SleepGCN-Transformer: A Hybrid Graph Convolutional and Transformer Network for Sleep Stage Classification

A Master's Thesis

Submitted in partial fulfillment of the requirement for the award of the degree of

Master of Technology in Artificial Intelligence

by

Tanmay Rathod

23MAI007

Under the guidance of

Dr. Santosh Kumar Satapathy

Department of Information and Communication Technology

School of Technology

Pandit Deendayal Energy University

Gandhinagar – 382426. Gujarat - India

May, 2025

Approval Sheet

This thesis entitled **"SleepGCN-Transformer: A Hybrid Graph Convolutional and Transformer Network for Sleep Stage Classification"** by **Tanmay Rathod** is recommended for the degree of **M.Tech** in **Artificial Intelligence**.

Guide : Dr Santosh Satapathy,

Assistant Professor

Dept. of ICT

Dr Nitin Singh Rajput,

Assistant Professor

Dept. of ICT

Dr. Paawan Sharma,

Head Of Department

Dept. of ICT

Date: _____

Place: _____

Acknowledgment

While my name appears as the sole contributor to the completion of this summer internship, it is important to recognize that the guidance and support of many individuals played a significant role in its success. This internship is the result of a collective effort, shaped by the contributions of numerous people, to whom I am deeply grateful.

I extend my heartfelt appreciation to my internal guide, Dr. Santosh Kumar Satapathy Author, for his consistent guidance, encouragement, and insightful suggestions throughout this internship. His belief in my abilities provided the confidence and motivation necessary to complete this project. Dr. Santosh Kumar Satapathy Author's support, along with his generous provision of time and access to resources, was instrumental in achieving the objectives of this internship.

I am also profoundly thankful for the unwavering support of my parents. Their continuous inspiration, moral backing, and blessings have been the cornerstone of my journey. Their understanding and encouragement have been invaluable, and I am truly fortunate to have had their steadfast support throughout this endeavor.

Tanmay Rathod

Student Declaration

I, **Tanmay Rathod**, hereby declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Pandit Deendayal Energy University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

Tanmay Rathod

Roll No: 23MAI007

Date: _____

Abstract

Correct and effective sleep stage classification is crucial in the diagnosis of sleep disorders, conventionally done through manual interpretation of polysomnography (PSG) signals. Automated sleep stage classification is investigated in this thesis by a series of increasingly sophisticated models, from traditional machine learning to sophisticated deep learning architectures. Initial trials on the SleepEDF dataset compared models like Random Forest, XGBoost, and ensemble models, which obtained maximum accuracy of 85.35%, and set the foundation for automated systems to perform consumer-level sleep monitoring. Future work introduced deep learning models like BiLSTM and RNNs, and BiLSTM reached 81.13% accuracy, which proved to have better ability to model temporal sequences.

Based on these building blocks, we introduce *SleepGCN-Transformer*, a new hybrid model that combines Graph Convolutional Networks (GCNs) to learn spatial relations between multi-channel EEG, EMG, and EOG signals, and Transformer encoders to learn temporal patterns. The model applies Focal Loss to handle class imbalance and a CosineAnnealingLR scheduler for dynamic learning rate adaptation. Trained with the AdamW optimizer on preprocessed 30-second epochs, SleepGCN-Transformer is 93.12% training and 93.04% validation accurate. Feature attribution by LIME indicates EMG and EEG Pz-Oz to be the most significant channels, demonstrating the model's interpretability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sleep is a fundamental aspect of human health that greatly contributes to cognitive function, memory consolidation, and emotional regulation. Proper and quality sleep guarantees proper brain function and overall physical and mental health. Sleep disturbances result in several diseases, such as impaired concentration, mood disorders, and chronic conditions like cardiovascular diseases.

Sleep plays a critical function in sustaining physical and mental well-being. It is not a passive resting state but a sophisticated biological process crucial for memory consolidation, emotional homeostasis, metabolic health, and immune system function. Alterations in sleep quality or sleep patterns can reflect or lead to many health problems, such as insomnia, depression, cardiovascular disease, and neurodegenerative disorders. It is essential for clinicians and scientists to precisely assess sleep stages for the diagnosis of such conditions and the prescription of proper treatment plans.

Staging of the sleep stages is the basis of understanding the architecture of sleep and the diagnosis of sleep disorders. Sleep is divided into five major stages—N1,

N2, N3, REM, and Wake—based on unique patterns seen in brain function. Every stage of sleep is associated with particular frequencies of brainwaves, which can be picked up using the technique of electroencephalography (EEG) signals. The precise staging is important to study the quality of sleep and to detect abnormalities like insomnia, sleep apnea, and narcolepsy.

1.0.1 The Role of EEG in Sleep Analysis

Electroencephalography (EEG) is among the major methods of tracking brain activity while asleep. EEG captures electrical activity created by neurons as they fire within the brain, recorded with electrodes applied to the scalp. These electrical signals are critical in distinguishing various stages of sleep since each stage has different patterns of brain wave activity. Whereas other physiological signals are indirect, EEG is a direct view into the electrical activity of the brain and hence the basis for research and diagnosis into sleep.

1.0.2 Understanding the SleepEDF Dataset

The Sleep-EDF (Sleep European Data Format) corpus, popular among researchers, comprises polysomnography (PSG) recordings taken from healthy subjects and mildly disordered sleep patients. Recordings are taken non-invasively and usually overnight in a laboratory setting. In the course of a session, several sensors are applied to the subject's body to record a number of different physiological signals such as EEG (electroencephalogram), EOG (electrooculogram), and EMG (electromyogram).

Specifically, the database contains two EEG channels, namely Fpz-Cz and

Pz-Oz, which record frontal and parietal brain activity. Additionally, it contains EOG signals to record eye movements and EMG signals to record muscle activity, particularly around the chin region. The recordings are manually annotated by experienced sleep technicians using visual patterns and set guidelines to mark sleep stages. The last annotation is retained in a hypnogram — a time series plot that shows the changes between various stages of sleep throughout the night.

1.0.3 Sleep Stages and Signal Patterns

Human sleep is commonly divided into five stages: Wake (W), Non-Rapid Eye Movement (NREM) stages N1, N2, and N3, and Rapid Eye Movement (REM). Each stage is characterized by specific frequency patterns in EEG signals:

TABLE 1.1
Brainwave Types and Their Characteristics in Sleep Staging

Wave Type	Frequency Range	Associated Sleep Stage / Behavior
Alpha waves	8–13 Hz	Relaxed wakefulness, especially with eyes closed
Beta waves	13–30 Hz	Alertness and active thinking; decrease during sleep
Theta waves	4–8 Hz	Light sleep (Stages N1 and N2)
Delta waves	0.5–4 Hz	Deep sleep (Stage N3); synchronized neuronal firing
Sleep spindles & K-complexes	12–15 Hz (spindles)	Characteristic of Stage N2; used for stage classification

These patterns are extracted from raw EEG signals through filtering and segmentation. Experts use these patterns, along with eye movement and muscle tone data, to classify each 30-second segment (epoch) into one of the five stages.

1.0.4 The Need for Automation

Manual scoring of sleep stages, while precise, is time-consuming and subject to inter-rater reliability. It takes hours of professional scoring for one night's worth

of recording. Automated sleep stage scoring provides a quicker, more scalable, and more possibly consistent solution. Using machine learning and deep learning models, particularly those that can address intricate spatial and temporal patterns in physiological data, we can develop systems that emulate expert opinion and aid in clinical decision-making. This thesis suggests just such a system — *SleepGCN-Transformer* — that unifies graph-based representation of EEG sensor relationships with temporal learning from transformers. Our strategy not only targets high classification performance but also contributes to the increasing body of research in interpretable AI in medicine, moving us closer to clinically integrated, explainable, and fully automated sleep diagnostics.

Chapter 2

Literature Review

[1] The proposed architecture, Efficient Sleep Sequence Network (ESSN), has overcome the limitations of existing automatic sleep stage algorithms. This model addresses two main challenges. First, the model is quite complex, and often low-end systems are unable to process it; therefore, this model is designed to work on lightweight systems. The second challenge is the misclassification of the N1 stage, where models often confuse wake and REM stages. To address this, it introduces the N1 structure loss function. The ESSN model has achieved impressive metrics: 88.0% accuracy, 81.2% macro F1, and 0.831 Cohen's kappa. These results were obtained on the SHHS dataset. Additionally, it has reduced computational requirements, with only 0.27M parameters and 0.35G floating-point operations, and it claims to be faster than models like L-SeqSleepNet.

[2] The Multi-Domain View Self-Supervised Learning Framework (MV-TTFC) introduces a new approach to classify sleep stages by leveraging self-supervised learning (SSL) on unlabeled EEG data. By incorporating multi-view repre-

sentation technology, this model enhances information exchange across different views. It also introduces the multisynchrosqueezing transform, which improves the quality of the time-frequency view. Ultimately, it captures the latent features within EEG signals. It was evaluated on two datasets (SleepEDF-78 and SHHS), and MV-TTFC achieved state-of-the-art performance with accuracies of 78.64% and 81.45%, and macro F1-scores of 70.39% and 70.47%, respectively.

[3] The proposed CNN-Transformer-ConvLSTM-CRF hybrid model presents a new integration method between local and global feature extraction to enhance the classification ability of sleep stages. The model can identify relationships among EEG features by applying a multi-scale convolutional neural network combined with a Transformer for encoding features of the EEG signal and a spatio-temporal encoder via ConvLSTM. Additionally, the adaptive feature calibration module improves the extracted features, and there is efficient learning of the transition relationships between the stages of sleep by the CRF module. Based on evaluations on three datasets, this hybrid model outperforms existing state-of-the-art methods, demonstrating its efficacy in sleep stage classification.

Chapter 3

Automated Sleep Staging System with EEG Signal using Machine Learning Techniques

3.1 Methodology

We begin by importing the Sleep-EDF data provided in EDF (European Data Format). The raw EEG recordings are loaded using MNE's `read_raw_edf` function, and corresponding annotations are aligned using `read_annotations`. We optionally exclude all non-EEG channels unless explicitly required, focusing on EEG signals for our analysis. The dataset includes two EEG channels: `Fpz-Cz` and `Pz-Oz`, which are the primary input sources in our work.

During loading, we crop the signal to reduce unnecessary wake-time data, retaining 30 minutes before the first sleep stage and 30 minutes after the last. This ensures that our training data remains focused around relevant sleep activity. The

channel names are cleaned by stripping prefixes for standardization.

After loading and cropping, we segment the data into fixed-length 30-second epochs using the known sampling frequency of the recordings. Only epochs labeled with valid sleep stages — specifically stages W, N1, N2, N3, and REM — are retained. For each epoch, the raw data is extracted and structured into a format suitable for training.

Once all epochs are collected, we flatten the 3D EEG data (epochs $\times$ channels $\times$ time points) into 2D arrays (epochs $\times$ features). This is necessary for traditional machine learning classifiers. To address class imbalance in sleep stages, we apply SMOTE (Synthetic Minority Oversampling Technique) to generate balanced training samples.

The balanced dataset is then split into training and testing sets in an 80-20 ratio. We train a Machine Learning on the training data. After training, predictions are made on the test set, and the model is evaluated using standard metrics including classification report and confusion matrix. The confusion matrix is visualized using a heatmap to give a clear view of the stage-wise performance of the classifier.

3.2 Dataset Information

We have used the Sleep-EDF Dataset available at `https://www.physionet.org/content/sleep-edfx/1.0.0/`. For our experiments, we selected recordings from approximately 30 patients, resulting in 60 files — 30 EDF (European Data Format) recordings and their corresponding hypnogram annotation files. The EDF files include various bio-signals captured through multiple channels. The

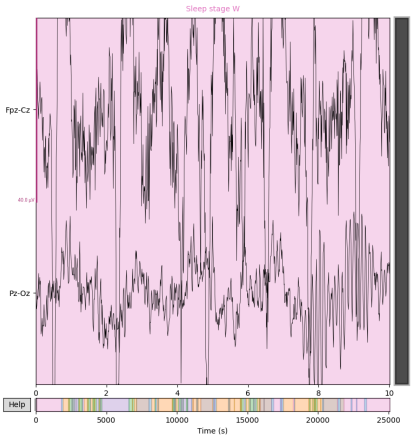

Figure 3.1. Filter EEG Chanels

hypnogram files contain sleep stage annotations, specifying the start time and type of each stage (Wake, N1, N2, N3, N4, REM) with timestamps in the HH:MM:SS format.

The EDF recordings consist of a range of physiological signals, including EEG (`Fpz-Cz` and `Pz-Oz`), EOG, EMG (submental), rectal temperature, and oro-nasal respiration. Each channel provides continuous signal recordings at a specific sampling frequency (Hz), which are used as input features for our downstream machine learning pipeline. From these signals, we derive two essential inputs: spatial features, used to construct graph-based representations, and temporal features, which capture the sequence and timing dynamics necessary for modeling the sleep process.

3.3 Preprocessing

This study applied machine learning techniques to classify sleep stages using EEG signals. The preprocessing pipeline included several crucial steps: channel map-

ping, data cropping, signal filtering, epoch creation, label mapping, and class balancing using SMOTE. These steps ensured clean, well-structured input for feature extraction and model training. The extracted features spanned time-domain, frequency-domain, and time-frequency representations. Multiple machine learning models—Gradient Boosting, Random Forest, Support Vector Machine (SVC), and Bagging—were trained and evaluated using metrics such as accuracy, precision, recall, and F1-score with cross-validation to assess generalization. The trained models were finally used to classify unseen EEG data into five sleep stages: Wake, N1, N2, N3, and REM.

A. Dataset Acquisition EEG signals were recorded using non-invasive electrodes placed on the scalp of human volunteers, capturing brain electrical activity during overnight sleep studies. Additional demographic data such as age, gender, medical history, and sleep habits were collected to provide contextual understanding. The data collection took place in controlled conditions to minimize noise and interference. Manual inspection of the raw EEG was performed to reject or correct artifacts where necessary, ensuring the quality of the input data. The final recordings were saved in standard EDF format for compatibility with downstream analysis. The dataset used in this study was sourced from publicly available Sleep-EDF recordings, which include EEG data and associated hypnogram annotations reflecting sleep stages.

B. Data Preprocessing Preprocessing was applied to prepare EEG data for modeling. Channel mapping was done to identify relevant EEG channels, particularly `Fpz-Cz` and `Pz-Oz`, and irrelevant or noisy channels were excluded. Data

cropping was applied to isolate the sleep period from wake periods based on annotation markers. The signals were then filtered to remove noise and isolate specific frequency bands important for sleep analysis.

Epochs of 30-second duration were created from continuous EEG signals to structure the data for machine learning. Sleep stage labels corresponding to these epochs were derived from hypnogram annotations. Given the naturally imbalanced class distribution in sleep stage data (e.g., fewer REM and N1 stages), the SMOTE algorithm was employed to oversample the minority classes, thereby enhancing model robustness. After preprocessing, the dataset was split into training and testing subsets. This enabled reliable training of models and unbiased performance evaluation. The output from preprocessing was then used in the feature extraction and modeling stages.

