Automatic detection of sleep micro-events with convolutional recurrent neural networks

Thomas Mason
Student Number: 170279349
Dr Huy Phan
MSc. Artificial Intelligence
Queen Mary University of London

***Abstract*—Detecting micro-events in sleep studies currently remains a manual and time-consuming process carried out by sleep professionals. In this paper, the convolutional recurrent neural network architecture is used to automatically detect these micro-events, and localise them at temporal positions in the original input for visualization. The network uses a multi-label and multi-task infrastructure, that supports simultaneous event occurrence and uses joint modelling of the event state classification and event boundary estimation to penalize the network. A confidence score is also used to quantify the confidence of each micro-event occurring at each temporal position. The final network model was able to successfully detect and distinguish between the micro-events and achieved F1-scores of 0.541, 0.447 and 0.625 for the Arousal, Limb movement and Sleep disordered breathing micro-events respectively.**

***Index Terms*—Automatic event detection, micro-events, event localisation, convolutional recurrent neural networks**

I. Introduction

The role of sleep is an essential part for living a healthy life, both mentally and physically. The average adult human spends around one third of their entire life sleeping, with the National Sleep Foundation recommending between 7 and 9 hours of sleep per night. Sleep problems, otherwise referred to as sleep disruptions, are linked to both harmful short and long term effects caused by multiple factors such as lifestyle, environmental conditions, sleep disorders and medical conditions. Sleep problems remain a widespread health concern with millions of people impacted by chronic sleep disorders that cause a reduction in their quality of life (Medic et al., 2017).

Automatic event detection is an area of research that continues to expand, due to its valuable potential in a vast number of areas particularly in the medical field explored in this paper. Currently, sleep studies called polysomnographies are manually analysed by sleep professionals and scored into sleep stages, as well as identifying the micro-events that occur throughout. This procedure of analysing the sleep data can take hours for just a single patient, thus automation has the goal of being a key part in speeding up this process. The sleep professionals look for three key micro-events: arousal, limb movement and sleep disordered breathing. Evidence has shown that micro-arousal events during sleep have been associated with increased levels of lipids, cortisol and blood pressure (Ekstedt et al., 2004).

However, the use of automated sleep analysis techniques remains limited in the professional setting even though AI systems have shown to outperform humans. This is due to concerns around the explainability of AI in healthcare, which is a problem that affects the use of AI for the entire medical field (Amann et al., 2020). Two main obstacles have been argued to require solutions in order for an automated system to be used by sleep professionals, as described in Phan et al. (2021). These are: Interpretability and Quantification. Interpretability is the capacity for the model to justify how it made a decision given a particular input, to mitigate the problems caused by ambiguity. Quantification is a metric that measures how confident the model is in its prediction, where in a practical case a low confidence score would require further investigation from a sleep professional. Furthermore, a confidence score will be used in this paper so that the confidence of the model can be quantified to support further progress towards AI in healthcare.

The fundamental parts of the project will consist of exploring, implementing and employing the Convolutional Recurrent Neural Network model. This type of neural network is a Machine Learning architecture that makes use of the Convolutional Neural Network and the Recurrent Neural Network. The role of the convolutional component is to extract relevant features from the polysomnogram signals to find patterns across space by applying different filters. These extracted features are then analysed by the recurrent component, where information flows forward and backwards through the features to find patterns across time. The Convolutional Recurrent Neural Network has been employed in similar research areas and has shown promising results in audio event detection (Phan et al., 2019) as well as sleep scoring (Biswal et al., 2018). The project aims to implement and train a working model that successfully learns to detect arousal, limb movement and sleep disorder breathing micro-events, as well as localise the events for visualization. The model will then be able to take new polysomnographies and obtain a confidence score for each of the micro-events across all temporal positions.

II. Background

A. Polysomnography

A Polysomnography (**PSG**) is a comprehensive sleep study conducted on a patient while they are sleeping, typically

carried out in a clinical setting with the main purpose of diagnosing sleep disorders (Clinic, 2020). One of the great benefits of a Polysomnography is that the test is completely non-invasive and painless, meaning there are no associated risks by performing it. A Polysomnogram monitors different electrophysiological signals using a polygraph, which is an instrument that records electrical signals transmitted from the body by attaching sensors (electrodes) to different body parts. The electrophysiological signals include: Electroencephalogram (**EEG**) that records brain activity, Electrooculogram (**EOG**) that records eye movement, Electromyogram (**EMG**) that records muscle activity and Electrocardiogram (**ECG**) that records heart rate. Various additional parameters are also measured such as Airflow (nasal pressure), Respiratory Effort (Respiratory inductance plethysmography around the thorax and abdomen) and Body position (Tripathi, 2008).

B. Micro-events

The micro-events were scored following the American Academy of Sleep Medicine (**AASM**) guidelines (Blackwell et al., 2011). Arousal's (**AR**) were scored when abrupt changes in EEG frequency including alpha, theta and beta occurred for longer than 3 seconds, following at least 10 seconds of stable sleep prior. Limb movements (**LM**) were scored when more than 4 consecutive movements between 0.5 and 5 seconds occurred, where each movement was separated by at least 5 seconds but no more than 90 seconds. Sleep disordered breathing (**SDB**) events were split into apneas (complete blockage of airway) and hypopneas (partial blockage of airway). Apneas were scored when there was a complete or near absence of airflow for more than 10 seconds. Hypopneas were scored when there was a decrease of more than 30% in airflow for more than 10 seconds, combined with a decrease of 4% in oxygen saturation or a decrease of 3% in oxygen saturation with the occurrence of an arousal.

C. Spectrogram

A spectrogram is a type of time-frequency representation that records the strength (loudness) of a signal across time at various frequencies. It is used frequently in signal processing and a wide range of fields as it is very useful for visualisation and analysis.

There are several key components used in computing the spectrogram. The Discrete Fourier Transform (**DFT**) is a method used to decompose a signal into a superposition of sinusoids (elementary signals), represented as a spectral (magnitude and phase) vector in the frequency domain. It is computed using an algorithm called the Fast Fourier Transform (**FFT**). The Short-Time Fourier Transform (**STFT**) is related to the DFT, however it also incorporates a temporal dimension by using a window function such that a fixed sized window slides across the signal at a constant interval (hop length). The spectral vector is then computed for each window by using the FFT to obtain the STFT. The spectrogram can then be computed by squaring the magnitudes of the STFT.

A filtering technique can be applied to the spectrogram for frequency smoothing and dimensionality reduction, by using a linear-frequency triangular filter bank (Phan et al., 2018a). The filter bank contains a pre-defined number of triangular filters, that are equally spread out between the range of the lowest and highest frequency values in the spectrogram. Each filter has a weight of 1 at its center frequency, where the weight decreases linearly towards 0 until it reaches the centers of the adjacent filters. The filtered spectrogram is then obtained by computing the dot product between the filter bank and the spectrogram.

