Neural Models for Detection and Classification of Brain States and Transitions

Replication Study

Leela Srinivasan
Columbia University

Abstract

Authors Marin-Llobet et al. present a computational framework for detecting and classifying anesthesia-induced brain states and state transitions in the cerebral cortex, aiming to examine spatiotemporal activity and underlying brain function. The dual-CNN and self-supervised autoencoder-based multimodal clustering algorithm together classify slow oscillations, microarousals, and wakefulness, and identify movement from one state to another.

In this project, I reproduce the classification and transition results reported by Marin-Llobet et al. and make modifications to multiple sections of the model architecture. Although original methods were replicated closely referencing models and training from the authors' publicly available code base, access to one data subject of four makes drawing direct comparison between results ill-posed. With data limitation in mind, I demonstrate that I can surpass the reported prediction results using optimized CNN architecture and training, and replicate autoencoder-based power spectral density clustering methods to identify state transitions and reveal oversimplified classification.

I report three models, the superior of which classifying awake, slow up-down, asynchronous microarousals, and slow microarousals with 0.99, 1.00, 0.99, and 0.99 accuracy, respectively, as against the reported global accuracy of 0.91, with a maximum of 0.96 for certain states. My recreation of autoencoder-based power spectral density clusters can be visually validated in the identification of transition states, due to the lack of ground truth labels.

1. Introduction

The ability to describe distinct brain states is of great clinical importance, with sleep brain states guiding anesthesia implementation and contributing to the overarching understanding of both healthy and pathological brain dynamics. Sleep states are characterized by distinct spatiotemporal patterns of neuronal firing and functional connectivity, demonstrating similar brain activity patterns over time and space, making them ideal for studying how the brain can operate under conditions of lower excitability and synchronized activity patterns.[1,2]

This distinction is precisely what defines a well-posed classification problem for neural networks, aiming to learn nonlinearities to differentiate and identify complex patterns in the neural signal at hand.

Here, I reproduce and aim to improve the dual CNN-autoencoder based classification algorithm presented by Marin-Llobet et al.[3] to classify Awake (AW), Slow Up-Down (SO), Asynchronous Microarousal (Asynch MA), and Slow Microarousal (Slow MA) sleep states, induced via anesthesia[4] in labeled data[5]. I create three distinct classification models using two different architectures and two different training regimes. I replicate the authors' state clustering algorithm[6] in samples that do not meet the CNN confidence bound specifications via a self-supervised autoencoder. Though the CNN class was carefully documented and training regimes were accessible via tutorial python notebooks, the computational architecture for the autoencoder based clustering algorithm was not available for reference. I carefully followed the described methods and referenced external power spectral density analysis in literature to complete this portion of the project.

My motivation for this project stems from a personal interest in developing algorithms to classify and decode minimally invasive neural data in the presence of high and/or variable noise. The CNN offered an opportunity to experiment with multiclass classification, while the autoencoder demonstrates how compression and reconstruction can strategically improve chances at decoding meaning from noisy neurophysiological data. I present this paper and a Github repository[7] containing user friendly tutorials to work through the following material.

2. Summary of the Original Paper

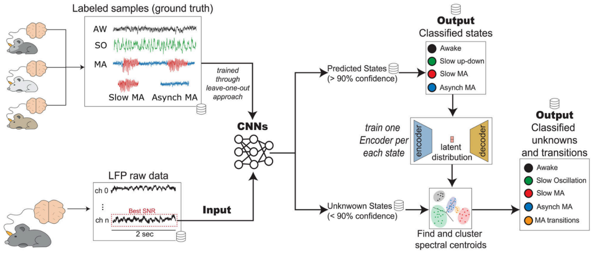
2.1 Methodology

The authors devise a computational framework for detecting anesthesia-induced brain sleep states in labeled data. They use a dual-CNN coupled with an autoencoder-based clustering algorithm for samples that did not meet the CNN's confidence bound. The initial

CNN is a 3-class classification problem, labeling awake, slow up-down, or microarousal (MA) states. The second CNN separates the MA classified states into a binary classification problem, dividing them into asynchronous (asynch MA), or slow (slow MA). The model only outputs classifications for samples if it is at least 90% confident, otherwise feeding these “unknown” samples to the autoencoder. The CNNs were trained with a leave-one-out approach for broad usage across subjects.

Autoencoders were trained for each state using samples that were successfully predicted above the confidence bound of 90%. Unknown samples were compressed and reconstructed individually using the trained autoencoder that corresponded to the state of maximal CNN confidence. The power spectral density was computed on reconstructed unknown samples in the delta power, theta power and gamma power bands, and spatial clustering was performed in three dimensions using the three power bands.

The paper targets transitions within the microarousal state, when a subject moves from asynch MA to slow MA or vice versa. These transitions do not have a ground truth transition label, so the authors used clustering position to estimate these samples, hypothesizing that samples under transition would have PSD band values between the discrete states. They denote the transition centroid as the midpoint between the asynch MA and slow MA centroids, and classify any sample closer to the transition centroid than the ground truth centroid as a transition sample.


Labeled samples (ground truth)
AW
SO
MA
Slow MA
Asynch MA
trained through leave-one-out approach
CNNs
LFP raw data
ch 0
ch n
Best SNR
2 sec
Input
Predicted States (> 90% confidence)
Output
Classified states
Awake
Slow up-down
Slow MA
Asynch MA
train one Encoder per each state
encoder
latent distribution
decoder
Unknwown States (< 90% confidence)
Find and cluster spectral centroids
Output
Classified unknowns and transitions
Awake
Slow Oscillation
Slow MA
Asynch MA
MA transitions

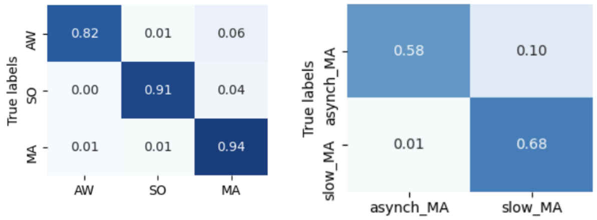
Marin-Llobet, Fig. 1B

2.2 Key Results

The following results all pertain to a confidence bound of 90%. The dual-CNN achieved a global classification accuracy of 91% across all subjects and states, including the “unknown” classifications. The state-level accuracies were as follows:

CNN State-Level Accuracies			
AW	SO	Slow MA	Asynch MA
81%	95%	93%	74%

The confusion matrices for each block of the dual-CNN in the matching subject "Phoebe" are shown below. The confusion matrices for unused subjects were omitted.

