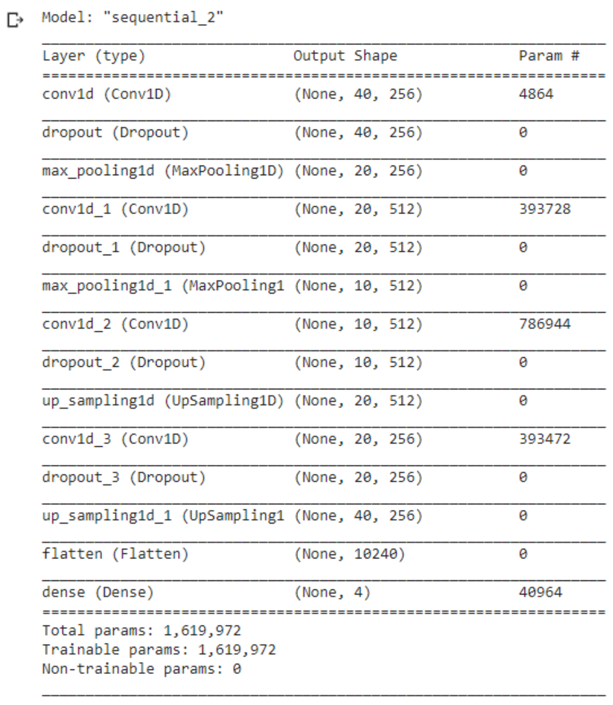
[Team 28] Proj-C: Human Activity Recognition Using 1D CNN

Unnati Nadupalli (upnadupa), Apurva Sonavane (asonava), Nicholas Viado (ndviado)

I. Methodology

In our dataset, we have x-axis, y-axis, and z-axis readings of accelerometer and gyroscope sensors. The readings have the frequency of 40 Hz and the labels for the activity to be predicted have the frequency of 10 Hz. We first interpolate the label values using the pandas series interpolate() function to be at the same frequency as the sensor recordings. We used the sklearn LabelEncoder class to convert our target labels to values such as to 0,1,2, and 3 so that they do not convert to float values at any point. After that, we used the StandardScaler library to normalize the acceleration and gyroscope measurements in the dataset so that all the values are within the new range of 0 and 1 [3]. We divided the data into non-overlapping frames of 40 rows each. We do this because the frequency of x(sensor recordings) is 40 Hz which corresponds to one second. In each frame, we take the statistical mode of the y value for that window using the bincount() method. We handle the class imbalance by using the compute_class_weights() function from the sklearn class_weight library that assigns a higher penalty for mislabelling the minority class. For our model, we have used a 1D convolutional autoencoder, this is used for denoising. This particular model for time series ensures that the pattern that gets repeated for a movement is distinctly visible and the noise otherwise is smoothened out. The most repeated pattern for all the movements in a second will be clearly visible because of the decoder reconstruction that denoises the data. We use 4 CNN layers with the first layer having its input dimensions as (40,6). 40 for the size of the window and 6 for the number of features produced by the accelerometer and gyrometer data. The four CNN layers have 256, 512, 512 and 256 neurons respectively with the parameter named padding set to be true. This parameter ensures that the CNN layers do not reduce the dimensions of the frame. Each of these layers has a kernel size of 3 and the activation function used is ‘relu’[5]. We reduce the dimensions by applying Max Pooling with a pool size of 2 on the first two layers for feature detection. We add a drop out of 0.5 for regularization after every CNN layer. This ensures that the output has the same dimensions as the original frame after reconstruction. We upsample using the decoder having the size parameter as 2. We use flatten to reshape the vector into a single dimension and then pass this input to a dense layer with 4 units. A dense layer is a fully connected layer that fetches the actual class. We use the softmax activation function because we want to use this model for classification. We give 4 units in this layer as we have 4 different classes that we can classify to. These classes correspond to “standing/walking”, “going downstairs”, “going upstairs”, and “walking on the grass”. Then we split the data into training and testing data using the sklearn train_test_split in a 8:2 ratio and train our model.