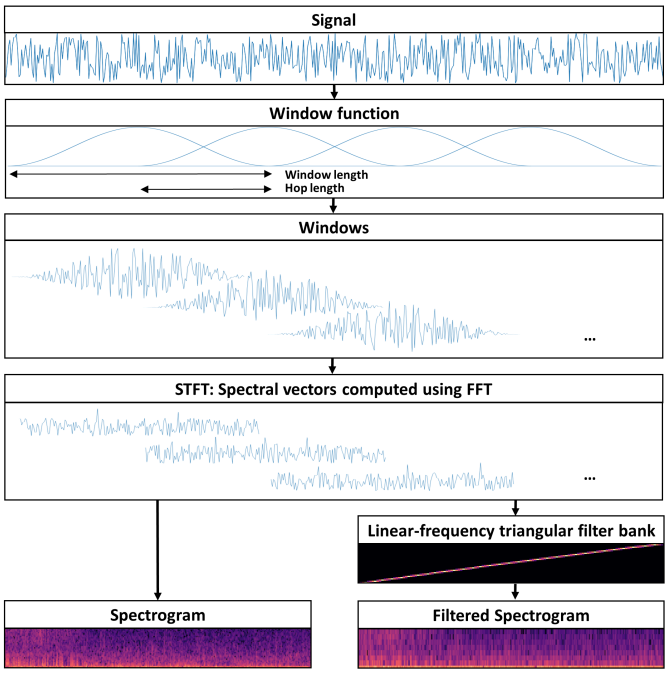

Fig. 1. The process of converting a signal to a spectrogram

Figure 1 is a visual illustration showing the process of converting a signal to either a spectrogram or a filtered spectrogram by applying a filter bank.

An important factor of the spectrogram is the window size. There is a trade-off between time resolution and frequency resolution depending on the window size used. A large window equals good frequency resolution and poor time resolution while a small window equals good time resolution but poor frequency resolution.

D. Convolutional Neural Networks

A Convolutional Neural Network (**CNN**) is composed of key components called convolutional layers. In a convolutional layer, multiple convolution operations are performed by convolving a number of filters (kernels) over the input to form a feature map. In other words, a sliding window approach is used to slide the filters over the input and for each segment multiply them by the filters (weight matrices) to obtain a stack of filtered segments. These filters are what the network uses to extract important information and the feature maps are the result of applying these filters to the input. Multiple

convolutional layers can be stacked on top of each other, where the output of the previous convolutional layer is used as the input for the next. Essentially, this means that the early layers learn to extract low-level features while the later layers learn to extract high-level features (Brownlee, 2020).

This convolutional component is the fundamental part of the CNN, though in practice it includes other components such as pooling, batch normalization and ReLU which optimize performance. Pooling is used to reduce the dimensionality of the feature maps after the convolutional layer which helps mitigate overfitting as well as improving generalization. Batch normalization is a process used to keep the network fast and stable by normalizing the output from the convolutional layer. It is typically used in conjunction with a Rectified Linear Unit (**ReLU**) function. The ReLU function ensures that the feature maps contain no negative values by replacing them with zeros and is a solution to the vanishing gradient problem.

E. Recurrent Neural Networks

A Recurrent Neural Network (**RNN**) consists of a recurrent layer containing a looping mechanism that iterates through all time-steps in an input. The output at each step is computed using the current input as well as the hidden state of the previous step. The hidden state acts as the internal memory of the network as it is a representation of the previous steps. Therefore, the output at each step is influenced by the previous steps, which is useful as it allows the network to make a more informed prediction (Phi, 2018). However, the standard RNN suffers from a short-term memory due to the vanishing gradient problem that occurs during back-propagation, meaning early steps in the input do not learn.

A Gated Recurrent Unit (**GRU**) is a specialised RNN created as a solution to this problem. It has the ability to regulate the flow of information using internal mechanisms called gates. The update gate is responsible for deciding what information is important to keep or discard. The reset gate decides how much of the past information should be forgotten (Kostadinov, 2017). This allows the GRU to retain and discard information throughout the time-steps. The GRU can also be bidirectional, allowing the input to be traversed both forwards and backwards. The forward and backward GRU are independent with no interaction made between them. The outputs are then concatenated so that each time step contains a forward and backward output.

F. Convolutional Recurrent Neural Networks

The Convolutional Recurrent Neural Network (**CRNN**) is a network model consisting of the CNN followed by the RNN/GRU. The convolutional block is the first part of the architecture, which is composed of multiple pairs of convolutional and pooling layers. Using a spectrogram as input, the convolutional block obtains a set of feature maps containing important information about the spectrogram in the frequency domain. The convolutional feature maps are then concatenated and passed as input to the recurrent layer. The recurrent layer then extracts temporal features from the convolutional feature maps to obtain long contextual information (Sun et al., 2021). The resulting recurrent features are then passed through a fully connected layer to reach the output layer. This structure of a convolutional block followed by a recurrent layer has been shown to work exceptionally well.

There are many types of variations of the CRNN, due to the fact that there are many hyper-parameters that can be adjusted to suit the specific problem, such as the kernel and filter size as well as the number of pairs of convolutional and pooling layers. Some variants also include additional layers, such as a residual connection so that both recurrent and convolutional features are connected to the fully connected layer.

G. Related work

Research has been conducted using types of CRNNs in a vast amount of areas. In particular, similar areas include: sleep micro-event detection, audio event detection and sleep scoring.

Olesen et al. (2021) used a CRNN with an attention mechanism to detect AR, LM and SDB micro-events in a PSG. Initially, the channels in the PSG were split into segments containing T samples. A set of event windows were then generated for each segment, where the true events were localized to the event windows. The model used a split stream architecture using 3 input streams; AR, LM and SDB. Each stream took the segments from C pre-defined channels as input. Each stream was passed through multiple convolutional layers, where the feature maps from each stream were then concatenated and passed as input to the bidirectional GRU. The recurrent feature vectors for the forward and backward direction were computed and an additive attention mechanism was used. The final output of the model was the event class probabilities for each event window, as well as the onset and duration times. The model was able to successfully detect AR, LM and SDB events and showed a positive correlation of agreement between the model and human detection.

Phan et al. (2019) used a CRNN model in the area of audio event detection as a multi-label multi-task approach. An audio signal was converted into a log Mel-filtered spectrogram representation, where audio segments of T frames were used as input to the network. Each frame was associated with a set of triplets (one for each event) containing the active state, onset, and offset distances. The goal of the network was to map the audio segments to sequences containing sets of triplets. A multi-loss function is used including binary cross-entropy, distance and confidence loss to penalize the network for event state prediction and event boundary estimation. The results achieved outperformed previous state-of-the-art results. This project will implement a similar architecture to the one used in this paper.

The CRNN architecture has also been employed for sleep scoring in Biswal et al. (2018), by classifying sleep epochs into sleep stages. It has matched the performance of sleep experts for sleep staging, showing that deep learning algorithms can obtain 'human-level' performance.

III. Methodology

A. Dataset

The data was collected from the MrOS Sleep Study, which was an ancillary study part of the Osteoporotic Fractures in Men Study performed on men aged 65 and older (Zhang et al., 2018). It includes PSG recordings of 2909 participants that underwent an overnight in-home polysomnography. The micro-events from the PSG recordings were scored by trained sleep technicians, following the guidelines as specified in section II-B. Data cleaning was then performed to remove incomplete and corrupted files such that $I = 2806$ PSG recordings were used in total.

B. Preprocessing

1) PSGs: A total of $C = 11$ channels were extracted from the PSG recordings, as used in Olesen et al. (2021). The channels included electrophysiological signals; the left and right central EEG (C3, C4), left and right EOG (LOC, ROC), left and right chin EMG, left and right leg EMG as well as parameters; Cannula Flow (Nasal pressure), Thoracic and Abdominal (movements detected using chest belt). The chin channel was combined by subtracting the right chin from the left chin signal. The PSG signals used different sampling rates, therefore all of the signals were resampled to a common sampling frequency of $f_s = 128$ Hz.

