G33: Terrain Identification from Time Series Data using 1-D Autoencoders

Akhil Bommadevara
abommad@ncsu.edu

Ishan Bhatt
ivbhatt@ncsu.edu

Aishwarya Seth
aseth@ncsu.edu

Code(NCSU access): https://github.ncsu.edu/ivbhatt/DLNN-ProjC1

I. METHODOLOGY.

There are two competent Deep Learning architectures for sequence-to-sequence classification. LSTMs[1] and 1-D Convolutional Autoencoders[2]. In this paper, **we have decided to experiment with 1-D Convolutional Autoencoders using Keras and other Python packages.**

1-D Convolutional Autoencoders are very similar to autoencoders[3] except that all Convolutional and MaxPooling/UpSampling operations are done in 1-D (i.e. the filter slides only in one direction).

Auto-encoders typically have a bunch of CONV-blocks which consist of a ReLU-activated CONV layer followed by a Dropout layer (optional) followed by a Batch Normalization layer (optional) followed by a MaxPooling layer (which typically shrinks by a factor of 2 all but the last dimension of the input_tensor). Number of CONV filters are typically doubled in each successive CONV-block. This standard was first set by the VGG-architectures[4]

Following the CONV-blocks, we usually see an equal number of DECONV-blocks with each CONV block having typically the same number of features as its corresponding CONV-block. MaxPooling is replaced by UpSampling (which typically expands by a factor of 2 all but the last dimension of the input_tensor).

After the DECONV-blocks, we typically see some additional CONV layers with the last of them being softmax-activated and having filters equal to the number of output_classes. Therefore, we typically find the output_shape = [*batch_size, sequence_length, number_classes*].

Layer (type)	Output Shape	Param #
input_11 (InputLayer)	[(None, 160, 6)]	0
conv1d_70 (Conv1D)	(None, 160, 25)	475
dropout_60 (Dropout)	(None, 160, 25)	0
max_pooling1d_30 (MaxPooling	(None, 80, 25)	0
conv1d_71 (Conv1D)	(None, 80, 50)	3800
dropout_61 (Dropout)	(None, 80, 50)	0
max_pooling1d_31 (MaxPooling	(None, 40, 50)	0
conv1d_72 (Conv1D)	(None, 40, 100)	15100
dropout_62 (Dropout)	(None, 40, 100)	0
max_pooling1d_32 (MaxPooling	(None, 20, 100)	0
conv1d_73 (Conv1D)	(None, 20, 100)	30100
dropout_63 (Dropout)	(None, 20, 100)	0
up_sampling1d_30 (UpSampling	(None, 40, 100)	0
conv1d_74 (Conv1D)	(None, 40, 50)	15050
dropout_64 (Dropout)	(None, 40, 50)	0
up_sampling1d_31 (UpSampling	(None, 80, 50)	0
conv1d_75 (Conv1D)	(None, 80, 25)	3775
dropout_65 (Dropout)	(None, 80, 25)	0
up_sampling1d_32 (UpSampling	(None, 160, 25)	0
conv1d_76 (Conv1D)	(None, 160, 4)	304

Total params: 68,604
Trainable params: 68,604
Non-trainable params: 0

Figure 1: Ensemble unit summary
(21 of these are used to get the final prediction)

We decided to build an ensemble system of weaker models to remove the occasional misclassifications & obtain a more stable and smooth results. We built an ensemble of 21 classifiers. We chose hyperparameters that produce models with a slightly less number of trainable parameters than our best-performing stand-alone model. This is in accordance with the bagging strategy for building ensemble systems[5]. Bagging, in short, is a strategy where we use small random subsets of our dataset to train multiple smaller models and let them over-fit to a subset of the dataset. At prediction-time, we poll the results from all such classifiers to produce a final prediction.

Figure 1 shows the model.summary() of the weak-model we are using for our ensemble system.

II. MODEL TRAINING AND SELECTION

A. Model Training

The provided data had two major issues: no synchronization between x_attributes and y_labels; class-imbalance. In order to rectify this, we performed two operations on the dataset.

