Class: CSE 144 Spring 2023
Group Number: 5
Members: Adam Elaidy, Adam Hammond, Regan Miller, Audrey Ostrom, Sreyes Venkatesh

Automated Sleep Stage Scoring in Mice

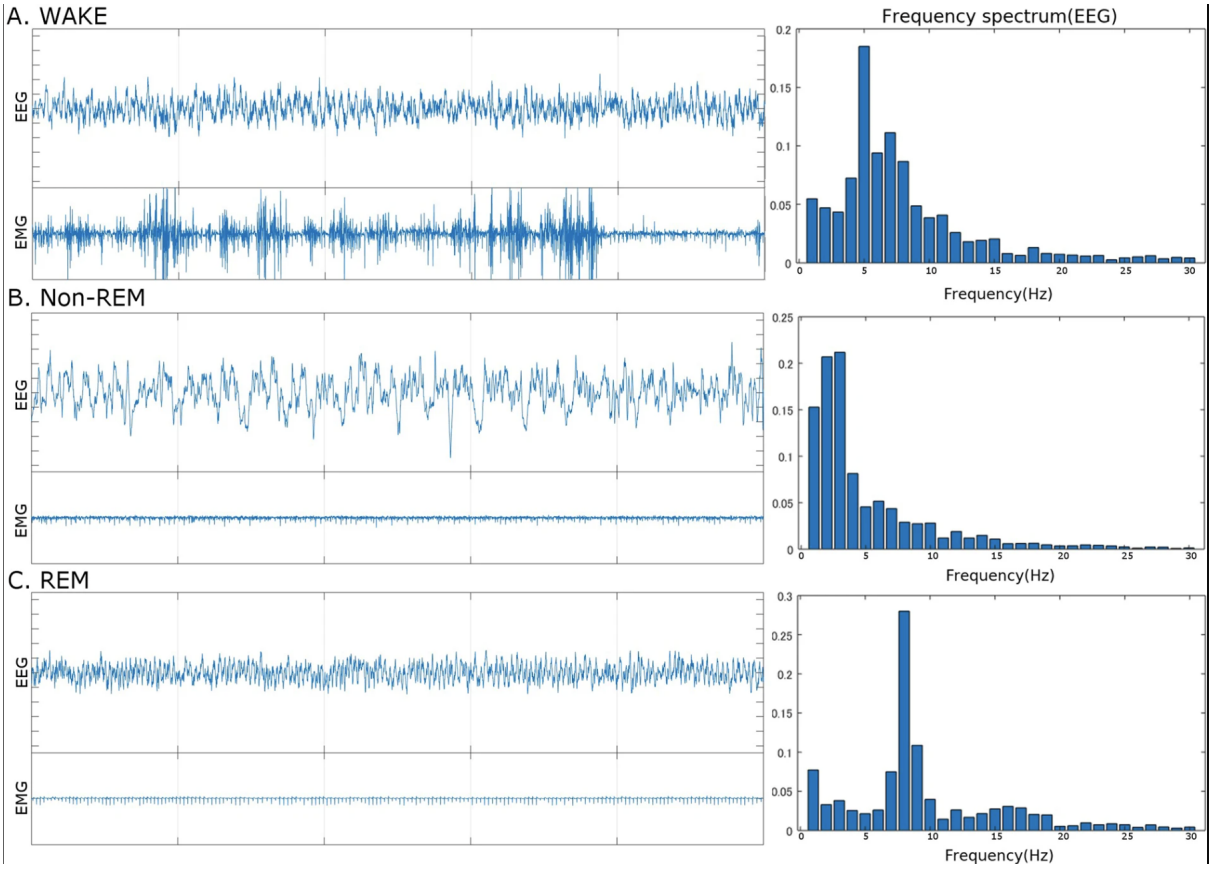
Abstract

For our project, our team developed a real-time sleep stage classification model with convolutional neural networks (CNNs) using two channels of electro-encephalogram (EEG) and electromyography (EMG) sources collected from mice across 24-hour time periods. Our finished model, when given a set of data points, will be able to detect whether or not a mouse is in one of the three stages of sleep: wake, non-rapid eye movement sleep (non-REM), and rapid eye movement sleep (REM). If it's too hard to tell, our model will the classify the sleep stage the mouse is in as "Artifact". Our model will display a degree of robustness against noise (as EEG and EMG signals are semi-dependent on placement of wires) and the individualities of one mouse. The inspiration for our project comes from the neuroscience labs at the University of California - Santa Cruz. Currently, the researchers there must score sleep stages manually, which requires considerable time and specialized knowledge on the subject. This process is rather tedious as a result. One mouse alone in the span of 24 hours produces 20278848 rows of data. There's an intense need for automation here so that the researchers can divert their focuses elsewhere from menial tasks such as this. As for why mice specifically, they're chosen for their genetic proximity to humans[1] and the relative ease of obtaining EEGs from them humanely, compared to other species like reptiles[2] and humans. Majority of the work done so far into developing an automated method for sleep stage scoring hasn't been publicly released or isn't friendly enough to use for people with no machine learning background.

I Introduction

There's three separate stages of sleep in mice, similar to humans, which I will expand upon in further detail:

Figure 1: Examples of EEG/EMG signals in each stage. (A) WAKE, (B) non-REM, and (C) REM.



Source: M. Yamabe, K. Horie, H. Shiokawa, H. Funato, M. Yanagisawa, and H. Kitagawa, "MC-SleepNet: Large-scale Sleep Stage Scoring in Mice by Deep Neural Networks," *Scientific Reports*, vol. 9, no. 1, Oct. 2019, doi: 10.1038/s41598-019-51269-8, Figure 1.

Wake

This stage is often quantified by mixed frequencies of EEG signals, or a greater range in the oscillations, as seen in Figure I. This is because this stage of sleep is typically the transition period between "wakefulness" to sleep (hence the name). Thus, a mouse's body and brain tends to be rather active still.

