An Analysis of Electroencephalogram (EEG) with Machine Learning

Isra Jime, Jesse Emery, Nhi Phan

MacEwan University

CMPT 496: Capstone

Dr. Dana Cobzas

April 16, 2024

CMPT 496: Capstone Report

Isra Jime, Jesse Emery, Nhi Phan

CMPT 496: Capstone Report

Isra Jime, Jesse Emery, Nhi Phan

1. Introduction

Our capstone project was done in collaboration with Dr. Cameron Hassall from the Psychology department at MacEwan University. Our data was based on one of Dr. Hassall's papers on "Task-level value affects trial-level reward processing" (Hassal, C, 2022), where he wanted to determine if the Anterior Cingulate Cortex was responsible or involved in decision making. To determine this, the task sequence as shown in Figure 1 was carried out 427 times using 12 participants over a 52 minute period. While the participants completed these tasks, brain activity was being measured using an electroencephalogram (EEG). For our project, the goal was to train a machine learning model to accurately classify an EEG event after training on past events. In greater detail, we focus on the brain signal when the participant hit the left or right button in response to the stimulus which are colored shapes.

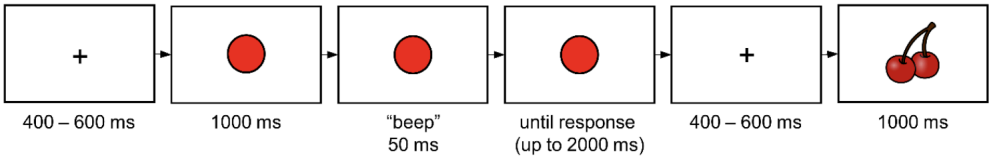

Figure 1. Experiment tasks example

2. Methods

2.1 *Data*

The data used in our project was taken from a single EEG recording from subject 27 in the previously mentioned experiment by Dr. Hassall. The EEG recording consists of 31 channels, corresponding to nodes placed on the subject's scalp to measure the brain's electrical activity. The electrical activity is measured in microvolts (μV) at a sampling rate of 1000 Hz (1000 samples per second). The EEG recording for this subject had 3161620 individual sample points, or just over 52 minutes at 1000 samples/second. The subject performed 427 tasks over the duration of the experiment. Each task was split into 5 events: fixation cross, cue, beep, motor response, and outcome, as seen in Figure 2. In a separate file, the start of each event is recorded as an annotation linking the event onset time with the specific event type. Our project focused solely on the motor response, which was a left or right response from the subject that shows as an event in the EEG data after the brain responded when pressing the left or right button.