First, we have to synchronize the attributes and labels. We noticed that y (10Hz) was sampled at a slower rate than x (40 Hz). This meant, we would have multiple time-stamps of the attribute-sequence corresponding to a single time-stamp of the label-sequence. As demonstrated in figure 2, our idea was simple: **While assigning a label to the attribute sequence, we simply broadcast the labels to attribute-sequence such that each attribute-x is matched with the closest possible (in time) label-y. While down-sampling the predictions-y (which are generated at 40Hz) to 10 Hz output-sequence, we simply take the vote (statistical mode) of all the predictions-y's that fall under that particular time-frame .**

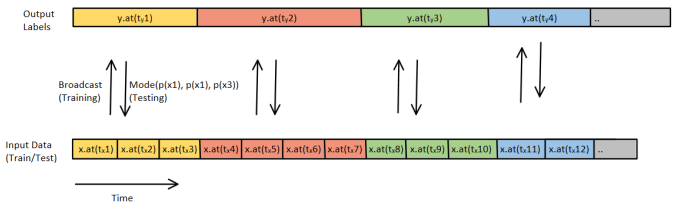

Figure 2: Synchronizing data with labels
Time scales of x(tx) and y(ty) are different. Each cell represents 1 unit of time on their respective scales. The function p(x) = prediction(x) and xi = x.at(txi)

Second, we address the class imbalance present in the dataset. After we synchronize the data, we perform some exploratory analysis on it. We find that class_0 is over-represented while all other classes are under-represented. Distribution of the classes are shown in Figure 3.

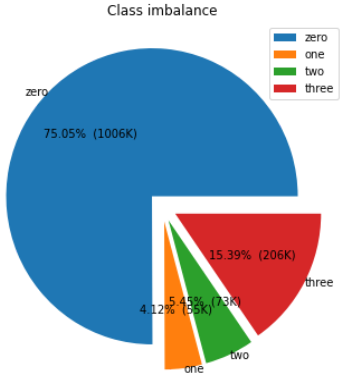

Figure 3: Class-imbalance in the original dataset

We over-sample the sequences where the under-represented classes are present by a factor proportional to their under-representation. In other words, class_1 is over-sampled at a rate 3 times of the rate with which class_3 is over-sampled (because, in the original dataset, class_1 is 3 times more underrepresented than class_3). Since we cannot sample individual instances from the data-set (we must sample sequences), we can only aim to have class distribution approximately equal (and not exactly equal). Figure 4 shows class distributions after variable-sampling.

After some experimentation, we realize that it is much more difficult for a model to differentiate between class_0 (walking on solid ground) and class_3 (walking on grass). For this reason, we purposefully oversample frames representing those two classes. We aim to have a 30-20-20-30 distribution for the classes 0-1-2-3 respectively.

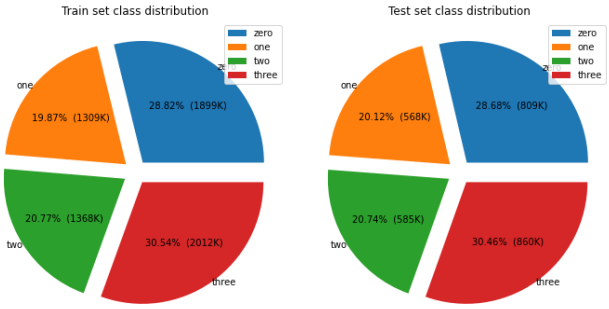

Figure 4: Class representations in train_set(left) and test_set(right)

We also perform data-augmentation on the entire training dataset. We generate five files out of each file present in the original dataset. We produce the following versions of each file. We add a small random-noise between 0-5% of the value to each data-point:

- Fast (10% faster than the original)
- m_Fast (5% faster than the original)
- Original
- m_Slow (5% slower than the original)
- Slow (10% slower than the original)

Figure 5 illustrates how this scheme of data-augmentation produces high-quality augmented data.

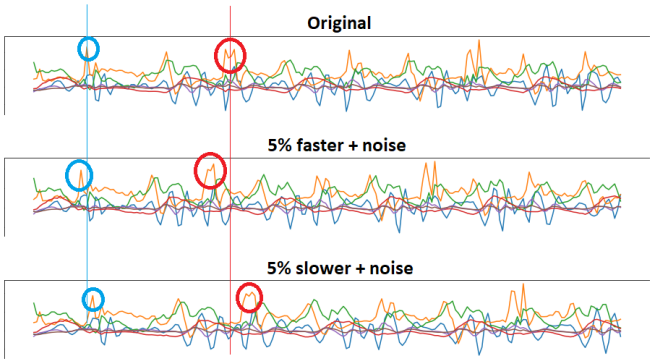

Figure 5: Time-series data-augmentation (shrink)

After this we use about **70% of the data for training and the rest 30% is reserved for testing.**

After noise is added, we perform **Z-score normalization attribute-wise.** In other words, we normalize all values of gyro_x (say) present in all sequences present in the training set in one batch. Figure 6 shows histogram of an attribute after normalization. We can see 160 small bars per value showing its presence at all 160 positions in the sequence.

