[Team 05] Exploring LSTM Architectures in Terrain Classification

Matthew Houk
Dept. of Electrical and Computer Engineering
mjhouk@ncsu.edu

Jeremy Park
Dept. of Computer Science
jipark@ncsu.edu

Tyrone Wu
Dept. of Computer Science
tkwu@ncsu.edu

I. Methodology

A. Data Description and Analysis

For the competition project, we are given a collection of time series data from 8 subjects, each with varying number of trials, to accurately identify the terrains in real time. In each trial, the following 6 features have been collected in 40 Hz:

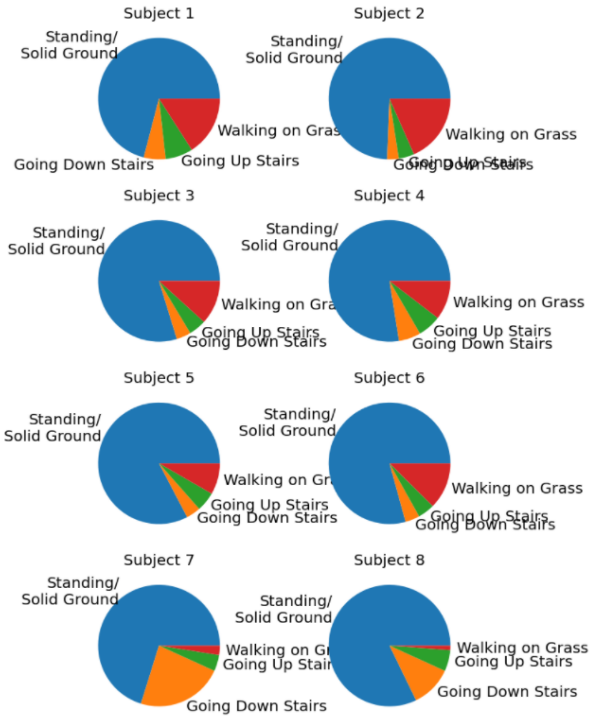
- x accelerometer
- y accelerometer
- z accelerometer
- x gyroscope
- y gyroscope
- z gyroscope

At the same time when when the 6 features above are measured, the following terrain labels are measured in 10 Hz:

- 0 - Standing or walking on solid ground
- 1 - Going down stairs
- 2 - Going up stairs
- 3 - Walking on grass

We first observed there was a heavy class imbalance in all subjects and trials. Fig. 1 depicts the distribution of the terrain labels for each subjects. To compensate for the class imbalance, we plan to apply weightings to the categorical cross-entropy loss function so that our model may pay more attention to the minority classes when training.

Fig. 1. Class Distribution of the 8 Subjects



B. Data Preparation

Our method of data preparation utilizes a fixed, sliding window that captures the sequential data along a time series trial. To include at least one full, movement step in a window, we have configured our window length to record 2.0 seconds of data. The stride of the window is set to 0.5 seconds, which gives each window a 1.5 second overlap. The shape of the single trial can be modeled as:

$$(N, 80, 6)$$

Where $(80, 6)$ represents a window from 2 sec $*$ 40 Hz and 6 features respectively, and N represents the total number of windows extracted from a trial, which can be computed as the following:

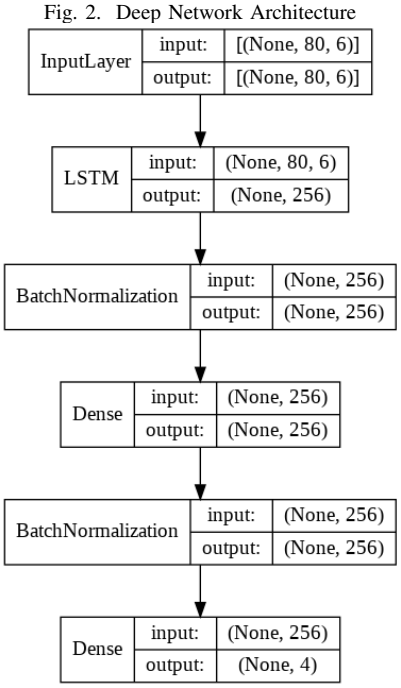
$$N = \left\lfloor \frac{trial\_duration - window}{stride} \right\rfloor$$
$$= \left\lfloor \frac{trial\_duration - 2}{0.5} \right\rfloor$$

To ensure that a window does not falsely capture sequential information between trials, our data preprocessing method is applied independently for each trial before shuffling occurs. Finally, the class associated for a window is determined by the last measured label of the window.

