DeepSleepNet: A Model for Automatic Sleep Stage Scoring Based on Raw Single-Channel EEG

Akara Supratak, Hao Dong, Chao Wu, and Yike Guo

***Abstract*— This paper proposes a deep learning model, named DeepSleepNet, for automatic sleep stage scoring based on raw single-channel EEG. Most of the existing methods rely on hand-engineered features, which require prior knowledge of sleep analysis. Only a few of them encode the temporal information, such as transition rules, which is important for identifying the next sleep stages, into the extracted features. In the proposed model, we utilize convolutional neural networks to extract time-invariant features, and bidirectional-long short-term memory to learn transition rules among sleep stages automatically from EEG epochs. We implement a two-step training algorithm to train our model efficiently. We evaluated our model using different single-channel EEGs (F4-EOG (left), Fpz-Cz, and Pz-Oz) from two public sleep data sets, that have different properties (e.g., sampling rate) and scoring standards (AASM and R&K). The results showed that our model achieved similar overall accuracy and macro F1-score (MASS: 86.2%–81.7, Sleep-EDF: 82.0%–76.9) compared with the state-of-the-art methods (MASS: 85.9%–80.5, Sleep-EDF: 78.9%–73.7) on both data sets. This demonstrated that, without changing the model architecture and the training algorithm, our model could automatically learn features for sleep stage scoring from different raw single-channel EEGs from different data sets without utilizing any hand-engineered features.**

***Index Terms*— Sleep stage scoring, deep learning, single-channel EEG.**

I. Introduction

SLEEP plays an important role in human health. Being able to monitor how well people sleep has a significant impact on medical research and practice [1].

Typically, sleep experts determine the quality of sleep using electrical activity recorded from sensors attached to different parts of the body. A set of signals from these sensors is called a polysomnogram (PSG), consisting of an electroencephalogram (EEG), an electrooculogram (EOG), an electromyogram (EMG), and an electrocardiogram (ECG). This PSG is segmented into 30-s epochs, which are then be classified into different sleep stages by the experts according to sleep manuals such as the Rechtschaffen and Kales (R&K) [2] and the American Academy of Sleep Medicine (AASM) [3]. This process is called sleep stage scoring or sleep stage classification. This manual approach is, however, labor-intensive and time-consuming due to the need for PSG recordings from several sensors attached to subjects over several nights.

There have been a number of studies trying to develop a method to automate sleep stage scoring based on multiple signals such as EEG, EOG and EMG [4]–[6], or single-channel EEG [7]–[9]. These methods firstly extract time-domain, frequency-domain and time-frequency-domain features from each recording epoch. In the case of multiple signals, the features from all signals in one epoch were concatenated into one feature vector. The features are then used to train classifiers to identify the sleep stage of the epoch. However, we believe that these methods may well not generalize to a larger population due to the heterogeneity among subjects and recording hardware. This is because these features were hand-engineered based on the characteristics of the available dataset.

Recently, deep learning, a branch of machine learning that utilizes multiple layers of linear and non-linear processing units to learn hierarchical representations or features from input data, has been employed in sleep stage scoring. For instance, Längkvist *et al.* [10] have investigated a capability of Deep Belief Nets (DBNs) to learn probabilistic representations from preprocessed raw PSG. Convolutional Neural Networks (CNNs) have also been applied to learn multiple filters that are used to convolve with small portions of input data (i.e., convolution) to extract time-invariant features from raw Fpz-Cz EEG channel [11]. However, the results from the literature showed that applying deep learning on hand-engineered features performed better than on raw signals [7], [10]. This might well be because the authors did not consider temporal information that sleep experts use when they determine the sleep stage of each epoch.

Only a few number of literature have explored Recurrent Neural Networks (RNNs) in sleep stage scoring. RNNs are capable of conditioning outputs on all previous inputs, as they maintain internal memory and utilizes feedback (or loop) connections to learn temporal information from sequences of inputs. The main advantage of RNNs is that they can be trained to learn long-term dependencies such as transition rules [3] that sleep experts use to identify the next possible sleep stages from a sequence of PSG epochs. Elman RNNs have been applied on energy features from the Fpz-Cz EEG channel [12]. In our previous work [13], we also applied Long Short-Term

Manuscript received January 16, 2017; revised June 9, 2017; accepted June 23, 2017. Date of publication June 28, 2017; date of current version November 6, 2017. *(Corresponding author: Yike Guo.)*

The authors are with the Department of Computing, Imperial College London, London SW7 2AZ, U.K. (e-mail: as12212@ic.ac.uk; hao.dong11@ic.ac.uk; chao.wu@ic.ac.uk; y.guo@ic.ac.uk).

Digital Object Identifier 10.1109/TNSRE.2017.2721116

Memory (LSTM) on time-frequency-domain features from the F4-EOG and Fp2-EOG channels separately. Even though the reported results were promising, these methods still rely on hand-engineered features.

This paper introduces *DeepSleepNet*, a model for automatic sleep stage scoring based on raw single-channel EEG, which is different from the existing works that develop algorithms to extract features from EEG. We aim to automate the process of hand-engineering features by utilizing the feature extraction capabilities of deep learning. The main contributions of this work are as follows:

- We develop a new model architecture that utilizes two CNNs with different filter sizes at the first layers and bidirectional-LSTMs. The CNNs can be trained to learn filters to extract time-invariant features from raw single-channel EEG, while the bidirectional-LSTMs can be trained to encode temporal information such as sleep stage transition rules into the model.
- We implement a two-step training algorithm that can effectively train our model end-to-end via backpropagation, while preventing the model from suffering class imbalance problem (i.e., learning to classify only the majority of sleep stages) presented in a large sleep dataset.
- We show that, without changing the model architecture and the training algorithm, our model could automatically learn features for sleep stage scoring from different raw single-channel EEGs from two datasets, that have different properties (e.g., sampling rate) and scoring standards (AASM and R&K), without utilizing any hand-engineered features.

II. DeepSleepNet

The architecture of DeepSleepNet consists of two main parts as shown in Fig. 1. The first part is representation learning, which can be trained to learn filters to extract time-invariant features from each of raw single-channel EEG epochs. The second part is sequence residual learning, which can be trained to encode the temporal information such as stage transition rules [3] from a sequence of EEG epochs in the extracted features. This architecture is designed for scoring 30-s EEG epochs following the standard of AASM and R&K manuals.

A. Representation Learning

We employ two CNNs with small and large filter sizes at the first layers to extract time-invariant features from raw single-channel 30-s EEG epochs. This architecture is inspired by the way signal processing experts control the trade-off between temporal and frequency precision in their feature extraction algorithms [14]. The small filter is better to capture temporal information (i.e., when certain of EEG patterns appear), while the larger filter is better to capture frequency information (i.e., frequency components).

