Deep Space-Time CNN - A Deep Learning Framework for Automatic Sleeping Stage Prediction

Tianjun Yao, MS[1], Hongshen Zhao, MS[2], Shujie Chen, MS[3], Zhaoying Tang, MS[4]
[1,2,3,4]Georgia Institute of Technology, Atlanta, Georgia, US

Abstract

For this report, we will cover two experiments we conducted over two literatures we based on. One is a sleep staging prediction algorithm utilizing hybrid frequency and time domain information extracted from multiple channel sleeping EEG signal, ***Deep Space-Time CNN****; the other is a sleep stage scoring algorithm based on the raw single-channel EEG, utilizing a hybrid model of both CNN and RNN, called DeepSleepNet. We reproduced the two models and compared our result with the ones mentioned in paper. Finally, we ensembled the two models and validate this hybrid model with* ***Sleep-EDF*** *data. By looking at the result, we found that the ensemble model is able to beat the two models respectively mentioned in the original paper in fewer epochs, which indicates that it is very efficient. Besides, we have found some defects in our current result which are indications for the improvement we can do for the final report.*

Motivation and Introduction to Architecture

Sleep is central to human health, and the health consequences of reduced sleep, abnormal sleep patterns or desynchronized circadian rhythms can be emotional, cognitive, or somatic. Recent research suggests that detection of sleep and circadian disruption could be a valuable marker of vulnerability and risk in the early stages of neurodegenerative diseases, such as Alzheimer's disease and Parkinson's disease, and that treatment of sleep pathology can improve patient quality of life measures. However, most stage scoring needs to use prior knowledge, like expert judgment, which may delay the time of the early treatment. In this study, we use bio-medical applications signal data recorded through polysomnogram (PSG), which includes an electroencephalogram (EGG), and electrooculogram (EOG), an electromyogram (EMG) and an electrocardiogram (ECG) to classify sleep signal scoring without prior knowledge and expert judgment. We will use different machine learning models on the signal data and solve the sleep stage scoring problem more efficiently. Briefly speaking, we mainly involved two networks: *Deep Space-Time CNN* and *DeepSleepNet*.

For *Deep Space-Time CNN*, one fundamental problem in deep learning is the strong assumption of iid data, which makes the feature engineering of time-series data hard to deal with. In (4), (5), the author leverages Convolutional Neural Network for feature extraction and Recurrent Neural Network for time-series data prediction. Although achieving satisfactory performance, it also requires expensive computational power. Another deficiency of the CNN+RNN model is the high model complexity would require large amount of data to achieve good performance. In this research, we want to tackle this problem, that is, we want to answer the following question: is there a model that can achieve the same performance with a simpler model and less data? In Reinforcement Learning community, DQN-Deep Q-Learning Network-has been successful in many tasks, some of which involves time-series data. It has been proved that by decorrelating the time-series data, convergence can be achieved with good policy obtained. Hence, in this project, we borrow this idea of decorrelating time-series data and utilize only CNN for our sleeping stage classification task. To fully utilize sleeping data in hand, all six-channel of the EEG data is used for data preprocessing and feature engineering. In specific, we utilize both frequency domain and time domain data to feed into the CNN and perform classification task. The detailed method will be described in the next section.

For *DeepSleepNet*, it is a model for automatic sleep stage scoring based on raw single-channel EEG. Compared with other existing models in the literatures which are mainly based on hand engineered extracting features from EEG, this is obviously more convenient and make it possible for a fully automatic workflow. The architecture of *DeepSleepNet* is composed of two parts.

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. This model starts with two CNNs with small and large filter sizes at the first layers to extract time-invariant features from raw single channel 30s EEG epochs. 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). 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, and applying the rectified linear unit (ReLU) activation. Each pooling layer drops off inputs using max operation.