3.4 Model Architecture and Learning Framework

The core modeling pipeline begins with transforming the raw EEG data into structured epochs suitable for machine learning classification. Each EEG recording is segmented into 30-second epochs using the sampling frequency and annotated sleep stages. Only valid sleep stages (Wake, N1, N2, N3, and REM) are retained for downstream processing. After epoching, each segment is extracted using the `mne.Epochs` function, resulting in structured EEG data with consistent time windows. These segments are then reshaped into two-dimensional feature vectors where each row corresponds to an epoch and each column to a time-series sample across all EEG channels.

To address the class imbalance inherent in sleep stage data, the SMOTE (Syn-

Algorithm 1 load_sleep_physionet_raw

Input: raw_fname, annot_fname, load_eeg_only, crop_wake_mins
Output: raw

mapping ← {'EOG horizontal': 'eog', . . . }
exclude ← mapping.keys() **if** load_eeg_only **then do nothing**
raw ← read_raw_edf(raw_fname, exclude)
annots ← read_annotations(annot_fname)
raw.set_annotations(annots)
if not load_eeg_only **then**
raw.set_channel_types(mapping)
if crop_wake_mins ¿ 0 **then**
mask ← [x[-1] in {'1','2','3','4','R'} for x in annots.description]
inds ← where(mask)
tmin ← annots[inds[0]].onset - crop_wake_mins * 60
tmax ← annots[inds[-1]].onset + crop_wake_mins * 60
raw.crop(tmin, tmax)
rename EEG channel names
subj_nb ← int(fname[3:5]), rec_nb ← int(fname[5])
raw.info.subject_info ← {id: subj_nb, rec_id: rec_nb}
Return raw

Algorithm 2 Load and Filter All Files

Input: fnames, l_freq, h_freq
Output: raws

raws ← [load_sleep_physionet_raw(f[0], f[1]) for f in fnames]
for each raw **in** raws **do**
raw.load_data()
raw.filter(l_freq, h_freq)
Return raws

Algorithm 3 Epoch Creation

Input: raw_data_list, epoch_length = 30
Output: epochs_list, labels

epochs_list ← [ ]
labels ← [ ]
for each raw **in** raw_data_list **do**
sfreq ← int(raw.info['sfreq'])
(events, event_ids) ← events_from_annotations(raw)
sleep_ids ← [1,2,3,4,5]
filtered_events ← [event for event in events if event[2] in sleep_ids]
filtered_event_ids ← {k:v for k,v in event_ids.items() if v in sleep_ids}
filtered_events ← np.array(filtered_events, dtype=int)
epochs ← Epochs(raw, filtered_events, filtered_event_ids, 0, epoch_length - 1/sfreq)
epochs_list.append(epochs)
labels.extend([event[2] for event in filtered_events])
Return epochs_list, labels

Algorithm 4 ML Training and Evaluation Pipeline

Input: epochs_list, labels
Output: model performance metrics

(epochs_list, labels) ← create_epochs(raws)
X ← concatenate([epochs.get_data() for epochs in epochs_list], axis=0)
(n_epochs, n_channels, n_times) ← X.shape
X ← reshape(X, (n_epochs, n_channels * n_times))
y ← np.array(labels)
(X_resampled, y_resampled) ← SMOTE().fit_resample(X, y)
(X_train, X_test, y_train, y_test) ← train_test_split(X_resampled, y_resampled, 0.2)
model ← initialize_ML_model()
model.fit(X_train, y_train)
y_pred ← model.predict(X_test)
report ← classification_report(y_test, y_pred)
cm ← confusion_matrix(y_test, y_pred)
display(report)
display_confusion_matrix(cm)

Algorithm 5 General ML Pipeline with K-Fold Cross-Validation

Input: list of (signal_file, annotation_file) pairs
Output: average accuracy, classification report, confusion matrix

```
epochs_list = []
for file_pair in fnames:
  raw = read_raw_edf(file_pair[0])
  annot = read_annotations(file_pair[1])
  raw.set_annotations(annot)
  annot.crop(annot[1].onset - 1800, annot[-2].onset + 1800)
  events = events_from_annotations(raw)
  epochs = Epochs(raw, events)
  epochs_list.append(epochs)


all_epochs = concatenate_epochs(epochs_list)
y = all_epochs.events[:, 2]
```

Define feature extraction:

```
def eeg_power_band(epochs):
  psds = compute_psd(epochs)
  psds = normalize(psds)
  features = []
  for band in freq_bands:
    features.append(mean_psd_in_band)
  return concatenate(features)
```

Train ML pipeline with KFold:

```
pipeline = [eeg_power_band, Classifier()]
kf = KFold(n_splits=5)
results = {}
for train_idx, test_idx in kf.split(all_epochs):
  X_train = all_epochs[train_idx]
  X_test = all_epochs[test_idx]
  y_train = y[train_idx]
  y_test = y[test_idx]
  pipeline.fit(X_train, y_train)
  y_pred = pipeline.predict(X_test)
  store accuracy, y_true, y_pred
```

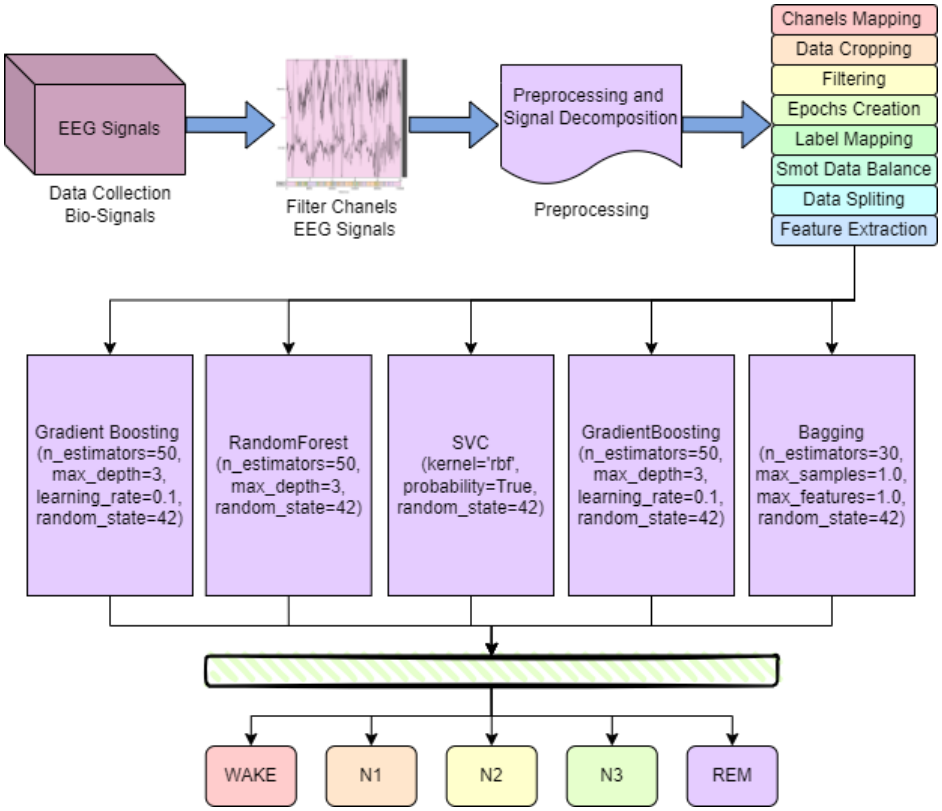

Figure 3.2. Architecture of Machine Learning Classifiers for Sleep Stage Classificatio

thetic Minority Over-sampling Technique) algorithm is applied. This balances the dataset by synthetically generating new examples in underrepresented classes. The balanced dataset is then split into training and testing sets in an 80:20 ratio.

3.5 K-Fold Cross-Validation

K-Fold Cross Validation 5-fold cross-validation approach was employed to evaluate the model's performance. The 5-fold cross-validation procedure divides the dataset into 5 equal subsets (or folds). In each iteration, the model is trained on 80% of the data (four folds) and tested on the remaining 20% (the fifth fold). This

process is repeated 5 times, with each fold serving as the test set once, ensuring that each data point is tested exactly once.

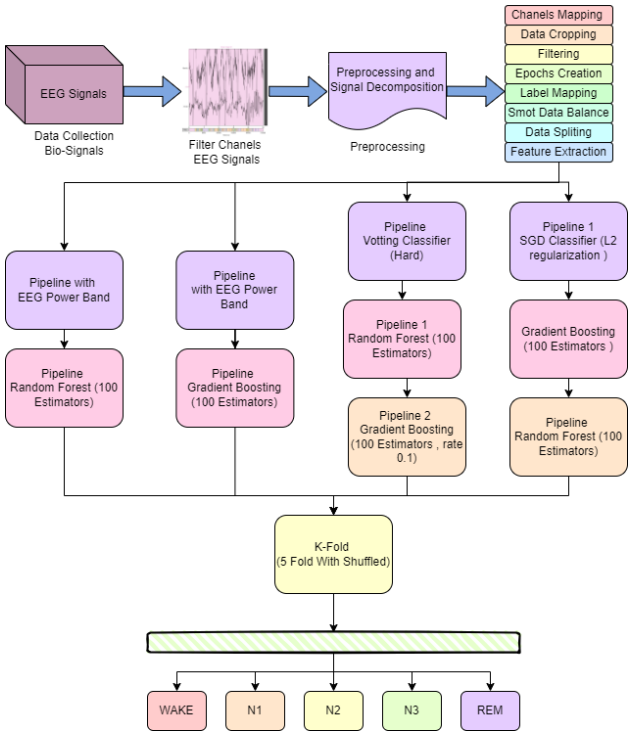

Figure 3.3. Architecture of Machine Learning Classifiers for Sleep Stage Classificatio

This approach provides a more reliable estimate of model performance by reducing the potential bias that can arise from using a single train-test split. The final performance of the model is obtained by averaging the results from each fold, offering a comprehensive evaluation of its ability to generalize to unseen data.

The advantage of K-fold cross-validation lies in its ability to provide a more robust and generalized estimate of model performance by using different subsets for training and testing in each iteration. This process reduces the risk of overfitting and ensures that the model's evaluation is based on multiple different test sets.

3.5.1 Random Forest with K-Fold Cross-Validation Pipeline

The model pipeline was created using `make_pipeline` from `sklearn.pipeline`, which integrates two main components: a function transformer and a random forest classifier. The function transformer is applied to extract the EEG power bands from the data, which are then used as features for training the model. The `RandomForestClassifier` was chosen for its ability to perform well with high-dimensional data, and it was set with 100 estimators to ensure robust performance. The random forest classifier was trained and tested on each fold using the 5-fold cross-validation setup, providing an unbiased evaluation of the model's performance.

3.5.2 Ensemble Classifier with K-Fold Cross-Validation Pipeline

The pipeline was constructed using `make_pipeline` from `sklearn.pipeline`, which integrates a function transformer for extracting EEG power bands as features and an ensemble classifier for prediction. The function transformer applies the `eeg_power_band` function to preprocess the data, and the `VotingClassifier` combines the predictions from a `RandomForestClassifier` and The ensemble classifier was trained and tested on each fold using the 5-fold cross-validation setup, offering a more robust evaluation by leveraging the strengths of both classifiers.

3.5.3 Gradient Boosting with K-Fold Cross-Validation Pipeline

In this section, we employed Gradient Boosting with 5-fold cross-validation to classify the EEG data. Gradient Boosting is an ensemble learning technique that

builds models sequentially, where each model attempts to correct the errors made by the previous ones. This method can improve predictive performance by focusing on difficult-to-predict instances. The dataset was split into five subsets, and each subset was used as the test set once, while the remaining four were used for training. The model's generalization ability was thus thoroughly evaluated.

A pipeline was constructed using `make_pipeline` from `sklearn.pipeline`. It incorporates a function transformer that preprocesses the EEG data by extracting power band features through the `eeg_power_band` function. The processed data is then passed to a `GradientBoostingClassifier`, which is used for prediction. The `KFold` cross-validation technique with 5 splits ensures that the model is trained and validated on different portions of the dataset, offering a reliable estimate of its performance.

3.5.4 Random Forest Classifier

We implemented a Random Forest classifier to establish a strong baseline model for sleep stage classification. The raw EEG recordings were segmented into 30-second epochs, and features were extracted by flattening the multidimensional epoch data. To handle class imbalance, the SMOTE algorithm was applied prior to training. The model was initialized with **100 decision trees** (`n_estimators=100`) and a fixed **random seed** (`random_state=42`) to ensure reproducibility. This configuration allowed the model to generalize well while maintaining robust performance across varying sleep stages.

3.5.5 Gradient Boosting Classifier with PCA

To improve computational efficiency and potentially boost model accuracy, we introduced Principal Component Analysis (PCA), retaining **95% of the explained variance** (`n_components=0.95`). The reduced feature set was passed to a **Gradient Boosting classifier** configured with **30 estimators** (`n_estimators=30`), a **maximum tree depth of 3** (`max_depth=3`), and a **learning rate of 0.1**. Class balancing was again addressed with SMOTE. The same 80-20 train-test split was maintained with `random_state=42`. This model leverages the sequential learning of weak classifiers to optimize classification performance over multiple iterations.

3.5.6 Ensemble Learning (Voting Classifier)

For further enhancement, we employed a soft voting ensemble that combines multiple classifiers, each with complementary strengths. The ensemble includes a **Gradient Boosting Classifier** (`n_estimators=50`, `max_depth=3`, `learning_rate=0.1`), a **Random Forest Classifier** (`n_estimators=50`, `max_depth=3`), a **Support Vector Classifier (SVC)** with an RBF kernel and probability estimates enabled, and a **Bagging Classifier** (`n_estimators=30`, `max_samples=1.0`, `max_features=1.0`). These classifiers were integrated using a **soft voting strategy** in the `VotingClassifier(voting='soft')` to average their probabilistic predictions.

Feature reduction was again performed using PCA with 95% variance retention. This ensemble approach leverages model diversity to achieve improved generalization and more stable predictions across sleep classes.

