Ethan Piette and Zach Sevrens
CSDS 340: Case Study 2

Introduction:
In this case study, we used deep learning to build a Human Activity Recognition (HAR) model. This model was trained on unprocessed data from the accelerometer and gyroscope inside of smart phones. The data is in the form of 6 files from the X,Y, and Z direction of both components, split into rows and columns representing minutes and the 60 seconds in that minute respectively.

Pre Processing:
In order to decide which models to experiment with we performed extensive exploratory data analysis (EDA). The majority of the visualizations can be found in the appendix section at the end of the paper. Initially, we determined the shape, mean, and standard deviation of each of the components of the dataset (See Appendix 1/2). From this we found the accelerometer had peaks around $\pm$ the speed of gravity and 0, with frequency of each peak varying based on the orientation of the phone. In contrast, the accelerometer data from the X direction had a roughly normal distribution (See Appendix 3). Next, we checked the correlation matrix of all of the features (See Appendix 4) and discovered there was little to no correlation between the features. We did a plurality of tests to determine the presence of outliers, their distribution among the different features, and sampling different feature creations. In summary, there was a large amount of outliers scattered throughout the different features. None of the basic feature engineering attempts we made created any separation. We then experimented with entropy, low and mid frequency power, and variance. From this we found that the frequency features provided useful information for our model to evaluate based on the strong negative correlation between them, and through testing the weak correlation from entropy was helpful as well. Variance significantly hindered our algorithm's performance when it was included in the final model (See Appendix 5 and 6 for correlation and boxplots). Based on this EDA and testing, we settled on using Frequency power, entropy, and handling the class imbalance. We utilized SMOTE (Synthetic Minority Oversampling Technique) to handle the large class imbalance (See Appendix 7 for its effect).

Model:
We used a multi-task deep learning model that combines raw sequential data and extracted features for improved performance. The architecture integrates convolutional, recurrent, dense, and auxiliary outputs for multi-task classification, focusing on robustness and interpretability. The raw sequential data is processed through the convolutional and LSTM layers, while the extracted features are passed through dense layers, the outputs from these are fused for primary predictions, and supported by auxiliary outputs for specific problematic classes. A full description of the layers and their parameters can be seen below.

Convolutional Layers:
We had two convolutional layers to help capture local temporal patterns in the raw data. The first layer used 64 filters, kernel size of 3, and ReLU activation. The second layer used 128 filters, kernel size of 5, and ReLU activation. These will detect dependencies in the sequence.

Pooling Layer:
We used a MaxPooling1D layer with a pool size of 2 to down sample feature maps, which helps to reduce computational cost and focusing on dominant features.

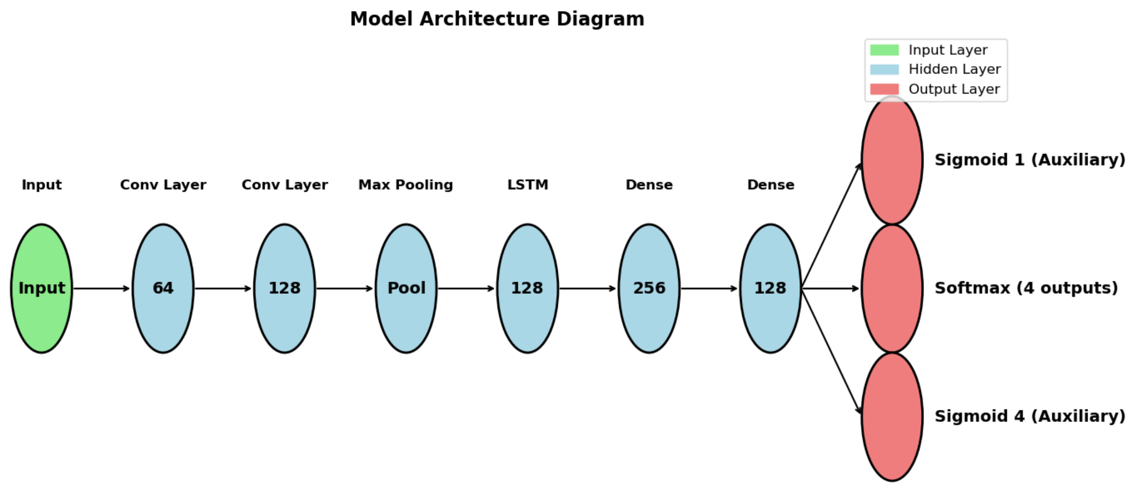
LSTM Layer:
We use a single LSTM layer with 128, and return sequence enabled. This captures long-term dependencies in the data, and prepares it for the attention layer.

Attention Layer:
The attention layer emphasizes the most relevant time steps, which helps to improve interpretability and performance by focusing on critical features.

