SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.039 Deep Learning

Final Project Report
Group 19 (Inertial-Based Real-Time Human Action Recognition For Physiotherapy)

Github Repo: Link

Zach Lim	1002141
Soh Pei Xuan	1005552

Content Outline

1. Introduction

Human kinetic analysis through automated interpretation of movement data has grown as an important application of deep learning. Models trained on data from wearable sensors and other modalities have shown potential for gaining meaningful insights into human biomechanics, activities, and performance over time. Deep learning has demonstrated potential in applications such as motion capture, action recognition and rehabilitation monitoring.

In this project, we sought to develop a physical therapy exercise classifier using inertial sensor data from the PHYTMO (PHYsical Therapy MOnitoring) dataset [1], which contains inertial measurement unit (IMU) sensor recordings of common physical therapy exercises .

2. Problem Definition

2.1. Task Definition

Our task is a many-to-one classification problem, with high-dimensional time series data (in the form of accelerometer, gyroscope and magnetometer readings) as input, and a single class prediction as output.

2.2. Dataset

The PHYTMO Database contains data from physical therapies recorded with four IMUs and an accurate optical system known as OptiTrack. Wearable sensors include 3-axis gyroscope, accelerometer and magnetometer, with a range of 2000°/s, 16 g and 1,300 μT, respectively [1]. It encompasses the recordings of 30 volunteers, aged between 20 and 70 years old, performing a total of 6 exercises and 3 gait variations recorded. The volunteers performed two series with a minimum of 8 repetitions in each one.

The database contains both inertial and optical measurements of 9 exercises (3 Upper Body, 3 Lower Body and 3 Gait-Focused), with inertial data coming from Inertial Measurement Units (IMUs) placed at 8 locations on the body (Right/Left Shin/Thigh/Wrist/Upper Arm).

While the PHYTMO dataset provided both inertial and optical data, our group chose to focus on using the inertial measurements only. IMUs are portable, affordable devices that do not require specialised equipment or dedicated motion capture spaces to collect data. This makes inertial sensors much more accessible and feasible for widespread data collection and model testing in real-world settings. The inertial data in the PHYTMO dataset consists of 3D measurements of linear acceleration, angular velocity and magnetic field, along with timestamps, from wearable IMUs.

To optimise our use of limited time for model training, we have chosen to exclusively utilise the upper arm (i.e. left and right arm) data in our project. By narrowing our focus, we can simplify the problem space and allocate more time to refining model architectures and hyperparameters. Additionally, the upper arm serves as a representative region for capturing movement patterns during upper body exercises, providing valuable insights within our exercise domain.

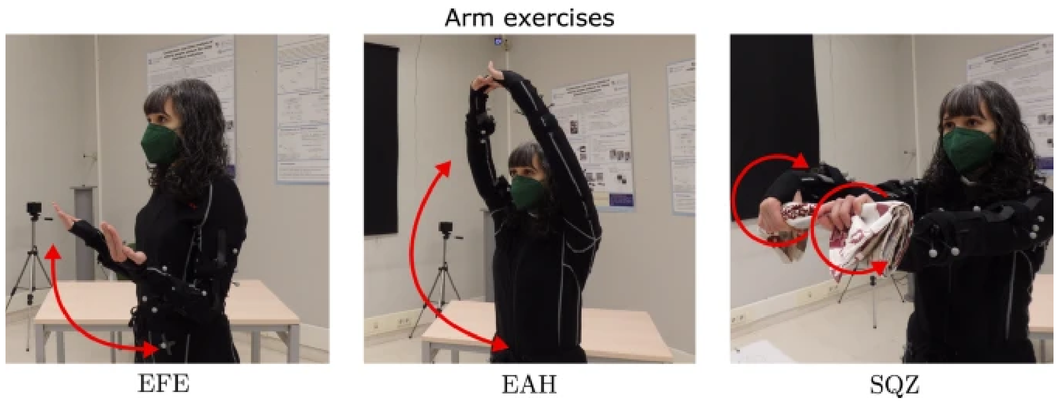

Figure 1: Sample images of upper arm exercises performed

Primarily, the 3 exercises that are recorded with the upper arms are elbow flex-extension (EFE - Class 0), extension of arms above head (EAH - Class 1) and squeezing (SQZ - Class 2). The dataset contains examples of both the exercises performed correctly and incorrectly. All of the incorrectly performed actions were labelled into a single class (Class 3).