Non-REM

In this stage, a mouse's body starts to rest, and its brain tends to exhibit "widespread cortical synchronization with rhythmic activities in the delta (0.5–4 Hz) and sigma range (12–15 Hz)"[3]. In essence, responses to external stimuli are slowed. Thus, "the peak EEG frequencies become lower and the amplitude of EEG signals becomes higher than in the WAKE stages"[4]. Majority of sleep in both humans and mice take place within this stage - taking up approximately 90% of sleep in the latter.

REM

This stage is considered the point of "deepest" sleep: mice in this stage will be completely stationary excepting breathing, eye movement, and the occasional muscle twitch. This stage is entered fairly frequently from the non-REM once the mouse has "fallen asleep", and can last up to several minutes.

All together, mice (and humans) will cycle through all these stages regularly - so understanding the nuances of each stage will be critical in training our model. Our task, given EEG and EMG signal data from a mouse, is to identify what sleep stage the mouse is currently at in a given period of time.

II Methodology

II.I Technologies Used

We decided early on that we wanted to develop our model(s) using PyTorch due to its wide variety of built-in functions and classes. The amount of support it has made it easier to debug later on, and we had to do a lot of debugging and refactoring with this project.

For reports, sklearn has so many functions that automatically calculate the statistics that we deemed relevant to the analysis of our models - so we deemed it remiss to not use them. For our graphs, we used matplotlib due to prior familiarity with it.

Another thing we used was the tkinter library. Thanks to Adam H's prior familiarity with using this library, we decide the best course of action to make the program that trains and runs this model have a workable user interface. The user gets prompted to upload the data

files they would like to work with, to avoid issues and complications of getting specific file paths. After this class concludes, we would like to develop this GUI even further to make it easier for people with no background in computer science to use.

II.II Data Collection

We have already obtained all of the EEG data from the UCSC neuroscience labs necessary to both train our model, validate its results, and then test it subsequently. Majority of this data is unlabeled, while some portions are annotated. We mostly trained/validated our with the dataset UCSC's lab, since that's who this model is ultimately intended for. As alluded to in the abstract, EEG signals are contingent somewhat on where wires are placed on the scalp of the mouse - so it'll be important to train our model as much as possible to be able to account for this.

II.III Parsing

For the CNN, which is our main model, we need to split the data points into equally sized chunks which we will refer to as periods. These periods will be classified as Wake, Non-REM, REM, or Artifact. They need to be equally sized to fit into the CNN input without padding or truncating, which would fabricate or lose data respectively.

The annotated data (Figure 2) is a text document where each line is a stop or start timestamp, with the classification made by the researcher. We simply made a 2D array of the stop times, and the annotated state.

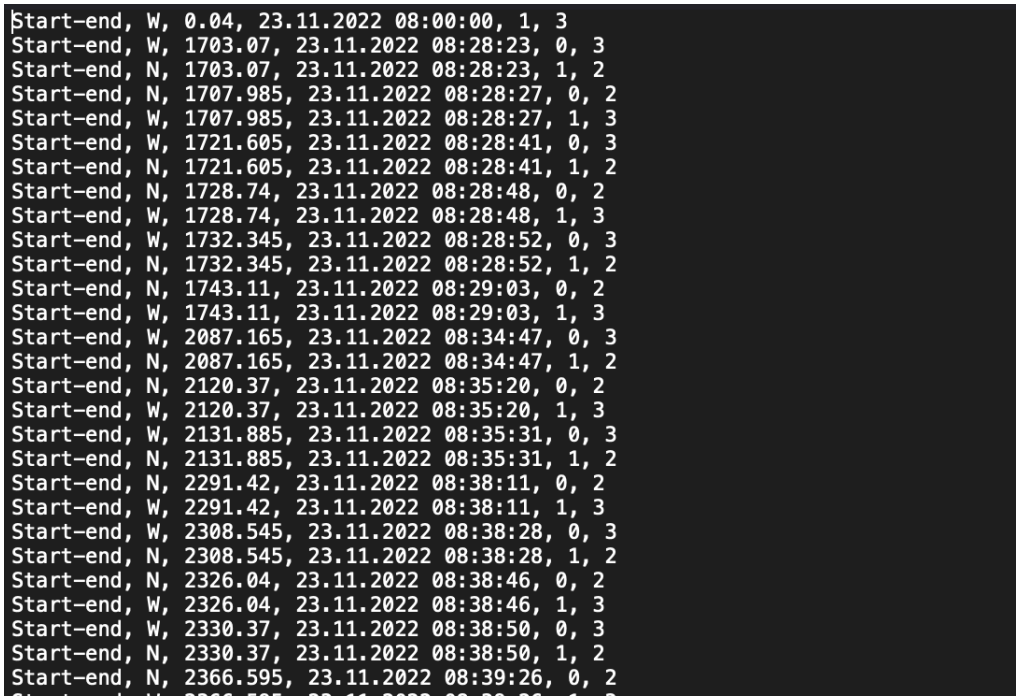
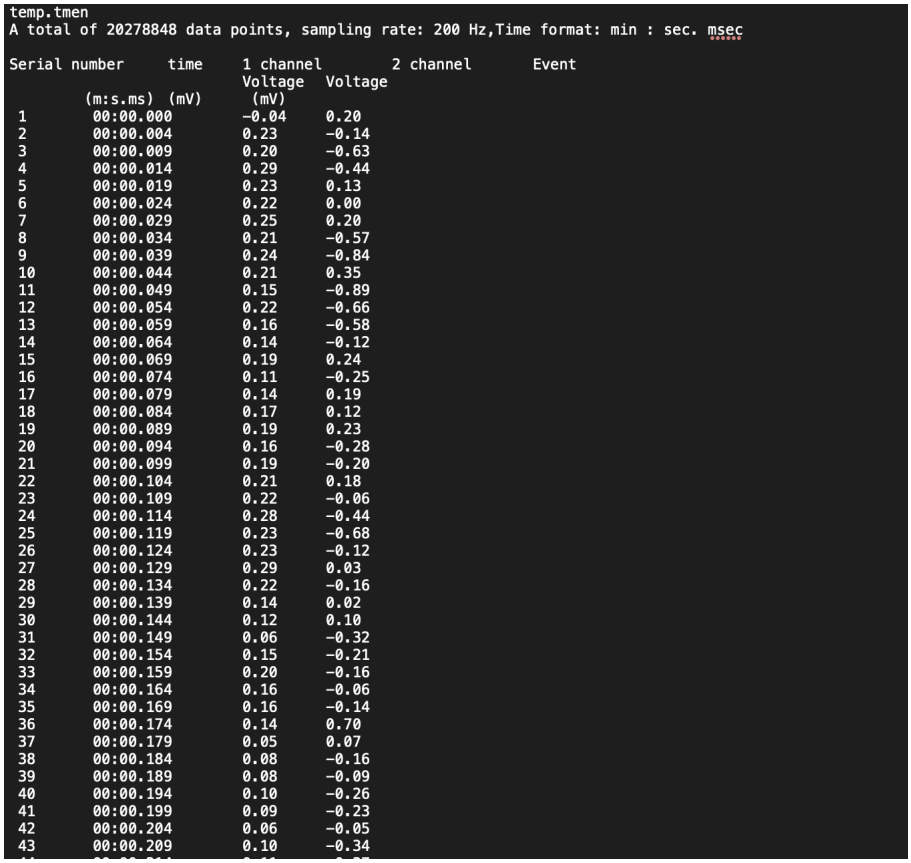