We see that the values are not perfectly normalized between the range (-3, +3). This is expected because the input data to normalization is not from a normal distribution. There are outliers in the data that exist outside the range: *mean∓3*std_dev* and they remain outliers even after normalization. This is okay [6]

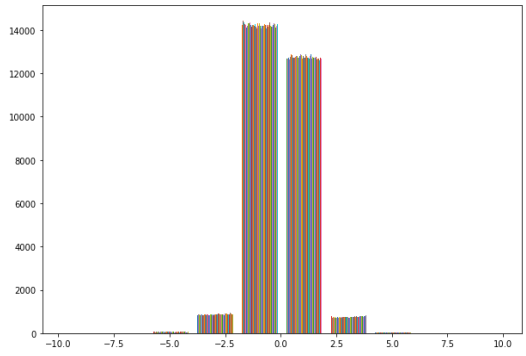

Figure 6: Histogram of a typical attribute (train_set) after Z-score normalization

B. Model Selection

As per the architecture description in section I, we notice that we need to find out optimum values for the following hyper-parameters: depth (number of CONV/DECONV-block), dropout-rate, number of CONV-filters in the first CONV-layer, BatchNormalization and BatchSize. We perform a grid-search for hyper-parameters and the results are presented in Table 1. We use categorical_crossentropy loss to gauge the performance of the model.

After our examples, we find that depth=3, dropout-rate=0.4 and start_filters=32 gives the best result (ORANGE). We find that adding BatchNormalization does not improve the model-performance. On the contrary, we notice that adding BatchNormalization worsens the model performance.Finally, we attempted to vary the batch_size while training, which did not improve speed or performance significantly.

We use a model with slightly less number of trainable parameters to build our ensemble system. Since using 16 filters made the model unable to fit properly on the training data, we tried the next natural choice of 25 (YELLOW).

Table 1: Results of grid-search for hyper-parameters

depth	dropout	start_ filters	other	train _loss	test_ loss
3	0.3	16		0.37	0.41
3	0.4	16		0.36	0.42
4	0.3	16		0.29	0.34
4	0.4	16		0.32	0.36
3	0.3	32		0.25	0.32
3	**0.4**	**32**		**0.24**	**0.31**
3	0.4	25		0.25	0.32
4	0.3	16		0.18	0.27
4	0.4	16		0.22	0.29
best_combination from above			Include BatchNormalization after Dropout	0.24	0.34
0.4	**3**	16	batchSize = 256 (instead of 512 everywhere else)	0.24	0.31
0.5	3	16	no	0.27	0.36

To address the problem mentioned in section II-A (about difficulty in differentiating class 0 from class 3), we also implement a custom loss function. **Our custom loss function is weighted categorical cross-entropy** with weights for classes 0-1-2-3 are 3-1-1-3. In short, we weigh a misclassification among classes 0 and 3 three times as much as a misclassification among other classes.

$L_\Theta(h, y) = -w*y*\log(h)$ | where
w = [3 1 1 3]
h = one-hot predictions
y = one-hot labels

For post-processing, we apply a smoothening sliding-window of size 21 (2.1 seconds) where output at each point is set as the mode of the neighbouring 21 outputs. However, this smoothening is done only in cases where the output prediction is either class 0 or class 3, because we observed a lot of flickering-predictions between those classes.

III. EVALUATION

With our best hyper-parameters, we were able to achieve about 93.1% categorical accuracy on the training_set and about 92.8% categorical accuracy on the test_set. Training graphs for a typical ensemble unit are presented in Figure 7(a) and Figure 7(b). Validation split during model-training was set to be 0.2 We notice minimum over-fitting in the training graphs.