2.3. Proposed Methodology

Our proposed solution is to train an Autoencoder Model, and use the features extracted from the Encoder portion to train a separate classifier. This architecture was chosen because

Autoencoders are very powerful at compressing a multidimensional sequence of time series data down into a single vector for classification. Also, this method is able to better capture long-term dependencies in the data, which is problematic for standard LSTMs (especially for long sequences).

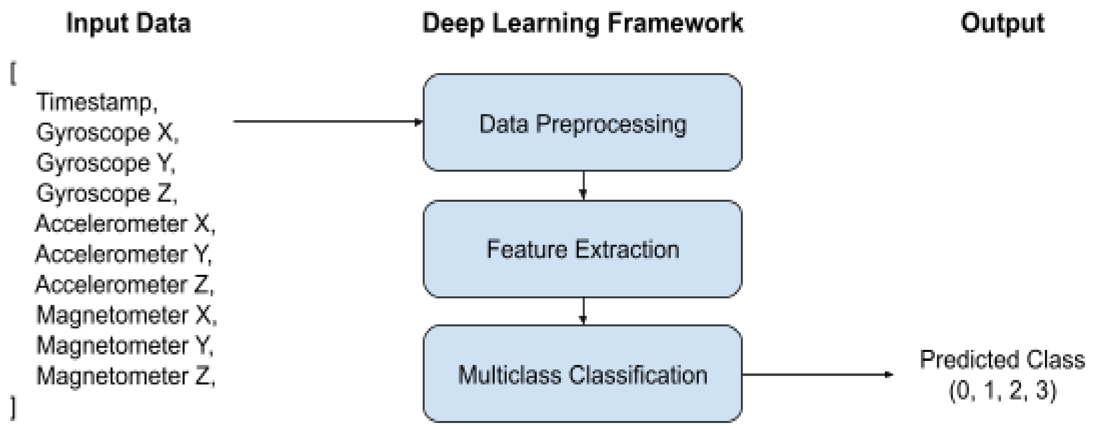

Figure 2: Overview of our solution workflow

Our solution involves 3 main stages: Data Pre-Processing, Feature Extraction and Multiclass Classification.

2.3.1. Data Pre-Processing

Before training models on the IMU data, some pre-processing steps were required. Recordings in the PHYTMO dataset files contained periods at the beginning and end that captured the subject transitioning into and out of the exercise motion, rather than the movement itself. This initial and final "dead data" without meaningful kinetic content was trimmed from each recording. An example of such data can be seen in Figure 3 below.

