ABSTRACT

IMPLEMENTATION OF IMAGE COMPRESSION AS APPLIED TO ELECTROENCEPHALOGRAPHY IN OPENBCI

BY

Joshua Tellez, B.S.

EE 597

Neural Signal Processing

New Mexico State University

Las Cruces, New Mexico

In this work, we implement an image compression scheme to compress electroencephalographic (EEG) signal data in OpenBCI, an open-source brain-computer interfacing software. With our widget implementation, we introduce the ability to read and write compressed EEG data files in the native OpenBCI user interface. The image compression scheme is modeled from previous work [9], utilizing the JPEG 2000 standard and the *jp2-blk* matrix reshaping method. When testing the proposed compression algorithm on a proprietary dataset, we yield a 2.17:1 lossless compression ratio on average relative to the original EEG data file size, and a 6.3:1 lossless compression ratio on average when employing additional ZIP compression. The reduced file size allows for increased file storage capacity and faster data transfers. The JPEG 2000 image compression standard offers features like progressive decoding and region-of-interest coding, which can be utilized in client–server browsing when handling large files. These benefits are particularly useful in telemedicine and archival applications. We originally planned on implementing browser features in this widget application that would allow the user to view the compressed EEG time series data at a lossy quality, which could theoretically save computational resources, network bandwidth, and decoding time for the user. We were only partially successful on this front due to programming complexity and time constraints.

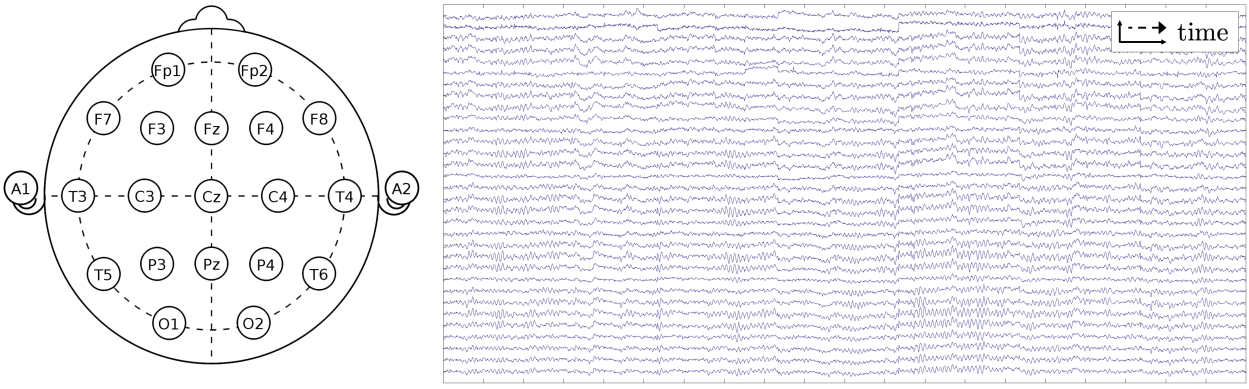

Figure 1: 10-20 electrode diagram and corresponding time series.

1 BACKGROUND

Electroencephalography (EEG) is a method of monitoring electrical activity in the brain, where electrodes sample voltage produced by vicinal populations of neurons over time. Depending on specifications of the hardware, this sampled data can occupy a large amount of storage space. Modern EEG equipment can sample hundreds of electrodes at sampling rates above 1kHz with high bit rate precision, and often this equipment is used to monitor brain activity for hours at a time. So, naturally, data compression becomes a necessary consideration. In medical applications, ethics are prioritized when applying compression to data. Lossy algorithms have the potential to erase patterns of data that are crucial in diagnosing illness and disorder. Lossless algorithms are therefore overwhelmingly preferred due to the lack of standardization present for compressing EEG data [2], though there is research dedicated to quantifying the thresholds in which data of this nature can be discarded without sacrificing performance in the context of an intended purpose [12, 11, 3, 7, 10].

Several compression schemes have been proposed to solve the EEG data storage and transmission problem over the last 26 years, starting with the seminal paper published by Antoniol and Tonella in 1997 [2]. Their experiment involved evaluating the performance of twelve general compression schemes on a dataset of 154 EEG signals, wherein they concluded that the Huffman coding compression method yielded the best balance between computational complexity and lossless compression rate. Following this, an experiment performed by Dauwels in 2012 involved compressing EEG signal data in a variety of tensor arrangements using a set of decomposition models. Here, the author assesses and compares the performance of unique tensor arrangement/decomposition model combinations to ultimately find an optimal system that yields high compression rates for both lossless and lossy cases while maintaining low error for highly lossy cases [4]. Lastly, a work published by Hejrati in 2017 employed a compression algorithm which utilizes blocked k-means clustering and arithmetic encoding to compress EEG signals. The author tests varying block size and cluster quantity to arrive at an optimal solution for yielding the highest lossless compression ratio for a variety of standardized EEG datasets [6]. All of these previous reports motivate and reinforce our objective, which is to implement a standardized, lossless EEG compression algorithm in OpenBCI GUI, an open-source brain-computer interfacing software.