Model Summary Diagram:



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 40, 256)           4864
_________________________________________________________________
dropout (Dropout)            (None, 40, 256)           0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 20, 256)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 20, 512)           393728
_________________________________________________________________
dropout_1 (Dropout)          (None, 20, 512)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 10, 512)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 10, 512)           786944
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 512)           0
_________________________________________________________________
up_sampling1d (UpSampling1D) (None, 20, 512)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 20, 256)           393472
_________________________________________________________________
dropout_3 (Dropout)          (None, 20, 256)           0
_________________________________________________________________
up_sampling1d_1 (UpSampling1 (None, 40, 256)           0
_________________________________________________________________
flatten (Flatten)            (None, 10240)             0
_________________________________________________________________
dense (Dense)                (None, 4)                 40964
=================================================================
Total params: 1,619,972
Trainable params: 1,619,972
Non-trainable params: 0
_________________________________________________________________
```

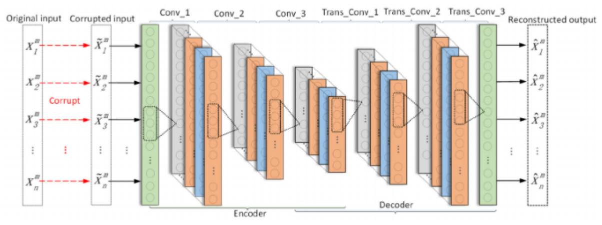


Fig. Structure of a denoising Autoencoder. **[8]**

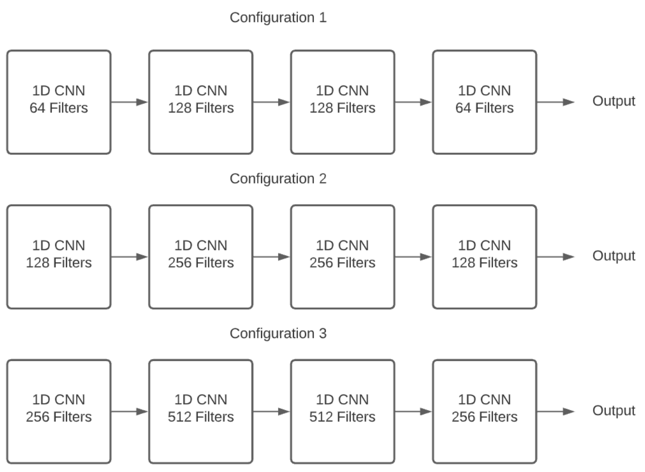
II. Model Training and Selection

A. *Model Training*

Before training the model, our data was split into 80% Training and 20% Validation. This resulted in around 25,000 inputs for training and 5000 for validation. Since our model was based on a memoryless CNN and the inputs were organized into 1 second samples, we were able to shuffle the inputs after every epoch to help avoid overfitting.

B. *Model Selection*

There were several hyper parameters to be tuned for our model. Our model has 4 different 1D CNNs that feed into each other. The first hyper parameter to tune was the number of filters in each CNN layer. We tested 3 different configurations.

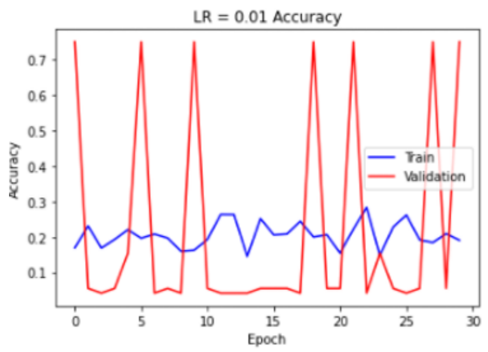


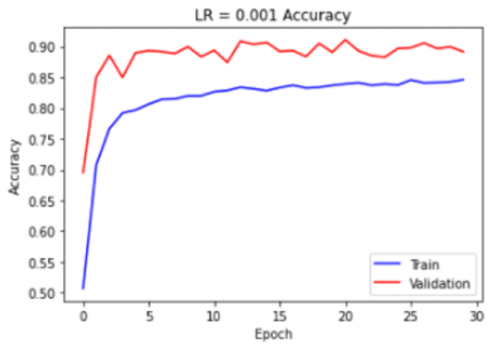
Sklearn metrics after 45 epochs (‘Weighted’ Parameter)

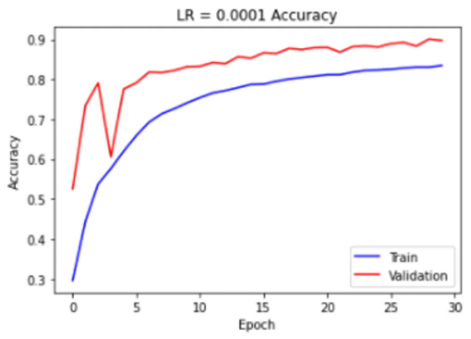
	F1 Score	Precision	Recall
Configuration 1	0.479	0.794	0.422
Configuration 2	0.645	0.584	0.859
Configuration 3	0.870	0.858	0.786

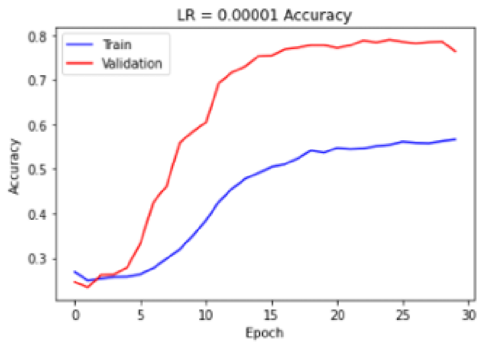
The 3rd configuration had the best results and was chosen for our final model. Due to its large number of filters, it was able to capture more features before it started to overfit [6]. Because of the high number of neurons in each of these layers, it took significantly longer to train. For other hyper-parameters the 1st configuration was used so that we could efficiently train each model.