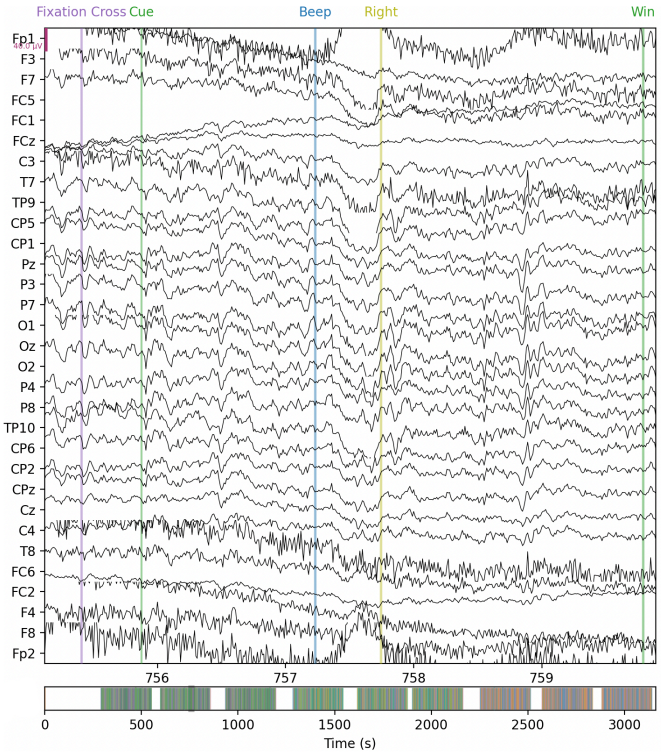

Figure 2. EEG of participant 27, depicting each of the five events in a task

2.2 *Preprocessing*

For the EEG data to be usable in machine learning, we first needed to preprocess the raw data. To do this we used MNE, an open-source python package purpose built for analyzing EEG data. This involved limiting the annotations to only the left and right motor response annotations and removing all other annotations from the data. The onsets of the remaining annotations were shifted *o* milliseconds forward or backwards, where *o* was one of the parameters used in tuning our machine learning models. By shifting the onset, we can train on a range of event start times, which may make a difference in finding the best features. We then resampled the raw data to a lower resample frequency *f*, another parameter used in tuning our machine models. The idea was to resample the raw data to speed up machine learning, while keeping enough features to train on. MNE resamples by applying a low-pass FIR filter and sub-selecting samples from the data ("FAQ – MNE 1.6.1 Documentation", 2024). The last parameter used in preprocessing to tune our model was the number of event samples *n* taken at each event onset. The number of samples taken *n* were affected by the downsample frequency *f*, since downsampling meant having less samples per second, resulting in n' event samples as seen in (2).

$$\frac{n'}{f} = \frac{n}{1000} \tag{1}$$

$$n' = \frac{n * f}{1000} \tag{2}$$

For example, if $n = 400$ and $f = 200$, $n' = 80$. This makes sense as $f = 200$ Hz is 1/5th the original frequency of 1000 Hz, and $n' = 80$ is 1/5th the $n = 400$ samples. Both n and n' samples span 400 milliseconds of samples in their respective sample rates.

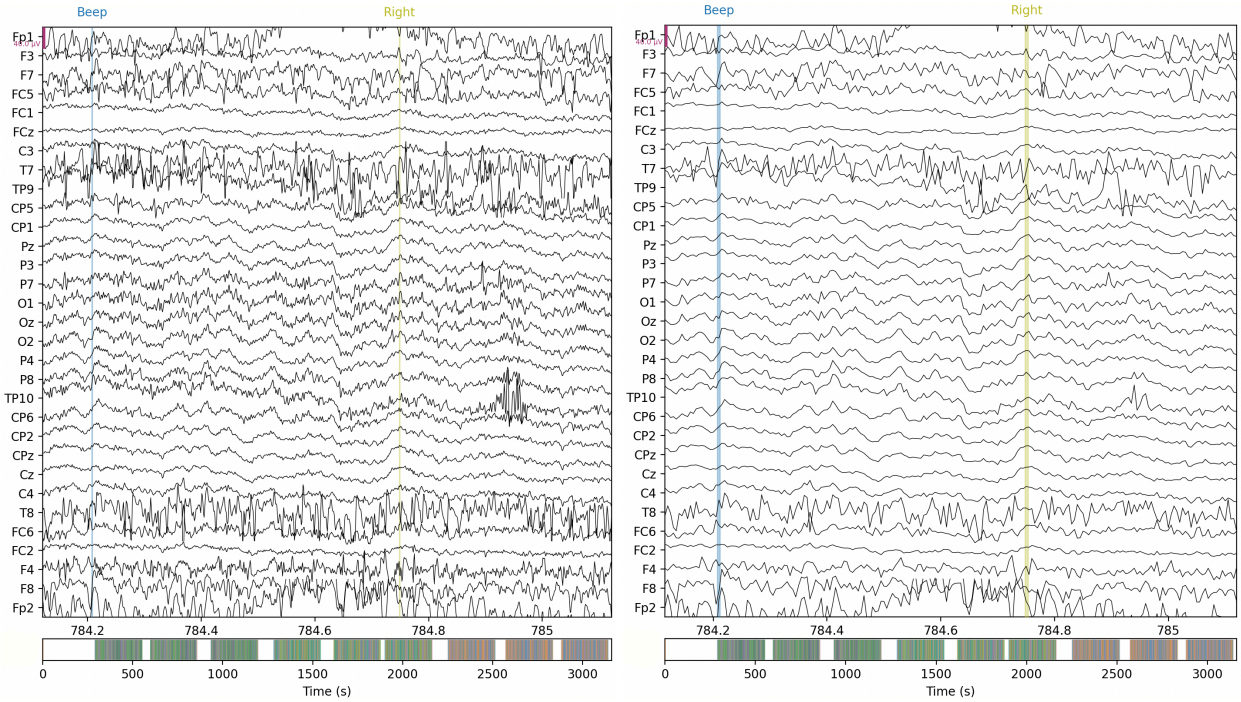

Figure 3. Original data at 1000 Hz on left, and resampled data on right.