Figure 2: What the annotated data looks like

The EEG/EMG data is a text document (Figure 3) where each line has a timestamp, and a voltage reading for each channel. We split each datapoint into lists of `period_size`, which is the number of signal data points being grouped together into a period.

We create an array with each row being the period start, period end, and period readings. We then use the 2 arrays, to iterate through the periods and label each accordingly. We added a `tolerance` parameter that tunes what percent of a period's time needs to be classified. 0% means we would not classify any artifacts, 100% means we'd classify any period with disjoint classifications as "Artifact".

As mentioned before, "Artifact" is a unique classification that doesn't represent a sleep stage (Wake, Non-REM, REM). Instead, it's the classification for period where it's too hard to tell what sleep stage the period truly falls under. While it may seem unconventional to create a class for this, this was a special request made by the research team at the neuroscience Labs. They wanted human intervention in the scenarios where the model might fail.



temp.tmen
A total of 20278848 data points, sampling rate: 200 Hz,Time format: min : sec. msec

Serial number	time (m:s.ms)	1 channel Voltage (mV)	2 channel Voltage (mV)	Event
1	00:00.000	-0.04	0.20	
2	00:00.004	0.23	-0.14	
3	00:00.009	0.20	-0.63	
4	00:00.014	0.29	-0.44	
5	00:00.019	0.23	0.13	
6	00:00.024	0.22	0.00	
7	00:00.029	0.25	0.20	
8	00:00.034	0.21	-0.57	
9	00:00.039	0.24	-0.84	
10	00:00.044	0.21	0.35	
11	00:00.049	0.15	-0.89	
12	00:00.054	0.22	-0.66	
13	00:00.059	0.16	-0.58	
14	00:00.064	0.14	-0.12	
15	00:00.069	0.19	0.24	
16	00:00.074	0.11	-0.25	
17	00:00.079	0.14	0.19	
18	00:00.084	0.17	0.12	
19	00:00.089	0.19	0.23	
20	00:00.094	0.16	-0.28	
21	00:00.099	0.19	-0.20	
22	00:00.104	0.21	0.18	
23	00:00.109	0.22	-0.06	
24	00:00.114	0.28	-0.44	
25	00:00.119	0.23	-0.68	
26	00:00.124	0.23	-0.12	
27	00:00.129	0.29	0.03	
28	00:00.134	0.22	-0.16	
29	00:00.139	0.14	0.02	
30	00:00.144	0.12	0.10	
31	00:00.149	0.06	-0.32	
32	00:00.154	0.15	-0.21	
33	00:00.159	0.20	-0.16	
34	00:00.164	0.16	-0.06	
35	00:00.169	0.16	-0.14	
36	00:00.174	0.14	0.70	
37	00:00.179	0.05	0.07	
38	00:00.184	0.08	-0.16	
39	00:00.189	0.08	-0.09	
40	00:00.194	0.10	-0.26	
41	00:00.199	0.09	-0.23	
42	00:00.204	0.06	-0.05	
43	00:00.209	0.10	-0.34	

Figure 3: What the unannotated data (EEG & EMG signals) looks like

III Our Models

DoubleCNN

Four features are inputted to our model to classify each period of data: **EEG signals, EMG signals, the power spectrum of the EEG signals and the power spectrum of the EMG signals (which are obtained through Fourier transformations)**. The power spectrums provide information about the amplitude of different frequencies present in the signal, which is a significant distinguishing characteristic between sleep stages.

The DoubleCNN is made up of two parallel architectures that EEG and and EMG data are both passed through before being concatenated and passed through two linear layers. The final output is produced by passing those numbers through a soft max function to compute the "probabilities" of each category. We chose to use separate CNN structures because the