In our model, each CNN consists of four convolutional layers and two max-pooling layers. Each convolutional layer performs three operations sequentially: 1D-convolution with its filters, batch normalization [15], and applying the rectified linear unit (ReLU) activation (i.e., $relu(x) = max(0, x)$). Each pooling layer downsamples inputs using max operation. The specifications of the filter sizes, the number of filters, stride sizes and pooling sizes can be found in Fig. 1. Each *conv* block shows a filter size, the number of filters,

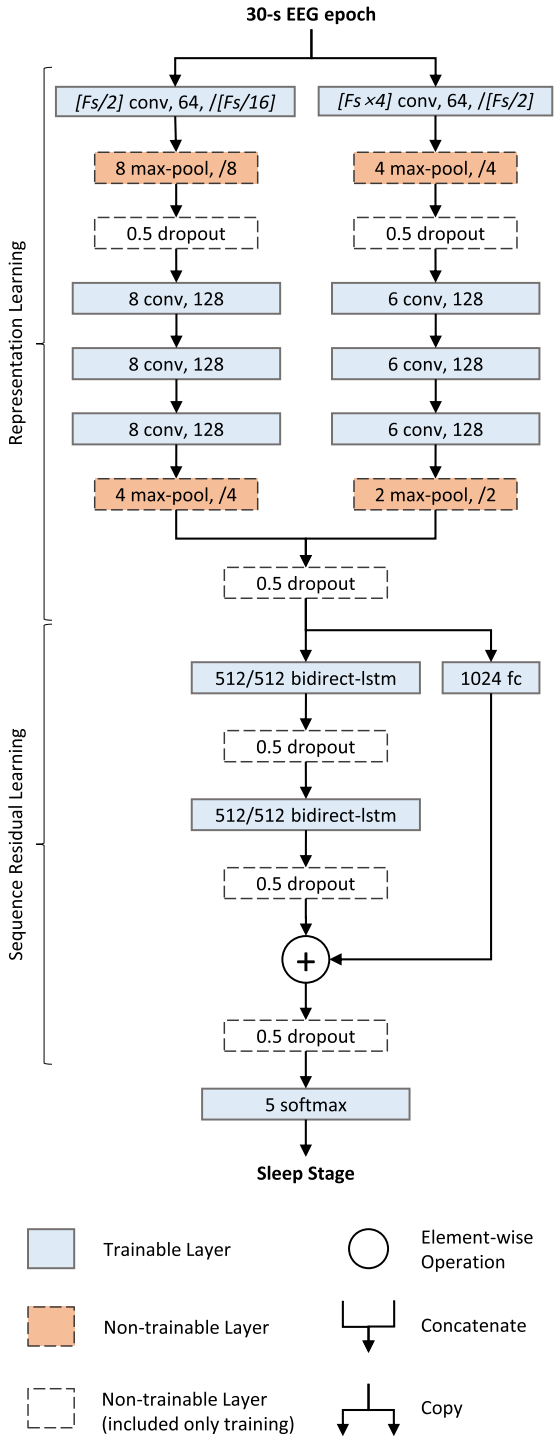

Fig. 1. An overview architecture of DeepSleepNet consisting of two main parts: representation learning and sequence residual learning. Each trainable layer is a layer containing parameters to be optimized during a training process. The specifications of the first convolutional layers of the two CNNs depends on the sampling rate (Fs) of the EEG data (see Section II-C).

and a stride size. Each *max-pool* block shows a pooling size and a stride size. We will explain *dropout* blocks later in Section III-C.

Formally, suppose there are N 30-s EEG epochs $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ from single-channel EEG. We use the two CNNs to extract the i-th feature $\mathbf{a}_i$ from the i-th EEG epoch $\mathbf{x}_i$ as follows:

$$\mathbf{h}_i^s = CNN_{\theta_s}(\mathbf{x}_i) \quad (1)$$

$$\mathbf{h}_i^l = CNN_{\theta_l}(\mathbf{x}_i) \quad (2)$$

$$\mathbf{a}_i = \mathbf{h}_i^s || \mathbf{h}_i^l \quad (3)$$

where $CNN(x_i)$ is a function that transform a 30-s EEG epoch $\mathbf{x}_i$ into a feature vector $\mathbf{h}_i$ using a CNN, θ_s and θ_l are parameters of the CNNs with small and large filter sizes in the first layer respectively, and $||$ is a concatenate operation that combines the outputs from two CNNs together. These concatenated or linked features $\{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$ are then forwarded to the sequence residual learning part.

B. Sequence Residual Learning

We apply the residual learning framework [16] to design our sequence residual learning part. This part consists of two main components: bidirectional-LSTMs [17] and a shortcut connection (see Fig. 1).

We employ two layers of bidirectional-LSTMs to learn temporal information such as stage transition rules [3] which sleep experts use to determine the next possible sleep stages based on the previous stages. For instance, the AASM manual suggests that if a subject is in sleep stage N2, continue to score epochs with low amplitude and mixed frequency EEG activity as N2 even though K complexes or sleep spindles are not present. In this case, the bidirectional-LSTMs can learn to remember that it has seen the stage N2, and continue to score successive epochs as N2 if they still detect the low amplitude and mixed frequency EEG activity. Bidirectional-LSTMs extends the LSTM [18] by having two LSTMs process forward and backward input sequences independently [17]. In other words, the outputs from forward and backward LSTMs are not connected to each other. The model is therefore able to exploit information both from the past and the future. We also use peephole connections [19], [20] in our LSTMs which allow their gating mechanism to inspect their current memory cell before the modification.

We use a shortcut connect to reformulate the computation of this part into a residual function. This enables our model to be able to add temporal information it learns from the previous input sequences into the feature extracted from the CNNs. We also use a fully-connected layer in the shortcut connection to transform the features from the CNNs into a vector that can be added to the output from the LSTMs. This layer performs matrix multiplication with its weight parameters, batch normalization, and applying the ReLU activation sequentially.

Formally, suppose there are N features from the CNNs $\{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$ arranged sequentially and $t = 1 \ldots N$ denotes the time index of 30-s EEG epochs, our sequence residual learning is defined as follows:

$$\mathbf{h}_t^f, \mathbf{c}_t^f = LSTM_{\theta_f}(\mathbf{h}_{t-1}^f, \mathbf{c}_{t-1}^f, \mathbf{a}_t) \quad (4)$$

$$\mathbf{h}_t^b, \mathbf{c}_t^b = LSTM_{\theta_b}(\mathbf{h}_{t+1}^b, \mathbf{c}_{t+1}^b, \mathbf{a}_t) \quad (5)$$

$$\mathbf{o}_t = \mathbf{h}_t^f || \mathbf{h}_t^b + FC_\theta(\mathbf{a}_t) \quad (6)$$

where $LSTM$ represents a function that processes sequences of features $\mathbf{a}_t$ using the two-layers LSTM parameterized by θ_f and θ_b for forward and backward directions; $\mathbf{h}$ and $\mathbf{c}$ are vectors of hidden and cell states of the LSTMs; $\mathbf{h}_0^f, \mathbf{c}_0^f, \mathbf{h}_{N+1}^b$ and $\mathbf{c}_{N+1}^b$ of forward and backward LSTMs are set to zero vectors; FC represents a function that transform features $\mathbf{a}_t$ into a vector that can be added (element-wise) with the concatenated output vector $\mathbf{h}_t^f || \mathbf{h}_t^b$ from the bidirectional-LSTMs. The specifications of the hidden size of forward and backward LSTMs, and the fully-connected layers can be found in Fig. 1. Each *bidirect-lstm* block shows hidden sizes of forward and backward LSTMs. Each *fc* block shows a hidden size.