The second part is *Sequence Residual Learning*. It's composed of two sub sections: bidirectional-LSTMs and a shortcut connection. Like the first paper we dig into, this part employs RNN. The main advantage of RNNs is that they can be trained to learn long-term dependencies such as transition rules, and they are able to condition outputs on all previous inputs by maintaining internal memories and utilizing the feedback connections to learn temporal information from sequences of inputs. By structure, two layers of bidirectional-LSTMs are used learn temporal information such as stage transition rules which sleep experts use to determine the next possible sleep

stages based on the previous stages. Bidirectional-LSTMs extends the LSTM by having two separated LSTMs process forward and backward input sequences independently. As a result, the outputs from forward and backward LSTMs are not connected to each other. Besides, there are three 0.5 dropout layers respectively before and after the bidirectional-LSTMs. Last but not the least, a shortcut is connected 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. A fully-connected layer is also used 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. We will discuss about the algorithm in the next section.

Methods

Deep Space-Time CNN

I. Data Preprocessing and Feature Engineering

For data preprocessing stage, the first thing to do is to remove data points with unknown target label, then we remove the event-marker signal from the current dataset since it adds no value to our model. Third thing to do is to normalize the other signals from 6 channels-EEG Fpz-Cz, EEG Pz-Oz, EOG horizontal, Resp oro-nasal, EMG submental, Temp rectal- to become zero-mean and unit-variance to facilitate gradient flow.

For Feature Engineering, we want to fully utilize our dataset for the classification task. Among data from the six channels, three are sampled at 1Hz while the other three at 100Hz. Intuitively, the 100Hz signals may contain rich frequency domain information while the other three 1Hz signals may not. Therefore, we extract frequency domain information from EEG Fpz-Cz, EEG Pz-Oz, EOG horizontal and time-domain information from Resp oro-nasal, EMG submental and Temp rectal. Specifically, one data point is formed every 30 seconds. Within each 30-second period, time signals are extracted directly, forming a R^{3*30} tensor, for the other three 100Hz signals, Fast Fourier Transform is performed for the 2-second duration every second and a 200-bin spectrogram is used for the frequency signal which gives us a $R^{3*28*200}$ tensor. In conclusion, each data point consists of a time-domain signal which is a R^{3*30} tensor and a frequency-domain signal which is a $R^{3*28*200}$ tensor. Our dataset consists of 35 subjects chosen randomly from PhysioBank database.

II. Model Architecture

Having transformed our raw data into hybrid frequency and time domain signal, our deep learning model to perform the classification task is a CNN that is comprised of two sibling networks that deal with frequency-domain signal and time-domain signal respectively. For the first sibling network that take time-domain as input, no convolution operator is required as we assume the 30 elements within a 30-second window in the data point is interacted and there is no need to explore local connectivity, therefore our choice for the time-sibling network is fully-connected layers with ReLu activation function to add non-linearity in our model. As for the frequency-sibling network, our choice is to use convolutional layers to process spectrograms which is essentially a $R^{3*28*200}$ tensor. We use 3-by-3 kernels for all the convolutional layers except for the first layer. Asymmetric Kernel is leveraged for the first convolution layer as our input spectrogram in not balanced in width and height, to get our input data more balanced, the kernel size is 3-by-7 in the first layer with stride to be 1-by-2, by doing so we get a more balanced input shape while maintaining a large receptive field using the kernel size of 7 in the horizontal axis. For the remaining convolution layers our choice is 3- by-3 kernel for several reasons. First, compared to a large kernel size,
a 3-by-3 kernel lead to fewer model parameters thus not prone to overfitting and is computationally efficient. Second, with several 3-by-3 conv layer stacked up, we still maintain large receptive field with more non-linearity, thus it doesn't impair performance. Our final design choice is to replace the commonly-used MaxPooling layer with AveragePooling layer as we assume the property of frequency signal is somewhat different with human-visual signals, by applying average pooling locally, it might help us smoothing the signal while maintaining the information in frequency-domain, as we will illustrate in the following section, with average pooling layer, our performance is increased by 4% compared with max pooling layer. To prevent from overfitting, Dropout layer is leveraged with dropout parameter to be 0.25. Finally the output from two sibling network are concatenated and is fed into several FC layers and softmax layer, our design choice of loss function is Cross-Entropy Loss, however, we also experiment with Hinge Loss.

