1. Introduction

Human action recognition is an important task which involves using sequential data to detect human activity. One of the challenges is to deal with class imbalance, since certain activities happen more frequently than others. Hence, the goal of this project is to design a machine learning system that is suitable for detecting imbalanced human actions. 9388 sequences of 3D skeleton data were provided, with each sequence consisting of 16 frames and each frame containing 20 skeleton joint features. The skeleton joint features were derived from measurements taken along the x, y and z axis.

2. Project challenges

2.1 How to classify sequential data

There are a number of approaches to sequential data classification (Table 1) each with its own advantages and disadvantages. For example, the distance-based approach is very time consuming and the neural network is less interpretable. Having reviewed the project specification and relevant literature on human action recognition, we decided to design deep learning based systems. Deep learning has produced good results consistently across different datasets in the literature, and could therefore potentially achieve high accuracy. The project specification does not require the results to be interpretable to other stakeholders and hence the 'black-box' nature of deep learning models is less of an issue.

Table 1: Sequential Data classification approaches

Distance based	Feature based	Neural Network
Stage 1: Distance between pairs of sequential data is calculated e.g., dynamic time warping method Stage 2: Classification e.g., knn	Stage 1: Feature generation e.g., hand crafted features / statistical features / sliding window Stage 2: Classification e.g., SVM and trees.	Feature generation (through kernels) and classification are performed at one stage

2.2 How to handle imbalanced datasets

According to our research, the two main approaches for handling imbalanced datasets are re-weighting and re-sampling based on the number of observations of each class. For the given dataset, our initial experiments showed that re-sampling outperformed re-weighting. However, there are other more advanced re-weighting techniques e.g. effective number of samples based approach [1]. However, due to time constraints we did not explore this further.

When re-sampling, over sampling was adopted rather than down sampling. This was because having more samples will generally improve the performance of neural networks due to their inherent complexity. In particular, we used SMOTE-Tomek Links method which consists of two stages during sampling. The first stage is Synthetic Minority Oversampling Technique (SMOTE), which is to oversample the non-majority classes. Synthetic data is generated through linear interpolation of a randomly selected non-minority sample and one of its k nearest neighbors. The second stage of SMOTE-Tomek Links involves randomly choosing data from the majority class and removing its nearest neighbor if it is from a different class.

With k = 11 (around half the number of observations that the minority class contains), our model outperformed models with k = 23 or k = 5. We believe this intermediate value performs well as it is able to capture more variance whilst reducing the risk of interpolating a data point from the wrong class.

Hence by implementing oversampling with a carefully selected k, we were able to address the imbalanced dataset issue.

3. Experiments

From all of the available deep learning networks, we identified two families that can process sequential data: Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN).

3.1 RNN - Bidirectional Long Term Short Memory (Bi LSTM)

In RNN, each sequence is processed step by step. However, with long sequential input data, RNN suffers from vanishing gradients. Various RNN variants have been proposed to address this problem including LSTM. Instead of unidirectional LSTM, we adopted bidirectional LSTM as it takes the previous (t-1) and the next (t+1) frame into consideration for output at time t [2]. In our work, we have added a single bidirectional LSTM layer in a regular neural network. This model scored an accuracy of 0.29102. We set this as a baseline accuracy level to improve our model.

3.2 CNN - One Dimensional Convolutional Neural Network (1D CNN)

While RNN feeds the results back to the network step by step, CNN feeds the data forward by using convolutional and pooling layers. The input size for 1D CNN is the same as the input size for LSTM, hence we would be able to learn the differences between their performance. Note that since the kernel only slides one dimensional (along time), hence the spatial information between joints is not yet taken into account for this model. 2 stacked 1D convolutional layers of 64 filters and a kernel size of 3 were used to extract features followed by max pooling layer of size 2. Model accuracy was 0.30405.

4. Final proposed model - 2D CNN bidirectional LSTM network

To fully leverage the power of CNN, our final proposed framework uses 2D CNN where kernels move in both directions to capture the spatial information i.e. the relationship between different skeleton joints. We have added additional bidirectional LSTM layers at the end to capture high level sequential information.

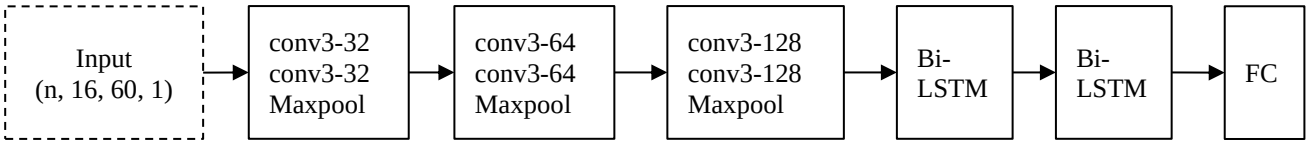

Figure 1: Network Architecture of proposed model

Figure 1 depicts the network architecture chosen which consists of 6 convolutional layers each with 3 x 3 kernels. It is followed by Exponential Linear Unit (ELU) as the activation function and then the Batch Normalization layer. With a kernel size of 2 x 2, a max-pooling layer is added to reduce the spatial size of the representation and to increase the size of the receptive field. We implement these stacked layers 3 times with 32, 64 and 128 kernels respectively. We then reshape the 3D output from the last convolutional layer to 2D to input it into Bi-LSTM. A fully connected layer is added to classify the actions. Stochastic Gradient Descent optimizer is used for cost optimization with a learning rate of 0.001. Categorical cross entropy is used as the loss function.

