Team 21 Proj-C: Terrain Identification from Time Series Data

Mihir Rajesh Khara
Department of Computer Engineering
North Carolina State University
Raleigh, NC
mkhara@ncsu.edu

Kush Paresh Jain
Department of Computer Engineering
North Carolina State University
Raleigh, NC
kpjain@ncsu.edu

Amarnath Vikramsingh Shinde
Department of Electrical Engineering
North Carolina State University
Raleigh, NC
avshinde@ncsu.edu

I. Methodology

The time series data that was given to us comprises a training dataset and a testing dataset which is received from an IMU sensor. In this project, we aim to classify to find different terrains from time series data. For predicting time series data, Convolutional Neural offers dilated convolutions in which filters can be used to compute dilations between cells. The size of the space between each cell allows the neural network to understand better the relationships between the different observations in the time series.[1]

The input file 'x' consists of sensor readings xyz accelerometers and xyz gyroscope measurements from the lower limb. 'y' file contains the labels. 0 indicates standing or walking on solid ground, 1 indicates going down the stairs, 2 indicates going up the stairs and 3 indicates walking on grass. 'xtime' files contain time stamps for the accelerometer and gyroscope measurements with a sampling rate of 40Hz and 'ytime' files contain time stamps for the labels with a sampling rate of 10Hz.

For data visualization, exploration, and preprocessing we have used libraries like Numpy, pandas, and matplotlib. An example of the data which we get from our first subject is shown in below figure 1.

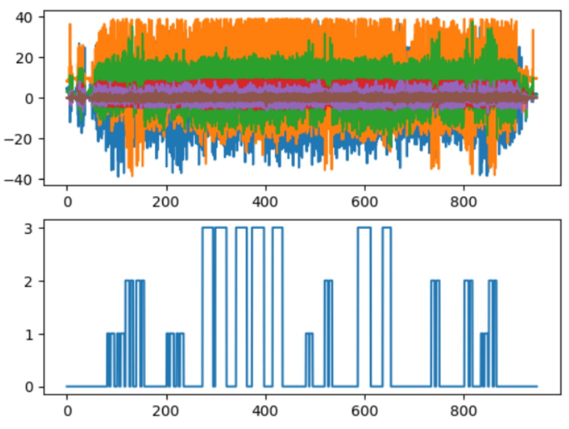


Fig.1. Input data x from subject_001_01

Based on the training data given, we are able to extract attributes as xyz readings from the accelerometer and gyroscope, their respective time stamps, y labels, and their respective time stamps. Hence we created data frames from csv files for readability. Then we combined both x and y datframes from all eight subjects along with time stamps by using the concat() function of pandas. We skipped the missing values after combining them to have the data in the most readable format.To delete the missing values we import all x and y data in lists and sliced the lists using iloc() function in pandas. Then we resized the indexs for x and y data.

We explored and found out that the given data is in an imbalanced state. To balance it we performed an undersampling process on our data. Undersampling is a technique to balance uneven datasets by keeping all of the data in the minority class and decreasing the size of the majority class. It is one of several techniques data scientists can use to extract more accurate information from originally imbalanced datasets.

In this project, we have used a Convolutional neural network to train and test our data after random undersampling as shown in below figure 2.

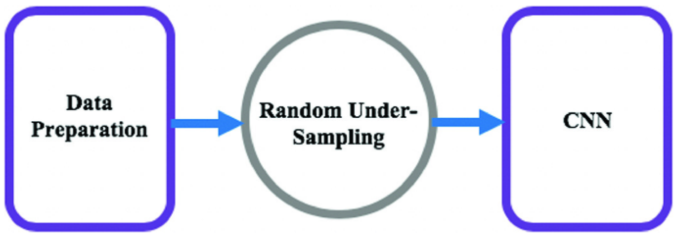


Fig.2. Overview of project

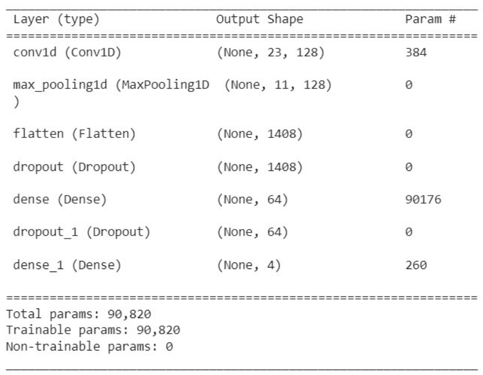
We used convolutional, max pooling, flatten, dropout, and dense layers in our CNN.[2]

