[Team 06] Proj-C: Terrain Identification from Time Series Data

Michał Stępień Email: mstepie@ncsu.edu Om Rajyaguru Email: orajyag@ncsu.edu Jose Molina-Melendez Email: jmmolina@ncsu.edu

I. Methodology

This is a classification task to find different terrains from time series data. The idea is to train a neural network using given data to classify which terrain an unknown data represents. Our approach to this task is to first establish a baseline model using a simple artificial neural network (ANN). For the baseline, we first pre-processed the data using the $pandas$ python libraries to build $dataframes$ and interpolated data samples. Additionally we used Synthetic Minority Oversampling Technique (SMOTE) to address data imbalance [1] leveraging the $imbalanced-learn$ Python library from $scikit-learn$. We built our final baseline model using $keras$ which consisted of six fully connected layers. Four hidden layers with 64, 128, 196, and 32 neurons per layer respectively. Rectified linear units were used as activation functions in every layer except for the output layer. The final baseline model also used a softmax as the activation function for the last layer. Lastly we introduced an Adam optimizer of learning rate 0.001 with a categorical cross entropy as our training loss function.

For our enhanced model approach we use a convolutional neural network (CNN) model. We first pre-processed the data using the *pandas* python libraries. We imported the data and stored it as *DataFrame* objects. To build the CNN model we used the $convolutional$ modules from the $keras$ toolkit libraries. In order to feed the data to the CNN, we first augmented the data by using a window slicing technique which consisted of creating windows of 4 second readings for each class label. Along with leveraging the $StandardScaler$ module from $scikit-learn$ for data standardization. This data preprocessing approach is further explained in section II-B. The final enhanced CNN model architecture was built with 2 convolutional 1-D input layers with 64 filters, kernel size of 3 and Rectified linear units as activation functions. Followed by a Dropout layer of retention rate 0.5, a Max Pooling layer of pool size 2 and a fully connected layer with 100 neurons and a Rectified linear units activation function, which was fed with flattened output of the pooling operation. The enhanced CNN model also used a softmax as the activation function for the last layer. Along with an Adam optimizer of learning rate 0.001 with a categorical cross entropy as our training loss function. Section II-B details our CNN model selection process.

II. Model Training and Selection

A. Model Training

For the baseline model we began by examining the data set and concatenate it. Since the data class labels and IMU readings were recorded at 10Hz and 40Hz respectively, we had to up sample. We applied data augmentation, by using the *numpy* library to interpolate using a nearest neighbor technique. Another challenge we faced was data imbalance, as you can see from Figure 1 the "walking/standing" class has more data labels than the other classes. We approached this challenge by applying a Synthetic Minority Oversampling Technique (SMOTE) [1]. Figure 2 shows the data after SMOTE is applied. Lastly the data set was split into a training and validation sets. We did this by leveraging the $train\_test\_split$ function from scikit-learn Machine Learning in Python, with a $test\_size = 0.3$, and a $random\_state = 42$. That left us with 70% of the data for training.

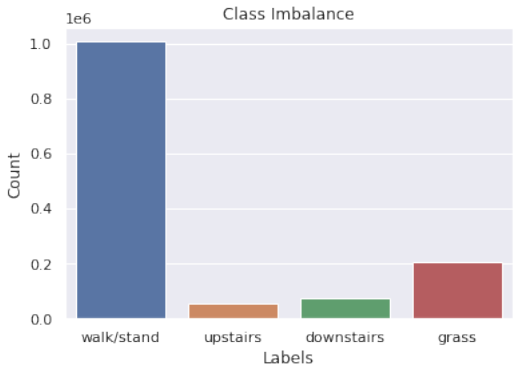

Fig. 1: Imbalanced Classes

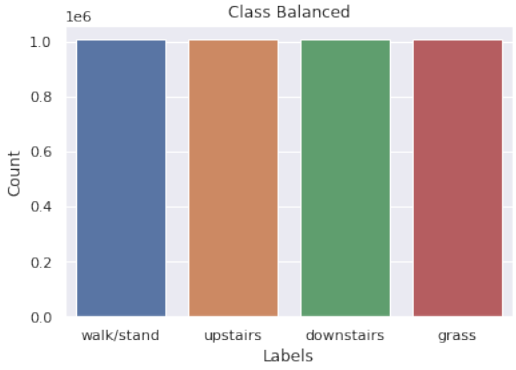

Fig. 2: Balanced Classes

In contrast, for our enhanced CNN model, we began by examining the data set, since the data class labels and IMU readings were recorded at 10Hz and 40Hz respectively, we had to organize the correct labels to readings. Figure 3 Below illustrates the challenge of features to label misalignment, notice the x-axis time stamps are not in sync for the first 230 data point from the same subject data files.

