscope (x, y, z) with a sampling rate of 40Hz, and the **terrain label files** indicating the identified terrains with a sampling rate of 10Hz. The possible labels are (0) indicates standing or walking on solid ground, (1) indicates going down the stairs, (2) indicates going up the stairs, and (3) indicates walking on the grass.

To achieve terrain identification from time-series data with the sensor input, we decide to use a recurrent neural network (RNN). More specifically, **Long Short Term Memory** networks (LSTM) are selected because it can eliminate the vanishing gradient problem in the traditional neural network.

We explore and visualize the data set to get insights about the data and remove noises that will decrease the performance of the model. Then, we select a time window to construct the input with shape (sample_size, time_window, feature_size = 6) for the LSTM network. For highly imbalanced data sets in which the majority label consists of **75%** of the total, we implement a **customized down sampling** technique to reduce the bias of the LSTM network.

Finally, we construct various LSTM networks and perform hyper-parameter tuning to find out the best performing model. For detailed information, please refer to the model training and model selection section.

Numpy, **pandas**, **sklearn**, and **matplotlib** are used for data exploration, visualization, and preprocessing. Python **random** is used for developing customized down sampling. **Keras** is used for building the LSTM networks.

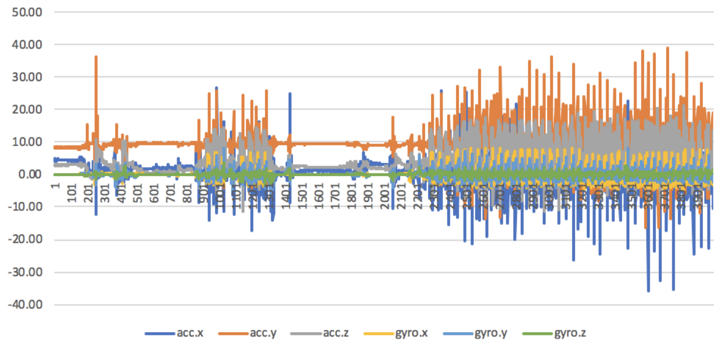
A. Data Pre-processing

While we are visualizing the time-series inputs from the sensor, we identify noises both at the beginning and at the end of the data. For example, **Figure 1** shows the first 10 seconds of the sensor reading from the subject_001_01. The y-axis indicates the magnitude of the sensor reading, and the x-axis indicates the timestamps (e.g. x=100 corresponds to timestamp at 100 $\times$ 0.025 = 2.5 seconds). As you can see, for the first 50 seconds, the sensor reading is mostly constant with the corresponding label of 0.

We remove the noises manually by selecting a beginning and ending **cut-off timestamp** based on the visualization. All the sensor reading before the **beginning cut-off timestamp** or after the **ending cut-off timestamp** are removed. We have to inspect the input sensor files one by one because the cut-off timestamp varies for each. Furthermore, we need to make sure for every 4 rows removed from the sensor reading file, 1 row is removed for the terrain label file due to the difference in sampling frequency.

Finally, we perform **mean subtraction** to center the accelerometer and gyroscope reading across different sensor input files.

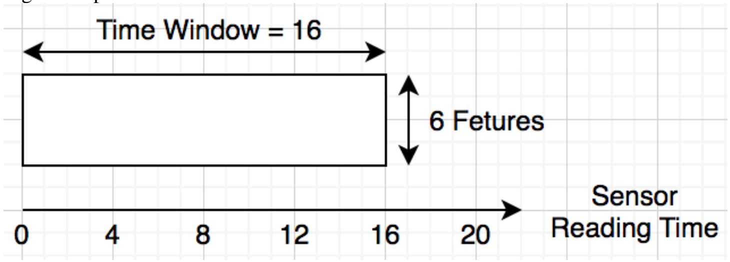
Fig. 1. Sensor Input for Subject_001_01



B. Construct Input for LSTM Network

To construct the LSTM network input, we select a **time window** for the time series sensor data so that the model can learn the temporal information. An **offset** that equals the time window is introduced so that we do not introduce noise during input construction. The time window is a hyperparameter that we will tune to uncover the best value.

Fig. 2. Input Dimension for the Model for Time Window = 16



C. Handle Imbalanced Labels in Training Data

To solve the problem of imbalanced terrain labels, we calculate the **re-sampling probability** for each labels based on the following equation:

$$resample\_prob\_x = \frac{min\_label\_count}{label\_count\_x} \times resample\_mul \tag{1}$$

During the re-sampling process, we consider each record in the time order and generate a random number between 0 and 1. If that random number is smaller than the re-sampling probability, then that record is added to the final input to the model. Otherwise, it is discarded.