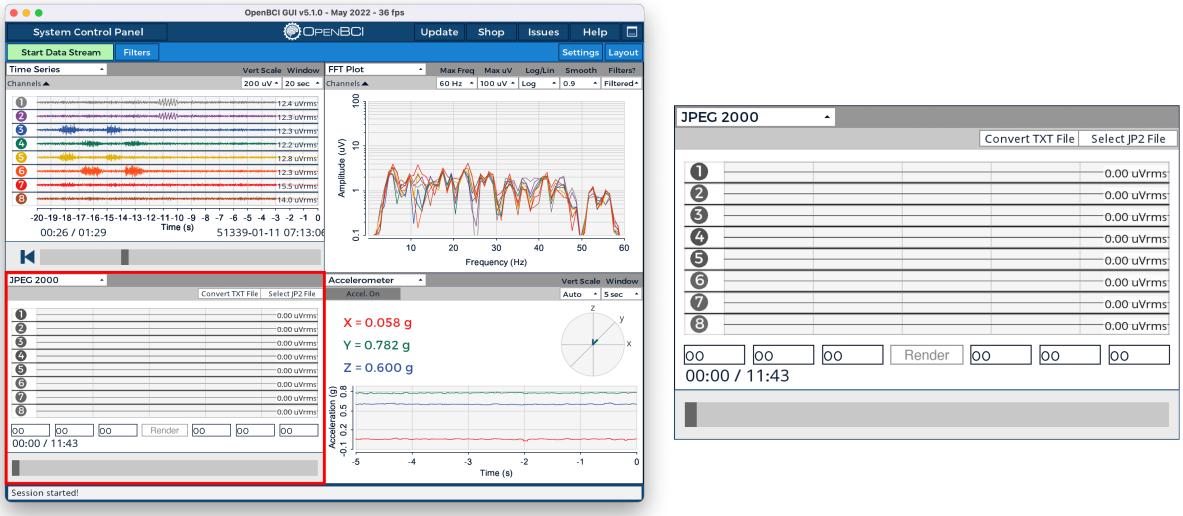

Figure 2: The JP2-EEG widget shown within the OpenBCI GUI interface.

2 PROJECT MOTIVATION

Spatial correlation between electrodes on the scalp often leads to inter- and intra-channel correlation in the EEG signal. With an effective compression algorithm we can exploit this correlation to reduce the file size of EEG data. Through the lens of experiments discussed in the previous section, we created a proof of concept in our own previous work, wherein we compressed EEG data from the Brain Computer Interface Competition IV (BCICIV) dataset using the JPEG 2000 image compression algorithm with the *jp2-blk* pre-processing scheme applied [9]. The *jp2-blk* pre-processing scheme simply rearranges the signal such that temporal blocks of the entire EEG signal are positioned in sequential columns whose temporal direction alternates in a raster scan. A diagram of this process can be seen in Figure 3. We choose to employ the JPEG 2000 image compression algorithm to compress the EEG data because it offers a suite of features that are useful in medical imaging applications while simultaneously yielding noteworthy, consistent, and scalable compression performance [9]. JPEG 2000 is available as an open-source, royalty-free codec and is highly compatible, meaning files can be exchanged universally without the need for paid or proprietary software. For our purposes, the two most pertinent features of JPEG 2000 are progressive decoding and Region of Interest (ROI) coding, both accessible via the JPEG 2000 Interactive Protocol (JPIP). ROI coding enables the user to decode segments of data from the image, and progressive decoding allows the user to access this data at varying compression levels. Progressive decoding and ROI coding are especially advantageous in low-bandwidth cloud applications [1], as they provide a means for the user to browse large EEG data files without needing to access and render the entirety of the file at once, which would otherwise demand lots of network bandwidth and computational resources.