It should be noted that the hidden and cell states $\mathbf{h}_t^f, \mathbf{h}_t^b, \mathbf{c}_t^f$ and $\mathbf{c}_t^b$ in (4) and (5) will be re-initialized to zeros at the beginning of each patient data during the training and testing. This is to make sure that the model uses only temporal information from the current subject data for both training and testing.

C. Model Specification

For the representation learning part, the parameters of the CNN-1 and CNN-2 were selected with the aim to capture temporal and frequency information from the EEG according to the guideline provided by [14]. For instance, in Fig. 1, the filter size of the *conv1* layers of the CNN-1 was set to Fs/2 (i.e., half of the sampling rate (Fs)), and its stride size was set to Fs/16 to detect when certain of EEG patterns appear. On the other hand, the filter size of the *conv1* layer of the CNN-2 was set to Fs×4 to better capture the frequency components from the EEG. Its stride size was also set to Fs/2, which is higher than the *conv1* layer of the CNN-1, as it is not necessary to perform a fine-grained convolution to extract frequency components. The filter and stride sizes of the subsequent convolutional layers *conv2_[1-3]* were chosen to be small fix sizes. It is believed that the use of multiple convolutional layers with a small filter size instead of a single convolutional layer with a large filter can reduce the number of parameters and the computational cost, and can still achieve the similar level of model expressiveness [21].

For the sequence residual learning part, the parameters of the *bidirect-lstm* and *fc* layers were set to be smaller than the output of the representation learning part, which is 1024 in Fig. 1. This is to restrict our model to select and combine only the important features to prevent overfitting.

III. Two-Step Training Algorithm

The two-step training algorithm (see Algorithm 1) is a technique we develop to effectively train our model end-to-end via backpropagation, while preventing the model from suffering class imbalance problem (i.e., learning to classify only the majority of sleep stages) present in a large sleep dataset.

The algorithm first pre-trains the representation learning part of the model and then fine-tunes the whole model using two different learning rates. We use the cross-entropy loss to quantify the agreement between the predicted and the target sleep stages in both of these training steps. The combination of the softmax function (i.e., the last layer in Fig. 1) and the cross-entropy loss are used to train our model to output probabilities for mutually exclusive classes.

A. Pre-Training

The first step is to perform a supervised pre-training on the representation learning part of the model with a class-balance training set so that the model does not overfit to the majority of sleep stages. This can be seen in Algorithm 1, lines 1-8. Specifically, the two CNNs are extracted from the model and then stacked with a softmax layer, $softmax$. It is important to note that this $softmax$ is different from the last layer in the model (see Fig. 1). This stacked softmax layer is only used in this step to pre-train the two CNNs, in which its parameters are discarded at the end of the pre-training. We denote these two CNNs stacked with $softmax$ as pre_model. Then the pre_model is trained with a class-balance training set using a mini-batch gradient-based optimizer called Adam [22] with a learning rate, lr. At the end of the pre-training, the softmax layer is discarded. The class-balance training set is obtained from duplicating the minority sleep stages in the original training set such that all sleep stage have the same number of samples (i.e., oversampling).

B. Fine-Tuning

The second step is to perform a supervised fine-tuning on the whole model with a sequential training set. This can be seen in Algorithm 1, lines 9-19. This step is to encode the stage transition rules into the model as well as to perform necessary adjustments on the pre-trained CNNs. Specifically, the parameters θ_s and θ_l of the two CNNs of $init\_model$ are replaced with the ones from the pre_model, resulting in $model$. Then the $model$ is trained with the sequence training set using a mini-batch Adam optimizer with two different learning rates, lr_1 and lr_2. As the CNNs part has already been pre-trained, we, therefore, use a lower learning rate lr_1 for the CNNs part and a higher learning rate lr_2 for the sequence residual learning part, and a softmax layer. We found that when we used the same learning rate to fine-tune the whole network, the pre-trained CNN parameters were excessively adjusted to the sequential data, which were not class-balanced. As a consequence, the model started to overfit to the majority of the sleep stages toward the end of the fine-tuning. Therefore, two different learning rates are used during fine-tuning. Also, we use a heuristic gradient clipping technique to prevent the exploding gradients, which is a well-known problem when training RNNs such as LSTMs [23]. This technique rescales the gradients to smaller values using their global norm whenever they exceed a pre-defined threshold. The sequential training set is obtained by arranging the original training set sequentially according to time across all subjects.

Algorithm 1 Two-step Training

Input: $init\_model$, $data$
Output: $model$
Initialization:
1: $init\_CNN_{\theta_s,\theta_l} \leftarrow extract\_cnns(init\_model)$
2: $pre\_model \leftarrow stack(init\_CNN_{\theta_s,\theta_l}, softmax)$
3: $data_{over} \leftarrow oversample(data)$
Pre-training Step:
4: **for** $i = 1$ **to** $n\_pretrain\_epochs$ **do**
5: **for each** $batch$ **in** $shuffle(data_{over})$ **do**
6: $pre\_model \leftarrow adam_{lr}(pre\_model, batch)$
7: **end for**
8: **end for**
Fine-tuning Step:
9: $pre\_CNN_{\theta_s,\theta_l} \leftarrow extract\_cnns(pre\_model)$
10: $model \leftarrow replace\_cnns(init\_model, pre\_CNN_{\theta_s,\theta_l})$
11: **for** $i = 1$ **to** $n\_finetune\_epochs$ **do**
12: **for each** $subject$ **in** $data$ **do**
13: $model \leftarrow reset\_lstm\_cell\_state(model)$
14: $subject\_data_{seq} \leftarrow arrange\_sequence(subject)$
15: **for each** $batch$ **in** $subject\_data_{seq}$ **do**
16: $model \leftarrow adam_{lr_1,lr_2}(model, batch)$
17: **end for**
18: **end for**
19: **end for**
20: **return** $model$

C. Regularization

We employed two regularization techniques to help prevent overfitting problems. The first technique was dropout [24], [25] that randomly sets the input values to 0 (i.e., dropping units along with their connection) with the specified probability during training. Dropout layers with the probability of 0.5 were used throughout the model as shown in Fig. 1. It is important to note that these dropout layers were used for training only, and were removed from the model during testing to provide deterministic outputs.

The second technique was L2 weight decay, which adds a penalty term into a loss function to prevent large values of the parameters in the model (i.e., exploding gradients). We only applied the weight decay on the first layers of the two CNNs because of the two main reasons. Firstly, it is pointed out in [23] that L2 weight decay can limit the model capabilities of learning long-term dependencies. Secondly, we found that, without weight decay, the filters of the first layers of the CNNs overfitted to noises or artifacts in EEG data. This weight decay helped the model learn smoother filters (i.e., containing less high-frequency elements) which resulted in slightly performance gains. The weight decay parameter that defines the degree of penalty, lambda, was set to 10^{-3}.