III. Training/Validation Method

As mentioned previously, to decorrelate the training examples, in each training epoch, we randomly shuffle the dataset that consists of around 80000 training examples, each of which consists of a tuple of 30-second period transformed spectrogram and time-series signal and draw 40% from the entire dataset for training phase due to the memory size limitation. Having notice that our dataset is imbalanced, stratified sampling method is also leveraged to make the label distribution for each training epoch balanced.

We hold out 10% data randomly from the entire dataset for validation purpose.

Evaluation Metric

In this project, we use accuracy, precision, recall and confusion matrix as our evaluation metric to provide us a holistic view for our model performance.

Our data pipeline and model architecture of Deep Space-Time CNN is illustrated as following.

DeepSleepNet

Based on the two-stage architecture of *DeepSleepNet*, we have two main phases for the algorithm: pre-training and fine tuning. The main purpose is to effectively train the model end-to-end via backpropagation, while preventing the model from suffering class imbalance problem present in a large sleep dataset.

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. One thing should be pointed out that two CNNs are extracted from the model and then stacked with a *softmax* layer.

It is important to note that this *softmax* is different from the last layer in the model. Please note that 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 call the output as the *pre_model*. Then the *pre_model* is trained with a sample balanced training set using a mini-batch gradient-based optimizer called Adam with an appropriate learning rate. We will discuss about the balancing strategy in detail in data information section. This pre-training is only conducted over the Representation Learning.

The second step is fine-tuning, performing a supervised improvement on the whole model with a sequential training. Contrast to the pre-training, fine tuning is done over the whole model, i.e. both Representation Learning and Sequence Residual Learning. This step is to encode the stage transition rules into the model as well as to perform necessary adjustments on the pre-trained CNNs. Since we already have done the pre-training and stacked the two CNN's parameters, q_s and q_t with *softmax*, here we use them to replace the ones in the initial model. Then the model is trained with the sequence training set using a mini-batch Adam optimizer with two different learning rates, *lr1* and *lr2*. As the CNNs part has already been pre-trained, we use a lower learning rate *lr1*(1e-6) for the CNNs part and a higher learning rate *lr2*(1e-4) for the sequence residual learning part, or the pre-trained CNN parameters were excessively adjusted to the sequential data. Also, we use a heuristic gradient clipping technique to prevent the exploding gradients. This technique rescales the gradients to smaller values using their global norm whenever they exceed a pre-defined threshold. You will see the same technique is used to regularize the data in the next section.

Data Information

I. Data Source

Like most of the literatures, we use the dataset that we used to evaluate our method is a publicly available sleep PSG dataset from the PhysioNet repository. It has 7 stages: Wake, Stage 1 to 4, REM and Uknown. For data preprocessing, the very first thing we do is to remove the Unknown stage, since it's meaningless to our experiment. It's done last time before the proposal.

In detail, the original paper utilized different EEG channels from two public datasets: Montreal Archive of Sleep Studies (MASS) and Sleep-EDF. Since MASS requires a permission to access their dataset, we only used Sleep-EDF for this draft purpose. There were two sets of subjects from two studies for Sleep-EDF: age effect in healthy subjects (SC) and Temazepam effects on sleep (ST). We used 20 subjects (age 28.7±2.9) from SC. The signal 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.

II. Data Preprocessing

Deep Space-Time CNN used manual handled signal data (FFT). In contrast, as we emphasized, *DeepSleepNet* aims to automate the process of hand-engineering features by utilizing the feature extraction capabilities of deep learning. And according to this paper, we also removed MOVEMENT stage since it is overwhelmed in dataset, and again it doesn't help with the sleep research. We also merged the N3 and N4 stages into a single stage N3 to use the same AASM standard as the MASS dataset. At the same time, to alleviate the unbalance between samples of different classes (N1 and N3 take place more frequently than the others), we use the *oversample*, i.e. 1. Find the class that has the largest number of samples 2. randomly select samples repeatedly in each class until its amount equal to that largest number.