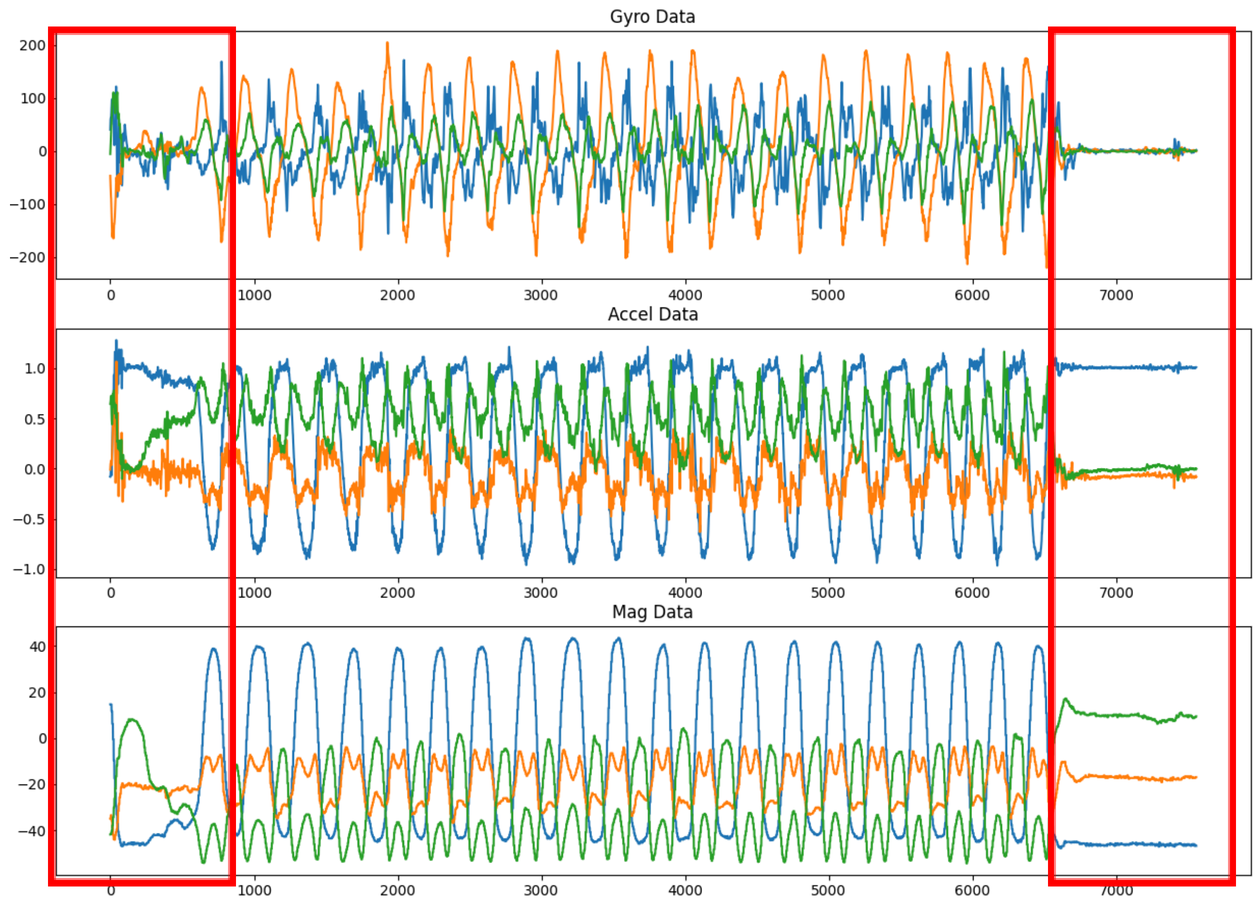

Figure 3: Example of original inertial data with 'dead' data at the start and end

The data is then downsampled from the original sample rate of 100Hz down to 20Hz, by taking the mean of every 100/20 = 5 datapoints. In addition to reducing computation costs, this step also helps to reduce the effect of outliers on our data.

The data is then scaled using Min-Max Scaling, which scales each feature to between 0 and 1, according to the minimum and maximum values within that feature range.

Finally, the data is 'windowed' into segments that are 3s long, at an interval of 0.5s. These windows were then concatenated to form our dataset. The window size is chosen as that is about the time it takes to perform one cycle of the longest action in our dataset (EAH).

Dataset	Train	Test	Validation
L_Arm	11,578	3,308	1,654
R_Arm	11,582	3,310	1,655

Figure 4: Train-test-validation split distribution for our dataset

To evaluate our models, the dataset was randomly split with a fixed seed into training, test, and validation sets in a 7:2:1 ratio. Figure 4 provides a breakdown of the number of samples allocated to each set after partitioning the data. This distribution allowed us to sufficiently train our models on a large portion of the data while maintaining independent subsets to tune hyperparameters and assess generalizability.

2.3.2. Feature Extraction

We proposed multiple methods for the feature extraction (Encoder) and classification stages, which are listed as follows:

Feature Extraction Models (Encoder)	Classification Models
LSTM	SVM
CNN-LSTM	MLP

Figure 5: Various feature extraction and classification models explored in this project

A more detailed analysis on the models used can be found in the following section.

The primary setup of our encoder training procedure consists of the following hyperparameters:

- Optimizer: Adam
- Loss Function: MSE Loss
- Number of Epochs: 50
- Early Stopping: Patience = 5, Min Delta = 0

The Adam optimizer was used for model training as it has been shown to be highly effective for a broad range of deep learning problems. Mean squared error (MSE) loss was used as our task was a regression problem of predicting continuous IMU values. Through preliminary experiments, 50 epochs were determined sufficient as we observed that the training loss began to plateau thereafter. Early stopping with patience of 5 epochs (which is about 10% of the overall number of training epochs) and minimum delta of 0 was applied to prevent overfitting by monitoring validation loss.