C. Model Architecture

Since we are dealing with time series, we felt that it was appropriate to incorporate an LSTM for the first layer of our model. As mentioned in our *Data Preparation* section, we have formatted the data so that each window preserves 80 instances of our 6 features, so that our LSTM layer may learn the sequential patterns in a window. A unit size of 256 was arbitrarily chosen to obtain a wide, initial feature analysis of the window. This is followed by a FNN of same size 256, which finally leads to a Softmax, output layer. In between each layer, Batch Normalization was incorporated to stabilize the inputs of the layers [1]. Fig. 2 depicts the result of our final architecture.

Fig. 2. Deep Network Architecture



II. Model Training and Selection

A. Model Training

Given the temporal relationships between time-series data, traditional k-fold cross validation cannot be used, as the order of the time-series data is important [2]. Therefore, we used the TimeSeriesSplit function in sklearn, where the training set is provided by the first K folds and the testing set is the (K+1)th fold [2]. Thus, the training set grows and includes all of the folds up until the K-th fold, and we chose K=5 for this submission. For each iteration up to K folds, we train a model using a 80-20 training-validation split on the training fold. We use TimeSeriesSplit to evaluate the performance across folds, and we also compare against an 80-20 split of the entire training data. Lastly, we retrained the model using all of the available data and used that model for our final prediction submissions.

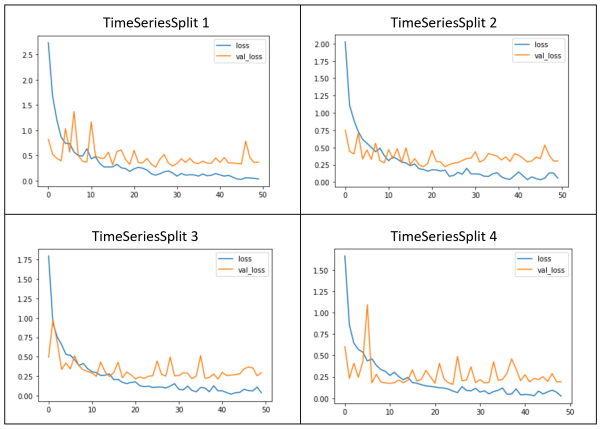
For our Network Architecture, we have trained the model against the following hyperparameters:

- Optimizer: Adam with learning rate = 0.001
- Epochs: 10
- Batch Size: 32
- Class Weights: $\{0:1, 1:4, 2:4, 3:6\}$

B. Model Selection

To begin with, an SVM, RFC, and LSTM were implemented in code using the default parameters of the Keras library. At this point models were compared for a baseline performance, before every model underwent basic hyper-parameter tuning, where values were modified and the delta accuracy was used to explore the parameter space and identify reasonably well performing values. This is acknowledged to not be the best method for hyper-parameter selection. For the final model, Fig. 3 demonstrates the model training throughout each TimeSeriesSplit. It can be observed that the model quickly converges, before gradually overfitting slightly near the end of training. With the final TimeSeriesSplit fold appearing to converge the best, likely due to having the most data to train on.

Fig. 3. KFold Training



In Table I the difference in model performance between the tuned and untuned model parameters can be observed for each model tested. It can be observed that the LSTM performance exceeded the performance of both SVM and RFC classifiers. For this reason we chose to continue with the LSTM model, as seen in Fig. 2.

TABLE I
Tuning Accuracy

Model	***Untuned***	***Tuned***
LSTM	89.23%	95.99%
SVM	67.13%	75.11%
RFC	79.53%	85.42%

III. Evaluation

The model was evaluated upon a test set of data set aside prior to the beginning of the training period. We used 80-20 training-testing split for all of the available data. The prediction results on the test set from the final model can be shown in table II.

For the final predictions, the model was trained on all of the available training data.

TABLE II
Classification Metrics

Test Set Performance					
Class	***Precision***	***Recall***	***Accuracy***	***F1 Score***	***Support***
Standing	0.98	0.94	0.94	0.96	2761
Down Stairs	0.89	0.88	0.88	0.89	164
Up Stairs	0.88	0.89	0.89	0.88	218
Grass	0.77	0.96	0.96	0.85	480
Total	0.94	0.93	0.93	0.94	3623

REFERENCES

[1] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167 [cs], Mar. 2015, Accessed: Nov. 19, 2021. [Online]. Available: http://arxiv.org/abs/1502.03167.

[2] "sklearn.model_selection.TimeSeriesSplit," scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html (accessed Oct. 15, 2021).