Sleep Scoring in Alzheimer Mouse Models: Comparing Traditional and Spatial Approaches using Feed Forward and Convolutional Neural Networks

Oscar Atle Brovold, Eskil Grinaker Hansen, Håkon Ganes Kornstad

December 8, 2024

`Project GitHub Repository`

Abstract

It is established that sleep plays a role in Alzheimer's Disease, influencing processes that may impact disease progression. Sleep scoring, vital for understanding these mechanisms, is typically labor-intensive and performed manually. Automating sleep scoring with machine learning is therefore potentially a great tool for improving the efficiency in Alzheimer research. We build two machine learning models for sleep scoring in mice based on the same dataset of 14 different test mouse models. Datasets are from trials measuring brainwaves with Electrocorticography (ECoG) and muscle activities with Electromyography (EMG). We train our Feed Forward Neural Network (FFNN) in tandem with an extensive feature engineering, yielding an overall accuracy of 87%, as well as F1-scores of 67% and 48% on Intermediate Sleep (IS) and REM, respectively. Then we compare it to a CNN-model with two convolution layers, kernel size `(2, 2)` and `MaxPooling`, using ReLU as activation function in the dense layers. It is trained on plots of ECoG and EMG in the frequency domain, and yields an overall accuracy of 92%, and sleep stage individual F1-scores of 74% and 69% on IS and REM, respectively. We conclude that while FFNNs can effectively learn diverse tasks, the CNN outperforms it in our testing. We propose this is due to the CNN's ability to extract spatial patterns from frequency-domain data, making it better suited for sleep scoring.

INTRODUCTION

Sleep Scoring is used to determine the vigilance state of an organism, typically distinguishing between states like *wakefulness*, *REM* (Rapid Eye Movement), and *non-REM* sleep. Traditionally, sleep scoring in mammals is performed manually by an expert, by comparing the dominance of certain brain waves across different frequency bands. This is a repetitive and labour intensive work. An automated sleep scoring algorithm would therefore save time and free up the researchers to do other work.

GliaLab is a Norwegian research group at the Medical Department of The University of Oslo (UiO). It conducts research on the role of glial cells, particularily **astrocytes**. It is thought that these cells are important in slowing down the progression of Alzheimer's Disease (AD) and Parkinson's Disease, by clearing particles that contribute to amyloid placque formation in the Hippocampus. [1] Astrocytes operate mainly in the area around the blood vessels, the **perivascular space** [2]. Observing the brain of test mice with AD during sleep, *GliaLab* investigates how the vascular contraction in each vigilance state contribute to opening up the perivascular space, thus facilitating the astrocytes in their cleaning work. [2]

There have been several experiments on using increasingly advanced machine learning models for sleep scoring in recent years. [3] [4] This can be credited to the advancements in hardware, and the access to increasingly extensive health data. [3] Being three bachelor students in the subject FYS-STK3155 *Applied Machine Learning* at the University of Oslo, we are grateful to be granted access to the data from *GliaLab*'s mouse trials.

We aim to implement and then compare a standard Feed Forward Neural Network (FFNN), with a slightly simplified version of one of the most successful Convolutional Neural Network (CNN) models, namely *SPINDLE* [5], trained on frequency plots. For both networks we also make use of feature enginering to further enhance the data according to domain knowledge. We focus on using precision, accuracy and F1-score as our main metrics, both globally and per sleep stage, for comparing the FFNN and CNN models' performance.

THEORETICAL BACKGROUND

Trials in Mouse Models at *GliaLab*

An opening in the mouse models' skull revealing the brain's blood vessels is observed under a two-photon microscope. This enables the researchers to find correlations of sleep patterns with vascular contractions, and the accumulation of the AD-related amyloid placque. If sleep plays a role in the drainage of the agents that accumulate amyloid placque, it would be of great value to assess the sleep stages of the mouse in trial. [2]

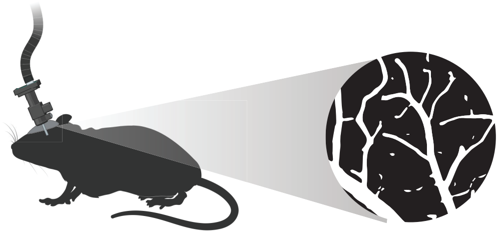


Figure 1: A two-photon microscope photographs the brain of a mouse, showing the vascular contractions during different sleep stages. By attaching Electrocorticography (EcOG)-electrodes on the model, brainwave activity can be measured in real time. In parallel, electromyography (EMG) electrodes are mounted on the model's neck musculature, tracking the mouse model's movement. Both the EcOG and EMG data is used in manual sleep scoring. **Illustration of mouse:** *BioRender*, used with an Open Source permission.

SLEEP SCORING

Electrocorticography (ECoG)

ECoG is a method for measuring electrical activity in the brain. Unlike EEG, where electrodes are placed *on* the scalp, ECoG electrodes are placed directly on the surface of the cerebral cortex through surgery. This provides ECoG with better spatial resolution and stronger signal strength compared to EEG, making it less susceptible to artifacts such as noise. [6]

Electromyography (EMG)

The electromyogram (EMG) measures electrical activity in muscles, and thereby tracking movement across different sleep stages. An electrode is mounted on the neck musculature to infer changes in motor activity associated with wakefulness, non-rapid eye movement (NREM), and rapid eye movement (REM) sleep. During wakefulness, EMG signals typically exhibit high-amplitude, irregular bursts due to active movement. In contrast, NREM sleep is characterized by a reduction in muscle tone, with lower-amplitude, more stable EMG signals. REM sleep is marked by a near-complete loss of muscle tone, resulting in minimal or absent EMG activity. [7]

Machine Learning based Sleep Scoring Setups

Machine learning for sleep scoring has been explored for decades. However, it remains challenging to develop models that perform well across the diverse genetic backgrounds of mouse models and their varied research environments. Similarly, these models often fail to generalize to other mammals, making them unsuitable for human sleep scoring. Despite these challenges, the significant amount of time spent annually by research groups on manual sleep scoring makes it worthwhile to continue pursuing automated approaches. [8]

Our CNN model is inspired by *SPINDLE*, a model developed at ETH Zurich in 2019, and collaborating institutions. *SPINDLE* employs a CNN architecture, complemented by a hidden Markov model (HMM), to generalize sleep stage predictions across experimental settings, species, and labs. The study obtains an impressive overall accuracy of 93-99%. For the hard-to-predict REM-stage, they offer a F1-score ranging between 85-98%, dependent on animal cohorts and the addition of HMM. [5]

Feed Forward Neural Network (FFNN)

A Feed Forward Neural Network (FFNN) is a basic artificial neural network. Information flows sequentially from the input layer, through a series of hidden layers, and finally to the output layer. FFNNs are highly versatile and often deliver strong results across a wide variety of machine learning tasks. For the underlying theory and training methodology of FFNNs, please refer to the section on Feed Forward Neural Networks in our Report 2 [9].

