בס"ד

DL workshop - assignment 2 report

1 - EDA

- **Part a**

I:

The given data is Human Activity Recognition time-series data collected by sensors from various subjects. This competition aims to recognize activity according time series data. In addition, this project aims to develop a machine-learning model based on data collected from a wearable sensor. It represents positions and accelerations over a short time period (1-4 seconds), in 100 Hz granularity.

II:

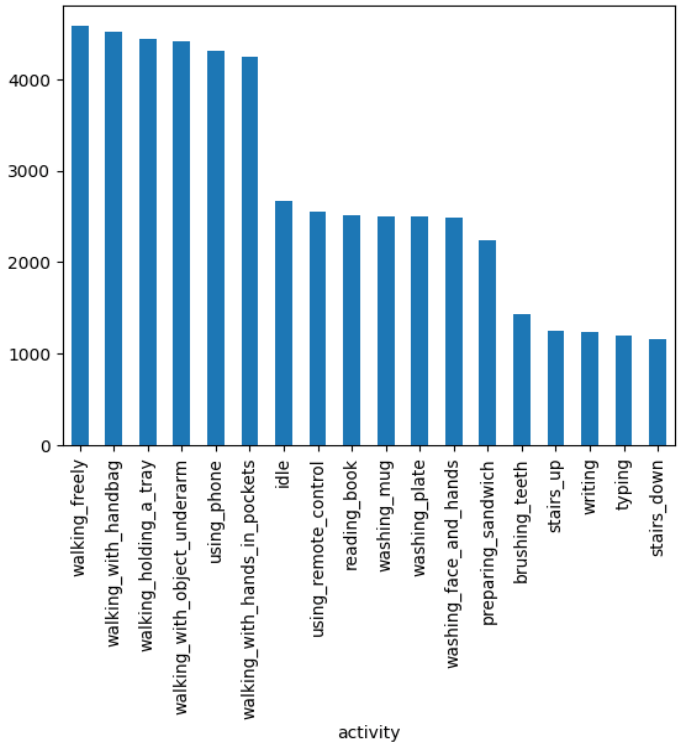
The data appears to be homogenous in most cases. The sensor readings show repetitive patterns and trends, and a steady seasonality. The positions is corresponding to same axes of x, y and z. Acceleration is [m/s/s] and we took him as a common measure in the dataset with the acceleration to all the samples in this kind of datasets.

III:

It was labeled by monitoring the movement of 8 subjects during specific activities that were asked of them. Each sample done within 1/100 sec and some samples show the same activity. Some of the activities takes more time to do and the result is that in the dataset we have more samples for one class than another.

IV:

Labels should be treated equally but the dataset is unbalanced. We assume that some activities took more time than others and the samples stopped when the movement ended.
We took each class and evaluated them like they were equal but we didn't do any preprocessing of data to make them with the same number of samples, we thought it would be better to keep the state of the records as they are and not to lose information later in our process.



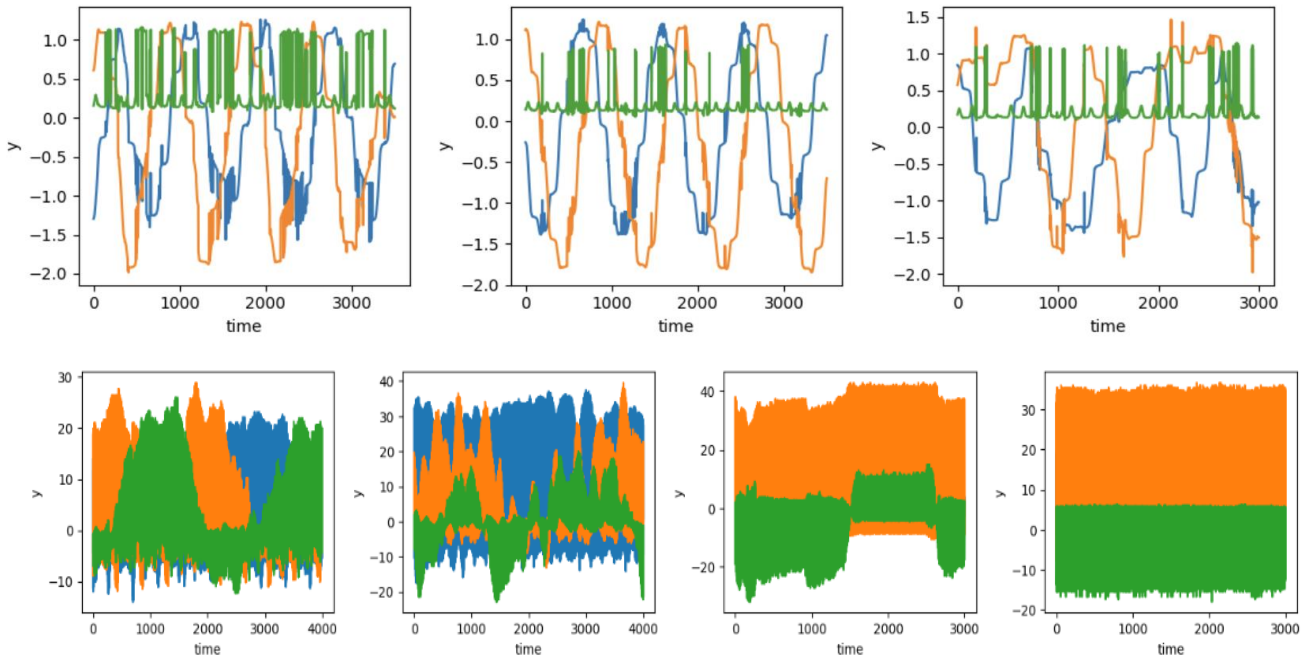
V:

In the training data, there are 8 different subjects. Some of the subjects in the test set weren't included in the training set. The train set contains about ~50K files, each representing a measured activity, while the test set contains ~75K files.