3.6 Results and Evaluation With Machine Learning

3.6.1 Random Forest

For the Random Forest model, we present the following evaluation metrics:

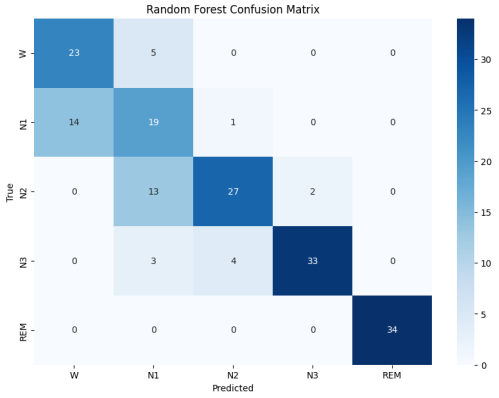

Figure 3.4. Random Forest Confusion Matrix

Figure 3.5. Random Forest Accuracy Curve

Figure 3.6. Random Forest Loss Curve

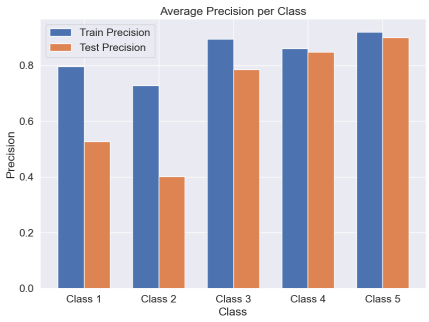

Figure 3.7. Random Forest Precision Per Class

The Random Forest classifier demonstrates solid performance with an accuracy of 0.764, precision of 0.777, and sensitivity of 0.770. Its F1-Score of 0.766 reflects a balanced performance between precision and recall.

3.6.2 Bagging Classifier

For the Bagging Classifier model, we present the following evaluation metrics:

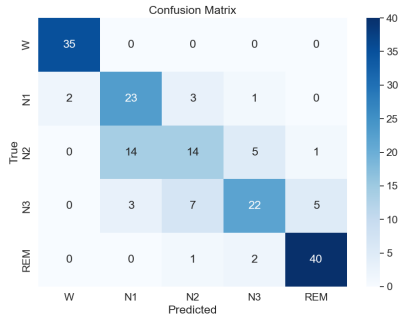

Figure 3.8. Bagging Classifier Confusion Matrix

Figure 3.9. Bagging Classifier Accuracy Curve

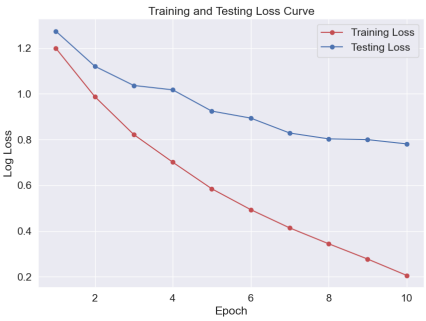

Figure 3.10. Bagging Classifier Loss Curve

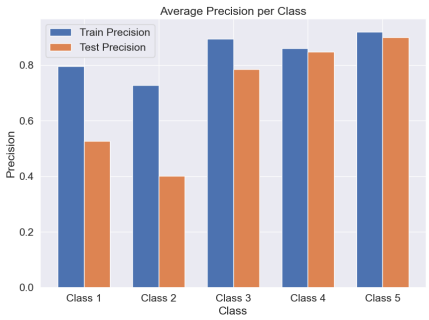

Figure 3.11. Bagging Classifier Precision Per Class

The Gradient Boosting classifier achieves moderate performance with an accuracy of 0.702, precision of 0.687, and sensitivity of 0.689. Its F1-Score of 0.687 indicates a fairly balanced trade-off between precision and recall.

3.6.3 Ensemble Learning

For the Ensemble Learning model, we present the following evaluation metrics:

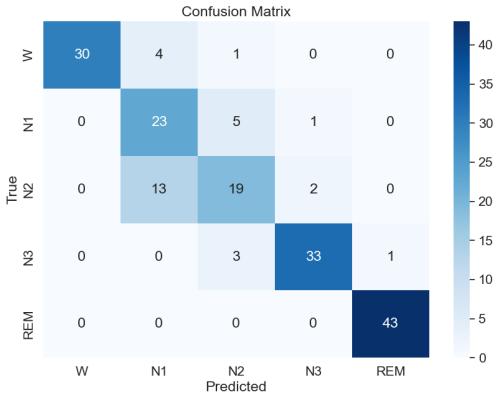

Figure 3.12. Ensemble Learning Confusion Matrix

Figure 3.13. Ensemble Learning Accuracy Curve

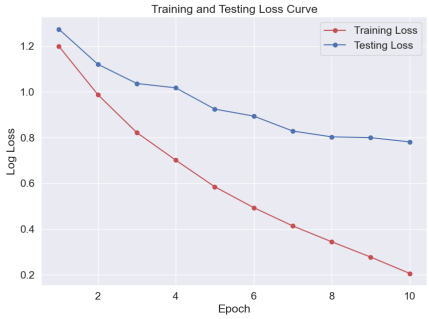

Figure 3.14. Ensemble Learning Loss Curve

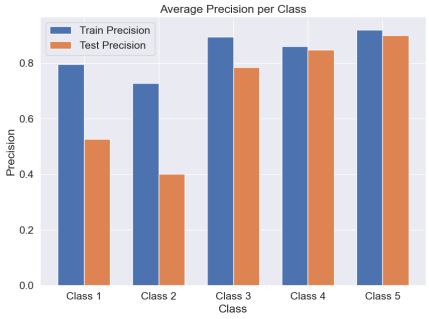

Figure 3.15. Ensemble Learning Precision Per Class

The Ensemble Learning model exhibits strong performance with an accuracy of 0.831, precision of 0.830, and sensitivity of 0.820. Its F1-Score of 0.819 underscores its effectiveness in maintaining a balance between precision and recall.

3.6.4 Gradient Boosting

For the Gradient Boosting model, we present the following evaluation metrics:

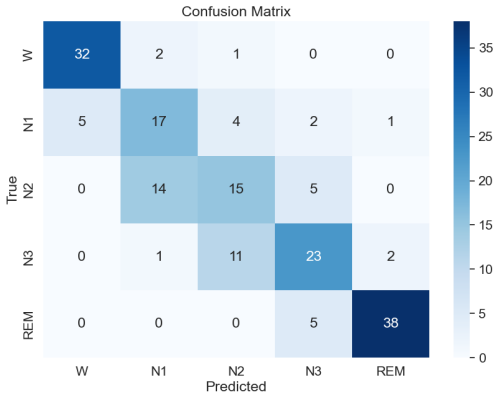

Figure 3.16. Gradient Boosting Confusion Matrix

Figure 3.17. Gradient Boosting Accuracy Curve

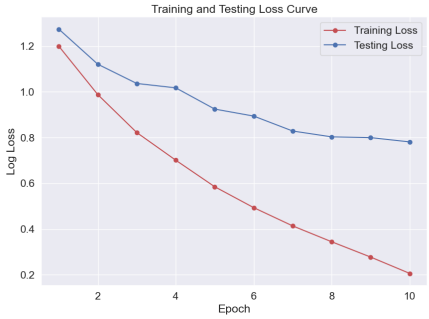

Figure 3.18. Gradient Boosting Loss Curve

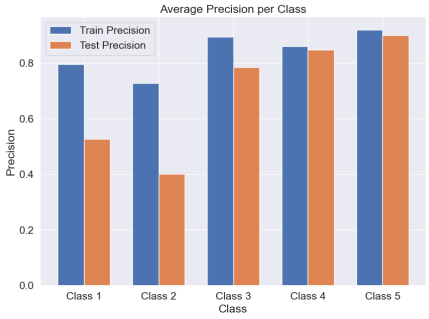

Figure 3.19. Gradient Boosting Precision Per Class

The Gradient Boosting Classifier shows competent performance with an accuracy of 0.753, precision of 0.737, and sensitivity of 0.746. Its F1-Score of 0.734 suggests a solid equilibrium between precision and recall.

TABLE 3.1
Classifier Performance Summary

Classifier	Accuracy	Precision	Recall	F1-Score
Ensemble Learning	0.95	0.86	0.79	0.82
Random Forest	0.88	0.85	0.81	0.82
Gradient Boosting	0.87	0.83	0.81	0.82
SGD Classifier	0.80	0.73	0.70	0.70

3.7 Results and Evaluation with K-fold Cross Validation

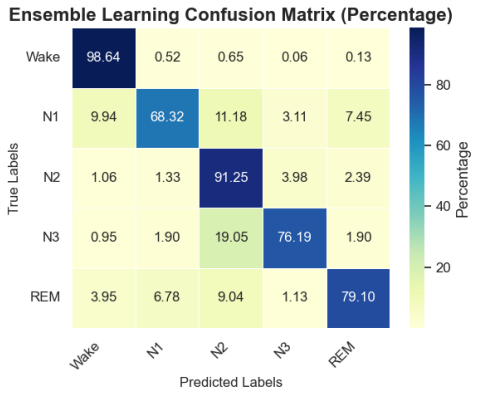

Figure 3.20. Ensemble Learning Percentage-Based Confusion Matrix

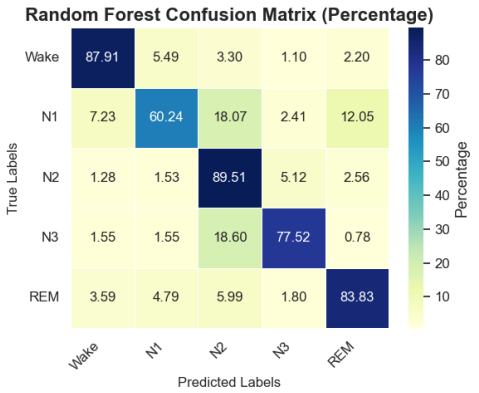

Figure 3.21. Random Forest Percentage-Based Confusion Matrix

Analysis: Ensemble Learning outperforms all other models with the highest average accuracy (95%) and strong macro F1-score (0.82), particularly excelling in the Wake stage with near-perfect precision and recall. Random Forest and Gradient Boosting both deliver balanced performance with 88% and 87% accuracy respectively, showing robust results in N2 and REM stages. SGD Classifier, while

computationally efficient, lags behind in performance, especially in accurately detecting the N1 stage, reflecting a much lower F1-score (0.70).

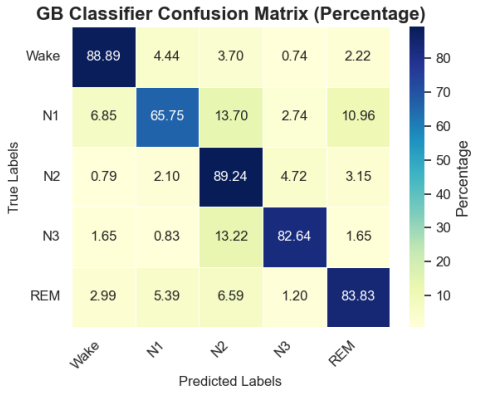

Figure 3.22. Gradient Boosting Percentage-Based Confusion Matrix

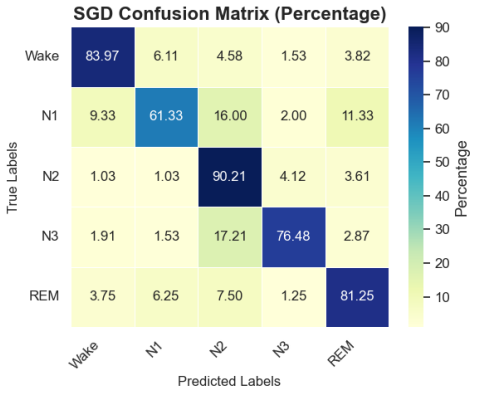

Figure 3.23. SGD Percentage-Based Confusion Matrix

3.8 Comparison Sections

The comparison tables highlight that Gradient Boosting and Ensemble Learning consistently outperform other classifiers across overall and per-stage metrics, particularly in precision and F1-score.

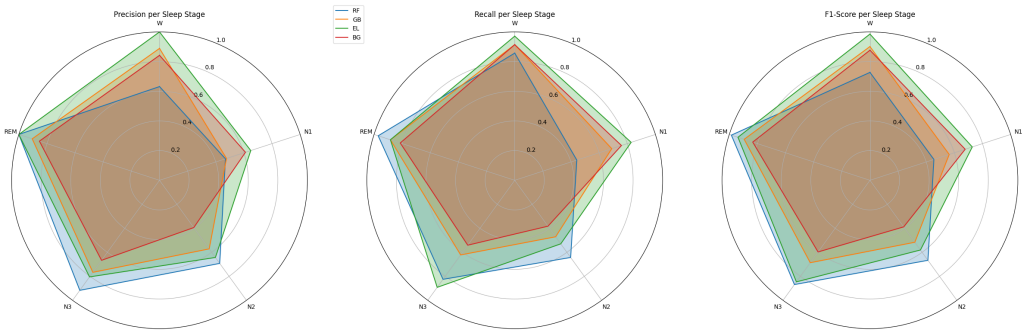

Figure 3.24. Comparison of Classifier Performance Based on Precision, Recall, and F1-Score

Notably, REM and Wake (W) stages achieve the highest precision and recall, while N1 and N2 stages remain more challenging for all models.

Classifier	Accuracy	Precision	Sensitivity	F1-Score
Random Forest	0.764	0.777	0.770	0.766
Gradient Boosting	0.702	0.687	0.689	0.687
Ensemble Learning	**0.831**	**0.830**	**0.820**	**0.819**
Bagging Classifier	0.753	0.737	0.746	0.734

TABLE 3.2
Performance Metrics of Classifiers

Stage	Metric	Random Forest	Ensemble (Gradient)	Bagging
W	Precision	0.86	**0.97**	0.95
	Recall	**0.91**	0.86	**0.97**
N1	Precision	0.50	**0.57**	0.57
	Recall	0.59	**0.79**	**0.79**
N2	Precision	0.48	**0.68**	0.56
	Recall	0.44	**0.56**	0.41
REM	Precision	0.93	**0.98**	0.87
	Recall	0.88	**0.97**	0.93

Chapter 4

Deep Neural Model for Automated Sleep Staging System using Single-Channel EEG Signal

4.1 Methodology

We employed the publicly available **Sleep-EDF** dataset for automated sleep stage classification. This dataset consists of two primary file types: Polysomnographic (PSG) recordings in European Data Format (EDF) and corresponding Hypnogram annotation files. The EDF files contain multi-channel biosignals, including EEG, while the Hypnogram files provide ground-truth sleep stage labels aligned with specific time intervals.

The dataset was partitioned into **training**, **validation**, and **testing** subsets using a **90:5:5** split. Given the inherent class imbalance in sleep stage distribution, we employed the **SMOTE** (Synthetic Minority Over-sampling Technique)

to synthetically balance the dataset, improving model robustness across minority classes.