A STFT was then computed for each channel, using a hamming window with a window (frame) size of 1 second and a hop size of 0.5 seconds, equating to 50% overlap. It was then filtered using a linear-frequency triangular filter bank with $F = 40$ filters, by computing the dot product between the STFT and the filter bank. The magnitude values were then mapped to the decibel scale (dB) for logarithmic transformation to obtain the spectrogram.

The filtered spectrograms of all signals for each PSG recording were also standardized using:

$$z^{(i)} = \frac{s^{(i)} - \mu^{(i)}}{\sigma^{(i)}} \tag{1}$$

where $s^{(i)} \in \mathbb{R}^{T \times F \times C}$ is the PSG matrix with T frames, F filters and C channels. $\mu^{(i)}$, $\sigma^{(i)} \in \mathbb{R}^{C}$ correspond to the mean and standard deviation of the *i*-th PSG recording. The benefit of standardizing was to ensure that the PSGs were on the same scale, by reducing the coefficients so that the mean and standard deviation were 0 and 1 respectively, which helps in improving training speed.

A frame size of T frames was used to split each standardized PSG matrix $z^{(i)}$ into input sequences $X^{(i)} \in \mathbb{R}^{N \times T \times F \times C}$, where N is the number of sequences, T is the number of frames per sequence, F is the number of filters and C is the number of channels for each *i*-th PSG.

2) Annotations: The groundtruth values were extracted from the annotations, which included the type, onset and duration times, for the $E = 3$ event categories. The duration time $d^{(i)} \in \mathbb{N}$ in samples for each *i*-th PSG was also extracted. A groundtruth active matrix $g^{(i)} \in \{0, 1\}^{E \times d^{(i)}}$ was computed, which contained the active state of every sample for each event class, where E is the event classes and $d^{(i)}$ is the number of samples in the *i*-th PSG.

Similarily to Phan et al. (2019), a set of E triplets $\varepsilon = \{(y^e, p^e, q^e)\}, 1 \leq e \leq E$ was used to annotate each frame, as multiple events could occur simultaneously. $y^e \in \{0, 1\}$ denotes the event state where an event of class e is active '1' or non-active '0' for the current frame. $p^e, q^e \in \mathbb{R}_+$ denote the distances from the current frame to the onset and offset for each event, where both are equal to 0 when an event is non-active. p^e, q^e are also normalized between [0,1], by dividing each event by the corresponding maximum value of the event class in the training data.

To localize the events to each frame, a sliding window approach was used on the groundtruth active matrix g, with a window size of 1 second and hop size of 0.5 seconds, equivalent to 128 and 64 samples respectively. At each frame, the intersection-over-union (IoU) was computed, where frames with an $IoU \geq 0.5$ were declared active. The triplets were computed and stored as a flattened vector with size $3E$.

In the same way as the input sequences X, the frames were split into output sequences $Y^{(i)} \in \mathbb{R}^{N \times T \times 3E}$, where N is the number of sequences, T is the number of frames per sequence and $3E$ is the flattened triplet vectors for each *i*-th PSG recording. The triplets were stored as a flattened vector of size $3E$. Each sequence was associated with a sequence of T triplet sets $\phi = (\varepsilon_1, \varepsilon_2, ..., \varepsilon_T) \in \mathbb{R}^{T \times 3E}$ where $\varepsilon_t = \{(y_t^e, p_t^e, q_t^e)\} \in \mathbb{R}^{3E}, 1 \leq e \leq E$ represents the set of triplets at frame t, $1 \leq t \leq T$.

C. The proposed model architecture

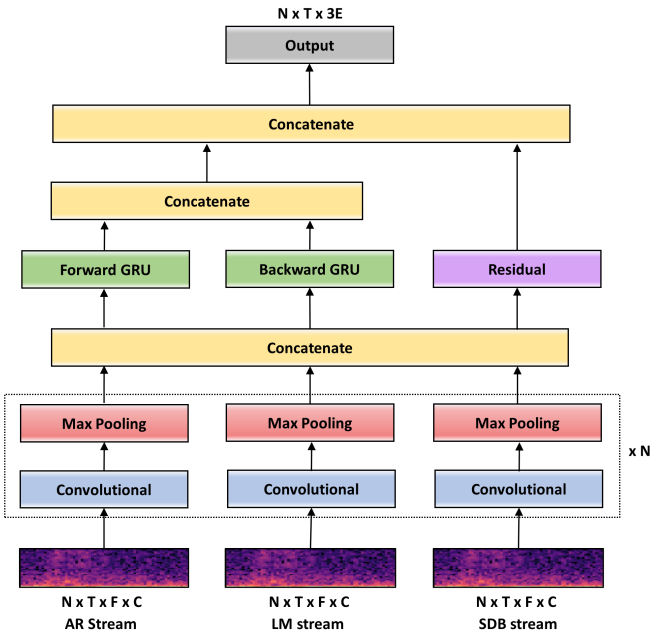

Fig. 2. The proposed model architecture

Figure 2 shows the model architecture of the CRNN, which is a modified version of the network used in Phan et al. (2019).

The goal of the network is to map $\bar{x} \rightarrow \phi$, where $\bar{x} \in \mathbb{R}^{T \times F \times C}$ represents a sample (sequence) in the input sequences X and $\phi \in \mathbb{R}^{T \times 3E}$ represents the corresponding T triplet sets in the output sequences Y, thus being a multi-label and multi-task problem. The network model consists of multiple components which will be explained in the following sub-sections.

1) Input layer: The model uses 3 input streams, which are the AR, LM and SDB. Each input stream consists of a number of channels taken from each sample $\bar{x}$ with shape $T \times F \times C$. The AR stream uses $C = 5$ channels; the EEGs (C3, C4), the EOGs (LOC, ROC) and the EMG (L Chin - R Chin). The LM stream has $C = 2$ channels; the EMGs (Left leg, Right leg). The SDB stream has $C = 3$ channels; Cannula Flow, Thoracic and Abdominal.

2) Convolutional block: The convolutional block is composed of multiple components, where each component consists of a convolutional layer, batch normalization, ReLU activation function, followed by a pooling layer. A two-dimensional kernel convolves $\hat{F}$ filters over the input to obtain the feature maps. Batch normalization is then applied to the feature maps which are then passed through a ReLU activation function. Finally, pooling is applied to reduce the dimensionality of the frequency filters F while maintaining temporal frame T size. The subsequent components then use the feature maps from the previous components as input for higher level feature extraction. Each input stream is fed into its own convolutional block, where the outputs are then concatenated ($\oplus$) to combine all streams into one feature vector:

$$K = (k_{\text{AR}} \oplus k_{\text{LM}} \oplus k_{\text{SDB}}) \in \mathbb{R}^{T \times 3\hat{F}} \tag{2}$$

where $k_{\text{AR}}, k_{\text{LM}}, k_{\text{SDB}} \in \mathbb{R}^{T \times \hat{F}}$ are the feature maps of the AR, LM and SDB streams respectively, after passing through the convolutional block. The concatenated outputs K are represented as a sequence of convolutional feature vectors $K \equiv (k_1, k_2, ..., k_T)$, where $k_t \in \mathbb{R}^{3\hat{F}}, 1 \leq t \leq T$.