The next hyper parameter to tune was the learning rate for the Adam optimizer. This optimizer was chosen due to its low learning curve since we only had to figure out the best learning rate value. The learning rate varied from 0.01 to 0.00001 and gave the following results on our training and validation accuracy after 30 epochs.









Looking at the results, setting the learning rate from 0.01 to 0.0001 resulted in overfitting. While the training and validation accuracies are slightly better with a 0.0001 learning rate, the curve shows more overfitting than the model using 0.00001 learning rate. The smaller learning rate requires more epochs to run in order to reach higher accuracies, but in the end will result in less overfitting.

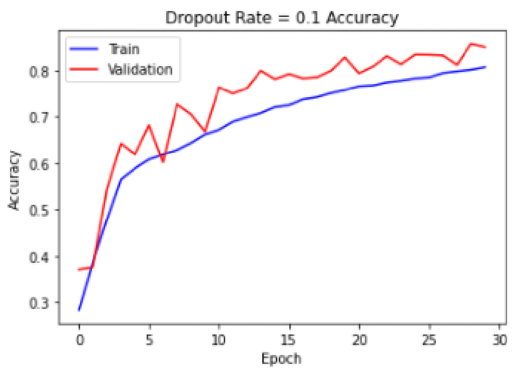
As our time series dataset for human activity recognition is imbalanced in nature. We have tried several different techniques for imposing a balance. In the previous iterations, we used SMOTE NC which is a Synthetic Minority Over-sampling Technique for Nominal and Continuous features along with class weights. This is used for creating data points of the minority class using KNN clustering. We were trying to fit the model on this data, however, our accuracy started jumping points the moment we quit using SMOTE NC. This is because SMOTE NC sometimes creates data points that could be from the majority class and misclassify it to be in the minority class.

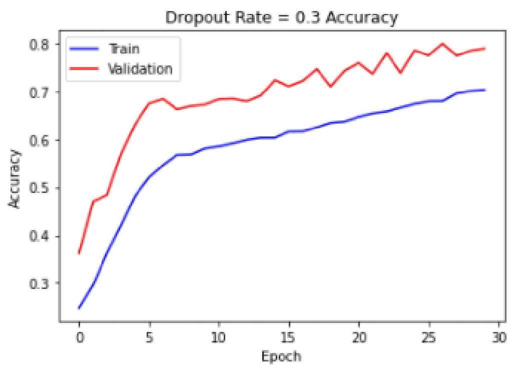
Hence we removed SMOTE NC and only added class weights to help mitigate the issue of data imbalance. The majority class makes up around 80% of the output data. Class weights were set using the compute_class_weight function from the sklearn library. These weights would penalize the minority class if it predicted wrong to help the model learn better [2].

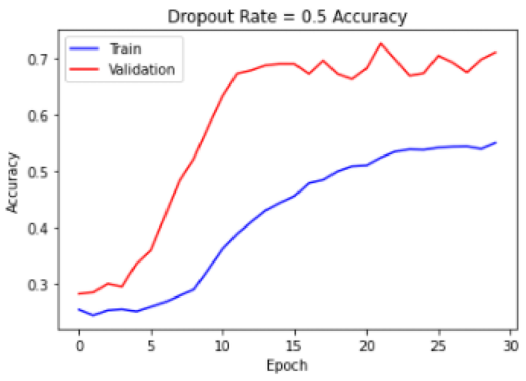
Label	Without Weights	With Weights	Actual Outputs
0	11750	8170	9892
1	230	90	143
2	40	1090	678
3	10	2680	1322

When class weights are used, the model is able to predict the minority classes better. When class weights are not used, the model struggles to predict labels 2 and 3 and heavily favors label 0 which is the majority class.

The last hyper parameter to train was dropout rate. Values of 0.1, 0.3, and 0.5 were used to help avoid overfitting.



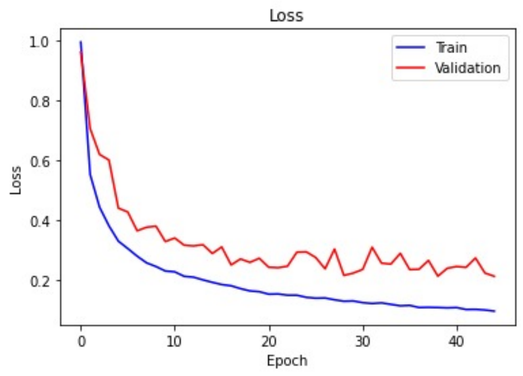


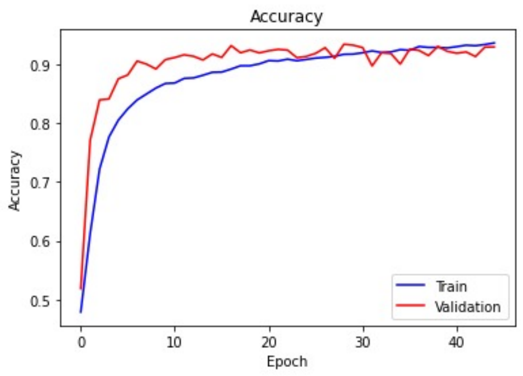


The dropout rate of 0.1 had the worst signs of overfitting compared to 0.3 and 0.5. When looking at the accuracies for 0.3 and 0.5 there are similar signs of overfitting. However, the 0.5 dropout rate shows a better learning curve at the beginning of training. When using 0.3 and 0.5 dropout rates on the final predictions, the 0.5 model performed better.

III. Evaluation

Our final model utilizes a 4-layer 1D CNN autoencoder with a learning rate of 0.00001, a dropout rate of 0.5, and class weights to help balance the dataset. It performed very well and achieved an F1 score of 0.87 on the hidden prediction datasets. We were unable to calculate precision and recall scores since we did not have access to the actual labels.





While training on 7 different subjects and using a completely different subject for test data, our model achieved the following metrics.