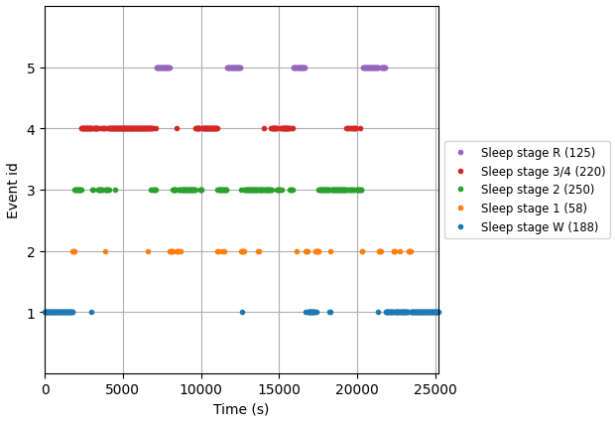

Figure 4.1. Event Mapping Plot

4.1.1 Modeling Approach

We implemented and evaluated several machine learning and deep learning architectures for the classification task. One of the core models used is the **Deep Neural Network (DNN)**, designed as a fully connected feedforward architecture suitable for non-sequential input.

Deep Neural Network (DNN)

The DNN consists of an input layer followed by three fully connected (`Dense`) layers comprising **128**, **64**, and **32** neurons respectively. Each layer employs the **ReLU** activation function to introduce non-linearity. To mitigate overfitting, **Dropout layers** with a dropout rate of **0.2** were interleaved between each dense layer. The final output layer uses a **Softmax** activation to produce class probabilities for the five sleep stages.

4.2 Dataset Information

ensure efficient implementation and reproducibility, this research adheres to a well-structured computational framework, including consistent system configuration, version control, and the integration of essential libraries for data processing and deep learning tasks. The experiments were conducted on both Windows and Linux operating systems using an **Intel Core i5 11th Gen** processor (with *Iris Xe Graphics* and a clock speed of **3.9 GHz**). The programming environment was built with **Python version 3.13.1**, managed via the `pip` package installer.

The study utilized the **Sleep-EDF** dataset, which contains EEG recordings and corresponding hypnograms for sleep stage classification. Prior to model training, the dataset was preprocessed into uniform, fixed-length segments. The dataset is publicly available at: `https://www.physionet.org/content/sleep-edf/1.0.0/`.

For project collaboration and version control, **GitHub** was employed, with the full implementation accessible at: `https://github.com/tanmay007thor/ContraWR`.

Several key libraries were used throughout the development pipeline:

- **NumPy (2.2.2)** – Numerical computations
- **scikit-learn (1.6.1)** – Machine learning utilities
- **imbalanced-learn (0.13.0)** – Handling class imbalance (SMOTE)
- **MNE (1.9.0)** – EEG signal processing
- **TensorFlow/Keras (2.16.1)** – Deep learning model implementation

- **Matplotlib (3.10.0)** and **Seaborn (0.13.2)** – Data visualization

This setup ensures smooth execution of machine learning and deep learning models, including BiLSTM and other architectures, for automating sleep stage classification.

4.3 Preprocessing Techniques

To ensure consistency in the input data, all recordings were resampled to a **100 Hz** sampling frequency. Each signal was segmented into **30-second epochs**, resulting in **3000 time steps per sample**. These samples were stored in `PKL` (Pickle) format, each approximately **48 KB** in size, enabling efficient storage and fast access, particularly beneficial for edge device deployment.

Each sample consisted of input-output pairs: the inputs (X) included two EEG channels—**FPZ-CZ** and **PZ-OZ**—while the labels (Y) represented the corresponding sleep stages. The stages were categorized into five classes: **Wake, N1, N2, N3,** and **REM**. To streamline processing, raw labels were mapped to a standardized class format.

4.4 Model Architecture and Learning Framework

Deep Neural Network (DNN)

A Deep Neural Network (DNN) is a type of multilayer feedforward architecture composed of successive fully connected layers. In this model, the input layer is followed by three dense layers containing 128, 64, and 32 neurons respectively.

Each dense layer uses the Rectified Linear Unit (ReLU) activation function to introduce non-linearity.

To improve generalization and prevent overfitting, dropout layers with a dropout rate of 0.2 are inserted between the dense layers. The final output layer employs a Softmax activation function to map the learned features to the five target sleep stages: Wake, N1, N2, N3, and REM.

TABLE 4.1
Architecture of the Deep Neural Network

Layer Type	Units	Activation Function
Input Layer	`X_train`	-
Dense Layer	128	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	64	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	32	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Output Layer	Number of classes (e.g., 5)	Softmax

While DNNs are effective in extracting complex feature representations from EEG signals, they are inherently limited in modeling temporal dependencies due to their feedforward nature. This limits their standalone utility for time-series classification tasks like sleep staging.

Forward Propagation

Forward propagation computes the activations at each layer using the following equations:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{4.1}$$

$$a^{(l)} = \sigma(z^{(l)}) \tag{4.2}$$

where:

- $z^{(l)}$ is the pre-activation (linear combination of inputs) at layer l,
- $a^{(l)}$ is the activation output at layer l,
- $W^{(l)}$ and $b^{(l)}$ denote the weights and biases of layer l,
- σ is the activation function (ReLU or Softmax depending on the layer).

Backward Propagation

During training, gradients of the loss function with respect to weights and biases are calculated using backpropagation. The process involves the following steps:

$$\delta^{(L)} = \frac{\partial L}{\partial a^{(L)}} \odot \sigma'(z^{(L)}) \tag{4.3}$$

$$\delta^{(l)} = \left(W^{(l+1)}\right)^T \delta^{(l+1)} \odot \sigma'(z^{(l)}) \tag{4.4}$$

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} \left(a^{(l-1)}\right)^T \tag{4.5}$$

$$\frac{\partial L}{\partial b^{(l)}} = \sum \delta^{(l)} \tag{4.6}$$

where:

- $\delta^{(l)}$ is the error signal at layer l,

- $\sigma'(z^{(l)})$ is the derivative of the activation function,
- $\frac{\partial L}{\partial W^{(l)}}$ and $\frac{\partial L}{\partial b^{(l)}}$ are gradients used for weight and bias updates.

Activation Functions

The Softmax function is used in the output layer to produce class probabilities:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{4.7}$$

where:

- z_i is the input to the i^{th} output neuron,
- N is the number of output classes,
- The denominator ensures the outputs form a probability distribution summing to 1.

For hidden layers, the ReLU activation function is applied:

$$f(x) = \max(0, x) \tag{4.8}$$

where:

- x is the input to the neuron,
- The function outputs x if $x > 0$, and 0 otherwise.

Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are specifically designed for modeling sequential data by incorporating feedback connections, allowing them to retain in-

formation across time steps. This architecture is well-suited for EEG signal analysis due to its ability to capture temporal dependencies.

In the proposed model, two RNN layers are employed: the first with 128 units and the second with 64 units, both using the Tanh activation function. These layers process sequential inputs while maintaining internal memory of past computations.

TABLE 4.2
Recurrent Neural Network Architecture for Sleep Stage Classification

Layer Type	**Units**	**Activation Function**
Input Layer	`X_train`	-
RNN Layer 1	128	Tanh
Dropout Layer	-	0.2 (Dropout Rate)
RNN Layer 2	64	Tanh
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	64	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	32	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Output Layer	Number of classes (e.g., 5)	Softmax

Dropout layers with a 0.2 rate are introduced after each RNN and dense layer to mitigate overfitting. The dense layers, activated using ReLU, help to refine temporal features extracted by the RNN layers. The final classification is performed by a Softmax-activated output layer corresponding to the target sleep stages.

Although RNNs are effective for learning short-term dependencies in sequential EEG data, they are prone to issues such as vanishing gradients, which can hinder the learning of long-range patterns.

The operation of an RNN cell at each time step t is mathematically represented as:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h) \tag{4.9}$$

where:

- h_t is the hidden state at time step t,
- x_t is the input at time step t,
- W_h and W_x are the recurrent and input weight matrices, respectively,
- b_h is the bias vector,
- tanh is the hyperbolic tangent activation function.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks extend traditional RNNs by incorporating memory cells and gating mechanisms, enabling them to retain long-term dependencies in sequential data. This makes them particularly effective for time-series applications like EEG-based sleep stage classification.

The proposed LSTM architecture includes two stacked LSTM layers with 128 and 64 units, respectively, both using the Tanh activation function to handle non-linear temporal patterns in the data. To prevent overfitting, dropout layers with a rate of 0.2 are placed after each LSTM and dense layer.

TABLE 4.3
LSTM Neural Network Architecture for Sleep Stage Classification

Layer Type	**Units**	**Activation Function**
Input Layer	`X_train`	-
LSTM Layer 1	128	Tanh
Dropout Layer	-	0.2 (Dropout Rate)
LSTM Layer 2	64	Tanh
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	64	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	32	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Output Layer	Number of classes (e.g., 5)	Softmax

The dense layers following the LSTM blocks (with 64 and 32 units) refine the extracted temporal features using ReLU activations. The final output layer uses the Softmax function to perform multi-class classification of sleep stages. LSTM models are particularly advantageous in capturing long-range dependencies and complex temporal dynamics, thus often outperforming simple RNNs for EEG sequence modeling.

LSTM cells operate through a series of gating mechanisms that regulate the flow of information. The key equations governing an LSTM cell at time step t are:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad \text{(Forget Gate)} \tag{4.10}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad \text{(Input Gate)} \tag{4.11}$$

$$\tilde{c}t = \tanh(W_c x_t + U_c ht - 1 + b_c) \quad \text{(Candidate Cell State)} \tag{4.12}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{(Cell State Update)} \tag{4.13}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad \text{(Output Gate)} \tag{4.14}$$

$$h_t = o_t \odot \tanh(c_t) \quad \text{(Hidden State Update)} \tag{4.15}$$

where:

- f_t, i_t, o_t represent the forget, input, and output gates, respectively.
- $\tilde{c}_t$ is the candidate cell state.
- c_t is the updated memory cell state.
- h_t is the hidden state at time t.
- σ denotes the sigmoid activation function.
- $\odot$ represents element-wise multiplication.
- W, U, b are the learned weight matrices and biases.

Bidirectional Long Short-Term Memory (Bi-LSTM)

The Bidirectional Long Short-Term Memory (Bi-LSTM) network enhances the LSTM model by processing input sequences in both forward and backward directions, enabling the model to learn context from both past and future data. This

bidirectional approach improves the model's ability to capture dependencies in temporal data, making it highly effective for tasks such as EEG-based sleep stage classification.

The architecture consists of two bidirectional LSTM layers with 128 and 64 units, respectively, both utilizing the Tanh activation function. Dropout layers (with a rate of 0.2) are applied after each LSTM layer to prevent overfitting. Following the bidirectional LSTM layers, dense layers with 64 and 32 units, using ReLU activation, are added to refine the learned features before the final classification layer, which uses the Softmax activation to classify the sleep stages.

TABLE 4.4
Bidirectional LSTM Neural Network Architecture for Sleep Stage Classification

Layer Type	**Units**	**Activation Function**
Input Layer	`X_train`	-
Bi-LSTM Layer 1	128	Tanh
Dropout Layer	-	0.2 (Dropout Rate)
Bi-LSTM Layer 2	64	Tanh
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	64	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Dense Layer	32	ReLU
Dropout Layer	-	0.2 (Dropout Rate)
Output Layer	Number of classes (e.g., 5)	Softmax

Bi-LSTM networks are particularly well-suited for EEG-based sleep stage classification, as they allow the model to capture both past and future dependencies in the EEG signal, leading to improved accuracy in detecting sleep patterns.

The Bi-LSTM model processes input sequences in both forward and backward directions, maintaining two hidden states for each timestep. The key equations describing the forward and backward passes are:

$$\overrightarrow{h}t = \text{LSTM}(x_t, \overrightarrow{h}t-1) \tag{4.16}$$

$$\overleftarrow{h}t = \text{LSTM}(x_t, \overleftarrow{h}t+1) \tag{4.17}$$

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t] \tag{4.18}$$

where:

- $\overrightarrow{h}_t$ is the forward LSTM hidden state at time step t.
- $\overleftarrow{h}_t$ is the backward LSTM hidden state at time step t.
- h_t is the concatenated hidden state, combining both forward and backward states, used for final predictions.

4.5 Results and Evaluation

4.5.1 Neural Network (NN) Model

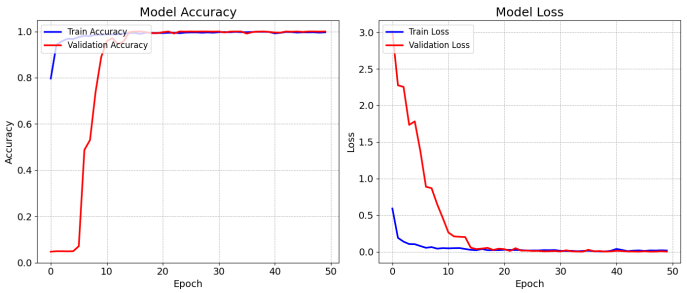

Figure 4.2. Neural Network Accuracy and Loss Comparison

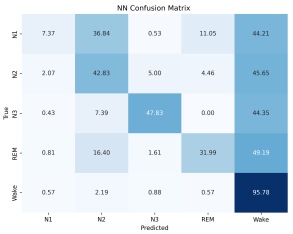

Figure 4.3. Neural Network Confusion Matrix

4.5.2 LSTM Model

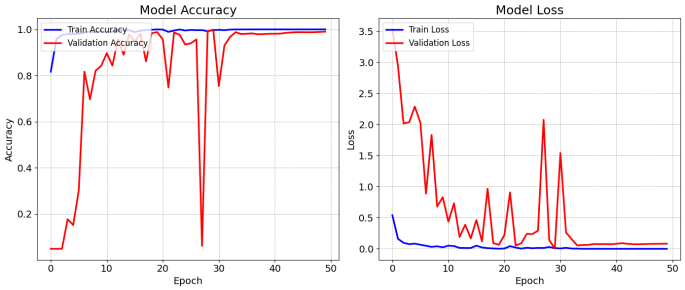

Figure 4.4. LSTM Accuracy and Loss Plot

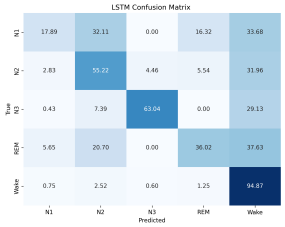

Figure 4.5. LSTM Confusion Matrix

4.5.3 RNN Model

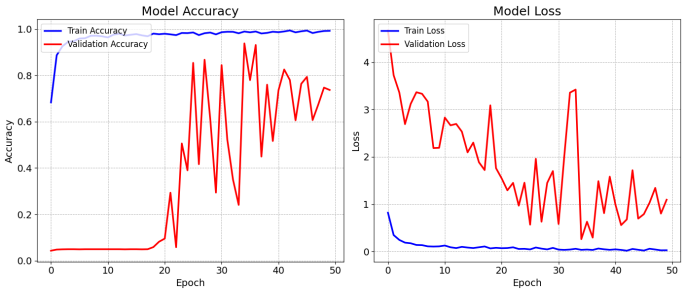

Figure 4.6. RNN Accuracy Plot

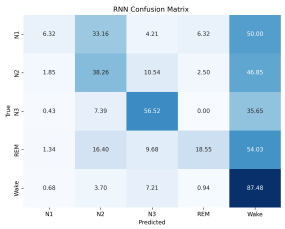

Figure 4.7. RNN Confusion Matrix

4.5.4 BiLSTM Model

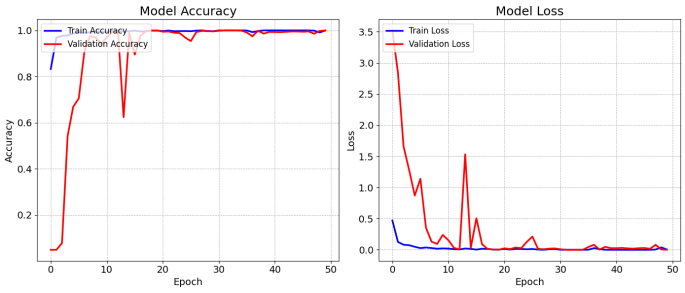

Figure 4.8. BiLSTM Accuracy Plot

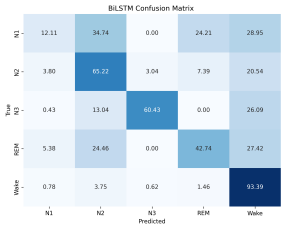

Figure 4.9. BiLSTM Confusion Matrix

4.6 Comparison of Results

This section presents a comparative analysis of multiple machine learning models applied to the sleep stage classification problem. The evaluation metrics considered include accuracy, weighted and macro averages of precision, recall, and F1-score.