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized class of neural networks, well-suited for analyzing visual data. Unlike standard feed-forward neural networks [9], CNNs incorporate convolution layers that are useful for recognizing patterns, such as edges and textures. Following the convolutional layers, the CNN further applies fully connected layers, similar to traditional feed-forward networks. This specialized structure aims to effectively solve tasks such as image recognition, object detection, and other applications involving spatial data. [10]

Each convolutional layer consists of three stages; a `convolution stage`, a `detection stage` and finally a `pooling stage`.

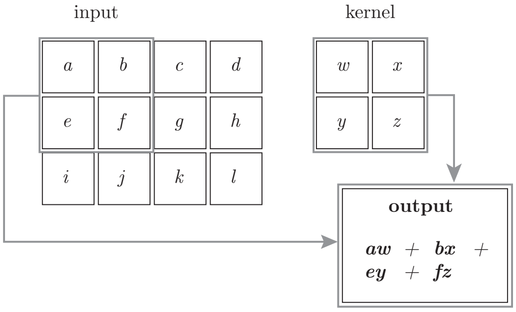


Figure 2: The convolution stage, convolving a 3×4 image with a 2×2 kernel. (Freely after Illustration 9.1 in Goodfellow et al.[10])

The convolution stage is illustrated in Figure 2, and is for most machine learning purposes mathematically defined as:

$$S(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Here I is the two-dimensional image and K is the kernel/filter (a weighted matrix). $S(i,j)$ is the output, which is sometimes referred to as the **feature map** [10]. This process is useful for extracting specific features of I. The next stage of the convolutional layer is a **detection stage** — here we apply a non-linearity, e.g the rectified linear unit (ReLU). The final step is the so-called **pooling stage**, implemented to downsample the feature map. This is done by calculating a "summary statistic" for a certain location based on nearby outputs. This speeds up calculation and possibly reduces overfitting. Some common pooling strategies consists of **max pooling**, **average pooling** and L^2 **pooling**. Finally, the output of the last convolutional layer is flattened and sent through a series of **fully connected** ("dense") layers, yielding the final prediction. [10]

Evaluation Methods

When dealing with classification tasks there are several methods to evaluate a model's predictions on a test set of data.

		predicted	
		`0`	`1`
actual	`0`	True Negative	False Positive
	`1`	False Negative	True Positive

Figure 3: The confusion matrix. In this example, `0` depicts "Negative" and `1` "Positive".

A **confusion matrix** (Figure 3) gives a visual representation of how the model is predicting correct or not, including the mistakes. From the confusion matrix it is possible to calculate different scores, such as precision, recall, F1 and accuracy. When classifying a dataset using a set of labels, we evaluate the performance by analyzing the following metrics for each class: the number of correct predictions for the class (true positives/TP), the number of instances incorrectly predicted as belonging to the class (false positives/FP), and the number of instances belonging to the class that were incorrectly predicted as another class (false negatives/FN).

Precision is a metric that calculates the the amount of true positives divided by all classifications of the same class. This method works well for evaluating imbalanced classes, as it provides the model correctness for each class.

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

Recall gives a ratio of correct predictions compared to the missed cases. As with precision, this method works well for imbalanced classes because it identifies objects of the target label.

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

Lastly there is **accuracy** and **F1-score**. Accuracy measures all correct predictions across all classes against all test cases. The method gives an overview over the model's entire ability to predict correctly, however it can be misleading when having class imbalance. F1-score takes both recall and precision into consideration, using their harmonic mean. F1-score is calculated as:

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

[11]

PyTorch

PyTorch is an open-source library developed by Meta AI. In PyTorch, neural networks can be built with `torch.nn`, optimizers are available through `torch.optim`, and batch processing is handled by the `DataLoader` class. [9] [12]

PREPROCESSING THE DATA

The Dataset Provided

The dataset obtained from *GliaLab* comprises multiple trials from various mouse models and has been preprocessed using an algorithm based on the **Fast Fourier Transform**. This preprocessing converts ECoG values into the frequency domain, providing power values for the `delta`, `theta`, `sigma`, and `beta` bands for each epoch in the trial, with each epoch spanning 3 seconds. Additionally, the dataset includes EMG power values for each epoch and human sleep scoring labels, categorized as `unknown`, `Awake`, `IS` (intermediate sleep), `NREM`, or `REM`.

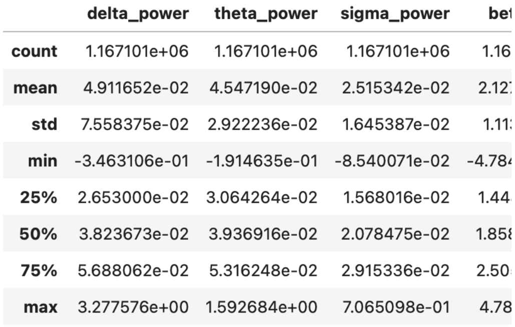
The trials are organized into multiple files, each named in the format `trial_#_mouse_genome_m#.csv`. Each file contains columns for `delta_power`, `theta_power`, `sigma_power`, `beta_power`, `emg_power` and `sleep_episode`. We implement a `DataLoader` class to concatenate data from various mouse models, creating `train_data`, `validation_data`, and `test_data` (Figure 7). It is crucial to ensure that data from a single mouse model is not included in both the training and test sets. This is to prevent artificially high accuracy when testing.

Removing the `unknown` labels

The sleep scoring trials include multiple stretches labeled as `unknown`, indicating epochs without a class label, where the sleep stage has not been manually scored. All instances of `unknown` are removed from the dataset, as they do not contribute meaningful information for our training.

Inspecting the Dataset

We start by setting up a Jupyter Notebook (found in the `Project GitHub Repository`) to gain more insight into the data at hand. Pandas' `.describe()` function [13] (Figure 4) provides us with the mean, standard deviation, minimum and maximum values, as well as the quartiles. This gives us a first glance at the data, informing us about possible artifacts (i.e., outliers) that need to be addressed.



	delta_power	theta_power	sigma_power	bet
count	1.167101e+06	1.167101e+06	1.167101e+06	1.16
mean	4.911652e-02	4.547190e-02	2.515342e-02	2.12
std	7.558375e-02	2.922236e-02	1.645387e-02	1.11
min	-3.463106e-01	-1.914635e-01	-8.540071e-02	-4.78
25%	2.653000e-02	3.064264e-02	1.568016e-02	1.44
50%	3.823673e-02	3.936916e-02	2.078475e-02	1.85
75%	5.688062e-02	5.316248e-02	2.915336e-02	2.50
max	3.277576e+00	1.592684e+00	7.065098e-01	4.78

Figure 4: Facsimile of Pandas' `.describe()` function printout. Please refer to the Jupyter Notebook for all the details.