To solve the overfitting problems, we regularize the data. For this purpose, two techniques are employed. The first is the dropout layers are employed during the training process. It should be pointed out that to provide deterministic outputs, the dropouts are removed during testing. The second technique is to avoid exploding gradients, we use L2 weight decay, which adds a penalty term into loss function. Since L2 weight decay can limit the data capabilities of learning long-term dependencies, and the filters of the first layers of the two CNNs overfitted to noises in EEG data, we only apply the L2 weight decay for the first layer.

Baseline Results

The sleeping signal prediction is multi-class problem, and Neural Network is good in the signal classification. However, there are also a bunch of methodology can solve multi-class problem, and in this section, we develop two experiments in using other classification method and compare the result with DeepSleepNet.

In order to approve the advantage of both data processing and methodology in our DeepSleepNet model, here we use the data without too much data processing.

6 features are utilized in the modelling, they are EEG Fpz-Cz, EEG Pz-Oz, EOG horizontal,Resp oro-nasal, EMG submental, Temp rectal, Event marker. The raw data contains sleeping signal of 4 patients and have 6 labels, W, N1, N2, N3, N4, R, and the amount of signal for each label is shown as below:

Table 1: Count of Each Class

Target Label	Count
W (label =0)	7914
N1 (label=1)	2687
N2 (label=2)	17176
N3 (label=3)	5186
REM (label=4)	7377

It can be noted that Wake label (label=0) has much more signals than other labels, in order to balance the data, sampling is used in the data processing. For each label, we sampled 17176 classes with replacement.

There are total X models are used in this experience, SVM Linear, Ada Boost Decision Tree, Random Forest, K Nearest Neighbor, XGboost.

Linear SVC is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

The single decision tree is weak learner, and the performance is not good in most times, so here we use boosting tree: Ada boost decision tree and the algorithm are set as 'SAMME'. Random forest always performance good in big data, and we select his algorithm to compare the performance with Neural Network. The reason we choose K Nearest Neighbor is we also want to see how lazy learning algorithm perform. The final one is XGBoost, which is popular in Kaggle competition, and can have high accuracy score when the iterations are big. For XGBoost, we customize the parameter in order to get high accuracy, the max depth is 4, and learning rate is 0.1, the round is 2000.

The accuracy result is in below table:

Table 2: Accuracy, Precision and Recall results of multiple model

Model	Accuracy	Recall	Precision
Linear Regression	22.37%	22.37%	22.31%
Random Forest	35.67%	35.75%	40.34%
SVM	33.18%	38.12%	37.49%
KNN	23.52%	25.45%	28.66%
XGBoost	39.84%	39.73%	42.58%

Just as is expected, the simple supervised learning doesn't work well, but we can still see that XGBoost beats the other models. This inspires our idea that for the report part, we can try to "boost" over the deep learning models too. This will be discussed at the end.

Then, we tried to out the precision and recall score, as it may happen that models predict the test data into some specific class and this can have high overall accuracy but does not balance in all class.

An interesting observation here is that N1 and REM tend to be recognized as N2. Based on the above performance result, we can see our predictions have low precision and recall. This also shows both our *Deep Space-Time CNN* and *DeepSleepNet* work better not only on accuracy but also have balance prediction on each class.

Experiment Results

Deep Space-Time Net

We experiment with various model architecture and loss function for *Deep Space-Time Net*. In the figure below, it is obvious that with MaxPooling layer, the validation loss and accuracy is unstable and fluctuates a lot. The best validation accuracy obtained is also inferior to that from Average Pooling layer by around 4%. This proves our intuition that average pooling might be a better design choice for out input data, which is spectrogram rather than data for human-vision system.

We also experiment with Hinge Loss and Cross-Entropy Loss, as shown in the figure below, although the best validation accuracy for both Hinge Loss and Cross-Entropy Loss is similar- around 89.6%, with Hinge Loss the learning curve is not as stable as that using Cross-Entropy Loss, possibly due to the max operator that intends to find max margin.

