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

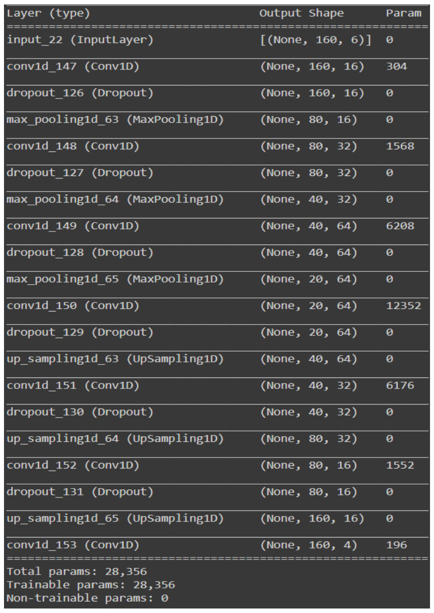
Akhil Bommadevara
abommad@ncsu.edu

Ishan Bhatt
ivbhatt@ncsu.edu

Aishwarya Seth
aseth@ncsu.edu

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. The MaxPooling operations are replaced by UpSampling operations (which typically expand 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*].
Figure 1 shows model.summary() of our best-performing model is



```
Layer (type)                          Output Shape        Param
================================================================
input_22 (InputLayer)                 [(None, 160, 6)]    0
conv1d_147 (Conv1D)                   (None, 160, 16)     304
dropout_126 (Dropout)                 (None, 160, 16)     0
max_pooling1d_63 (MaxPooling1D)       (None, 80, 16)      0
conv1d_148 (Conv1D)                   (None, 80, 32)      1568
dropout_127 (Dropout)                 (None, 80, 32)      0
max_pooling1d_64 (MaxPooling1D)       (None, 40, 32)      0
conv1d_149 (Conv1D)                   (None, 40, 64)      6208
dropout_128 (Dropout)                 (None, 40, 64)      0
max_pooling1d_65 (MaxPooling1D)       (None, 20, 64)      0
conv1d_150 (Conv1D)                   (None, 20, 64)      12352
dropout_129 (Dropout)                 (None, 20, 64)      0
up_sampling1d_63 (UpSampling1D)       (None, 40, 64)      0
conv1d_151 (Conv1D)                   (None, 40, 32)      6176
dropout_130 (Dropout)                 (None, 40, 32)      0
up_sampling1d_64 (UpSampling1D)       (None, 80, 32)      0
conv1d_152 (Conv1D)                   (None, 80, 16)      1552
dropout_131 (Dropout)                 (None, 80, 16)      0
up_sampling1d_65 (UpSampling1D)       (None, 160, 16)     0
conv1d_153 (Conv1D)                   (None, 160, 4)      196
================================================================
Total params: 28,356
Trainable params: 28,356
Non-trainable params: 0
```

Figure 1: Best model summary

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 .**

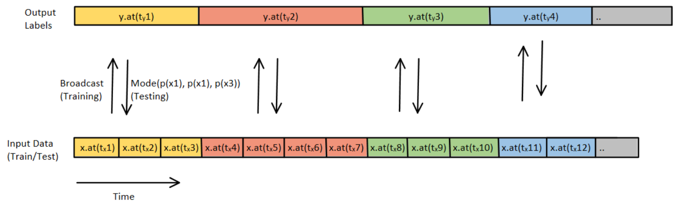


Figure 2: Synchronizing data with labels
Time scales of x(tx) and y(ty) are different. Each cell represents 1 unit of time on their respective scales. The function p(x) = prediction(x) and xi = x.at(txi)

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.

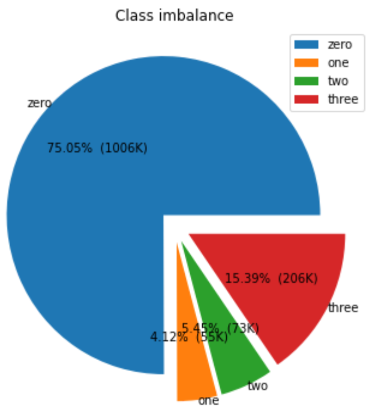


Figure 3: Class-imbalance in the original dataset

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.

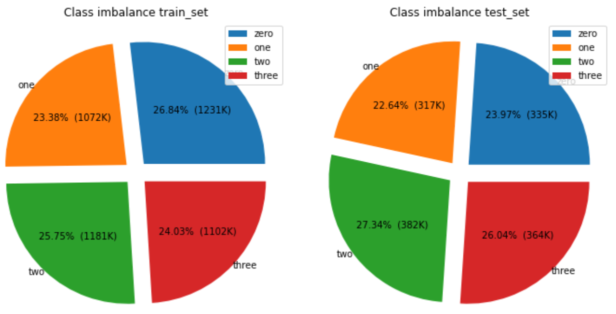


Figure 4: Class representations in train_set(left) and test_set(right)

After over-sampling, **we add a random noise of 15% of individual-magnitude to each time-step for each attribute** to make sure that same/similar samples are not being fed to the model. This should also help in preventing overfitting. 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 5 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 [5]

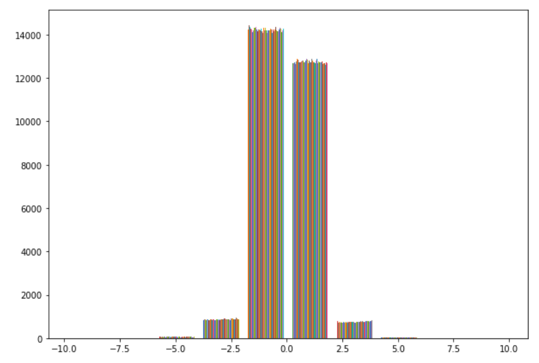


Figure 5: Histogram of a typical attribute (train_set) after Z-score normalization

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=16 gives the best result. 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.