We observe that there are no significant outliers in the data, and the standard deviations within the individual power bands are low. This suggests that the "raw" power values are well-suited for scaling with Z-score.

Class Weights

Examining the class balance using `.value_counts()` [13] reveals what we expected: there is a significant imbalance in the `REM` class, which could pose a potential risk of underpredicting this class.

Class weights are then calculated. PyTorch provides a mechanism to handle this imbalance by allowing its cost function to accept a **weight tensor**, enabling the model to assign different levels of importance to each class during training. This helps to ensure that underrepresented classes, such as `REM`, are not overlooked and contributes to a more balanced model performance.

Interdependency of the features

Next, we aim to gain insights into the correlations between the different brainwaves. We do this by plotting a `pairplot` (please refer to the Jupyter Notebook) [14] as well as a standard correlation matrix. In the matrix, linear dependencies between the brainwaves and EMG classes are revealed as higher values (Figure 5). We note that the `theta`, `sigma`, and `beta` bands show high correlation values, indicating they are similar predictors for the dataset.

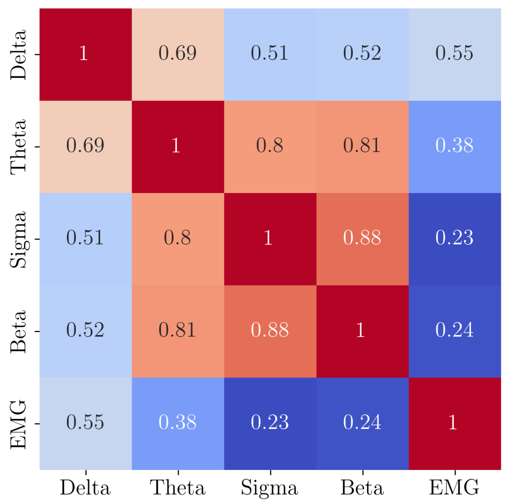


Figure 5: Correlation matrix of “raw” power values, without feature engineering. Low values indicate low correlations, and vice versa.

Feature Engineering: Power Ratios

We plan to run training on a dataset with all the raw data. However, in an effort to combat the high interdependency of the brainwaves, we create alternative datasets. In addition to this, we also experiment with a dataset where we add the power ratios.

If some features in the data set are interdependent, they can be to replaced them with ratios of power values. If these power ratios generate added value for our data, it will be visible in the pairplot and correlation matrix. We once again run the dataset through Seaborn’s `pairplot` (again, please refer to the Jupyter Notebook) and create a correlation matrix. Comparing Figure 5 and Figure 6, we clearly see significantly less interdependency between the power bands. Since this could lead to better and more stable training, we choose to add the ratios to the raw values. Specifically, we find that `delta_theta_ratio`, `sigma_beta_ratio`, `theta_sigma_ratio`, as well as `emg_delta_ratio` could prove beneficial to the training.

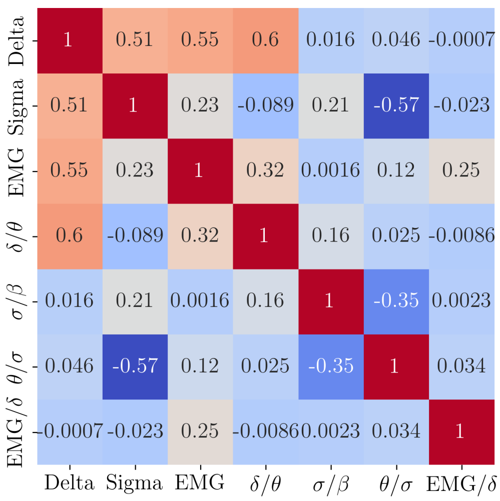


Figure 6: Correlation matrix after feature engineering, where the power band ratio `delta_theta, sigma_beta, theta_sigma, emg_delta` have been inserted, replacing some of the possibly redundant “raw” bands.

Scaling the Data

Log Power

To mitigate the impact of outliers (i.e., unusually high power values) in the dataset, we apply a logarithmic transformation to all power values in the training, validation, and test datasets, as suggested by [5]. To handle small values and avoid the issue of log(0), a safety offset of `1` is added to all power values before applying the transformation.

Standardizing with z-score

Z-score standardization is performed to center the data around $\mu = 0$ with a standard deviation of $\sigma = 1$. This ensures that all features have the same scale, improving model convergence and stability during training. The standardization is applied to all datasets using the formula:

$$z = \frac{x - \mu}{\sigma}$$

where x is the original value, μ is the mean, and σ is the standard deviation.

Splitting into training, validation and test sets

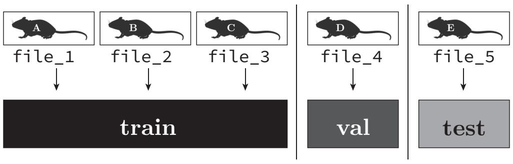
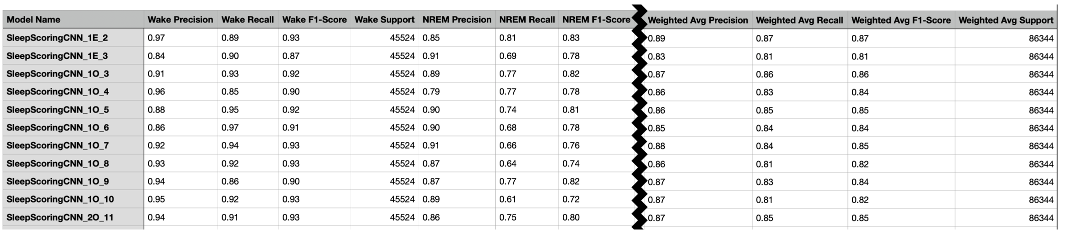


Figure 7: Splitting the data for training, validation and testing based on the mouse individuals.

The `MouseLoader` class is set up to load data for individual mice, using the structure of their filenames. We are then able to partition these files into training, validation, and test sets, ensuring that no mouse appears in multiple sets. This *file-level approach* (Figure 7) maintains data integrity while preserving the biological uniqueness of each mouse.