IV. Results

A. Data

We evaluated our model using different EEG channels from two public datasets: Montreal Archive of Sleep Studies (MASS) [26] and Sleep-EDF [27], [28].

TABLE I
NUMBER OF 30-s EPOCHS FOR EACH SLEEP STAGE FROM TWO DATASETS

Dataset	W	N1	N2	N3 (N4)	REM	Total
MASS	6227	4724	29534	7651	10464	58600
Sleep-EDF	7927	2804	17799	5703	7717	41950

1) Mass: In MASS cohort 1, there were five subsets of recordings, SS1-SS5, which were organized according to their research and acquisition protocols. We used data from SS3, which contained PSG recordings from 62 healthy subjects (age 42.5±18.9). Each recording contained 20 scalp-EEG, 2 EOG (left and right), 3 EMG and 1 ECG channels. The EEG electrodes were positioned according to the international 10-20 system, and the EOG electrodes were positioned diagonally on the outer edges of the eyes. EEG and EOG recordings were pre-processed with a notch filter of 60 Hz, and band-pass filters of 0.30-100 Hz (EEG) and 0.10-100 Hz (EOG). All EEG and EOG recordings had the same sampling rate of 256 Hz. These recordings were manually classified into one of the five sleep stages (W, N1, N2, N3 and REM) by a sleep expert according to the AASM standard [3]. There were also movement artifacts at the beginning and the end of each subject's recordings that were labeled as UNKNOWN. We evaluated our model using the F4-EOG (Left) channel, which was obtained via montage reformatting [29] without any further pre-processing.

2) Sleep-EDF: There were two sets of subjects from two studies: age effect in healthy subjects (SC) and Temazepam effects on sleep (ST). We used 20 subjects (age 28.7±2.9) from SC. Each PSG recording contained 2 scalp-EEG signals from Fpz-Cz and Pz-Cz channels, 1 EOG (horizontal), 1 EMG, and 1 oro-nasal respiration signal. All EEG and EOG had the same sampling rate of 100 Hz. These recordings were manually classified into one of the eight classes (W, N1, N2, N3, N4, REM, MOVEMENT, UNKNOWN) by sleep experts according to the R&K standard [2]. We evaluated our model using the Fpz-Cz and Pz-Cz channels without any further pre-processing. We also merged the N3 and N4 stages into a single stage N3 to use the same AASM standard as the MASS dataset. There were long periods of awake or stage W at the start and the end of each recording, in which a subject was not sleeping. We only included 30 minutes of such periods just before and after the sleep periods, as we were interested in sleep periods.

We excluded MOVEMENT and UNKNOWN (which were at the start or the end of the each recording) stages, as they did not belong to the five sleep stages [3]. Table I summarizes the number of 30-s epochs for each sleep stage from these two datasets.

B. Experimental Design

We evaluated our model using a k-fold cross-validation scheme, where k was set to 31 and 20 for the MASS and Sleep-EDF datasets respectively. Specifically, in each fold, we used recordings from $N_s - (N_s/k)$ to train the model, and from the remaining N_s/k subjects to test the trained model, where N_s is the number of subjects in the dataset. This process was repeated k times so that all of the recordings were tested. Then we combined the predicted sleep stages from all folds and computed the performance metrics, which will be discussed in Section IV-C.

C. Performance Metrics

We evaluated the performance of our model using per-class precision (PR), per-class recall (RE), per-class F1-score (F1), macro-averaging F1-score (MF1), overall accuracy (ACC), and Cohen's Kappa coefficient (κ) [30], [31]. The per-class metrics are computed by considering a single class as a positive class, and all other classes combined as a negative class. The MF1 and ACC are calculated as follows:

$$\text{ACC} = \frac{\sum_{c=1}^{C} \text{TP}_c}{N} \tag{7}$$

$$\text{MF1} = \frac{\sum_{c=1}^{C} \text{F1}_c}{C} \tag{8}$$

where TP_c is the true positives of class c, F1_c is per-class F1-score of class c, C is the number of sleep stages, and N is the total number of test epochs.

D. Training Parameters

The representation learning part was pre-trained using the oversampled training set with the mini-batch size of 100. The Adam optimizer's parameters lr, $beta1$, and $beta2$ were set to 10^{-4}, 0.9 and 0.999 respectively. Then the whole model was fine-tuned using the sequential training set. Specifically, we equally split the sequences of 30-s EEG epochs from each subject data into 10 sub-sequences (i.e., batch size was 10). Then we fed 25 epochs (i.e., sequence length was 25) from each sub-sequence yielding 250 epochs per one step training. The Adam optimizer's parameters were similar to the pre-training step except that the learning rate of each part of the model, $lr1$ and $lr2$, were set to 10^{-6} and 10^{-4} respectively. The threshold of the gradient clipping was set to 10. The numbers of epochs for the pre-training and the fine-tuning steps were set to 100 and 200 respectively. There was no early stopping as there was no validation set in our evaluation scheme.

For the batch normalization in *conv* and *fc* blocks, the ϵ constant of 10^{-5} was added to the mini-batch variance for numerical stability. The mean and variance of the training set, which were used as fixed parameters during testing, were estimated by computing the moving average of with a decay rate of 0.999 from the sampling mean and variance of each mini-batch.

E. Implementation

We implemented our model using TensorLayer (https://github.com/zsdonghao/tensorlayer), which is a deep learning library extended from Google Tensorflow [32]. This library allows us to deploy numerical computation such as the training and validation tasks to multiple CPUs and GPUs. We ran the k-fold cross-validation using the eTRIKS

Analytical Environment (eAE) (https://eae.doc.ic.ac.uk/), which provides a cluster of high-performance computing nodes. Each node was equipped with an NVIDIA GeForce GTX 980. The training time for each validation fold was approximately 3 hours on each node. The testing or prediction time for each batch of 25 EEG epochs (according to the sequence length specified during training in Section IV-D) was approximately 50 milliseconds on each node. Our code is publicly available at https://github.com/akaraspt/deepsleepnet.

F. Initial Experiments

We initially conducted experiments with the first fold of the 31-fold cross-validation with the MASS dataset. This was to design the architecture and the parameters for DeepSleepNet. For model architecture, we tried several configurations such as increasing/decreasing convolutional layers, changing the number of filters and the stride sizes, and changing the number of hidden sizes in the bidirectional-LSTMs and the fully-connected layer. The architecture in Fig. 1 gave us the best performance. For regularization parameters, we tried several values for the weight decay parameters ranging from 10^{-1} to 10^{-5}. The value of 10^{-3} gave us the best performance. For training parameters, we tried several values of learning rates ranging from 10^{-3} to 10^{-8}. We also experimented with the mini-batch size (from 50 to 200) during the pre-training, the batch size (from 5 to 40) and sequence length (from 5 to 40) during fine-tuning. Other parameters such as $beta1$, $beta2$ and the threshold of the gradient clipping were chosen from the default values reported in the literature. The training parameters mentioned in Section IV-D gave us the best performance. With these settings, the pre-training and fine-tuning steps started to converge after 100 and 200 epochs respectively.

