ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Sleep Stage Classification

Master Thesis

Ambarish Sridhar Prakash

November 20, 2023

Supervisor: Prof. Dr. Joachim M. Buhmann
Advisor: Ami Beuret
Department of Computer Science, ETH Zürich

Abstract

Classification of brain signals into sleep stages is a tedious and time consuming manual process that has shown great potential to be automated by machine learning models. In 2019, Sleep Phase Identification with Neural networks for Domain-invariant LEarning aka Spindle, was introduced by members of the ETH ISE Lab to automatically classify sleep stages in Mice. Using Convolutional Neural Networks, Spindle pushed the boundaries of automated sleep stage classification, and is still actively used by Sleep Scientists today. In the time since, the world of Machine Learning, specifically Computer Vision, has expanded with the introduction of new architectures such as Transformers and Vision Transformers.

In this thesis, we explore these different architectures and learning techniques and propose a new model Swin Transformer for sleep Classification, aka SwinTsle. SwinTsle is backed by a Swin Transformer for feature extraction as well as secondary Transformer layer for learning temporal information between Sleep Stage Epochs, and is one of the first proposed architectures to use Vision Transformers for the classification of Sleep Stages in Mice. SwinTsle is also trained using data augmentation techniques that reduce over fitting and help improve generalization. With these improvements, SwinTsle is able to achieve a very high prediction accuracy of 96.03%, beating Spindle's performance of 94.67% by nearly 1.5% when tested on the Spindle data set. When tested on a human data set, SwinTsle outperforms Spindle's prediction accuracy by nearly 10%, achieving a prediction accuracy comparable to state-of-the-art models specifically built for sleep stage classification in humans. SwinTsle shows the potential of Vision Transformers and Attention for the sleep stage classification problem.

Contents

1. Introduction

Sleep is an activity found in a majority of animals and is a vital component of good health [1]. Humans spend approximately one third of their life sleeping, and hence the tracking of sleep stages is important to understand the variations in altered sleep which can subsequently shed more light on sleep based disorders such as Sleep Apnea and Narcolepsy [2].

During sleep the brain is alive with neuronal activity accompanied by other changes in the body such as changes in the breathing and heart rate along with a reduction in motor activity. Based on these factors, we can identify at least three major states. First is the state of wakefulness, characterized by large motor activity as well as high frequency beta brain waves with a low amplitude. The second state, known as Non-rapid Eye Movement Sleep (or NREM) is characterized by high amplitude, low frequency, theta and delta brain waves along with low or no motor activity. The third state, also known as Rapid Eye Movement Sleep (or REM). REM sleep is associated with dreaming and is characterized with low motor activity, but more pronounced brain activity than NREM sleep. In humans, NREM sleep is generally additionally classified into 3 stages - N1, N2 and N3, based on characteristic brain wave patterns. [3, Chapter 2: Sleep Physiology]

While there are many tracking methods available to detect the sleep stages such as accelerometers to measure movement or Photoplethysmography (PPG) devices to measure heart rate [4], Polysomnography which involves directly tracking the brain waves by using an Electroencephalography (EEG) to measure the brain activity and Electromyography (EMG) to measure the motor activity remains the gold standard for assessing sleep [5]. Sleep Stage classification for research or clinical studies is hence performed on EEG signals usually along with an EMG signal. In mice the signals are usually divided into 1-8s long epochs and are classified into the different stages per epoch. In humans however, the signals are generally segmented into 30s long epochs for classification.

The classification is generally done manually by trained experts. Not only is the manual visual inspection time consuming, ambiguous and prone to errors but also requires training of specialists to be able to identify the states. These difficulties paved the way for automatic sleep stage classification.

Multiple approaches have been devised for the automatic classification of sleep states. Most of these initial attempts were based on manual feature

extraction. These features, acquired by using existing domain knowledge, usually consisted of energies from the standard frequency bands of the EEG signal such as delta δ(0.1–4Hz), theta θ(6–9Hz), sigma σ(10–15Hz) and beta β(15–30Hz) as well as combinations of these values such as the ratio of θ to δ. These features vectors for each epoch were then classified into the different states based on either algorithms based on fixed thresholds [6, 7] or used machine learning models such as Support Vector Machines or Bayesian Classifiers [8, 9, 10, 11]. These ML models were trained in either a supervised [8, 9] or unsupervised [10, 11] fashion to classify the epochs.

Following the success of Deep Neural Networks(DNNs) in other domains such as image analysis and audio recognition [12, 13], DNNs have been put to use for this particular task as well. The results have been highly successful with many state-of-the-art models achieving around a 95% accuracy on the sleep stage classification for mice. Particularly noteworthy are Convolutional Neural Networks(CNNs) [14] that are used to extract time invariant features either from the signals directly or from their time frequency transformations. In addition to feature extraction, most State-Of-The-Art DNNs also have a layer to capture sequential information of sleep state transitions, either via Hidden Markov Models (HMMs) [15], Long Short Term Memory Networks(LSTMs)[16] or Attention Networks[17].

Similar architectures are also found in models built to classify sleep stages in humans, with changes such as increase in layer count or addition of skip connections, incorporated to handle the higher complexity, longer time per epoch and larger number of sleep states [18, 19, 20].

Along with the advent of attention, came a new architecture for sequential information processing called the Transformer [21]. Transformers revolutionized the natural language processing landscape due to their ability to parallelize the training of sequential information. They are bases for a lot of large language models (LLMs) such as BERT and GPT [22]. This concept was then applied to the processing of images, leading to the introduction of Vision Transformers (ViTs) [23]. ViTs have one of the highest performances of image classification tasks when compared against the standard benchmarks sets such as ImageNet. Many different versions of the Vision Transformer have also been developed such as the Swin Transformer [24] and the Compact Convolutional Transformer [25] building on various aspects of the original design. In addition existing convolution networks have also been redesigned to incorporate the transformer either directly or indirectly. Most notable state-of-the-art models include ConvNeXt [26] and MobileFormer [27].

In this thesis we look to explore the performance of these new models on the sleep stage classification problem, specifically on mice. First we start by trying to compare Spindle with some of the other state-of-the-art models, facing issues with code reproduction. We instead form a study and explore the

effects of the different proposed sequential processing layers in these models. We then evaluate the performance of new architectures such as ViTs for this task. We also explore the effects of different training parameters such as the loss function, number of neighbours as well as layers to process sequential information. Using these we propose a new architecture - SwinTsle that is able to outperform our local implementation of Spindle. Finally we test the performance of the Spindle and SwinTsle on human data. The contributions are detailed as follows:

- Test the improvements provided by different sequential processing layers such as Bi-directional LSTMs and Transformers.
- Evaluate the performance of various Vision Transformers on this task of Sleep Stage classification on rodent data, as well as pre trained versions of Image classification models.
- Explore effect of different optimizations to address issues such as over fitting and low mean F1 class scores due to class imbalanced data sets. Notable we evaluate the effect different data augmentation techniques and loss functions.
- Combining the results, we propose a new architecture, SwinTsle, that is able to outperform Spindle's accuracy by nearly 1.5% when tested on the Spindle data set. We also introduce a new data set where SwinTsle is again able to outperform Spindle.
- We also test the performance of these models when applied to the classification of human data. We show that SwinTsle is able to learn these features also, achieving a prediction accuracy comparable to other state-of-the-art models.

2. Related Work

Various methods for autonomous sleep stage classification have been devised and introduced over the last decade. As mentioned in the introduction, many of the early innovations involved manually extracted features with different classifiers. However DNNs trained for these tasks have shown the best performance till date. In this section we will have a deeper look at some of the state of the art models for sleep stage classification at present.

2.1 Sleep Stage Classification on Mice

This thesis was based on Spindle [15] , the work done in the lab by a former student Đorđe Miladinović as well as my supervisor Ami Beuret. As such this thesis focuses more on the sleep stage classification of EEG signals from mice using the same dataset made publicly available by the lab.

In mice, sleep stages are generally classified into 3 different categories - Wake, NREM and REM sleep based on epochs that range between 1-8 seconds. Some papers also classify the input data based on the level of noise present into a category known as artifacts. Artifacts are hence input sections that exhibit outliers in the trends due to the noise.

2.1.1 Spindle

Sleep Phase Identification with Neural networks for Domain-invariant LEarning [15], aka Spindle, was introduced by Đorđe Miladinović et al in order to introduce a classifier that could generalize well across different setups such as genetics mutations among mice, recording setups for signals or even different species such as rats. Spindle was inspired by innovations in ASR (Automatic Speech Recognition) tasks and uses a convolutional network to extract features from preprocessed inputs from the signal.

Spindle uses a set of pre processing steps to convert the EEG and EMG signals from their 1D time series representation to their normalized 2D frequency time representation. Spindle then uses a Convolutional Neural Network to extract translation invariant features from this frequency time representation upon which a linear classifier is used.

Spindle also introduces the use of a Hidden Markov Model (HMM) to model the state transitions between consecutive epochs. Using prior knowledge, Spindle uses an HMM with state transition probabilities that prohibit the transition between certain stages such as from moving directly from Wake to REM. This allows the model to learn about such transitions and give more plausible and hence more accurate results.

Spindle introduces a public dataset, hereby referred to as the Spindle dataset, consisting of readings from 24 subjects from four different labs. After training on a subset of 2 of the mice, Spindle boasts an accuracy of 99%, 98%, 93% and 97% on the remaining sets of subjects when classifying the signals into the 3 sleep stages. When artifact prediction is included, Spindle is still able to keep the accuracy close to 90%.

Spindle is also available as a free to use model online and can be found at https://sleeplearning.ethz.ch/.

2.1.2 CSSleep

CSSleep [17], a Cross Species rodent Sleep scoring network builds on the concepts of Spindle. CSSleep uses the same dataset and preprocessing used by Spindle, as well as a Convolutional Neural Network to extract time invariant features from the preprocessed signal.

Where CSSleep differs is first in the convolution layers. CSSleep employs two convolutional layers with different kernel sizes in order to extract features from various sleep related frequency bands from the signal.

On top of this, CSSleep also employs a bidirectional layer to capture the global transition rules similar to the function of the HMM in Spindle. Each block in this layer consists of a multi headed masked attention layer followed by a temporal causal convolution network and finally mixed with an attention weighted residual of the original features. Both layers help integrate past information into the current epoch via a mask as well as the use of causal convolution respectively. To capture future information a similar layer is applied on the reverse signal to make it bi-directional.

The CSSleep paper uses the same dataset as Spindle. It also uses a LOCO-CV (Leave One Cohort Out - Cross Validation) method for training where they train on 3 of the cohorts and test on the 4th. With this, CSSleep achieves an accuracy of approximately 95%, 93%, 88% and 92% on the 4 different cohorts. While this seems lower than Spindle, it cannot be compared directly as it uses all the input data instead of just the epochs where both experts intersect.

2.1.3 MC-Sleep Net

Another high performing model is MC-Sleep Net. Similar to the other two, MC-Sleep Net uses two convolutional layers for feature extraction, with a key difference being that it applies 1D convolution directly on the signal without any preprocessing. For the scoring part, MC-Sleep Net uses a linear layer to identify the target epoch, along with a bi directional Long Short Term Memory (LSTM) layer to identify state transitions.

The authors of MC-Sleep Net show that it is able to achieve a 96.6% accuracy when applied via a 5-fold cross validation cycle on their own large-scale dataset consisting of 4,200 records.

2.2 Sleep Stage Classification on Humans

Sleep Stage classification in Humans follows a similar pattern. Matched EEG and EMG signals are broken down into epochs that are usually 30s long. Moreover the NREM sleep stage is further classified into four different types - N1, N2, N3 and N4. Classification tasks generally combine N3 and N4 stages together due to their similarity and low occurrence leading to models that predict one of five different sleep stages per epoch.

While this thesis mostly focuses on the model performance on Mice datasets, it is interesting to view the model architectures used in classification of human datasets as similar principles and ideas apply to both scenarios.

2.2.1 SleepViTransformer

Published in September 2023, SleepViTransformer [28] is one of the first architectures to use a Vision Transformer based model. SleepViTransformer uses a patch style processing of the 2d spectrogram, akin to a Vision Transformer, in order to extract features from the input signals. They also use a bi-directional GRU layer in order to process temporal information between epochs. In addition they also use some augmentation methods during training to help the model against over fitting.

In terms of performance, the authors show SleepViTransformer is able to beat the overall accuracy of other state-of-the-art models when tested on the SleepEDF (20 and 78), Physio-2018 and SHHS datasets. However considering they use two EEG signals as well as 2 additional neighbours per epoch, the direct comparison may not be valid.

2.2.2 SeqSleepNet and SleepTransformer

SeqSleepNet [29] and SleepTransformer [30] have similar overarching heirarchical architectures, but use different components for the layers. Both models extract features per epoch by focusing on intra-temporal connections of the signal within the epoch. They then use a different sequential processing layer to identify the inter epoch relations between different epochs. SeqSleepNet published in 2019 uses recurrent neural networks (RNNs) to extract the features as well as the relations between epochs, while SleepTransformer uses the encoder section of the transformer to handle both the intra and inter epoch dependencies.

Both models achieve high accuracies when tested on the public datasets. SeqSleepNet is able to achieve an overall accuracy of 86% and 83.8% on the SleepEDF 20 and 78 datasets respectively, while SleepTransformer is able to achieve overall accuracies of 84.9% and 87.7% when tested on the SleepEDF78 and SHHS datasets.

2.2.3 AttnSleep

Looking at AttnSleep [18] we are able to see some similar components as in the previous models as well as some new components. AttnSleep uses a multi resolution CNN consisting of two CNNs with different kernel sizes to extract information directly from the signal. To score the features, they employ a multi-headed attention layer with the embeddings formed by causal convolutions in order to capture temporal dependencies in each feature.

AttnSleep also uses only 1 EEG signal as an input. When tested on public datasets - Sleep-EDF-20, Sleep-EEDF-78 and SHHS, AttnSleep is able to achieve an overall accuracy of 84.4%, 81.3% and 84.2% respectively when using an 80-20 split of the data.

3. Data Sets and Metrics

Three different datasets were used in experiments performed for this thesis. Two of them were mice datasets, while one was a human dataset. In this section, we go through the three different datasets in more detail as well as the preprocessing performed.