Subsequently, the `CNNContainer` and `FFNNContainer` classes split mouse data into fixed-size windows for training, validation, and testing. The parameter `window_size` determines the number of time steps included in each window. The `sleep_episode` label for the window is assigned based on the middle time step `(i + window_size // 2)`. Each window and its label is then stored as a tuple `(window_data, label)` in the separate containers `_train_container, _val_container, _test_container`.

GENERAL FRAMEWORK FOR TRAINING



Model Name	Wake Precision	Wake Recall	Wake F1-Score	Wake Support	NREM Precision	NREM Recall	NREM F1-Score	Weighted Avg Precision	Weighted Avg Recall	Weighted Avg F1-Score	Weighted Avg Support
SleepScoringCNN_1E_2	0.97	0.89	0.93	45524	0.85	0.81	0.83	0.89	0.87	0.87	86344
SleepScoringCNN_1E_3	0.84	0.90	0.87	45524	0.91	0.69	0.78	0.83	0.81	0.81	86344
SleepScoringCNN_1O_3	0.91	0.93	0.92	45524	0.89	0.77	0.82	0.87	0.86	0.86	86344
SleepScoringCNN_1O_4	0.96	0.85	0.90	45524	0.79	0.77	0.78	0.86	0.83	0.84	86344
SleepScoringCNN_1O_5	0.88	0.95	0.92	45524	0.90	0.74	0.81	0.86	0.85	0.85	86344
SleepScoringCNN_1O_6	0.86	0.97	0.91	45524	0.90	0.68	0.78	0.85	0.84	0.84	86344
SleepScoringCNN_1O_7	0.92	0.94	0.93	45524	0.91	0.66	0.76	0.88	0.84	0.85	86344
SleepScoringCNN_1O_8	0.93	0.92	0.93	45524	0.87	0.64	0.74	0.86	0.81	0.82	86344
SleepScoringCNN_1O_9	0.94	0.86	0.90	45524	0.87	0.77	0.82	0.87	0.83	0.84	86344
SleepScoringCNN_1O_10	0.95	0.92	0.93	45524	0.89	0.61	0.72	0.87	0.81	0.82	86344
SleepScoringCNN_2O_11	0.94	0.91	0.93	45524	0.86	0.75	0.80	0.87	0.85	0.85	86344

Figure 8: A facsimile of the `.csv` file, where information about all the models are saved. In addition, our model saves `.pth` files, to make the trained networks easy to reuse for prediction. Please refer to the file `models.csv` in the `models`-folder.

To begin with, we establish a framework for saving the performance of each trained model. The process starts with defining the hyperparameters for the network using PyTorch. We then store our best model during training based on its validation set accuracy. The best model from each training is stored in a `.pth`-file and we can load it whenever needed. Along with the model, we store all of its hyperaparameters in a `.txt`-file and each of the formerly discussed metrics in a `.csv`-file (Figure 8). With this we are prepared to compare different models, based on a preferred metric.

We begin with preliminary heuristic tuning to establish a baseline for further optimization. During this step, we conduct testing on both the network structure and feature configuration.

For the **network structure**, we experiment with various architectures, including the number of layers, the number of parameters per layer, activation functions, and optimizers. For the **feature configuration**, we test different window sizes and apply more fundamental feature engineering techniques, as discussed earlier.

From these preliminary tests, we observe that hyperparameter tuning has minimal impact on the weighted precision. However, altering window size and feature engineering lead to notable variability in the accuracy distribution across the model's sleep stage classes. These findings are detailed in the accompanying `.csv` and `.txt`, and will be further discussed below.

We use **cross entropy loss** [9] [15] as cost/loss function for training all models.

TRAINING WITH A FEED FORWARD NEURAL NETWORK

Based on the evaluation in the `.csv` file, we select a feedforward neural network model with a good initial performance. Specifically we choose model `NNModel1O_45.pth`. Using the `.txt` file, we then extract the specific parameters of this model and conduct further training while experimenting with variations in its structure.

Effects of Network Structure

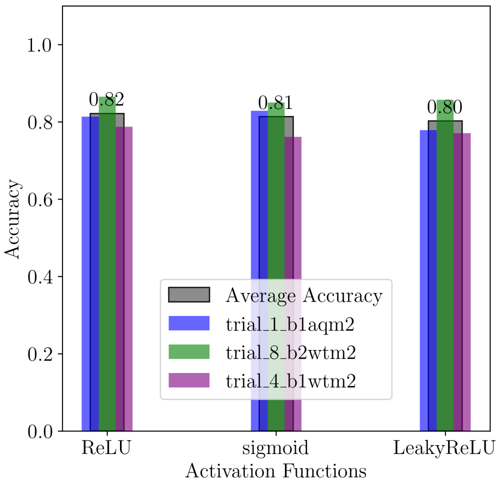


Figure 9: Accuracy as a function of different mice and activation functions. The following hyper parameters were used: `2 hidden layers. Layer sizes: 64 and 11. Window size: 15. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.`

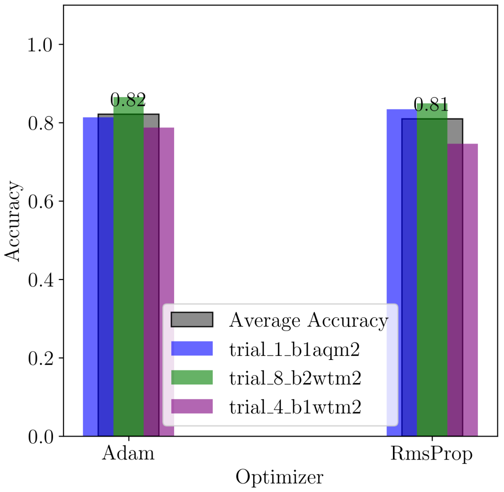


Figure 10: Accuracy score as function of different mice and optimizers. Hyper parameters: `2 hidden layers. Layer sizes: 64 and 11. Window size: 15. Optimizer parameters: Adam: Rho1=0.9, Rho2=0.999, LR=0.001, RMSProp: Rho=0.99, LR=0.01. Cross Entropy Loss.`

As shown in Figure 9 and Figure 10, our observations align with the results from the preliminary testing: additional network structure tuning has minimal impact on model performance. We propose this is due to the inherent flexibility of feed forward neural networks. Provided the network structure is reasonably designed and sufficient training data is available, the model converges effectively towards a minimum. In other words, the models tend to reach a **performance plateau** after only a few epochs of training.

This suggests that while hyperparameter tuning can optimize performance to a certain extent, further significant improvements are likely dependent on more fundamental changes in the machine learning approach, such as feature engineering, alternative architectures, or improved data representation.