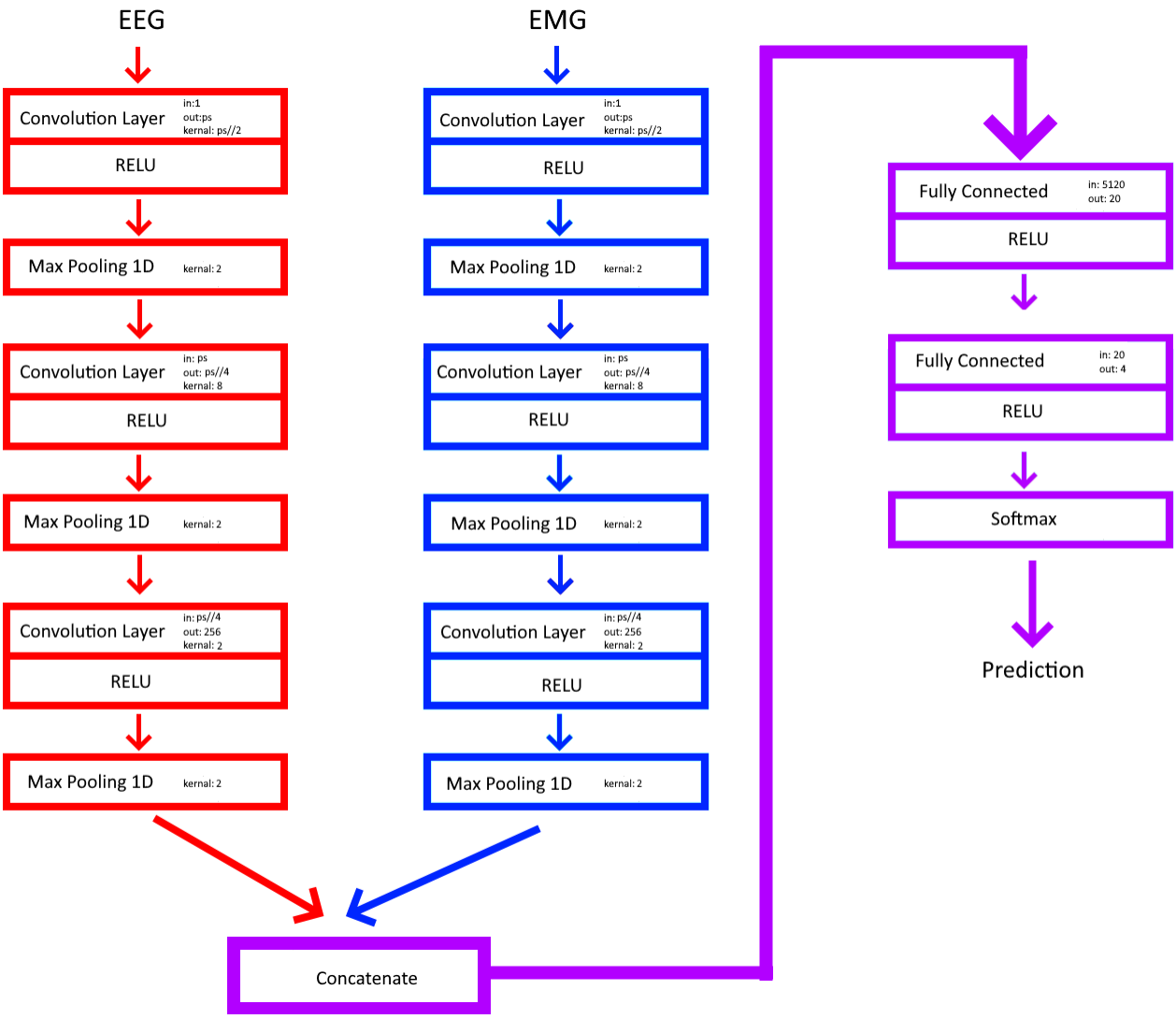


Figure 4: The architecture of this model visualized

signals represent different types of information that don't have a direct relationship with each other.

The basic philosophy that underlies the structure of our convolutional layers is to originally generate enough filters to pick up on wave patterns that occur at any position in an entire period of sampled points. To accomplish this, we set the number of output channels in the first layer equal to period size and the kernel size equal to period size divided by 2. In the two subsequent layers, we chose to decrease kernel size with each layer to pick up on more specific features of the data.

For optimization, we utilize stochastic gradient descent (SGD). While the calculations may be expensive, we figured that SGD computations coupled with back-propagation would allow us to finalize the best model possible given our resources. We often would set the learning rate rather low or below < 0.1, because we didn't want our model to be too confident in itself. We would like to ideally artifact some periods, and the model learning too fast would be counter-intuitive to that goal. We decided to use some momentum in our calculations for SGD as well. Since computing the gradients is an iterative process, momentum used to remember all the updates of the gradient at every iteration so the calculation for the next update then becomes a linear combination of the currently-computed gradient and the previous update[5]. We figured this feature would be important so the computations to combat potential noisy data and excessive, unintended oscillations in the data.

We used cross-entropy loss because it made intuitive sense because that's the loss function of choice for optimizing classification models, like ours!

From there, we also gave some weights to each class to encourage REM classification, which is an issue we'll expand upon later on. The batch size and period size are parameters the user can provide, but we've seen great results with batch sizes between 100-128 and period sizes of 200 (which is approximately a period of a second long). Additionally, we also provide a confidence level to our model during evaluation. We calculate how confident our model is in its prediction for a period, and if that confidence doesn't reach the necessary threshold (which is based on the user-given parameter): the period and its label gets classified as "Artifact".

SingleCNN

We created a CNN that is identical to the DoubleCNN, but only takes 1 channel instead of the two. This is useful to understand the benefits of analyzing one channel over the other, as well as the benefits of using both channels.

When we first started seriously working on this project, we were more focused on creating a usable dataframe from our given data from the research lab. The parsing of the EEG signals and EMG signals was repeatedly difficult. Our philosophy in these early stages was just to create a model that would work with our dataframe once we finally finished parsing - and then from there, we can expand our model further. This model, which we've dubbed "SingleCNN", is merely a convolutional neural network that only takes in one input channel. The one input channel it takes in is the raw EEG signals, which are split into periods as discussed in the Parsing section of this report.

We decided EEG signals would be more relevant than EMG signals due to multiple biological factors[6]. EEG signals are used more for neurological studies and analysis, which is more of what our task pertains to. While EMG signals can also help us understand sleeping patterns in mice, they're mostly used for studying the behaviors of skeletal muscles. Thus, our usage of EMG signals in the DoubleCNN is to give more data that'll support our improvement of the predictions that are made based on the EEG data, hence why we concatenate the CNNs for the EEG data and the EMG data at the end. Because of that, EMG data is considered less important in the grand scheme of things.

A lot of things like the cost and loss function were shared from the DoubleCNN, so our analysis of them carries over and still applies here.