3) Recurrent layer: The recurrent layer uses a bidirectional gated recurrent unit, which allows the model to use both past and future information at each timestep. The forward and backward recurrent layer take the convolutional feature vectors K as input. The recurrent feature vectors $H \equiv (h_1, h_2, ..., h_T)$ are then computed for the forward and backward layers:

$$h_t^f = \mathcal{H}(h_{t-1}^f, k_t) \tag{3}$$

$$h_t^b = \mathcal{H}(h_{t+1}^b, k_t) \tag{4}$$

$$h_t = \boldsymbol{\rho}\big((h_t^f \oplus h_t^b) \cdot W_h + b_h\big) \tag{5}$$

where $h_t^f, h_t^b \in \mathbb{R}^H$ are the hidden state vectors of size H (units) for the forward and backward recurrent layer at time-step t. They are computed using the hidden layer function $\mathcal{H}$ which uses Gated Recurrent Units (GRU). The function $\mathcal{H}$ takes the current input k_t and the previous hidden state h_{t-1}, h_{t+1} as input for the forward and backward GRU respectively. $h_t \in \mathbb{R}^{2H}$ represents a recurrent feature vector at time-step t, where the hidden state vectors are concatenated $(h_t^f \oplus h_t^b) \in \mathbb{R}^{2H}$ and multiplied by weight matrix $W_h \in \mathbb{R}^{2H \times 2H}$ with an added bias $b_h \in \mathbb{R}^{2H}$, then subsequently passed through a ReLU activation function ρ. Thus, the recurrent feature vectors H capture the time dependencies from the sequence of convolutional feature vectors K.

4) Residual layer: A fully connected residual layer is also used that bypasses the recurrent layer using a skip connection. It takes the convolutional feature vectors K as input. The residual feature vectors $U \equiv (u_1, u_2, ..., u_T)$ are computed as:

$$u_t = \boldsymbol{\rho}(k_t \cdot W_k + b_k) \tag{6}$$

where the convolutional feature vectors k_t are multiplied with a weight matrix $W_k \in \mathbb{R}^{3\hat{F} \times 2H}$ plus an added bias $b_k \in \mathbb{R}^{2H}$ and passed through a ReLU activation function ρ, such that $u_t \in \mathbb{R}^{2H}$. The residual feature vectors U and recurrent feature vectors H are then concatenated to form the combined feature vectors $V \equiv (v_1, v_2, ..., v_T)$:

$$v_t = u_t \oplus h_t \tag{7}$$

where $v_t \in \mathbb{R}^{4H}$ is the concatenated residual feature vector u_t and recurrent feature vector h_t at time-step t. This gives the network the opportunity to learn from both the convolutional and recurrent features.

5) Output layer: The combined feature vectors V are then passed to the fully connected output layer to obtain the output sequence $O \equiv (o_1, o_2, ..., o_T)$:

$$o_t = \boldsymbol{\sigma}(v_t \cdot W_v + b_v) \tag{8}$$

where the output vector $o_t \in \mathbb{R}^{3E}$ represents a flattened triplet vector $\hat{\epsilon}_t = (\hat{y}_t^{\text{AR}}, \hat{p}_t^{\text{AR}}, \hat{q}_t^{\text{AR}}, \hat{y}_t^{\text{LM}}, \hat{p}_t^{\text{LM}}, \hat{q}_t^{\text{LM}}, \hat{y}_t^{\text{SDB}}, \hat{p}_t^{\text{SDB}}, \hat{q}_t^{\text{SDB}})$ prediction at time-step t. It is computed by multiplying the combined feature vectors v_t with a weight matrix $W_v \in \mathbb{R}^{4H \times 3E}$ plus an added bias $b_v \in \mathbb{R}^{3E}$ and passed through a sigmoid σ activation function. The final output O corresponds to the predicted sequence of T triplet sets $\hat{\phi} = (\hat{\epsilon}_1, \hat{\epsilon}_2, ..., \hat{\epsilon}_T) \in \mathbb{R}^{T \times 3E}$.

D. Loss function

The network uses a multi-loss function, which is similar to the loss used in Phan et al. (2018b). The goal of the loss function is to penalize event state classification and event boundary estimation. The loss function takes the groundtruth T triplet sets $\phi = (\epsilon_1, \epsilon_2, ..., \epsilon_T)$ and the predicted T triplet sets $\hat{\phi} = (\hat{\epsilon}_1, \hat{\epsilon}_2, ..., \hat{\epsilon}_T)$ for each sample $\bar{x}$. The event state classification uses a binary cross-entropy loss to penalize miss-classifications errors for each of the E event classes at all T frames, supporting simultaneous events at each frame. The classification loss is computed as:

$$J_{\text{class}}(\phi, \hat{\phi}) = \frac{1}{T} \sum_{t=1}^{T} \sum_{e=1}^{E} \Big(-y_t^e \cdot \log(\hat{y}_t^e) - (1 - y_t^e) \cdot \log(1 - \hat{y}_t^e) \Big) \tag{9}$$

To penalise the event boundary estimation errors, the difference in the onsets and the offsets was calculated. The distance loss was computed using the sum of the squared

euclidean norm (L^2 norm) of the absolute distances between the groundtruths and predictions for both onsets and offsets:

$$J_{\text{dist}}(\phi, \hat{\phi}) = \frac{1}{T} \sum_{t=1}^{T} \sum_{e=1}^{E} \Big(\| \, p_t^e - \hat{p}_t^e \, \|_2^2 + \| \, q_t^e - \hat{q}_t^e \, \|_2^2 \Big) \quad (10)$$

Lastly, further penalization for both misclassification and event boundary estimation errors is applied by comparing the groundtruth and predicted boundaries. This provides an equilibrium in the losses so that one isn't dominated by the other. Therefore, the confidence loss is computed by subtracting the intersection-over-union (IoU) from the active state and then computing the squared euclidean norm:

$$\begin{aligned} J_{\text{conf}}(\phi, \hat{\phi}) &= \frac{1}{T} \sum_{t=1}^{T} \sum_{e=1}^{E} \Big(\| \, y_t^e - \text{IoU}(\phi, \hat{\phi}) \, \|_2^2 \Big) \\ \text{IoU}(\phi, \hat{\phi}) &= \frac{\min\left(p_t^e, \hat{p}_t^e\right) + \min\left(q_t^e, \hat{q}_t^e\right)}{\max\left(p_t^e, \hat{p}_t^e\right) + \max\left(q_t^e, \hat{q}_t^e\right)} \end{aligned} \quad (11)$$

The overall loss function is comprised of a weighted combination of these losses:

$$J = \sum_{n=1}^{N} \Big(J_{\text{class}}(\phi_n, \hat{\phi}_n) + J_{\text{dist}}(\phi_n, \hat{\phi}_n) + J_{\text{conf}}(\phi_n, \hat{\phi}_n) \Big) \quad (12)$$

IV. Inference

The inference scheme uses joint modelling of the event state classification and event boundary estimation, to incorporate multi-label and multi-task inference likewise as Phan et al. (2019).