Effects of window size

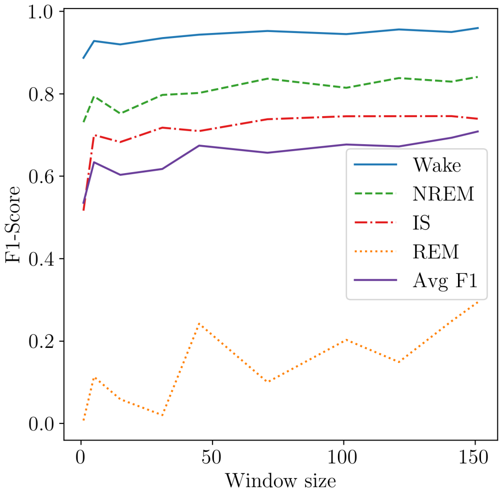


Figure 11: F1 score for all sleep stages and its averages as function of window sizes, for the specific mouse individual `B2WTM2` Network architecture: `2 hidden layers. Layer sizes: 64 and 11. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.`

In the preliminary testing, we observed greater variance in model performance when altering the window size. This trend was further confirmed during the evaluation phase, as shown in Figure 11. Selecting a model with a larger window size (approximately 50, corresponding to a time period of over 150 seconds) resulted in improved overall F1-score.

For all sleep stages except REM-sleep, the models stabilize as we increased the window size. However, for the REM sleep stage we can observe larger variance as we vary the window size. We attribute this to the rapid fluctuations characteristic of REM sleep, where windows size variation could include irrelevant information, potentially diluting the model's ability to identify specific patterns.

To address this, we propose a balanced approach that optimizes the tradeoff between precision and sleep stage-specific accuracy.

Feature Engineering and Power Ratios

To overcome the performance plateau, we explore adding additional power ratios to the model through feature engineering. Using the preprocessing framework outlined earlier, we selectively include features with low correlation to minimize redundancy. In specific, we use `delta_theta_ratio`, `sigma_beta_ratio`, `theta_sigma_ratio`, `emg_delta_ratio`, `delta_power`, `sigma_power` and `emg_power` as features.

As illustrated in Figures 12 and 13, the feature-engineered model (`NNModel10_51`) demonstrates improvements over the original FFNN model in classifying sleep stages. Notably, the new model performs significantly better in predicting REM sleep and maintains comparable accuracy in classifying Wake and NREM stages. However, it exhibits a decline in its ability to classify IS sleep accurately. Given this tradeoff, the choice between models should be guided by the importance of accurately classifying each sleep stage for the specific application.

Evaluation of model `NNModel10_51`

With the training stage complete, we can draw several conclusions. Among the factors considered, adjustments to the window size and feature engineering has the most significant impact on model performance. In contrast, changes to hyperparameters yield minimal improvements.

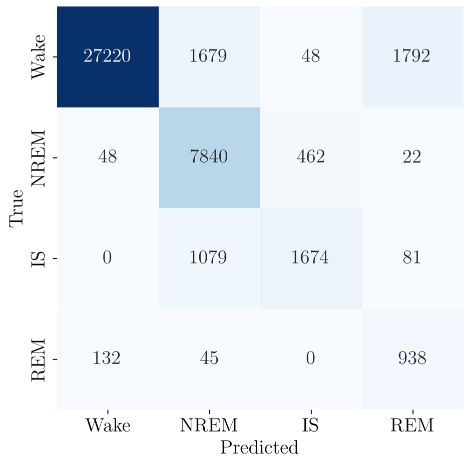
The confusion matrix in Figure 12 along with the classification report from model `NNModel10_51` tested on test mouse: `trial8_b2wtm2` outlines its overall performance and challenges across the four sleep stages: Awake, NREM, IS and REM. The overall test accuracy of 87% reflects strong general performace. However, the report also reveals its challenges, particulary with predicting less represented sleep stages (REM, IS).

For the Wake stage, the model performs exceptionally well, with a precision of 99% and f1-score of 94%. The model is able to classify most Wake instances correctly. However, a small portion is misclassified as primarly REM and NREM, this is reflected by the recall of 89%. Similarly, the model performs well on NREM, which is the class with the second most support. It achieves general high recall and F1-score, 94% and 82% respectively. However, the lower precision of 74% implies some misclassifications, where other stages, especially IS and REM, are mistakenly labeled as NREM.

In contrast, predicting Intermediate Sleep proves to be a significant challenge for the model. With a precision of 77% and a recall of 59%, the F1-score of 67% reflects the difficulty the model has in accurately identifying IS instances. Many true IS cases are misclassified as NREM or Wake, indicating room for improvement in capturing the unique features of this stage.

The REM stage classification has significantly improved from the non-feature engineered models. However it still presents difficulties. The high recall of 84% shows that the model successfully identifies most REM cases; however, the precision drops to 33%, resulting in an F1-score of 48%. This indicates that, while the model is sensitive to REM instances, it often misclassifies other stages as REM — particularly wake.

By feature engineering we are able to boost the REM stage classification, but it comes at a cost: a significant drop for the IS sleep stage.



	Wake	NREM	IS	REM
Wake	27220	1679	48	1792
NREM	48	7840	462	22
IS	0	1079	1674	81
REM	132	45	0	938

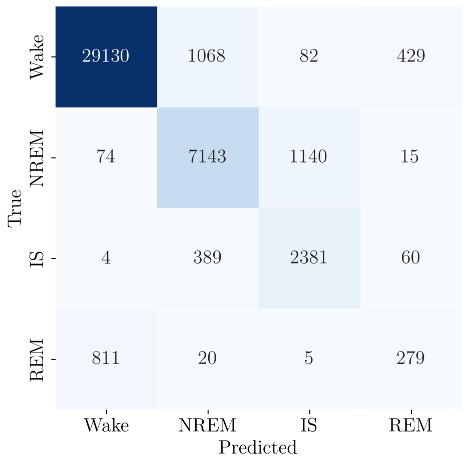
Figure 12: Model `NNModel10_51`**with feature engineering.** Confusion matrix for FFNN testing on feature engineered mouse data (1) for mouse model `B2WTM2, trial 8`. Hyper parameters: `2 hidden layers. Layer sizes: 64 and 11. Window size: 121. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.`

```
Classification report:
           prec.  rcl   f1  support

    Wake   0.99  0.89  0.94  30739
    NREM   0.74  0.94  0.82   8372
      IS   0.77  0.59  0.67   2834
     REM   0.33  0.84  0.48   1115

accuracy               0.87  43060
macr avg   0.71  0.81  0.73  43060
 wgt avg   0.91  0.87  0.88  43060

Accuracy of test data netw.:  87 %
Accuracy for class: Wake  is 88.6 %
Accuracy for class: NREM  is 93.6 %
Accuracy for class: IS    is 59.1 %
Accuracy for class: REM   is 84.1 %
```