While we have long since abandoned this style of model, we thought it would be interesting to analyze its results and cross-compare it to the performances of both the DoubleCNN and the RNN. Of course, this model is the most flawed since not much thought was put into its architecture. But it's important to reflect on it to re-affirm that not all types of models are well-suited for certain tasks.

RNN

When we gave our middle-of-the-quarter update, we were told by the professor that we should investigte how well an RNN would perform at this task. This made logical sense to us, given the fact that RNNs are well-suited for tasks dealing with sequential data.

By this point, however, we'd already made tremendous progress with the DoubleCNN. We still wanted to do some exploration on RNNs though. As such, because we didn't have enough time to devote to working on it, we decided to make a single RNN that works with EEG signals (EEG signals were chosen for the same reasons mentioned in SingleCNN) only because we weren't quite sure how concatenation of RNNs would be best implemented.

There's not too much complexity to our RNN because there aren't many layers (an RNN layer and a linear layer) . We still have to think about the typical things like our fine-tuned hyperparameters, but the input size to the RNN layer and then the final number of classes were evident, so it was just a matter of playing around with what happens in the middle.

For optimization, we use stochastic gradient descent because that was considered the standard. We used cross-entropy loss because it made intuitive sense because that's the loss function of choice for optimizing classification models, like ours! We also implemented artifiacting during evaluation for this model as well using confidence levels.

IV Results

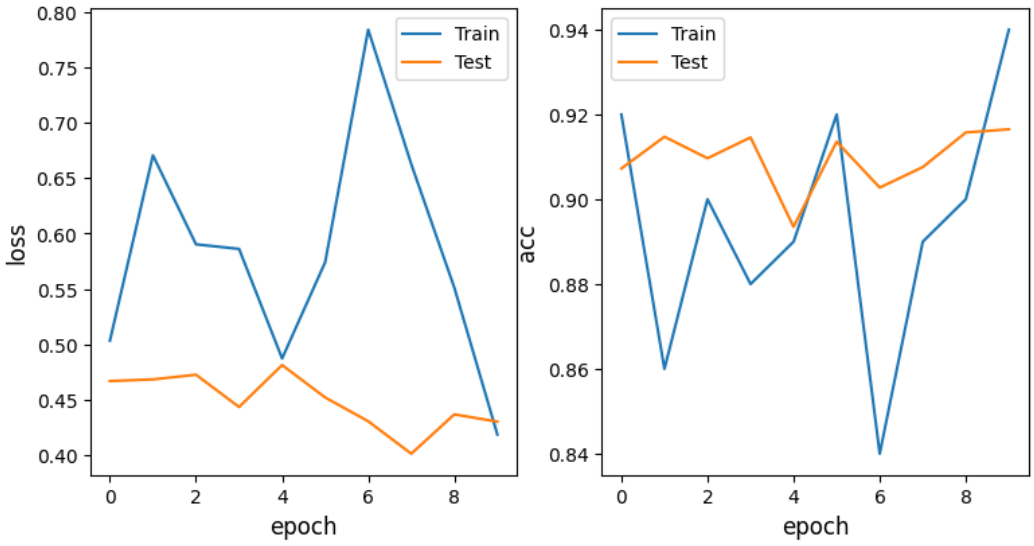
IV.I A Small Comparison of Models

As mentioned before, we developed three models for our chosen task during this course. We'll be comparing the three based on the following parameters that we discussed into detail in the "Methodology section":

- 60,000 periods (approximately 14 hours of data) from one mouse
  - Split into 80% training data and 20% validation data (12,000 periods for validation)
- Period size of 200 (approximately a second long)

- Learning Rate for SGD: .01 for CNNs, .001 for RNN
- Momentum for SGD: 0.9
- Epochs: 10
- Batch size: 100
- Confidence Level for Artifacting: -0.75

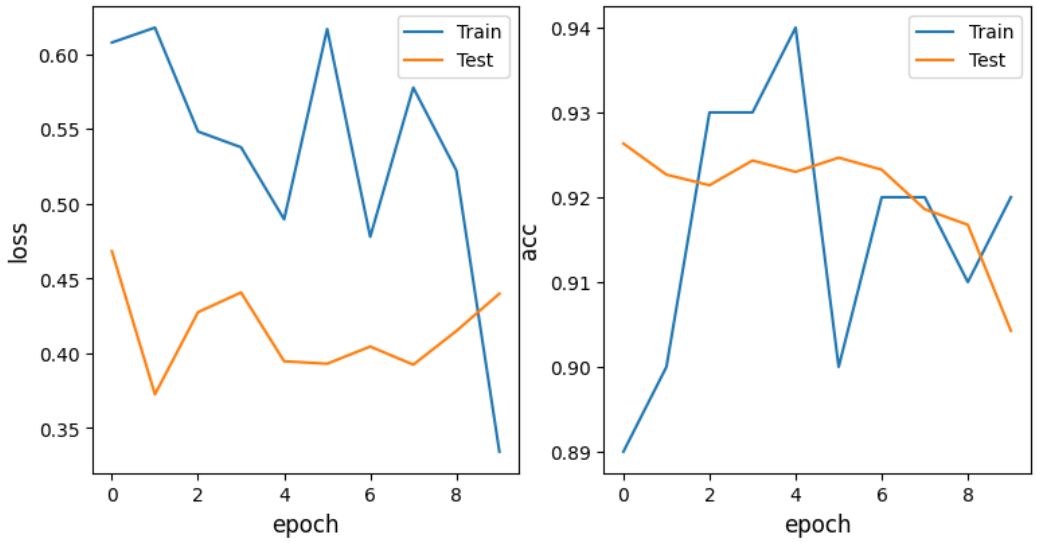
SingleCNN




```
Testing Report for Single CNN for Accuracy, Precision, Recall, and F1-Score
--------------------------------------------------------------------------------------------------

              precision    recall  f1-score   support

         REM       0.00      0.00      0.00       465
     Non-REM       0.91      0.94      0.93      3754
        Wake       0.92      0.96      0.94      7485
    Artifact       1.00      1.00      1.00       296

    accuracy                           0.92     12000
   macro avg       0.71      0.73      0.72     12000
weighted avg       0.88      0.92      0.90     12000

Classified Periods:
Class 3, Wake: 7814
Class 2, Non-REM: 3890
Class 4, Artifact: 296
```