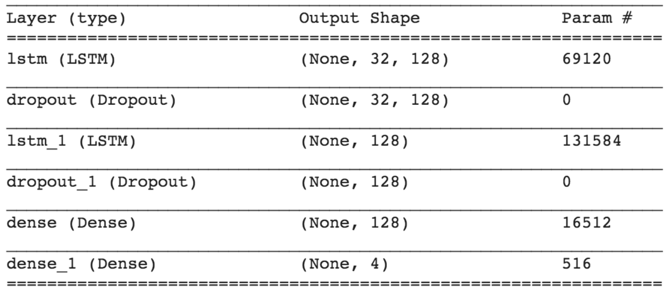
The **re-sampling multiplier** is used to control the maximum number of the non-minority labels that are allowed given the number of minority label. For example, given the re-sampling multiplier equals 1, each terrain label after re-sampling will consist of $\sim 25\%$ of the total labels. The re-sampling multiplier is a hyperparameter that we will tune to find out the optimal down-sampling magnitude for the model.

II. MODEL TRAINING AND SELECTION

A. Model Training

Provided 29 input files in total, we decided to split these files into 21 for training, 4 for validation, and 4 for testing (**70/15/15 split**). The validation files are subject_006_01, subject_006_02, subject_006_03, and subject_008_01. The testing files are subject_007_01, subject_007_02, subject_007_03, subject_007_04. The training data is the concatenation of all the training files with down-sampling while the validation data is the same without down-sampling.