Dense Layers:
We used two dense layers for our extracted features. Dense layers had 256 and 128 units, and both used ReLU activation. One was also used for the fusion, the layer had 128 units and ReLU activation. They all used a dropout rate of 0.3, with the dropouts coming between the first two, and after the third. These layers work to capture interactions between extracted features and combine inputs effectively, and the dropouts prevent co-dependencies among nodes and help prevent overfitting.

Auxiliary Outputs:
We used two binary auxiliary outputs for classes 1 and 4 due to how challenging it was to classify the two accurately. We utilized sigmoid functions and weighted losses to help these outputs learn the distinction between resting and driving given the periods of inactivity.

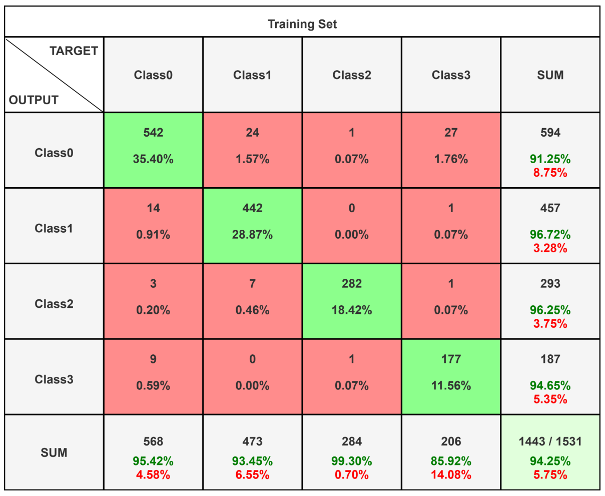
Model Compilation/Parameters:
We used an Adam optimizer, to ensure adaptive learning rates. For our main output we used sparse categorical cross entropy as our loss function, and for the auxiliary outputs we used binary cross entropy. We settled on using 50 epochs, with a batch size of 32, which balanced efficiency and stability of our model, and made sure we had enough epochs to converge without overfitting.



Post-Processing:
We used post-processing strategies of implementing a confidence threshold for a problematic class, and smoothing our predictions. The confidence threshold was implemented on class 4, as the model struggled to differentiate between class 1 and 4, by implementing a threshold of 0.92, we were able to much better capture the difference between the two classes leading to a better overall performance of the model. In addition, we then smoothed our predictions by applying a mode filter. A sliding window of size 5 was used to smooth the data, which helps reduce noise in the sequence of predictions, which leads to more consistent temporal outputs, which helps improve the overall performance of the model.

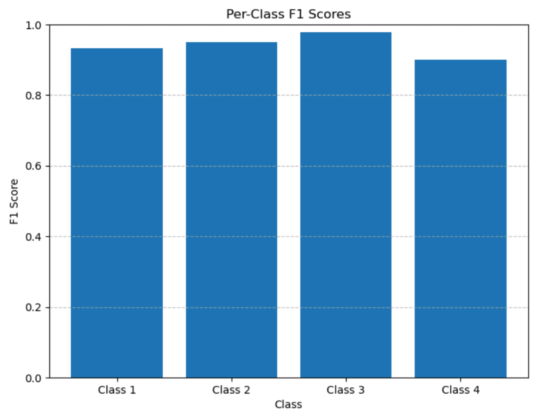
Other Models:
In addition to evaluating the deep learning model we developed, there were several other model architectures we experimented with. Originally, we believed that we could use feature extraction to aid in the creation of a linear regression model. However through our testing, the best macro and micro F1 scores we got were ~10% worse than the deep learning model. Beyond the performance difference, we thought that the sequential and multivariate nature of the data would require a model that could capture temporal dependencies more effectively. We also experimented with using models that were solely CNNs or LSTMs. These algorithms were very effective on their own, but combined (and with the addition of Dense Layers and Auxiliary Outputs) these models covered each other's shortcomings and added additional features. For example, the CNNs captured temporal patterns very well, whereas the LSTMs are designed to handle sequential data. The Dense layers combined this information and also contained drop outs to help prevent overfitting in the model.