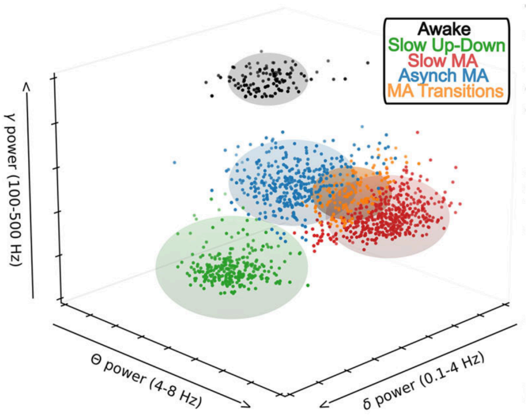


Marin-Llobet, Tutorial Python Notebook

The authors report the following percentages of data, by class, that were labeled “unknown” by the CNN and require further classification.

“Unknown” Proportions			
AW	SO	Slow MA	Asynch MA
14%	11%	38%	20%

They present a computational framework capable of detecting transitions between microarousal states. As transitions cannot be validated with ground truth labeling, they present sample evidence and spatial clustering results. The following figure aims to highlight microarousal (MA) transitions, which can be identified in orange.



Marin-Llobet, Fig. 2C

3. Student Project Methodology

3.1 Methods Overview

I aimed to answer the two principle questions posed by the team of authors:

(1) Can we correctly classify labeled anesthesia induced sleep states?

(2) Can we detect transition segments between distinct sleep states without ground truth?

I began by examining whether the dual-CNN structure was the optimal classification method for the given problem, tackling the first question. With access to only one subject of four, I had to make forced alterations to certain aspects of the problem. I was not able to employ the leave-one-out methodology, and that simplified the classification problem. To adapt, I experimented with CNN architecture, and found that a single, four-class classification performed better than a stacked 3-class and subsequent binary classification problem.

I built a default architecture that mirrored the first block of the authors' dual-CNN, adapted for four-class classification. This is used in Model 1 and 2 described in the subsequent text. I created a second CNN architecture that modified and aimed to improve the first that is used in Model 3, incorporating the following: Global Average Pooling, larger kernels to expand the receptive field, Batch Normalization, smaller Dense layers to reduce parameters and improve training, Dropout, and Adaptive Pooling. The idea behind Adaptive Pooling was to offer flexibility in time series length, which was not needed for the given dataset, but could prove useful in other settings.

I created two different training regimes. The first mirrored the paper as closely as possible and followed a simple baseline training routine with a batch size of 32, a fixed Adam optimizer with LR=0.001, and no LR scheduling. It saves a model checkpoint every 5 epochs, and validation losses/accuracies are logged each epoch. In the second training function I aimed to optimize and improve training with the following changes. I increased the batch size to 128, added a LR scheduler, and replaced periodic checkpointing with smart checkpointing that only saves the model when the validation loss improves. The rest of the training loop is similar to the first but designed for more stable, adaptive optimization. The default training is used in Model 1, and the optimized training is used in Models 2 and 3.

In the first tutorial python notebook entitled “classify_states.ipynb” and corresponding code, I refer to these three models as follows. Model 1: cnn_standard_default, a single 4-class CNN trained with the default function; Model 2: cnn_standard_optimized, a single 4-class CNN trained with the optimized training function; and finally, Model 3: cnn_modified_optimized, the modified 4-class CNN with the optimized training function. I did not provide tutorials on the additional models I tested with less remarkable results to avoid additional clutter.

In the evaluation stage, I apply the 90% confidence bound and track the data samples that do not meet the requirement and remain unclassified. These “unknown” samples are referenced throughout the paper and used in the autoencoder-based clustering approach.

At this stage, I perform an additional analysis to validate their choice of the 90% confidence bound. I evaluate each model against a threshold of 50%, 60%, 70%, 80% and 90%. Since Models 1 and 2 performed remarkably well, the distinction between thresholds was not notable. I therefore study the effect of threshold on precision, recall and f-1 score for Model 3.

The authors trained autoencoders for each sleep state in order to learn a compressed representation of the input data. I followed their approach of first performing dimensionality reduction by mapping the input into a 55-dimensional latent space using a ReLU-activated encoding layer, then reconstructing the original input through a linear decoding layer. I trained the model end-to-end using mean squared error (MSE) and included a separate validation set to monitor reconstruction performance during training. Though the authors did not report autoencoder reconstruction MSE, I opted to use their saved models instead of my own, as they had been trained on all four subjects and would introduce an aspect of generalization to my approach that I could not access without more subject data.

For each “unknown” sample output by the CNN, I followed the paper’s initiative and reconstructed each sample using the autoencoder trained on the state with maximal CNN confidence. I then computed the PSD in the delta, theta and gamma power bands. Projecting the bands into three dimensions, I computed the spatial centroid of each state, and denoted the transition centroid as the midpoint of the microarousal (MA) centroids. If any sample lay closer to the transition centroid than the centroid of its own ground truth class, I classified it as a transition sample. This process is demonstrated in detail in the second tutorial..

3.2. Objectives and Technical Challenges

The primary hurdle of my project was data limitations. The authors only uploaded data for one of four test subjects to their public Zenodo database, though they do mention that the full dataset is available to researchers upon reasonable request. As my data inquiry was left unanswered, I constrained the problem to work with subjects from one participant, dropping the leave-one-out methodology that was dependent on multiple subjects. I acknowledge the heavy within-subject bias that would render my model weak in generalization, and instead turn my focus to understanding and improving model architecture and the signal processing techniques employed by the authors.

The second challenge was recreating the autoencoder-based spatial clustering model that relied on power-spectral density analysis. I could not locate the code for this portion of the project and opted to build these pipelines from scratch. I put effort into making the transition detection process user friendly through the development of the python notebook tutorials.

The third challenge stems from user-friendliness. Though the dataset is large *[41728 x 2000 timesteps],* I wanted users to be able to train and evaluate models using Google Colaboratory. I ensured that models could be easily trained on CPUs if needed, and that models were saved and reloaded between training and evaluation in case of Jupyter runtime connection problems.

3.3. Problem Formulation and Design

3.3.1 Python Notebook Tutorials

There are two principle python notebooks tutorials in my Github repository that aim to walk users through the software described in this paper. They are targeted towards interpretability and figure production. Each notebook answers one of the the two main questions of the project, which I restate:

(1) Can we correctly classify labeled anesthesia induced sleep states?

(2) Can we detect transition segments between distinct sleep states without ground truth?

The first tutorial python notebook, entitled “classify_states.ipynb” details the training and testing of each of the three CNN-based classifications, generating confusion matrices and summary metrics. It generates “unknown” samples that do not meet the CNN classification criteria that will be loaded and analyzed in the subsequent notebook. If users do not wish to train the models but rather to evaluate trained models directly, they can run blocks to load the models I have saved to the /models directory.