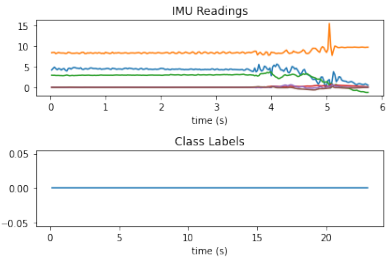

Fig. 3: Our of Sync Data Window Sample

We applied window slicing data augmentation technique to address this. Creating window of 4 second readings for each label. At the end we lose 40 or 39 labels of each file (20 from the front and either 20 or 19 from the end for each subject file, depending on the file) due to missing readings and time alignment for the window. To mitigate the problem of missing label predictions for the testing phase, we pad the predictions with 20 zeros in the front and either 20 or 19 in the end. Label 0 is vastly predominant among labels, therefore we reason that padding with values of 0 will cause the smallest error.

After applying the window slicing to each subject data file, we concatenated the data sets. We then split the data into a training and validation sets along the subjects to keep the labels localized instead of randomized. The training set amounts to 93.5% of the overall data and the validation set amounts to 6.5%. The training set consists of 27 files (all subjects up to *subject_007_03*). The validation set contains 2 files: *subject_007_04* and *subject_008_01*. Then we also stadardized the data sets in the data preprocessing stage. The idea behind $StandardScaler$ is that it will transform our data such that its distribution will have a mean value 0 and standard deviation of 1. Finally we performed a one-hot encoding $to\_categorical$ of the data class labels. The following figures 4 and 5 show a sample window of data for two of the class labels, notice the axis values where the data has been distributed.

To better understand the data we work with, we made box plots of each feature (Figure 6 and 7). The plots show that the accelerometer and gyroscope readings have a big number of outlier values. However, those outliers can be easily explained with the ratio of the classes. The data we work with is heavily imbalanced, and for the CNN model, we decided not to implement a mean to mitigate the imbalance (this is an area of future improvement). Therefore, since one class is vastly predominant, the readings of those class are considered "normal" readings, and the readings belonging to underrepresented classes are treated as anomalies. We have seen however that this does not cause any major problem for the CNN model.