Model Evaluation:
For model evaluation we focused on optimizing the micro and macro F1 scores. Since we are trying to accurately predict human activity based on sensor data, it is important to have a good balance between precision and recall, since we want to limit both false positives and capture as many true positives as possible for all of the classes. When evaluating our model we used a visualization of the F1 scores for each of the classes, along with a confusion matrix to evaluate the performance of our model, along with the micro and macro averaged. These evaluations for our final model can be seen below: (Correlation Matrix Left, Class F1 Scores Right)



Training Set

TARGET / OUTPUT	Class0	Class1	Class2	Class3	SUM
Class0	542 35.40%	24 1.57%	1 0.07%	27 1.76%	594 91.25% 8.75%
Class1	14 0.91%	442 28.87%	0 0.00%	1 0.07%	457 96.72% 3.28%
Class2	3 0.20%	7 0.46%	282 18.42%	1 0.07%	293 96.25% 3.75%
Class3	9 0.59%	0 0.00%	1 0.07%	177 11.56%	187 94.65% 5.35%
SUM	568 95.42% 4.58%	473 93.45% 6.55%	284 99.30% 0.70%	206 85.92% 14.08%	1443 / 1531 94.25% 5.75%

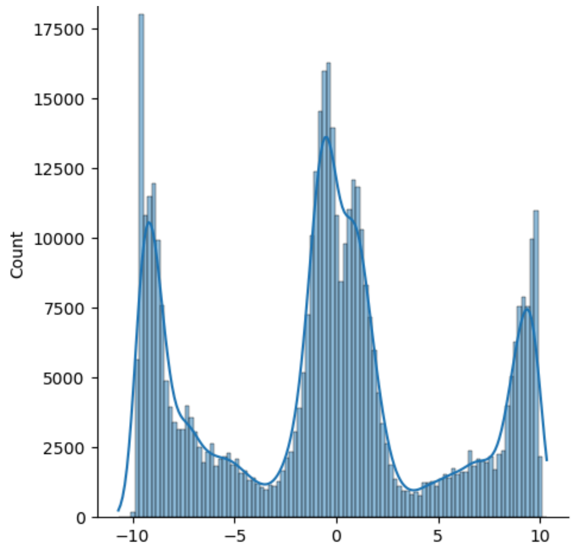


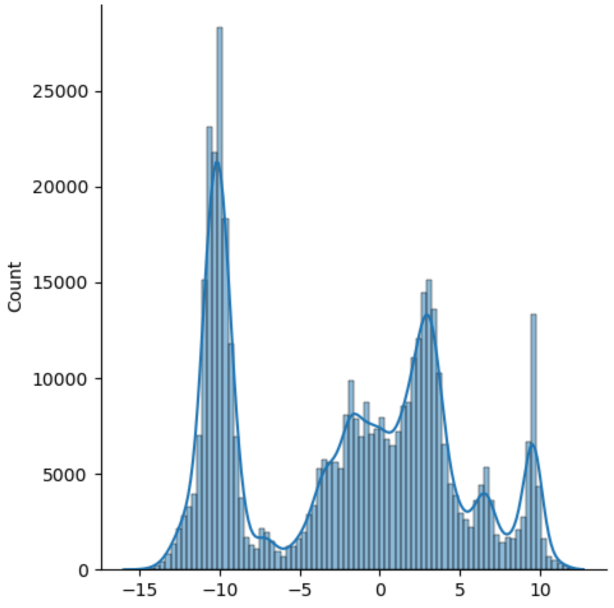
In addition to the ones we used for evaluating our model's performance there are some other ways that we could evaluate the performance of the model. For example, we could look at the temporal consistency of the model, in other words we could evaluate the smoothness of predictions over time, using a temporal smoothness score, and a transition matrix analysis. Another evaluation that could be done is to look specifically at our binary auxiliary outputs for classes 1 and 4, and evaluate their performance by using the AUC metric, and examining the ROC curve to see how they are performing.

Modifications for Real-Time Predictions:
In order for this model to work well for making real-time predictions, such as in a HAR app on a smartphone, we would need to modify parts of the model for use without access to any future data. It would require changes to pre-processing, as we would need to change the feature extraction to not use entire sequences, in particular our model uses FFT and spectral entropy which would need to be changed to use only rolling windows of past data. In addition the model itself would need to be tweaked slightly as the attention layer would need to be causal so that it only uses past and current data. In addition to these major changes, the overall receiving of data, and making predictions on it would need to be modified so that it can handle receiving new data constantly over time.

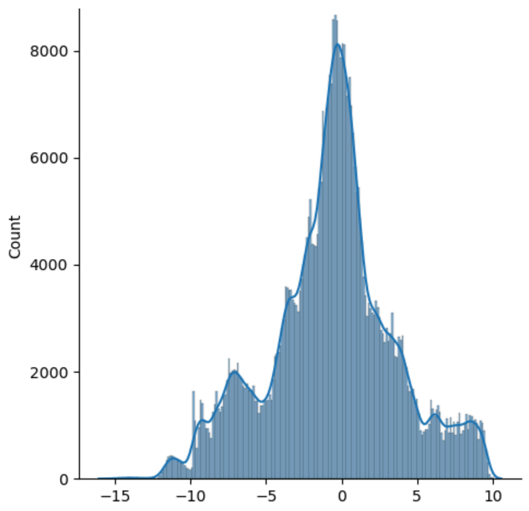
Appendix:

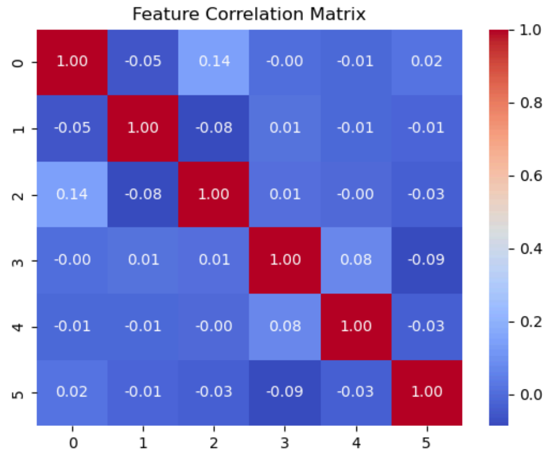
1/2: Accelerometer data from Y and Z axis respectively)



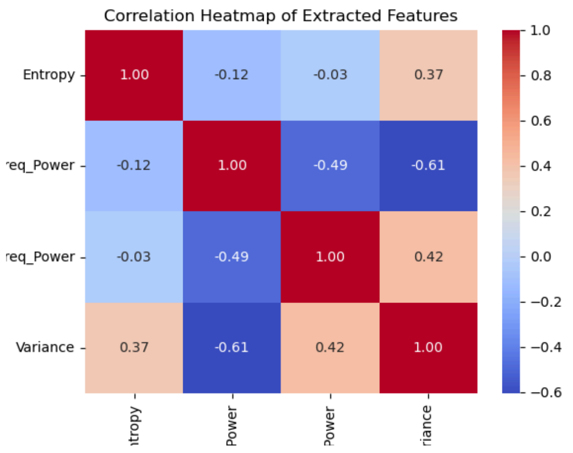


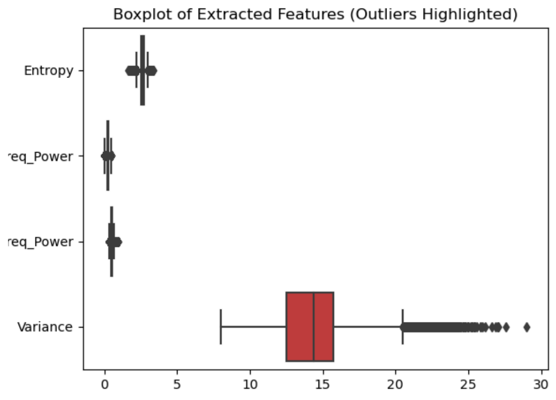
3/4: Accelerometer data from X and Correlation Matrix)



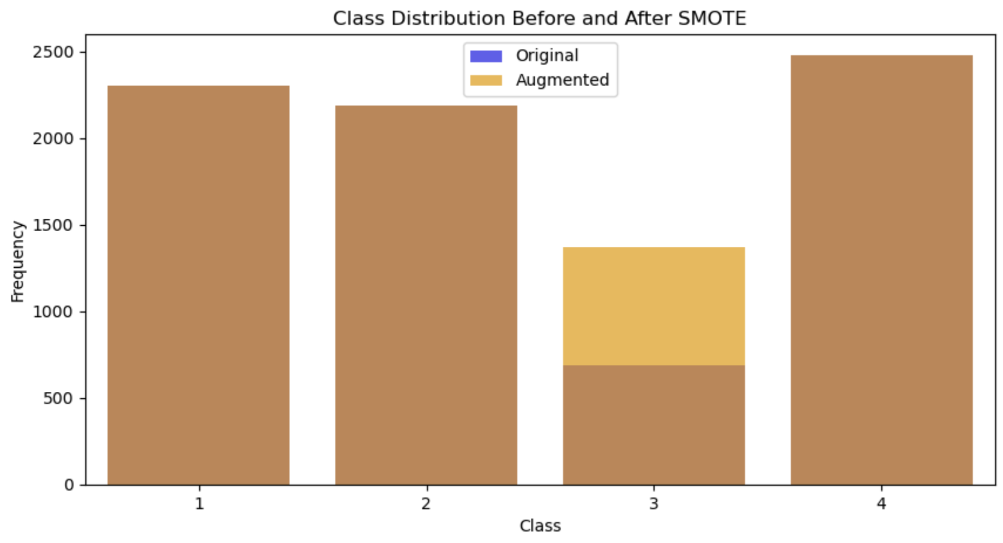


5/6: Correlation of Engineered Features and Box Plots)





7: Effect of SMOTE)


Class Distribution Before and After SMOTE
Original
Augmented
Frequency
Class