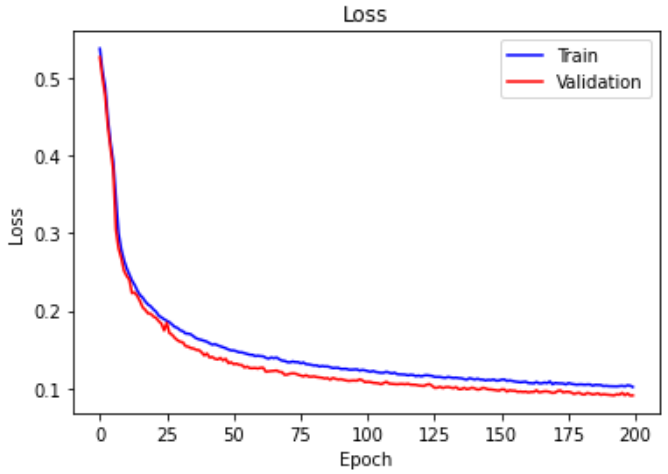

Figure 7(a): Train and Validation loss-plots

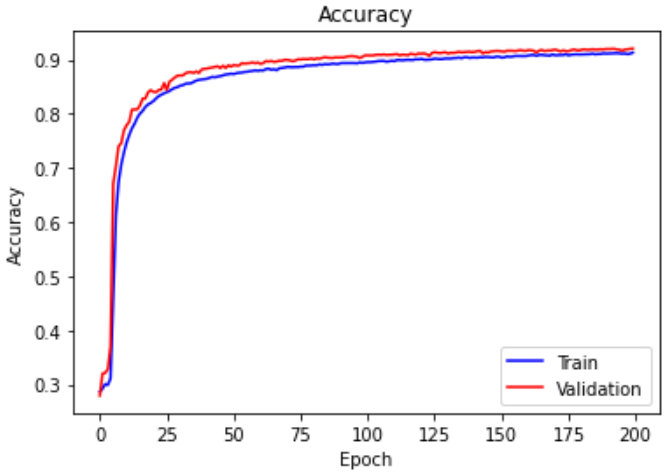

Figure 7(b): Train and Validation accuracy-plots

We work out the Confusion Matrix for the predictions on test-set (unseen data). This is presented in Table 2.In table 3, we present the class-wise Precision, Recall and F1 scores.

Table 2: Confusion Matrix on unseen data

		Prediction-Class				
		0	1	2	3	Total
Ground Truth-Label	0	713829	16032	10535	69584	809980
	1	28403	539163	123	643	568332
	2	28789	566	555799	668	585822
	3	51392	2536	125	806133	860186
	Total	822413	558297	566582	877028	2824320

Table 3: Classification report on unseen data

Class	Precision	Recall	F1-Score	Support (Number samples)
0	0.87	0.88	0.87	809980
1	0.97	0.95	0.96	568332
2	0.98	0.95	0.96	585822
3	0.92	0.94	0.93	860186
Weighted Average	0.93	0.93	0.93	2824320

From the confusion matrix, we notice comparable performance on each of the classes. We notice maximum miss-classifications between classes 0 (walking on solid-ground) and 3 (walking on grass) which is expected since both the activities are very similar. We find relatively few miss-classifications between classes 1 (going up), 2 (going down) and classes 0,3. This is also expected behaviour since both the activities are opposite to each other. Over-all, the confusion matrix on test-data paints a good-picture of the model.

From the report, we note that even after over-sampling classes 0 and 3 and having a custom weighted loss-function, F1 scores for class 0 and class 3 are consistently less than those of class 1 and class 2.

Some sample output-predictions made by our model are presented in the appendix section. On the x-axis, we show the time. On y-axis, we plot the six input-attributes of the test-input normalized using mean and standard-deviation of the training-set. As per expectations, class-0 and class-3 signals look very similar.

REFERENCES

[1] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[2] J. Yu and X. Zhou, "One-Dimensional Residual Convolutional Autoencoder Based Feature Learning for Gearbox Fault Diagnosis," in IEEE Transactions on Industrial Informatics, vol. 16, no. 10, pp. 6347-6358, Oct. 2020, doi: 10.1109/TII.2020.2966326.

[3] G. Dong, G. Liao, H. Liu and G. Kuang, "A Review of the Autoencoder and Its Variants: A Comparative Perspective from Target Recognition in Synthetic-Aperture Radar Images," in IEEE Geoscience and Remote Sensing Magazine, vol. 6, no. 3, pp. 44-68, Sept. 2018, doi: 10.1109/MGRS.2018.2853555.

[4] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, Malaysia, 2015, pp. 730-734, doi: 10.1109/ACPR.2015.7486599.