- **Part b**

The task at hand is a classification problem since the objective is to recognize activity by the given datasets. The goal is not to predict future events or additional information about past events but to accurately identify and classify activities in the provided data.

Plots representing samples of the data:



The plots show the change of x, y, z variables, measured over time in first picture and show the change of the acceleration measured over time in the second picture. Our objective is to create a model capable of classifying past events as shown above into activities.

- **Part c**

We will likely achieve the best results by fine-tuning the pre-trained time-series representation model since it is trained on large-scale datasets.

One possible self-supervised pre-training task is to feed the model with previous sequence as an input sequence and make it predict the next word of the sequence. This task can be used as a self-supervised task because the original data labels are not needed. We can drop the forecasting head of this model and use it to get latent representations of a given sequence.

Another option is training an auto-encoder model-based LSTM to compress a given sequence through a bottleneck using an encoder and then reconstructing the input from the encoding using a decoder. In this way, the model learns to create meaningful representations the data in an unsupervised manner of the

input sequence, potentially improving the model's performance on the classification task. In actual, we can drop the LSTM decoder and use the LSTM encoder to get those representations.

2 - Model

- **Part a**

Because the data is relatively large, using LOOCV or even K-FOLD CV will take too much time and use more computing power than is available.
Considering there is no apparent order between the training set's samples, we have decided to use a hold-out set that will be used as our validation.

- **Part b**

To create a naive baseline solution, we used the most frequent activity in the training set. The 'walking_freely' activity was the prediction for all the samples. In other words, we used the majority class of the training set as the prediction for all instances in the validation set. This approach is simple yet effective in establishing a lower bound for the model's performance. This solution achieved an accuracy score of **0.09** on both the training and testing set.

- **Part c**

After creating a naive baseline solution, the next step is to fit a classical machine learning model to some features extracted from the data to obtain a better and more solid benchmark for the neural network model. We trained 3 different classical ML models on features derived from the original dataset. The classical machine learning models used were XGBoost, Random Forest and SVM. We found that training different models for the two types of sequences (positions & acceleration) yields better results:

Model	Accuracy - positions	Accuracy - acceleration
XGBoost	0.619	0.804

Model	Accuracy - positions	Accuracy - acceleration
Random Forest	0.883	0.943
SVM	0.515	0.79

As expected, the classical ML models predicted better than naïve baseline and got very good results. The simplicity of these models, gives them to be very accurate. Moreover, intensive feature selection is needed to get the most out of the data and do hyperparameter tuning to find the best parameter for each model. Overall, the Random Forest model performs relatively well across all classes, followed by the XGBoost model.

- **Part d**

In our data, there are 18 possible activities (classes). We also have additional information available - body part, side and sensor. To allow the models to gain a better understanding of the input sequences, the models' output consists of 21 neurons - 18 of them are used for the prediction of the activity, and the other 3 predict the additional variables.
The loss of the first 18 values is the NLL loss (denoted as class_loss), and for the other three, we used BCE loss.
In addition, we decreased the sequences' granularity by averaging close time points into a single point, thus accelerating the learning process and simplifying it.
The stopping condition was val_class_loss didn't improve for 5 consecutive epochs.

1D-CNN

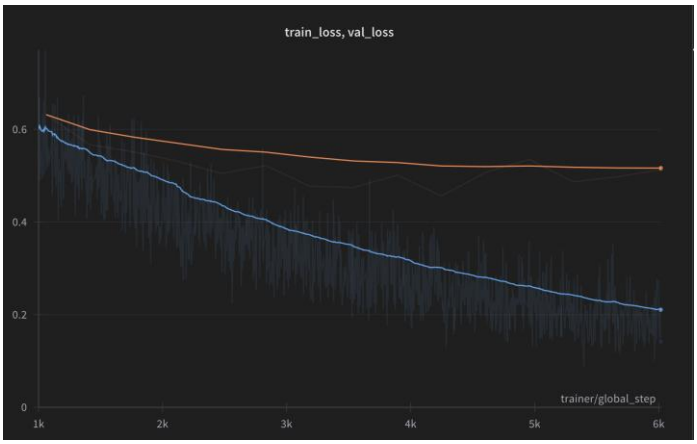
The 1D-CNN Model is a 1-dimensional convolutional neural network (1D CNN) for time series classification.
For our 1D-CNN model, we used the following architecture:

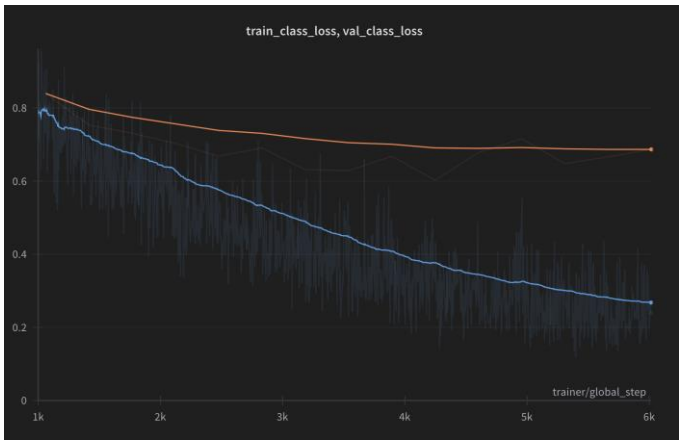
Layer	Parameters
Conv1d	3 -> 64, kernel size = 3, padding = 1
ReLU	-
MaxPool1d	kernel size = 2, stride = 2

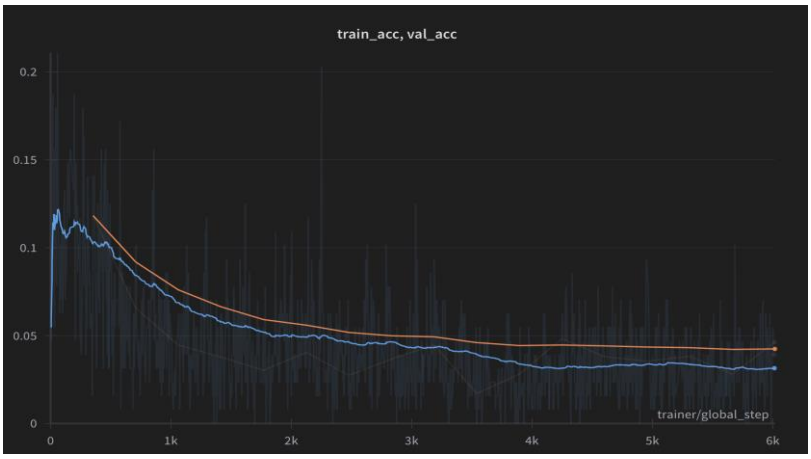
Layer	Parameters
Conv1d	64 -> 128, kernel size = 3, padding = 1
ReLU	-
MaxPool1d	kernel size = 2, stride = 2
Linear	12,800 -> 128
Linear	128 -> 21

The motivation behind using a 1D CNN for time series classification is to exploit the Model's ability to learn local and time-invariant features effectively while reducing computational complexity compared to higher-dimensional CNNs.

The learning curves of the above architecture:







The lowest train class_loss achieved is 0.201, and the lowest validation class_loss is 0.602. The model stopped learning after 17 epochs.

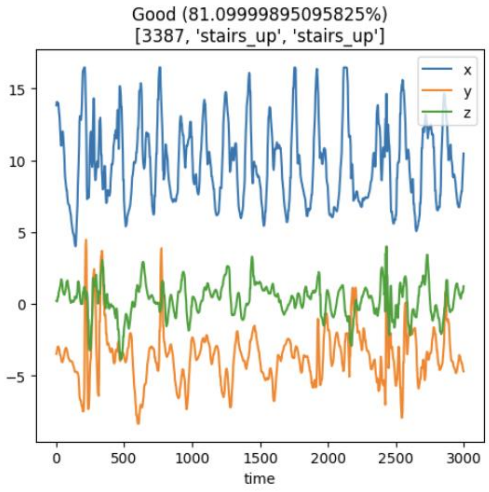
In addition, it looks like over time the model learns and decreases the loss value, but the accuracy of the model on the train as well as the validation set is decreased too (below 0.05 for both).

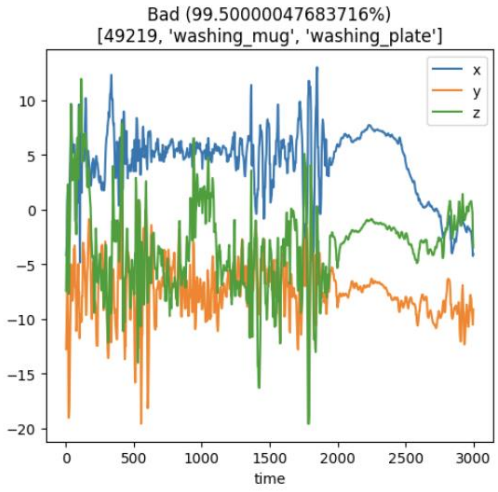
Results and experiments, organized in a table:

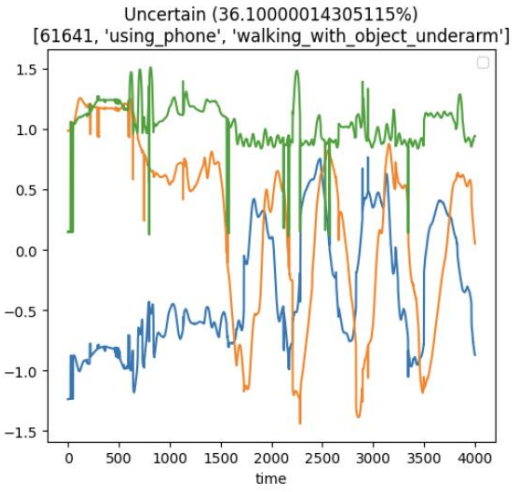
	1D-CNN v1	1D-CNN v2	1D-CNN v3
Convergence time	75m	58m	51m
Batch size	128	128	128
Learning rate	1E-03	1E-03	1E-03
Preprocessing	FFT	MinMaxScaling	FFT + MinMaxScaling
Epochs	17	13	11
Total train loss	0.173	0.618	0.155
Total validation loss	0.456	0.998	0.621
Train class loss	0.201	0.603	0.234
Validation class loss	0.602	1.348	0.807
Train accuracy	0.039	0.18	0.375
Validtion accuracy	0.046	0.138	0.3573

As we can see, the model performed the best when using only Fourier transformation on the input sequence.

Examples of good, bad, and uncertain classifications:







The model classified sample 3387 correctly with high confidence. We assume that's because walking up the stairs has a distinct pattern of movement compared to other activities. On the other hand, washing a mug and washing a plate are very similar activities that involve pretty much the same movements (washing the dishes in general), which explains why the model was confident in its classification.

Using a phone was classified as walking with an object underarm probably because the hand's posture is almost identical between the two, making the model misclassify the sample with a low level of confidence.

LSTM

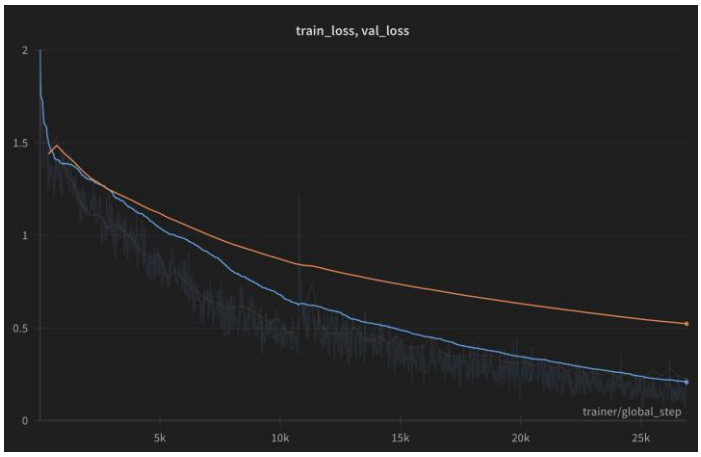
The LSTM Model is a recurrent neural network (RNN) based on Long Short-Term Memory (LSTM) cells designed for time series classification. LSTMs effectively capture and maintain long-range dependencies in the input sequences.

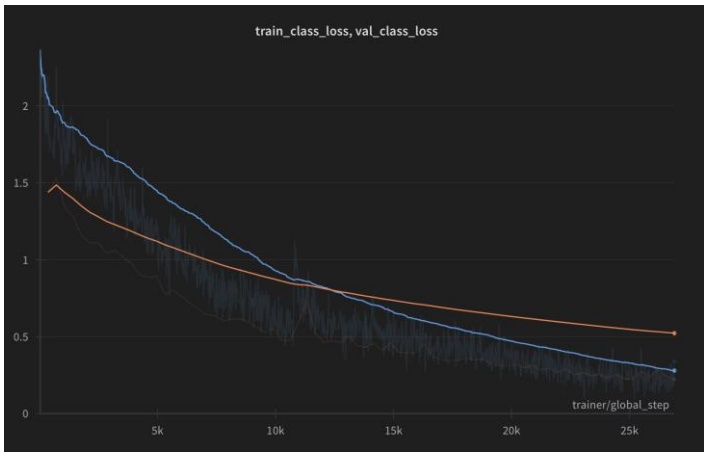
For our RNN architecture, we trained a bi-directional LSTM model with the following architecture:

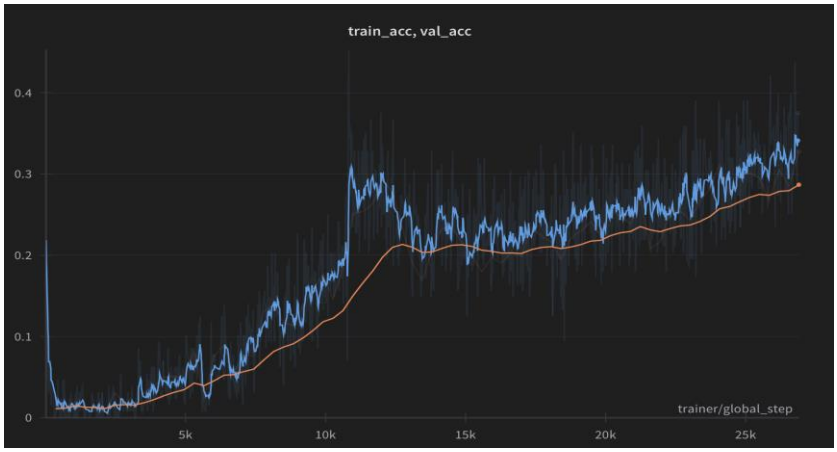
Layer	Parameters
LSTM	Input size = 3, hidden size = 128, 2 layers, 50% dropout
Linear	128 -> 21

The fully connected layer maps the final hidden state of the LSTM layers to the desired number of classes for classification.