The predicted output $\hat{Y} \in \mathbb{R}^{N \times T \times 3E}$ obtained from using the trained model on a test PSG input $\hat{X} \in \mathbb{R}^{N \times T \times F \times C}$ (excluding the PSG indices (i) for simplicity), contains N sequences where an input sequence $\hat{X}(n)$ has output sequence $\hat{Y}(n)$ corresponding to the predicted T triplet sets $\hat{\phi}$ at sequence n. Using time indices $n, n^* \in \mathbb{Z}_0^+$, where n denotes the current sequence $0 \leq n \leq N$ and n^* denotes the temporal position of the current frame $0 \leq n^* \leq N \cdot T$ for all frames in sequences N. The confidence score at temporal position n^* for all event-classes E is computed as:

$$\begin{aligned} f_e\big(n^* | \hat{X}(n)\big) = \sum_{t=1}^{T} \hat{y}_t^e(n) \cdot \mathbb{1}\big(\hat{y}_t^e(n) > \alpha_e\big) \\ \cdot \mathbb{1}\big(n \in \text{RoI}(\hat{p}_t^e(n), \hat{q}_t^e(n))\big) \end{aligned} \quad (13)$$

where the event state $y_t^e(n)$ at frame t in sequence n for event e contributes to the confidence score at temporal position n^* for event e if it meets two conditions. (1) The first condition ensures that the current event state is greater than the event-class threshold α_e. (2) The second condition ensures that the temporal position n^* is within the region of interest (RoI), such that:

$$n \cdot T + t - \hat{p}_t^e(n) \leq n^* \leq n \cdot T + t + \hat{q}_t^e(n) \quad (14)$$

where the onset and offset estimations are denormalized back to their original values (by multiplying the corresponding event-class with the maximum values in the training data). Thus, the confidence score at temporal position n^* is a sum of all applicable event states in the current sequence $\hat{Y}(n)$, controlled using the two indicator functions $\mathbb{1}$. Therefore, the confidence score at each temporal position across all sequences is:

$$f_e\big(n^*\big) = \sum_{n=1}^{N} f_e\big(n^* | \hat{X}(n)\big) \quad (15)$$

such that the predicted output $\hat{Y}$ is now represented as a temporal sequence of segments f_e, where each segment corresponds to a temporal position in the PSG with a confidence score for each event-class e. Therefore, the model's confidence of each event-class occurring is quantified at each segment. A confidence threshold β_e is then applied to the segments in the temporal sequences for each event-class:

$$f_e\big(n^*\big) \begin{cases} 1, & \text{if } n^* > \beta_e \\ 0, & \text{otherwise} \end{cases} \quad (16)$$

where a segment with a confidence score greater than the confidence threshold β_e is assigned an active event state, or otherwise a non-active event state.

V. Experiments

The $I = 2806$ PSG recordings were randomly shuffled and partitioned into training, validation and testing sets, where $I_{\text{train}} = 1606, I_{\text{val}} = 200, I_{\text{test}} = 1000$ are the number of PSG recordings in each set respectively.

A. Network configuration

A hyper-parameter grid search was used to find the optimal hyper-parameters for the model, using a similar procedure used in Cakır et al. (2017). The grid search included: frame size T {16,32,64,128}, kernel size {(3,3),(5,5),(11,11),(1,5),(5,1)} and filter size $\hat{F}$ / hidden units H {96,256} for the convolutional feature maps and recurrent/residual layers respectively. The grid search was iterated through all combinations to find the optimial combination that achieved the greatest F1-score on the validation set. Table I shows the optimal configuration settings for the model after using grid search.

TABLE I
Optimal parameter configuration

	Frame size	**Kernel size**	**Filter size**
Optimal value	64	(5,5)	256

The network was set to use 3 pairs of convolutional and pooling layers in the convolutional block, where the pooling configuration is set to (5,4,2) in the spectral dimension so that the frequency filters $F = 40$ are reduced to a single feature: $40 \rightarrow 8 \rightarrow 2 \rightarrow 1$. The convolutional layers were set to use 'SAME' padding so that temporal size was preserved during convolution. To combat over-fitting and improve generalization, regularization was applied using dropout with a rate of 0.25 to the convolutional, recurrent and residual layers. Figure A.1 shows the final model architecture using the described network configuration.

B. Training

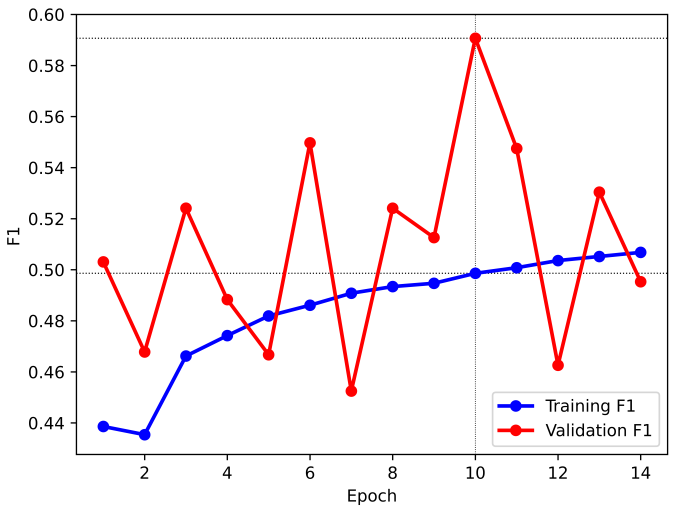

Fig. 3. Training results

The network model was trained for 15 epochs, using an early stopping callback with a patience of 4 to obtain the weights of the best epoch that maximized the validation F1. The weights at epoch 10 were saved with a training F1 of 0.499 and validation F1 of 0.591 shown in Figure 3. The network was also trained using the 'Adam' optimization algorithm, with a learning rate of 10^{-4}. Due to the significant amount of data, it was not possible to fit it all into memory. Thus, a data generator was used to load each PSG into memory consecutively, where a batch size of 128 was used meaning 128 sequences were processed at each step.

C. Data sampling

A data sampling technique was used to balance both event classes as well as active and non-active events. The data classes were highly imbalanced, with 8.7% AR, 7.7% LM and 83.6% SDB, and the active event state was very under-represented compared to the non-active event state with a ratio of 1:5.88. Therefore, each sample $\bar{x}$ was weighted so that the overall weighting for each event class was $\frac{1}{k}$ and the weighting for the active/non-active event states was equally balanced with a 1:1 ratio relative to the entire training set. Applying the sampling weights saw a improvement in performance, especially for the AR and LM event classes.

D. Evaluation metrics

To evaluate the model's performance at detecting micro-events at the frame level, two evaluation metrics are used; segment-based F1-score (F1) and segment-based error rate (ER). The metrics are applied to a temporal sequence of segments f_e computed for a test PSG using the inference scheme described in section IV. At each segment in a sequence, the groundtruth and predicted value comparison is regarded as one of four outcomes: **True positive** (TP), where a active event state is correctly predicted as active. **True negative** (TN), where a non-active event state is correctly predicted as non-active. **False positive** (FP), where a non-active event state is incorrectly predicted as active. **False negative** (FN), where a active event state is incorrectly predicted as non-active.

The total values for all outcomes are then calculated by summing over all segments. The F1 is computed using the recall and precision metrics:

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN} \tag{17}$$

where precision is the percentage of active predictions that were correct, and recall is the percentage of active groundtruths that were predicted correctly. Using both the precision and recall metric, the harmonic mean equating to the F1 is computed as:

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{18}$$

The error rate proposed in Mesaros et al. (2016) is computed using several components, which include the substitution (S), insertion (I) and deletion (D) errors as well as the number of active groundtruth events (A) for each segment t:

$$\begin{aligned} S(t) &= \min\big(FP(t), FN(t)\big) \\ D(t) &= \max\big(0, FN(t) - FP(t)\big) \\ I(t) &= \max\big(0, FP(t) - FN(t)\big) \\ A(t) &= TP(t) \end{aligned} \tag{19}$$

where at each segment t; substitutions are the number of groundtruth events with an incorrect prediction, where one substitution is equivalent to one FP and one FN. The remaining outcomes lie within the deletions and insertions, where deletions are the number of active groundtruth events incorrectly predicted, and insertions are the number of incorrect active predictions. The active events are simply the number of events with an active state. Therefore, the ER for all segments is computed as:

$$ER = \frac{\sum_{t=1}^{N \cdot T} S(t) + \sum_{t=1}^{N \cdot T} D(t) + \sum_{t=1}^{N \cdot T} I(t)}{\sum_{t=1}^{N \cdot T} A(t)} \tag{20}$$

The overall F1 and ER metrics were then obtained by averaging the results of the F1 and ER for all of the PSGs. The F1 and ER metrics are explained in greater detail in Mesaros et al. (2016) with visual illustrations.