After offsetting the annotation onsets and resampling, each event is cropped to n' samples, and saved into a list of the crops. The function to crop raw data in MNE works by cropping a number of seconds from a start time, with an option to include the last sample instead of excluding it. Since resampling reduces the number of samples per second, the onset time and end time might lie between two sample points in the resampled data. This can cause the cropping function to sometimes crop 1 too many or too few samples compared to n'. If it crops 1 too many samples, we re-crop the previously cropped event, and if it crops 1 too few, we re-do the previous crop and include the last sample. This ensures each event has the same number of samples, which is necessary when inputting the data into our machine learning models.

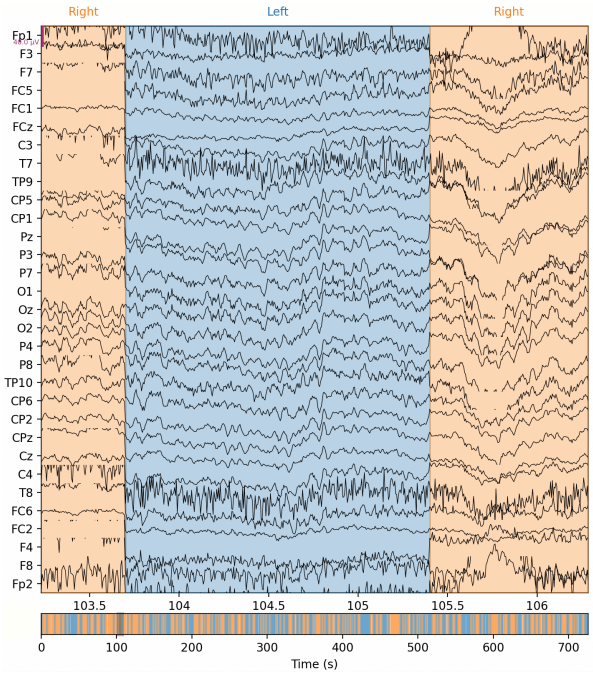

Figure 4. Portion of combined 427 left and right motor response events.

All of the crops in the list of crops are combined together to create a single set of raw data, as seen in figure 4. At this point, the data is converted to a numpy array with a shape of 31 channels down × (427 events × n' samples per event) across. The numpy data should be either 427 down × (31 × n') across for 2D input, or 427 down × (31 down × n' across) for 3D input as seen in figure 5. We also record the corresponding label for each of the 427 events as either 0 for left or 1 for right, since this is only a binary classification.

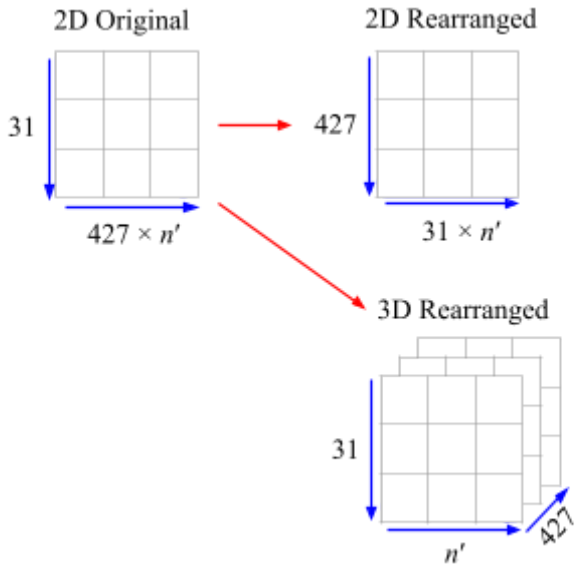

Figure 5. Numpy data of left right data, rearranged into 2D and 3D.