2.3.3. Classification

We experimented with two classifiers: Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM). Given that our task involves multi-class classification, the loss function we employed is the cross-entropy loss function.

For our testing, we used a fixed CNN-LSTM encoder model to output encoded data. This encoded data was then used to train our proposed classifiers. The MLP training was run with the following parameters:

- Optimizer: Adam
- Loss Function: Cross Entropy Loss
- Number of Epochs: 50
- Layer sizes: Variable
- Early Stopping: 5 (Min delta of 0)

The results of our testing is summarised in the table below:

Classifier	**SVM**	**MLP (360-64-16-4)**	**MLP (360-128-32-4)**	**MLP (360-64-4)**
Accuracy on Test Set (%)	**97.22**	95.79	93.89	95.90

Figure 6: Accuracy results from training on different classifiers

We found that the SVM with the Radial Basis Function (RBF) kernel consistently outperformed the MLP variations that we tried. Hence, we used the SVM Classifier for all subsequent testing going forward.

3. Experimental Evaluation

After training, the following folders contain the respective data:

- `/images`: contains all plotted graphs for loss + confusion matrix
- `/logs`: contains all training logs
- `/models`: contains all trained and saved model weights

- `/notebooks/training`: contains all notebooks used for training various models
- `/tensorboard`: contains all tensorboard logs from training

3.1. Long-Term Short Memory (LSTM)

3.1.1. Model Structure

Long short-term memory (LSTM) networks were employed to model the temporal dependencies in the inertial sensor data. LSTMs are a type of recurrent neural network well-suited for sequence modelling tasks, as they contain memory cells that allow the model to retain information for longer durations compared to standard RNNs. This 'long term memory' capability makes LSTMs effective at learning patterns in time series data by capturing long-range temporal dependencies.

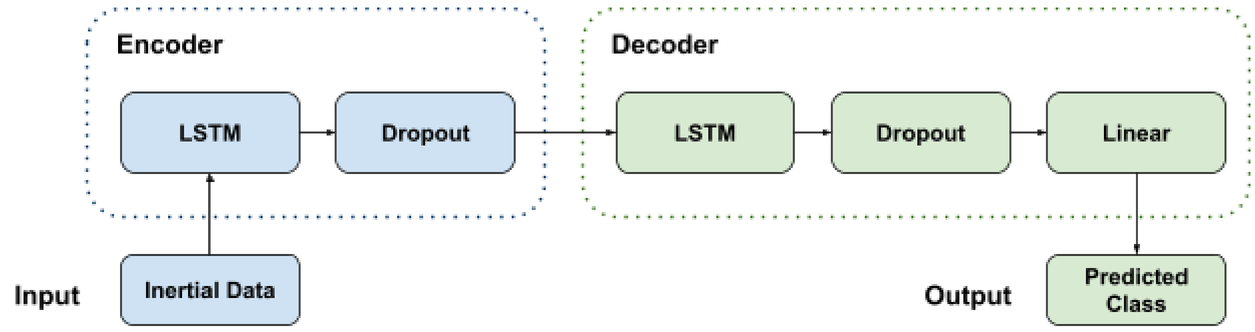

Figure 7: Proposed LSTM autoencoder architecture

The proposed LSTM autoencoder architecture is depicted in Fig 7. To ensure a consistent feature space across the encoding and decoding stages, both stages employ an LSTM layer with a hidden size of 360 cells, a relatively large hidden size to provide the model with more capacity to learn complex patterns.

3.1.2. Experiment Results

To identify an optimal model configuration, we conducted experiments with varying hyperparameters. Specifically, we explored LSTM architectures with varying number of layers, bidirectionality, dropout rate, and learning rate. All networks utilised early stopping with a patience of 5 epochs and minimum delta of 0.