4.6.1 Testing Models Overview

TABLE 4.5
Performance Comparison of Models

Model	Accuracy	Wgt. F1	Macro F1	Macro Prec.	Macro Rec.
Random Forest	**0.843**	**0.823**	0.641	**0.901**	0.558
XGBoost	**0.854**	**0.843**	0.644	**0.776**	0.599
Neural Network	**0.777**	**0.750**	0.487	0.567	0.452
LSTM	**0.800**	**0.785**	0.461	0.499	0.436
BiLSTM	**0.811**	**0.803**	0.472	0.494	0.457
RNN	0.706	0.690	0.332	0.381	0.345
AdaBoost	0.312	0.353	0.273	0.323	0.399
KNN (with PCA)	0.194	0.189	0.272	0.296	0.359

4.6.2 Discussion

Among all the models tested, the **XGBoost classifier** achieved the highest overall accuracy (0.854) and weighted F1-score (0.843), narrowly outperforming the Random Forest model. While Random Forest achieved a slightly higher macro precision (0.901 vs. 0.776), its recall across minority classes such as N1 and REM was relatively lower.

The deep learning-based models (LSTM and BiLSTM) followed closely behind the tree-based models, with BiLSTM slightly outperforming LSTM (macro

F1 of 0.472 vs. 0.461). The **Neural Network model**, although simpler, showed moderate performance (macro F1 of 0.487), better than AdaBoost and KNN.

K-Nearest Neighbors (KNN) and **AdaBoost** performed poorly in this context. KNN failed to handle the class imbalance effectively, leading to very low overall accuracy (0.194). AdaBoost performed slightly better but still struggled to generalize, especially for majority class (Wake) classification.

The **RNN model** had the lowest macro F1-score (0.332) among deep models, indicating potential vanishing gradient issues or insufficient context capture compared to LSTM-based architectures.

4.6.3 Conclusion

Tree-based ensemble methods (XGBoost, Random Forest) consistently outperformed others across all key metrics. Deep learning models, especially BiLSTM, offer competitive alternatives with relatively strong performance. Simpler models like KNN and AdaBoost underperform and are not suitable for this imbalanced multiclass classification task without further tuning or class balancing.

4.6.4 Comparison Plot

The following plot presents a comparison of the accuracy scores for different models.

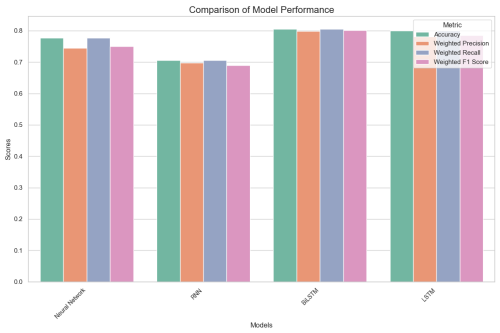

Figure 4.10. Accuracy Comparison Across All Models

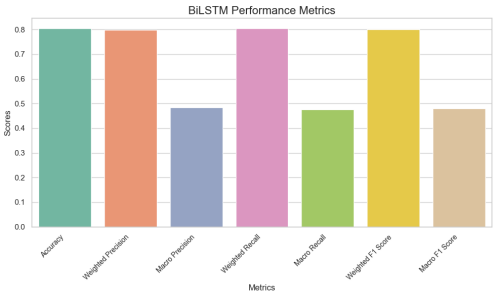

Figure 4.11. Bi-LSTM Model Performance

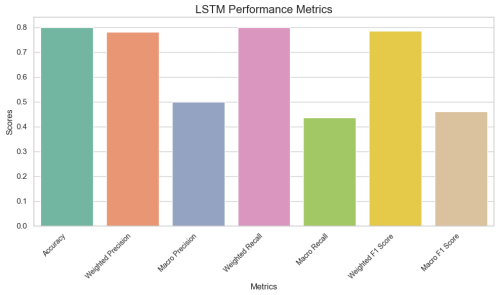

Figure 4.12. LSTM Model Performance

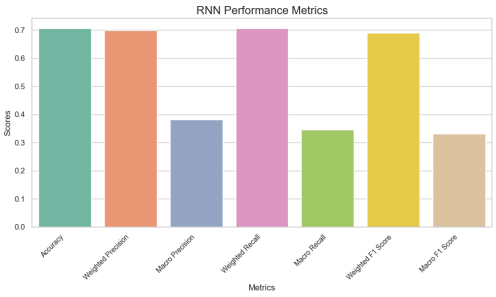

Figure 4.13. RNN Model Performance

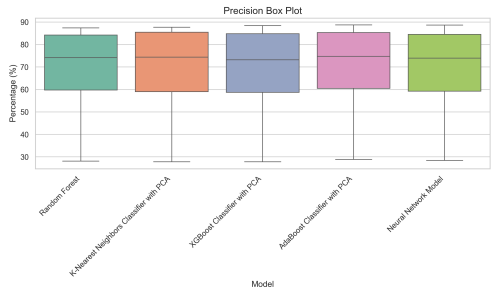

Figure 4.14. Box Plot of ML Precision Across Models

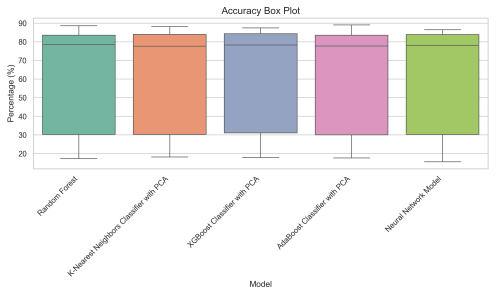

Figure 4.15. Box Plot of ML Accuracy Across Models

Chapter 5

SleepGCN-Transformer: A Hybrid Graph-Convolutional and Transformer Network for Sleep Stage Classification

5.1 Methodology

The SleepGCN-Transformer is a hybrid deep learning framework designed for accurate sleep stage classification by combining the strengths of Graph Convolutional Networks (GCNs) and Transformer architectures. This model is trained on the Sleep-EDF dataset, utilizing four critical physiological signal channels: EEG Fpz-Cz, EEG Pz-Oz, EOG (horizontal), and submental EMG.

In this architecture, GCNs are employed to capture spatial relationships between the multimodal input signals, while the Transformer encoder is tasked with modeling the temporal dependencies across time sequences. This dual approach ensures that both spatial and sequential dynamics are effectively learned.

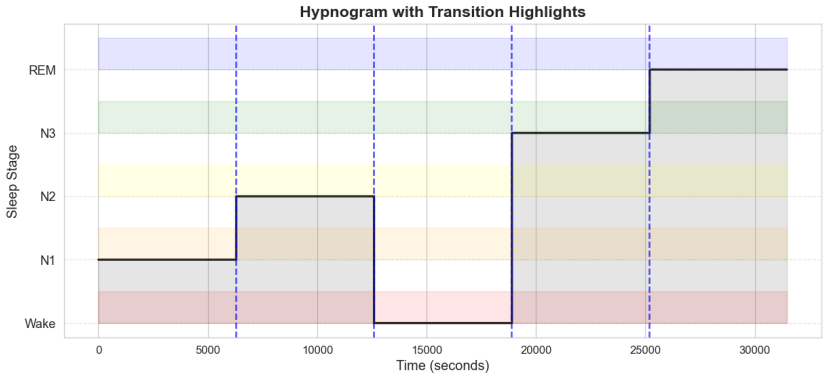

Figure 5.1. Hypnogram With Transition Heighlights

Experimental results confirm the effectiveness of the model, achieving an accuracy of 93.12% on the training set and 93.04% on the validation set. To address class imbalance—particularly the underrepresentation of certain sleep stages—the model incorporates Focal Loss, which emphasizes harder-to-classify samples during training.

Further analysis of feature importance revealed that the EMG and EEG Pz-Oz channels play a significant role in predicting sleep stages. This methodological framework not only enhances classification performance but also supports the integration of explainable AI techniques, contributing to the development of interpretable tools for clinical sleep analysis and decision-making.

5.2 Dataset and Implementation Environment

Our experiments were conducted using the publicly available Sleep-EDF dataset, which is widely utilized for sleep stage classification research. The dataset can be accessed at `https://physionet.org/content/sleep-edfx/1.0.0/`, and it includes EEG recordings labeled with corresponding sleep stages, enabling ef-

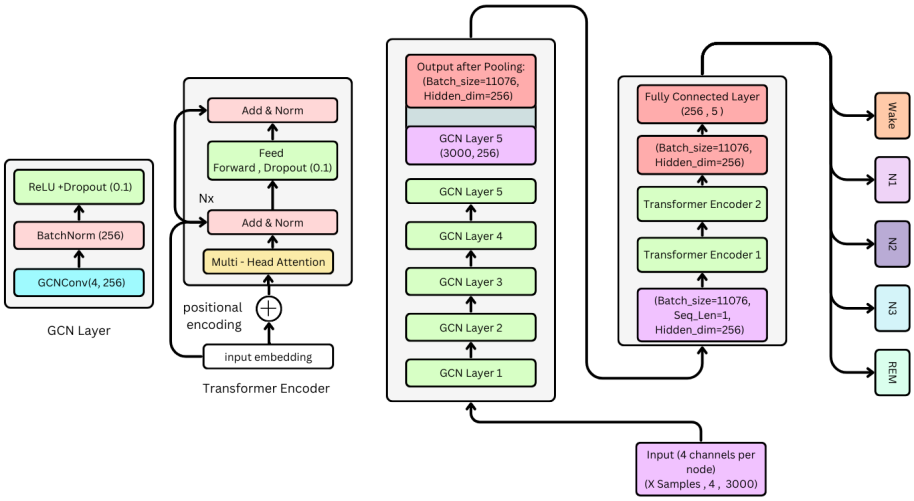

Figure 5.2. Proposed Architecture of the sleep GCN- Transformer Architecture

fective supervised learning.

The implementation was carried out in Python 3.13.1 using the PyTorch deep learning framework. Development was performed in Visual Studio Code (VS Code) on a machine powered by the Apple M1 chip, utilizing both CPU and Metal acceleration capabilities for training efficiency.

For version control and collaborative development, we used GitHub to manage our codebase. The repository containing our source code is publicly available at `https://github.com/tanmay007thor/GCN`.

The source code of this research is licensed under the Apache License 2.0, which allows for open use and distribution. The license details are available at `http://www.apache.org/licenses/`.

To support data processing, model training, and evaluation, we relied on several key Python libraries. The primary libraries and their versions used in our project include: NumPy 2.2.2, Pandas 2.2.3, Scikit-learn 1.6.1, SciPy 1.15.1, Matplotlib 3.10.0, Seaborn 0.13.2, PyTorch 2.6.0, Torch Geometric 2.6.1, NetworkX

3.4.2, and MNE 1.9.0. This software environment ensured a reliable, efficient, and reproducible workflow throughout our research.

5.3 Preprocessing Techniques

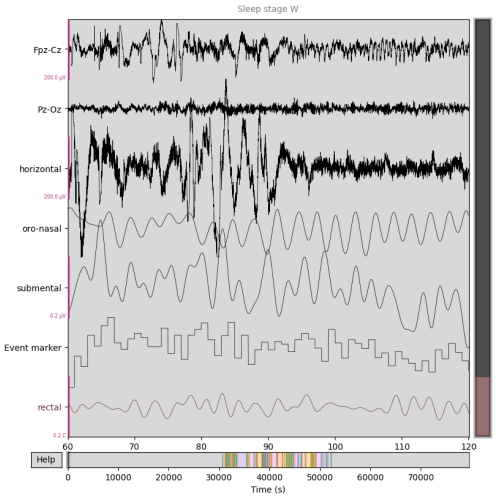

Figure 5.3. Raw signal channels from the Sleep-EDF dataset

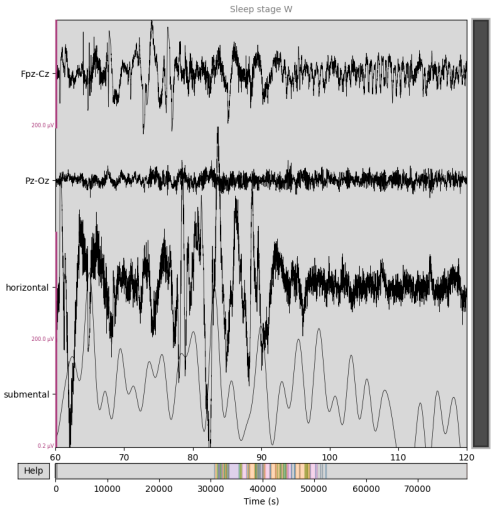

Figure 5.4. Preprocessed and filtered signal channels

5.3.1 Data Information

The Sleep-EDF dataset comprises multichannel physiological signals, including EEG, EOG, EMG, nasal airflow, and temperature, each accompanied by event markers. These signals are recorded over 60-second intervals, with frequency content varying between 60 and 120 Hz. The dynamic nature of these signals introduces significant variability, making it challenging to identify consistent patterns for classification. Such rapid fluctuations can increase the likelihood of misclassification. Furthermore, the dataset presents several challenges such as

high-frequency noise, class imbalance (notably a dominance of the Wake stage), and a substantial number of time steps—approximately 100 time steps per second—adding complexity to the task of modeling sleep stages.

5.3.2 Preprocessing Pipeline

To address these challenges, we developed a comprehensive preprocessing pipeline focused on selecting the most informative channels and reducing irrelevant data. Initially, we discarded several channels that were found to contribute minimally to sleep stage classification. The retained signals included the EEG channels (Fpz-Cz and Pz-Oz), the horizontal EOG channel (to track eye movement), and the submental EMG channel (for monitoring chin muscle activity).