2.3 *Data Randomization and Splitting*

To ensure the same portion of tasks are used for training the models, we randomized the task order and stored it in a file. It is important to randomize the order, as we are unsure if the motor response will look different over the course of the lengthy experiment. We can then read the task order file, using the first 60% for training, the next 20% for validation, and the last 20% for testing. Using a custom function for this ensures consistency in always using the exact same tasks. The appropriate labels for each portion are also returned. By splitting the data into three portions, we can train the model on the training portion by tuning the *o*, *f*, and *n* parameters, finding the model with the most accurate classification on the validation data. Once the parameters with the best validation accuracy is found, we finally test the model on the test data for our final results.

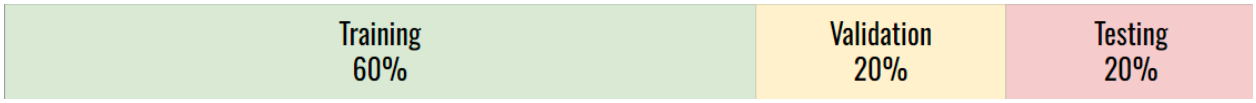

Figure 6. Splitting up the data into training, validation, and testing.

2.4 *Traditional Machine Learning*

We used the python machine learning package scikit-learn to perform traditional machine learning using Linear Discriminant Analysis (LDA), which was suggested as a good classifier in the MNE tools. We did not modify any parameters when creating the classifier. The classifier was trained by fitting it onto the training data, then used to predict the labels of the validation data. The predicted labels were then compared with the known labels to get a classification accuracy. The trained classifier was also used to predict the labels on the training data to ensure it was working correctly.

To find the best parameters to when resampling and cropping the raw data, we tested a range of values in nested loops for each of the *o*, *f*, and *n* parameters, applying LDA for each parameter combination. We tested various combinations of ranges of the *o* range from 0 to 900 ms, the *f* range from 50 to 1000 Hz, and the *n* range from 300 to 2100 samples. After first testing a coarse search with a large step, we refined the search with smaller steps to narrow down the best parameters.

2.5 *Deep Learning*

We used both TensorFlow and PyTorch to create our models using the deep learning model Convolutional Neural Network (CNN). The model consists of 2 convolutional layers with a max pooling layer after each one followed by 2 fully connected layers. For our convolutional layers, we used a 3x3 kernel for both. The first convolutional layer had 8 filters with the second one containing 16 filters. The models were fit using the training and validation data and labels and then evaluated with the test data and labels. The same

models were created and trained in both TensorFlow and PyTorch.