3.1 Preprocessing

For all three datasets, the same preprocessing procedure was used. This preprocessing is the same one performed by Spindle [15]. The EEG and EMG signals are first re sampled to match a 128Hz frequency in order to handle differences in various datasets. These signals are then converted to the time frequency domain via the application of short fourier transformers on overlapping 256 length (2 second) windows with a stride of 16. Hamming windows were used when performing the fourier transforms for smoother edges. These signals were then band pass filtered between 0 and 24Hz to ignore the higher frequency values as they are not as characteristic of the different stages as the lower frequencies. The EEG signals spectrograms are then log scaled and normalized per frequency. We use the EMG signal to measure motor activity of the subject, and this is measure by the total energy across all the frequencies for the given time point. Hence the EMG spectrogram is summed up row-wise to get the muscle activity of the subject over time. To ensure this shares a 2D structure similar to the EEG spectrograms, the EMG signal is replicated to match the 2nd dimension of the EEG spectrogram input. The signal was then split into epochs corresponding to 4 seconds of the original signal. To incorporate neighbouring information when predicting the epoch, 4 neighbours were added to each input additionally. Hence each input x_i consists of the spectrograms of signals from x_{i-2} to x_{i+2}.

3.2 Spindle Data Set

This data set comprising of 22 animals from 3 independent labs was made publicly available by the ISE group at ETH when publishing the Spindle [15] paper.

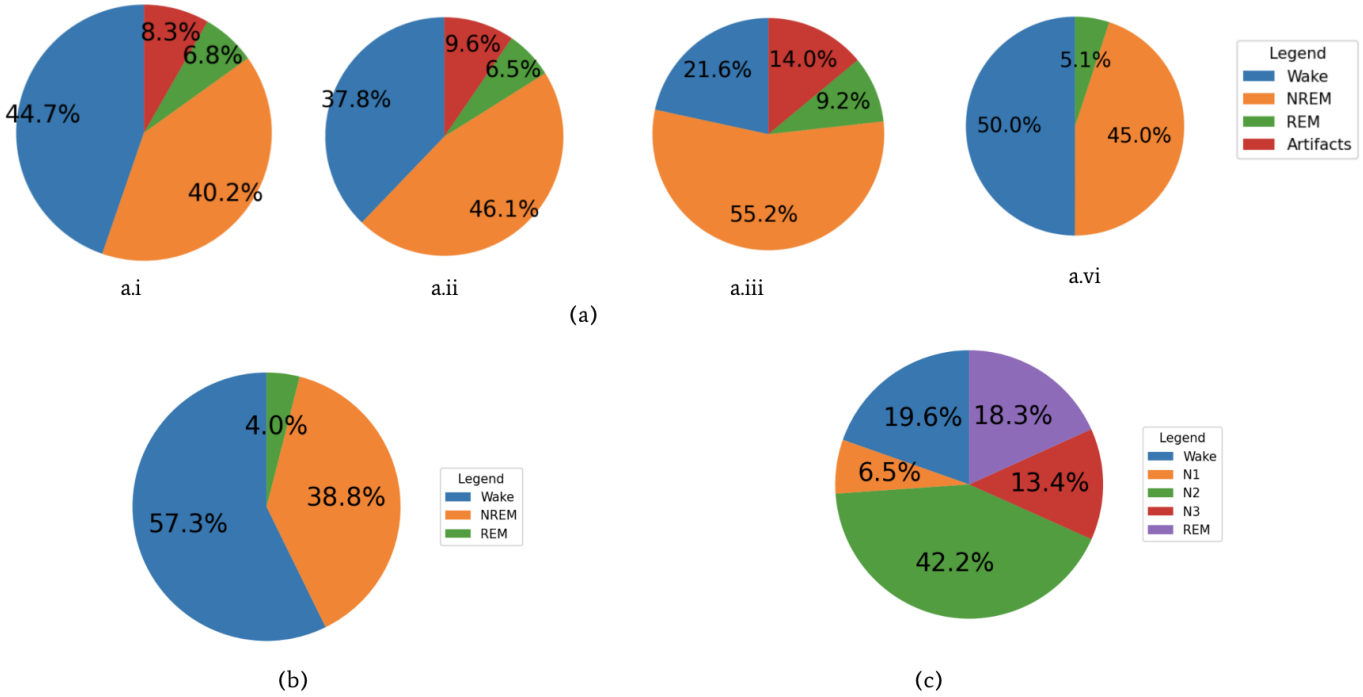

Figure 3.1: Data Distributions over the three different data sets. (a) Distributions of the 4 different cohorts A, B, C and D in the Spindle data set shown from left to right. (b) Distributions of the Slumber data set. (c) Distributions of the Sleep-20 EDF data set after removing the excess wake epochs 30 minutes before and after sleep.

The animals were part of 4 different cohorts. Two cohorts - Cohort A and Cohort B contain readings for 4 wildtype mice each, with cohort B containing mice with a genetic mutation. Cohort C contains 8 rats and hence have signals that differ slightly from the rest, while Cohort D contains recordings from 6 wildtype mice. For all of the recordings, the signals are broken into 4s epochs and each epoch has been labelled by two different experts leading to epochs with multiple labels. Each epoch was classified into 6 different categories, namely W, NREM and REM, as well as artifacts (epochs that had high noise) of W, NREM and REM.

The distributions of the different classes in the 4 cohorts can be seen in Figure 3.1. Artifacts have been grouped together here as their further distributions into sub classes causes some sub classes to be close to 0%. We can observe that the distributions of REM classes and Artifacts is very small in all 4 cohorts showing the data is imbalanced. Cohort C, shown in Figure 3.1.a.iii, which is made up of rats is also quite different distributed as compared to the other cohorts, with a lot more NREM class instances than the other classes.

More details into the data set such as number of subjects, the collection

methods as well as data availability can be found on the Spindle paper.

3.3 Slumber Data Set

This dataset was collected and introduced in the paper SlumberNet [31] released by the University of Pennsylvania. The dataset contains a large set of EEG and EMG signal data (corresponding to 635526 epochs), but unfortunately does not specify the number of mice or the ordering. The hope is the data is sequential (used by models that identify the relation between epochs) but that is not confirm able. Each epoch is labelled between one of the three states - W, NREM and REM. The data set also does not contain any artifacts as they have been preprocessed and filtered out before making the data set available.

The data distribution can be seen in Figure 3.1. This distribution is similar to the mice cohorts (A,B and D) in the Spindle data set.

More details on the acquisition protocols as well as data availability can be found on the SlumberNet paper.

3.4 Sleep EDF 20 Data Set

Sleep EDF is a publicly available data set of Human Sleep metrics provided by PhysioBank [32]. Sleep EDF 20 consists of PSG recordings from 20 subjects while Sleep EDF 78 contains the readings from 78 subjects. The subjects in the PhysioBank sets participated in two studies a Sleep Cassette (SC) study and a Sleep Telemetry(ST) study. Following other papers, we use the Sleep Cassette recordings of the Sleep 20 EDF dataset, specifically the two EEG signals (Fpz-Cz/Pz-Oz EEGs) and the chin EMG signal. We also follow the cleaning steps followed in other papers, namely removing the epochs without any label as well as restricting the data set to only include the Wake epochs 30 minutes before and after sleep.

The distribution of the data can be found in Figure 3.1. We see that it is more balanced than the mice data sets, with sleep stage N2 most present and sleep stage N1 least present.

More details about the acquisition of the data can be found on the PhysioBank website. The data is accessible via the PhysioBank website. In this study we downloaded the files using the links provided in the AttnSleep code repository[1].

[1] https://github.com/emadeldeen24/AttnSleep/blob/main/prepare_datasets/download_edf20.sh

3.5 Metrics

Evaluation of the different data sets use some of the standard metrics. We mainly measure the Accuracy of all the predictions, as well as the per class metrics of Precision, Recall and F1-score for each of the different sleep stages.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Population}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Additionally we also calculate the Mean F1 (MF1) score of the 3 classes which can distinguish model performances especially in cases where there is a class imbalance. Finally we also calculate the Cohen's Kappa value between the model's predictions and the expert's predictions which computes the agreeability between the two values. Cohen's Kappa is calculated using the observed agreement (p_o) which is the proportion of agreement between the raters, and the expected agreement (p_e) which represents the agreement that would be expected by chance.

$$\text{Mean F1 Score} = \frac{1}{N} \sum_{i=1}^{N} \text{F1 Score}_i$$

$$\text{Cohen's Kappa} = \frac{p_o - p_e}{1 - p_e}$$

4. Comparative Study

With many models already designed to work on this task, we look at a few existing architectures to set up our baselines as well as compare their performance on our dataset.

4.1 Spindle

For this thesis we used the previous implementation of Spindle that runs on the Sleep Learning website as well. As mentioned in the related work section, Spindle uses a convolution and max pool layers in order to extract features and then uses a set of linear layers to classify the image into the required sleep states. Similar to the paper, we trained the model on the first two mice of Cohort A and tested it on the remaining subjects, using only the epochs that the two experts agreed upon. The results showing the mean and standard deviation over 5 runs is shown in Table 4.1. The mean values have been rounded down to integer values to match up to the results in the paper.

However with this code and the hyper parameters taken from the paper, we were unable to reproduce the exact performance stated in the paper. Instead as we can see in Table 4.1 we were able to achieve a very comparable performance that lies within the threshold for most of the metrics, and hence use this as our baseline for the rest of the report.

Additionally to ensure that Spindle was not overfitting to the data when comparing its performance to other papers, we also tested its performance on the Slumber Data Set without having seen any of the inputs in the training before. From the last row in Table 4.1 we see that Spindle is able to achieve a high accuracy on this new data set even when trained with just the first 2 mice from Cohort A of the Spindle data set.

4.2 CSSleep

As mentioned in the related work, while the CSSleep paper compares its performance with respect to Spindle, it does so only for the four class classification task where it shows that CSSleep was able to obtain better results when trained with the same parameters as mentioned in the Spindle

Table 4.1: Comparison of Spindle's performance from the paper as well as the implementation running on the Sleep Learning website. In both cases, the models were trained on the first two mice of Cohort A and tested on the remaining subjects, using only the agreement epochs (epochs where both experts shared the same label). The last line of the table, namely the Cohort called Slumber corresponds to the results of Spindle's performance on the Slumber Data set.

Cohort	Accuracy(%)	Wake			NREM			REM		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Spindle Paper Results:										
A	99±0.5	100±0.3	99±0.4	99±0.3	99±0.8	99±0.4	99±0.6	98±1.6	97±2.6	98±1.2
B	98±0.3	98±2.1	97±1.1	98±0.7	97±0.9	98±1.1	98±0.1	96±2.2	95±1.9	96±0.9
C	92±3.2	80±10.0	97±1.5	87±6.0	99±1.0	94±3.4	96±1.8	86±5.2	70±16.8	76±12.2
D	97±1.3	98±1.0	99±0.7	98±0.6	97±2.7	98±1.1	97±1.2	96±3.5	79±23.0	85±17.7
Spindle Recreation Results:										
A	99±0.07	100±0.05	99±0.1	99±0.05	99±0.13	99±0.13	99±0.09	96±0.48	98±0.12	97±0.25
B	97±0.22	95±0.48	99±0.11	97±0.2	99±0.12	94±0.59	96±0.25	94±0.89	96±0.32	95±0.37
C	91±0.25	83±0.61	97±0.37	90±0.24	99±0.19	92±0.49	95±0.21	86±0.97	66±1.96	75±1.21
D	96±0.17	95±0.37	99±0.06	97±0.17	98±0.1	94±0.43	96±0.18	91±0.93	86±0.46	88±0.23
Slumber	90±0.38	92±0.24	92±0.76	92±0.38	89±0.75	89±0.4	89±0.29	78±3.55	79±1.51	78±1.3

paper. The results for the 3 class classification task cannot be compared as directly as even though CSSleep uses the same dataset as Spindle, it uses only the label of Expert 2 and does not focus only on epochs where the labels of the two experts intersect. Not only that, CSSleep also performs a LOCO-CV (Leave one cohort out cross validation) method, hence training on 3 cohorts and testing on the 4th. Finally, the results in the paper are when CSSleep uses only a single EEG signal and not the EEG and EMG signals.

In order to perform a more balanced comparison and test Spindle's performance against newer architectures, we wanted to have a version of CSSleep to be able to run different experiments. However since CSSleep did not make the code available, we needed to re implement the model based on the details provided in the paper. This proved to be a challenge as all the model parameters were not clearly stated in the paper. Our implementation of the CSSleep model was able to score well achieving results within 1% of the results stated in the paper.

To compare the two methods together, we set the conditions to be the same. We would use CSSleep's Cross Validation and the labels from the 2nd Expert, but also use all three signals like Spindle.

First we note that while the CSSleep paper claims an overall accuracy of 91.3% on the entire data set, we see from Table 4.2 that our implementation achieves a 92.63% accuracy. This is expected as our implmentation of CSSleep uses 3 signals (both EEGs as well as the EMG signal) as compared to the results published in the paper which only used one EEG signal. Secondly

Table 4.2: Summarized performance of CSSleep Implementation on Spindle Data set when using the LOCO-CV method

CSSleep		Predicted Label			Per-class metric (%)		
		Wake	NREM	REM	Precision	Recall	F1-Score
True Label	Wake	**213868**	9744	7944	96.54	92.36	94.41
	NREM	7262	**195188**	7293	94.18	93.06	93.62
	REM	394	2318	**30089**	66.38	91.73	77.03

Overall: Accuracy: 92.63% ; MF1: 88.35% ; Kappa: 87.11%

Table 4.3: Summarized performance of Spindle Implementation on Spindle Data set when using the LOCO-CV method

Spindle		Predicted Label			Per-class metric (%)		
		Wake	NREM	REM	Precision	Recall	F1-Score
True Label	Wake	**223593**	6438	1525	92.31	96.56	94.39
	NREM	15306	**191343**	3094	95.55	91.23	93.34
	REM	3319	2465	**27017**	85.4	82.37	**83.86**

Overall: Accuracy: 93.22% ; MF1: 90.53% ; Kappa: 87.87%

we see that overall Spindle is able to perform better here, with an overall accuracy of 93.22%. This can be mainly attributed to its better performance in the classification of REM stages achieving an F1-score of 83.86% for REM stage classification to CSSleep's 77.03%.

Given the general positive performance of Spindle we use it as a baseline for our experiments.