The learning curves of the above architecture:







This model appears to perform better, reaching lower loss values and higher accuracy scores compared to the previous CNN-based model.
It has gained a deeper understanding of the data and its temporal patterns.

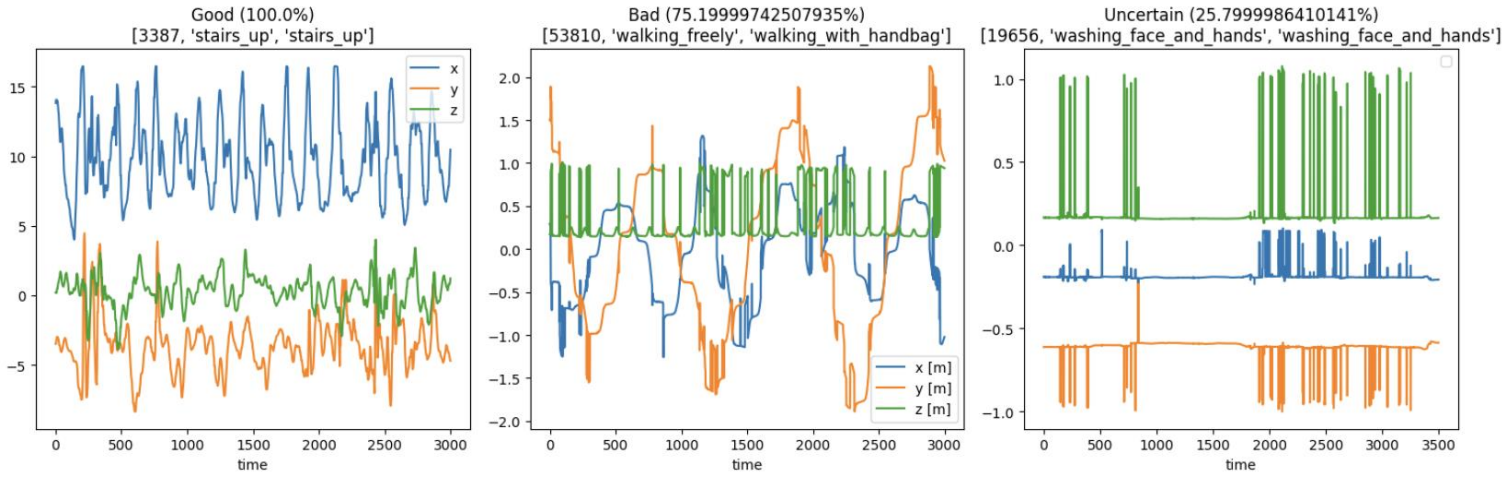
Results and experiments, organized in a table:

	BiLSTM v1	BiLSTM v2	BiLSTM v3
Convergence time	368m	245m	172m
Batch size	128	128	128
Learning rate	1E-03	1E-03	1E-03
Preprocessing	-	FFT	FFT + MinMaxScaling
Epochs	76	48	34
Total train loss	0.1998	0.1751	0.2839
Total validation loss	0.223	0.3712	0.453
Train class loss	0.339	0.3765	0.4177
Validation class loss	0.305	0.4921	0.589
Train accuracy	0.375	0.359	0.57
Validtion accuracy	0.3276	0.4072	0.587

The BiLSTM that learned without any preprocessing was able to extract the most information from the data and achieved the lowest loss value on the validation set, while the model that used both MinMaxScaling and FFT got the poorest loss results. Interestingly enough, just like the CNN-based model, the accuracy scores

seem to be correlated with the loss, which is counter-intuitive, because usually as the loss goes down, the accuracy gets better.

Examples of good, bad, and uncertain classifications:



The model correctly classified walking up the stairs with 100% certainty, just like our convolutional model. That supports our claim that this activity has unique patterns compared to the rest.

Walking freely was classified as walking with a handbag. We assume that it's because walking in general affects the signal in a specific way, giving it specific attributes that confuse the model.

The model correctly classified washing face and hands, but with low certainty. This class has a relatively low amount of labeled samples and that might have affected the model's confidence when classifying examples as such.