E. Testing and visualizing confidence

During the testing phase, cross-validation was used on the validation set to find the optimal α_e and β_e parameter values for each event that maximized the F1. The confidence scores were normalized between [0,1], where values between [0,1] with increments of 0.01 were explored for both α and β. The optimal values for α; 0.28, 0.24, 0.37 and for β; 0.03, 0, 0.14 were obtained for AR, LM and SDB events respectively.

The model was then tested on new unseen PSGs in the test set to produce predicted outputs that were then used in inference to compute temporal sequences of segments. Each segment contained a confidence score for each event across

all temporal positions. The β thresholds were then applied on the confidence scores to assign each segment an event state to obtain the model predicted event states.

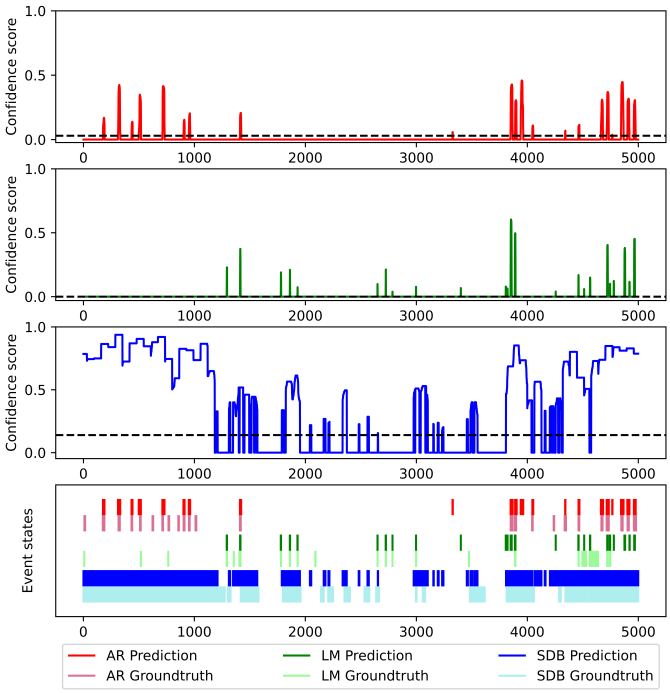

Fig. 4. The model's predicted and the groundtruth event states along with the confidence scores obtained during inference for the events; AR, LM and SDB from part of a single PSG in the test set. The dashed line in the confidence plots represents the β_e threshold for the corresponding event. The legend below the plots signifies the predicted and groundtruth labels.

The confidence scores for the AR, LM and SDB events can be visualized with a comparison between the model's prediction and groundtruth event states, as shown in Figure 4 using part of a PSG in the test set. It demonstrates that the model is able to successfully detect simultaneous events and distinguish between them very well.

F. Results

To measure the performance, the model was evaluated using the recall, precision, F1 and ER metrics as described in section V-D. Table II contains the results of the averaged evaluation metrics for each event and the overall performance,

TABLE II
EVALUATION METRICS ACHIEVED ON THE TEST SET USING THE FINAL MODEL

Event	Precision	Recall	F1	ER
AR	0.545±0.110	0.593±0.184	0.541±0.113	0.985±0.491
LM	0.382±0.130	0.589±0.142	0.447±0.125	1.913±5.742
SDB	0.512±0.147	0.859±0.132	0.625±0.134	1.131±0.715
Overall	0.506 ±0.129	0.815±0.119	0.612±0.117	1.066±0.526

The evaluation metrics are computed as an average across all of the $I_{\text{test}} = 1000$ PSGs in the test set. The results are recorded as mean±standard deviation.

and Figures A.2, A.3, A.4 and A.5 correspond to the boxplots for each evaluation metric.

Analysing the results, the SDB event obtained the highest F1 of 0.625 which is substantially more than the AR and LM events with 0.541 and 0.447 respectively. This correlates with the event sizes, as SDB events were the largest events followed by AR then LM events, indicating that the model performs much better on longer events. The AR event obtained the lowest ER of 0.985 in comparison to 1.131 and 1.913 for the SDB and LM events respectively. The ER for the LM event is particularly high with a very large standard deviation illustrated in the boxplot, which is due to a large amount of insertion errors. This is most likely because of its low β threshold value of 0, thus leading to a large amount of false positives. This also coincides with why the precision of the LM event is significantly lower compared to the other events.

Comparing the results of the final model with the model used in Olesen et al. (2021), the final model achieves a slightly better F1 for the SDB event but is outperformed for AR and LM events by a significant margin. The mentioned model is more complex than the final model, including extra components such as an attention mechanism that gives the network more learning power which would explain the better performance. That being said, the network has achieved reasonable results and has shown to succeed in the requirements set out, and with possible future additions as mentioned in section VII, there is high potential for even better performance.

VI. Conclusion

The main purpose of this paper was to explore the CRNN architecture in sleep micro-event detection and employ a model that is able to detect these micro-events in the sleep signals. The proposed network model, which uses a multi-label and multi-task infrastructure has shown it is able to successfully detect and distinguish between multiple events occurring simultaneously, by using joint modelling of the event state classification and event boundary estimation. The inference scheme used converts the model prediction into a sequence of segments, where each segment contains a confidence score quantifying the confidence of each micro-event occurring at each temporal position. The network was evaluated on the test set and obtained obtained F1-scores of 0.541, 0.447 and 0.625, for the AR, LM and SDB micro-events respectively.

VII. Future Work

In future work, several new additions could be explored. The network could be further expanded to include an attention mechanism which would allow the network to gain greater insight on important connections within the network. Another important future focus is interpretability, which is the ability for the network to justify how it made a decision as an extended solution to combat ambiguity. This would help in the progress of adopting AI algorithms in the medical field. Some more work could also be done surrounding experimentation, to explore more hyper-parameters to optimize the network further.

Acknowledgements

A special thanks goes out to Dr Huy Phan for the time he spent supervising me. He guided me throughout the duration of this project, providing his expertise and knowledge in this area, which was very helpful and much appreciated.

References

Amann, J., Blasimme, A., Vayena, E., Frey, D., Madai, V.I., 2020. Explainability for artificial intelligence in healthcare: a multidisciplinary perspective. 10.1186/s12911-020-01332-6 https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-01332-6.

Biswal, S., Sun, H., Goparaju, B., Westover, M.B., Sun, J., Bianchi, M.T., 2018. Expert-level sleep scoring with deep neural networks. 10.1093/jamia/ocy131 https://academic.oup.com/jamia/article/25/12/1643/5185596.

Blackwell, T., Ancoli-Israel, S., Redline, S., Stone, K.L., 2011. Factors that may influence the classification of sleep-wake by wrist actigraphy: The mros sleep study. 10.5664/JCSM.1190 https://jcsm.aasm.org/doi/pdf/10.5664/JCSM.1190.