II. Model Training and Selection

A. Model Training

We had a total of 56 input files in our training data. We split that data into sets of 27 and 27 for training and validation respectively. There were a total of 58452 line rows of data frames after we performed random undersampling. We used 75% for the training data frame which is 43839 line rows of data frames as the training set. For the validation set, we used 7306 data frame rows.

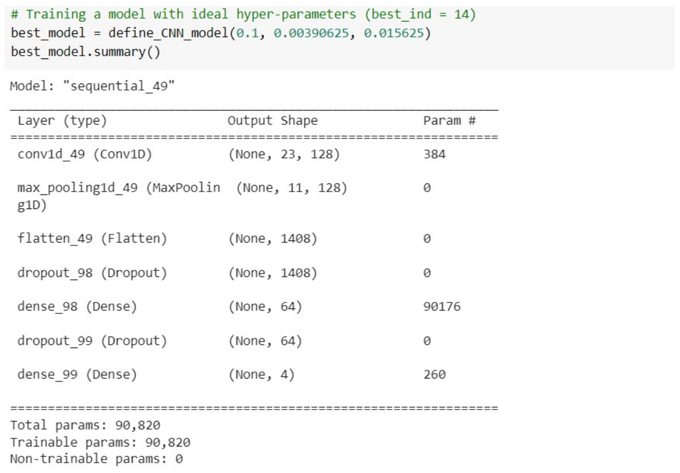
For model training, we defined functions to plot loss and accuracy plots. We resized the shape of the training and validation data of x and y. Then we converted the data to the binary class matrix. CNN which we used has activation function relu, loss function as categorical_crossentropy and optimizer as adam. The baseline of our defined CNN is shown below in figure 3.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 23, 128)           384

 max_pooling1d (MaxPooling1D  (None, 11, 128)          0
 )

 flatten (Flatten)           (None, 1408)              0

 dropout (Dropout)           (None, 1408)              0

 dense (Dense)               (None, 64)                90176

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 90,820
Trainable params: 90,820
Non-trainable params: 0
_________________________________________________________________
```

Fig.3. Baseline of CNN Model

After training the model 48 times we got the best-fit model. We again trained the best-fit model with ideal hyperparameters. We got ideal hyperparameters as drop rate = 0.1, relu kernel regularizers as 0.00390625, and 0.015625. We used 50 epochs to train the best-fit model. After that, we got the below CNN layer output as shown in figure 4.



```
# Training a model with ideal hyper-parameters (best_ind = 14)
best_model = define_CNN_model(0.1, 0.00390625, 0.015625)
best_model.summary()
```

```
Model: "sequential_49"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_49 (Conv1D)          (None, 23, 128)           384

 max_pooling1d_49 (MaxPoolin  (None, 11, 128)          0
 g1D)