Each sleep stage in the dataset was mapped to a simplified class label to consolidate similar stages and reduce classification ambiguity. Table 5.1 presents the sleep stage-to-class mapping:

TABLE 5.1
Sleep Stage Mapping

Sleep Stage	Label	Mapped Class
Wake	0	0
Stage 1	1	1
Stage 2	2	2
Stage 3	3	3
Stage 4	4	3
REM	5	4

The preprocessing begins with reading the EDF recordings and assigning sleep labels based on the mapped classes. Subsequently, we applied a bandpass filter to isolate relevant frequency ranges. Specifically, the signal was filtered to retain only the 0.3–30 Hz band using the firwin filter design, effectively removing noise

and irrelevant frequencies outside this band. Though not a standard approach, this customized filtering was found to be better suited to our specific dataset and classification task.

We then segmented the signals into fixed-length epochs of 30 seconds. The final preprocessed data was shaped into a three-dimensional format: $(X, 4, 3000)$, where X is the number of epochs (samples), 4 is the number of selected channels, and 3000 corresponds to the number of time steps per epoch.

5.3.3 Filtering Method

The filtering was implemented using a windowed time-domain design via the fir-win method. A Hamming window was employed, configured with a 0.0194 passband ripple and 53 dB stopband attenuation. The filter characteristics were as follows: lower passband edge at 0.30 Hz, lower transition bandwidth of 0.30 Hz (with a -6 dB cutoff at 0.15 Hz), upper passband edge at 30.00 Hz, and an upper transition bandwidth of 7.50 Hz (with a -6 dB cutoff at 33.75 Hz). These settings were carefully chosen to preserve the signal components relevant to sleep staging while minimizing noise.

Algorithm 6 Main Pipeline for Sleep Stage Classification

Input: list of (PSG file, Hypnogram file) pairs
Output: trained SleepGCN_Transformer model

1. fnames $\leftarrow$ [('SC4001E0-PSG.edf','SC4001EC-Hypnogram.edf')]
2. $(X_{\text{all}}, Y_{\text{all}}) \leftarrow$ `preprocess_data`$(fnames)$
3. $X_{\text{tensor}} \leftarrow$ `to_tensor`(X_{all}); $Y_{\text{tensor}} \leftarrow$ `to_tensor`(Y_{all})
4. dataset $\leftarrow$ `SleepEEGDataset`$(X_{\text{tensor}}, Y_{\text{tensor}})$
5. train_size $\leftarrow 0.8 \times |\text{dataset}|$; val_size $\leftarrow |\text{dataset}| - \text{train\_size}$
6. (train_ds, val_ds) $\leftarrow$ `split`$(dataset, [\text{train\_size}, \text{val\_size}])$
7. model $\leftarrow$ `init_model`$(4, 256, 5, 5, 4, 0.1)$
8. trainer $\leftarrow$ `init_trainer`$(model, \text{train\_ds}, \text{val\_ds}, 32, 20, 1e-4)$
9. `trainer.train()`

Algorithm 7 SleepEEGDataset Class Definition

Input: feature matrix X, labels Y
Output: graph dataset with node features and edges

1. **class** SleepEEGDataset
   (a) __init__(self,X,Y):
      i. self.X$\leftarrow X$; self.Y$\leftarrow Y$
      ii. $A \leftarrow$ `build_adjacency`(X)
      iii. self.edge_index$\leftarrow$ `find_edges`(A)
      iv. self.edge_weights$\leftarrow$ `edge_weights`(A)
   (b) len(self): **return** |self.X|
   (c) get(self,idx):
      i. node_feat$\leftarrow$ self.X[idx]
      ii. label$\leftarrow$ self.Y[idx]
      iii. **return** (node_feat,self.edge_index,self.edge_weights,label)

Algorithm 8 Preprocess Sleep EEG Data

Input: list of (psg_file, ann_file)
Output: feature matrix X_{all}, labels Y_{all}

1. $X_{\text{all}} \leftarrow \emptyset$, $Y_{\text{all}} \leftarrow \emptyset$
2. **for each** (p, a) in fnames:
   (a) raw$\leftarrow$ load PSG from p
   (b) annots$\leftarrow$ load annotations from a
   (c) raw$\leftarrow$ filter and preprocess(raw)
   (d) $E \leftarrow$ extract epochs(raw,annots)
   (e) $L \leftarrow$ extract labels(annots)
   (f) $X_{\text{all}} \leftarrow X_{\text{all}} \cup E$
   (g) $Y_{\text{all}} \leftarrow Y_{\text{all}} \cup L$
3. **return** $X_{\text{all}}, Y_{\text{all}}$

Algorithm 9 Training Pipeline for Sleep EEG Model

Input: training set, validation set, model, epochs, batch size
Output: trained model, performance metrics

1. loss_fn← define loss()
2. optimizer← init optimizer(model)
3. scheduler← init scheduler(optimizer,epochs)
4. **for** epoch = 1 to epochs:
   (a) model.train()
   (b) **for each** batch in train_ds:
      i. output←model(batch)
      ii. loss←loss_fn(output, batch.y)
      iii. backpropagate and step optimizer
   (c) scheduler.step()
   (d) model.eval()
   (e) **for each** batch in val_ds:
      i. output←model(batch)
      ii. compute validation metrics
5. **return** trained model, metrics

Algorithm 10 SleepGCN_Transformer Model Declaration

Input: input_dim, hidden_dim, output_dim, num_layers, nhead, dropout
Output: network layers initialized

1. Initialize list of GCN layers and batch norms
2. **for** *i* in 1...num_layers:
   (a) append GCNConv layer
   (b) append BatchNorm layer
3. Initialize Transformer encoder with nhead and dropout
4. Initialize final fully-connected layer
5. **return** model components

5.4 Model Architecture and Learning Framework

To extract spatial features from the multi-channel data, we construct a graph using the four selected channels, where each channel acts as a node and the edges represent their interrelationships. These edges are weighted based on prior literature and empirical observations, assigning higher importance to EEG channels due to their strong influence on sleep stage detection. In contrast, connections involving EMG and EOG are assigned lower weights, as EOG shows weaker correlations. Although assigning self-loop weights is possible, we chose not to include them in this implementation.

The final dataset comprises 11,076 samples, structured as `Data(x=[3000, 4], edge_index=[2, 12], edge_attr=[12], y=[1])`. Here, `x` represents the time-series data with 3,000 time steps across 4 channels, `edge_index` defines the graph connectivity, `edge_attr` contains the edge weights, and `y` denotes the sleep

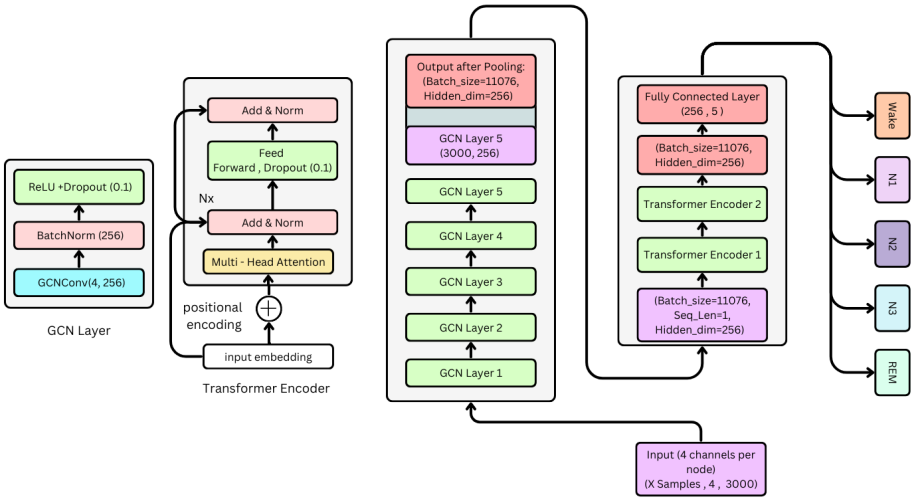

Figure 5.5. Proposed Architecture of the sleep GCN- Transformer Architecture

stage label. This dataset is created using the `SleepEEGDataset` class and is designed to effectively model spatial dependencies across EEG, EMG, and EOG signals, thereby enhancing sleep stage classification performance.

5.4.1 GCN Architecture

To capture spatial dependencies across the selected EEG, EMG, and EOG channels, we incorporate a Graph Convolutional Network (GCN) into our model. Each sample is structured as a graph with four nodes—corresponding to the four selected channels—and each node contains a time series of 3,000 time steps. The dataset comprises 11,076 such graph-based samples.

The GCN consists of five identical graph convolution layers. The initial input has a shape of (4, 3000), which is projected to (4, 256) after the first graph convolution. Each layer includes batch normalization (256 units), followed by a ReLU activation function and a dropout of 0.1. After the final GCN layer, we apply global mean pooling to obtain a graph-level representation.

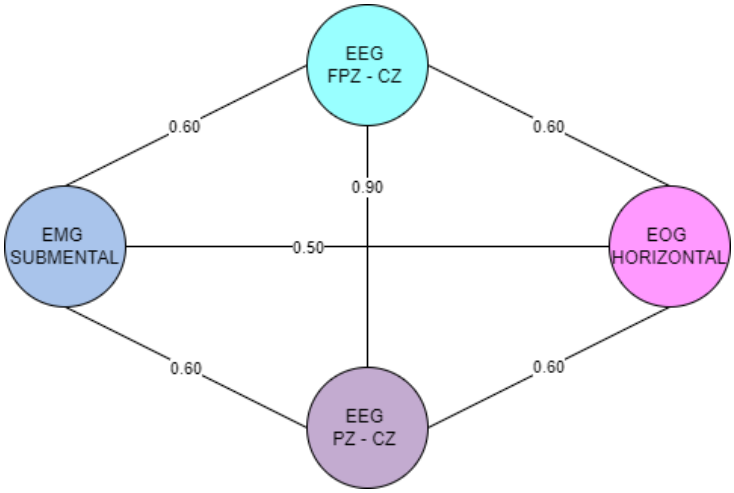

Figure 5.6. Graph Dataset Creation

The propagation rule used in each GCN layer is given by:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right) \tag{5.1}$$

- $H^{(l)} \in \mathbb{R}^{N\times D}$ is the feature matrix at layer l, where N is the number of nodes and D is the feature dimension.
- $\tilde{A} = A + I$ is the adjacency matrix with self-loops.
- $\tilde{D}$ is the degree matrix corresponding to $\tilde{A}$.
- $W^{(l)}$ is the learnable weight matrix for layer l.
- σ denotes a non-linear activation function, typically ReLU.

The simplified form for one GCN layer operation is:

$$X' = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW\right) \tag{5.2}$$

To aggregate node features into a graph-level embedding, we use global mean pooling:

$$x = \frac{1}{|V|} \sum_{v \in V} h_v \tag{5.3}$$

- V is the set of nodes in the graph.
- h_v is the feature vector of node v.
- x is the pooled feature vector representing the entire graph.

We apply standard activation and normalization functions during training:

$$\text{ReLU}(x) = \max(0, x) \tag{5.4}$$

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5.5}$$

$$x'' = \frac{x' - \mu}{\sigma} \tag{5.6}$$

- μ is the mean of x'.
- σ is the standard deviation of x'.
- x'' is the normalized feature representation.

5.4.2 Transformer Architecture

To capture temporal dependencies in the input signal, we utilize a Transformer encoder comprising two stacked layers. Initially, the input is projected into a continuous vector space using an embedding layer. Since Transformers lack inherent

mechanisms for handling sequential order, positional encodings are added to preserve time-step information.

Following embedding and encoding, we use multi-head self-attention to allow the model to focus on different aspects of the signal simultaneously. The attention outputs are concatenated and passed through a feedforward network with dropout regularization. Finally, a fully connected classifier outputs logits for four-class sleep stage classification.

The operations within the Transformer encoder are summarized using the following equations:

5.4.3 Transformer: Key Components and Equations

- **Input Embedding:** Maps input tokens into a continuous vector space.

$$X_{\text{embed}} = XW_{\text{embed}} \tag{5.7}$$

- **Positional Encoding:** Adds sequence order information using sine and cosine functions.

$$PE(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right) \tag{5.8}$$

$$PE(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right) \tag{5.9}$$

- **Query, Key, and Value Computation:** Core of attention mechanism, derived from input.

$$Q = XW_q, \quad K = XW_k, \quad V = XW_v \tag{5.10}$$

- **Scaled Dot-Product Attention:** Computes weighted output of values using attention scores.

$$H_1 = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5.11}$$

- **Multi-Head Attention:** Combines multiple attention outputs for diverse feature learning.

$$H = \text{Concat}(H_1, H_2, ..., H_h)W_0 \tag{5.12}$$

5.4.4 Training Pipeline

Fully Connected Layer: The final stage of the model includes a linear transformation layer that maps a 256-dimensional input vector to 5 output logits, corresponding to the five target sleep stages. This transformation enables direct classification output.

Why Focal Loss Instead of Standard Cross-Entropy?

Standard cross-entropy loss treats all training samples equally, which can lead to a bias toward majority classes in imbalanced datasets. In such settings, the model tends to suppress the prediction of minority classes, resulting in poor generalization. Focal Loss addresses this by dynamically adjusting the contribution of each sample based on the confidence of its prediction. This mechanism reduces the influence of well-classified samples and increases the focus on harder, misclassified instances.

Focal Loss introduces two key parameters: the focusing parameter γ, which controls the down-weighting of easy examples, and the class weighting factor α, which helps balance class frequencies during training. These features make

Focal Loss particularly effective for classification tasks involving significant class imbalance.

Focal Loss Formulation

The Focal Loss is mathematically expressed as:

$$FL(p_t) = -\alpha(1-p_t)^{\gamma}\log(p_t)$$

Here, p_t denotes the predicted probability for the target class, α is the balancing factor to compensate for class imbalance, and γ adjusts the focus on misclassified examples.

To enhance numerical stability and improve generalization, label smoothing is applied:

$$y_{\text{smooth}} = y(1-\varepsilon) + \frac{\varepsilon}{C}$$

This prevents issues such as $\log(0)$ and ensures a more stable optimization process. The combined implementation of Focal Loss with label smoothing can be written in a PyTorch-like style as:

$$L = \alpha(1-p)^{\gamma}(-y_{\text{smooth}}\log(p))$$

Learning Rate Scheduling

The learning rate is a critical hyperparameter that governs the pace of model updates during training. A high learning rate may cause divergence, while a low

learning rate can result in slow convergence or suboptimal local minima. Therefore, adaptive learning rate scheduling is essential for efficient training.

