Body-Rocking Behavior Recognition

Anshul Atriek *(aatriek),* Ronil Pancholia *(rpancho)* and Vincent Tompkins *(vrtompki)*

I. Methodology

This project proposes a machine learning (ML) classification model for the detection of body-rocking behavior in blind subjects using inertial measurements from a wearable system. Two sensors, one on a wrist and another on an arm, are used to record motion using accelerometers and gyroscopes. The training as well the test data consists of several sessions, each about 1 to 2 hours long and sampled at 50 Hz, with annotations about when the behavior was observed.

Having witnessed first-hand the limitations of several rudimentary ML techniques such as feedforward networks and support vector machines, this study implements a Convolutional Neural Network (CNN) to obtain the desired performance. We introduce and describe the key components of the proposed approach in this section. Section II explains how the optimal parameters and hyper-parameters for our model have been chosen. Section III evaluates the performance of the adopted approach.

To extract the time-domain frames from the raw data, the latter is segmented with a sliding window of a fixed width = 3 seconds, with an overlap of 50%. Since the raw data has been recorded at 50 Hz sampling frequency, this essentially segments the data into vectors of 150 points each. We treated the size of the sliding window as a hyper-parameter and 3 seconds seemed to work well compared to other window sizes evaluated. The *mode* metric is used to calculate the corresponding annotations or labels.

Next, since the data is made up of multiple time-series channels, each channel needs to be normalized separately to force them all to a common scale. The *standard score* is used, thus all values (within a channel) are scaled using the formula $x_{standardized} = (x - \bar{x})/\sigma$. Here, x is the data from a channel, $\bar{x}$ is the mean of the values in the channel, and σ is the standard deviation.

Now that the data is in a suitable form, we implement a 1D convolutional model to learn the characteristic data features, and subsequently make intelligent classification predictions given a similar set of features. A CNN is typically composed of a mix of convolutional, pooling and activation layers, followed by one or more fully-connected layers. A similar architecture has been used in similar studies [1]. We build a sequential network with three convolutional layers. As seen earlier, an input vector of length 150 is employed. Thus, the first convolutional layer filters the 12-Channel 150 * 1 input signal (i.e., 150 * 1 * 12 input matrix) with 96 kernels of size 9 * 1. The second convolutional layer takes as its input the output from the first convolutional layer and filters it with 192 kernels of size 7 * 1. Similarly, the third convolutional layer filters its input with 300 kernels of size 5 * 1. No pooling layers are present in the architecture; instead a stride of step 3 is used in each convolutional layer. Also, the network uses rectified linear units (ReLU) as the activation function for each of the convolutional layers. Finally, this sequence of layers is followed by two fully-connected layers - the former vectorizes the output of the third convolutional layer into a 500-dimensional feature vector; and the latter has only two neurons - one for each possible output class - and is subsequently followed by a softmax layer. We also add a dropout layer after the first fully-connected layer. Table I sums up the network structure for our final model.

TABLE I: CNN Model

Layer	**Input Channels**	**Output Channels**	**Kernel Size**
Convolutional L1	12 channels	96 channels	9x1
Convolutional L2	96 channels	192 channels	7x1
Convolutional L3	192 channels	300 channels	3x1
Fully Connected L1	Inputs: 1200	Outputs: 500	-
Dropout (p=0.5)	-	-	-
Fully Connected L2	Inputs: 500	Outputs: 2	-

As discussed earlier, the input while training is segmented in vectors of length 3 seconds each with an overlap of 50%. This enables our model to discern interesting features from these short segments, instead of only from single discrete data points taken one at a time. The location of the features within these segments is no longer important - rather, the model looks at a bigger picture. Similarly, during testing, the input is segmented into vectors of length 3 seconds with an overlap of 99.33%. Note that even though the CNN model takes in the same input vector of length 150, the overlap of 149 points means that each data point can now be predicted explicitly by the model.

II. Model Training and Hyper-Parameter Selection

The dataset we used has detections for six sessions. We have done holdout validation and testing by splitting the data in to training, testing and validation sets. Since, we do not want to leak information from a session, we do not use time windows from the same session in both training and validation. Hence, we divided the total data into three splits with 2 sessions for training, 2 sessions for validation and 2 sessions for testing. The choice of which sessions go in training/validation/testing sets was random and the model performance did not change much with different choices of sessions in either of the three sets.