- **Part E**

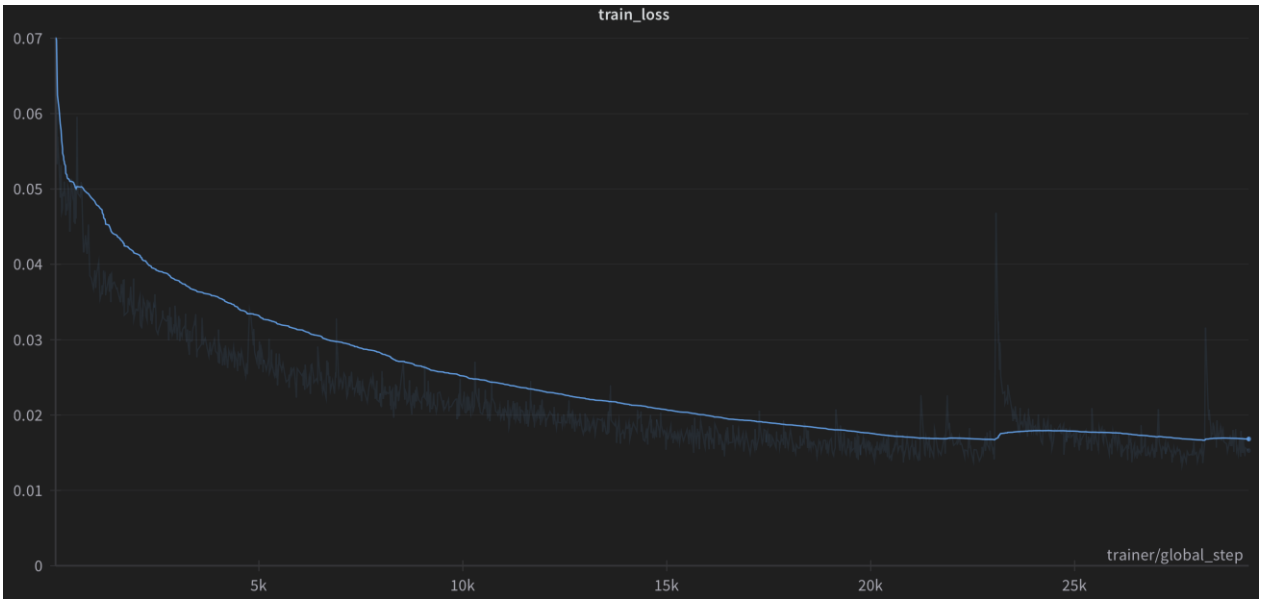
For this part, we chose to implement an auto-encoder that's based on the BiLSTM model shown in the previous section, using the following architecture:

Encoder layer	Parameters
LSTM	input size = 3, hidden size = 128, 2 layers, bidirectional

Decoder layer	Parameters
LSTM	input size = 256, hidden size = 256, 2 layers
Linear	256 -> 3

We trained the auto-encoder on all of the data, training and testing, using the MSE loss. Our goal was trying to get the model overfit the data and reduce the loss as much as possible. In that way, we tried to achieve an encoder that outputs meaningful latent representations for input sequences taken from our data.

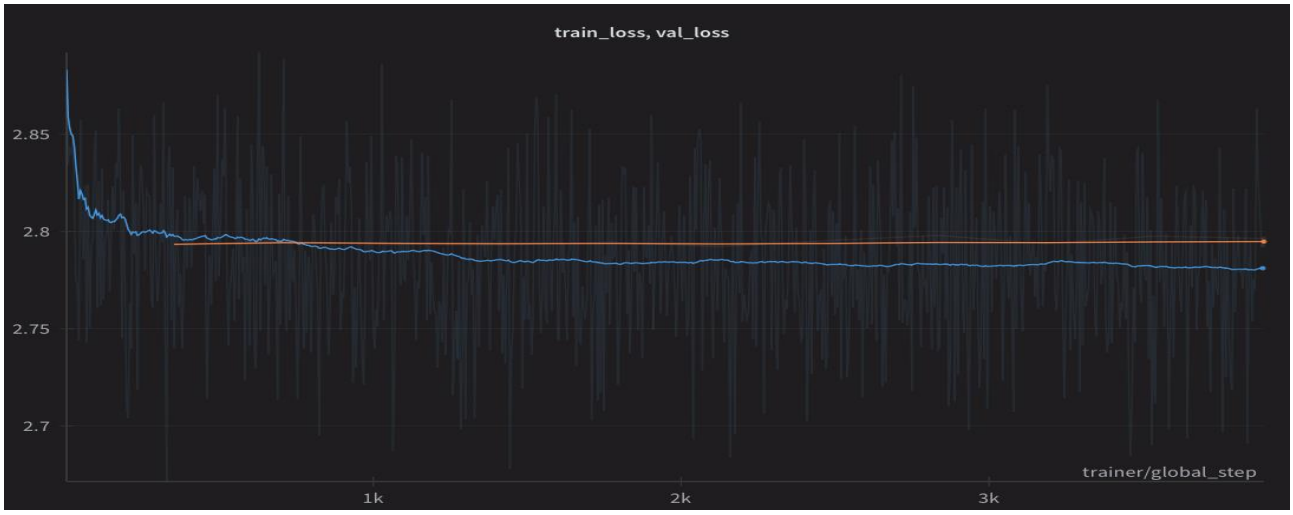
The auto-encoder's learning curve:



Results and experiments, organized in a table:

	Autoencoder, dim 128	Autoencoder, dim 64	Autoencoder, dim 32	Autoencoder, 1 layer, dim 64
Convergence time	91m	25m	27m	20m
Batch size	128	128	128	128
Learning rate	1E-03	1E-03	1E-03	1E-03
Epochs	30	24	30	23
Loss	0.015	0.02	0.0206	0.0243

The 2 layered auto-encoder with an embedding dimension of 128 got the best results, so we chose to move on with it to the next stage, of using the encoder as a feature extractor and classifying the features with an additional linear layer.