4.3 Sequential Processing Study

We encountered a similar issue of lack of existing code when comparing Spindle against another model - MC-SleepNet which used a bi-directional LSTM to learn the state transitions between epochs.

Instead we looked to compare the different sequential architectures by creating a study to remove additional effects and only test the additional performance gain by using these particular architectures. With this we could compare the performance of the different layers such as the HMM used by Spindle, the top attention layer of CSSleep and the bi-directional layer of MC-SleepNet.

4.3.1 Setup

Since we were testing the sequential information layers, we decided to use the method employed by CSSleep for the dataset creation. We used only the labels produced by the 2nd Expert for the Spindle dataset so as to not have gaps in the input sequence due to disagreements between the two experts. We also re-labeled all the artifacts to their corresponding sleep stage (eg re labelling a Wake Artifact to Wake).

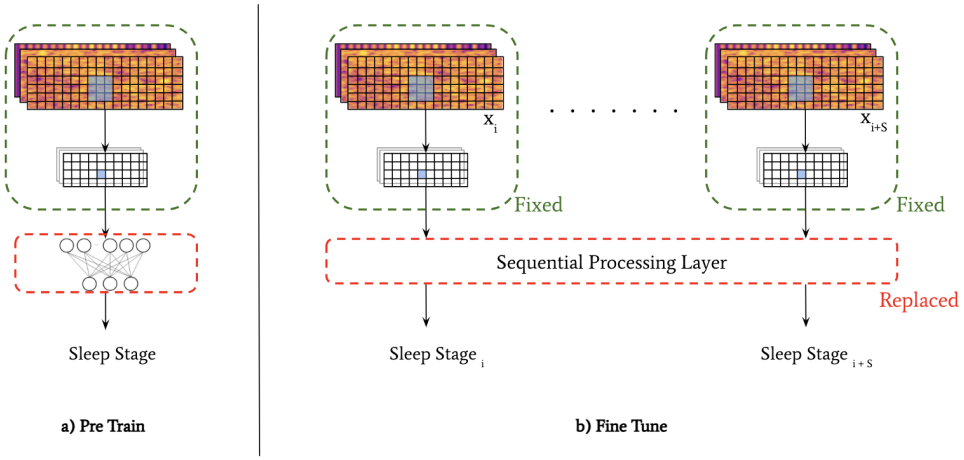

Figure 4.1: Sequential Processing Comparative study process. Spindle was first pre trained on the data set. The final linear layer was then replaced with a sequential processing layer that combined features from multiple consequent epochs to then classify epochs into sleep stages.

We used Spindle's initial convolution layer for the feature extraction. We first pre-trained the Spindle Convolution layer with a Fully Connected layer and saved the learned model weights of the Convolution layer. We then incorporated different layers on top of the pre trained convolutional layer. In our study, we tried the following layers:

- **Baseline:**

  This was the output generated by the linear layer directly from the convolutional features, as described in the Spindle paper. This has no sequential information processing layer and each epoch is predicted independently. We set the intermediate feature size to 1024, hence each input was first passed through a convolution layer and then a linear layer to get a feature vector of size 1024, before being classified.

- **HMM:**

  The use of an HMM is described in Spindle. We used an HMM to predict the final sleep stages based on the sleep stage probabilities

generated by the network and a set of sleep state transition probabilities setup on prior knowledge. These transition probabilities disallow the movement of Wake directly to REM sleep as well as from REM to NREM. The HMM takes the predicted output probabilities and uses the Viterbi algorithm to give the most probable set of states based on these probabilities and the state transition matrix.

- **LSTMs:**

  Long short term memory networks are a type of Recurrent Neural Network (RNN). LSTMs are equipped with memory cells that can store and retrieve information over extended sequences, making them effective for processing sequential information. In this study we used a bi-directional LSTM each with two layers of an LSTM (where the output of the first model was the input for the second). The model was initialized with an hidden state and a cell state of 0s, and this state was reset before each epoch and before validation and testing.

- **Transformer:**

  Transformers [21] were introduced in 2019 along with the concept of self attention. They introduce the concept of positional encoding mixed with self attention in order to pay attention to neighbouring contexts. Here we used the encoder section of the transformer model to learn the temporal information first before passing it through a linear layer for classification. This included adding a positional encoding and using two encoder blocks consisting of multi head attention, layer norm and feed forward networks.

- **CSSleep Temporal Layers:**

  We used the temporal layer described in CSSleep in order to compare it's performance as well. This layer is made up of bidirectional attention and temporal convolutional blocks (ATCN) in order to capture both past and future information. Each ATCN block is made up of layer normalization, masked attention followed by stacks of causal convolutions instead of the linear feed forward layer. The masked attention and causal convolution help the model only concentrate on one direction. The normed input features along with an attention weighted set of input features are also passed through as residual information.

Each of these models were trained on two mice from Cohort A and tested on the remaining subjects. For the sequential layers, we used a batch size of 50 when training and/or evaluating the model.

Table 4.4: Comparison of different temporal information processing architectures. Each model was setup with a convolution layer with pretrained weights to extract feature information and fine tuned again on the 2 mice from Cohort A. The metrics here show the overall performance on all the remaining mice.

Layer	Acc(%)	MF1(%)	Kappa(%)	Wake			NREM			REM		
				P	R	F1	P	R	F1	P	R	F1
Linear (Baseline)	92.25	87.81	86.47	95.33	93.21	94.26	95.22	91.24	93.19	64.74	91.94	75.98
HMM	92.45	88.87	86.78	94.79	93.67	94.23	94.88	91.04	92.92	69.37	92.96	79.45
Bi-LSTM	92.97	89.88	87.58	94.5	94.45	**94.47**	94.6	91.88	93.22	75.51	89.6	81.95
Transformer	92.92	89.93	87.54	94.75	94.11	94.43	94.55	91.94	93.23	74.41	90.93	81.84
CSSleep	**93.22**	**90.78**	**88.02**	94.34	94.48	94.41	94.58	92.25	**93.4**	79.14	90.74	**84.54**

4.3.2 Results

The predictions from all 4 cohorts were combined to receive the overall metrics and have been tabulated in 4.4. What we noticed was that the sequential information layers improved the overall prediction accuracy of the model. While these models do have an improvement, the overall accuracy increase is minimal for most of them. However if we look at the per class metrics, we see that the effect is much more apparent. The precision and recall of the REM class is much higher than before as more epochs are learnt to correctly being classified as REM rather than Wake. This is visible via an increase in the overall Mean F1 score.

Overall we see that the attention along with causal convolutions presented in the CSSleep paper has the highest overall accuracy across all cohorts with a nearly 1% improvement on the accuracy as compared to the baseline. Three key attributes set apart the CSSleep architecture. The first is a masked attention that along with a causal convolution instead of feed forward linear layers, which let data flow only in one direction. The attention layers allow points to be influenced by all points behind it directly, unlike the LSTMs which carry information forward through each input. The second is the use of bidirectional layers similar to the bi directional LSTM. And finally the use of a residual along with an attention weighted residual which enhances features of an epoch extracted by the convolution layer.

Comparing this to the HMMs used by Spindle, we see that DNNs are capable of learning transition rules on their own without the prior domain knowledge showing their strengths. We can also look at the transition epochs, i.e. epochs that have a label different to the previous or next epoch. These are usually a source of contention and harder to classify as they may contain features from both sleep stages if the transition is happening during the epoch. In Figure 4.2 we can see the classification accuracies of the different layers with respect just to these transitions epochs. As expected we see a 2-5% increase

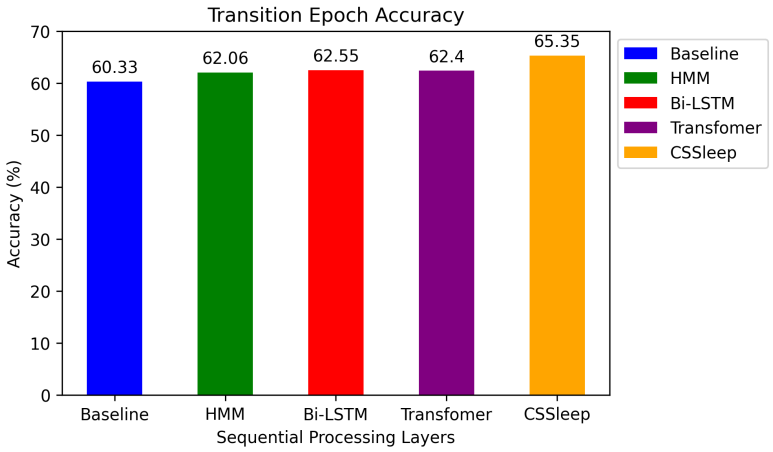

Figure 4.2: Prediction accuracies in % of the different sequential processing layers on transitional epochs. Transition epochs here are defined as epochs that have a label different to either of its neighbours.

in the accuracy of predicting these transitions when using temporal data from neighbours.

5. Experiments and Methodology

In this chapter we detail the different experiments we performed trying out new architectures and techniques along with their results.

5.1 New Architectures

First we look at some of the new State-of-the-Art models in Image Classification. While CNNs have been predominantly in the forefront for feature extraction in image tasks, the advent of transformers has brought around a new set of architectures known as Vision Transformers or ViTs. New Convolutional Neural Networks based on transformers have also been introduced such as ConvNeXt. With a common drawback being the large amount of data required, we check to see if these new architectures are able to improve of the performance of existing models.

5.1.1 Vision Transformers

Vision Transformers (ViTs) have gained attention for their unique approach to computer vision tasks. Unlike traditional Convolutional Neural Networks (CNNs), ViTs rely on self-attention mechanisms, similar to those in natural language processing. This allows them to capture long-range dependencies and global context in images, making them highly versatile for various vision tasks. ViTs can be pre-trained on large datasets, enabling transfer learning, and they are capable of handling different input resolutions without architectural changes. However, ViTs may require extensive computational resources, limiting their accessibility. Moreover, they are more sensitive to input data size and might need larger datasets to generalize effectively.

Here we look at some of the ViT architectures and how they perform on the sleep stage classification problem.

Vanilla ViT

The ViT architecture, introduced in 2020 [23], was designed with the principles of the Transformer architecture in mind, which had seen significant success in natural language processing. However, the key innovation was in adapting the Transformer's mechanisms to the realm of computer vision. Instead of treating words in a sentence as tokens, ViTs divided an image into fixed-size patches, treating each patch as a visual token. These image patches retained both spatial and feature information, allowing ViTs to capture relationships between regions in the image while preserving their distinct features. ViTs copy the Transformer encoder section, employing a combination of self-attention layers and feedforward neural networks to process these patches, enabling them to understand context and semantics within images. This is shown in Figure 5.1.

Each image patch is projected to an embedding via a linear layer and mixed in with a positional embedding that is learnt during training. For classification, an additional learnable 'class' encoding is also added and its output is taken to finally classify the image. The weighted NLL loss was used to train the model against the labelled output.

The best model we got was by using a patch size of 16, along with a depth of 8 for the transformer encoder. Each of the encoders also use 16 heads for the multi head attention layer.

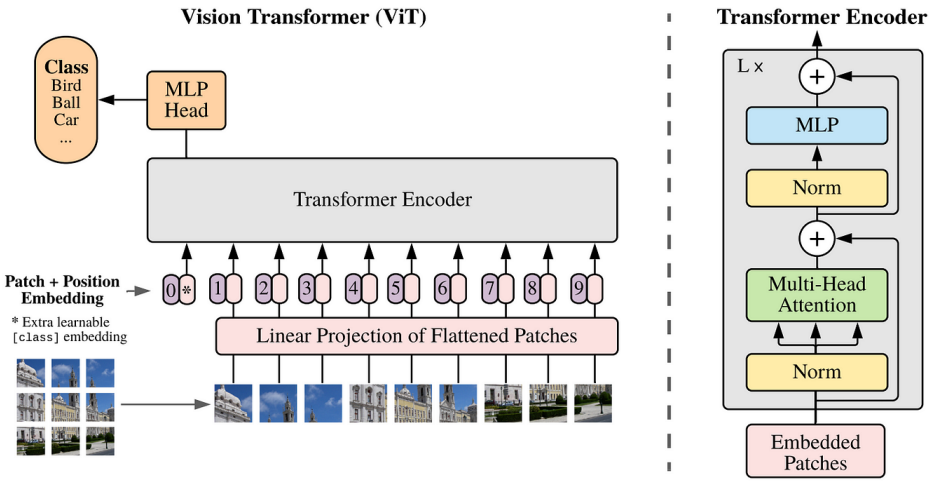

Figure 5.1: ViT architecture as published in [23]. Images are patched, mixed with positional embeddings and passed to a Transformer encoder to obtain feature values that are then classified.

Convolutional vision Transformer (CvT)

With the success of convolution for feature extraction, we tried out two Convolution based Vision Transformers. The first, called Convolution Vision Transformer [33] or CvT uses convolutional layers in order to create the patches and their embeddings. Rather than breaking the image into patches, they use a convolution layer with overlap to create the patches as well as another convolution layer to create the query, key and value embeddings which is then flattened and passed through a transformer encoder layer.

They implement this in a three stage manner in order to borrow the multi-stage hierarchy design from traditional CNNs. The additional Class (CLS) token is added only in the third layer for the classification purpose. Furthermore they show that the additional of positional embeddings provides negligible improvements as the convolutional layers already incorporate spatial information contexts. This is shown in Figure 5.2.

We used the standard dimensions provided in the paper for the kernel sizes and strides for the different layers.

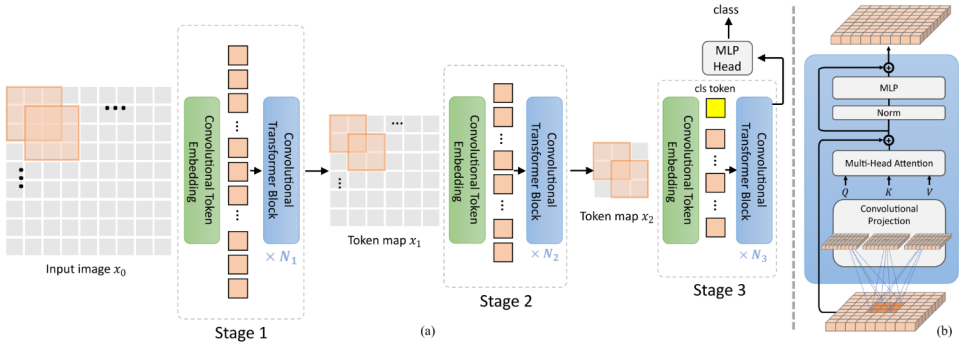

Figure 5.2: CvT architecture as published in [33]. The model extends a ViT by using CNNs for creating the patches and their embedding and introduces an hierarchical structure.