No	No of LSTM Layers	Bidirectional	Dropout	Learning Rate	SVM Test Accuracy (L_Arm)	SVM Test Accuracy (R_Arm)
1	1	False	0	0.001	95.71	95.65
2	1	True	0	0.001	95.53	95.65
3	2	False	0	0.001	95.25	94.41
4	2	True	0	0.001	96.89	96.47
5	**1**	**True**	**0.1**	**0.001**	**97.07**	96.31
6	**1**	**True**	**0.2**	**0.001**	**95.71**	**96.53**
7	1	True	0.3	0.001	96.80	96.16
8	2	True	0.1	0.001	96.92	95.86
9	2	True	0.2	0.001	96.67	95.74
10	2	True	0.3	0.001	96.43	96.01
11	1	True	0.1	0.1	49.43	50.48
12	1	True	0.1	0.01	97.25	96.44
13	1	True	0.1	0.05	49.43	50.48
14	2	True	0.1	0.1	49.43	50.48
15	2	True	0.1	0.01	49.43	58.07
16	2	True	0.1	0.05	49.43	50.48

Figure 8: Training performance for LSTM hyperparameter tuning with 1 linear layer

To investigate whether capturing high-level features could boost performance even further, we used the most optimal model configuration (Run 5 and 6) and experimented with augmenting the decoder with an additional linear layer after the LSTM, as we hypothesised that this may allow the model to better learn abstractions of input data.

No	No of LSTM Layers	Bidirectional	Dropout	Learning Rate	SVM Test Accuracy (L_Arm)	SVM Test Accuracy (R_Arm)
17	1	True	0.1	0.001	96.67	96.83
18	1	True	0.2	0.001	96.65	96.59

Figure 9: Training performance for LSTM with 2 linear layers

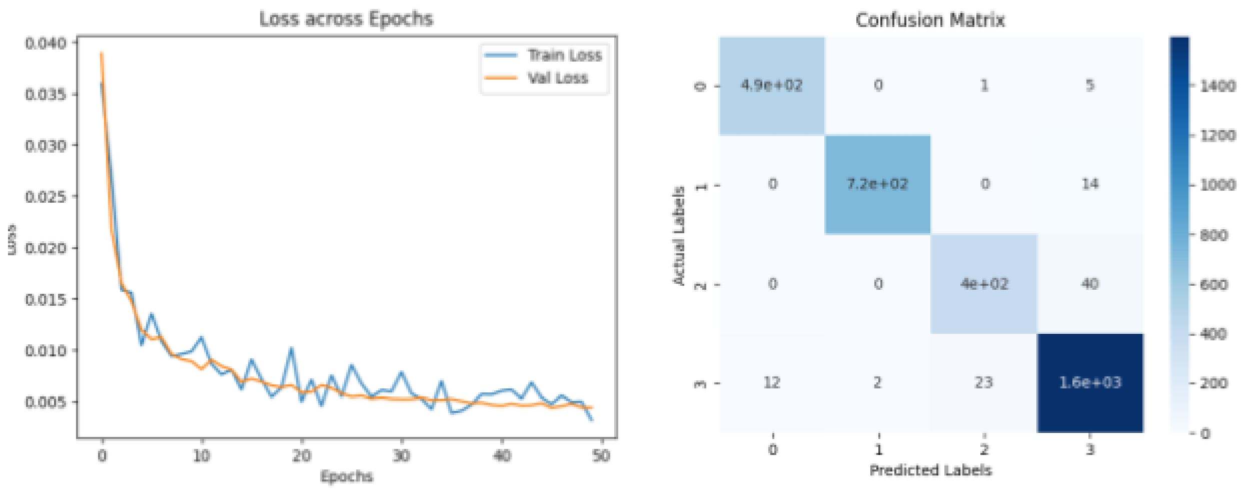

Figure 10: Loss and confusion matrix for Run 5, based on l_arm dataset

3.1.3. Experiment Findings

Overall, the following observations were derived:

- When comparing bidirectionality of LSTMs, bidirectional LSTMs were found to perform better than unidirectional LSTMs. This is because bidirectional LSTMs are able to get context from both past and future data, which seems to improve its ability to predict over a unidirectional LSTM.
    - Although our data is streamed, predictions are run on 'windows' of data 3 seconds long, hence using a bidirectional LSTM does not increase latency.
- Implementing some amount of dropout at 0.1 helped improve overall performance.
- Increasing learning rate beyond 0.001 generally resulted in poor accuracy, suggesting that any further increase in learning rates might have resulted in overfitting of data.
- Adding an additional LSTM or linear layer did not seem to improve accuracy, which suggests that the additional layer might have increased the complexity of the decoder beyond what is necessary for the task at hand

The model configuration and training script for the LSTM model can be found in the `LSTM_Larm_train.ipynb` and `LSTM_Larm_train.ipynb`, which are located in the `/notebooks/training` folder.