	Wake	NREM	IS	REM
Wake	29130	1068	82	429
NREM	74	7143	1140	15
IS	4	389	2381	60
REM	811	20	5	279

Figure 13: Model `NN_Model_1_19` without feature engineering. Confusion matrix for FFNN testing on "clean" mouse data (2) for mouse model `B2WTM2, trial 8`. Hyper parameters: `2 hidden layers. Layer sizes: 64 and 11. Window size: 121. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.`

```
Classification report:
           prec.  rcl   f1  support

    Wake   0.97  0.95  0.96  30709
    NREM   0.83  0.85  0.84   8372
      IS   0.66  0.84  0.74   2834
     REM   0.36  0.25  0.29   1115

accuracy               0.90  43030
macr avg   0.70  0.72  0.71  43030
 wgt avg   0.91  0.90  0.90  43030

Accuracy of test data netw.:   90 %
Accuracy for class: Wake  is 94.9 %
Accuracy for class: NREM  is 85.3 %
Accuracy for class: IS    is 84.0 %
Accuracy for class: REM   is 25.0 %
```

TRAINING THE *SPINDLE*-INSPIRED CNN-MODEL

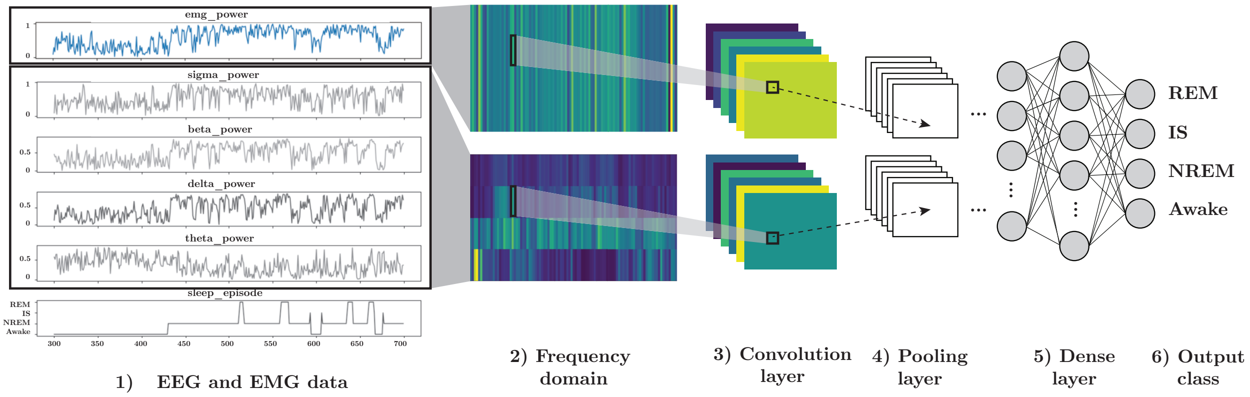


Figure 14: 1) Power data for EEGs, as well as EMG received from *GliaLab*, with the corresponding labels (`sleep_episode`). 2) Treating these frequency plots like images make the inputs in a convolutional neural network. 3) Convolution layer. 4) Pooling layer. 5) Fully connected (dense) layer. 6) Predicted classes (`REM, IS, NREM` or `Awake`)

In another effort to combat the performance plateau, specifically related to the underrepresented classes REM-sleep and IS, we change the type of artifical neural network. Inspired by SPINDLE [5] we now created a Convolutional Neural Network (CNN) using Pytorch.

Creation of Images

As a starting point, we consider the four ECoG power values and one EMG power value for a given 3-second epoch, shown in stage 1 of Figure 14. By stacking the ECoG power values vertically, we form a (4×1)-pixel image, which serves as the foundation for one of the CNN input images.

To incorporate the EMG power value, we construct a second image by replicating the EMG power value four times in a vertical stack. This results in another (4×1)-pixel image, providing the basis for the second input to the CNN.

To capture temporal context from surrounding sleep episodes, we extend this approach using window sizes. A window size of n produces two $(4 \times n)$ images. This is depicted in stage 2 of Figure 14.

Training Evaluation

In an effort to deepen our analysis, we heuristically experiment with CNN-specific architectures. A wide range of tunable parameters are tested, including pooling configurations such as kernel sizes and the number of pooling layers. For the convolution stage, we explore variations in kernel sizes, the number of layers, and the inclusion of padding.

Our experiments reveal a consistent trend similar to that observed with the Feed Forward Neural Network (FFNN): specific structural tuning has had minimal impact on performance. Ultimately, we select model `SleepScoringCNN_10_7` for further testing, as it demonstrates consistent results.

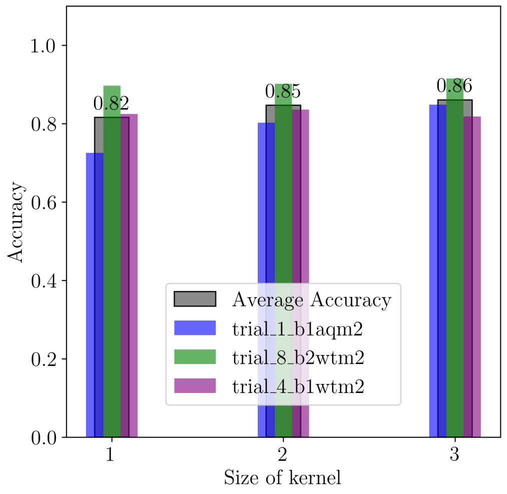


Figure 15: Accuracy given differentkernel sizes in CNNs tested on three different mouse models, `b1aqm2`, `b2wtm2`, `b1wtm2`. Hyper parameters: `2 convolutional layers. MaxPooling. Stride: 1. Padding: 2. (Layer 2 has no padding). Dense layers: 64, 16. Activation fn: ReLU. Window size: 41. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.`

Figure 15 illustrates the network accuracies as functions of convolutional kernel sizes. It further reinforces the idea that additional tuning yields minimal impact. For a more comprehensive overview of the tested parameters, please refer to the accompanying `.csv` and `.txt files`.