Figure 4: Learning Curve for Deep Space-Time CNN

although the accuracy score is very high for the validation dataset, it is not sufficient to indicate the expressive power of our model as our dataset is unbalanced, hence to truly examine

our model performance, we also evaluate the F1 score and illustrates the confusion matrix for a holistic view.

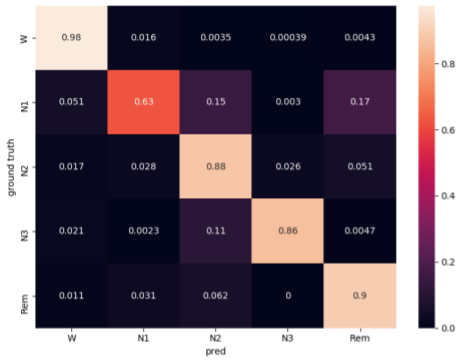

Figure 5: Heatmap of Improved Deep Space-Time CNN

Table 4: Fine-tuning result for Deep Space-Time CNN

Fine-tuning	accuracy	F1-score
training	92.3%	0.9322
validation	89.67%	0.9095

Table 4: Fine-tuning result for Deep Space-Time CNN

DeepSleepNet

We experimented the DeepSleepNet architecture with 38 patients. The size of training set was 40340 and the size of validation set was 1968. The distribution of the classes was: 7914 W, 2687 N1, 17176 N2, 5186 N3 and 7377 REM. We trained 100 epochs in the pre-train stage to train the CNN representation layers only. Following that was the fine-tuning stage with 200 epochs to train the representation layers and sequence residual layers together. The loss and accuracy curves are shown in Figure x and Figure x. Both pre-training and fine-tuning validation curves saturated after 25 epochs which meant the model actually overfitted the training data set after 25 epochs. The final validation accuracy reached 88.2% with an F1-score of 0.83. Referring to the confusion matrix shown in Figure x, the least accurately predicted class was N1 whose class accuracy was only 48%. This was because we have the least number of unique samples with N1 label from the original data set, only 2687. Although oversampling was used to resolve class imbalance problem, the model only saw duplicated samples again and again from the very small pool which had little help in capturing the features for the N1 class. The lack of enough training data contributed to the high variance in predictions. The second lowest class accuracy was 85% for the class N3 which had the second lowest number of unique samples in training.

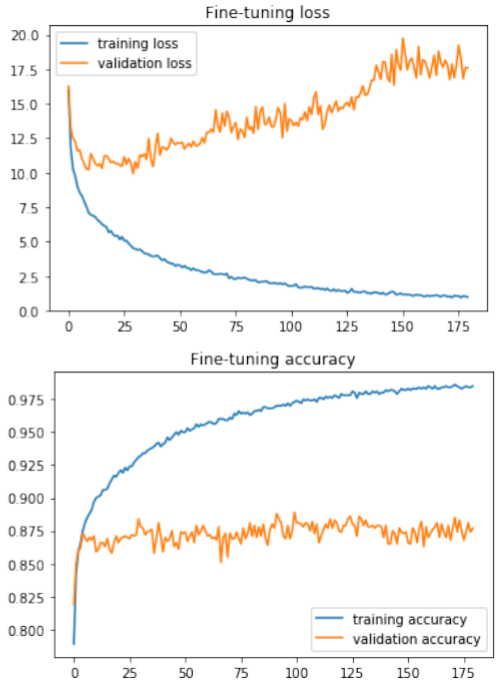

Figure 6: Learning Curve for DeepSleepNet (Validation)