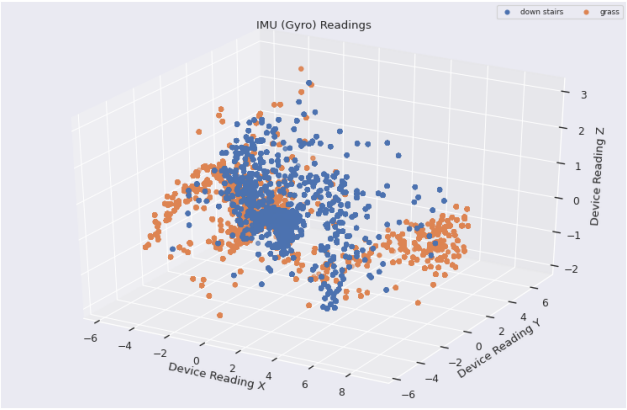

Fig. 4: Un-standardized Data Sample

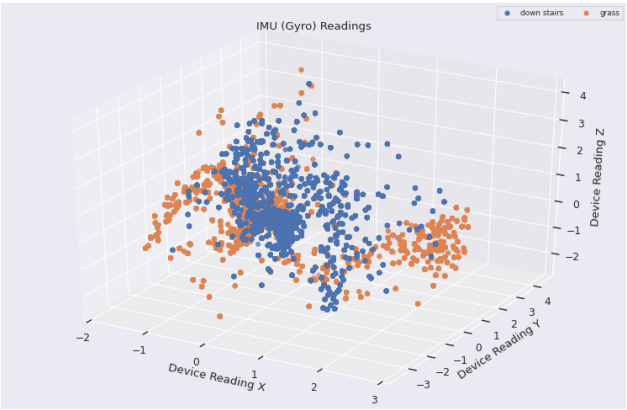

Fig. 5: Standardized Data Sample

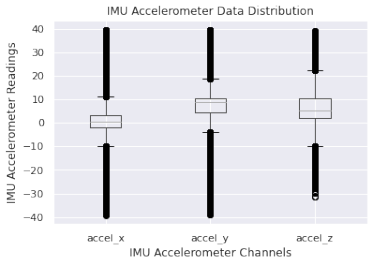

Fig. 6: Accelerometer Data Quartiles

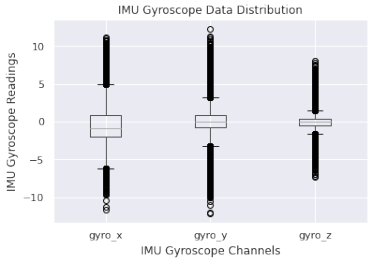

Fig. 7: Gyroscope Data Quartiles

B. Model Selection

After performing subsequent hyper-parameter tuning, the baseline ANN with Adam optimizer of learning rate equal

to 0.001 was used. The model used rectified linear unit as activation functions in every layer except for the output layer. The last layer uses softmax as activation function. The loss function used in training for the model was categorical cross entropy. The results of training and validation sessions are presented below as plot of accuracy history (Figure 8).

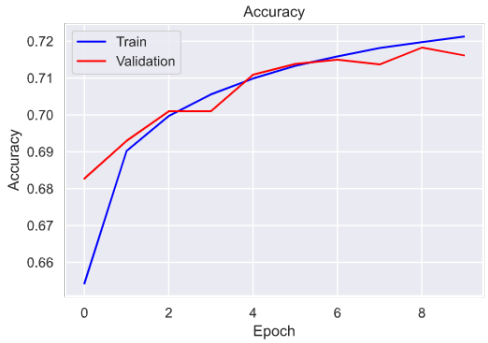

Fig. 8: Accuracy of ANN Model

As the plot shows, the peak validation accuracy of the model was 71.8%. Convolutional Neural Networks or CNNs are a type of neural network that analyze data in a window-like manner, considering all the inputs in the window as a context for calculations. The ability of CNNs to learn and automatically extract features from raw input data can be applied to time series forecasting problems like this. A sequence of observations can be treated like a one-dimensional window that a CNN model can read and distill into the most salient elements. For the enhanced CNN model we took guidance from a developed one-dimensional convolutional neural networks for time series classification on the problem of human activity recognition by Jason Brownlee [2]. As it has been mentioned in the Methodology section I, we have decided to use an CNN as the final model for the project. Prior to arriving at the final architecture described in Methodology, again we performed subsequent hyper-parameter tuning.

The CNN model consisted of 2 convolutional 1-D input layers with 64 filters, kernel size of 3 and Rectified linear units as activation functions. Followed by a Dropout layer of retention rate 0.5, a Max Pooling layer of pool size 2 and a fully connected layer with 100 neurons and a Rectified linear units activation function, which was fed with flattened output of the pooling operation. The enhanced CNN model also used a softmax as the activation function for the last layer. Along with an Adam optimizer with a categorical cross entropy as our training loss function.

The results of training and validation sessions are presented below as plots of accuracy history (Figures 9). As the plots show, the peak validation accuracy of the models is 96.95%

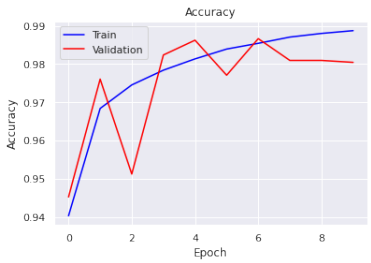

Fig. 9: Accuracy of CNN Model

Next, we advanced into hyper-parameter tuning stage. We decided to optimize the optimizer's learning rate. To do this, we have run grid search over the following learning rates for Adam optimizer: [0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001]. The mean test score and the rank given by the grid search are presented in Table I. It turns out that the best learning rate out of those we have checked is 0.001, which is the default learning rate. Based on the findings presented in this subsection, we decided to use a learning rate equal to 0.001 for the Adam optimizer.

Learning rate	0.1	0.05	0.01	0.005	**0.001**	0.0005	0.0001
Mean test score	0.7464	0.7464	0.7621	0.8502	**0.8783**	0.8678	0.8482
Rank	6	6	5	3	**1**	2	4

TABLE I: Grid search results.

III. Evaluation

The final CNN model with the default learning rate of 0.001 was used with the Adam optimizer and had the following loss and accuracy curves in Figures 10 and 11 respectively.