Compact Convolutional Transformer (CCT)

The Compact Convolutional Transformer (CCT) builds on the original ViT in 3 different ways. First, they introduce a version of the ViT called ViT-Lite which adjusts the patch sizes and the number of encoder layers to form a smaller model. They then introduce a Sequence Pooling (SeqPool) layer at the end before classification. While the standard ViT uses the final embedding of the learned classifier token alone for classification, the Sequence Pooling layer uses an attention based layer to pool information over all the patch outputs. Finally similar to the CvT, the CCT also uses a convolution layer for

the initial generation of patch tokens from the image. By using a convolution layer with overlap, they are able to gain more spatial information in the tokens as well as having a patch size that is not restricted to the dimensions of the image.

By using smaller number of layers, a convolutional layer for the token generation as well as smaller patch sizes, the parameters for the model are small giving rise to the name "Compact" Convolutional Transformers. This is shown in Figure 5.3.

We tried different variations of the model on our data set, with the best output received from using a kernel size of 16 for the convolution layer.

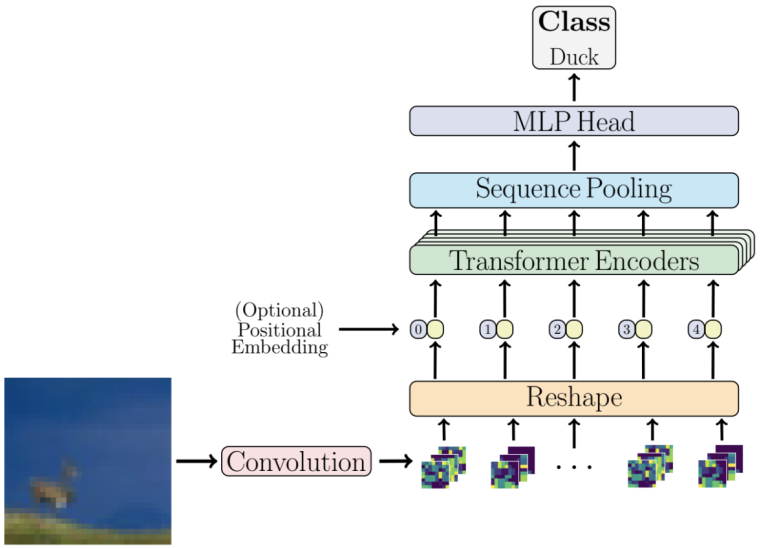

Figure 5.3: CCT architecture as published in [25]. CCTs use CNNs to create the patch embeddings and also use a sequence pooling layer to collate information from all patches instead of using a CLS token as done in ViTs.

Vision Transformers for Small Data Sets

Compared to Convolutional Neural Networks, ViTs lack the inductive biases of locality and translational equivariance, which is overcome via the use of a large training data set. While the use of convolutional layers is one way to incorporate this bias as seen in the previous two ViT versions, Lee et al [34] introduced two changes to standard ViTs to enable them to learn features better using smaller datasets. The first is the introduction of image shifting, which they called Shifting Patch Tokenization (SPT). Here they shift the input images in all 4 directions, crop the shifted image to match the input size and concatenate all the shifted patches with the input to form an input with 5

times the number of channels as before. Thus each created patch now has additional overlapping context information with its neighbouring patches.

The second change they introduce is called Locality Self Attention or LSA. To ensure a stronger attention on neighbouring features they add two changes to the multi head attention layer in the Transformer. Firstly, they use diagonal masking where they set the diagonals of the attention matrix to -∞ in order remove attention of a patch to itself. Secondly, they normalize and pass the attention matrix through a softmax layer with a learn able temperature. This learned temperature is shown to be lower than the standard ViT, thus sharpening the distribution by paying more attention to the stronger inter patch interactions.

With these changes applied to the various ViT networks, they show that the overall accuracy achieved by the network can be improved.

Swin Transformer

The Swin Transformer [24] based on the concept of **S**hifted **Win**dows aims to alleviate the problem of image patches in Transformers being a fixed size, while image features are more diverse. It addresses this issue in three ways. Firstly it uses a hierarchical structure of Transformers with increasing capacity (input feature dimension). Lower layers attend to patches of smaller sizes, while a merging layer in between combines neighbouring patches together before the next layer, thus attending to patches of different resolutions. Secondly, in order to maintain a linear computational cost, Swin Transformers calculate attention within limited non-overlapping 'windows'. Patches no longer attend to every other patch, but to neighbours in the same window, thus keeping the time complexity linear. Finally in order to bridge the gap between these windows, Swin Transformers uses shifted windows between each alternate layer hence ensuring information to be passed around in deeper layers.

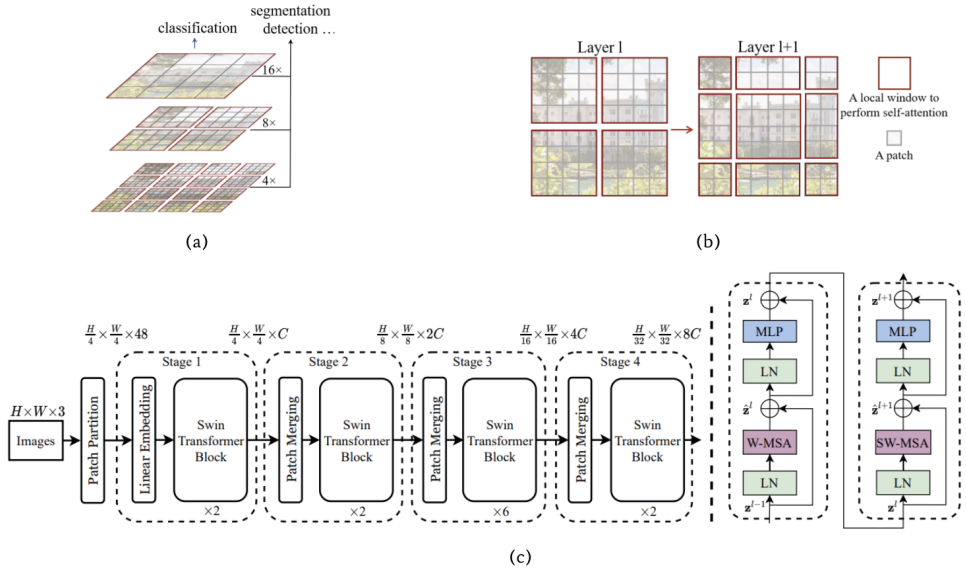

Figure 5.4: Swin Transformer architecture as published in [24]. (a) Shows the hierarchical architecture pattern, (b) shows the use of sliding windows and (c) shows the full architectural specs as well as the composition of a Swin Transformer block.

With these changes, Swin Transformers are able to achieve high accuracies over classification, detection as well as segmentation tasks due to their ability to process higher resolutions. We use the Swin Tiny (Swin-T) architecture specifications as described in the paper.

ViT Results

All models were trained with the first two mice in Cohort A and tested on the remaining rodents , similar to Spindle. All models were trained with a learning rate of 5x10-5 using the AdamW optimizer commonly used for training ViTs. The negative log likelihood loss with weighted class distributions same as the one used to train Spindle was used. Models were also trained for 25 epochs as compared to 20 from Spindle as some of them took longer to reach saturation. All runs were done 5 times and the mean and standard deviations were calculated.

The prediction accuracies of the different models per cohort have been tabulated in Table 5.1. The table does not include the per class metrics as they do not substantially introduce any different results. We can observe that while all ViTs have good performances, CCTs were able to perform the best. However all ViTs were unable to beat Spindle's single convolution network. Spindle was able to achieve higher accuracies in all 4 cohorts of the Spindle data set while having a relatively high accuracy on the Slumber data set.

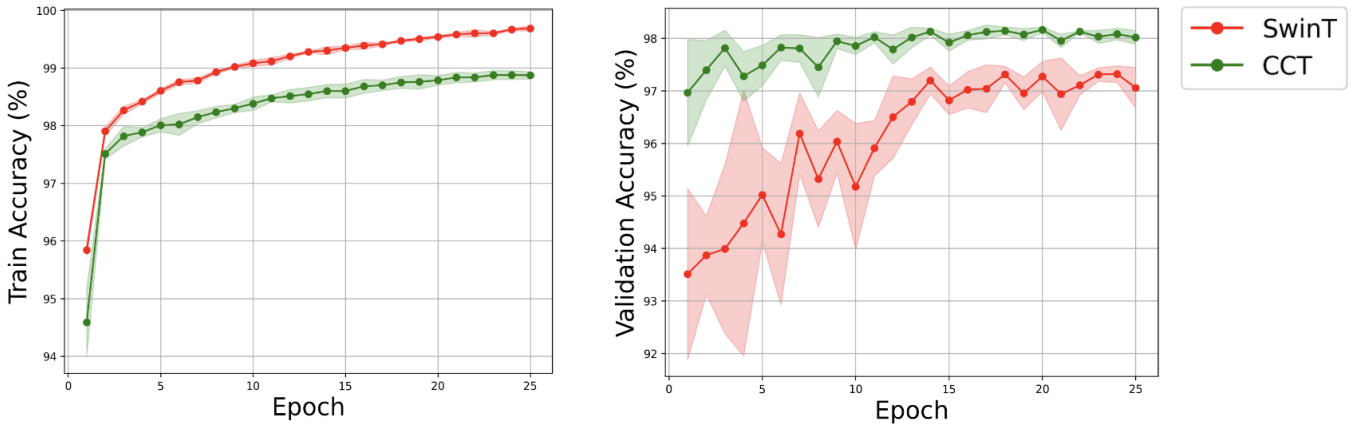

Figure 5.5: Training and Validation Accuracies of the CCT and SwinT models.

Table 5.1: Mean Accuracies (in %) of the different ViT models when trained on the 1st two mice of Cohort A of the Spindle data set. Accuracies are shown per Spindle data set cohort as well as for the Slumber training set. Values for the Spindle model are taken from Table 4.1

Model	Cohort A	Cohort B	Cohort C	Cohort D	Slumber
ViT-16	97.31 ± 0.26	94.25 ± 0.39	89.79 ± 0.14	95.07 ± 0.14	89.5 ± 0.40
CvT	97.68 ± 0.16	95.34 ± 0.40	90.28 ± 0.24	95.75 ± 0.13	90.1 ± 1.09
CCT	98.02 ± 0.13	95.92 ± 0.47	89.89 ± 0.46	95.51 ± 0.48	90.28 ± 0.31
ViT for SD	96.39 ± 0.43	92.45 ± 0.76	89.10 ± 0.49	94.17 ± 0.48	87.68 ± 0.73
Swin Transformer	97.06 ± 0.39	93.79 ± 0.76	89.9 ± 0.31	95.01 ± 0.4	88.53 ± 1.5
Spindle	99.08 ± 0.07	96.54 ± 0.22	91.27 ± 0.25	95.98 ± 0.17	89.88 ± 0.38

One interesting observation we noticed during this was also a case of over fitting. As displayed in Figure **??**, the training and validation accuracies of the CCT and SwinT models show a variation in performance. The SwinT model is able to achieve a very high training accuracy fitting very well to the data, but its performance on the validation and test sets falls when compared with the CCT model. This is an important observation that gave importance to and directed us to look at Data Augmentation techniques which we look at in the next section.

5.1.2 ConvNeXt

ConvNeXt [26], published in 2022, aims to push the boundaries of what a convolutional neural network can achieve. Rather than incorporating CNN inductive biases in the ViT space such as the use of convolutions in CCTs

[25] or the hierarchical structure as in a Swin Transformer [24], the authors apply the concepts of a Transformer to ResNet [35].

Multiple steps are performed to 'modernize' ResNet. First the authors use augmentation techniques such as Mixup or CutMix in order to improve the generalization of the model. Secondly they perform some 'Macro' adjustments to the model to emulate the Swin Transformer architecture. They change the initial convolution of the ResNet, known as the stem, to use a 4x4 kernel with a stride of 4 in order to simulate the 'patches' used in Transformers. They also change the stage compute ratio of the ResNet from (3,4,6,3) blocks per stage to (1,1,3,1) blocks per stage similar to Swin Transformers. Inside each block they introduce a couple of changes. They add separate channel wise and depth wish convolution in order to separate the spatial and temporal convolutions. They also use an inverted bottle neck design where the hidden dimension in each block(384 channels) is 4 times the size of the input dimension (96 channels).

Finally they apply multiple 'Micro' adjustments to the overall architecture. They use the GeLU activation function which is generally used in Transformers instead of ReLU. Also in order to emulate Transformers they reduce the number of normalization layers and activation functions used in each block to just one, also switching out Batch Normalization for Layer Normalization more typically found in Transformers.

With these changes, ConvNeXt is able to outperform the Swin Transformers achieving a maximum top-1 accuracy of 87.8% and an 85.5% on ImageNet-1k data set with and without pre-training on the ImageNet-22K data set respectively.

Results

We trained ConvNeXt similar to the ViTs and Spindle, training it on the first two mice of Cohort A and testing on the remaining rodents of the other cohorts and the Slumber data set. The model was trained with an SGD optimizer and a learning rate of 5x10-5, along with the Negative Log Likelihood loss with class weights, all similar to Spindle. The model also was trained for 20 epochs,

The results of ConvNeXt's prediction accuracies on the different cohorts is listed in Table 5.2. The table also contains the performance of the CCT model as well as Spindle to show an objective comparison. From what we see, while ConvNeXt is able to achieve a high accuracy, better than most ViT models, it is unable to beat the prediction accuracies of the CCT model and Spindle.

Table 5.2: Mean Accuracies (in %) of the ConvNeXt, Spindle and ViT for Small Data sets models when trained on the 1st two mice of Cohort A of the Spindle data set. Accuracies are shown per Spindle data set cohort as well as for the Slumber training set.