We trained the encoder + FC layer with the NLL loss on our labeled training data, and got the following learning curves:
The best validation loss we got was 2.792, indicating that our classifier had failed to understand the training data sufficiently. Compared to the previous results we got, the current model is as twice as bad compared to our poorest-performing model.

- **Part F**

As we mentioned, the fully connected layer has failed to model the connection between the embedding vector to the activity/class. Generally, as we learned in class, when we have a set of features, classical ML models will usually work better compared to deep learning methods.

Our 3 suggestions for improving the model are:

1. Classifying the embedding vector using a classical ML model.
2. Instead of using the raw data, normalizing or standardization the embeddings should improve the results.

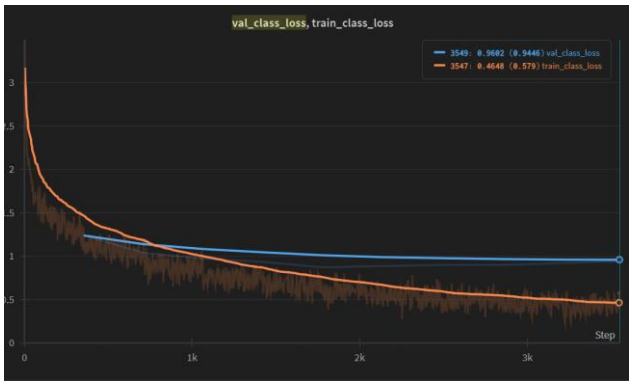
3. Deepening the classifier linear layer using additional layers, thus increasing the complexity and the model's ability to comprehend patterns in the data.

- **Part G**

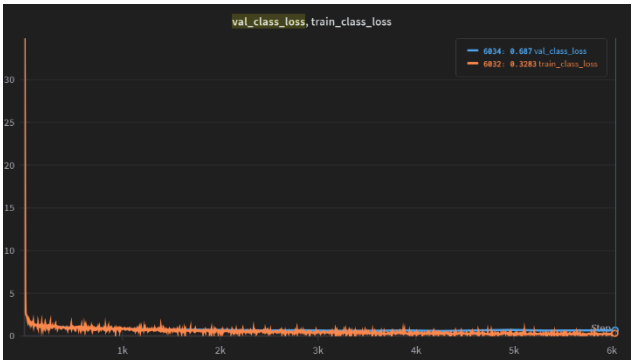
We have implemented the first 2 improvement suggestions.

Using a data frame containing the embeddings created by the encoder for each of the training samples, we trained a XGBoost model instead DL model that give us low results and improve the results because of that.
Then, we preprocessed the inputs using a MinMaxScaler or FFT or both and trained an XGBoost model on them.
The accuracy score we got was **0.086** using a MinMaxScaler, a poor result compared to the original XGBoost we trained, which gained an accuracy score of 0.711, and the FFT show better results then the original as you can see in section C.
The normalization and standardization compared to most of the other methods we've tried (all LSTM + one CONV) as you can see above in the tables.

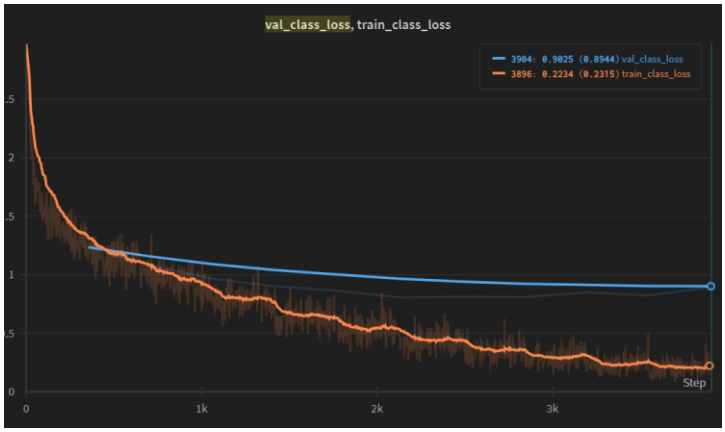
<u>Here we show the visualizations of each Conv1D and LSTM version:</u>
Conv1D 2 FC without normalization and FFT:



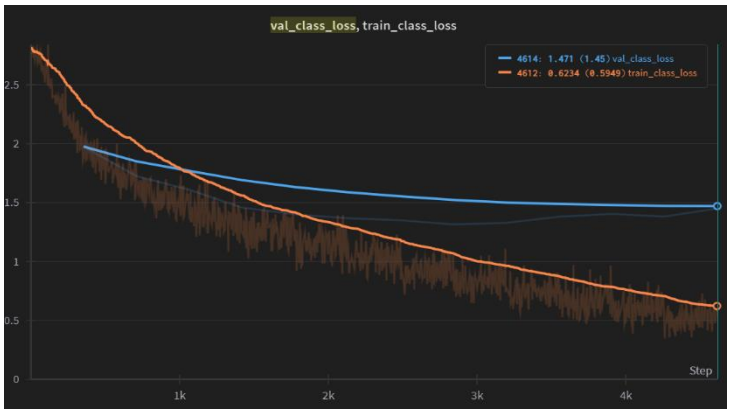
Conv1D 2 FC with FFT only:



Conv1D 2 FC with FFT + normalization:

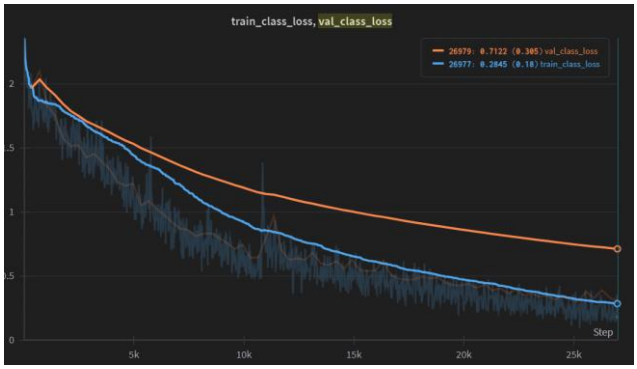


Conv1D 2 FC with normalization only:

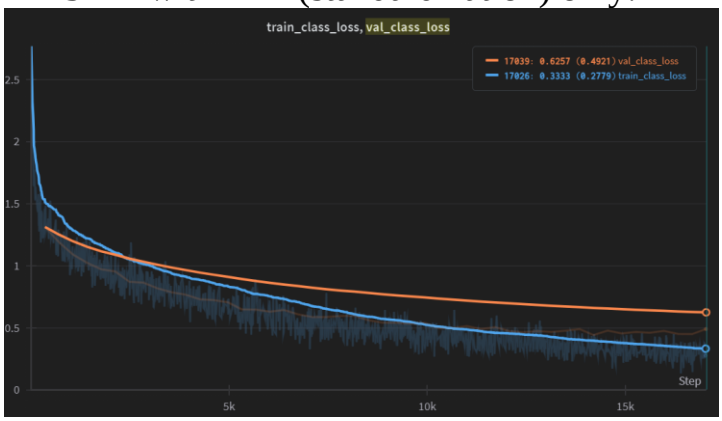


When comparing the four cases for the Conv1D model, the loss values for conv1d with stand. appear to be lowest on training and validation and more. This suggests that using a furrier transformation can help the model learn more efficiently and make better predictions.

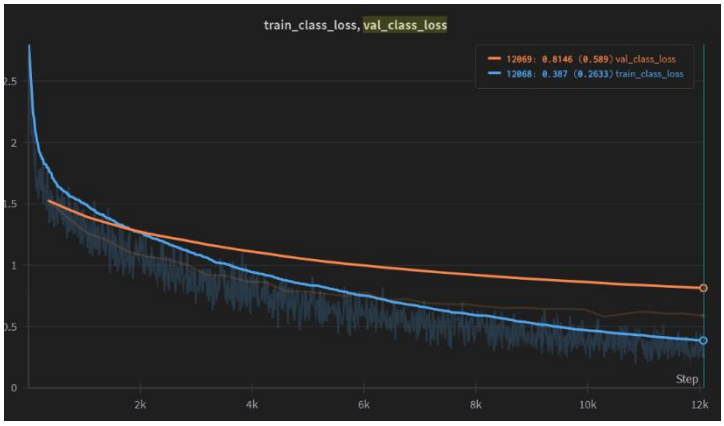
BiLSTM without normalization and standardization:



BiLSTM with FFT (standardization) only:



BiLSTM with normalization and standardization:



When comparing the three cases with and without and only with furrier transformation. The validation loss and the training loss is generally lowest for only FFT than for other models.

As we can see from the tables above and from visualizations, our initial LSTM and 1DCNN models achieved poor results in this time series task. However, we can also see that classical LSTM models, such as XGBoost, got better results than these models. Moreover, we can see that our autoencoder gained very poor results when we trained on the unlabeled data. In addition, improving the LSTM architecture increased our results.

We conclude that the embeddings themselves weren't informative enough and the auto-encoder hasn't learned to represent our data as well as we thought it did. In addition, the normalization and standardization worked well in some cases and in some cases worked very bad.

Summary

We started by creating a naive baseline that scored an accuracy of 0.09. Then we used several mathematical methods to extract features from the sequences and trained ML classifiers on them. The highest accuracy score we got was 0.91.

Later, we developed through trial and error 2 models - convolution-based and RNN-based.
For the convolution-based model, we experimented with models with different amounts of layers and different layer combinations. The best-performing convolution model we achieved had 2 Conv1d layers with ReLU and MaxPool1d followed by 2 Linear layers.
For the RNN-based model, we've performed the same process, trying one-directional and bidirectional LSTMs with different combinations of num_layers, dropout values, and hidden sizes. The best-performing LSTM had 2 layers, a hidden size of 128 and a 50% dropout.

Then we tried pre-training an auto-encoder to provide meaningful feature vector representations of our input sequences. We experimented with different architectures (conv, LSTM) with different embedding sizes. Our best-performing auto-encoder was LSTM-based with an embedding dim of 128.
Although we got seemingly good results while training the auto-encoder, we failed trying to achieve good classifications using the embeddings. Apparently, the representations aren't as good as we thought, and reducing the granularity has probably made the results worse.

Overall, the method that got the best validation loss was the BiLSTM with FFT only, but its accuracy was not as good.

Here is a link to our weights and biases to show all our experiences: WANDB