[Team 20]

Terrain Identification from Time Series Data

Sweta Subhra Dutta
Department of ECE
North Carolina State University
Raleigh, NC
ssdatta@ncsu.edu

Shreedhar Devendra Kolhe
Department of ECE
North Carolina State University
Raleigh, NC
skolhe@ncsu.edu

Sahil Avinash Deshpande
Department of ECE
North Carolina State University
Raleigh, NC
sadeshp2@ncsu.edu

I. METHODOLOGY

The data provided to us needed augmentation with time windows because the accelerometer and gyroscope data were sampled at 40 Hz, and the labels were sampled at 10 Hz. So, our first milestone was to match the sampling rate for the input and the class labels. Secondly, we tried three different ways to augment the data named *A1, A2*, and *B*, which will be explained in the data augmentation section. *A1* and *B*, combined with different models, gave us the most promising scores. We decided not only to restrict ourselves to recurrent memory-bound networks and explored other models as well. We tried Conv1D with GRU, LSTM with Batch Normalization, and LSTM with dropout. From this, LSTM with Batch Normalization gave us the best results. We have used *Keras* APIs to implement model layer, *matplotlib* to plot the accuracy and loss graphs, *sci-kit learn metrics* to get the classification report, and for hyperparameter tuning, we have used *GridSearchCV* for some parameters, but for most, we went with random selection and conventional approaches.

II. DATA AUGMENTATION

The given training dataset consists of accelerometer and gyroscope x,y,z values sampled at 40 Hz and the classification labels sampled at 10 Hz. This data is for eight subjects. To provide better training data for the NN model, we matched the sampling frequency for the classification labels by interpolating the data using the *NumPy* library. We introduced a new column consisting of classification labels now sampled at 40Hz in the data frame.[2] As we are dealing with temporal data, we divided the feature sets into optimized time windows before feeding them into the model. As mentioned earlier, we used the following three data augmentation techniques – *A-1*, *A-2*, and *B*. In the *A-1* data augmentation technique, we uniformly distributed the data of all subjects and took a maximum of 3 datasets and a minimum of one dataset from each subject. In *A-2*, we stacked all the data for all the subjects into a single file, and in *B*, data augmentation is done for each subject individually on a local machine, and model fitting is done on the cloud. There was a considerable class imbalance in the data (class 0:251733, class 1: 13804, class 2: 18267, class 3: 51609 samples). We tried *SMOTE* (Synthetic Minority Oversampling Technique**)** for imbalanced classification [7], random over-sampler, *ADASYN* However, it was difficult to optimize the entire oversampled data set as it required a lot of computing resources. We got unsatisfactory results during the final predictions while handling class imbalance.

III. MODEL

A. Model Training:

We now have multivariate time series data with multiple input series, dependent on the same input time series. We split this data into time windows by defining a custom function. The function takes a group of 100 features and stores them in a *NumPy* array. And hence they are sent to the model as time windows of 100 seconds each.[1],[2]

For *A-1* and *A-2*, the validation set was a part of the training set, and it was split in a ratio of 80:20 using train-test split. For *B*, we kept subject eight as the test set, and the rest was used as the training set [1]. *A-1* and *B* gave the most promising results; however, we decided to go forward and use *A-1* as it gave the highest average f1-scores on the test set (see Fig. 10).

B. Model selection

We tried seven different models, and out of these, the following three models gave us promising results on final predictions: Conv1D with GRU [6],[8], LSTM with BatchNormalization, and LSTM with dropout.[3]

To optimize the parameters, firstly, we focused on time windows. We tried different time steps like 100, 120, 160, and we found that as we increase the time step above 100 with a higher precision data type like float64 or float128, we were running out of computing resources. Thus, we deduced through many trials and errors, that the optimal range for time steps should be 50 – 100. Then, we experimented with datatype precision and using float32, and we were able to save a lot of computing resources. We then included Dropout and Batch Normalization in our model to reduce overfitting and increase training speed. We tried various dropout rates, and we used a rate equal to 0.5 due to satisfactory results. After comparing the model with dropout to the model with BatchNorm, the later model gave us better results and took less time to train. We have used GridSearchCV [9] to choose the other hyperparameters like epochs, and batch size and used the trial-and-error method to get the best value for hidden layers and learning rate.