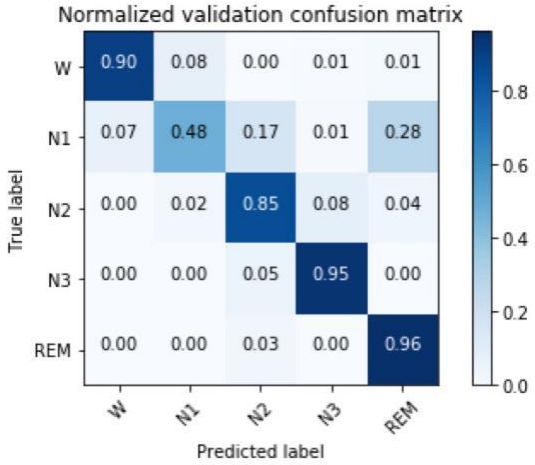

Figure 7: Heapmap for DeepSleepNet (Validation)

Table 5: DeepSleepNet pre-training accuracy

Pre-training	accuracy	F1-score
training	94.4%	0.944
validation	85.7%	0.766

Table 6: DeepSleepNet Fine-Tuning accuracy

Fine-tuning	accuracy	F1-score
training	98.5%	0.982
validation	88.2%	0.83

Improvement and Results

Overview

Again, from the two validation confusion matrices, we can see that for the two architectures we are using, they perform similar to each other. Both of them work not quite well when estimating N1, and they have relatively better performance over the other cases. Here the drawback is obvious, and our main goal is clear: we need to increase the correctness over the prediction of N1. Out of instinct, the idea of ensemble naturally jumps into our mind. In one word, we would like to let the two models do the training and prediction works in parallel, and finally combine the results to make a better estimation of N1. Actually, ***Peer Review*** may be a better term to describe this kind of combination rather than ensemble. Before we elaborate how the workflow is, one thing we want to point out here is that since either model works well over other labels, we don't have to worry about too much over the side effects resulted from this kind of combination of two models. Here are the data we need for ensemble: the *y_predicted* (predicted label of a single data point), *y_true* (true label of a single data point) and the *y_prob*, the confidential level (the probability that the prediction is correct, compared with other labels). In detail, for the predictions of W and N2, since both models works well, it is meaningless to compare which one works better: we just randomly pick one of the results. For the rest, we do need to do some data analysis here.

Data analysis over two evaluations

Here we use the evaluation based on DeepSleepNet as an example. From the project proposal, we can see the distribution of labels are as below:

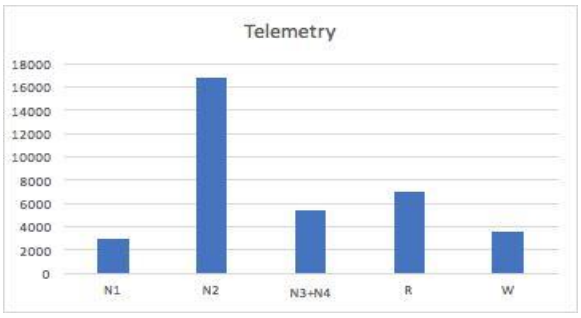

Figure 8: Statistics for different labels in dataset

Obviously, the distribution of N1 takes about 8% of the total, which takes the least part of this data sample. This gives us another reason why N1 is the most inaccurate: the dataset is too small. While in one of validation result shows that our model obviously tends to underestimate the occurrence of N1. For the validation, there are in total 1704 data points, and the number of *y_true* = N1 is 109 (around ~6.5% of the dataset), and *y_pred* = 96. For all the predictions, only 38 of them are correct. For the 109 N1, besides 38 of them are correctly predicted, there are 34 of them are predicted as N2, 35 of them are predicted as REM, and 2 of them are predicted as N3. And on the other hand, we also take a look of the 96 predictions. 38 of them correctly get N1, and all of the rest 58 give the prediction as W, or *y_pred* = N1 while *y_true* = W. In contrast, there's no such a case that *y_pred* = W and *y_true* = N1. This indicates the usage of ***Peer Review***. Whenever *DeepSleepNet* predicts the label as N1, we should check what *Deep Space-Time CNN* says. If it predicts the label as W, then we would tend to believe it is W instead N1. Similarly, if *DeepSleepNet* gives y_pred = N2, while *Deep Space-Time CNN* shows y_pred = N1, we may also believe *y_true* is N1. Certainly, coefficient may not be same in the two cases and needs further tuning. One thing we should point out is that the confusion matrix above of both *DeepSleepNet* and *Deep Space-Time CNN* are for validation (after training), not prediction. For prediction, the performance are different. See below. We can see above that *DeepSleepNet* works better in predicting N3, while *Deep Space-Time CNN* works better in N1(a little bit) and N2.