 flatten_49 (Flatten)        (None, 1408)              0

 dropout_98 (Dropout)        (None, 1408)              0

 dense_98 (Dense)            (None, 64)                90176

 dropout_99 (Dropout)        (None, 64)                0

 dense_99 (Dense)            (None, 4)                 260

=================================================================
Total params: 90,820
Trainable params: 90,820
Non-trainable params: 0
```

Fig.4. Best-fit trained Model using CNN

B. Model Selection

We used CNN because it offers dilated convolutions in which filters can be used to compute dilations between cells. The size of the space between each cell allows the neural network to understand better the relationships between the different observations in the time series.[4]

In CNN we used ReLU function. ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

A pooling layer is added after convolution to reduce dimensionality, which can both reduce computational time to train by reducing parameters and can also reduce the chances of overfitting. The most common type of pooling is max pooling which returns the max value in an NxN matrix pooling filter.

We found the ideal dropout rate as 0.1 and added two dropout layers. Also, we added two dense layers to make it more easier to classify the terrains[3].

III. Evaluation

After training the best-fit model for 50 epochs we got the respective accuracy and loss data. We found that the maximum accuracy is between 25 to 50 epochs. So we plotted the plots for minimum and maximum accuracy as shown in below figure 5 and figure 6.

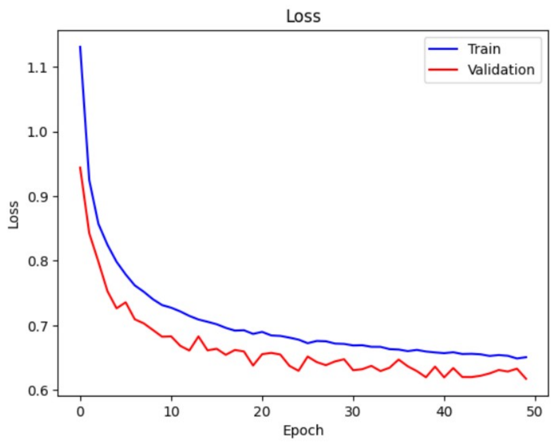


Fig.5. Loss in best-fit model

We found out the predictions for y files. The data we used is validation data of x which we found out earlier, 100 dataframes of that validation set and verbrose we kept to 0 to get the best predictions.

The prediction report is shown in figure 7.

We can see clearly that the best precision is 0.93, the best recall is 0.92, the best F1 score is 0.92 and the accuracy is

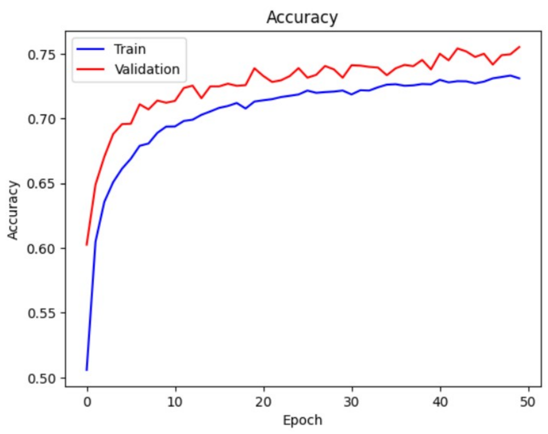
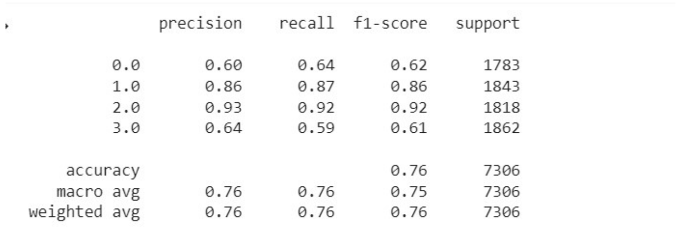


Fig.6. Accuracy in best-fit model



```
              precision    recall  f1-score   support

         0.0       0.60      0.64      0.62      1783
         1.0       0.86      0.87      0.86      1843
         2.0       0.93      0.92      0.92      1818
         3.0       0.64      0.59      0.61      1862

    accuracy                           0.76      7306
   macro avg       0.76      0.76      0.75      7306
weighted avg       0.76      0.76      0.76      7306
```

Fig.7. Prediction report

76%.

In the end, we tested the data for the given test files from subjects 9 to 12 and got the respective predictions that we submitted. The test data was used from subject_009_01 of x to subject_012_01 of x. Similarly, we kept the batch size =100.

References

[1] S. Albawi, T. A. Mohammed and S. Al-Zawi, ”Understanding of a convolutional neural network,” 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186

[2] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8

[3] B. Zhao, H. Lu, S. Chen, J. Liu and D. Wu, ”Convolutional neural networks for time series classification,” in Journal of Systems Engineering and Electronics, vol. 28, no. 1, pp. 162-169, Feb. 2017, doi: 10.21629/JSEE.2017.01.18

[4] P. Khanna and A. Narayan, ”Light Weight Dilated CNN for Time Series Classification and Prediction,” 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 2020, pp. 2179-2183, doi: 10.1109/SMC42975.2020.9283052