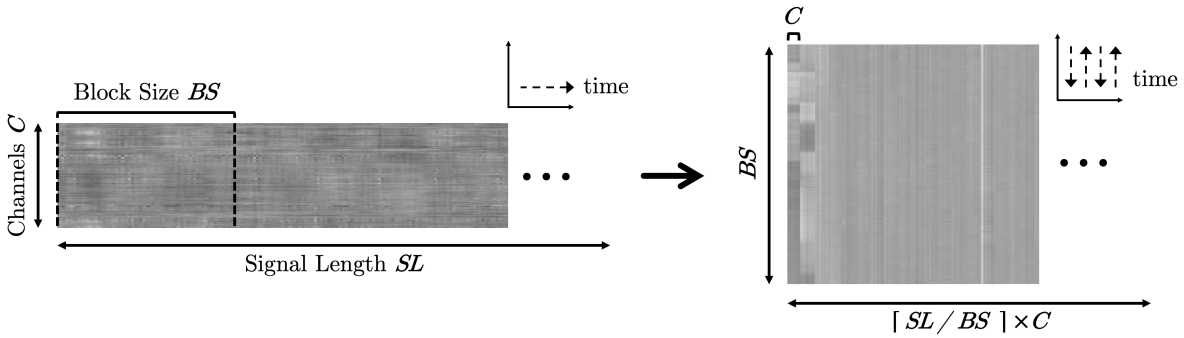

Figure 3: *jp2-blk* pre-processing method diagram.

3 OPENBCI GUI IMPLEMENTATION

In this section, we describe the steps taken to bring the functionality of JPEG 2000 and our *jp2-blk* pre-processing scheme to a real-world application. Specifically, we are interested in adding the functionality of JPEG 2000-based EEG data compression to an application called OpenBCI GUI. OpenBCI GUI is a popular Java-based EEG management software which allows users to record, process, and visualize EEG data in real-time. The primary reason we've chosen OpenBCI GUI as the vessel for this implementation is that it is an open-source system, meaning we have unrestricted access to the source code, allowing for freedom and transparency in programming. The OpenBCI GUI interface is composed of modular elements called *widgets*. Widgets provide lots of different capabilities in this system. For example, the "Time Series" widget, shown in the top-left of Figure 2, displays plots of the electrode time courses. In the case of EEG data playback, we can scroll through the time course, and in the case of EEG headset streaming, we simply see the real-time feed of electrode data being plotted continuously. The widgets are modular, meaning the user can declare which widgets they wish to view at any time in any layout within the OpenBCI GUI window. The OpenBCI Software Development Kit (SDK) provides a template for widget design, which we employ in our development.

3.1 Functional Overview

Our widget is depicted in Figure 4, hereby referred to as *JP2-EEG*. Firstly, we note that the backend of JP2-EEG accesses a fundamentally independent source for the data that it displays. Widgets like Time Series pull data from a global source, meaning the data stream that is active in the program will be broadcasted to all of these widgets synchronously, so the user may process and visualize different facets of the same data simultaneously using the functionality of multiple widgets. This global data stream can originate from a few different sources, namely a live EEG headset (Ganglion, Cyton), a synthetic source, or a playback file. In our application, we are primarily concerned with storage and retrieval of playback sources. To stream a playback source, the user must select and load a playback file. Playback source files utilize a comma-delimited text file format. Upon streaming, the program parses and interprets these text files, which contain all of the EEG data along with some ancillary data like timestamps, accelerometer values, and event markers (per Figure 7).