Model	Cohort A	Cohort B	Cohort C	Cohort D	Slumber
CCT	98.02 ± 0.13	95.92 ± 0.47	89.89 ± 0.46	95.51 ± 0.48	90.28 ± 0.31
ConNeXt	97.95 ± 0.04	95.63 ± 0.08	90.55 ± 0.16	95.35 ± 0.07	87.26 ± 0.15
Spindle	99.08 ± 0.07	96.54 ± 0.22	91.27 ± 0.25	95.98 ± 0.17	89.88 ± 0.38

5.1.3 Pre Trained Models

Many studies have shown the advantage of using pre-trained architectures such as ConvNeXt [26] which increases its accuracy by 2% on the ImageNet-1k data set with pre-training on the Image-22k data set. Given the performance of ViTs unable to reach the level of Spindle, we wanted to see if pre-trained models could add in some bias and improve the performance.

We took models from the pytorch 'torchvision' library using the uploaded weights as a starting point. We modified the final layers of these models to classify the input into just 3 classes as compared to the default 1000 classes. The models we used are as below:

- **ResNet50**: We used the 50 layer version of ResNet [35] using the IMAGENET1K_V2 version of the weights available in torchvision.
- **ViTs**: We used the 16 and 32 patch size version of the base ViT, ViT-B, [23]. For the weights we used the 'IMAGENET1K_V1' versions of the respective models.
- **SwinT**: We used the standard Swin Transformer (Swin_T) [24] model (Tiny version) in torchvision along with the 'IMAGENET1K_V1' version of the weights.
- **RegNet**: RegNet [36] is a model present with pretrained weights in the torch vision library. We had not evaluated its performance before but it has promising results on ImageNet classification. Hence we tested it out with its pre-trained version using the 8G Flop X version. We used the 'IMAGENET1K_V2' version of the weights.

Results

Since these models already had trained weights, we were fine tuning them to work on our data set. As a result the training parameters had to be updated for the different models.

In order to get the best performance, the learning rate had to be modified and a scheduler needed to be used. For example, ResNet50 with the original

Table 5.3: Accuracies (in %) of the torch vision models with pre trained weights when trained on the 1st two mice of Cohort A of the Spindle data set. Accuracies are shown per Spindle data set cohort as well as for the Slumber training set. Values for the Spindle model are a reference to Table 4.1

Model	Cohort A	Cohort B	Cohort C	Cohort D	Slumber
ResNet-50	98.75 ± 0.04	96.69 ± 0.11	92.3 ± 0.84	96.19 ± 0.32	56.55 ± 8.82
ViT-B 32	97.64 ± 0.19	94.77 ± 0.45	85.2 ± 0.99	93.36 ± 0.7	68.28 ± 0.75
Swin-T	**99.15 ± 0.06**	**97.06 ± 0.21**	90.33 ± 0.51	**96.29 ± 0.24**	**91.07 ± 0.29**
RegNetX 8GFlops	98.32 ± 0.16	96.07 ± 0.4	**92.6 ± 0.51**	95.24 ± 0.66	62.68 ± 7.42
Spindle	99.08 ± 0.07	96.54 ± 0.22	91.27 ± 0.25	95.98 ± 0.17	89.88 ± 0.38

learning rate of $5x10^{-5}$ did not learn the new task well enough, and hence a learning rate of $5x10^{-4}$ with a scheduler. The other models needed to use the scheduler as well, but were able to keep the same learning rate. For the scheduler, we used an exponential scheduler with a gamma (reduction rate) of 0.9. The ViT-B models were also quite big and hence a small batch size of 4 had to be used without exceeding memory value of 8G on the GPU.

All models were run for 5 times and the means and standard deviations of the accuracies were tabulated in Table 5.3. We did not include the ViT-B 16 model results as it took over 2 hours to train and gave a slightly worse result than the ViT-B 32 model. Firstly we noticed that for ResNet, RegNet and the ViT models, their performance on the Slumber data set was quite poor, even with the exponential LR and the high training accuracy. The Swin Transformer on the other hand benefited a lot from the pre training, gaining a nearly 1-3% increase in accuracy over training the model from scratch. This improved performance was able to outperform our local version of Spindle as well, promising a strong alternative architecture for feature extraction.

5.2 Transfer Learning

Transfer learning [37] is a machine learning paradigm where a pretrained model, initially developed for one task, is adapted for a different but related task. This approach leverages the knowledge gained from solving a source task to improve performance on a target task, especially when the amount of labeled data for the target task is limited.

In our case we tried to extract features from the input signal using a different method, and use those learnt features to classify the signals into their respective stages. To this end we tried using Auto Encoders (AEs) to acquire a feature representation of the input signals. We trained an auto encoder to extract feature representations by trying to encode and then decode it back to the original image. This was guided by using a reconstruction loss

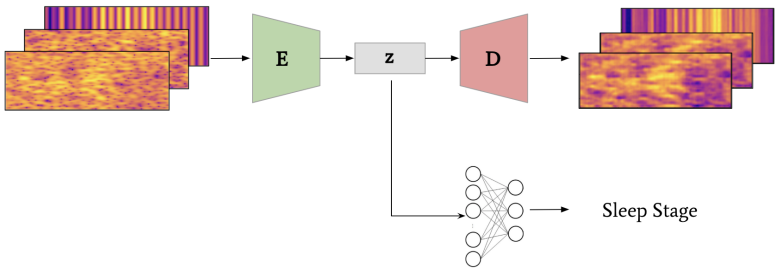

Figure 5.6: Use of Auto Encoder to extract feature representation of input signals. The Encoder (E) is first pre trained to extract features (z) by reconstructing the same input via a decoder(D). These features are then used to classify the signals into the different sleep stages.

between the image and it's generated output. We tried auto encoders with linear and convolutional layers, as well as Variational Auto Encoders (VAEs) by using the KL divergence loss as a regularizer. After training the Auto Encoder, we passed the signal through the AE to get the features and then passed that through a linear layer similar to Spindle to classify the image as seen in Figure 5.6. We used an L1 loss for the reconstruction of the output during the pretraining of the Auto Encoders, while using the Dice Loss for the training of the classification.

We tried different sizes and combinations of layers, and tabulated the specifications and metrics in table **??**. Each auto encoder was first pre trained with the reconstruction loss in order to regenerate the output. These weights were then used in order to generate the feature representation of inputs which were then classified by a linear layer into the sleep stages. The final fine tuning was run 5 times and the mean and standard deviation were computed.

The convolution auto encoder, similar to Spindle, was able to learn feature representations that had the highest accuracy during classification. The ability of CNNs to extract spatial information from the signal is shown to have a high impact. In contrast the Linear Auto Encoders were not able to generate features all that helpful for classification as compared to the Convolutional Auto Encoders. For the linear encoders, we see that that the accuracy is considerably lower than the convolution based encoder. It's possible to hypothesise that the linear layer causes the auto encoder to learn a representational space that does not align well with the sleep stages, thus making classification slightly worse. Using variational versions of the auto encoders by introducing separate mean and standard deviation feature layers, as well as the KL divergence loss, unfortunately were not able to achieve better predictions as well.

Table 5.4: Architecture specifications and overall accuracy (in %) for different Auto Encoder based sleep stage classification. The accuracy is calculated based on the predictions of the models when trained on 2 mice from Cohort A of Spindle and tested on the remaining. Convolution layers are defined by 5 values, the input channels, output channels, kernel size, stride and padding. Linear layers are defined by two values input dimension and output dimension. And finally Max Pool layers are defined by the kernel height and kernel width.

Model	Encoder Specification	Overall Accuracy (%)
Convolutional AE	Conv2d(3,50, (3,3), (1,1), 1) ReLU + MaxPool2d(2,2) Conv2d(50, 4, (3,3), (1,1), 1) ReLU + MaxPool2d(2,2)	92.68 $\pm$ 1.86
Linear AE	Linear(23040, 2048) ReLU Linear(2048, 1024)	89.69 $\pm$ 1.66
Convolutional + Linear AE	Conv2d(3,50, (3,3), (1,1), 1) ReLU + MaxPool2d(2,2) Conv2d(50, 4, (3,3), (1,1), 1) ReLU + MaxPool2d(2,2) Linear(1920, 1024)	88.78 $\pm$ 0.09
Linear VAE	Linear(23040, 2048) ReLU 2 * Linear(2048, 1024) (Mean and Std)	84.84 $\pm$ 1.55
Convolutional VAE	Conv2d(3,50, (3,3), (1,1), 1) ReLU + MaxPool2d(2,2) Conv2d(50, 4, (3,3), (1,1), 1) ReLU + MaxPool2d(2,2) 2 * Linear(1920, 1024) (Mean and Std)	89.11 $\pm$ 0.84

5.3 Hyperparamater Optimization

The other way to improve the model is to change the training parameters such as the loss used or the data preparation such as pre processing and augmentation. In this chapter we look at three different optimizations we tried namely Data Augmentation, different Loss Functions as well as preprocessing in terms of neighbours added per epoch.

5.3.1 Data Augmentation

As we saw from the ViT example, over fitting can be a common problem when training Neural Networks. Data Augmentation can help the training

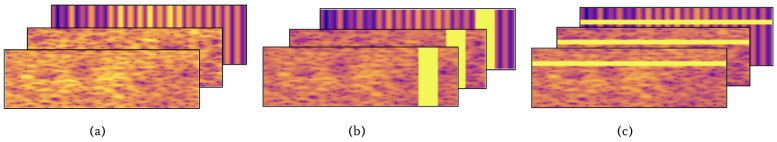

Figure 5.7: Masking applied to the input signals. a) shows the original image while b) shows masking all frequencies of a certain time frame and c) shows masking all frequency values of a given frequency range.

procedure in multiple ways. Augmentation exposes the model to a wider range of patterns and features, enabling it to learn more invariant and discriminative features. By transforming or masking the data, the model learns to distribute the importance of different metrics and helps it become more robust, while also increasing the size of the data set.

Masking

Masking or Random Erasing [38] involves covering up part of the input, similar to a dropout layer in the neural network. By masking the input signals, we ensure the model is not solely focused on certain parts of the signal and is able to learn from all parts of the signal.

We implemented a vertical and horizontal masking of the image. Each signal epoch was preprocessed into 48x32 sized spectrograms and combined with 4 neighbouring epochs to give an input spectrogram of 48x160. For time based or vertical masking, we selected a random section 16 units wide (corresponding to 2 seconds worth of signal data) and masked all frequency values by setting them to 0 across all 3 signals. For the frequency based or horizontal masking, we selected a random horizontal section of 4 units (corresponding to a frequency range of 2 Hz out of the 24 Hz) high and masked it out by setting it's value to 0. Each of these operations had a probability of *mask_prob* of happening at each batch, with *mask_prob* set to 0.5.

Mixup

Mixup [39] involves the combination of images in order to regularize the network in order to favour linear behaviour. Mix up tries to build the fact that linear interpolation of the feature vectors should lead to linear interpolations of their corresponding labels.

Mixup is implemented as below:

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

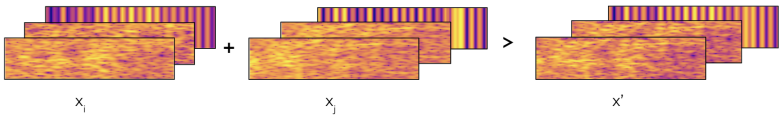

Figure 5.8: Example of Mixup applied to input signals. Here x_i and x_j are combined with a ratio of 0.5 each to give rise to the augmented signal x'.

where x_i, y_i and x_j, y_j are two pairs of inputs and labels. We implmented Mixup in each batch by first permuting the data and labels, and then using a linear combination of the two. λ was randomly sampled from a Beta distribution with parameters α and β set to 0.1. Mixup was applied to each batch with a probability of *mixup_prob* set to 0.5.

In order to account for the mixup, the loss was also calculated proportionately. The loss was calculated as:

$$loss = \lambda.NLL(f(x_i), y_i) + (1 - \lambda).NLL(f(x_j), y_j)$$

where f(x) is the model mapping inputs x to the output class probabilities.

Analysis

We trained Spindle with both these data augmentation techniques, each on its own as well as together. We used the same training procedure as before, training Spindle on the 2 mice of Cohort A and testing it on the remaining subjects. All runs were run 5 times with the mean values tabulated (Mean and standard deviation shown in Appendix).

Table 5.5 shows the prediction performances of Spindle trained using these methods. Since these methods helped the model perform better, we will look at the overall accuracy as well as the per cohort metrics. The baseline was taken as the same Spindle model with results in Table 4.1. In these results we show the overall performance when tested on the other cohorts of the Spindle data set. We do not include the Slumber data set results as the volume of data in the slumber data set is much larger and hence the performance on that dataset affects the overall metrics by a higher percentage.

We see that using these augmentations has a positive impact on Spindle's predictions. Specifically if we look at the per class metrics, we see an improvement in the F1-score for all classes when using any of the data augmentation metrics, with the highly imbalanced REM class gaining a 2-3% increase in F1-score. This lead to an increase in the Overall Accuracy as well as Mean-F1 and Kappa metrics.

Table 5.5: Mean performance metrics of Spindle with data augmentation methods. The results show the combined metrics of the model's predictions on the 4 different test set cohorts of the Spindle data set.

Layer	Acc(%)	MF1(%)	Kappa(%)	Wake			NREM			REM		
				P	R	F1	P	R	F1	P	R	F1
Without Aug.	94.68	91.8	90.5	91.57	98.55	94.93	98.5	93.32	95.84	90.37	79.55	84.61
Masking Only	95.13	92.53	91.34	92.8	98.37	95.5	98.58	93.76	96.1	87.97	84.16	86
Mixup Only	95.37	93.05	91.76	93.23	98.23	95.67	98.38	94.25	96.27	89.54	85.06	**87.21**
Both	**95.52**	93.09	92.04	94.08	97.86	**95.94**	98.27	94.61	**96.4**	86.86	87.02	86.92

5.3.2 Loss Functions

Loss functions play a crucial role during training by guiding the optimization process, where the goal is to minimize the loss. The choice of an appropriate loss function is quite relevant, as it directly impacts the model's ability to generalize and make accurate predictions on unseen data. We saw one such example when training the ViTs where they were able to learn the training set better than ConvNeXt but performed worse on the validation set.

In this section we look at three different losses: the Focal loss, the Dice loss and a custom loss aimed at focusing on epochs that transition between sleep stages.