The second tutorial python notebook, entitled “detect_transitions.ipynb”, walks users through the state-specific autoencoder-based clustering algorithm. Users load and run trained autoencoders on a sample “unknown” segment of data, and then conduct power spectral density analysis on the reconstructed data, projecting power bands into 3d-space to visualize state clusters. Spatial information is retrieved to identify potential transition states, validated through example.

A third notebook provides supplementary information on autoencoder use and is detailed in the Appendix of this report.

3.3.2 Code

The full repository breakdown is detailed in section 4.5, Software Design. All .py files can be located in the /utils folder of the repository.

The code to load, normalize, encode, split, and plot the data can be found in preprocess.py. My CNNs, the author CNNs, and the author autoencoder can be found in single_cnn_models.py, dual_cnn_models.py, and autoencoder_models.py, respectively. All training and evaluation functions can be found in train.py and eval.py, respectively.

4. Implementation

Section 4.1 describes the properties of the data and how to download and access the files. Section 4.2 describes the CNN Deep Learning Network, and the three models that are trained and evaluated in this project. Layer Dimensions and Training Protocol are described in detail. Section 4.3 describes the Autoencoder model and section 4.4 describes the Power Spectral Density analysis; how the autoencoder was used to detect transitions in the LFP signal.

4.1 Data

4.1.2 Overview

The data consists of cortical LFPs from different brain areas from four freely moving Lister Hooded rats (6–10 months old) with each session lasting an average of 6 ± 1 h. The electrode from either the orbitofrontal or primary motor cortex with the highest signal-to-noise ratio (SNR) was used for the current analysis. Simultaneous EMG was captured, positioned around the animal's neck, and was used to monitor animal movement. This information was a supplementary source of information in ground truth labeling, conducted by an expert.

Only one dataset of four, belonging to the subject 'Phoebe', is publicly available for download and analysis. I contacted the author to request the full dataset but did not receive an answer for the time being. The single-subject dataset and metadata can be downloaded directly from Zenodo, linked on the author's Github repository[3] and in my references. The majority of the preprocessing, aside from normalization, has already been done. The file "data_states.npy" contains 41728 2-second clips sampled at 1000 Hz, yielding a 41728 by 2000 matrix with samples at every millisecond.

The dataset was too large to upload as part of the project Github repository and should be downloaded directly from Zenodo. The accompanying metadata with ground truth labeling can be accessed from Zenodo or the /data folder in my repository.

4.1.2 Data Distribution

From the full 41728 sample dataset for subject "Phoebe", the distribution across classes is as follows, leaning heavily towards the Slow Up-Down (SO) sleep state. The predictive accuracy for this state is substantially higher, shown in subsequent results, as expected given the data imbalance.

Class Distribution		
State	**Encoding**	**Count**
Awake	0	968
Slow Up-Down	1	24720
Asynch MA	2	9596
Slow MA	3	6444

4.2 CNNs

In Tutorial 1, three models are trained and evaluated to perform a 4-class classification problem, funneling "unknown" classifications to the autoencoder.

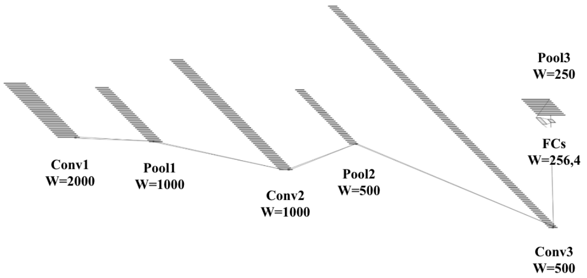


Fig. 1. Model 1 & 2 Architecture
CNN Models 1 and 2 follow this general architecture, while Model 3 has a slight modification in the FC layers. Numbers have been scaled down for visualization, but are intended to show convolutional filters and demonstrate how the input width changes over layers. The diagrams only depict Convolution and Pooling to highlight dimensional changes; ReLU is not shown. Image generated through NN-SVG[8]

4.2.1 CNN Model 1 & 2

Models 1 and 2 share the same base CNN architecture, and can be found in the code base under the class CNN(). As we are working with 1D time series data, the width for each layer is 1. Padding is described in each convolutional layer. The dimensions are as follows

Model 1 & 2 Layer Dimensions			
Layer	**H**	**D**	**Filter**
Input	2000	1	–
Conv1 (p=2)	2000	32	3
ReLU1	2000	32	–
MaxPool1	1000	32	1
Conv2 (p=2)	1000	64	3
ReLU2	1000	64	–
MaxPool2	500	64	1
Conv3 (p=2)	500	128	3
ReLU3	500	128	–

Model 1 & 2 Layer Dimensions			
MaxPool3	250	128	1
Flatten	1	32000	–
FC1	1	256	–
Output	1	4	–

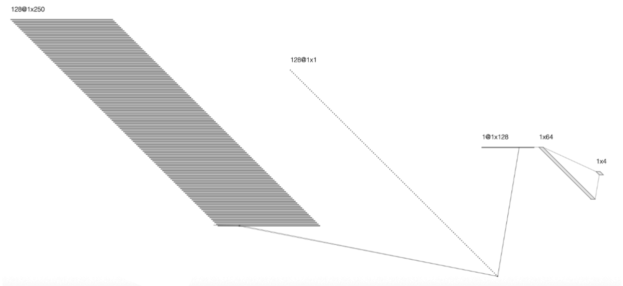


Fig. 2. Model 3 Output Layers
The third model follows a similar architecture as the first, where Batch Normalization precedes ReLU. The above diagram depicts solely the FC layers to scale, including Adaptive Global Pooling.

4.3.2 CNN Model 3

The third model uses a modified structure, and can be found in the code base under the class ModifiedCNN(). The layer dimensions are as follows

Model 3 Layer Dimensions

Layer	H	D	Filter
Input	2000	1	–
Conv1	2000	32	7
BatchNorm	2000	32	–
ReLU1	2000	32	–
MaxPool1	1000	32	2
Conv2 (p=3)	1000	64	7
BatchNorm	1000	64	–
ReLU2	1000	64	–
MaxPool2	500	64	2
Conv3 (p=3)	500	128	7
BatchNorm	500	128	–
ReLU3	500	128	–
MaxPool3	250	128	2
Adaptive AvgPool	1	128	Global
Flatten	128	–	–
Linear	64	–	–
Output	4	–	–

4.3.3 CNN Training

Model 1 used the default training algorithm described below, while Models 2 and 3 used the optimized algorithm. All models were trained for 10 epochs. Many components are shared across algorithms, where the default follows the paper's protocol with fixed learning rate and periodic checkpointing, and the optimized training introduces a larger batch size, adaptive learning rate scheduling, and validation-driven checkpointing to improve convergence stability and generalization.