G. Sleep Stage Scoring Performance

Table II and III show confusion matrices obtained from the 31-fold and the 20-fold cross-validation on the F4-EOG (Left) and the Fpz-Cz channels from the MASS and Sleep-EDF datasets respectively. We did not include the confusion matrix obtained from the Pz-Oz channel from the Sleep-EDF dataset as the Fpz-Cz channel gave a better performance. Each row and column represent the number of 30-s EEG epochs of each sleep stage classified by the sleep expert and our model respectively. The numbers in bold indicate the number of epochs that were correctly classified by our model. The last three columns in each row indicate per-class performance metrics computed from the confusion matrix.

It can be seen that the poorest performance was noted for the stage N1, with the F1 less than 60, while the F1 for other stages were significantly better, with the range between 81.5 and 90.3. Most of the misclassified stages were between N2 and N3. It can also be seen that the confusion matrix is almost symmetric via the diagonal line (except the pair of N2-N3). This indicates that the misclassifications were less likely to be due to the imbalance-class problem.

Fig. 2 demonstrate examples of hypnograms that were manually scored by a sleep expert, and automatically scored by our DeepSleepNet for Subject-1 from the MASS dataset.

TABLE II
CONFUSION MATRIX OBTAINED FROM 31-FOLD CROSS-VALIDATION ON F4-EOG (LEFT) CHANNEL FROM THE MASS DATASET

	Predicted					Per-class Metrics		
	W	N1	N2	N3	REM	PR	RE	F1
W	**5433**	572	107	13	102	87.3	87.2	87.3
N1	452	**2802**	827	4	639	60.4	59.3	59.8
N2	185	906	**26786**	1158	499	89.9	90.7	90.3
N3	18	4	1552	**6077**	0	83.8	79.4	81.5
REM	132	356	533	1	**9442**	88.4	90.2	89.3

TABLE III
CONFUSION MATRIX OBTAINED FROM 20-FOLD CROSS-VALIDATION ON Fpz-Cz CHANNEL FROM THE SLEEP-EDF DATASET

	Predicted					Per-class Metrics		
	W	N1	N2	N3	REM	PR	RE	F1
W	**6614**	745	181	81	306	86.0	83.4	84.7
N1	295	**1406**	631	30	442	43.5	50.1	46.6
N2	391	618	**14542**	1473	775	90.5	81.7	85.9
N3	29	9	291	**5370**	4	77.1	94.2	84.8
REM	360	457	419	7	**6474**	80.9	83.9	82.4

H. Comparison With State-of-the-Art Approaches

Table IV shows a comparison between our method and other sleep stage scoring methods across ACC, MF1, κ and F1. These methods include the ones that utilize hand-engineered features [7]–[9], [12], CNNs only [11] or LSTMs only (which is our previous work) [13]. The other methods' metrics were computed using the confusion matrices reported in their papers. We classified the methods into two groups: *non-independent* and *independent* training and test sets. The non-independent ones were the methods that included parts of the test subjects' epochs in the training data, while the independent ones were the methods that excluded all epochs of the test subjects from the training data. We believe that the practical evaluation scheme should not include any epochs from the test subjects. Also, it has been shown that the non-independent scheme resulted in an improvement of the performance [7]. Thus we did not compare the performance of our method with the non-independent group. The numbers in bold indicate the highest performance metrics of all methods in each dataset of each group.

Among the methods in the independent group, it can be seen that our method achieved a similar performance compared to the state-of-the-art methods that used the same EEG channel and dataset. It should also be emphasized that our model achieved the similar performance without sacrificing the performance on the stage N1, which is the most difficult sleep stage to classify. This indicates that our method was not biased by favoring the majority of the sleep stages than the minority ones. According to the κ coefficient, it showed that the agreement between the sleep experts and our model were substantial (between 0.61 and 0.80) [9]. It should also be noted that our model performed better when applied on the Fpz-Cz channel compared to the Pz-Oz, which is similar to [7].

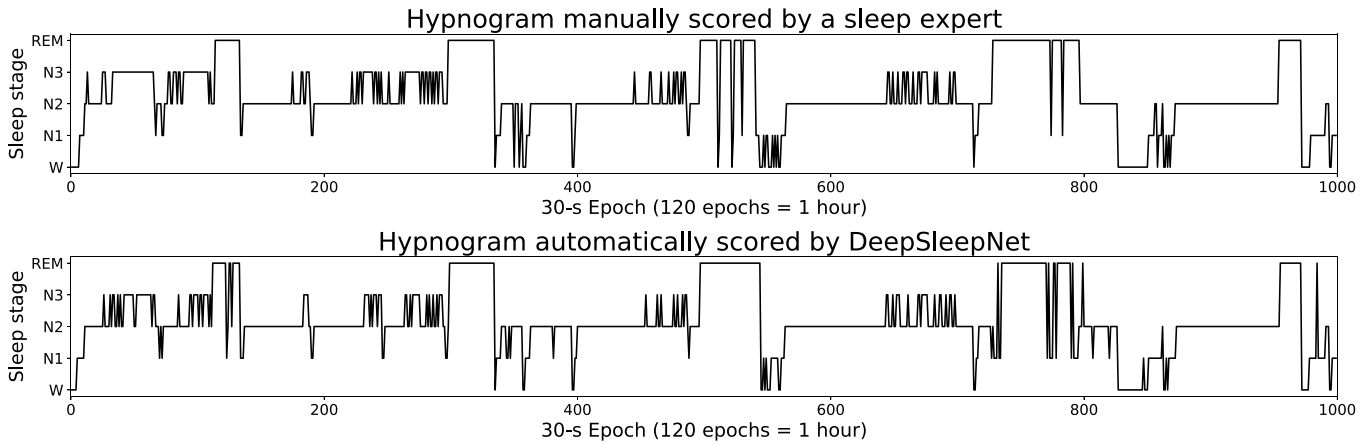

Fig. 2. Examples of the hypnogram manually scored by a sleep expert (top) and the hypnogram automatically scored by DeepSleepNet (bottom) for Subject-1 from the MASS dataset.