Dynamic Weighted Loss Functions

One of the features of this data set is the between classes. The number of REM epochs present in the data is much lower (less than 1-% of the data). To combat this, Spindle originally uses a weighted NLL loss where is weight is inversely proportional to number of epochs of that particular class. This however is a static weighing. It would always give a weight of x_I to class I. One thing we noticed when testing ConvNeXt was that ConvNeXt was able to achieve a lower accuracy on the validation set, but also had a lower loss than Spindle, which can be seen in Figure TODO**??**. Since the NLL loss is proportional to the predicted probability, while the output is chosen as the argmax, this could mean that while Spindle was predicting the correct output classes, it was not as confident in its predictions.

We instead look at Dynamically weighted loss functions, which are loss functions that dynamically adjust the weights applied to each input based on different criteria. The point of these functions is to focus more on inputs that are harder to classify as compared to inputs that are more straightforward.

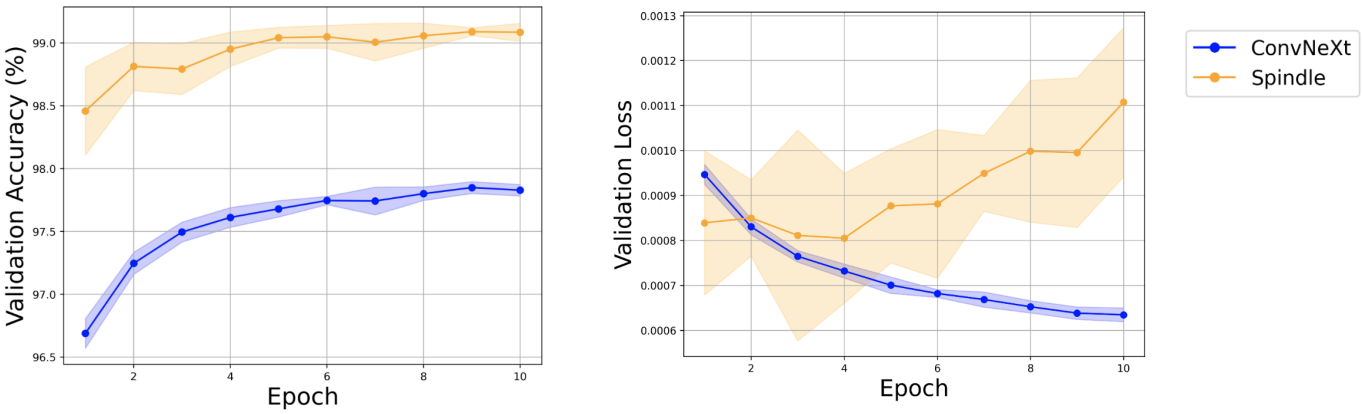

Figure 5.9: Validation accuracy and loss for the Spindle and ConvNeXt model during training. The train data set consists of two mice from Cohort A while the validation set consists of the other two mice from Cohort A of the Spindle data set.

Focal Loss

The Focal Loss Function [40] is a specialized loss function designed to address the issue of class imbalance in binary classification tasks. Introduced by Lin et al., it assigns higher weights to misclassified examples, emphasizing the learning of challenging instances and mitigating the impact of abundant easy-to-classify samples. The formula is given by:

$$\text{Focal Loss}(p_t) = -(1 - p_t)^{\gamma} \log(p_t)$$

where p_t is the predicted probability of the true class and γ is the focusing parameter. If the model is confident in its prediction, then $1 - p_t$ is small and hence the overall loss is reduced. However if the model is not confident or incorrectly predicts the class, then p_t is small and hence $1 - p_t$ is larger, giving more weight to this training example.

Dice Loss

The Dice Loss [41] was introduced mainly to handle class imbalances in Image Segmentation tasks. Given the output mask and the predictions in a Segmentation task, the formula for the Dice loss is given by:

$$\text{Dice Coefficient} = \frac{2 \cdot \text{Intersection}}{\text{Union} + \epsilon}$$

$$\text{Dice Loss} = 1 - \text{Dice Coefficient}$$

where the Intersection and Union are calculated between the true mask and the predicted mask. This can also be applied to our multi class classification problem by calculating the dice loss per class. When the model is confident in its prediction, then the intersection value is high. As a result the dice coefficient is also high and the loss becomes smaller. When the model is not as confident, the reverse is true and the loss increases. Hence, similar to the Focal loss, the Dice loss directs the model to learn the harder to classify examples.

Custom Loss - Transition focus

If we look at transition regions (times when the subject is moving between different sleep stages), what we noticed was that the transition regions are one of the harder regions to classify. This is mainly because transitions between stages are more gradual and such epochs would contain features from both stages. As seen in figure 4.2, models only reached about a 60% accuracy when classifying epochs that were transitioning between stages.

To focus on this we used a different version of the NLL loss. We wanted to weigh these transition epochs higher. The formula can be written as:

$$Loss(i) = \begin{cases} -log(p_i) * w_c & \text{if label}[i] \text{ != label}[i \pm 1] \\ -log(p_i) * \delta & \text{otherwise} \end{cases}$$

where p_i is the predicted probability for example i, w_c is the weight of the respective output class (same as the ones used for NLL loss) and δ is an importance factor that amplifies the loss for these examples.

Analysis

We used the Spindle model as the base in order the compare the performance of these loss functions using the same training procedure as mentioned. All runs were performed 5 times and the mean values were calculated (Full mean and standard deviation values can be seen in the appendix). In Table 5.6 we see the overall metrics of the predictions of the Spindle model on the Spindle data set. While the performance on the other two mice of Cohort A drops, there is an increase in the overall accuracy and specifically the F1 score of the REM class. This is similar to the effects of using the data augmentation methods, which shows that by using these new loss functions, the model was no longer directly over fitting on the easy W and N classes, and was instead generalizing more and focusing on the harder to classify epochs.

However one point we noticed was that the performance on the Slumber data set was inverted. While the overall accuracy and the per class metrics improved when testing the different loss functions on the Spindle data set, it was reduced when tested on the Slumber data set. This can be see from the results in Table 5.7. We assume that the Slumber data set is more similar to Cohort A of the Spindle data set, and hence when using the new losses and having a more generalized prediction, it does not overfit to Cohort A and hence also has a reduced performance on the Slumber data set.

Table 5.6: Mean performance metrics of Spindle with different loss methods on the Spindle data set. The accuracy shown is for both - Cohort A as well as combined metric over all 4 cohorts. The per class metrics are calculated from the combined predictions of all 4 cohorts.

Loss	Overall Acc(%)	Cohort A Acc (%)	Wake			NREM			REM		
			P	R	F1	P	R	F1	P	R	F1
NLL	94.67	99.11	91.54	98.61	94.94	98.6	93.21	95.83	89.87	79.81	84.54
Dice	95.53	98.73	95.18	96.95	96.06	97.69	95.0	96.32	84.12	90.18	87.05
Focal	95.54	98.88	94.61	97.38	95.98	97.89	94.94	96.39	86.18	88.01	87.05
Transition	95.48	98.97	94.59	97.73	96.13	98.17	94.68	96.39	85.23	88.22	86.7

Table 5.7: Mean Performance metrics of Spindle with different loss methods on the Slumber data set. The accuracy and per class metrics are calculated from the predictions of the model on the Slumber data set.

Loss	Overall Acc(%)	Wake			NREM			REM		
		P	R	F1	P	R	F1	P	R	F1
NLL	90.4	91.29	92.84	92.06	90.3	88.07	89.17	78.23	77.84	78.03
Dice	86.96	91.49	86.03	88.68	86	88.44	87.2	54.27	85.95	66.52
Focal	87.85	91.33	88.33	89.8	85.64	87.99	86.8	66.04	79.63	72.2
Transition	87.92	91.6	87.89	89.71	87.19	88.29	87.73	57.95	84.71	68.82

5.3.3 Neighbours per Epoch

The preprocessing for Spindle involves converting the signal into it's corresponding spectrogram, breaking it into epochs and then also attaching copies of neighbours to each input. As a result the input for sample *i* consists of the spectrogram of epochs *i-2* to *i+2* when having 4 neighbours. We do this in order to incorporate temporal information into each input.

In this section we wanted to quantify the impact of using neighbours in input data. Previously the pipeline would perform the preprocessing first and save the preprocessed input into an accessible HDF5 file which would then

Table 5.8: Performance of Spindle when using different number of neighbours in the input for each epoch.

# of Neighbours	Accuracy (%)	Runtime (s)	Model Parameters (M)
0	94.05	223	20.7
2	94.01	313	33.83
4	94.77	425	77.99
8	94.86	789	166.31
16	95.19	1507	342.95
20	95.51	1570	431.27

be trained on. Hence the input would already consist of the epoch signal data along with 4 neighbours. We changed this to instead only save each samples spectrogram data without any neighbours. We then changed the data loader to return the required number of neighbours in it's '_get_item' call. 0 neighbours would just return the sample's spectrogram data alone, while setting it to N neighbours would return spectrogram data from epoch *i* - *N/2* to epoch *i* + *N/2*. Epochs on the edges were padded with 0s.

The overall accuracy on the Spindle data set, the run time as well as the model parameters have been tabulated in Table 5.8. We notice that while there is an increment in the overall accuracy when increasing the number of neighbours, the cost in terms of run time and memory shoots up exponentially. For example when use 8 neighbours instead of 4 neighbours, the number of parameters in the model doubles from $\sim$78 million to $\sim$166 million parameters, as well as doubling the time taken to train the model. All this results in a 0.11% gain in accuracy.

Not only that, but in order to generate data with its neighbours quickly, the model either has to keep all the data in memory, precompute the data or read multiple items at a time, all of which require increased computing and memory resources. Hence, while there is an increase in performance, the additional computation required may not be justified.

6. New Proposed Architecture - SwinTsle

Combining the different architectures and techniques, we propose a new architecture - SwinTsle, or Swin Transformer for Sleep Classification.

6.1 Architecture

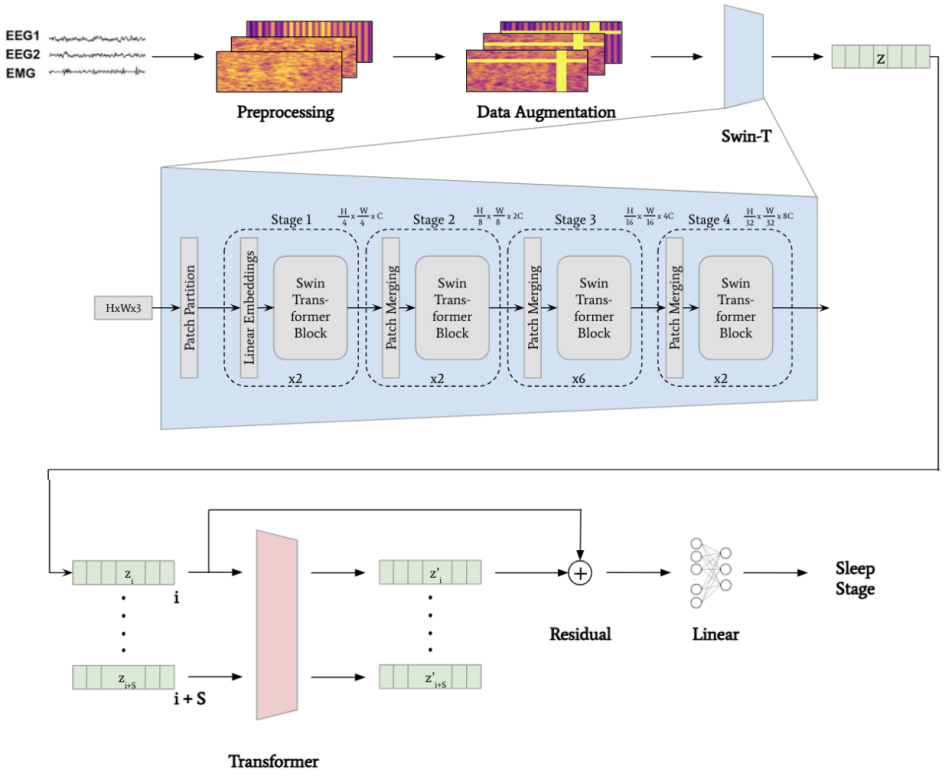

Figure 6.1: SwinTsle Model Architecture.

The new architecture (Figure 6.1) is composed of the different optimizations tried out in the previous chapters. We used the same preprocessing as Spindle, converting the signals into their Spectrograms. We used a pre trained Swin Transformer Tiny (Swin-T) as the base to extract features from the input. We used masking and mixup as data augmentation techniques in order to regularize the model, as well as the Dice Loss to handle the class imbalance. We added a transformer layer over the final extracted features to learn temporal context information and finally passed the transformer output along with a residual of the extracted features in to a linear layer in order to classify the epochs into the respective sleep stages.

6.2 Analysis

The different additional stages along with the overall performances have been tabulated in Table 6.1, while their Per Cohort Accuracies have been calculated and shown in Table 6.2. Overall our final model has been able to achieve a nearly 1.5% increment in the overall prediction accuracy. The per class metrics have also improved, with the Mean F1 score improving by around 2.5% mainly due to the improved Precision and Recall of the REM class.

Looking at the per cohort accuracies in Table 6.2 we also see a consistent increase in the prediction accuracies over all the cohorts. Due to the implementation of the data augmentation, dice loss function as well as temporal processing, we see that the performance for Cohort C, made up of rats having a different input distribution and signal set as the train Cohort A, has the most improvement showing that the model is also able to generalize better. The Slumber data set though shows a slightly different pattern. Given the data set being larger and the model trained on a completely different set, small variations in the learnt model affect the accuracy on the Slumber data set and hence not a incremental increase. However we can see that the final proposed architecture is capable of achieving a high accuracy on the Slumber data set.

Table 6.1: Mean Overall accuracy and per class metrics of the given architectures on the Spindle data set. Values are calculated as the average calculated over 5 runs.

Layer	Acc(%)	MF1(%)	Kappa(%)	Wake			NREM			REM		
				P	R	F1	P	R	F1	P	R	F1
Spindle	94.68	91.8	90.5	91.57	98.55	94.93	98.5	93.32	95.84	90.37	79.55	84.61
Pretrained Swin-T	95.05	91.4	91.12	91.64	99.0	95.18	98.7	94.58	96.59	94.07	73.38	82.42
+ Data Aug	95.46	93.24	91.91	92.92	98.78	95.76	98.7	94.0	96.29	91.01	84.62	87.65
+ Dice Loss	95.56	93.08	92.05	92.66	98.95	95.7	98.68	94.6	96.59	94.31	80.73	86.96
+ Transformer (SwinTsle)	96.03	93.74	92.92	93.69	98.89	96.22	98.71	95.29	96.97	93.63	83.18	88.04

Table 6.2: Per Cohort Accuracy(in %) of the given architectures on the Spindle data set as well as the Slumber data set.