[5] Breiman, L. (1996). Bagging predictors. Machine Learning, 24.2

[6] Saniya Parveez, "Machine Learning Standardization (Z-Score Normalization) with Mathematics", July 2020 on towardsai.net. Available at: https://towardsai.net/p/machine-learning/machine-learning-standardization-z-score-normalization-with-mathematics

APPENDIX - SAMPLE RESULTS

We show some sample results on training and unseen data and then present a short discussion on them in this section.

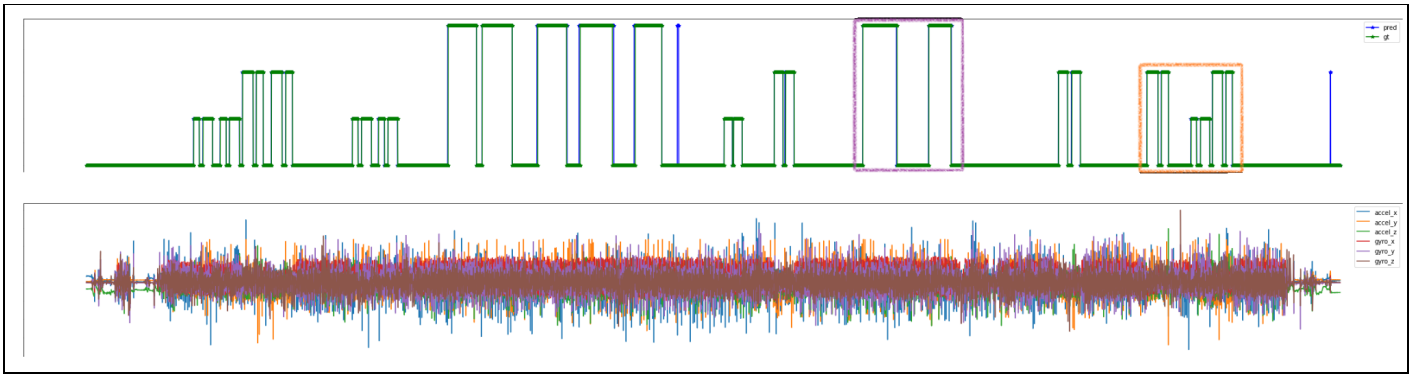

*Figure 8:Predictions, Ground Truth & attribute values for the entire **train-file** (subject 1, session 1) where GT and PRED are seen largely in agreement with each other; Notice the purple and orange sections; we zoom them in the next images*

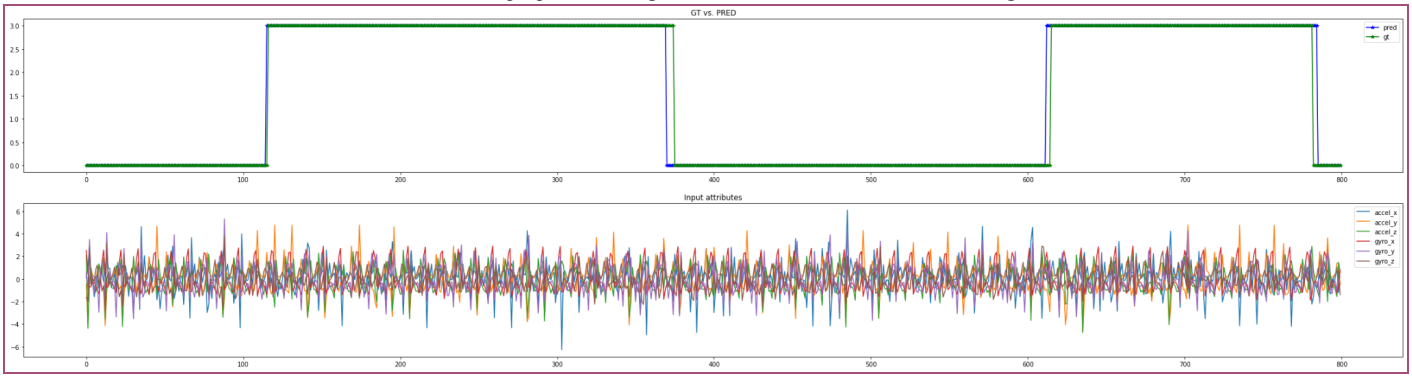

*Figure 9: A time-frame(575s to 655s from **train-file**: subject 1; session 1) where class-3 & class 0 are predicted*