Training Parameters

Component	Default Algorithm	Optimized Algorithm
Task	4-class classification	
Train / Val Split	80% / 20%	
Batch Size	32	128
Epochs	10	

Training Parameters		
Loss Function	CrossEntropyLoss	
Optimizer	Adam	
Initial LR	0.001	
LR Scheduler	None	ReduceLROnPlateau
Scheduler Patience	–	2 epochs
LR Reduction Factor	–	0.5
Validation-Based LR Update	No	Yes
Checkpointing Strategy	Every 5 epochs	Best validation loss only
Model Selection Criterion	Epoch number	Minimum validation loss

4.3 Autoencoder

Each autoencoder was built and trained using a simple 2-layer fully-connected architecture. The encoding dimension was 55, and the decoding dimension matches the input dimension, 2000.

Autoencoder Layer Dimensions			
Layer	Depth	Units	Activation
Input	2000	–	–
Encoding	55	55	ReLU
Decoding	2000	2000	Linear

4.4 Power Spectral Density Clustering

4.4.1 Welch's Method

Let $x(t)$ denote a discrete time-series signal sampled at $f_s = 1000\,Hz$. The signal is divided into overlapping segments of length N=256 samples. Each segment is windowed using a Hann window, and the PSD estimates are averaged.

$$P_{xx}(f) = Welch(x(t))$$

4.4.2 Band Power

The band power is computed as the integral of the PSD over that band. In discrete form:

$$\Sigma\, P_{xx}(f_k) \cdot \Delta f\ (P_{xx}(f_k))$$

The following bands are analyzed:
Delta: $0.1 - 1\,Hz$
Theta: $4 - 8\,Hz$
Gamma: $100 - 500\,Hz$

4.4.3 Spatial Projection

The three bands are used as coordinates, projecting each sample into 3-dimensional space. Coordinates are color labeled by class. The centroid for each class is computed as follows

$$for\ each\ class\ c$$

$$for\ i = 1{:}\,N\ samples$$

$$where\ each\ sample\ p_i = (delta_i,\ theta_i,\ gamma_i)$$

$$centroid_c = \mu_{delta},\ \mu_{theta},\ \mu_{gamma}$$

$$where\ \mu_j = \frac{1}{N}\sum_i j_i$$

4.4.4 Transition Detection

The transition centroid is the midpoint between the centroid for the asynch MA and slow MA classes.

$$centroid_t = \frac{1}{2}(centroid_{asynchMA} + centroid_{slowMA})$$

Each sample was then assessed against the transition centroid. If the sample was closer to the transition centroid than the centroid of its own ground truth class, it was labeled as a transition sample.[9]

4.5 Software Design

The top level structure of the repository, showing directories only, is as follows:

Shell

```
├── author_autoencoders
├── data
├── img
├── models
├── notebooks
└── utils
```

The jupyter notebook style user tutorials can be launched from */notebooks* and detail the full computational process. Tutorial 1, *classify_states.ipynb,* depicts the state classification process and tests the three CNN models. Tutorial 2, *detect_transitions.ipynb*, depicts the transition detection process via autoencoder-based spatial clustering of power spectral density bands.

Shell

```
├── notebooks
│   ├── classify_states.ipynb
│   ├── detect_transitions.ipynb
│   └── supplementary.ipynb
```

The raw data array is 667.6 MB and must be downloaded directly from Zenodo, linked in the References section, the author Github repository, and my own *README.md* file. The metadata with ground truth labeling can be found in the */data* directory as *metadata_states.pkl.*

Shell

```
├── data
│   ├── metadata_states.pkl
│   └── metadata.parquet
```

The code is contained in /utils and the file structure is shown below. The file *preprocess.py* contains files for loading, normalizing, splitting and encoding. The file *single_cnn_models.py* contains the classes used in model 1, 2 and 3, while dual_cnn_models.py contains the classes used in the paper. Both train.py and eval.py are self explanatory. The files *autoencoder_models.py* and *autoencoder_utils.py* contain the models and functions for executing and interpreting the power spectral density transition analysis, respectively.

Shell

```
└── utils
    ├── autoencoder_models.py
    ├── autoencoder_utils.py
    ├── dual_cnn_models.py
    ├── eval.py
    ├── preprocess.py
    ├── single_cnn_models.py
    └── train.py
```

The */models* directory contains pretrained and evaluated models, giving the user flexibility to train their own models or run pretrained models. The user has access to all metrics, unknown samples, and weights from trained models. The training and testing split has been saved as index arrays under *train_dataset_indices.pth* and *test_dataset_indices.pth*, respectively, if the user wishes to replicate or work from the same analysis reported in the paper.

Each model is named using the model and training type *(cnn_default_standard*, *cnn_default_optimized*, and *cnn_modified_optimized*) and the CL threshold (0.9% for analysis, and 0.5-0.9 in threshold testing). There are separate folders for training and evaluation. Here is one such folder breakdown for example.

Shell

```
├ models
│  └── train_dataset_indices.pth
│  ├── test_dataset_indices.pth
│  ├── cnn_standard_optimized_0.9
│  │    ├ all_preds_df...
│  │    ├ confusion_matrix.png
│  │    ├ metrics_df...
│  │    ├ unknown_df...
│  ├── cnn_standard_optimized_training
│  │    ├ ...metrics.json
│  │    ├ ...epoch5.pth
│  │    ├ ...epoch10.pth
```

All images can be found in */img*, and author uploaded autoencoders used in the transition detection notebook can be found in */author_autoencoders*.

5. Results

5.1 Project Results

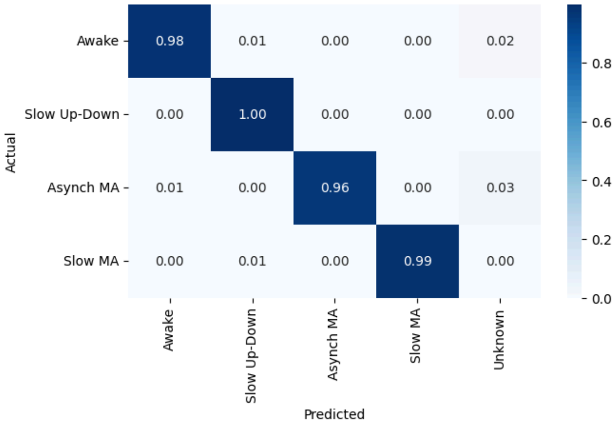
5.1.1 CNN Classification

The first set of results correspond to CNN accuracy, including unknown samples.