3.2. CNN-LSTM

3.2.1. Model Structure

While CNNs are commonly used for image processing, they can also be useful for processing inertial data, which can be represented as an array of 9 stacked 1-dimensional sequences. Since the 9 sensor readings are not independent, local patterns and correlations presented could be captured by a CNN. By applying convolutional operations, CNNs can extract both these ‘pseudo-spatial’ and temporal features from the IMU readings. This makes CNNs a viable option for capturing important patterns and features in the IMU data [2].

Thus, we hypothesise that a combination of CNN and LSTM, known as CNN-LSTM, can leverage the strengths of both architectures. CNNs can be used as feature extractors to capture local patterns in the IMU data, while LSTM models can handle the temporal dependencies and capture longer-term patterns. By combining CNN and LSTM, the CNN-LSTM model can effectively capture both local and global temporal patterns in the IMU data, potentially leading to improved performance compared to using either model independently.

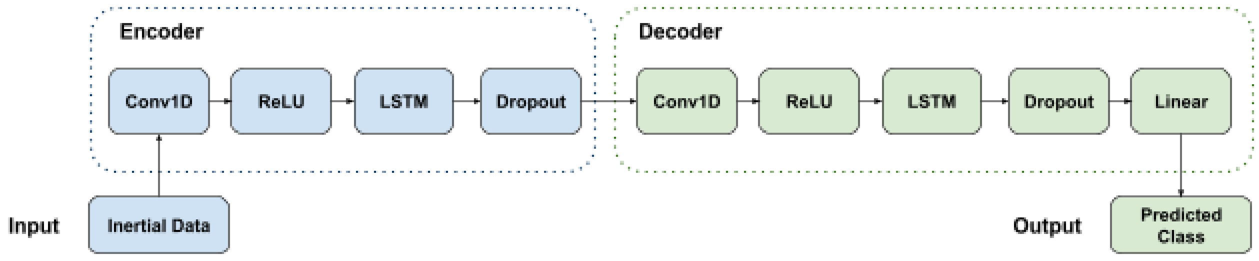

Figure 11: Proposed CNN-LSTM autoencoder architecture

The proposed LSTM autoencoder architecture is depicted in Fig 11. To maintain a consistent feature space throughout the encoding and decoding stages, we incorporated a Conv1D layer with 64 filters, leveraging the power of 2 for its suitability in deep learning models. The kernel size of 3 and padding of 1 was chosen to maintain the same length across encoding and decoding stages, whilst enabling the model to capture local dependencies between the different parameters through a small window. Similar to Section 3.1, the LSTM layer has a hidden size of 360.

3.2.2. Experiment Results

To identify an optimal model configuration, we conducted experiments with varying hyperparameters, such as the number of LSTM layers, bidirectionality and dropout rate. As varying learning rate did not derive variable results in Section 3.1, it was not pursued here. All networks utilised early stopping with a patience of 5 epochs and minimum delta of 0.

No	No of Conv Layers	No of LSTM Layers	Bidirectional	Dropout	Learning Rate	SVM Test Accuracy (%) (L_Arm)	SVM Test Accuracy (%) (R_Arm)
1	1	1	False	0	0.001	95.12	94.68
2	**1**	**1**	**False**	**0.1**	**0.001**	**97.22**	**96.89**
3	**1**	**1**	**False**	**0.2**	**0.001**	**97.28**	**96.34**
4	1	2	False	0	0.001	90.39	92.39
5	1	2	False	0.1	0.001	93.02	93.41
6	1	2	False	0.2	0.001	92.41	91.87
7	1	1	True	0	0.001	94.83	94.47
8	1	1	True	0.1	0.001	96.55	96.28
9	1	1	True	0.2	0.001	96.95	96.65
10	1	2	True	0	0.001	95.62	94.47
11	1	2	True	0.1	0.001	95.86	94.20
12	1	2	True	0.2	0.001	95.89	94.68