Figure 5: Pictures of training/validation loss + accuracy over epochs and testing report for validation set for each class on last epoch

DoubleCNN

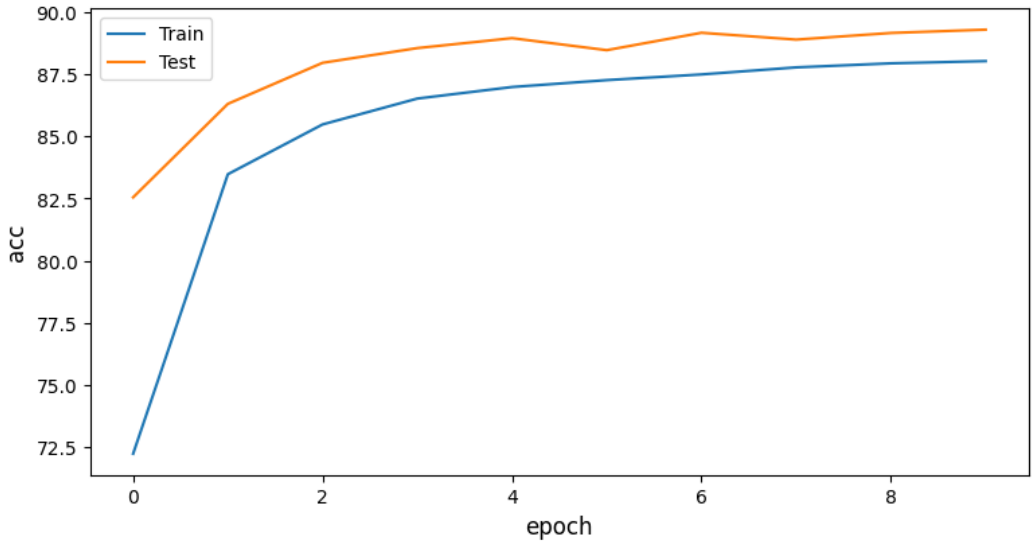



```
Testing Report for Dual CNN with Fourier Transform for Accuracy, Precision, Recall, and F1-Score
--------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

         REM       0.12      0.03      0.05       489
     Non-REM       0.90      0.95      0.92      3681
        Wake       0.92      0.93      0.93      7591
    Artifact       1.00      1.00      1.00       239

    accuracy                           0.90     12000
   macro avg       0.73      0.73      0.73     12000
weighted avg       0.88      0.90      0.89     12000

Classified Periods:
Class 3, Wake: 7722
Class 2, Non-REM: 3904
Class 4, Artifact: 239
Class 1, REM: 135
```

Figure 6: Pictures of training/validation loss + accuracy over epochs and testing report for validation set for each class on last epoch

You'll notice the accuracy is slightly lower than the SingleCNN overall, but it does a lot better at REM classifications and large datasets/multiple mice - which is why we chose to focus on this model in the end.

RNN



Testing Report for RNN with EEG signals

	precision	recall	f1-score	support
REM	0.00	0.00	0.00	456
Non-REM	0.86	0.92	0.89	3714
Wake	0.91	0.93	0.92	7689
Artifact	1.00	1.00	1.00	141
accuracy			0.89	12000
macro avg	0.69	0.71	0.70	12000
weighted avg	0.86	0.89	0.88	12000

Figure 7: Pictures of training/validation accuracy over epochs and testing report for validation set for each class on last epoch

IV.II More on DoubleCNN Results

The double CNN works pretty well, as we saw before. However, there's some noticeable flaws if we look at Figure 6.

The DoubleCNN really struggled to accurately predict REM sleep. The problem is somewhat caused because REM is underrepresented in our data (usually less than 10%). It's the least common stage of sleep as we discussed in our introduction. But it's also just plain difficult in classify REM - even human researhers struggle with.

Our first solution to this was adding class weights onto our loss function (encouraging choices of the REM class). We thought if we simply encouraged more REM classifications, we could start predicting it.

```
...
    Test set: Average loss: 0.4776, Accuracy: 17603/20178 (87%)

    Testing Report for Dual CNN with Fourier Transform for Accuracy, Precision, Recall, and F1-Score
    ---------------------------------------------------------------------------------------------------
                  precision    recall  f1-score   support

             REM       0.06      0.08      0.07       748
         Non-REM       0.92      0.91      0.92      5702
            Wake       0.90      0.91      0.91     13205
        Artifact       1.00      0.54      0.71       523

        accuracy                           0.87     20178
       macro avg       0.72      0.61      0.65     20178
    weighted avg       0.88      0.87      0.87     20178

    Classified Periods:
    Class 3, Wake: 13369
    Class 2, Non-REM: 5617
    Class 1, REM: 907
    Class 4, Artifact: 285
```

```python
def train_model(epochs, model):
    print("device:", device)
    model.to(device)
    model.train() # set model to training mode

    # our class weights to encourage underrepresente data like REM
    class_weights = torch.tensor([0.9, 7.0, 0.7, 0.5, 0.9], device=device)

    loss_fun = nn.CrossEntropyLoss(weight=class_weights) #define a loss function object
```

Figure 8: Testing report for encouraged REM classifications

Using this method, we can get the model to classify a correct proportion of REM periods for the dataset: (907 classified, 748 in dataset)! However it's clear we're just forcing the model to guess REM stages, since the REM precision is atrocious, and the model accuracy drops significantly. Maybe in the future we need to combine this solution with other solutions to under-represented data.

We also tried multiple datasets from multiple mice on the DoubleCNN. It's as easy as processing each dataset and concatenating them! We hoped this would increase our REM accuracy, since the model would have more REM data to learn from. We also wanted to see how it would perform with larger and more unpredictable datasets.