Brownlee, J., 2020. How do convolutional layers work in deep learning neural networks? https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

Cakır, E., Parascandolo, G., Heittola, T., Huttunen, H., Virtanen, T., 2017. Convolutional recurrent neural networks for polyphonic sound event detection. 10.1109/TASLP.2017.2690575 https://arxiv.org/abs/1702.06286.

Clinic, M., 2020. Polysomnography (sleep study) https://www.mayoclinic.org/tests-procedures/polysomnography/about/pac-20394877.

Ekstedt, M., Akerstedt, T., Söderström, M., 2004. Microarousals during sleep are associated with increased levels of lipids, cortisol, and blood pressure. 10.1097/01.psy.0000145821.25453.f7 https://pubmed.ncbi.nlm.nih.gov/15564359/.

Kostadinov, S., 2017. Understanding gru networks https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

Medic, G., Wille, M., Hemels, M.E., 2017. Short- and long-term health consequences of sleep disruption. 10.2147/NSS.S134864 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5449130/.

Mesaros, A., Heittola, T., Virtanen, T., 2016. Metrics for polyphonic sound event detection. 10.3390/app6060162 https://www.mdpi.com/2076-3417/6/6/162.

Olesen, A.N., Jennum, P., Mignot, E., Sorensen, H.B.D., 2021. Msed: a multi-modal sleep event detection model for clinical sleep analysis. arXiv:2101.02530v1 https://arxiv.org/pdf/2101.02530.pdf.

Phan, H., Andreotti, F., Cooray, N., Chèn, O.Y., Vos, M.D., 2018a. Dnn filter bank improves 1-max pooling cnn for single-channel eeg automatic sleep stage classification. 10.1109/EMBC.2018.8512286 https://kar.kent.ac.uk/72661/1/Phan2018c.pdf.

Phan, H., Chen, O.Y., Koch, P., Pham, L., McLoughlin, I., Mertins, A., Vos, M.D., 2019. Unifying isolated and overlapping audio event detection with multi-label multi-task convolutional recurrent neural networks. 10.1109/ICASSP.2019.8683064 https://ieeexplore.ieee.org/document/8683064.

Phan, H., Krawczyk-Becker, M., Gerkmann, T., Mertins, A., 2018b. Weighted and multi-task loss for rare audio event detection. 10.1109/ICASSP.2018.8461353 https://ieeexplore.ieee.org/document/8461353.

Phan, H., Mikkelsen, K., Chen, O.Y., Koch, P., Mertins, A., Vos, M.D., 2021. Sleeptransformer: Automatic sleep staging with interpretability and uncertainty quantification. arXiv:2105.11043 https://arxiv.org/pdf/2105.11043.pdf.

Phi, M., 2018. Illustrated guide to lstm's and gru's: A step by step explanation https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

Sun, C., Zhang, M., Wu, R., Lu, J., Xian, G., Yu, Q., Gong, X., Luo, R., 2021. A convolutional recurrent neural network with attention framework for speech separation in monaural recordings. 10.1038/s41598-020-80713-3 https://www.nature.com/articles/s41598-020-80713-3.pdf.

Tripathi, M., 2008. Technical notes for digital polysomnography recording in sleep medicine practice. 10.4103/0972-2327.41887 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2771960/.

Zhang, G.Q., Cui, L., Mueller, R., Tao, S., Kim, M., Rueschman, M., Mariani, S., Mobley, D., Redline, S., 2018. The national sleep research resource: towards a sleep data commons. 10.1093/jamia/ocy064 https://pubmed.ncbi.nlm.nih.gov/29860441/.

Appendix A
Figures

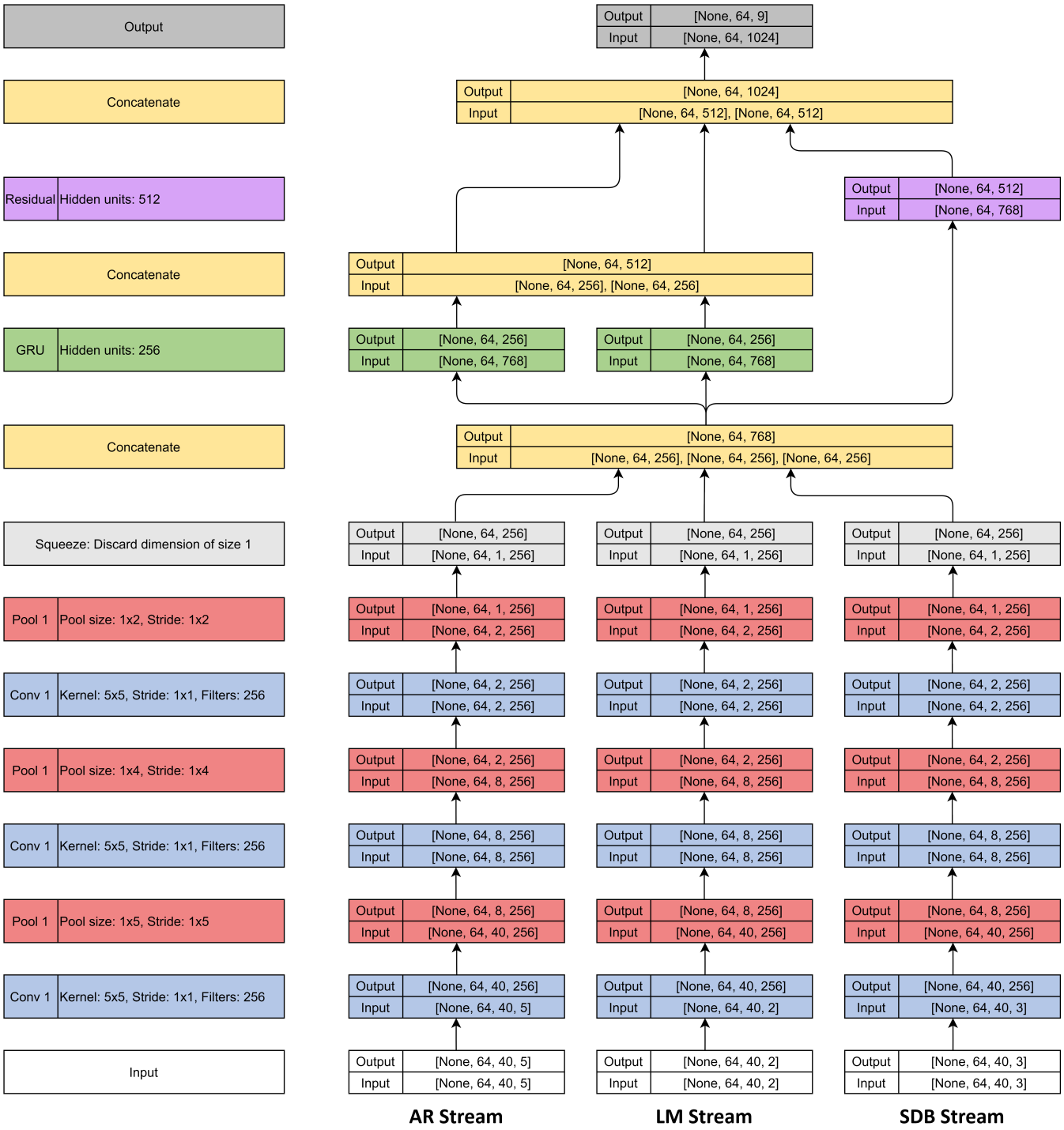

Fig. A.1. The final model architecture, using the proposed model in Figure 2 Section III. It includes the input and output shapes, where the first index 'None' represents and supports any batch size. Network layers and important parameters are listed in the boxes to the left side, which are colour coded to indicate the corresponding layers in the network.