As mentioned earlier, we used a time window of 3 seconds with an overlap of 50%. We treated the duration of time window and the overlap percentage as hyper-parameters. We experimented with overlap sizes of 25%, 50% and 75%. Results obtained with higher overlap were not significantly

TABLE II: Hyperparameters

Hyper-parameter	Range of Values	Final Value
No. of Convolutional Layers	2, 3, 5	3
Learning Rate	$10^{-4}, 10^{-5}, 10^{-6}$	10^{-5}
Epochs (with early stopping)	50	10
Batch Size	64	64
Time Window Duration	1, 3, 5	3
Overlap Percentage	25, 50, 75	50

better and required longer training times. So, we have used 50% overlap. We experimented using time windows of 1 second, 3 seconds and 5 seconds. The 3 second long time windows seemed to perform well.

For training the model, we have used early stopping with a patience of 10 epochs. If the validation loss does not improve for 10 consecutive epochs, we halt the training. We only save those models for which the validation loss improves. As shown in Figure 1 and Figure 2, we can see that training converges after around 10 epochs and after that, the model starts to overfit. We experimented with different number of convolutional layers and 3 layers seemed optimum. A model with just 2 layers was underfitting in our experiments. Adding more layers did not seem to provide any benefits either. The model seems complex enough as the training accuracy tends to reach 99% with just 20 epochs. Table II provides a comprehensive list of the hyper-parameters we have used.

Once we have tuned our model hyperparameters using 2-2-2 holdout validation, we train the final model with all six sessions in the training set and use it for further predictions on unseen data.

III. Evaluation

Model performance was evaluated using four metrics, precision, recall, the F1 score, and the model accuracy. The final model yielded a test accuracy of 81%. The positive class precision and recall values of 82% and 77%, respectively, were obtained. Precision is a metric used to evaluate the effects of false positives, or predicting a movement of interest when none actually occurred. Conversely, recall is a valid assessment of the model performance when the penalty for false negatives is high. That is, if there was a target action but none was predicted. The the disparity between the precision and recall (approximately 5%) suggests that the model penalizes false positives more so than false negatives. This maybe ideal if for more general cases where actions detected by sensor and predicted by the model maybe indistinguishable from rocking behavior but they are in fact not. As an added measure of evaluation of the model performance, the F1 score was taken to provide a comprehensive overview of the models ability to predict. These results were captured in Table III.

TABLE III: Evaluation results

Class	Precision	Recall	F1 Score
NOT detected ’0’	0.81	0.85	0.83
Detected ’1’	0.82	0.77	0.80
Weighted Average	0.81	0.81	0.81
Overall Accuracy		0.81	

As stated before the results captured in the previous table demonstrates the model’s predilection for minimizing the impact of the false negatives and emphasizes importance of true positives.

In Figure 1 it is apparent that the models training accuracy increases over the number of epochs, whereas the test accuracy tends to fluctuate about the 81% mark. Although training peaks around 99%, the test never exceeds a value greater than 85%. This may be indicative of a generalizable model, that may suffer from small inaccuracies due to an ineffective means of capturing more latent spatial features in conjunction with the latent temporal aspects of using the CNN model. In contrast to the accuracy curves in Figure 1, the loss trend of the both datasets decay as the epochs increase, as shown in Figure 2. The prediction and ground-truth for detections were tracked over time for Session06 (Figure 3) for a duration of 30 seconds, and from these results, we can see that the model has trouble disparaging sustained spontaneous changes in activity. For instance, during the time interval between second 9 and second 15 as shown in Figure 4(a), the predictions tend to favor detection ”1” until a few seconds later. This discrepancy may be resultant of the windowing function and the sampling of the time series. The overlap may be superfluous, or too small, to the extent that the reconstruction of the signal is