Model	Cohort A	Cohort B	Cohort C	Cohort D	Slumber
Spindle	99.08 ± 0.07	96.54 ± 0.22	91.27 ± 0.25	95.98 ± 0.17	89.88 ± 0.38
Pretrained Swin-T	99.18 ± 0.08	97.35 ± 0.15	91.17 ± 0.48	96.67 ± 0.16	91.0 ± 0.58
+ Data Aug	99.26 ± 0.02	97.08 ± 0.15	92.86 ± 0.67	96.24 ± 0.12	87.58 ± 1.68
+ Dice Loss	99.25 ± 0.03	97.22 ± 0.08	92.88 ± 0.56	96.41 ± 0.12	90.15 ± 0.86
+ Transformer (SwinTsle)	99.3 ± 0.04	97.41 ± 0.11	93.5 ± 0.45	97.01 ± 0.21	91.28 ± 0.16

Expanding the per cohort metrics, in Figure 6.2 we see the confusion matrices of one of the runs on the different Cohorts of the Spindle data set as well as the Slumber Data set. Swintsle is clearly able to learn the different classes well even in Cohort C and the Slumber data set.

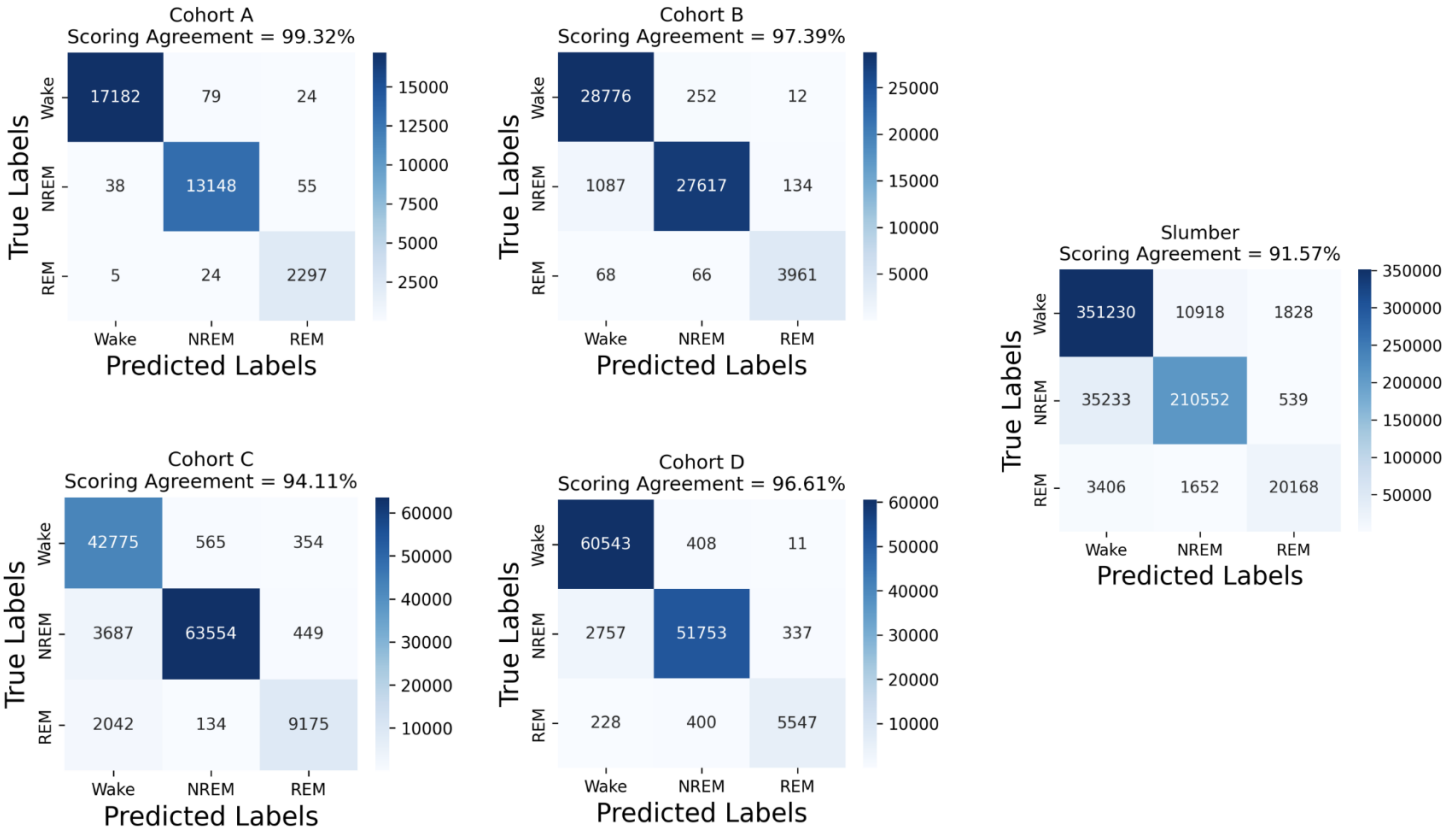

Figure 6.2: Confusion Matrix of SwinTsle predictions on the different Cohorts of Spindle as the Slumber data set.

7. Human Data Set

A large amount of research has also been conducted in the field of human sleep stage classification, which is a problem highly similar to the sleep stage classification in Mice. Humans display more complex brain activity leading to the classification of the NREM sleep stage further into N1, N2 and N3. Humans also sleep for longer periods in general with the transition between sleep stages being less frequent. Hence signals are usually classified in 30 second epochs, as compared to 4-8s epochs in mice. These changes lead to the development of different models for these two tasks.

In this section we explore the performance of Spindle as well as our newly proposed model on the Human data set. We also compare it with the SOTA Human sleep stage classification models based on the values reported in the papers. We do not have any strong expectations for the performance of these models on the human data set, but rather just wish to analyze the results and find out if any surprising results can be seen.

7.1 Setup

We use the Sleep-EDF 20 data set mentioned in Chapter 2. We pre process the data by converting the signals to their spectrograms normalizing them and splitting them into 30s epochs. Unlike for the mice data set, we do not add neighbouring signal data to the input for each epoch, following the common practice done in other Human Sleep Classification studies.

All data was taken and dynamically split into and 80-20 train and test data sets. In the case of our new proposed model, in order to ensure sequential information processing, the data loader was set to batch the data first, allowing for shuffling of batched sets. Since this data was more balanced due to the clipping of the input to within 30 minutes of sleep (as mentioned in chapter 2), the weighted NLL loss was used as compared to the Dice loss. Hyper parameter tuning gave us the learning rate of $5*10^{-5}$ with an exponential scheduler with gamma 0.9, that was trained for 30 epochs.

We trained both Spindle and our new proposed architecture of this dataset. Swindle was pretrained on the data without the Spindle does not use the HMM as there are five classes and the transition rules for Mice sleep stage does not apply here. Also, both models were trained on 1 EEG signal (the

Table 7.1: Mean predictions metrics for different models on the Sleep-EDF 20 data set.

Model	Input	Overall Metrics			Per-Class F1-Score				
		Accuracy	MF1	Kappa	W	N1	N2	N3	REM
AttnSleep	Fpz-Cz	84.4	78.1	79	89.7	42.6	88.8	90.2	79
SleepViTransformer	Fpz-Cz, ROC-LOC	87.8	81.5	83.4	93.8	48.4	89.2	88.4	87.9
Spindle	Fpz-Cz	75.86	69.37	67.49	80.17	31.45	82.45	81.59	71.2
Swin Transformer	Fpz-Cz	82.49	76.12	76.05	90.28	40.76	86.1	85.23	77.9
Swindle	Fpz-Cz	84.73	79.13	78.86	91.88	46.18	86.64	85.94	83.65

Fpz-Cz EEG signal) as most SOTA human models use only 1 EEG signal as input.

7.2 Results

The Overall prediction metrics as well as the F1-score of the different classes were calculated and have been tabulated in Table 7.1. Models were run for 5 runs with different seeds and the means were taken (mean and standard deviations can be found in the appendix). The results of AttnSleep and SleepViTransformer have been added in as well, having been taken from the AttnSleep paper [18] and SleepViTransformer paper [28] respectively.

First off we notice that Spindle is able to classify this problem decently well, but does not compare to the SOTA models. However we see that our new proposed architecture is able to much better than Spindle and even marginally surpasses the results from AttnSleep. This shows great promise for our new proposed architecture. SleepViTransformer still has the highest prediction metrics, though it does use the EEG and an EOG signal.

8. Discussion

Spindle was one of the first few architectures that used Convolutional Neural Networks in the Sleep Stage Classification Space. Combining CNNs with an HMM, Spindle was a SOTA model that achieved stellar results, still very strong to this date. With the leaps and bounds of progress happening in the field of Computer Vision, we were able to use new technologies to propose our new model - SwinTsle (Swin Transformer for Sleep Classification) that is able to achieve a nearly 2% improvement in accuracy on the Spindle data set than our local implementation of Spindle. One of the ways SwinTsle is able to achieve this is due to it's improved REM stage classification, improving the F1-score by more than 5% when tested on the Spindle data set. Even in the Slumber data set, SwinTsle is able to achieve a higher prediction accuracy than Spindle achieving a 91.57% accuracy as compared to Spindle which was able to achieve a 90.4% accuracy.

SwinTsle is able to achieve its level of performance through the combination of multiple features. Firstly it uses a fine tuned version of a pre trained Swin Transformer as it's base to extract features. Swin Transformers, a particular type of Vision Transformers are a SOTA vision model that combines a) the features of Vision Transformers by 'patchifying' images and using a transformer on the patch embeddings, b) the hierarchical features of architectures such as the U-Net by using increasing patch sizes in subsequent layers, and c) the features of local and global attention of convolutional networks by using shifting windows. Moreover by using a pre trained set of weights as compared to a random initialization of weights SwinTsle is able to achieve an accuracy already surpassing Spindle.

Secondly, SwinTsle uses Data Augmentation techniques enhancing the data set in order to handle the issue of over fitting the model to the train data set. Both Masking, the process of randomly removing parts of the input, as well as Mixup, the process of mixing up two inputs, help ensure the model does not pay attention to only small part of the training input data and forces it to learn features from all the different parts of the input. Both have been shown here to have positive impacts of the prediction performance of models.

SwinTsle was also trained using the Dice Loss in order to handle the issue of class imbalance. In all the training data sets, we see the presence of Sleep Stages that are in minority. Dice Loss, a dynamically weighted loss function, helps the model focus on harder to classify inputs (irrespective of their class)

causing the model to pay less attention to the easy inputs and focus on the harder ones.

Finally, while Spindle uses prior knowledge incorporated into HMMs in order to learn temporal information, SwinTsle uses a Transformer to learn these temporal inter epoch features instead. We show that it is able to learn these transitional rules quite well, and is able to be applied to any task such as the prediction of sleep stages in Humans. With this layer, SwinTsle is able to achieve it's best prediction performance.

During this thesis, we also explored various other models and techniques and evaluated their performance on the given problem. We explored the performance of various other vision architectures such as other ViTs like the CCT and the CvT as well as ConvNeXt. We also explored the performance of other pre trained models such as ResNet and RegNet, as well as the novel idea of using Auto Encoders to learn the feature representations. We investigated other loss functions such as the Focal loss and a custom loss built for this task, and also evaluated the effect of using additional neighbours. Finally we also tested the performance of SwinTsle on the Human Sleep Stage classification task, using one of the smaller publically available data sets. Here as well SwinTsle was able to significantly outperform Spindle, and also outperform one of the SOTA models, AttnSleep.

Using all the learnings from this Thesis, we propose SwinTsle, a next generation state of the art model for sleep stage classification.

9. Future Work

This is a problem still being worked on and improvements to be made. While we did experiment with a lot of different models and techniques there are still things that can be worked on in the future, in order to make a more robust and complete model.

- Data sets: Spindle already has a very high performance when it comes to the Spindle data set, achieving a 99% accuracy on one of the Cohorts and similar values for the other cohorts. In order to show variability and differences in performance, the use of more data sets can help enhance the study.
- Raw Signal Classification: Some of the published models extract features directly from the raw EEG/EMG signals without any preprocessing. This helps speed up inference time by a lot as the uploaded edf signals can be directly classified without requiring the use of multiple windowed Fast Fourier Transforms. If using a raw signal can achieve the same level of performance, it can enhance the experience of the Sleep Learning web tool to Sleep researchers.
- Artifacts: This study did not focus on the prediction of artifact epochs. Artifact epochs were just labelled as normal sleep stages. Focusing on the prediction of artifacts and increasing the 4-class prediction accuracy is something that can be worked upon.
- Human Models: While we saw that the Mice classification models were not able to perform as well as the Human classification models on the Sleep-EDF 20 data. It would be interesting to see if the other way holds or not.

10. Acknowledgements

I would like to express my gratitude to Dr Joachim Buhmann for the opportunity, the inspiration and the resources that helped me work on this project. I would also like to thank the students at the ISE group as well as the ETH AI center for their constant support throughout the project. Most importantly I would like to give a special thanks to my PhD supervisor, Ami Beuret. Even though this is not your main topic of study, you took time out to guide me every step of the way, helping ideate solutions when I got stuck, teaching me new concepts and gave me the support I required. Thank you.

This thesis also marks the culmination of this wondrous Master's journey, and it would not be possible without the support of the numerous friends made along. To my close friend and thesis buddy, Gül Sena, thank you for the encouragement and company these last 6 months, motivating me to be the best version of myself. To my batch mates Viktor, Masa, Martin, Francesco, Filipo, Liine, William, Matt and so many more, thank you for staying by my side and exploring these new fields with me. To my seniors Josephine, Tom, Ayush, David, Karolina and to my many Professors and TAs, thank you for your lessons and patience teaching me concepts in such great detail. To my Board Game Club friends, Marius, David, Oliver, Fabian, Reinhard, Simon, Filip and Emmanuel, thank you for all the fun times that kept me grounded and happy even when studies were stressful. And to my second family here, Ashish, Akshat, Arka, Alan, Aman, Saurabh, Ketan and Ipsita, thank you for all the support, memories and letting me be me. Not to mention, all the friends who have been with me from before that I could always call up at any time and count on for advice.