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

depth	dropout	start_ filters	other	train_l oss	test_l oss
3	0.3	8		0.35	0.41
3	0.4	8		0.35	0.42
4	0.3	8		0.28	0.33
4	0.4	8		0.31	0.35
3	0.3	16		0.26	0.32
3	**0.4**	**16**		**0.24**	**0.31**
4	0.3	16		0.18	0.26
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

III. EVALUATION

With our best hyper-parameters, we were able to achieve about 90% categorical accuracy on the training_set and about 87% categorical accuracy on the test_set. Training graphs for the best model are presented in Figure 6(a) and Figure 6(b). Validation split during model-training was set to be 0.2.5

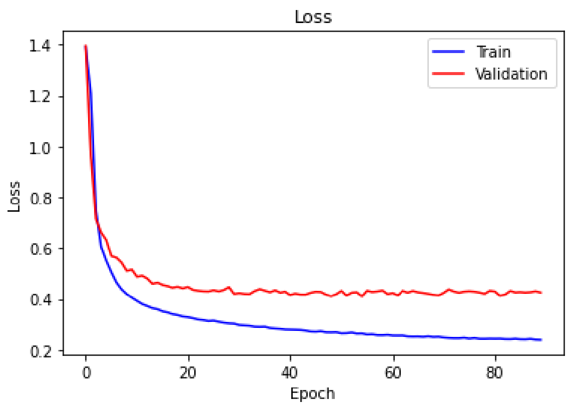


Figure 6(a): Train and Validation loss-plots

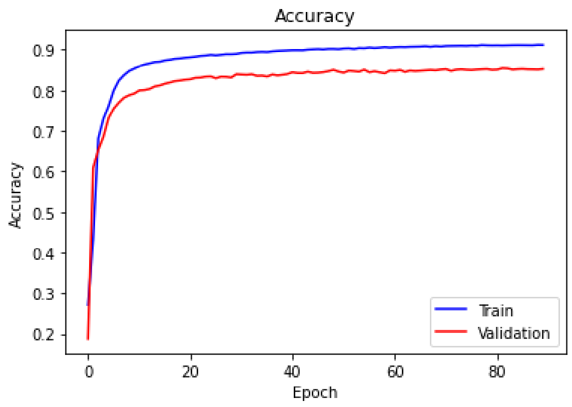


Figure 6(b): Train and Validation accuracy-plots

We notice a level of over-fitting in the training graphs but that may be due to variance between the train and the test sets. We did not have enough time to investigate this in detail but since the performance on unseen data was within 5% of performance on seen data, we think this might not be a critical issue to model stability.
We work the Confusion Matrix for the predictions on test-set (unseen data). This is presented in Table 2.

Table 2: Confusion Matrix on unseen data

		Prediction-Class				
		0	1	2	3	Sum
Ground Truth-Label	0	282372	12418	8243	32555	335588
	1	15725	300796	0	502	317023
	2	2667	1956	353651	545	358819
	3	70146	3814	488	290122	364570
	Sum	370910	318984	362382	323724	1376000

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.

We achieve a F1 score (macro-average) of about 0.88 on our test data. However, on the private test-set, our system could achieve only about 0.73 F1 score. This points to some unidentified issue in the pipeline.

Now, we show some sample output-predictions made by our model. 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.

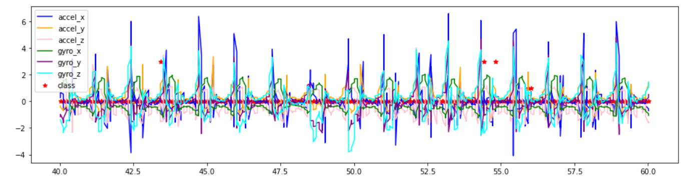


Figure 7: A time-frame(40s to 60s from test-file: subject 9; session 1) where class-0 is predicted

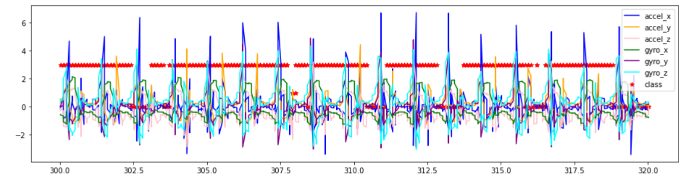


Figure 8: A time-frame(300s to 320s from test-file: subject 9; session 1) where class-3 is predicted

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] 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