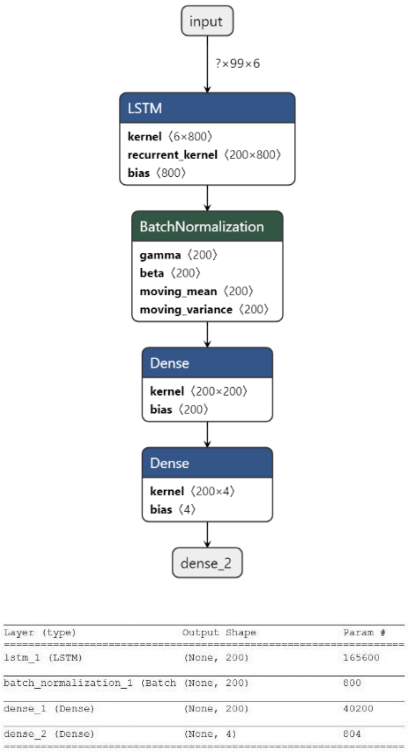
Parameters	Optimization Technique
To save RAM	
Time Windows	**100** vs 200
Datatype Precision	**float32** vs float64
Model Accuracy and Training Speed	
Optimization Technique	Dropout vs **BatchNorm**
Reaching the global minimum faster	
Learning Rate	**0.001** vs 0.01 vs 0.05

Fig 1. Optimization techniques used for the various parameters

Hyperparameter	Technique Used
Epochs	GridSearch CV
Batch Size	
Hidden Layer	Trial and Error
Dropout Probability	
Optimizers	By Convention

Fig 2. Techniques used for hyperparameter tuning

C. *Final Model Architecture*



Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 200)	165600
batch_normalization_1 (Batch	(None, 200)	800
dense_1 (Dense)	(None, 200)	40200
dense_2 (Dense)	(None, 4)	804

Fig 3. Final model architecture and summary

We have used four layers for our final model; the first one is the LSTM layer with 200 hidden units [4] fed with an input shape of (timesteps x number of features), and we set the batch size to 32. The second layer is a BatchNormalization layer. The third layer is the dense layer having hidden units equal to 100 and having a ReLU activation function. In the prediction layer, we have used SoftMax as our activation function.[4]

IV. EVALUATION

For the evaluation part, we predicted the classes for the test subjects 9,10,11, and 12 separately. The following plots show the training and validation loss and accuracy plots for each of the three models that gave promising results during the final predictions. Before getting the F1 score, we have down sampled our predictions to match the initial sampling frequency for the class labels using the NumPy library.

In this section, we will discuss the performance of our final model (LSTM + BatchNorm) both on the validation and the test data. We never used the given test data during learning. With a learning rate of 0.001, the curve converges to a global minimum at an optimal rate. We got an average training accuracy > 98% for all three models at each epoch and an average validation accuracy >94%.

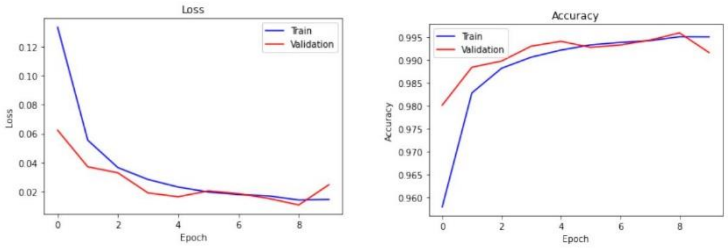


Fig. 4. LSTM + Dropout Model Loss and Accuracy Plots

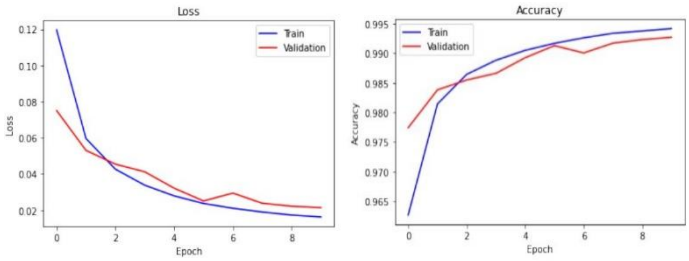


Fig. 5. CNN + GRU Model Loss and Accuracy Plots

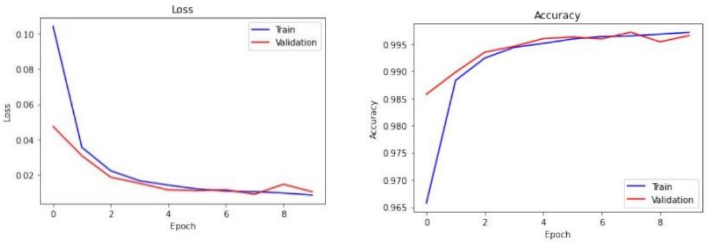


Fig. 6. LSTM + BatchNorm (Final Model) Model LSTM Loss and Accuracy Plots

Following are the classification metrics for the 3 models.[5]