Figure 12: Training performance for CNN-LSTM hyperparameter tuning

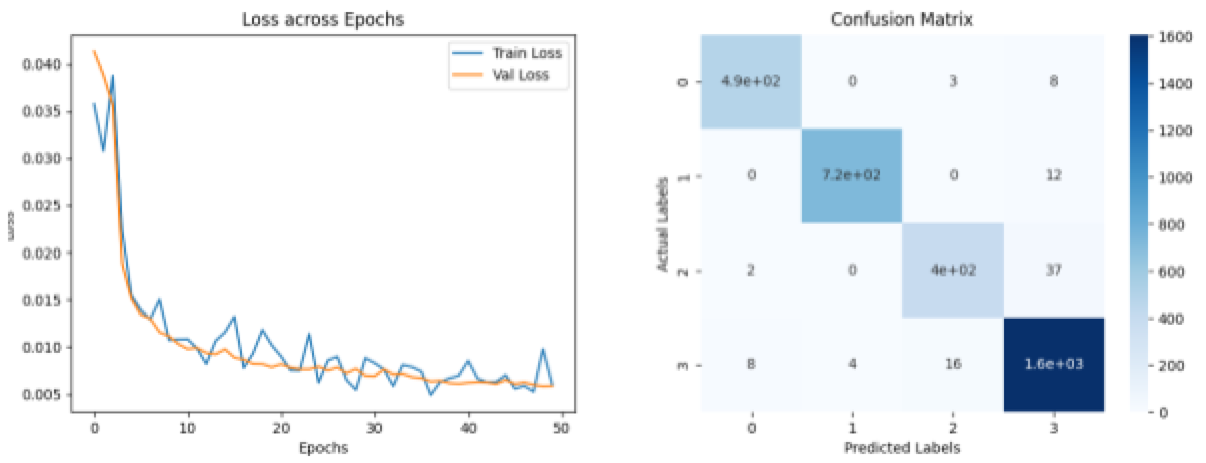

Figure 13: Loss and confusion matrix for Run 3, based on l_arm dataset

3.2.3. Experiment Findings

Overall, the following observations were derived:

- Unlike the LSTM models, bidirectional LSTMs mostly did not perform better than its unidirectional counterpart, suggesting possible overfitting due to the overcomplexity of the model with an additional CNN layer.
- Implementing some amount of dropout helped improve overall performance.
- Ultimately, it seems like adding any additional layers or bidirectionality did not seem to improve the model, which could suggest that any additional complexity would most likely result in overfitting and thus poorer performance.

The model configuration and training script for the LSTM model can be found in the `CNN_LSTM_Larm_train.ipynb` and `CNN_LSTM_Larm_train.ipynb`, which are located in the `/notebooks/training` folder.

3.3. Comparison against SOTA

There have not been many recent advancements in our particular problem area, but the current State-of-the-Art model for predictions on time-series data (excluding Transformers) is a Temporal Convolutional Network (TCN). A TCN is a network that utilises 1D convolution of sequential data together with dilation to increase the receptive field size while keeping the number of parameters relatively low. It has the added benefit of being an all-in-one solution, able to directly output a prediction without the need for training two separate models.

TCNs come in two forms: Causal and Non-Causal. Causal TCNs use only data from the past, as shown in Figure 14 below, while Non-Causal TCNs use information from both past and future, as shown in Figure 15. This is analogous to the Uni- and Bidirectional LSTMs discussed above.