For this we got an accuracy (with default settings described earlier, just with more periods and 5 epochs) of 86%. Unfortunately many of the new datasets/mice used have an issue of "baseline drift" where the channel readings drift higher or lower over time, however when we

concatenate them like this with other data, we get pretty acceptable accuracy of 86%. We also are predicting REM at a better rate than ever! The precision is still really bad, but it shows promise.

```
Test set: Average loss: 0.4304, Accuracy: 34834/40718 (86%)

Testing Report for Dual CNN with Fourier Transform for Accuracy, Precision, Recall, and F1-Score
-----------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

         REM       0.28      0.01      0.01      3530
     Non-REM       0.95      0.92      0.93     16320
        Wake       0.80      0.97      0.87     20333
    Artifact       1.00      0.27      0.42       535

    accuracy                           0.86     40718
   macro avg       0.76      0.54      0.56     40718
weighted avg       0.82      0.86      0.82     40718

Classified Periods:
Class 3, Wake: 24728
Class 2, Non-REM: 15753
Class 4, Artifact: 142
Class 1, REM: 95
```

Figure 9: Testing report for training/validation on three mice

Another tool we implemented into our model was the **confidence level**, as we discussed before. This is a weight in the final softmax layer that classifies anything the model is unsure about below this level. Since the artifact class is always correct, this basically trades increased accuracy for increased artifact periods.

Lowering our confidence level a bit allows us to artifact periods we're less confident with. This inflates our accuracy to 91%, at the cost of around 15% artifacting, which isn't bad! Unfortunately this removes all of our REM classifications because they're the classification the model is least comfortable with. This will always be the result unless we can teach the model to accurately classify REM.

```
Test set: Average loss: 0.4222, Accuracy: 37101/40718 (91%)

Testing Report for Dual CNN with Fourier Transform for Accuracy, Precision, Recall, and F1-Score
---------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

         REM       0.00      0.00      0.00      2621
     Non-REM       0.97      0.97      0.97     14707
        Wake       0.84      0.98      0.90     16807
    Artifact       1.00      0.96      0.98      6583

    accuracy                           0.91     40718
   macro avg       0.70      0.73      0.71     40718
weighted avg       0.86      0.91      0.88     40718

Classified Periods:
Class 3, Wake: 19613
Class 2, Non-REM: 14771
Class 4, Artifact: 6334
```

Figure 10: Testing report for training/validation on three mice with lowered confidence

Another notable result is that using lots of data seems to compensate for the damages of baseline drift! To save on computational costs, for this next experiment: we chose 3 mice, with 2 of them having baseline drift. The 2 with baseline drift have around 70% accuracy when run through the DoubleCNN alone. However, when put together with no extra settings, we have a default of 85% accuracy, which is great! This shows some real benefits to using larger amounts of data, and shows our model is rather robust against noise.

```
...
    Test set: Average loss: 0.4554, Accuracy: 52101/61293 (85%)

    Testing Report for Dual CNN with Fourier Transform for Accuracy, Precision, Recall, and F1-Score
    ---------------------------------------------------------------------------------------------
                  precision    recall  f1-score   support

             REM       0.31      0.02      0.03      5547
         Non-REM       0.94      0.93      0.94     26552
            Wake       0.78      0.95      0.86     28333
        Artifact       1.00      0.36      0.53       861

        accuracy                           0.85     61293
       macro avg       0.76      0.57      0.59     61293
    weighted avg       0.81      0.85      0.81     61293

    Classified Periods:
    Class 3, Wake: 34564
    Class 2, Non-REM: 26117
    Class 4, Artifact: 314
    Class 1, REM: 298
```

Figure 11: Testing report for training/validation on three mice with two mice having baseline drift

Why The Results Are The Way They Are

Precision, for each classification, is calculated as:

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

We intended for "Artifact" to have a 100% precision, because ultimately there's no such thing as a false positive for something being labeled "Artifact". Thus, our results for precision are artificially boosted in that way. REM is rather low because nothing is getting classified as REM to begin with usually without intervention.

Recall, for each classification, is calculated as:

$$2 \times \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Ideally, we would like to focus on maximizing recall because we would like to minimize false negatives the most (false positives are fine in the case of "Artifact", for example. However, due to the unbalanced datasets, this is lowered significantly because so many periods are getting classified as anything but REM. This is why we focused on developing the DoubleCNN so much, because it had the best recall of the three models in general.

F-1 Score, for each classification, is calculated as:

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

V Limitations

V.I Datasets

The EEG and EMG signals can change drastically based on how the electrodes are implanted in the brain. Our datasets and data collection prospects are limited to UCSC's Neuroscience data. If we venture out, we would introduce many other variables.

V.II Models

As previously stated, our models struggle to classify REM based on its underrepresentation, and inherent difficulty to classify. This is something that may be overcome with more research and measures. We already can use some weighting to encourage REM classifications, but we may need to train on a more equal sample size before training on the whole dataset.

The models also need to contend with baseline drift of many datasets, where the EEG and EMG signals float around to different baselines over the course of the training period. This is definitely an issue when run with small amounts of data, but may be mitigated with large datasets.

Overall

At the end of the day, the majority of our current issues all trace back to lack of hardware and resources. It's important to note there's so many variables that need to be managed. The genders of the mice, their health histories, and where the recording devices are placed can all throw off expected results. What we need is a powerful GPU to handle training for potentially days on end on hundreds of mice to protect our results against these factors.

We recognized this early on as a flaw even from our most initial course proposal. The research on this subject that had been conducted prior used "NVIDIA Quadro RTX 8000 with 48 GB memory"[7]. For reference, this singular component alone costs over $6,000. Our competitors have far better rigs and materials than us. However, we still believe we developed a rather robust model regardless - and we look forward to developing it more in the near future when we have access to the necessary materials. With that, then we can finally address the issue of lack of REM data points to analyze and learn from. At the moment, it's too computationally intensive for us to attempt using SMOTE[8] or clustering[9] - which are the normal techniques to combating these problems.