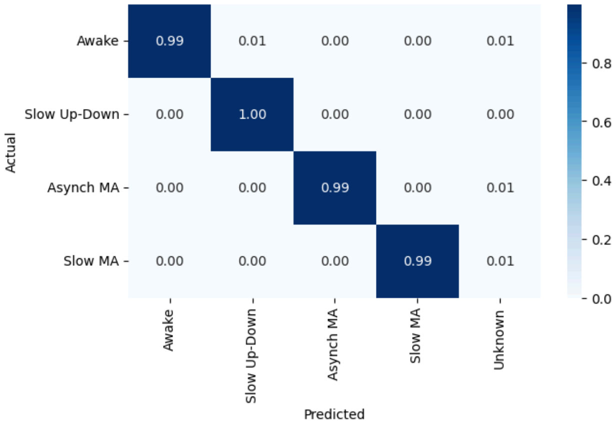
CNN State-Level Accuracies				
Model	AW	SO	Slow MA	Asynch MA
1	98%	100%	96%	99%
2	99%	100%	99%	99%
3	62%	100%	93%	75%

Confusion matrices for Models 1, 2 and 3 were computed and depicted below. Unknown samples correspond to a confidence level of 90% for all model implementations.

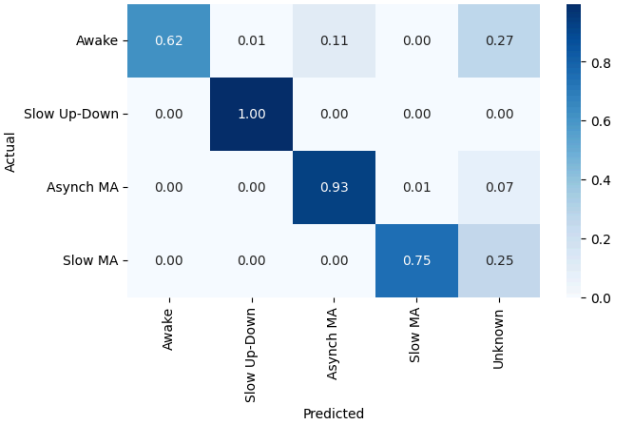
Model 1 Confusion Matrix



Model 2 Confusion Matrix



Model 3 Confusion Matrix



"Unknown" Proportions				
Model	AW	SO	Slow MA	Asynch MA
1	1.52%	0.02%	2.91%	0.31%
2	0.50%	0.26%	0.65%	0.62%
3	26.77%	0.34%	6.63%	24.82%

Though the results for Model 3 using the ModifiedCNN() class revealed the lowest accuracy, they served as the optimal dataset for testing the autoencoder, as the highest proportions of samples were classified as unknown across classes.

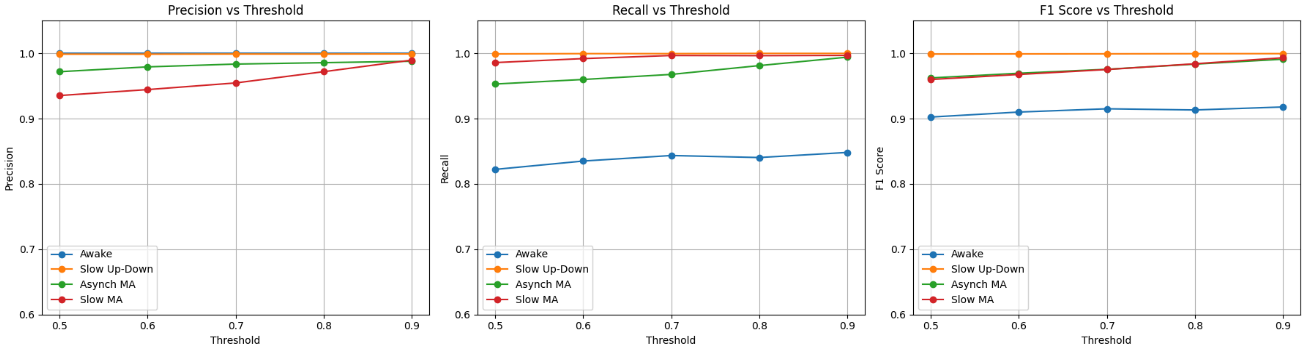


Fig. 4. Metrics Across Threshold

To validate the authors' choice of the 90% confidence bound, I performed an additional analysis, tracking (left to right) precision, recall and f1-score across threshold values. The jump in precision from 0.8 to 0.9 is a solid inclination to implement the more restrictive bound.

5.1.2 Training/Validation Error

For each model, training and validation error were computed across the 10 epochs. The curves are depicted in the following set of subplots.

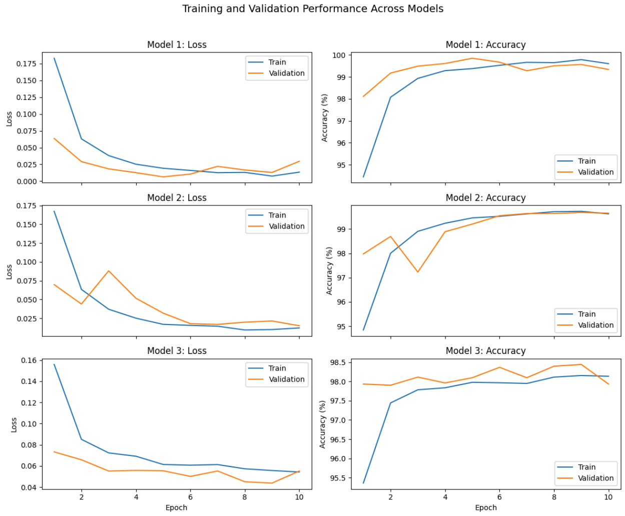


Fig. 3. Training and Validation Curves

From top to bottom: Model 1, 2 and 3, respectively. In the standard architecture used in Model 1 and 2, loss curves are lower and quicker to stabilize, and accuracy curves are higher and quicker to increase.

5.1.3 Transition Detection

Since the autoencoder-based clustering algorithm could not be validated against ground truth labels, I demonstrate a distribution of spatial state clusters in the 3 power spectral density band dimensions. Without ground truth labeling, detected transitions can be visually validated through identifying morphological similarity from more than one discrete class.