The advantages of this model include:

- A more powerful network with reduced computational cost: Stacking two 3 x 3 convolutional layers reduces the number of parameters compared to using a single 5 x 5 convolutional layer [3]. Also, more rectification layers lead to a more powerful network.
- Using ELU: ELU was used instead of ReLU as it smooths slowly and can produce negative outputs.
- Integration of Bi-LSTM layers: Increasing the number of layers can increase the network performance [2]. Hence, we stack two Bi-LSTM layers to increase its power of capturing high level sequential information.
- Avoids overfitting: Early stopping and l2 penalty on the weights were used to prevent overfitting. The Train-Validation split was 70% and 30%. We also tried to avoid stacking too many convolutional layers which may result in overfitting the data.

We noticed that although the SGD optimizer outperformed other optimizers we tried (e.g., RMS), it converged much slower. We used GPU provided by Google Collab to address this computational issue.

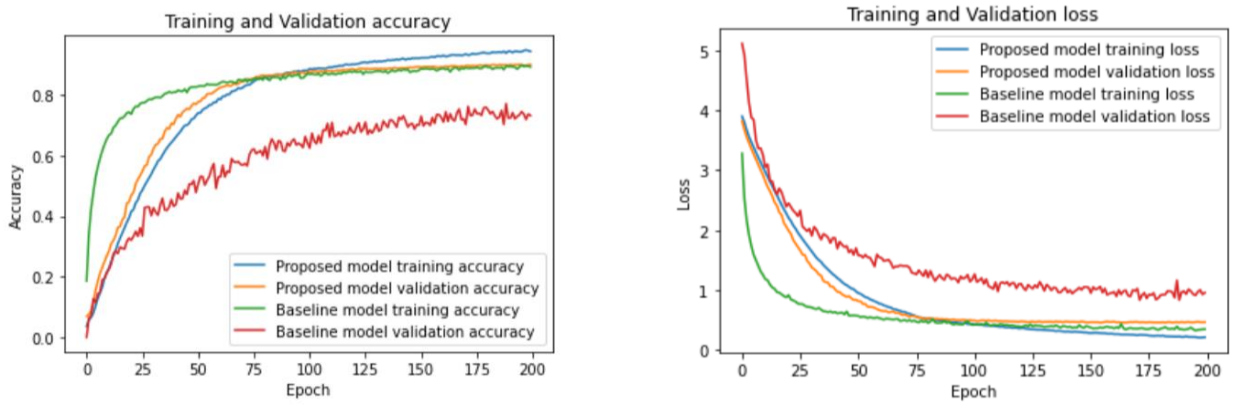

Figure 2: Training and Validation accuracy and loss trend for baseline model and proposed model

The baseline Bi-LSTM model seems to perform poorly. There is a large gap between the training and validation accuracy, and training and validation loss, for the baseline model as shown in Figure 2. This finding is consistent with research showing that Bi-LSTM is good for dealing with temporal signals but not spatially related data [2].

In the case of our proposed model, this gap is much less, as we applied both CNN along with Bi LSTM to handle the data. Table 2 summarizes the accuracy of the models that we experimented on and that of our proposed model.

Table 2: Model accuracy on Kaggle private data

Bi LSTM	1D CNN	2D CNN - BiLSTM (final model)
0.29102	0.30405	0.47007

5. Problems encountered and what we learned

As well as learning how to classify imbalanced and sequential data and how to avoid overfitting (as discussed above), we have also discovered that the control of randomness is important. Initially we tried to control randomness by using seed during train / test data split and oversampling stages, so that our results are reproducible. However, subsequently we learnt that there are other parts of the workflow that contribute randomness such as shuffled batches, weight initialization and dropout layers. Being able to control randomness can help us determine whether changes in performance are due to change of input parameters or are merely due to randomness. An effective machine learning system also needs some randomness e.g., when initializing weights. We believe that by managing randomness in a more controlled way, the overall performance of the model pipeline could be further improved.

References

[1] Y. Cui, M. Jia, T. Lin, Y. Song, and S. Belongie, "Class-Balanced Loss Based on Effective Number of Samples," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 15-20 June 2019 2019, pp. 9260-9269, doi: 10.1109/CVPR.2019.00949.

[2] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, "Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features," *IEEE Access*, vol. 6, pp. 1155-1166, 2018, doi: 10.1109/ACCESS.2017.2778011.

[3] V. Silva, F. Soares, C. P. Leão, J. S. Esteves, and G. Vercelli, "Skeleton Driven Action Recognition Using an Image-Based Spatial-Temporal Representation and Convolution Neural Network," *Sensors*, vol. 21, no. 13, p. 4342, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/13/4342.