labels	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	146238
class 1	0.99	0.99	0.99	7280
class 2	0.99	0.99	0.99	10053
class 3	1.00	0.99	0.99	27916
accuracy			1.00	191487
macro avg	0.99	0.99	0.99	191487
weighted avg	1.00	1.00	1.00	191487

Fig. 7. CNN + GRU model classification metrics

	precision	recall	f1-score	support
class 0	0.99	1.00	0.99	146191
class 1	0.99	0.98	0.98	7502
class 2	0.98	0.99	0.99	10160
class 3	0.99	0.98	0.98	27634
accuracy			0.99	191487
macro avg	0.99	0.98	0.99	191487
weighted avg	0.99	0.99	0.99	191487

Fig. 8. LSTM + Dropout model classification metrics

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	146546
class 1	0.99	1.00	0.99	7323
class 2	0.99	0.99	0.99	10042
class 3	1.00	0.99	1.00	27576
accuracy			1.00	191487
macro avg	1.00	0.99	0.99	191487
weighted avg	1.00	1.00	1.00	191487

Fig. 9. LSTM + BatchNorm (Final Model) model classification metrics

Average Final F1 Score		
Model Used	**Validation set f1-score (A1)**	**Final Predictions f1-score**
CNN + GRU	0.975	0.752
LSTM + Dropout	0.985	0.812
LSTM + BatchNorm	0.9925	0.858

Fig. 10. Average f1-scores

Following are the classification histograms for test-subjects 9, 10, 11 and 12 using final model LSTM + BatchNorm and data augmentation type *A-1*.

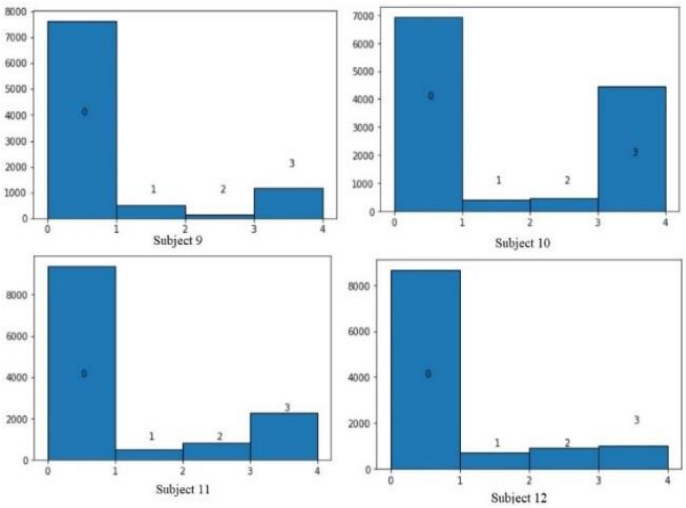


Fig. 11. Test set predictions

V. CONCLUSION

Theoretically, from all the models we have tried, Conv1D with GRU should have given the best results. As this model has been used as the industrial standard for accelerometer and gyroscope data from smart phones for human activity recognition. But though it gave promising results for IMU data, LSTM with BatchNorm gave a better final prediction accuracy in this case. In the future, we may try to understand and visualize the IMU data better and go for more complex data augmentation types to get better accuracy scores. Also, the total number of train data was only given for eight subjects, which was too little for the model to become robust. Hence, more data is needed to make the model provide much better accuracy.

REFERENCES

[1] Brownlee, J. (n.d.). Deep Learning for Time Series Forecasting (V1.4 ed.).

[2] Bryan Lim, and Stefan Zohren. (2020). Time Series Forecasting With Deep Learning: A Survey.

[3] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

[4] https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-ke

[5] https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, & Yoshua Bengio. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

[7] Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357.

[8] Subhrajit Roy, Isabell Kiral-Kornek, & Stefan Harrer. (2018). ChronoNet: A Deep Recurrent Neural Network for Abnormal EEG Identification.

[9] https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-lear