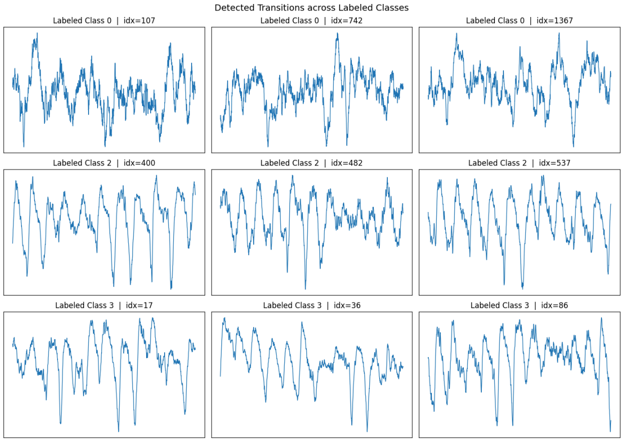


Fig. 5. Extended Transition Examples

Top row: Examples of transition detection in ground truth Awake class, encoded as 0. Middle row: Examples of transition detection in ground truth Asynch MA class, encoded as 2. Examples of transition detection in ground truth Slow MA class, encoded as 3. Slow Up-Down samples were rarely output as "unknown" and were excluded from the image due to scarcity.

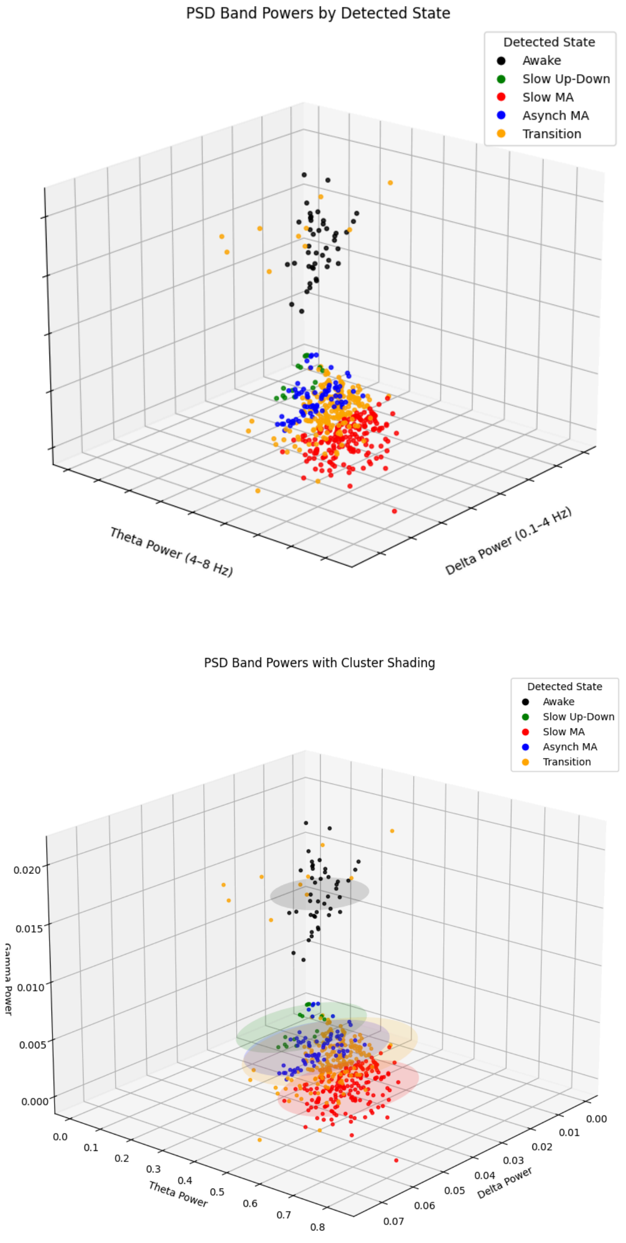


Fig. 6. Power Spectral Density Clusters and Transition Identification

Top: 3-dimensional scatter plot of delta, thea, and gamma power in autoencoder-reconstructed samples with transition samples between microarousal states plotted in orange. Bottom: Identical plot with ellipses for visual grouping, mimicking the author's depiction.

5.1.4 Training Times

One main goal of optimizing the training protocol described by the authors was to reduce the training time. Given the size of the training set *[33382 x 2000]* and the relatively modest nature of the CNNs, I wanted to begin by running the models on GPUs, and see if they were feasible to run on CPUs for user flexibility using Google Colaboratory.

On GPU, Models 1, 2, and 3 trained for 8, 6, and 5 minutes, respectively. On CPU, they trained for 41, 31, and 29 minutes, respectively. While preferable on GPU, I built the Tutorial notebooks to be run on Google Colaboratory with solely CPU access, saving and reloading models for evaluation to support runtime disconnections.

5.2 Results Comparison

5.2.1 CNN Accuracy Results

The best iteration of my CNN model outperformed the author classification, though as mentioned recurrently, on one test subject of four. The within state accuracy comparisons are as follows: 99% as against 81%, 100% as against 95%, 99% as against 93%, 99% as against 74%. The unknown proportions are significantly lower, once again reflecting the higher CNN classification accuracy. Subdivided by state, the differences are as follows: as against 14%, as against 11%, as against 38% and as against 20%.

5.2.2 Training/Validation Error

The authors did not comment on model training and validation error and I was therefore unable to make a comparison. I reported my own results above.

5.2.3 Transition Detection

My spatial clustering of autoencoder-reconstructed PSD demonstrated remarkable similarity, particularly when considering that I ran my analysis on one test subject out of four. One distinct difference is that my algorithm detected transition samples within the awake class, seeing that they lie closer to the transition centroid than the ground truth awake centroid. There are multiple plausible explanations for this. The first is that the distribution of the awake class was wider, and outliers were mistakenly lumped into the transition class. The second is that the awake samples demonstrated some morphological similarity, detected through power spectral density band analysis, with the transition group.

5.2.4 Training Time

The authors did not comment on model training time and I was therefore unable to make a comparison. I reported my own training time results above.

5.3 Discussion

Limited access to one subject's data as opposed to the full set of four limited my ability to draw direct comparisons between my results and that of the paper, but recreating and modifying their computational pipeline proved to be an incredibly rewarding experience. My models are inherently weaker and flawed for future applications demanding generalization, but the classification problem was simpler: because the model only sees patterns from a single person, it has the chance to learn that person's unique physiology, behavior, or sensor characteristics rather than learning general patterns that apply across subjects. As a result, performance may appear high but will not generalize to new subjects, and the model is effectively overfitting to that one individual's signature. Accepting the within-subject bias that was out of my control, I focused my attention on making architectural and training modifications that would strengthen the model goals, and on understanding the signal processing techniques that guided the autoencoder-based clustering. I took advantage of my weakest model classification to develop the autoencoder-based clustering, where the algorithm was strengthened by a larger portion of unknown samples to reconstruct and spatially construct.