TABLE IV
COMPARISON BETWEEN DEEPSLEEPNET AND OTHER SLEEP STAGE SCORING METHODS THAT UTILIZES HAND-ENGINEERING FEATURES ACROSS OVERALL ACCURACY (ACC), MACRO-F1 SCORE (MF1), COHEN'S KAPPA (κ), AND PER-CLASS F1-SCORE (F1)

Methods	Dataset	EEG Channel	Test Epochs	Overall Metrics			Per-class F1-Score (F1)				
				ACC	MF1	κ	W	N1	N2	N3	REM
Non-independent Training and Test Sets											
Ref. [12]	Sleep-EDF	Fpz-Cz	960	90.3	76.5	-	77.3	46.5	**94.9**	72.2	**91.8**
Ref. [8]	Sleep-EDF	Pz-Oz	15136	**91.3**	77	**0.86**	**97.8**	30.4	89	**85.5**	82.5
Ref. [9]	Sleep-EDF	Pz-Oz	7596	90.8	**80**	0.85	96.9	**49.1**	89	84.2	81.2
Independent Training and Test Sets											
Ref. [7]	Sleep-EDF	Fpz-Cz	37022	78.9	73.7	-	71.6	**47.0**	84.6	84.0	81.4
Ref. [11]	Sleep-EDF	Fpz-Cz	37022	74.8	69.8	-	65.4	43.7	80.6	**84.9**	74.5
DeepSleepNet	Sleep-EDF	Fpz-Cz	41950	**82.0**	**76.9**	**0.76**	84.7	46.6	**85.9**	84.8	**82.4**
DeepSleepNet	Sleep-EDF	Pz-Oz	41950	79.8	73.1	0.72	**88.1**	37	82.7	77.3	80.3
Ref. [13]	MASS	F4-EOG (Left)	59066	85.9	80.5	-	84.6	56.3	**90.7**	**84.8**	86.1
DeepSleepNet	MASS	F4-EOG (Left)	58600	**86.2**	**81.7**	**0.80**	**87.3**	**59.8**	90.3	81.5	**89.3**

I. Sequence Residual Learning

We performed additional experiments to verify the important of the sequence residual learning part with the MASS dataset. Table V shows a confusion matrix obtained from 31-fold cross-validation on the F4-EOG (Left) channel using DeepSleepNet without the sequence residual learning part (i.e., using the *pre_model* in Algorithm 1). It can be seen that the F1 of all sleep stages, except the stage N3, were lower than the ones in Table II. This was because of an increase in the misclassifications between the pairs of N1-N2, N2-N3 and N1-REM. This may well be due to the effects of oversampling the training set to have balanced-class samples. As a consequence the model tended to predict more of stages N1 and N3. These results indicated that the process to stack the pre-trained representation learning part with the sequence residual learning part, and then fine-tune the both parts with sequential training set helped improve the classification performance.

TABLE V
CONFUSION MATRIX OBTAINED FROM 31-FOLD CROSS-VALIDATION ON THE F4-EOG (LEFT) CHANNEL FROM THE MASS DATASET USING DEEPSLEEPNET WITHOUT SEQUENCE RESIDUAL LEARNING

	Predicted					Per-class Metrics		
	W	N1	N2	N3	REM	PR	RE	F1
W	**5215**	709	94	19	190	84.5	83.7	84.1
N1	468	**2582**	747	11	916	40.8	54.7	46.8
N2	241	1846	**24140**	2435	872	93.4	81.7	87.2
N3	19	3	472	**7156**	1	74.3	93.5	82.8
REM	227	1181	383	5	**8668**	81.4	82.8	82.1

J. Model Analysis

To better understanding how our model classified a sequence of 30-s EEG epochs, we analyzed and compared: 1) the learned filters at the first convolutional layers of the two CNNs in the representation learning part; and 2) the memory cells inside the bidirectional-LSTMs in the sequence residual learning part. This analysis was carried out with the MASS dataset across 31 cross-validation folds.

Firstly, we analyzed how our model utilized the learned filter at the first convolutional layers of the two CNNs to classify different sleep stages. Specifically, we determined which filters were mostly active for each sleep stage by computing the

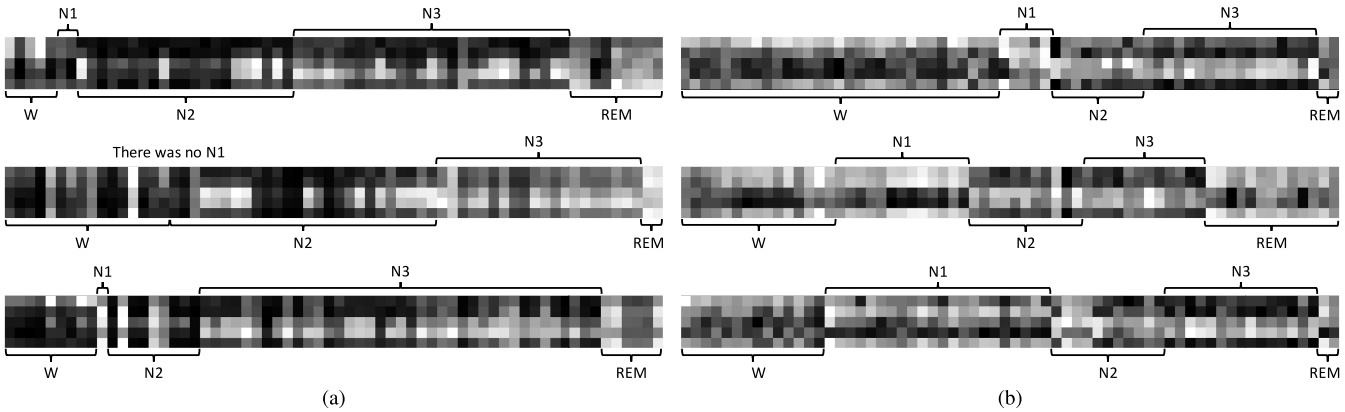

Fig. 3. Examples of the filter activations from the first convolutional layers of the two CNNs obtained by feeding our model with data from 3 subjects. The filter activations from the small filters are on the left (a), and the larger filters are on the right (b). Each image has 5 rows and 64 columns, corresponding to 5 sleep stages and 64 filters respectively. Each pixel represents the scaled value of $u_{c,k}$ from (9), where 1 (i.e., active) is white and 0 (i.e., inactive) is black. Each row corresponds to the 64-dimension vector (i.e., k is 64) for each sleep stage c. The first row is from stage W and the last row is from stage REM. Each image also has labels indicating which filters are mostly active for which sleep stages.

average of the sum of the activations of all filters across samples of each sleep stage. Formally, suppose there were N 30-s EEG epochs from each validation fold $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. We fed these epochs to our model to obtain activations $\mathbf{z}$ from the first convolutional layer of each CNN: $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, where $\mathbf{z}_i \in \mathbb{R}^{p \times q}$, and p and q are the activation output size and the number of filters of the first convolutional layer. The average of the sum of the activations of the filter k for the sleep stage c is computed as follows:

$$u_{c,k} = \frac{\sum_{i=1}^{N_{y_{pred}=c}} \sum_{j=1}^{q} z_{i,j,k}}{N_{y_{pred}=c}} \tag{9}$$

where $u_{c,k}$ is the average of the sum of the activation of the filter k for sleep stage c, $z_{i,j,k}$ is the j-th index of the activation vector $\mathbf{z}_i$ of the filter k, and $N_{y_{pred}=c}$ is the number of EEG epochs that our model predicted as stage c. After we computed the $u_{c,k}$ of all filters for sleep stage c, we rescaled them into a range of 0 and 1. We denote this scaled k-dimension vector $\mathbf{u}_c$ as *filter activations* for stage c. This process was repeated for all sleep stages. Once we got the filter activations from all sleep stages, we stacked them together, and rearranged the order of the filters such that the filters that were mostly active for each sleep stage were grouped together. Fig. 3 illustrates an example of the filter activations from the small (a) and large (b) filters obtained by feeding our model with data from 3 subjects. Each image has 5 rows and 64 columns, corresponding to 5 sleep stages and 64 filters respectively. Each pixel represents the value of $u_{c,k}$ from (9) scaled into a range of 0 and 1, where 1 (i.e., active) is white and 0 (i.e., inactive) is black. Each row corresponds to the 64-dimension vector (i.e., k is 64) for each sleep stage c. The first row is from stage W and the last row is from stage REM. Each image also has labels indicating which filters are mostly active for which sleep stages. We found that there were two types of filters: ones that were mostly active for each sleep stage, and the other ones that were mostly active for multiple sleep stages. For instance, some of the small and large filters were mostly active for both stage N2 and N3. After we had analyzed all of the filter activations from different cross-validation folds, we found that the number of active filters for different sleep stages varied across subjects, and most of the small filters were mostly active for stage N2 and N3. We also found that, for a few subjects, no small filter was active for stage N1. This might well be because there were only a few stage N1 in the dataset.