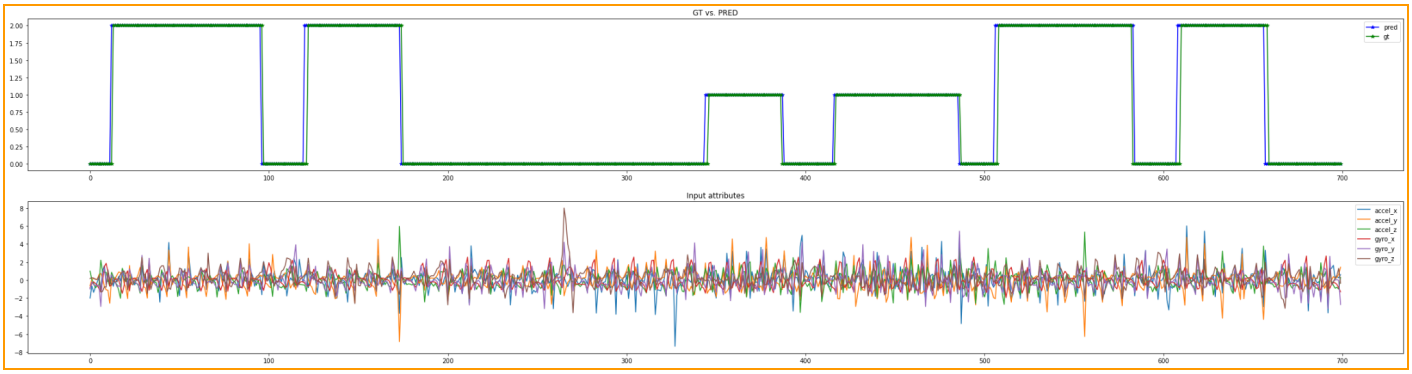

*Figure 10: A time-frame(800s to 870s from **train-file**: subject 1; session 1) where class-1 & class 2 are predicted*

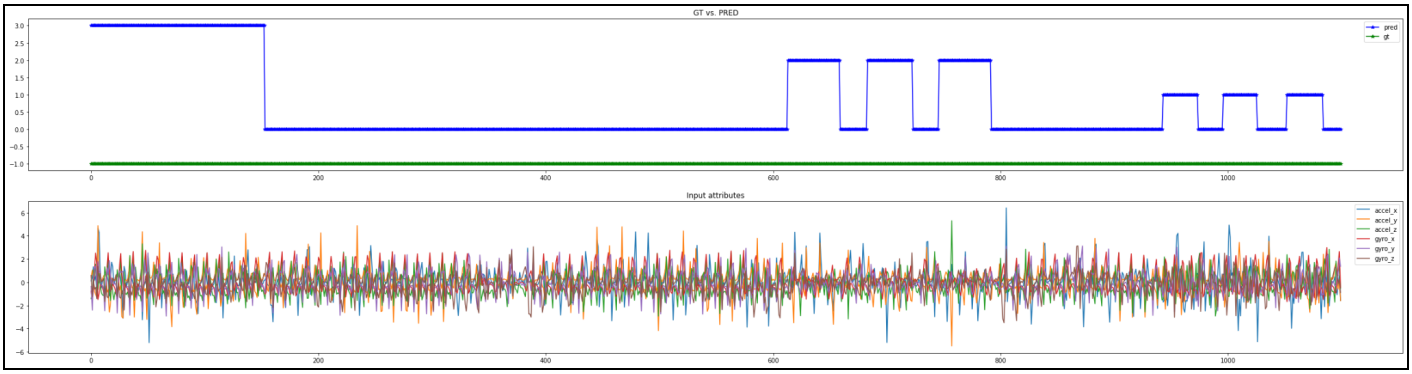

*Figure 11: An interesting time-frame(160s to 270s from **test-file**: subject 12; session 1) where all the classes are predicted; GT=-1 indicates GT labels not present*

Figures 8 through 11 show some interesting time-frames in the dataset. Figure 8 shows the attributes, labels as well as our final predictions for the train-file (Subject 1; Session 1). We can observe that the predictions largely agree with the GT labels other than a couple of noise-spikes (misclassifications between class 0 and class 3). In Figure 9, we show a slice of the train-file where classes 0 and 3 are present. In Figure 10, we show a slice where classes 1 and 2 are present. Finally, in Figure 11, we present an interesting file from the testing data. We notice how we are able to generate smooth and distinct predictions-blocks.