Experiment Conduction

So basically, we just wrote a simple script to apply the two rules mentioned above. One important premise is that our evaluation raw dataset feeded to two models should be the same, since we need to compare the outputs line by line. *DeepSleepNet* cut off both part of training and testing data for higher efficiency, while *DeepSpaceTime* doesn't since it is more light weighted. So we have to revisit the data loader of *DeepSleepNet* and eliminate the step of data selection. Now for evaluation data, we use SC4821G0, which is not known for both of our models. The statistics is listed below:

Number of examples = 2700

W: 1844; N1: 66; N2: 400; N3: 162; REM: 228

Experiment Results

The confusion matrix is shown below.

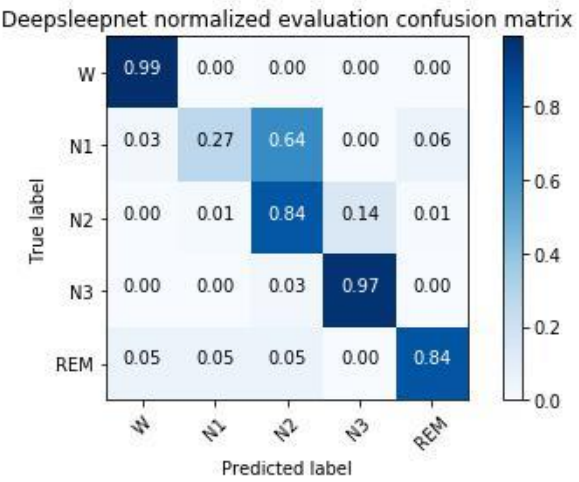

Figure 9: Confusion Matrix for prediction of DeepSleepNet

You can see that the precision of N1 increased 5%, and it doesn't harm the accuracy of N3. This proves the idea of ensemble is workable, and both of the rules are working for helping improving N1's accuracy. Since W is already high enough, the increasing is not revealed. But at the same time, it also reveals new problem, which we will discuss in the next section.