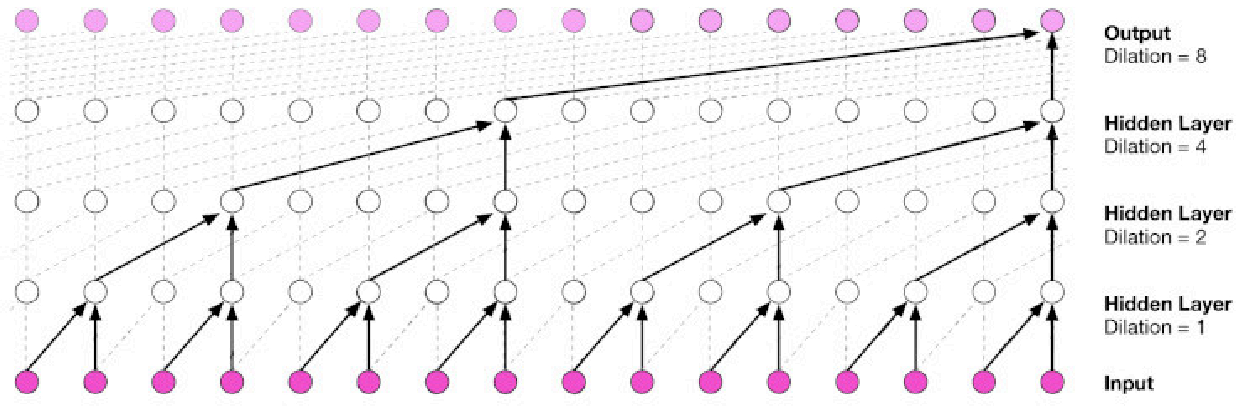

Figure 14: Causal TCN [3]

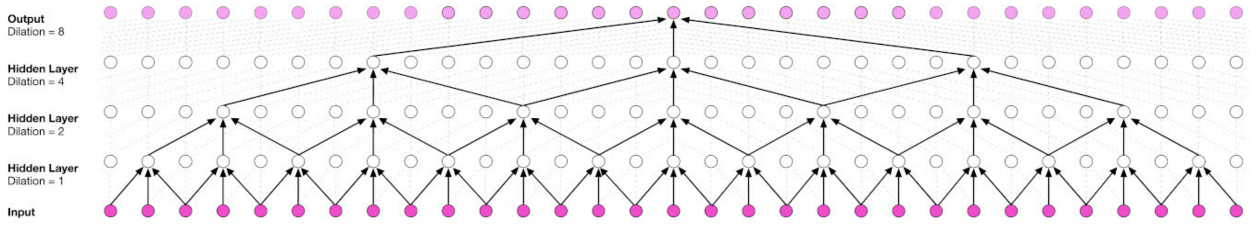

Figure 15: Non-Causal TCN [3]

For a quick comparison, we used an existing implementation of a TCN for which to compare our results against [3]. The parameters used to train the TCNs were as follows:

- Optimizer: Adam
- Loss Function: Cross Entropy Loss
- Number of Epochs: 15
- *Feature sizes: [128, 64, 32, 16, 4]
- Dilation Degree: 2^(1...n) for the residual blocks 1 to n
- Early Stopping: 5 (Min delta of 0)

**Feature sizes refer to the number of feature channels used in each residual block.*

To note, the implementation of Causal TCNs used for comparison did not accept batched inputs. Hence, training the TCN took a significant amount of time(~20 mins).

Both Causal and Non-Causal TCNs were tried, along with optionally including skip connections from the intermediate outputs of each layer directly to the final output.

TCN Variation	**Causal**	**Causal w Skip**	**Non Causal**	**Non Causal w Skip**
Accuracy (%)	**97.3**	95.5	83.8	93.7

Figure 16: TCN Model Results

The results in Figure 16 show that our model performs extremely favourably against the current state-of-the-art models, while being much faster to train. That being said, the TCN models were not tuned as thoroughly as our model, hence there is a possibility that the TCN models could still outperform ours.

4. Future Work

This model is intended to be used in a real-time application. However, the implementation of the actual real-time streaming service was beyond the scope of this project.

Class imbalance in the dataset could also be addressed for future improvements. For this project, it was not deemed necessary due to the high accuracy scores achieved even without it, but it could potentially further increase the accuracy of our predictions.

5. Contributions

Pei Xuan	Hyperparameter Tuning, CNN, LSTM + CNN Model Training + Evaluation
Zach	Data Preprocessing, LSTM Autoencoder, Classifier, SOTA Comparison

References

[1] García-de-Villa, S., Jiménez-Martín, A. & García-Domínguez, J.J. A database of physical therapy exercises with variability of execution collected by wearable sensors. Sci Data 9, 266 (2022). https://doi.org/10.1038/s41597-022-01387-2

[2] Mijanur Rahman. Different ways to combine CNN and LSTM networks for time series classification tasks (2022).
https://medium.com/@mijanr/different-ways-to-combine-cnn-and-lstm-networks-for-time-series-classification-tasks-b03fc37e91b6

[3] Paul Krug. (Realtime) Temporal Convolutions in PyTorch (2023).
https://github.com/paul-krug/pytorch-tcn