```
              precision    recall  f1-score   support

           0       0.46      0.97      0.63      4695
           1       0.01      0.05      0.02        40
           2       0.91      0.61      0.73      1010
           3       0.94      0.20      0.33      6290

    accuracy                           0.53     12035
   macro avg       0.58      0.46      0.43     12035
weighted avg       0.75      0.53      0.48     12035
```

Overall our model performs well despite the larger number of inputs and imbalance dataset. Our model could likely benefit from data augmentation to prevent more overfitting, however the current performance is adequate.

References

[1] J. Brownlee, *1D Convultional Neural Network Models for Human Activity Recognition,* Machine Learning Mastery, Aug. 28, 2020. Accessed on: March 20, 2021 [Online] Available: https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

[2] *How to improve Class Imbalance Using Class Weights in Machine Learning*, 2020. Accessed on: March 20,2021 [Online] Available:https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

[3]How to Use StandardScaler and MinMaxScaler Transforms in Python, June 2020. Accessed on: March 20, 2021. [Online] https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/

[4]Stratified K-Fold: What It Is & How to Use It, Jan 2020. Accessed on : March 25, 2021. [Online] Available: https://towardsdatascience.com/stratified-k-fold-what-it-is-how-to-use-it-cf3d107d3ea2

[5]A Practical Guide to ReLU, Nov 2017. Accessed: 20 March 2021. [Online] Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

[6] A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST, May 2018. Accessed [Online] : April 2021 https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d

[7]SMOTE for Imbalanced Classification with Python, Jan 2020. Accessed: March, 2021. Available [Online] : https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[8]Fault Diagnosis of Rotating Machinery under Noisy Environment Conditions Based on a 1-D Convolutional Autoencoder and 1-D Convolutional Neural Network. Feb, 2019. Accessed on : April, 2021. Available [Online] : https://www.researchgate.net/publication/331335652_Fault_Diagnosis_of_Rotating_Machinery_under_Noisy_Environment_Conditions_Based_on_a_1-D_Convolutional_Autoencoder_and_1-D_Convolutional_Neural_Network