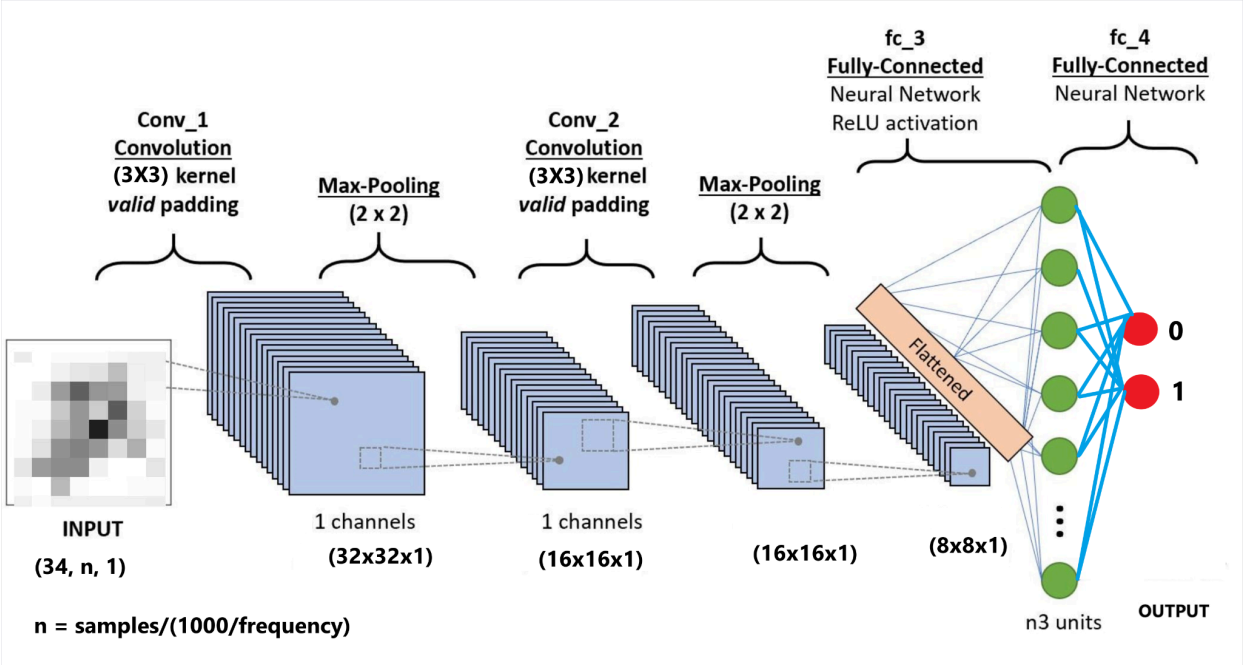

Figure 7. Convolutional Neural Network's Architecture.

Figure 7 shows the architecture of the CNN network that we used in great detail. The output of the first convolutional layer (conv_1) is called feature maps because usually it learns about the features of the input data (that resembles image's data) and outputs that to the next layer - pooling layer. Pooling layer is to downsample the previous layer feature map to half the size.

Fully connected layers are the final 2 layers in a network. It takes the features obtained by previous convolutional and pooling layers as input to produce prediction. This is where the magic happens! Like skilled workers constructing walls, convolutional filters slide across the data, detecting patterns and extracting features. For classification, the output of the final fully connected layer is applied with a ReLu function to produce probability-like classification.
We test with a number of filters in between layers and padding and stride and filter dimensions are taken on doing a number of experimentations.

3. Results

3.1 *Linear Discriminant Analysis*

The parameters that yielded the highest accuracy score on the validation data labels when tuning the LDA were f = 180 Hz and n = 1620 samples, with no increase in score found when changing the offset (o = 0 ms). The validation accuracy was 74.1%, with a training accuracy of 83.2%, indicating the model seemed to train properly. However, the testing accuracy was quite low at only 51.2%, meaning our classifier was not predicting the testing labels better than chance (50%).

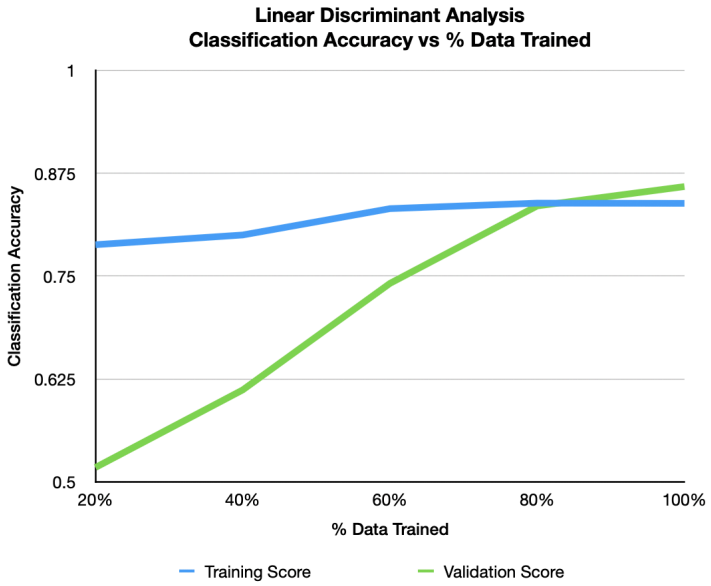

Figure 8. Score difference decreases as % of data trained on increases.

We confirmed the classifier should have trained correctly by comparing the classification accuracy for the training and validation scores for multiple % of the data trained on. The validation data remained consistent with the 20% of tasks after the first 60% as shown in figure 6, while the training data started from the first 20% of the randomized tasks to all 100% of them. The gap closes between them as the % data trained on increases, as seen in figure 8.