CosineAnnealingLR is used as the scheduling strategy, as it provides a smooth and gradual reduction in the learning rate. It allows for large updates in the early stages of training and fine-tuning in later stages, thereby helping the model generalize better. The learning rate at each epoch t is computed using the following cosine annealing formula:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{\pi T_{\text{cur}}}{T_{\max}}\right)\right)$$

In this equation, η_t is the learning rate at epoch t, $\eta_{\min}$ and $\eta_{\max}$ are the minimum and maximum learning rates respectively, T_{cur} is the current epoch, and $T_{\max}$ is the total number of epochs.

Training Methodology Overview

The training pipeline is encapsulated within a dedicated class named `SleepTrainer`, which manages model training, validation, and optimization. The training process begins with the computation of class weights to address data imbalance. The model is then trained using mini-batch gradient descent, where each batch is used to compute the loss and update the model parameters accordingly. Performance is periodically evaluated on a separate validation set, and the learning rate is adjusted dynamically using the scheduler.

Hyperparameters Used

The training setup uses a batch size of 32 and an initial learning rate of 0.0003. The optimizer of choice is AdamW, with a weight decay of 1×10^{-4} to prevent overfitting. Training is conducted for 20 epochs, and the CosineAnnealingLR scheduler is applied to adjust the learning rate over time. This configuration ensures smooth convergence and robust generalization.

Handling Class Imbalance

EEG sleep stage data is inherently imbalanced, with some stages appearing far more frequently than others. To address this, class weights are computed and incorporated into the loss function. The weighting formula is given by:

$$w_c = \left(\frac{\text{Total Samples}}{\text{Class Count} + 1} \right)^{0.5}$$

This formulation ensures that less frequent classes receive proportionally higher weights, encouraging the model to give them more attention during training. These computed weights are then integrated directly into the Focal Loss function to balance the influence of each class.

Testing Data Distribution Analysis

Balanced testing data is crucial for an unbiased evaluation of model performance. An imbalanced test set can lead to skewed metrics that do not accurately reflect the model's generalization capabilities. In this pipeline, the testing data is curated to ensure an even distribution across all sleep stages.

5.5 Results and Evaluation

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.13}$$

$$\text{Precision} = \frac{TP}{TP+FP} \tag{5.14}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{5.15}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.16}$$

$$\text{Support} = TP+FN \tag{5.17}$$

$$\text{Specificity} = \frac{TN}{TN+FP} \tag{5.18}$$

$$\text{Macro Precision} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i+FP_i} \tag{5.19}$$

$$\text{Weighted Precision} = \sum_{i=1}^{C}\frac{n_i}{N}\cdot\frac{TP_i}{TP_i+FP_i} \tag{5.20}$$

$$\mathscr{L}CE = -\sum i = 1^{C} y_i \log(\hat{y}_i) \tag{5.21}$$

$$\mathscr{L}_{FL} = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{5.22}$$

The model demonstrates high performance in classifying the Wake and N3 sleep stages, achieving correct classification rates of 97.14% and 97.94%, respectively. In contrast, the N2 stage is accurately classified 88.3% of the time, though it shows some confusion with neighboring N1 and REM stages. Misclassifications are more prominent for the N1 and REM stages, with the REM stage frequently misidentified as N1, exhibiting a 17.97% confusion rate. These results indicate strong general performance by the model, though enhancements may be beneficial for improving distinction between the N1 and REM stages.

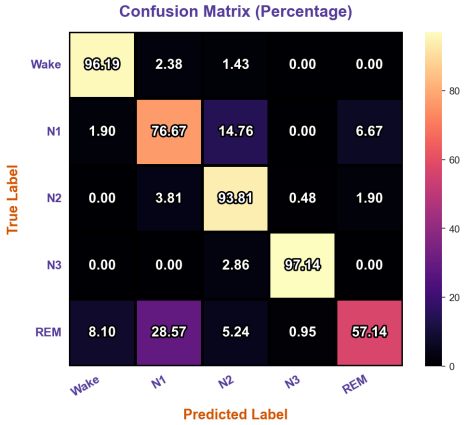

Figure 5.7. Confusion matrix (percentage)

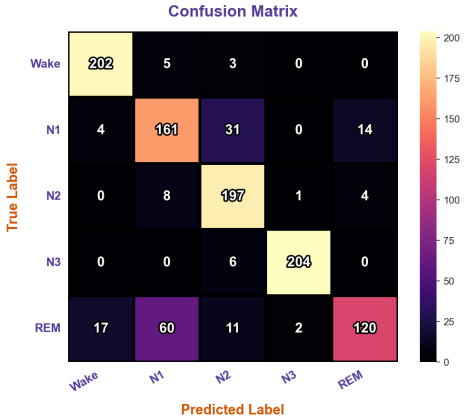

Figure 5.8. Confusion matrix (samples)

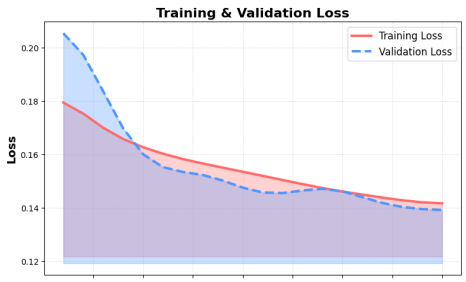

Figure 5.9. Loss plot

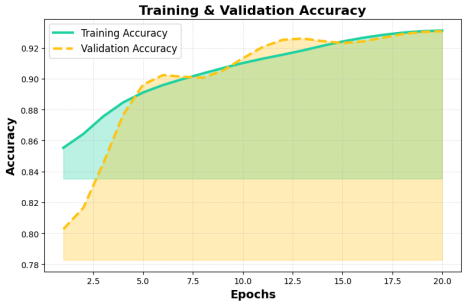

Figure 5.10. Accuracy plot

The class weight distribution used during training is as follows: 0: 1.1923, 1: 5.8152, 2: 2.4719, 3: 4.8223, 4: 4.0506. These values reflect adjustments to counter class imbalance and ensure fairer learning across all sleep stages.

The progression of model performance across selected epochs is summarized below:

The sample distribution plot reveals a nearly normal distribution of data across the various sleep stages, confirming a balanced dataset. For the test phase, approximately 1,050 samples were utilized, with 210 samples assigned to each class, ensuring equitable and unbiased performance evaluation.

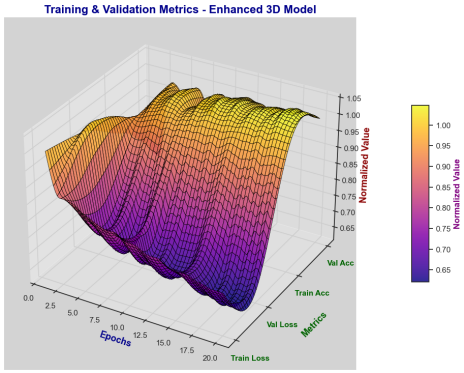

Figure 5.11. 3D Accuracy vs. Loss

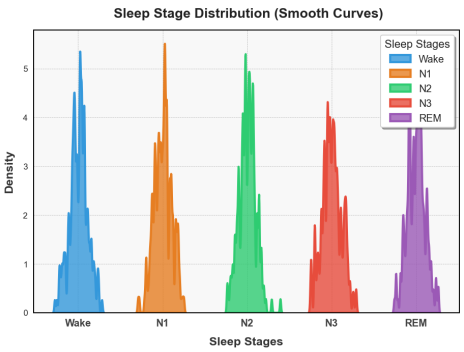

Figure 5.12. Sample distribution

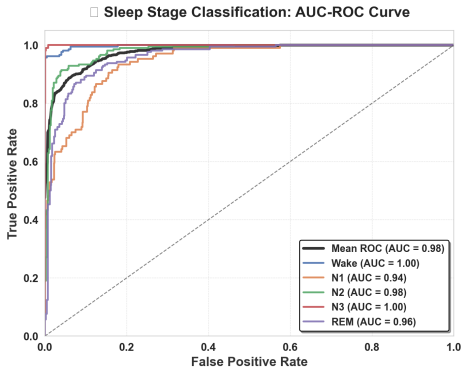

Figure 5.13. AUC vs. ROC Curve

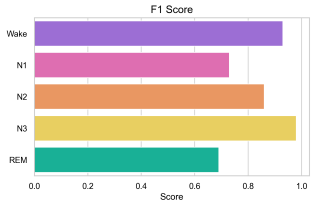

Figure 5.14. F1 Score

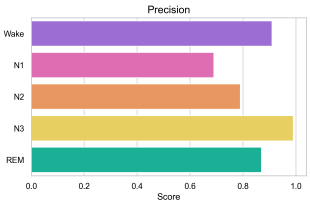

Figure 5.15. Precision

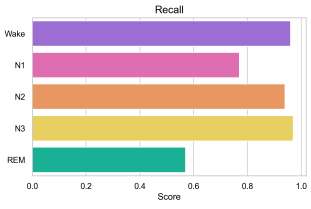

Figure 5.16. Recall

5.5.1 XAI Visualization with Heatmap and Plots

This section presents the distribution of sleep stage data, which approximates a normal distribution. The dataset is well-balanced, with an equal number of samples across all sleep stages. For testing purposes, approximately 1,050 samples were selected, ensuring that each sleep stage class contains around 210 samples. This balanced distribution supports a fair and unbiased evaluation of the model's performance.

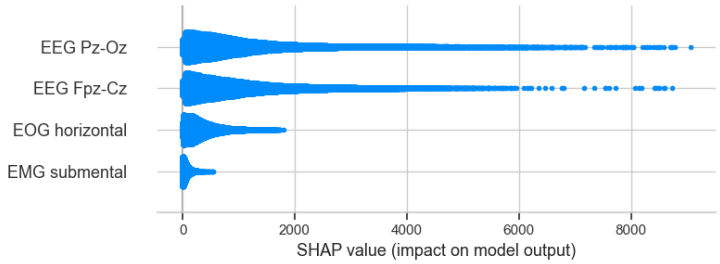

Figure 5.17. Bar plot illustrating EEG channel importance based on SHAP values. Higher SHAP values indicate greater influence on the model's sleep stage predictions.

Interpretable Sleep Stage Classification: Hypnogram Transitions and Feature Importance using LIME and SHAP

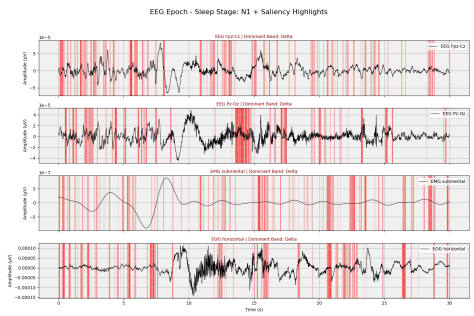

Figure 5.18. Stage N1

Figure 5.19. Stage N2

To improve the interpretability of the sleep stage classification model, we employ Explainable AI (XAI) techniques. Initially, we analyze the hypnogram transition plot, which illustrates the temporal sequence of sleep stages. This visualization helps assess whether the model's predictions align with the natural and expected progression of sleep cycles.

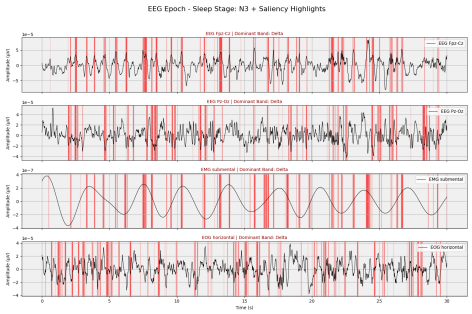

Figure 5.20. Stage N3

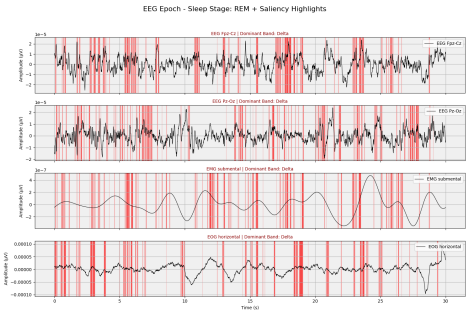

Figure 5.21. Stage REM

Subsequently, we utilize LIME (Local Interpretable Model-Agnostic Explanations) and SHAP (SHapley Additive exPlanations) to investigate feature importance. These tools provide both global and local explanations of the model's behavior. Specifically, they identify the EEG channels that significantly influence the model's decisions, offering insight into which features are most critical in sleep stage classification.

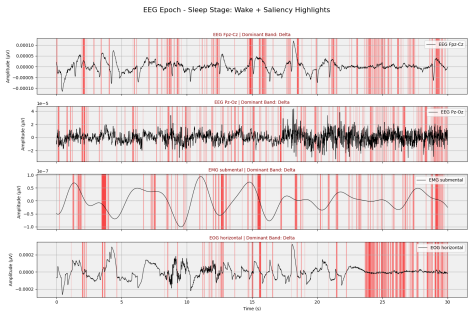

Figure 5.22. Stage Wake

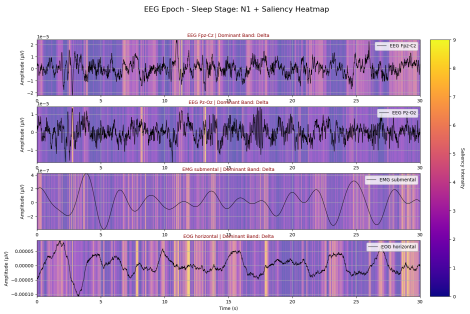

Figure 5.23. XAI Visualization

By leveraging these interpretability techniques, we ensure that the model's predictions are not only accurate but also aligned with known physiological sleep dynamics, thereby enhancing trust and transparency in the model's outcomes.