I aimed to replicate and further develop the results. As such, my models are geared towards (1) building a strong representation of the original specifications and (2) demonstrating that my own modifications can improve predictive accuracy. The first model is the most similar to the paper, designed with similar architecture and training. The second takes the original structure and introduces a modified training algorithm. The third modifies the architecture and again applies the modified training. With this division, I aim to show how I sequentially changed the experimental setup, and how the resulting prediction metrics change. I succeeded in achieving higher predictive accuracy, and was able to attain a visually similar spatial clustering of sleep states, including the transition identification. I demonstrate through example the identification of transition states, where the subject moves from one microarousal to another, challenging the original ground truth label. I additionally demonstrate the autoencoder denoising process, which can be an extremely useful addition to standard signal processing pipelines to address the presence of artifact or extensive noise.

In the recreation of the power spectral density algorithm, I became extremely curious why the authors chose to target solely the transition state between microarousal states, and to overlook the potential for transition between other states. It is possible that those transitions are less biologically relevant, and I would require an additional literature review to find an answer.

6. Future Work

6.1 Autoencoder Development

The authors employ a simple 2-layer architecture, and I would like to set up an experiment to test the potential correlation between the depth of autoencoder compression and transition detection. The question would be whether depth is the answer here; or more specifically, whether a shallow denoising autoencoder can outperform a deep one if it is better regularized.

6.2 Noise Addition

To compensate for the within-subject bias, I could experiment with varying levels of data augmentation, for example, perturbing the data with additions of Gaussian noise.

6.3 Broad Applicability

The authors focused on transitions between the microarousal states, but I would like to continue their work by first testing the transitions between other sleep-induced states, computing the centroid of each pairing of sleep-induced states.

I would then like the opportunity to extend this approach to non anesthesia induced sleep states, or a broader selection of neurophysiological states given the appropriate dataset. I believe the autoencoder-based spatial clustering could be of great utility, as power spectral density analysis is widely accepted, robust to noise and neurophysiologically meaningful. For example, bringing a similar spatial clustering approach in the study of epilepsy could support researchers and clinicians in the process of decoding EEG signals. Similar state classification could be applied to understand how brain rhythms differ between normal, interictal, pre-ictal, and ictal (seizure) states, and to target the crucial transition periods between such states. Tracking state transitions on a spatiotemporal scale, perhaps across the cortex, could be an additional further direction.

7. Conclusion

Marin-Llobet et al. present a deep learning framework for the detection and classification of anesthesia induced brain states, ranging from awake to microarousal states. In this reproduction, I recreate the CNN-based multiclassification problem and make various modifications to the architecture and training regimes to improve the prediction accuracy. I achieve a global accuracy of 99%, well above the reported 93%, albeit including the within-subject bias that could not be helped given the data limitations described in my report. I report higher accuracies across all sleep states, and confirm their choice of the 90% confidence bound using precision, accuracy and F-1 metrics.

To study the "unknown" samples that did not surpass the CNN confidence bound, I reproduce the autoencoder-based clustering algorithm, applying power spectral density bounds to the autoencoder reconstructed signal to identify transition samples between the slow and asynchronous microarousal states.

In this report, I replicate and build upon the principle findings and algorithmic strategies shared by Marin-Llobet et al. My best-performing model demonstrates a global accuracy of 99% as against the reported 91%. Though I recognize that the problem was significantly altered under the training and testing of data from one of four subjects, I was impressed by the transition detection algorithm, particularly as I was able to confirm the model's detection by eye. Detecting transitions in sleep states offers the opportunity to apply such algorithms more broadly to other neurophysiological states in the pursuit of a better understanding of brain states, both in basic and clinical neuroscience.

8. Acknowledgements

I thank Professor Kostic and E4040 class TAs for their support and help throughout the semester.

My understanding of the signal preprocessing and power spectral density analysis was largely supported by my time and collaborators at the Neurophysiology of Epilepsy Unit under Dr. Sara Inati at the National Institutes of Health.

9. References

[1] D. A. McCormick, D. B. Nestvogel, and B. J. He, "Neuromodulation of brain state and behavior," Annual Review of Neuroscience, vol. 43, 2020, pp. 391–415.
[2] E. Zagha and D. A. McCormick, "Neural control of brain state," Current Opinion in Neurobiology, vol. 29, 2014, pp. 178–186.
[3] A. Marin-Llobet, A. Manasanch, L. Dalla Porta, M. Torao-Angosto, and M. V. Sanchez-Vives, "Neural models for detection and classification of brain states and transitions," *Communications Biology*, vol. 8, 599, Apr. 11, 2025
[4] J. M. Tauber, S. L. Brincat, E. P. Stephen, J. A. Donoghue, L. Kozachkov, E. N. Brown, and E. K. Miller, "Propofol-mediated unconsciousness disrupts progression of sensory signals through the cortical hierarchy," Journal of Cognitive Neuroscience, vol. 36, no. 2, Feb. 2024, pp. 394–413.
[5] Data: https://zenodo.org/records/14990181
[6] Author Github Repository: https://github.com/arnaumarin/LFPDeepStates
[7] My Project Github Repository: https://github.com/ecbme4040/e4040-fall2025-project-lsls.git
[8] A. LeNail, "NN-SVG: Neural network SVG generator," [Online]. Available: https://alexlenail.me/NN-SVG/LeNet.html. Accessed: Dec. 13, 2025.
[9] E. P. Duff, L. A. Johnston, J. Xiong, P. T. Fox, I. Mareels, and G. F. Egan, "The power of spectral density analysis for mapping endogenous BOLD signal fluctuations," *Human Brain Mapping*, vol. 29, no. 7, Jul. 2008, pp. 778–790

10. Appendix

10.1 Author Contributions

This project was completed individually. All work is my own.

10.2 Credit

Sub-blocks of code have been pulled directly from the authors' publicly available codebase in attempts to recreate methodology. Code has been revised for reuse but some lines may be identical, particularly in architecture formation and model evaluation.