Secondly, we analyzed how our model utilized the bidirectional-LSTMs to learn the temporal information from a sequence of EEG epochs. Specifically, we investigated how the bidirectional-LSTMs managed their memory cells (i.e., c in (4) and (5)) using the visualization technique from [33]. We found several memory cells of the forward LSTMs that were interpretable. For instance, several cells were keeping track of the wakefulness or the sleep onset, which reset their values to positive numbers (i.e., active) when a subject was in the stage W or N1 respectively. The cell values then decreased to negative values (i.e., becoming inactive) during stages N2, N3 and REM (or R in short). Fig. 4 illustrates the changes of this cell value according to a sequence of sleep stages predicted by our model. The sleep stages are arranged according through time from left-to-right, and top-to-bottom. Each sleep stage color corresponds to $tanh(c)$, where $+1$ (i.e., active) is blue and -1 is red (i.e., inactive). There were also other interpretable cells such as the ones that started with a high value at the beginning of each subject data and then slowly decreased with each sleep stage until the end of the subject data, or the ones that turned on when they found a continuous sequence of stages N3 and REM. The existence of these cells showed that the LSTMs inside the sequence residual learning part learned to keep track of the current status of each subject, which is important to correctly identify the next sleep stages according to the stage transition rules [3].

V. Discussion

We propose the DeepSleepNet model that utilizes CNNs and bidirectional-LSTMs to automatically learn features for sleep stage scoring from raw single-channel EEGs without

Fig. 4. An example of the LSTM cell that is active at the beginning of wakefulness (i.e., stage W) or the sleep onset (i.e., stage N1). The sequences of sleep stages are the predictions from DeepSleepNet on one subject data, arranged through time from left-to-right and top-to-bottom. The background color of each stage corresponds to $tanh(c)$, where $+1$ is blue and -1 is red.

using any hand-engineered features. The results showed that, without changing the model architecture and the training algorithm, the model could be applied on different EEG channels (F4-EOG (Left), Fpz-Cz and Pz-Oz). It achieved similar overall accuracy and macro F1-score compared to the state-of-the-art hand-engineering methods on both the MASS and Sleep-EDF datasets, which have different properties such as sampling rate and scoring standards (AASM and R&K). The results also showed that the temporal information learned from the sequence residual learning part helped improve the classification performance. These demonstrated that our model could automatically learn features for sleep stage scoring from different raw single-channel EEGs.

There are two main reasons that we evaluated our model with the F4-EOG (Left) channel from the MASS dataset, which is different from most of the existing methods reported in the literature that rely on the electrodes at the central lobe such as Cz, C4 and C3. The first reason is to compare the scoring performance with our previous hand-engineering approach. The second reason is that it is much easier and more comfortable to collect data either at sleep clinics or from home environment compared to the existing methods. This is because both of the electrodes do not have problems of reading the electrical activity from the hairy scalp. Even though the F4-EOG (Left) channel does not have information from the central and occipital lobes as recommended in the AASM manual [3], our results showed that our model was still able to achieve a similar performance compared to the state-of-the-art methods.

Based on the results of our simple model analysis, we found that our model learned several interesting features that were consistent with the AASM manual (which is the same manual the experts followed to score the MASS dataset). In the representation learning part, some of the learned filters at the first convolutional layers of the two CNNs were mostly active for stage N2-N3 and W-N1-REM (see Fig 3). This implies that our model recognized some patterns that are similar among such stages. Our model might learn the filters to detect sleep spindles that can appear in both N2 and N3 stages, and to detect different features of the eye movements from the EOG (Left) that can be used to distinguish among W, N1 and REM stages. Also, in the sequence residual learning part, we found some interpretable memory cells in the bidirectional-LSTMs such as the cells that were keeping track of the wakefulness or the sleep onset, the cells that increased or decreased its value over time, and the cells that detected a train of stage N3 and REM. Our model utilized a combination of these cells to understand the current status of each subject, and to formulate transition rules. For instance, our model might remember that the subject was now awake or in the stage W. The next possible stage was very likely to be either stage W or N1. It should be emphasized that our model can learn these features from raw single-channel EEG without utilizing any hand-engineered features. Moreover, we observed that the features that our model learned were consistent across different folds. Therefore, we believe that DeepSleepNet is a better approach to implement automatic sleep stage scoring system compared to the hand-engineering ones that require prior knowledge to design feature extraction algorithms.

Even though our results are encouraging, our model is still subject to several limitations. Firstly, our model requires being trained with a sufficient amount of sleep dataset. This is due to the nature of the deep learning techniques that require a significant amount of training data to learn useful representations from the data. We performed additional experiments with the MASS dataset to estimate the number of epochs required to train our DeepSleepNet. We tried different numbers of folds (i.e., k) for the k-fold cross-validation from 2 to 31. We found that the scoring performance started to drop when k was less than 12, or when the number of training epochs was approximately less than 54000. Secondly, as our model learns features from the training data, it might not perform well when the trained model is applied to the data that have properties different from the training data such as data from different EEG channels. The model might have to be re-trained or fine-tuned before it can be applied to the data with different properties. Lastly, as our model utilizes bidirectional-LSTMs, the model has to wait until it has collected enough 30-s EEG epochs (depending on the sequence length of the EEG epochs used during the training process) before it can score these epochs. For instance, when the sequence length is set to 25, the model has to wait for 30×25 seconds (or 12.5 minutes) before it can identify sleep stages for these 25 EEG epochs.

VI. Conclusion and Future Work

We propose a deep learning model, named DeepSleepNet, for automatic sleep stage scoring based on raw single-channel EEG without utilizing any hand-engineered features. Our model utilizes CNNs to extract time-invariant features, and bidirectional-LSTMs to learn stage transition rules among sleep stages from EEG epochs. We also implement the two-step training algorithm that pre-trains our model with the oversampled dataset to alleviate class-imbalance problems, and fine-tunes the model with the sequences of EEG epochs to encode the temporal information into the model. Our results

showed that, without changing the model architecture and the training algorithm, our model was able to automatically learn features for sleep stage scoring from different raw single-channel EEGs from two datasets that have different properties and scoring standards. Our model analysis results also demonstrated that our model learned several features that are consistent with the AASM manual. As our model automatically learn features from raw EEG, we believe that DeepSleepNet is a better approach to realize a remote sleep monitoring compared to the hand-engineering ones.