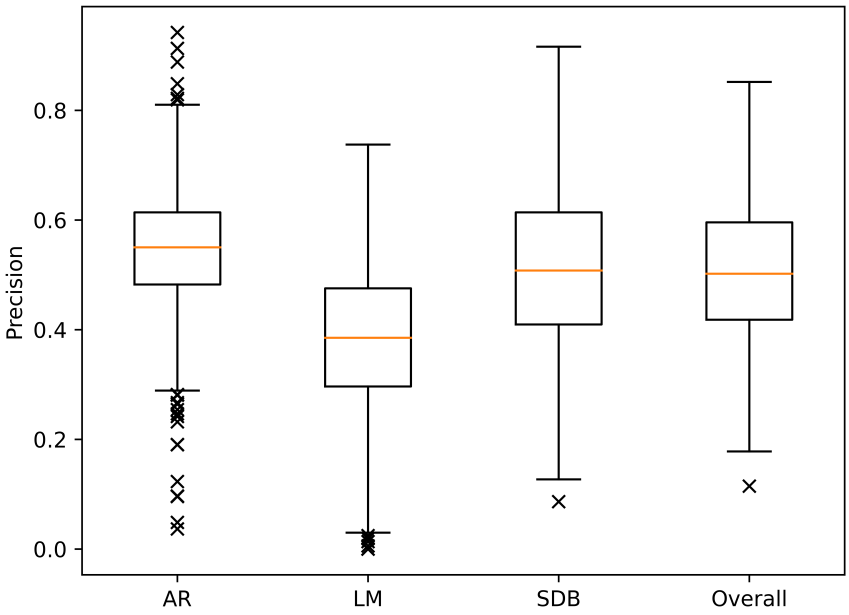

Fig. A.2. The boxplots of the precision for all micro-events and overall performance on the test set

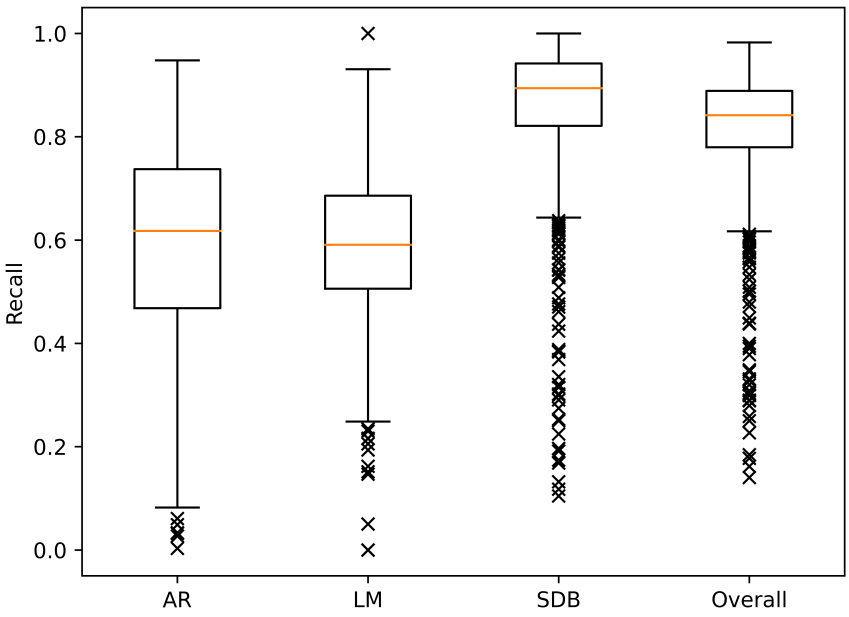

Fig. A.3. The boxplots of the recall for all micro-events and overall performance on the test set

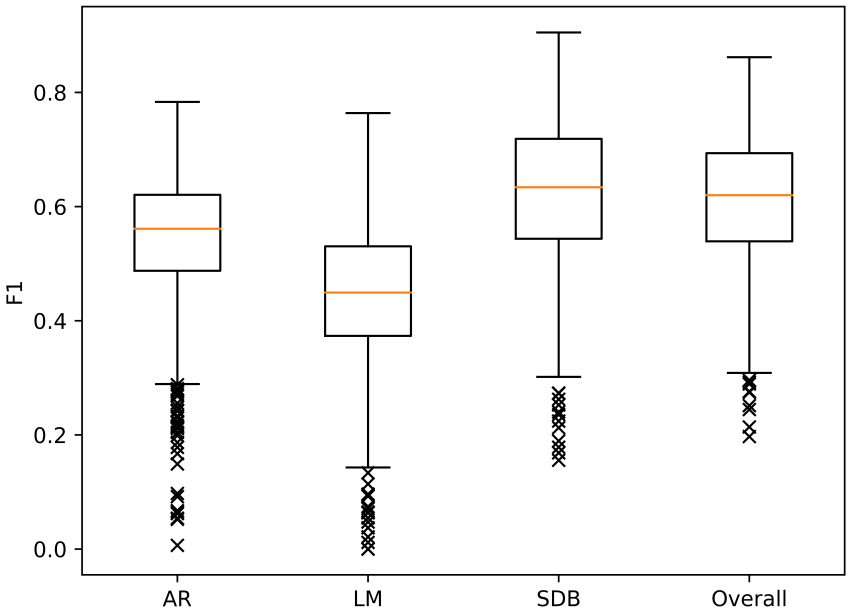

Fig. A.4. The boxplots of the F1-score (F1) for all micro-events and overall performance on the test set

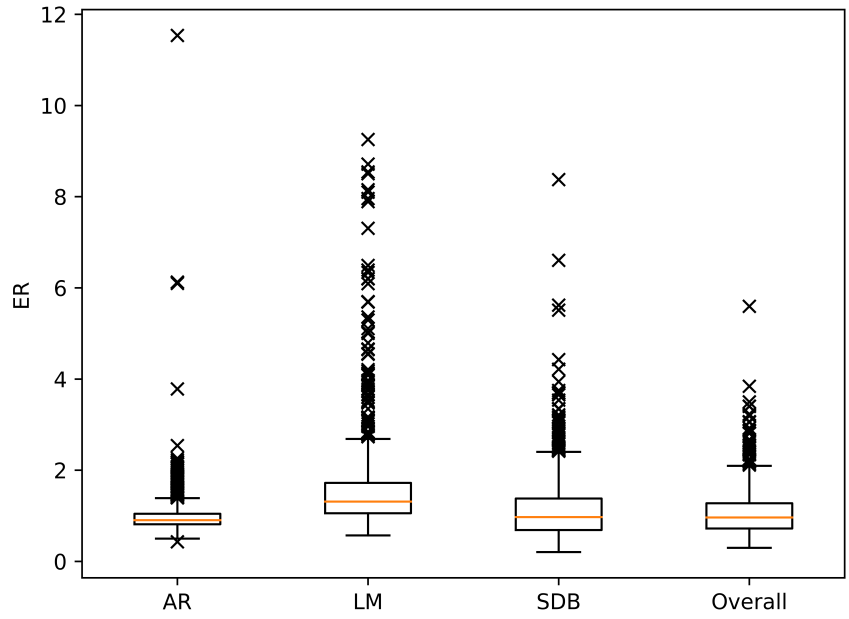

Fig. A.5. The boxplots of the error rate (ER) for all micro-events and overall performance on the test set