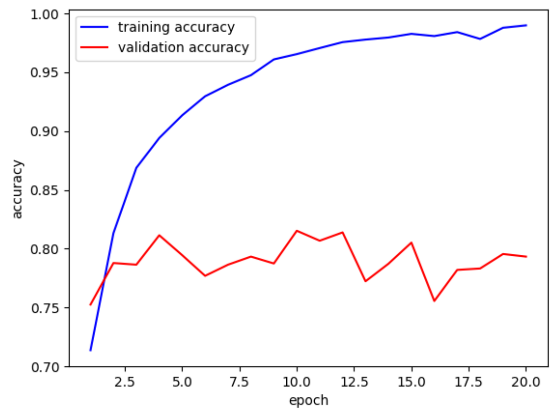


Fig. 1: Training and validation accuracy vs epoch

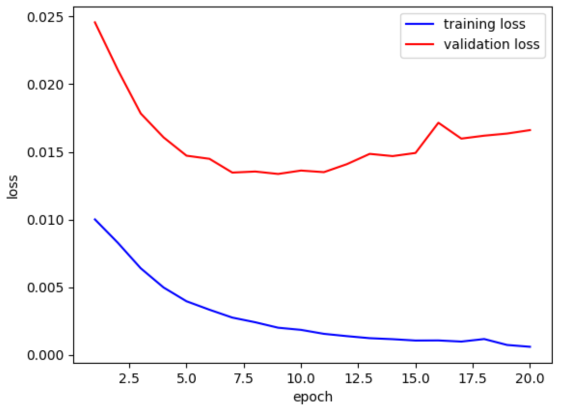


Fig. 2: Training and validation loss vs epoch

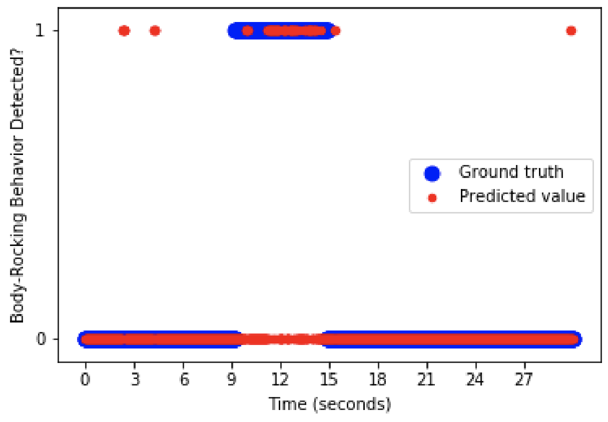
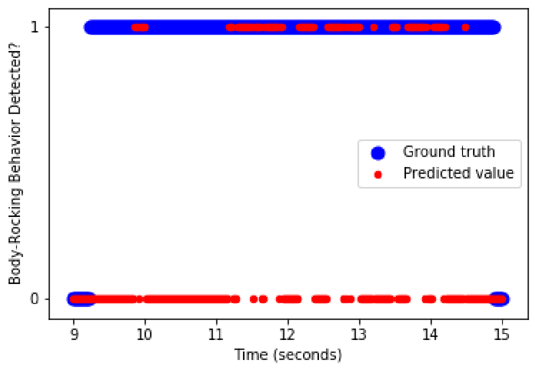
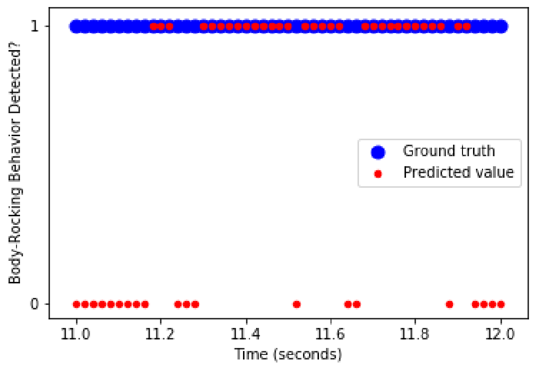


Fig. 3: Training and validation loss vs epoch

inadequate. This point is illustrated in Figure 4(b). This adds to the notion that the model penalizes false positives more so than false negatives. These results present a framework



(a) 9-15 second interval



(b) 11-12 second interval

Fig. 4: (a) Prediction and ground-truth during time interval 9 to 15 seconds; (b) Prediction and ground-truth during time interval 11 to 12 seconds.

for using deep learning to detect and track rocking behavior in blind subjects. The CNN tends to penalize false positives which may be useful for identifying true rocking-behavior, but accuracy could be improved. Future work is needed to better understand and decouple temporal and spatial learning of the algorithm, and propagation of these learned features to improve time-series predictions. Possible enhancements include investigating recurrent neural networks, or implication of traditional statistical methods together with deep learning methods.

REFERENCES

[1] Lamyaa Sadouk, Taoufiq Gadi, and El Hassan Essoufi, A Novel Deep Learning Approach for Recognizing Stereotypical Motor Movements within and across Subjects on the Autism Spectrum Disorder, Computational Intelligence and Neuroscience, vol. 2018, Article ID 7186762, 16 pages, 2018. https://doi.org/10.1155/2018/7186762.

[2] Nastaran Mohammadian Rad, Seyed Mostafa Kia, Calogero Zarbo, Twan van Laarhoven, Giuseppe Jurman, Paola Venuti, Elena Marchiori, and Cesare Furlanello. 2018. Deep learning for automatic stereotypical motor movement detection using wearable sensors in autism spectrum disorders. Signal Process. 144, C (March 2018), 180-191. DOI: https://doi.org/10.1016/j.sigpro.2017.10.011