Fig. 3. Baseline LSTM Network



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 32, 128)           69120
_________________________________________________________________
dropout (Dropout)            (None, 32, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 516
=================================================================
```

Our baseline model is inspired by Team 16 (Rank No.1 in Phase 1), and we perform hyperparameter tuning on this model with epoch $\in$ (12, 20, 25), time window $\in$ (16, 32, 64), and re-sampling multiplier $\in$ (1, 2, 3, 4, 5). The static hyperparameters for the model training are listed under the **Table I**.

Parameter Name	Value
Batch	64
Learning Rate	0.001
Loss Function	categorical_crossentropy
Optimizer	Adam

TABLE I
STATIC HYPERPARAMETER FOR TRAINING

After the process, we find out that the best hyperparameters combination is **(epoch = 20, time window = 32, re-sampling multiplier = 4)**. The performance is **(categorical_accuracy: 0.8950 , f1_m: 0.8949, precision_m: 0.8953, recall_m: 0.8946)** with 4 validation files (subject_006_01, subject_006_02, subject_006_03, and subject_008_01). We decide to extend our baseline model with the optimal hyperparameters we find.

B. Model Selection

We build different variations from our baseline LSTM network listed under the **TABLE II**, and the performance again the testing files are listed under **TABLE III**. The symbol **D** means Dropout and **BN** means Batch Normalization. The accuracy/precision/recall/f1-score reported are the average for 4 test files (subject_007_01, subject_007_02, subject_007_03, subject_007_04).

Model #	Structure
1	LSTM $\rightarrow$ D $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
2	(LSTM $\rightarrow$ D)x2 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
3	(LSTM $\rightarrow$ D)x3 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
4	(LSTM $\rightarrow$ BN $\rightarrow$ D)x2 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
5	(LSTM $\rightarrow$ BN $\rightarrow$ D)x3 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
6	(LSTM $\rightarrow$ BN $\rightarrow$ D)x2 $\rightarrow$ Dense(256) $\rightarrow$ Dense(4)
7	(LSTM $\rightarrow$ BN $\rightarrow$ D)x2 $\rightarrow$ Dense(128) $\rightarrow$ Dense(64) $\rightarrow$ Dense(4)

TABLE II
MODEL VARIATIONS

Model #	Accuracy	Precision	Recall	F1-Score
1	0.87	0.79	0.706	0.67
2	0.86	0.75	0.794	0.759
3	0.86	0.75	0.815	0.772
4	0.903	0.793	0.785	0.781
5	0.896	0.767	0.765	0.758
6	0.898	0.795	0.763	0.749
7	0.899	0.821	0.75	0.747

TABLE III
MODEL PERFORMANCE ON TEST FILES

Here is the list of all the observations given **TABLE III**:

- Adding neurons to the Dense layer does NOT increase the model performance
- Adding batch normalization to the model improves the overall accuracy
- Two or more LSTM layers have much better performance than a single LSTM layer

When analyzing the **confusion matrix** for each model, we find out that most of the models can identify label 1 (going down the stairs) and label 2 (going up the stairs) easily. However, some models identify label 0 better, while others identify label 3 better. Therefore, we decide to **Ensemble** 5 LSTM models for our final model. The resulting prediction is calculated based on the majority votes (**TABLE IV**).

Model #	Structure
1	(LSTM $\rightarrow$ D)x2 $\rightarrow$ Dense(128)