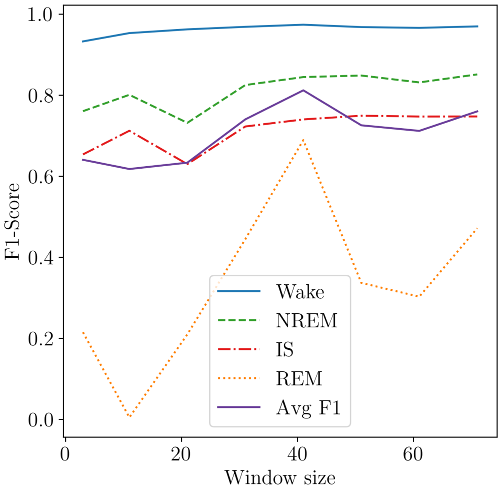


Figure 16: F1 score of varying window sizes as a function of different sleep stages in CNNs on mouse model `b2wtm2`. Hyper parameters: `2 convolutional layers. MaxPooling. Kernel size: (2x2). Stride: 1. Padding: 2. (Layer 2 has no padding). Dense layers: 64, 16. Activation fn: ReLU. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.`

The impact of window size is even more pronounced in the case of CNNs, as illustrated in Figure 16. Similar to the FFNN results, the Wake, NREM, and IS stages show only minor improvements with adjustments to window size, with a slight increase in F1-score observed for larger windows. In contrast, the REM sleep stage exhibits more significant variations. Larger window sizes initially improve the F1-score for REM sleep, but Figure 16 reveals a clear tradeoff: when the window size exceeds 40, the F1-score for REM sleep begins to decline. We attribute this to the rapid fluctuations that characterize REM sleep. Simiarly to the behaviour in FFNN: larger window sizes may incorporate irrelevant information, diluting the model's ability to identify key patterns unique to this stage.

Feature Engineering

As with the FFNN, we also explore the impact of manual feature extraction for the CNNs. Specifically, we attempt to generate a third input image incorporating specific power ratios. Figure 18 shows a confusion matrix and an additional classification report for a model with `beta_delta_ratio`, `delta_beta_ratio`, `beta_theta_ratio` and `theta_beta_ratio` as the third image. Choosing the particular ratios are done with inspiration from the report *Sleep-awake Classification using EEG Band-power-ratios and Complexity Measures* [16]. It suggests choosing ratios of high frequency to low frequency. In contrast to FFNNs, our manual feature engineering for CNNs do not lead to improved performance. The models trained with power ratios particularly struggle with the REM/Wake classification, a challenge that is outlined in the same mentioned report. Our testing suggests that the addition of power ratios, as a third input image, may not sufficiently address classification difficulties in these stages.

Concluding Training for Convolutional Neural Networks

Notably, we finally find the new models to outperform the FFNNs, yielding better overall validation accuracy and better performance on sleep stage-specific metrics.

For CNNs we achieved better results with one of the fine-tuned models. In specific, we find model `CNNModel_41W27` to provide good overall precision, also on stage-specific metrics. We will therefore evaluate this model further.

Based on the confusion matrix in Figure 17 along with the classification report for model `CNNModel_41W27` tested on test mouse `trail8_b2wtm2` the model demonstrates strong performance, with an accuracy of 92%. This shows robustness in classifying sleep stages, although some class specific areas reveal room for improvement.

The model performs exceptionally well in identifying the Wake stage, achieving a precision of 98%, recall of 96% and an F1-score of 97%. Its success in classifying this stage is likely aided by the large number of Wake samples in the dataset (support: 30818). Similarly, NREM classification is solid, with a precision of 87%, recall of 82% and F1-score of 84%. However, the confusion matrix reveals that the model misclassifies a significant part of NREM as IS, 1322 instances in specific.

The model's performance on IS-stage is more mixed. It achieves a high recall of 87%, suggesting effective detection when the stage is actually present. However, the precision drops to 64%, indicating frequent misclassifications, particulary with NREM. Furthermore, the REM-stage remains the most challenging for the model, with a precision of 65%, recall of 74% and F1-score of 69%. These metrics indicate difficulty in distinguishing REM from other stages, likely due to its smaller representation in the dataset (support: 1,115) and overlapping features with other stages, especially “Wake”.

Based on this analysis, we suggest that further improvements could be seen with increased training samples for underrepresented stages like IS and REM. Additionally, refined feature engineering methods to better capture the transitional characteristics of these stages may help reduce confusion.

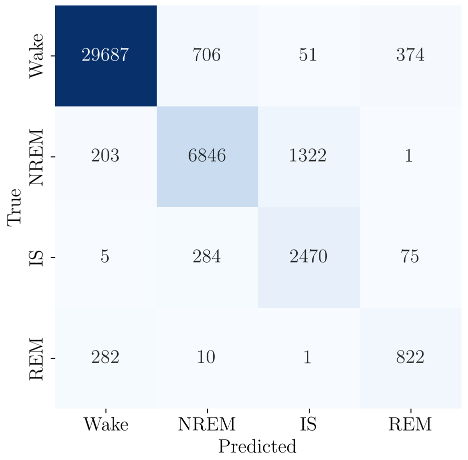


Figure 17: Without feature engineering. Confusion matrix for **CNN** testing on mouse model B2WTM2, trial 8 Hyper parameters: 2 convolutional layers. MaxPooling. Stride: 1. Padding: 2. (Layer 2 has no padding). Dense layers: 64, 16. Activation fn: ReLU. Window size: 41. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.

```
Classification report:
           prec.  rcl   f1  support

    Wake   0.98  0.96  0.97  30818
    NREM   0.87  0.82  0.84   8372
      IS   0.64  0.87  0.74   2834
     REM   0.65  0.74  0.69   1115

accuracy               0.92  43139
macr avg   0.79  0.85  0.81  43139
 wgt avg   0.93  0.92  0.93  43139

Accuracy of test data netw.:   92 %
Accuracy for class: Wake  is 96.3 %
Accuracy for class: NREM  is 81.8 %
Accuracy for class: IS    is 87.2 %
Accuracy for class: REM   is 73.7 %
```

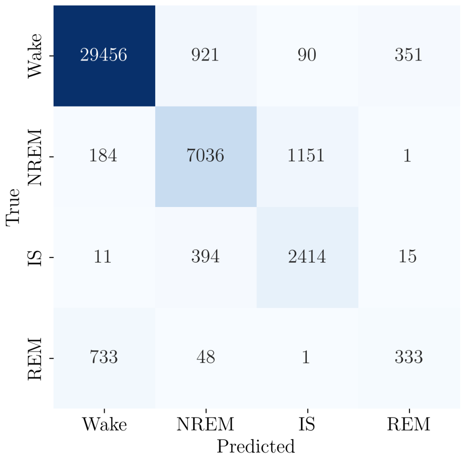


Figure 18: With feature engineering, including a third image of power_ratios. Confusion matrix for **CNN** testing on mouse model B2WTM2, trial 8 Hyper parameters: 2 convolutional layers. MaxPooling. Stride: 1. Padding: 2. (Layer 2 has no padding). Dense layers: 64, 16. Activation fn: ReLU. Window size: 41. Optimizer: Adam (Rho1=0.9, Rho2=0.999). Learning Rate=0.001. Cross Entropy Loss.

```
Classification report:
           prec.  rcl   f1  support

    Wake   0.97  0.96  0.96  30818
    NREM   0.84  0.85  0.84   8372
      IS   0.66  0.85  0.74   2834
     REM   0.48  0.30  0.37   1115

accuracy               0.91  43139
macr avg   0.74  0.74  0.73  43139
 wgt avg   0.91  0.91  0.91  43139

Accuracy of test data netw.:   90 %
Accuracy for class: Wake  is 95.6 %
Accuracy for class: NREM  is 84.0 %
Accuracy for class: IS    is 85.2 %
Accuracy for class: REM   is 29.9 %
```