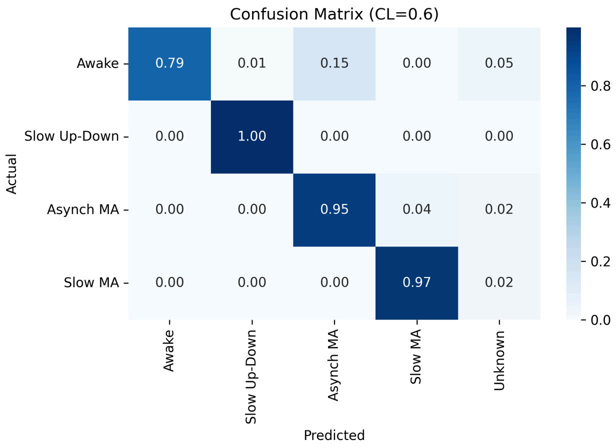
10.3 Supplementary Material

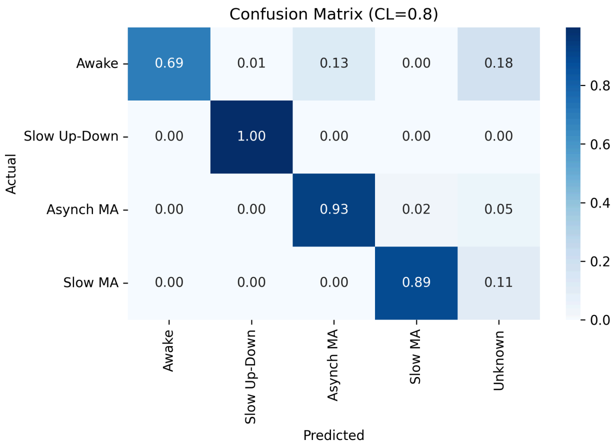
10.3.1 Supplementary Python Notebook

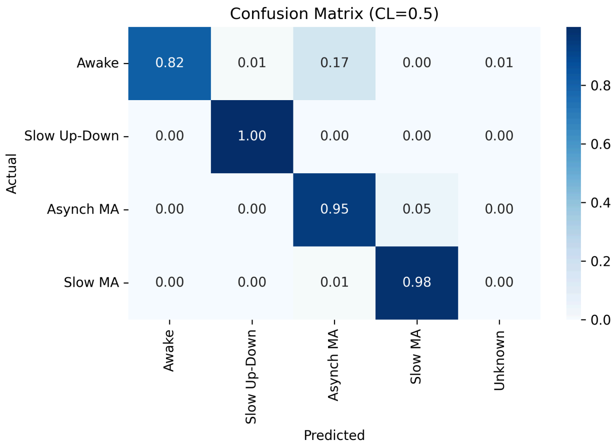
The third python notebook, supplementary.ipynb details an alternate use case for the authors' trained autoencoders. If a user wishes to generate the reconstruction loss for each autoencoder using a given sample, they could select the state with minimal loss as an alternate predictive mechanism. The tutorial walks the user through the process for a given sample.

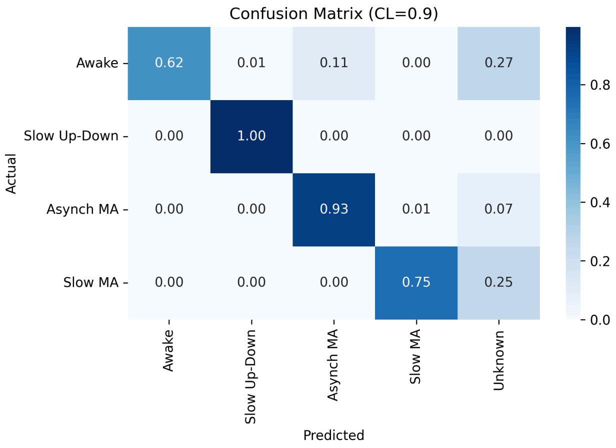
10.3.1 Confusion Matrices Across Thresholds

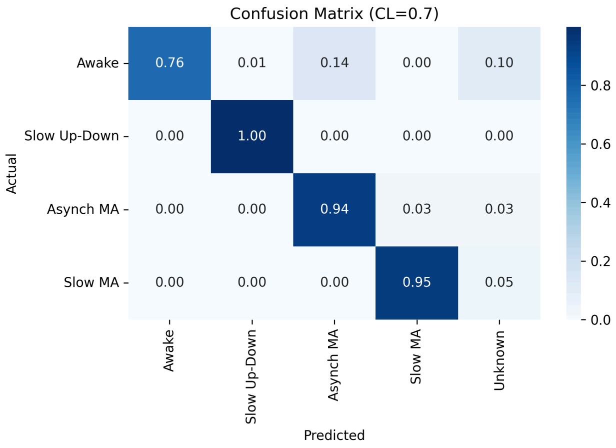
In the threshold justification analysis for Model 3, I computed the confusion matrices from threshold values of 0.5 to 0.9. The results are as follows:


Confusion Matrix (CL=0.6)
Actual
Awake 0.79 0.01 0.15 0.00 0.05
Slow Up-Down 0.00 1.00 0.00 0.00 0.00
Asynch MA 0.00 0.00 0.95 0.04 0.02
Slow MA 0.00 0.00 0.00 0.97 0.02
Awake
Slow Up-Down
Asynch MA
Slow MA
Unknown
Predicted
0.8
0.6
0.4
0.2
0.0


Confusion Matrix (CL=0.8)
Actual
Awake 0.69 0.01 0.13 0.00 0.18
Slow Up-Down 0.00 1.00 0.00 0.00 0.00
Asynch MA 0.00 0.00 0.93 0.02 0.05
Slow MA 0.00 0.00 0.00 0.89 0.11
Awake
Slow Up-Down
Asynch MA
Slow MA
Unknown
Predicted
0.8
0.6
0.4
0.2
0.0


Confusion Matrix (CL=0.5)
Actual
Awake 0.82 0.01 0.17 0.00 0.01
Slow Up-Down 0.00 1.00 0.00 0.00 0.00
Asynch MA 0.00 0.00 0.95 0.05 0.00
Slow MA 0.00 0.00 0.01 0.98 0.00
Awake
Slow Up-Down
Asynch MA
Slow MA
Unknown
Predicted
0.8
0.6
0.4
0.2
0.0


Confusion Matrix (CL=0.9)
Actual
Awake 0.62 0.01 0.11 0.00 0.27
Slow Up-Down 0.00 1.00 0.00 0.00 0.00
Asynch MA 0.00 0.00 0.93 0.01 0.07
Slow MA 0.00 0.00 0.00 0.75 0.25
Awake
Slow Up-Down
Asynch MA
Slow MA
Unknown
Predicted
0.8
0.6
0.4
0.2
0.0


Confusion Matrix (CL=0.7)
Actual
Awake 0.76 0.01 0.14 0.00 0.10
Slow Up-Down 0.00 1.00 0.00 0.00 0.00
Asynch MA 0.00 0.00 0.94 0.03 0.03
Slow MA 0.00 0.00 0.00 0.95 0.05
Awake
Slow Up-Down
Asynch MA
Slow MA
Unknown
Predicted
0.8
0.6
0.4
0.2
0.0