Fig. 10: Loss of the final CNN Model

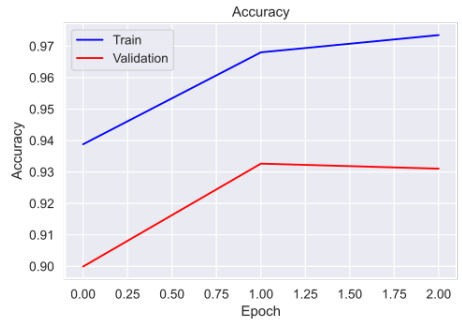

Fig. 11: Accuracy of the final CNN Model

The final CNN model also produced the following confusion matrix of the validation set, shown in Figure 12. Table II summarizes the precision, recall, accuracy and F1 scores and the average for each class. This is a very favorable comparison with the baseline model, with Figure 13 and Table III displaying the confusion matrix and various metrics for the baseline model.

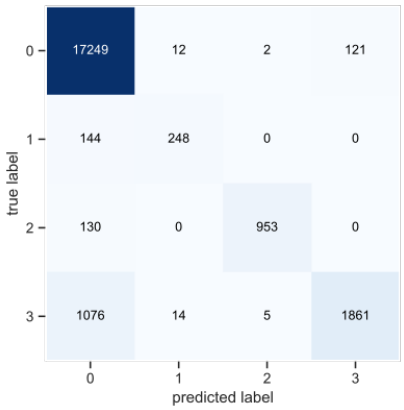

Fig. 12: Final Model Confusion Matrix

	Precision	Recall	F1-Score	Support
Class 0 (walk/stand)	0.93	0.99	0.96	17384
Class 1 (upstairs)	0.91	0.63	0.74	394
Class 2 (downstairs)	0.99	0.88	0.93	1083
Class 3 (grass)	0.94	0.63	0.75	2956
Accuracy			0.93	21815
Macro Average	0.94	0.78	0.85	21815
Weighted Average	0.93	0.93	0.93	21915

TABLE II: Final Model Metrics

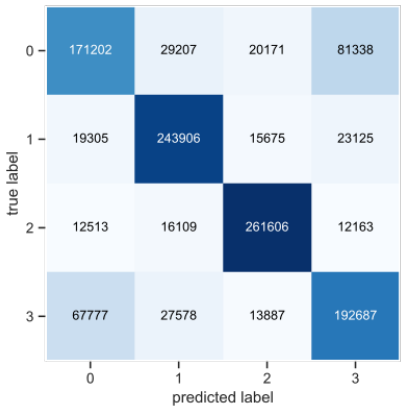

Fig. 13: Baseline Model Confusion Matrix

	Precision	Recall	F1-Score	Support
Class 0 (walk/stand)	0.63	0.57	0.60	301918
Class 1 (upstairs)	0.77	0.81	0.79	302011
Class 2 (downstairs)	0.84	0.87	0.85	302391
Class 3 (grass)	0.62	0.64	0.63	301929
Average	0.72	0.72	0.72	

TABLE III: Baseline Model Metrics

As previously stated, CNN evaluate data in a window-like manner. Clearly, the decision to implement a sliding window on the data is why this CNN model performs much better than an ANN, with higher values in all metrics. Since we did not balance the classes (no undersampling or oversampling) in the final model, we expected poor performance on the classes with less support. However, it turns out that this was not the case. The class with the highest F1-Score was indeed Class 0 (walking/standing), but other classes did not perform very poorly either. In fact, Class 2 (walking downstairs) had an F1-Score of 0.93, and it was the class with the 2nd lowest support (1083).

When we used our final model to predict the test set provided by the competition, we were able to achieve a $F1$ score of 0.83 for the predictions. Compared to our baseline, which achieved 0.43, it is a really big improvement. Figure 14 shows an illustration of our predictions on the provided test set using the final CNN model. Non-zero readings are predicted after about 35 seconds (1400 readings).

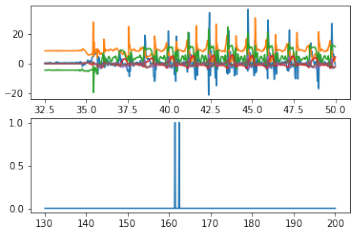

Fig. 14: Test Set Predictions

REFERENCES

[1] Alberto Fernandez, Salvador Garcia, Francisco Herrera, and Nitesh V. Chawla. SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. Journal of Artificial Intelligence Research, 61:863–905, 2018. 2

[2] Brownlee, Jason. (2018, September 21). 1D Convolutional Neural Network Models for Human Activity Recognition. Retrieved April 19, 2021, from https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/.

[3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium 24-26 April 2013.