In the future, we plan to improve our DeepSleepNet to be able to apply on the single-channel EEG such as the F4-EOG (Left) and Fp2-EOG (Left) collected from wearable devices.

ACKNOWLEDGMENT

The authors would like to thank Douglas McIlwraith and Axel Oehmichen from Imperial College London who reviewed the contents of the paper, and provided support for running the k-fold cross-validation.

REFERENCES

[1] K. Wulff, S. Gatti, J. G. Wettstein, and R. G. Foster, "Sleep and circadian rhythm disruption in psychiatric and neurodegenerative disease," *Nature Rev. Neurosci.*, vol. 11, no. 8, pp. 589–599, 2010.

[2] J. A. Hobson, "A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects," *Electroencephalogr. Clin. Neurophysiol.*, vol. 26, no. 6, p. 644, 1969.

[3] C. Iber, S. Ancoli-Israel, A. L. Chesson, Jr., and S. F. Quan, *The AASM Manual for the Scoring of Sleep and Associated Events*, Westchester, IL, USA: American Academy of Sleep Medicine, 2007.

[4] T. Lajnef *et al.*, "Learning machines and sleeping brains: Automatic sleep stage classification using decision-tree multi-class support vector machines," *J. Neurosci. Methods*, vol. 250, pp. 94–105, Jul. 2015.

[5] C.-S. Huang, C.-L. Lin, L.-W. Ko, S.-Y. Liu, T.-P. Su, and C.-T. Lin, "Knowledge-based identification of sleep stages based on two forehead electroencephalogram channels," *Frontiers Neurosci.*, vol. 8, p. 263, Sep. 2014.

[6] S. Güneş, K. Polat, and Ş. Yosunkaya, "Efficient sleep stage recognition system based on EEG signal using k-means clustering based feature weighting," *Expert Syst. Appl.*, vol. 37, no. 12, pp. 7922–7928, 2010.

[7] O. Tsinalis, P. M. Matthews, and Y. Guo, "Automatic sleep stage scoring using time-frequency analysis and stacked sparse autoencoders," *Ann. Biomed. Eng.*, vol. 44, no. 5, pp. 1587–1597, 2016.

[8] R. Sharma, R. B. Pachori, and A. Upadhyay, "Automatic sleep stages classification based on iterative filtering of electroencephalogram signals," in *Neural Computing and Applications*. London, U.K.: Springer, 2017, pp. 1–20.

[9] A. R. Hassan and A. Subasi, "A decision support system for automated identification of sleep stages from single-channel EEG signals," *Knowl.-Based Syst.*, vol. 128, pp. 115–124, Jul. 2017.

[10] M. Längkvist, L. Karlsson, and A. Loutfi, "Sleep stage classification using unsupervised feature learning," *Adv. Artif. Neural Syst.*, vol. 2012, Jan. 2012, Art. no. 107046.

[11] O. Tsinalis, P. M. Matthews, Y. Guo, and S. Zafeiriou. (Oct. 2016). "Automatic sleep stage scoring with single-channel EEG using convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1610.01683

[12] Y.-L. Hsu, Y.-T. Yang, J.-S. Wang, and C.-Y. Hsu, "Automatic sleep stage recurrent neural classifier using energy features of EEG signals," *Neurocomputing*, vol. 104, pp. 105–114, Mar. 2013.

[13] H. Dong, A. Supratak, W. Pan, C. Wu, P. M. Matthews, and Y. Guo. (Oct. 2016). "Mixed neural network approach for temporal sleep stage classification." [Online]. Available: https://arxiv.org/abs/1610.06421

[14] M. X. Cohen, *Analyzing Neural Time Series Data: Theory and Practice*. Cambridge, MA, USA: MIT Press, 2014.

[15] S. Ioffe and C. Szegedy. (Feb. 2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." [Online]. Available: https://arxiv.org/abs/1502.03167

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.

[17] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proc. IEEE-INNS-ENNS Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3. Jul. 2000, pp. 189–194.

[20] H. Sak, A. Senior, and F. Beaufays. (Feb. 2014). "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition." [Online]. Available: https://arxiv.org/abs/1402.1128

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. (Dec. 2015). "Rethinking the inception architecture for computer vision." [Online]. Available: https://arxiv.org/abs/1512.00567

[22] D. Kingma and J. Ba. (Dec. 2014). "Adam: A method for stochastic optimization." [Online]. Available: https://arxiv.org/abs/1412.6980

[23] R. Pascanu, T. Mikolov, and Y. Bengio. (Nov. 2012). "On the difficulty of training recurrent neural networks." [Online]. Available: https://arxiv.org/abs/1211.5063

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] W. Zaremba, I. Sutskever, and O. Vinyals. (Sep. 2014). "Recurrent neural network regularization." [Online]. Available: https://arxiv.org/abs/1409.2329

[26] C. O'Reilly, N. Gosselin, J. Carrier, and T. Nielsen, "Montreal archive of sleep studies: An open-access resource for instrument benchmarking and exploratory research," *J. Sleep Res.*, vol. 23, no. 6, pp. 628–635, 2014.

[27] A. L. Goldberger *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, pp. e215–e220, Jun. 2000.

[28] B. Kemp, A. H. Zwinderman, B. Tuk, H. A. C. Kamphuisen, and J. J. L. Oberyé, "Analysis of a sleep-dependent neuronal feedback loop: The slow-wave microcontinuity of the EEG," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 9, pp. 1185–1194, Sep. 2000.

[29] T. D. Lagerlund, "Manipulating the magic of digital EEG: Montage reformatting and filtering," *Amer. J. Electroneurodiagnostic Technol.*, vol. 40, no. 2, pp. 121–136, 2000.

[30] J. Cohen, "A coefficient of agreement for nominal scales," *Edu. Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, 1960.

[31] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, Jul. 2009.

[32] M. Abadi *et al.* (Mar. 2016). "TensorFlow: Large-scale machine learning on heterogeneous distributed systems." [Online]. Available: https://arxiv.org/abs/1603.04467

[33] A. Karpathy, J. Johnson, and L. Fei-Fei. (Jun. 2015). "Visualizing and understanding recurrent networks." [Online]. Available: https://arxiv.org/abs/1506.02078

Akara Supratak is currently pursuing the Ph.D. degree with the Department of Computing, Imperial College London, Data Science Institute. His research is in the area of bio-medical engineering, software engineering, and deep learning.

Hao Dong is currently pursuing the Ph.D. degree with the Department of Computing, Imperial College London, Data Science Institute. His research is in the areas of deep learning and bio-medical engineering, especially with EEG data.

Chao Wu is currently a Research Associate with the Department of Computing, Imperial College London, Data Science Institute. His research is in the area of big data analysis, modeling, and applications.

Yike Guo is currently a Professor of Computing Science and a Director of the Department of Computing with the Imperial College London, Data Science Institute. His research is in the areas of large scale scientific data analysis, data mining algorithms and applications, parallel algorithms, and cloud computing.