Conclusion

Our project aimed to develop an automated sleep stage scoring model using deep learning techniques for mice based on EEG and EMG signals. We successfully designed and im-

plemented a real-time sleep stage classification model using a convolutional neural network (CNN) that demonstrated some robustness against noise and individual variations in mice. Our model showed promising results in detecting wake and non-rapid eye movement sleep (non-REM), but struggled detecting rapid eye movement sleep (REM) stages.

The significance of our work lies in addressing the time-consuming and tedious process that is mice sleep scoring, it requires manual intervention and specialized knowledge. By automating this process, researchers can save valuable time and focus their efforts on more meaningful tasks. Our project contributes to the field of deep learning by providing a publicly available method for sleep stage scoring in mice. All of our work is commented and released on GitHub. When we get around to writing our README file, it will be very accessible for others to learn from!

While our model demonstrated high accuracy in classifying sleep stages, we acknowledge certain limitations that can be addressed in future work. One limitation is the woefully underrepresented REM class, which hurts our overall accuracy. Further research and measures, such as training on larger but more balanced sample size combined with classification weighting, could improve the model's performance in this area.

Another challenge is dealing with baseline drift in EEG and EMG signals, which can affect the model's accuracy when trained on small datasets. However, larger datasets seem to compensate for this issue, highlighting the benefits of using more extensive data.

In the future, our work can be extended and improved in several ways. Some shorter term improvements are to use our RNN to capture sleep stage transitions - which is an approach which may yield better accuracy detecting REM. We also can incorporate more datasets to build a more robust model.

In the longer term, most of us plan on continuing the project to assist the Neuroscience lab. We have plans to add a functional GUI where we can tweak model parameters, and choose training datasets. We plan on tweaking the model and its artifacting to achieve nearly 100% accuracy. Finally, we plan on making the model actually print out usable results for the neuroscience team, so they can go back and classify the artifacted portions.

Here is the GitHub link with all of our work, please check it out! There are currently some abandoned models and dead code leftover at the moment, we've made 70 commits so far. Our DoubleCNN is LilCNN.ipynb, our RNN is LilRNN.ipynb, and our SingleCNN is SingleCNN.ipynb (restored from an old commit in a separate file).

Contributions

Here is a list of all the contributions of our five members.

1. **Adam Hammond:**
    - Communicated with neuroscience lab, organized goals of the project
    - Co-wrote a large portion of input parsing functionality.
    - Tweaked hyperparameters and performed ran lots of tests on DoubleCNN
    - Co-wrote report and presentation
2. **Audrey Ostrom:**
    - Co-wrote presentation and report
    - Proofread and rewrote the report in LaTeX for clarity and organization
    - Devised the architecture for the DoubleCNN and RNN and then wrote a large portion of the code for both
    - Helped fine-tune artifacting for a 100% precision.
    - Re-factored the code for training and evaluating for the models to make it easier to create a table of relevant statistics and graphs
3. **Adam Elaidy:**
    - Co-wrote presentation
    - Wrote artifact tolerance dial functionality
4. **Sreyes Venkatesh:**
    - Co-wrote presentation
    - Co-wrote a large portion of input parsing functionality

- Co-wrote a large portion of artifact criteria conditions

5. **Regan Miller:**

   - Drafted initial model for task
   - Co-wrote a large portion of input parsing functionality
   - Handled the Fourier transformations of EEG signals and EMG signals, and the modifications to DoubleCNN to account for that

---

Special Citations

We would like to give a special citation to the example repository for our class, linked here. We specifically utilized the examples titled 06-convnet.ipynb and 08-seq_classification.ipynb, as they provided useful practical examples of how CNNs and RNNs are developed. Additionally, we found the latter example particularly helpful for figuring out how to graph our results. Thank you to all the wonderful course staff for providing us with these examples so we can understand how things are written in PyTorch.

References

[1] L. E. McKillop and V. V. Vyazovskiy, "Sleep and ageing: from human studies to rodent models," *Current Opinion in Physiology*, vol. 15, pp. 210–216, Mar. 2020, doi: 10.1016/j.cophys.2020.03.004.

[2] H. Norimoto *et al.*, "A claustrum in reptiles and its role in slow-wave sleep," *Nature*, vol. 578, no. 7795, pp. 413–418, Feb. 2020, doi: 10.1038/s41586-020-1993-6.

[3] S. Overeem, G. J. Lammers, and M. Tafti, "Disorders of Sleep and Circadian Rhythms," in *Elsevier eBooks*, 2007, pp. 409–426. doi: 10.1016/b978-012369509-3.50028-7.

[4] M. Yamabe, K. Horie, H. Shiokawa, H. Funato, M. Yanagisawa, and H. Kitagawa, "MC-SleepNet: Large-scale Sleep Stage Scoring in Mice by Deep Neural Networks," *Scientific Reports*, vol. 9, no. 1, Oct. 2019, doi: 10.1038/s41598-019-51269-8.

[5] "On the importance of initialization and momentum in deep learning — Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28." https://dl.acm.org/doi/10.5555/3042817.3043064 (accessed Jun. 9, 2023).

[6] C. Brambilla, I. Pirovano, R. M. Mira, G. Rizzo, A. Scano, and A. Mastropietro, "Combined Use of EMG and EEG Techniques for Neuromotor Assessment in Rehabilitative Applications: A Systematic Review," *Sensors (Basel)*, vol. 21, no. 21, p. 7014, Oct. 2021, doi: 10.3390/s21217014.

[7] T. Tezuka, D. K. Rai, S. Singh, I. Koyanagi, T. Naoi, and M. Sakaguchi, "Real-time, automatic, open-source sleep stage classification system using single EEG for mice," *Scientific Reports*, vol. 11, no. 1, May 2021, doi: 10.1038/s41598-021-90332-1.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[9] "What is Unsupervised Learning? — IBM." https://www.ibm.com/topics/unsupervised-learning (accessed Jun. 9, 2023).