3.2 *Convolutional Neural Network*

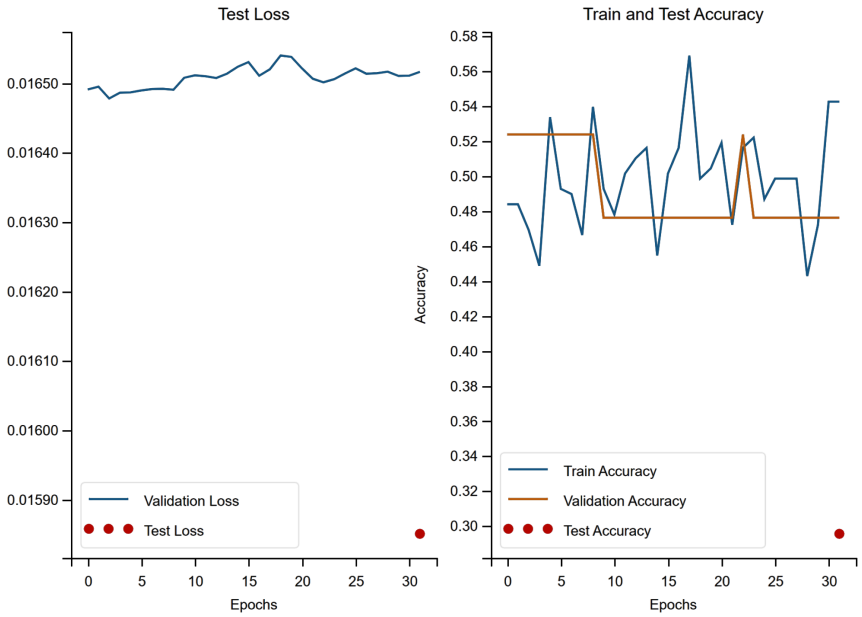

Figure 9. Accuracy plot of CNN training and test data

Our result for CNN is not a successful train. We used the same parameters that yielded the highest accuracy for the LDA, where $f = 180$ Hz and $n = 1620$ samples. Although this gave us the highest values, our results indicated that the model was not properly trained. For training, we had around a 48-50% accuracy with the validation data having an accuracy of around 50% as well. However, when running the testing data, the values fluctuate drastically. This number means that the model is extremely sensitive to noise and didn't train at all. On some runs, We would get an accuracy of 29% and sometimes the accuracy goes up to 70%. There are two indicators here that determine that our model is not accurately trained. The first being that the accuracy for the training data set is below 50%. The second is that the testing data accuracy should not be greater than the accuracy of the training data. Figure 9 shows the accuracy plot we get for the test and training. You can see that the graph is sporadic, further indicating an ill-trained model.

4. Discussion

4.1 *Linear Discriminant Analysis*

While we aren't certain exactly why LDA did not perform well on the test data, there are some possibilities to consider. The first is that the classifier may be overfit, indicating there was not enough data from the 427 tasks. Another possibility could be we were not decomposing the signal using the Common Spatial Pattern as MNE suggests, which we were unable to accomplish. Lastly, we might not have been preprocessing the data properly, as we did not remove noise, look for "bad" events, or get rid of unnecessary frequencies. This could involve a Fourier transform to decompose the frequencies, or a spectrogram analysis.

4.2 *Convolutional Neural Network*

There are several reasons that we suspect that the CNN we performed did not work. The first being that when the first convolution is formed, the channels for the electrodes would be lost. The second reason is that A convolutional neural network works where it looks for similarity between neighboring pixels and generates multiple feature representations. In our case, there might be little similarities found and mixing the channels would be unsuccessful. Unlike regular image classifications, channels in EEGs might not be anatomically similar. The channels might be physically close or far. Even if they are anatomically close together, there's a chance that there doesn't exist any correlation at all. Thus, mixing 3 channels of the time produces incorrect data for the neural network to process. For instance, the channels in the figure 10 show that certain triggers will fire at different locations across the brain.