5.5.2 Time Complexity Analysis

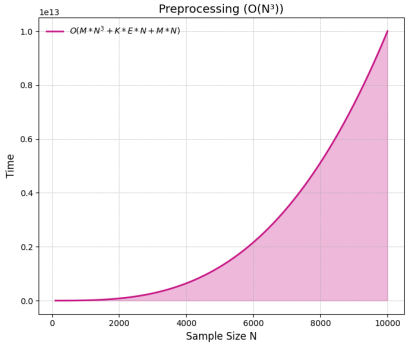

Figure 5.24. Preprocessing ($O(MN^3 + KEN + MN)$)

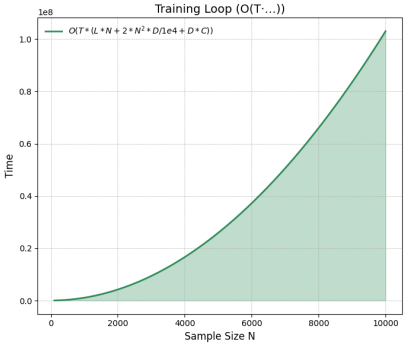

Figure 5.25. Training Loop ($O(T(LN + 2N^2D/10^4 + DC))$)

Preprocessing: $O(MN^3 + KEN + MN)$

Training Loop: $O(T(LN + 2N^2D/10^4 + DC))$

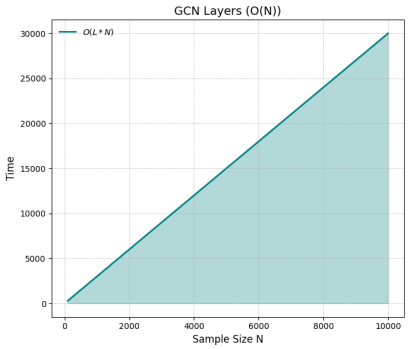

Figure 5.26. GCN Layers ($O(LN)$)

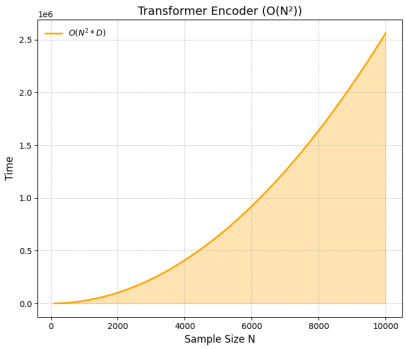

Figure 5.27. Transformer Encoder ($O(N^2D)$)

GCN Layers: $O(LN)$

Transformer Encoder: $O(N^2D)$

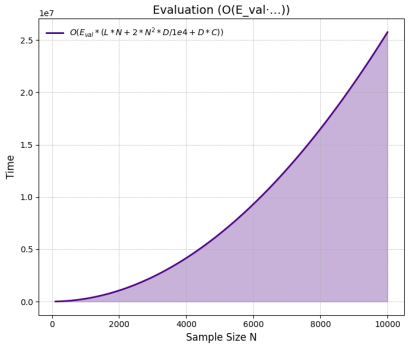

Figure 5.28. Evaluation Loop ($O(E_{\text{val}}(LN + 2N^2D/10^4 + DC))$)

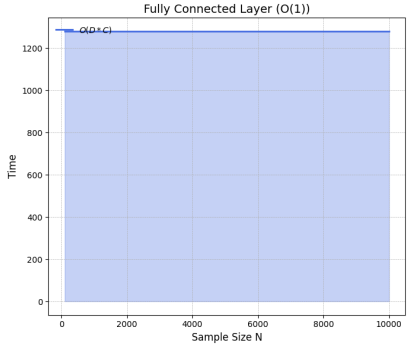

Figure 5.29. Fully Connected Layer ($O(DC)$)

Evaluation Loop: $O(E_{\text{val}}(LN + 2N^2D/10^4 + DC))$

Fully Connected Layer: $O(DC)$

Chapter 6

Discussion

6.0.1 Principal Contributions

This study proposes a novel graph-based representation for sleep EEG data, integrating a hybrid GCN-Transformer architecture to improve the classification of sleep stages. By effectively capturing both spatial and temporal dependencies, our model outperforms traditional deep learning approaches in terms of accuracy and robustness. The incorporation of graph structures, combined with attention mechanisms, facilitates a more nuanced understanding of sleep dynamics.

6.0.2 Distinctive Aspects of Our Approach

Conventional methods such as CNNs primarily emphasize spatial features, whereas RNNs address temporal dynamics but are limited in handling long-range dependencies. Our model bridges this gap by combining the spatial modeling capabilities of Graph Convolutional Networks (GCNs) with the sequence modeling strengths of Transformers. Moreover, the use of Focal Loss and Label Smoothing

addresses class imbalance effectively. The explainability component of our framework, supported by XAI techniques, identifies key EEG channels that influence sleep stage transitions, enhancing interpretability and clinical relevance.

6.0.3 Performance Highlights

Several components contribute to the model's success. The graph-based representation enables the retention of spatial relationships across EEG nodes, promoting more effective learning of brain activity patterns. The fusion of GCN and Transformer modules allows the network to model complex spatial-temporal interactions, yielding highly accurate sleep stage predictions. Techniques for mitigating class imbalance—such as Focal Loss and Label Smoothing—enhance reliability, especially in underrepresented stages. Additionally, explainability features provide actionable insights into the neural correlates of sleep, supporting broader medical and neuroscientific applications.

6.0.4 Limitations and Future Directions

Despite its strengths, the proposed method has notable limitations. The combined GCN-Transformer model incurs higher computational costs compared to lighter architectures like CNNs or LSTMs, which may restrict deployment in resource-constrained environments. The Sleep-EDF dataset, although widely used, includes a limited sample of subjects, potentially affecting generalizability. Furthermore, while acceleration via Apple M1 and Metal APIs improves efficiency, larger models would benefit from more powerful hardware such as GPUs or TPUs.

Chapter 7

Conclusion

The proposed SleepGCN-Transformer architecture demonstrates superior performance in sleep stage classification, achieving 93.12% training accuracy and 93.04% validation accuracy. By integrating GCNs to model spatial interdependencies and Transformers to extract temporal patterns, the model effectively captures the complexity of sleep physiology. The application of Focal Loss contributes to robust handling of class imbalance, improving classification across all stages. Feature importance analysis indicates that EMG and the EEG Pz-Oz channel play a critical role in prediction. This approach not only improves predictive performance but also enhances interpretability, laying a foundation for future work in explainable and clinically applicable sleep analysis tools.

Chapter 8

Appendix: Resources and Dataset Information

Figure 8.1. Automated Sleep Staging System with EEG Signal using Machine Learning Techniques

Figure 8.2. Deep Neural Model for Automated Sleep Staging System using Single-Channel EEG Signal

Figure 8.3. SleepGCN-Transformer: A Hybrid Graph-Convolutional and Transformer Network for Sleep Stage Classification

8.1 Dataset

We conducted our experiments using the **Sleep-EDF dataset**, a publicly available dataset for sleep analysis research.

`https://physionet.org/content/sleep-edfx/1.0.0/`

8.2 Development Environment

Our implementation was carried out using the following configurations:

- **Programming Language**: Python 3.13.1
- **Environment Manager**: pip (with virtual environment)
- **Deep Learning Frameworks**: PyTorch, TensorFlow
- **IDE**: VS Code, Jupyter Notebook (.ipynb)
- **Hardware**: Apple M1 Chip (CPU and Metal for acceleration)

8.3 Version Control (.GIT)

To manage our codebase, we used GitHub for version control.

QR code available alongside the figures.

8.4 License

The source code of this research is registered under the **Apache License 2.0**.

`http://www.apache.org/licenses/`

8.5 Key Python Libraries & Versions

- **NumPy**: 2.2.2
- **Pandas**: 2.2.3
- **Scikit-learn**: 1.6.1
- **SciPy**: 1.15.1
- **Matplotlib**: 3.10.0
- **Seaborn**: 0.13.2
- **PyTorch**: 2.6.0
- **Torch Geometric**: 2.6.1
- **NetworkX**: 3.4.2
- **MNE**: 1.9.0

This setup ensures a robust and reproducible environment for the development and evaluation of our sleep stage classification model.

Chapter 9

References

Bibliography

[1] H. Han et al., “Classification and automatic scoring of arousal intensity during sleep stages using machine learning,” *Scientific Reports*, vol. 14, no. 1, Dec. 2024, doi: 10.1038/s41598-023-50653-9.

[2] O. Tsinalis et al., “Automatic Sleep Stage Scoring Using Time-Frequency Analysis and Stacked Sparse Autoencoders,” *Annals of Biomedical Engineering*, vol. 44, no. 5, May 2016, doi: 10.1007/s10439-015-1444-y.

[3] A. Krakovská and K. Mezeiová, “Automatic sleep scoring: A search for an optimal combination of measures,” *Artificial Intelligence in Medicine*, vol. 53, no. 1, Sep. 2011, doi: 10.1016/j.artmed.2011.06.004.

[4] S. Biswal et al., “Expert-level sleep scoring with deep neural networks,” *Journal of the American Medical Informatics Association*, vol. 25, no. 12, Dec. 2018, doi: 10.1093/jamia/ocy131.

[5] M. Ronzhina et al., “Sleep scoring using artificial neural networks,” *Sleep Medicine Reviews*, vol. 16, no. 3, Jun. 2012, doi: 10.1016/j.smrv.2011.06.003.

[6] H. Alsolai et al., “A Systematic Review of Literature on Automated Sleep

Scoring," *IEEE Access*, vol. 10, pp. 79419–79443, 2022, doi: 10.1109/ACCESS.2022.3194145.

[7] L. Fiorillo et al., "Automated sleep scoring: A review of the latest approaches," *Sleep Medicine Reviews*, vol. 48, Dec. 2019, doi: 10.1016/j.smrv.2019.07.007.

[8] O. Faust et al., "A review of automated sleep stage scoring based on physiological signals for the new millennia," *Computer Methods and Programs in Biomedicine*, vol. 176, Jul. 2019, doi: 10.1016/j.cmpb.2019.04.032.

[9] P. Chriskos et al., "A review on current trends in automatic sleep staging through bio-signal recordings and future challenges," *Sleep Medicine Reviews*, vol. 55, Feb. 2021, doi: 10.1016/j.smrv.2020.101377.

[10] H. Phan, K. B. Mikkelsen, O. Y. Chen, P. Koch, A. Mertins, and M. De Vos, "SleepTransformer: Automatic Sleep Staging with Interpretability and Uncertainty Quantification," *IEEE Transactions on Biomedical Engineering*, 2022.

[11] H. Han et al., "Classification and automatic scoring of arousal intensity during sleep stages using machine learning," *Scientific Reports*, vol. 14, no. 1, Dec. 2024.

[12] O. Tsinalis et al., "Automatic Sleep Stage Scoring Using Time-Frequency Analysis and Stacked Sparse Autoencoders," *Annals of Biomedical Engineering*, vol. 44, no. 5, May 2016.

[13] A. Krakovská and K. Mezeiová, "Automatic sleep scoring: A search for

an optimal combination of measures," *Artificial Intelligence in Medicine*, vol. 53, no. 1, Sep. 2011.

[14] S. Biswal et al., "Expert-level sleep scoring with deep neural networks," *JAMIA*, vol. 25, no. 12, Dec. 2018.

[15] M. Ronzhina et al., "Sleep scoring using artificial neural networks," *Sleep Medicine Reviews*, vol. 16, no. 3, Jun. 2012.

[16] H. Alsolai et al., "A Systematic Review of Literature on Automated Sleep Scoring," *IEEE Access*, vol. 10, 2022.

[17] L. Fiorillo et al., "Automated sleep scoring: A review of the latest approaches," *Sleep Medicine Reviews*, vol. 48, Dec. 2019.

[18] O. Faust et al., "A review of automated sleep stage scoring based on physiological signals for the new millennia," *Computer Methods and Programs in Biomedicine*, vol. 176, Jul. 2019.

[19] P. Chriskos et al., "A review on current trends in automatic sleep staging through bio-signal recordings and future challenges," *Sleep Medicine Reviews*, vol. 55, Feb. 2021.

[20] H. Phan et al., "SleepTransformer: Automatic Sleep Staging with Interpretability and Uncertainty Quantification," *IEEE Transactions on Biomedical Engineering*, 2022.

[21] Y. Dai et al., "MultiChannelSleepNet: A Transformer-Based Model for Automatic Sleep Stage Classification With PSG," *IEEE J. Biomed. Health Inform.*, 2023.

[22] Y. Guo, M. Nowakowski, and W. Dai, "FlexSleepTransformer: A Transformer-Based Sleep Staging Model with Flexible Input Channel Configurations," *Scientific Reports*, 2024.

[23] Z. Yao and X. Liu, "A CNN-Transformer Deep Learning Model for Real-time Sleep Stage Classification in an Energy-Constrained Wireless Device," *medRxiv*, 2022.

[24] A. Supratak et al., "DeepSleepNet: A Model for Automatic Sleep Stage Scoring Based on Raw Single-Channel EEG," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 25, no. 11, pp. 1998–2008, 2017.

[25] E. Eldele et al., "AttnSleep: An Attention-Based Deep Learning Approach for Sleep Stage Classification With Single-Channel EEG," *IEEE Trans. Neural Syst. Rehabil. Eng.*, 2021.

[26] X. Ji, Y. Li, and P. Wen, "Jumping Knowledge-Based Spatial-Temporal Graph Convolutional Networks for Automatic Sleep Stage Classification," *IEEE Trans. Neural Syst. Rehabil. Eng.*, 2022.

[27] S. Mousavi, F. Afghah, and U. R. Acharya, "SleepEEGNet: Automated Sleep Stage Scoring With Sequence-to-Sequence Deep Learning Approach," *PLOS ONE*, 2019.

[28] H. Phan et al., "SeqSleepNet: End-to-End Hierarchical Recurrent Neural Network for Sequence-to-Sequence Automatic Sleep Staging," *IEEE Trans. Neural Syst. Rehabil. Eng.*, 2019.

[29] Zhang et al., “A CNN-Transformer-ConvLSTM-CRF Hybrid Network for Sleep Stage Classification,” *IEEE Sensors Journal*, 2024.

[30] Liu et al., “Automatic Sleep Stage Classification Using Deep Learning: Signals, Data Representation, and Neural Networks,” *Artificial Intelligence Review*, 2024.

[31] CT-DAL, “Convolutional Transformer with Domain Adversarial Learning for Multi-Channel Sleep Stage Classification,” *Available on Semantic Scholar*, 2022.

[32] W. Pei, Y. Li, S. Siuly, and P. Wen, “A Hybrid Deep Learning Scheme for Multi-Channel Sleep Stage Classification,” *Computers, Materials & Continua*, 2022.

[33] S. H. Mostafaei, J. Tanha, and A. Sharafkhaneh, “A Novel Deep Learning Model Based on Transformer and Cross-Modality Attention for Classification of Sleep Stages,” *Journal of Biomedical Informatics*, 2024.

[34] Jeong et al., “Explainable Vision Transformer for Automatic Visual Sleep Staging,” *npj Digital Medicine*, 2024.

[35] W. Pei, Y. Li et al., “An Automatic Method Using MFCC Features for Sleep Stage Classification,” *Biomedical Signal Processing and Control*, 2024.

[36] A. Sors et al., “A Convolutional Neural Network for Sleep Stage Scoring from Raw Single-Channel EEG,” *Biomedical Signal Processing and Control*, 2018.

[37] H. Seo et al., "Intra- and Inter-Epoch Temporal Context Network (IITNet) Using Sub-Epoch Features for Automatic Sleep Scoring on Raw Single-Channel EEG," *Biomedical Signal Processing and Control*, 2020.

[38] S. Biswal et al., "SLEEPNET: Automated Sleep Staging System via Deep Learning," *Journal of the American Medical Informatics Association*, 2017.

[39] H. Phan et al., "Joint Classification and Prediction CNN Framework for Automatic Sleep Stage Classification," *IEEE Transactions on Biomedical Engineering*, 2018.