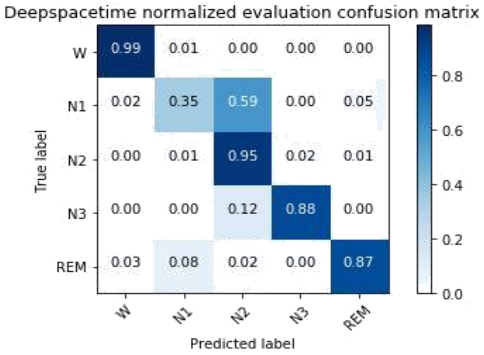

Figure 10: Confusion Matrix for prediction of Deep Space-Time CNN

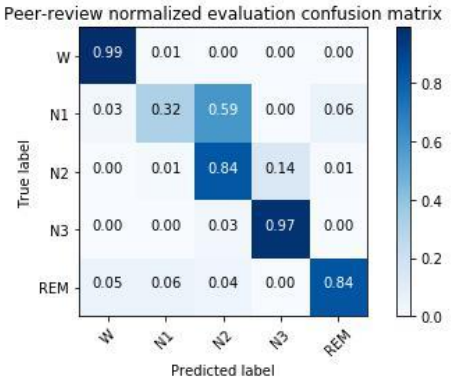

Figure 11: Confusion Matrix for Peer Review Model

Discussion and Conclusion

Overall, our idea is to ensemble multiple CNNs, using each one's advantages and results in the best performance. For our experiment, we use two models: *DeepSleepNet* and *Deep Space-Time CNN*. We use the *DeepSleepNet* as the "base" model, and *Deep Space-Time CNN* as a peer reviewer to fix the potential problem in the DeepSleepNet. Of course, we can do it oppositely. The result shows that we are able improve the drawback in our model without effecting the good ones, based on the data analysis we did, which is fine. Compared with the original results: for *Deep Space-Time CNN*, the result is comparable to many state-of-art deep learning models or even better in terms of both computational efficiency and prediction accuracy; for the DeepSleepNet, the original paper got an accuracy of 82% after fine-tuning, and our overall accuracy is 93% with fewer training epochs, which is far better than the original paper. The reason we get a higher accuracy is because we only test the model over one-fold, due to both the limit of time and hardware available. For the final report, we will go to test the model over more folds to get a better evaluation. Also, we find an interesting factor is that for the fine-tuning stage, the validation accuracy stops increasing after 25 epochs, and next time we may think about only make ~50 epochs for fine-tuning. And here are some problems we met up and possible improvement we think about.

The most import problem we find is that *Deep Space-Time CNN* has a higher value of N2 and higher value of REM, while we don't do anything for that. For the former, actually we do find a rule that for all the false positive cases of N3 (we predict it as N3 while it's not), but we cannot always choose to just believe what *Deep Space-Time CNN* or it may also harm the correctness of true labels. For example, if *Deep Space-Time CNN* says it is N2 but *DeepSleepNet* says it is N3, which one we should believe, since *Deep Space-Time CNN* works well in N2 while DeepSleepNet works not that bad in N3? For the latter, actually we failed to find this rule. But even if we find it, since both of their performance over REM are similar, simply ensemble them may harm the correctness of other labels too.

We think both problems above can be resolved by introducing new models for ensemble. For example, we can introduce a new model which is very reliable at predicting both N1 and N2. So by introducing another model with high correctness in N2, it can also help us to make better decision: if *DeepSleepNet* predicts some other labels while both of them say it is N2, then it's more believable and we will mark it N2. It will make a stronger peer review. Of course, it will be better if the new model has some different advantages with the existing ones and at least one advantage same as one of the existing models, which will make the peer review stronger. And we can give every model a weight for different labels. For example, model A is good at N1, B is good at N2 and C is good at N3. So if for A predicts a test case as N1, while B predicts REM and C predicts W, we may tend to believe A a little bit more.

Also, another method is to introduce the level of confidence. For example, for the dilemma we mentioned above, *Deep Space-Time CNN* says it is N2 but *DeepSleepNet* says it is N3. If *DeepSleepNet* only has 60% believing it is correct, while *Deep Space-Time CNN* is 99%. By knowing the drawback of *DeepSleepNet* in advance, we may want to believe *Deep Space-Time CNN* is correct.

References

1. A. Supratak, H. Dong, C. Wu, and Y. Guo. DeepSleepNet: a model for Automatic Sleep Stage Scoring based on Raw Single-Channel EEG. arXiv preprint arXiv: 1703.04046v2, 2017
2. O. Tsinalis, P. M. Matthews, Y. Guo, and S. Zafeiriou. Automatic sleep stage scoring with single-channel eeg using convolutional neural networks. arXiv preprint arXiv:1610.01683, 2016.
3. Iber, C., S. Ancoli-Israel, A. Chesson, and S. F. Quan. The AASM Manual for the Scoring of Sleep and Associated Events: Rules, Terminology and Technical Specications. Westchester, IL:American Academy of Sleep Medicine, 2007.
4. S. Biswal, J. Kulas, H. Sun, B. Goparaju, M. Brandon Westover, M. T.Bianchi, and J. Sun. SLEEPNET: Automated sleep staging system via deep learning. 26 July 2017.
5. Tsinalis, O., P. M. Matthews, and Y. Guo. Automatic sleep stage scoring using time-frequency analysis and stacked sparse autoencoders. Annals of Biomedical Engineering. 2015.
6. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
7. Pigou, Lionel, Van Den Oord, Aa ̈ron, Dieleman, Sander, Van Herreweghe, Mieke, and Dambre, Joni. Beyond temporal pooling: Recurrence and temporal convolutions for gesture recognition in video. IJCV, 2015