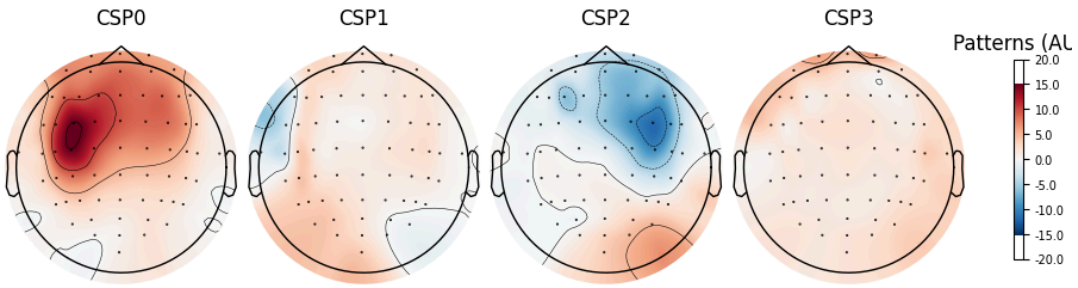

Figure 10. Channels from 1 to 31 on the brain for EEG data

The third reason being that this model is very sensitive to noise. If there's a huge artifact in the data, when participants scratch their head or a node falls out, this will impact the input and output of CNN.

4.3 *Challenges*

We encountered several challenges while pursuing this project. The first being that most of our time was spent learning about machine learning and the many tools and libraries like MNE. The majority of time was spent on the preprocessing data, rather than the processing. Although the tool MNE is robust, it lacks proper documentation and examples for newcomers, and seems to expect a background in machine learning and require pre-exposure with EEG data. On top of this, EEG machine learning is a topic that is rather niche, meaning that it is difficult to find other projects or tutorials to learn from.

Another challenge we faced was choosing the right neural network to follow. We were between treating the data as an image or as audio. Ultimately, we chose images but these may not have been the best choice. If we treat the data as audio and analyze the frequency decomposition of the data, we might have seen more patterns and take the next step from there.

This challenge is foreseen as we worked with raw, unprocessed data, yet, it is a great learning experience for all three of us in terms of teamwork, planning the route ahead and presenting all the interesting factors that we discovered during the Capstone 496 presentation day.

4.4 *Further Analysis*

If this project were to be elaborated on, there are a few things we believe would improve the results. The first is how we treat the data. When conducting our CNN analysis, we treated the data as 2D data. Instead, we believe it would be better to work in 1D by flattening the data and changing the kernel size from 3x3 to 3x1. This is a similar approach for the Linear Discriminant Analysis method. Next, it's possible that not every channel has a high predictive power. To mediate this, we can use Principal Component Analysis to determine those with the highest predictive power to use in our analysis. Another way to improve the results would be to use different models like the General Linear Model. Lastly, we could treat the data as sound and use frequency decomposition to analyze the relationships between channels in a more coherent and logical way because not all channels might be anatomically .

5. Conclusion

Although some of our results worked in the end, this project presents a significant potential for improvement with further investment of time and resources. Specifically, the result of the Linear Discriminant Analysis shows a promising case for the future enhancement. In retrospect, working with a Convolutional Neural Network (CNN) may not have been the best approach. Moving forward for the next student to take on this project, we suggest investing more in applying further the potential methods we highlighted earlier for preprocessing that could improve our ability to classify EEG events more effectively.

References

Hassall, C. D., Hunt, L. T., & Holroyd, C. B. (2022). *Task-level value affects trial-level reward processing.* NeuroImage, 260, 119456. *https://doi.org/10.1016/j.neuroimage.2022.119456*

Frequently Asked Questions (FAQ) — MNE 1.6.1 documentation. (2024, January 17). https://mne.tools/stable/help/faq.html#what-are-all-these-options-for-resampling-decimating-and-binning-data

Motor imagery decoding from EEG data using the Common Spatial Pattern (CSP) — MNE 1.7.0.dev181+ga6f033168 documentation. (2024, April 13). https://mne.tools/dev/auto_examples/decoding/decoding_csp_eeg.html

Convolutional Neural Network (CNN) : Tensorflow Core. TensorFlow. (n.d.). https://www.tensorflow.org/tutorials/images/cnn

SuperBruceJia. (n.d.). *EEG-DL/Models/Network/CNN.py at master · superbrucejia/EEG-DL. GitHub.* https://github.com/SuperBruceJia/EEG-DL/blob/master/Models/Network/CNN.py