CONCLUSION

Based on our findings, we reaffirm the established notion that Feed Forward Neural Networks (FFNNs) are generally capable of learning a wide range of tasks, but often perform poorly in domain-specific scenarios. Our FFNN model achieved an accuracy of 87% and struggled with the most challenging class, REM sleep, yielding an F1-score of only 48%. Basic preprocessing steps such as reducing redundancies and scaling enabled us to find meaningful stage specific improvements, particularly for REM-sleep.

Motivated by this, we implemented a Convolutional Neural Network (CNN) inspired by SPINDLE. This architecture outperformed the FFNN, achieving an accuracy of 92% and notably better REM classification for the correct window sizes (69% for a window size of 41). However, we suspect that the limitations of our dataset — specifically its low-resolution frequency spectrum, binned into power bands — may have contributed to our comparatively lower performance relative to SPINDLE.

With more time and a deeper understanding of Digital Signal Processing, we would have extended our work by applying our own Fourier Transform to convert the raw data into the spectral domain. Additionally, we would have incorporated a Hidden Markov Model (HMM) to better capture the temporal dependencies between different sleep phases. In the preliminary work for this paper, we also began experimenting with a Recurrent Neural Network (RNN) featuring a bidirectional LSTM structure, which *could* further enhance predictions by accounting for the sequence of sleep stages over time.

In the future, we would wish to explore RNNs in greater depth, alongside integrating an HMM for post-processing. Another option would be to deepen our domain knowledge in sleep scoring further. This would allow us to focus more effectively on advanced feature engineering techniques and consider approaches like Principal Component Analysis (PCA). Combined with logistic regression and potentially augmented by an HMM, such a pipeline could offer a simpler yet more efficient alternative for sleep stage classification. [17]

On a final note: having reviewed numerous reports highlighting how machine learning models often struggle with a relatively simple classification task such as sleep scoring, we are prompted to consider a more philosophical question: What is it that a human sleep scorer sees in the data—and in the *Alzheimer's Disease* mouse model—that a machine cannot? As we advance in this field, this question could indeed be worth remembering.

ACKNOWLEDGEMENTS

We would like to thank *GliaLab* at the Medical Department of The University of Oslo (UiO) for providing their data from trials in mouse model conducted at the Letten Center. Special thanks to Principal Investigator Rune Enger, MD, PhD for opening up the doors, and to PhD Student Vegard Broen, for providing us with the data sets, and answering all our questions along the way. Finally, a big thanks to Postdoctoral Researcher Laura Bojarskaite, PhD, who is *GliaLab*'s human sleep scoring expert, and responsible for the data that we trained on.

REFERENCES

[1] Rune Enger. GliaLab - Institute of Basic Medical Sciences (Website).

[2] Laura Bojarskaite, Alexandra Vallet, Daniel M. Bjørnstad, Kristin M. Gullestad Binder, Céline Cunen, Kjell Heuser, Miroslav Kuchta, Kent-Andre Mardal, and Rune Enger. Sleep cycle-dependent vascular dynamics in male mice and the predicted effects on perivascular cerebrospinal fluid flow and solute transport. *Nature Communications*, 14(1):953, February 2023.

[3] Maha Alattar, Alok Govind, and Shraddha Mainali. Artificial Intelligence Models for the Automation of Standard Diagnostics in Sleep Medicine—A Systematic Review. *Bioengineering*, 11(3):206, February 2024.

[4] Khald Aboalayon, Miad Faezipour, Wafaa Almuhammadi, and Saeid Moslehpour. Sleep Stage Classification Using EEG Signal Analysis: A Comprehensive Survey and New Investigation. *Entropy*, 18(9):272, August 2016.

[5] Đorđe Miladinović, Christine Muheim, Stefan Bauer, Andrea Spinnler, Daniela Noain, Mojtaba Bandarabadi, Benjamin Gallusser, Gabriel Krummenacher, Christian Baumann, Antoine Adamantidis, Steven A. Brown, and Joachim M. Buhmann. 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, April 2019.

[6] Geir Halnes, Torbjørn V. Ness, Solveig Næss, Espen Hagen, Klas H. Pettersen, and Gaute T. Einevoll. *Electric Brain Signals: Foundations and Applications of Biophysical Modeling.* Cambridge University Press, 1 edition, June 2024.

[7] Morten Engstrøm and Jan K. S. Jansen. elektromyografi, December 2024.

[8] Behrouz Alizadeh Savareh, Azadeh Bashiri, Ali Behmanesh, Gholam Hossein Meftahi, and Boshra Hatef. Performance comparison of machine learning techniques in sleep scoring based on wavelet features and neighboring component analysis. *PeerJ*, 6:e5247, July 2018.

[9] Oscar Atle Brovold, Eskil Grinaker Hansen, and Håkon Ganes Kornstad. Less is More: Comparing Neural Networks with Regression for Classification and Approximation, November 2024.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[11] Harikrishnan N. B. Confusion Matrix, Accuracy, Precision, Recall, F1 Score, June 2020.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, and others. PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, volume 32, 2019.

[13] Wes McKinney. Data Structures for Statistical Computing in Python. pages 56–61, Austin, Texas, 2010.

[14] Michael Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, April 2021.

[15] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer series in statistics. Springer, New York, NY, 2nd ed edition, 2009.

[16] Ramakrishnan Angarai Ganesan and Ritika Jain. Sleep-awake Classification using EEG Band-power-ratios and Complexity Measures. In *2020 IEEE 17th India Council International Conference (INDICON)*, pages 1–6, New Delhi, India, December 2020. IEEE.

[17] Jeroen Van Der Donckt, Jonas Van Der Donckt, Emiel Deprost, Nicolas Vandenbussche, Michael Rademaker, Gilles Vandewiele, and Sofie Van Hoecke. Do not sleep on traditional machine learning. *Biomedical Signal Processing and Control*, 81:104429, March 2023.