Lastly, I owe all this to my family who have been by my side constantly. I thank my sister, Dr Anupama, for being my constant source of support through all. From listening patiently to me while I complained about linear algebra and finding housing, to exploring the mountains of Switzerland together, I don't know how I would have done this without you. And finally, thank you to my parents, for always standing by my side at every point, high and low, and giving me the freedom to follow my dreams, even if those dreams needed me to be 7000km from home.

Bibliography

[1] Michael R Irwin. Why sleep is important for health: a psychoneuroimmunology perspective. *Annual review of psychology*, 66:143–172, 2015.

[2] Aakash K Patel, Vamsi Reddy, Karlie R Shumway, and John F Araujo. Physiology, sleep stages. In *StatPearls [Internet]*. StatPearls Publishing, 2022.

[3] Harvey R. Colten and Bruce M. Altevogt, editors. *Sleep Disorders and Sleep Deprivation: An Unmet Public Health Problem*. National Academies Press (US), Washington, DC, 2006.

[4] Zilu Liang and Mario Alberto Chapa-Martell. A multi-level classification approach for sleep stage prediction with processed data derived from consumer wearable activity trackers. *Frontiers in Digital Health*, 3:665946, 2021.

[5] Aliki Minaritzoglou and Emmanouel Vagiakis. Polysomnography: Recent data on procedure and analysis. *Pneumon*, 4:348–368, 2008.

[6] Sayaka Kohtoh, Yujiro Taguchi, Naomi Matsumoto, Masashi Wada, Zhi-Li Huang, and Yoshihiro Urade. Algorithm for sleep scoring in experimental animals based on fast fourier transform power spectrum analysis of the electroencephalogram. *Sleep and Biological Rhythms*, 6:163–171, 2008.

[7] Brooks A. Gross, Christine M. Walsh, Apurva A. Turakhia, Victoria Booth, George A. Mashour, and Gina R. Poe. Open-source logic-based automated sleep scoring software using electrophysiological recordings in rats. *Journal of Neuroscience Methods*, 184(1):10–18, 2009.

[8] Thomas Lampert, Andrea Plano, Jim Austin, and Bettina Platt. On the identification of sleep stages in mouse electroencephalography time-series. *Journal of Neuroscience Methods*, 246:52–64, 2015.

[9] Kirsi-Marja Rytkönen, Jukka Zitting, and Tarja Porkka-Heiskanen. Automated sleep scoring in rats and mice using the naive bayes classifier. *Journal of neuroscience methods*, 202(1):60–64, 2011.

[10] Genshiro A Sunagawa, Hiroyoshi Séi, Shigeki Shimba, Yoshihiro Urade, and Hiroki R Ueda. Faster: an unsupervised fully automated sleep staging method for mice. *Genes to Cells*, 18(6):502–518, 2013.

[11] Michael J Rempe, William C Clegern, and Jonathan P Wisor. An automated sleep-state classification algorithm for quantifying sleep timing and sleep-dependent dynamics of electroencephalographic and cerebral metabolic parameters. *Nature and science of sleep*, pages 85–99, 2015.

[12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[14] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[15] Đorđe Miladinović, Christine Muheim, Stefan Bauer, Andrea Spinnler, Daniela Noain, Mojtaba Bandarabadi, Benjamin Gallusser, Gabriel Krummenacher, Christian Baumann, Antoine Adamantidis, et al. Spindle: End-to-end learning from eeg/emg to extrapolate animal sleep scoring across experimental settings, labs and species. *PLoS computational biology*, 15(4):e1006968, 2019.

[16] Masato Yamabe, Kazumasa Horie, Hiroaki Shiokawa, Hiromasa Funato, Masashi Yanagisawa, and Hiroyuki Kitagawa. Mc-sleepnet: large-scale sleep stage scoring in mice by deep neural networks. *Scientific reports*, 9(1):15793, 2019.

[17] Yuzheng Liu, Zhihong Yang, Yuyang You, Wenjing Shan, and WeiKang Ban. An attention-based temporal convolutional network for rodent sleep stage classification across species, mutants and experimental environments with single-channel electroencephalogram. *Physiological Measurement*, 43(8):085002, 2022.

[18] Emadeldeen Eldele, Zhenghua Chen, Chengyu Liu, Min Wu, Chee-Keong Kwoh, Xiaoli Li, and Cuntai Guan. An attention-based deep learning approach for sleep stage classification with single-channel eeg. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29:809–818, 2021.

[19] Shahab Haghayegh, Kun Hu, Katie Stone, Susan Redline, and Eva Schernhammer. Automated sleep stages classification using convolutional

neural network from raw and time-frequency electroencephalogram signals: Systematic evaluation study. *Journal of Medical Internet Research*, 25:e40211, 2023.

[20] Akara Supratak, Hao Dong, Chao Wu, and Yike Guo. Deepsleepnet: A model for automatic sleep stage scoring based on raw single-channel eeg. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(11):1998–2008, 2017.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[22] OpenAI. Gpt-4 technical report, 2023.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[25] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

[26] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[27] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5270–5279, 2022.

[28] Li Peng, Yanzhen Ren, Zhiheng Luan, Xiong Chen, Xiuping Yang, and Weiping Tu. Sleepvitransformer: Patch-based sleep spectrogram transformer for automatic sleep staging. *Biomedical Signal Processing and Control*, 86:105203, 2023.

[29] Huy Phan, Fernando Andreotti, Navin Cooray, Oliver Y. Chén, and Maarten De Vos. Seqsleepnet: End-to-end hierarchical recurrent neural network for sequence-to-sequence automatic sleep staging. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(3):400–410, 2019.

[30] Huy Phan, Kaare Mikkelsen, Oliver Y. Chén, Philipp Koch, Alfred Mertins, and Maarten De Vos. Sleeptransformer: Automatic sleep staging with interpretability and uncertainty quantification. *IEEE Transactions on Biomedical Engineering*, 69(8):2456–2467, 2022.

[31] Pawan K. Jha, Utham K. Valekunja, and Akhilesh B. Reddy. Slumbernet: Deep learning classification of sleep stages using residual neural networks. *bioRxiv*, 2023.

[32] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[33] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers, 2021.

[34] Seung Hoon Lee, Seunghyun Lee, and Byung Cheol Song. Vision transformer for small-size datasets, 2021.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[36] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces, 2020.

[37] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[38] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017.

[39] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.

[40] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[41] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 240–248. Springer International Publishing, 2017.

A. Appendix

A.1 Data Augmentation Results

The extended version of table 5.5 with the standard deviation included. Values were averaged over 5 runs. The table is split into two in order to fit on to the page.

All models were trained with a batch size of 100 and a learning rate of $5*10^{-5}$ with an exponential learning rate scheduler with gamma set as 0.9, as well as the NLL loss.

Table A.1: Extension of Table 5.5 with mean and standard deviation

Method	Overall Acc(%)	MF1(%)	Kappa(%)
Without Aug.	94.68±0.18	91.8±0.25	90.5±0.33
Masking Only	95.13±0.18	92.53±0.22	91.34±0.31
Mixup Only	95.37±0.14	93.05±0.32	91.76±0.25
Both	95.52±0.11	93.09±0.3	92.04±0.2

Table A.2: Extension of Table 5.5 with mean and standard deviation

Method	Wake			NREM			REM		
	P	R	F1	P	R	F1	P	R	F1
Without Aug.	91.57±0.43	98.55±0.15	94.93±0.18	98.5±0.12	93.32±0.44	95.84±0.18	90.37±0.82	79.55±1.09	84.61±0.51
Masking Only	92.8±0.43	98.37±0.16	95.5±0.17	98.58±0.09	93.76±0.42	96.1±0.18	87.97±1.34	84.16±1.45	86.0±0.4
Mixup Only	93.23±0.41	98.23±0.16	95.67±0.16	98.38±0.04	94.25±0.16	96.27±0.08	89.54±1.07	85.06±2.37	87.21±0.77
Both	94.08±0.24	97.86±0.13	95.94±0.09	98.27±0.09	94.61±0.17	96.4±0.06	86.86±1.67	87.02±1.23	86.92±0.78

A.2 Loss Function Results

The extended version of tables 5.6 and 5.7 with the standard deviation included. Values were averaged over 5 runs. The table for results on Spindle is split into two in order to fit on to the page.

All models were trained with a batch size of 100 and a learning rate of $5*10^{-5}$ with an exponential learning rate scheduler with gamma set as 0.9.

A.2.1 Spindle Data Set

Table A.3: Expansion of Table 5.6 with standard deviation over 5 runs

Loss	Overall Acc(%)	Cohort A Acc (%)
NLL	94.67	99.11
Dice	95.53±0.07	98.73±0.04
Focal	95.54±0.09	98.88±0.04
Transition	95.48±0.07	98.97±0.02

Table A.4: Expansion of Table 5.6 with standard deviation over 5 runs

Loss	Wake			NREM			REM		
	P	R	F1	P	R	F1	P	R	F1
NLL	91.54	98.61	94.94	98.6	93.21	95.83	89.87	79.81	84.54
Dice	95.18±0.13	96.95±0.07	96.06±0.04	97.69±0.04	95.0±0.14	96.32±0.06	84.12±0.29	90.18±0.35	87.05±0.29
Focal	94.61±0.39	97.38±0.17	95.98±0.13	97.89±0.09	94.94±0.15	96.39±0.08	86.18±1.56	88.01±1.98	87.05±0.21
Transition	94.59±0.24	97.73±0.11	96.13±0.07	98.17±0.07	94.68±0.2	96.39±0.07	85.23±0.21	88.22±0.46	86.7±0.16

A.2.2 Slumber Data Set

Table A.5: Expansion of Table 5.7 with Mean and Standard Deviation

Loss	Overall Acc(%)	Wake			NREM			REM		
		P	R	F1	P	R	F1	P	R	F1
NLL	90.4	91.29	92.84	92.06	90.3	88.07	89.17	78.23	77.84	78.03
Dice	86.96±0.2	91.49±0.06	86.03±0.28	88.68±0.17	86.0±0.36	88.44±0.11	87.2±0.22	54.27±1.26	85.95±0.36	66.52±0.84
Focal	87.85±0.13	91.33±0.07	88.33±0.16	89.8±0.15	85.64±0.21	87.99±0.08	86.8±0.17	66.04±0.86	79.63±0.28	72.2±0.71
Transition	87.82±0.34	91.6±0.07	87.89±0.24	89.71±0.22	87.19±0.4	88.29±0.21	87.73±0.36	57.95±1.42	84.71±0.42	68.82±1.07

A.3 SwinTsle Results

The extended version of tables 6.1with the standard deviation included. Values were averaged over 5 runs. The table is split into two in order to fit

on to the page.

All models were trained with a batch size of 100 and a learning rate of $5*10^{-5}$ with an exponential learning rate scheduler with gamma set as 0.9. The sequential model with the transformer was trained with a batch size of 4, with each entry containing 32 sequential epochs.

Table A.6: Extension of Table 6.1 with mean and standard deviation

Method	Overall Acc(%)	MF1(%)	Kappa(%)
Spindle	94.68 ± 0.18	91.8 ± 0.25	90.5 ± 0.33
Pretrained Swin-T	95.05 ± 0.23	91.4 ± 0.54	91.12 ± 0.41
+ Data Aug	95.46 ± 0.27	93.24 ± 0.56	91.91 ± 0.49
+ Dice Loss	95.56 ± 0.19	93.08 ± 0.53	92.05 ± 0.34
+ Transformer (SwinTsle)	96.03 ± 0.13	93.74 ± 0.54	92.92 ± 0.25

Table A.7: Extension of Table 6.1 with mean and standard deviation

Method	Wake			NREM			REM		
	P	R	F1	P	R	F1	P	R	F1
Spindle	91.57±0.43	98.55±0.15	94.93±0.18	98.5±0.12	93.32±0.44	95.84±0.18	90.37±0.82	79.55±1.09	84.61±0.51
Pretrained Swin-T	91.64±0.41	99.0±0.04	95.18±0.23	98.7±0.04	94.58±0.41	96.59±0.2	94.07±0.88	73.38±2.34	82.42±1.45
+ Data Aug	92.92±0.68	98.78±0.15	95.76±0.3	98.7±0.12	94.0±0.33	96.29±0.14	91.01±1.03	84.62±3.03	87.65±1.27
+ Dice Loss	92.66±0.34	98.95±0.05	95.7±0.19	98.68±0.15	94.6±0.33	96.59±0.12	94.31±0.86	80.73±2.87	86.96±1.41
+ Transformer (SwinTsle)	93.69±0.38	98.89±0.09	96.22±0.18	98.71±0.06	95.29±0.27	96.97±0.11	93.63±1.05	83.18±3.28	88.04±1.6

A.4 Sleep-EDF 20 Results

The extended version of tables 6.1with the standard deviation included. Values were averaged over 5 runs.

Both models were trained with a learning rate of $5*10^{-5}$ with an exponential learning rate scheduler with gamma set as 0.9. The sequential model with the transformer was trained with a batch size of 4, with each entry containing 32 sequential epochs, while Spindle was trained with a batch size of 100.

Table A.8: Predictions metrics for different models on the Sleep-EDF 20 data set with mean and standard deviation. Extension of Table 7.1

Model	Overall Metrics			Per-Class F1-Score				
	Accuracy	MF1	Kappa	W	N1	N2	N3	REM
Spindle	75.86±0.23	69.37±0.3	67.49±0.3	80.17±1.1	31.45±1.2	82.45±0.3	81.59±0.4	71.2±0.4
Swin Transformer	82.49±0.6	76.12±0.9	76.05±0.7	90.28±1.4	40.76±3.4	86.1±0.7	85.23±2.7	77.9±2.3
Swindle	84.73±0.4	79.13±0.7	78.86±1.0	91.88±1.4	46.18±4.7	86.64±0.7	85.94±2.5	83.65±2.1

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

DATA DRIVEN SLEEP STAGE CLASSIFICATION

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
SRIDHAR PRAKASH	AMBARISH

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
ZÜRICH, 20/11/2023	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.