2	(LSTM $\rightarrow$ D)x3 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
3	(LSTM $\rightarrow$ BN $\rightarrow$ D)x2 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
4	(LSTM $\rightarrow$ BN $\rightarrow$ D)x3 $\rightarrow$ Dense(128) $\rightarrow$ Dense(4)
5	(LSTM $\rightarrow$ D)x2 $\rightarrow$ Dense(128) (re-sampling multiplier = 4.5)
Hyper-params	**Epoch = 20, Time Window = 32, re-sampling multiplier = 4**

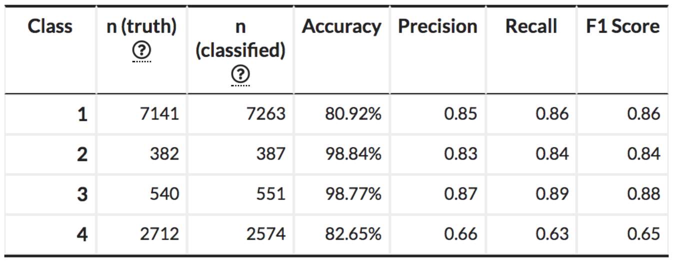
TABLE IV
LSTM MODELS FOR ENSEMBLE

III. EVALUATION

Figure 4-7 show the precision, recall, accuracy, and F1 scores for each label for each test file. The final model has average accuracy equals 89.62%, average precision equals 79.1%, average recall equals 81.8%, and average f1 score equals 79.9%. The final model has the highest f1 score compared to the ones previously built.

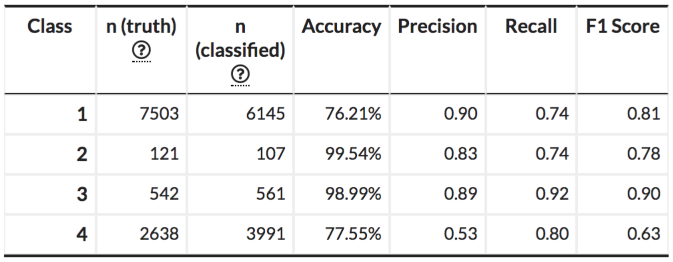
Figure 8-9 show the predictions on subject 11 and subject 12 for a selected time window. Both figures indicate that the subject walks on solid ground, goes down the stairs, walks on solid ground, and finally walks on the grass.

Fig. 4. Test result on subject_007_01



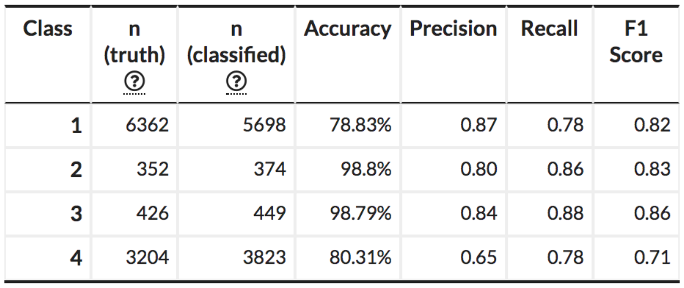
Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	7141	7263	80.92%	0.85	0.86	0.86
2	382	387	98.84%	0.83	0.84	0.84
3	540	551	98.77%	0.87	0.89	0.88
4	2712	2574	82.65%	0.66	0.63	0.65

Fig. 5. Test result on subject_007_02



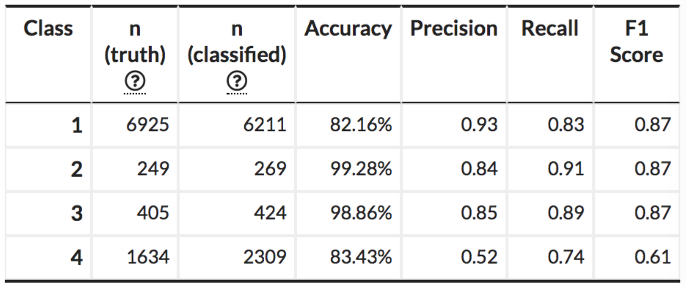
Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	7503	6145	76.21%	0.90	0.74	0.81
2	121	107	99.54%	0.83	0.74	0.78
3	542	561	98.99%	0.89	0.92	0.90
4	2638	3991	77.55%	0.53	0.80	0.63

Fig. 6. Test result on subject_007_03



Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	6362	5698	78.83%	0.87	0.78	0.82
2	352	374	98.8%	0.80	0.86	0.83
3	426	449	98.79%	0.84	0.88	0.86
4	3204	3823	80.31%	0.65	0.78	0.71

Fig. 7. Test result on subject_007_04



Class	n (truth)	n (classified)	Accuracy	Precision	Recall	F1 Score
1	6925	6211	82.16%	0.93	0.83	0.87
2	249	269	99.28%	0.84	0.91	0.87
3	405	424	98.86%	0.85	0.89	0.87
4	1634	2309	83.43%	0.52	0.74	0.61

References

[1] Dixon, Philippe Schütte, K.H. Vanwanseele, Benedicte Jacobs, Jesse Dennerlein, Jack Schiffman, Jeffrey Fournier, P-A Hu, Boyi. (2019). Machine learning algorithms can classify outdoor terrain types during running using accelerometry data. Gait Posture. 74. 10.1016/j.gaitpost.2019.09.005.

[2] Nampoothiri, M.G.H., Anand, P.S.G. Antony, R. Real time terrain identification of autonomous robots using machine learning. Int J Intell Robot Appl 4, 265–277 (2020). https://doi.org/10.1007/s41315-020-00142-3

Fig. 8. Test result on subject_011_01

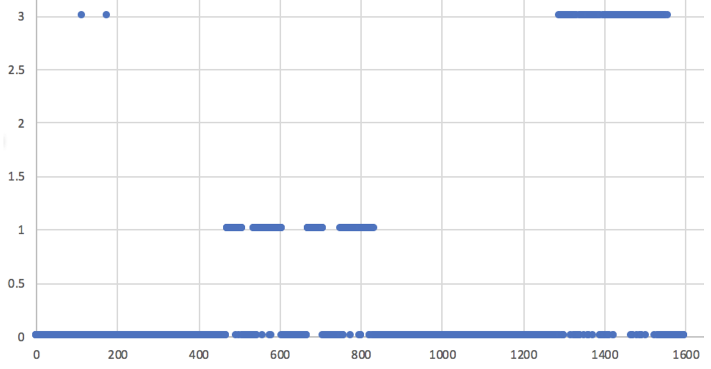
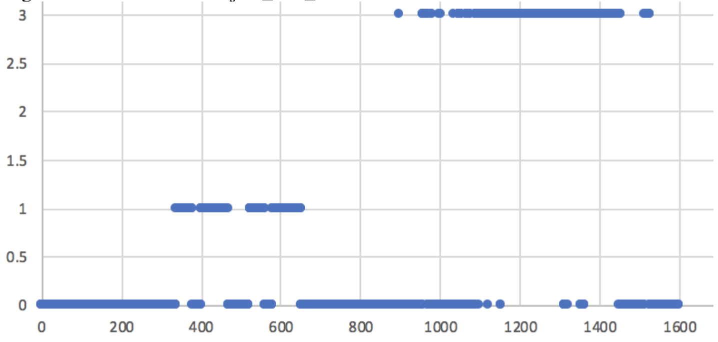


Fig. 9. Test result on subject_012_01



[3] Kelly Shen, Michael Kelly, Simon Le Cleac'h, *Terrain Classification for Off-Road Driving*. Stanford University, 2017.

[4] David Stavens and Sebastian Thrun. 2012. A self-supervised terrain roughness estimator for off-road autonomous driving. arXiv preprint arXiv:1206.6872 .

[5] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints abs/1605.02688. http://arxiv.org/abs/1605.02688.

[6] Krzysztof Walas. 2015. Terrain classification and negotiation with a walking robot. Journal of Intelligent Robotic Systems 78(3-4):401.