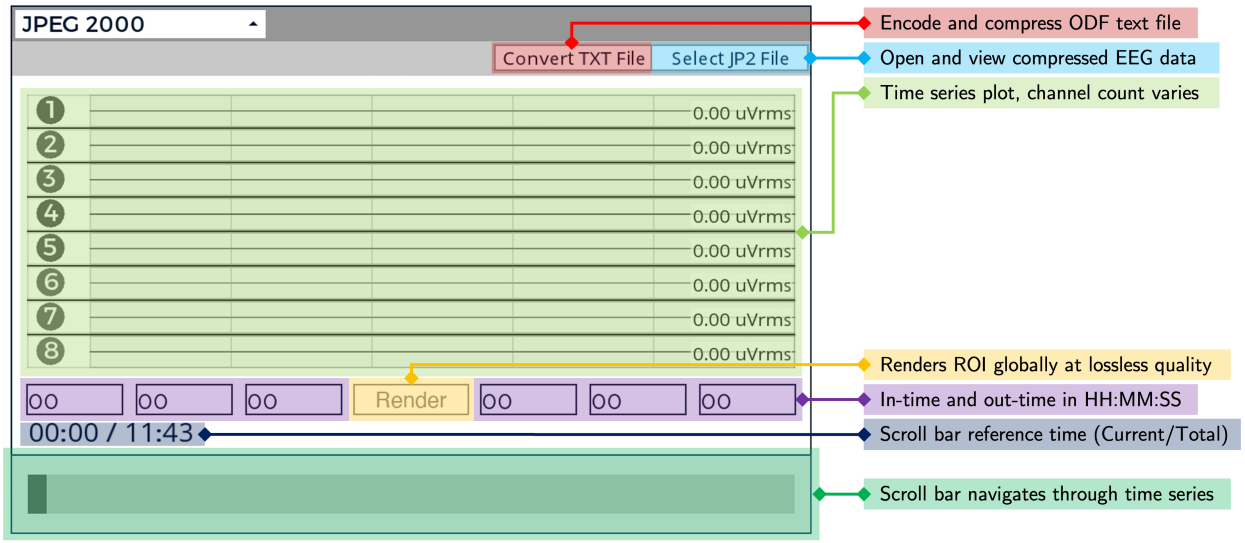

Figure 4: Functional breakdown of the proposed JP2-EEG widget graphical interface

3.1.1 Encoding and Decoding

Upon selecting a file using the "Convert TXT File" button, in the backend of JP2-EEG the program takes the chosen file and compresses the contained EEG data using JPEG 2000 with the *jp2-blk* pre-processing scheme, modified (per the process detailed in section 3.4) to accommodate the data format used by OpenBCI. This EEG image file is then saved to the directory where the original text file is located, along with a modified text file containing the ancillary information sans the EEG data (per the process detailed in section 3.5).

Originally, we intended to program the "Select JP2 File" button to execute a procedure that parses the EEG data from the selected EEG image file into the time series plot in JP2-EEG at a fixed, lossy quality. This would enable the user to browse this data in the JP2-EEG widget without having to use substantial computational resources in the case where the selected file is very large. Once loaded, the user would then select a temporal range of the data using the text boxes below the time series and click the "Render" button to render a lossless segment of the original EEG data in the range spanning from the in-time to the out-time to be broadcast as the global playback source. Unfortunately, this proved to be an arduous pursuit given the limited time we had available for development and the high complexity of the class additions/modifications required to implement this functionality.

Instead, in the current version of the JP2-EEG widget, upon selecting a file using the "Select JP2 File" button, the program parses the EEG data from the selected EEG image file and the ancillary information from the corresponding modified text file to reconstruct the original text file. The reconstructed text file can be read by OpenBCI natively, so the user may browse the data using the existing tools in the program.

3.2 Environments and Dependencies

In order to implement the JPEG 2000 codec, we must utilize third-party Java packages. To enable third-party package implementation, we must modify the development environment. This modification process is discussed in the following subsections.

3.2.1 The Processing Development Environment

Processing is an open-source Java library that allows for easy GUI development in the Java programming language. OpenBCI GUI has been developed entirely within the Processing Development Environment (PDE). The PDE consists of a text editor, a console, a compiler, and a debugger that operate on code written in *sketches*. Sketches are composed of abstracted text files with a *.pde* file extension contained in a sketch folder. Upon compilation, all *.pde* files in the sketch are consolidated into a single *.java* file and this file is interpreted in the Java programming language. In our project, for a couple of reasons, we were unable to develop in the PDE. The native environment does not allow for direct third-party package implementation, and the editor is not conducive to productive development for large-scale projects, as it lacks many basic features like auto-complete, code search tools, and persistent compilation. We therefore sought out tools more compatible with our application.

3.2.2 Java Development Kit

The Java Development Kit (JDK) is the lowest level of abstraction in the context of our program. The JDK is the package that provides us with the basic tools for developing applications in the Java programming language. Processing relies on the JDK to turn code written in the PDE into a program that can run on a Java Virtual Machine (JVM). Depending on the version of Processing used, one must be cognizant of the version of the JDK employed for that Processing distribution. In our case, we are utilizing Processing version 4.0b2, which employs JDK version 11.0.12+7. Thus, we need to ensure that we use the same JDK distribution when we run the OpenBCI GUI Processing sketch outside of the PDE.

3.2.3 IntelliJ IDEA

IntelliJ IDEA is a coding environment that provides effective management tools for large-scale Java developments. The feature provided by IDEA that we are most interested in is *dependency management*, which grants us the ability to keep track of the packages utilized in our program. IDEA enables modification and compilation of the Java source code that is synthesized upon compilation in the PDE (per the process detailed in section 3.2.1) by providing a means to create a virtual Processing environment. To create this virtual environment, we must include all necessary Processing libraries and a corresponding version of the JDK in the project's dependency list. With these files accounted for, we are able to run OpenBCI GUI from the consolidated Java source code in IDEA, meaning we can now implement third-party Java packages in the project. It is from this point that development of the JP2-EEG widget can be fully realized.

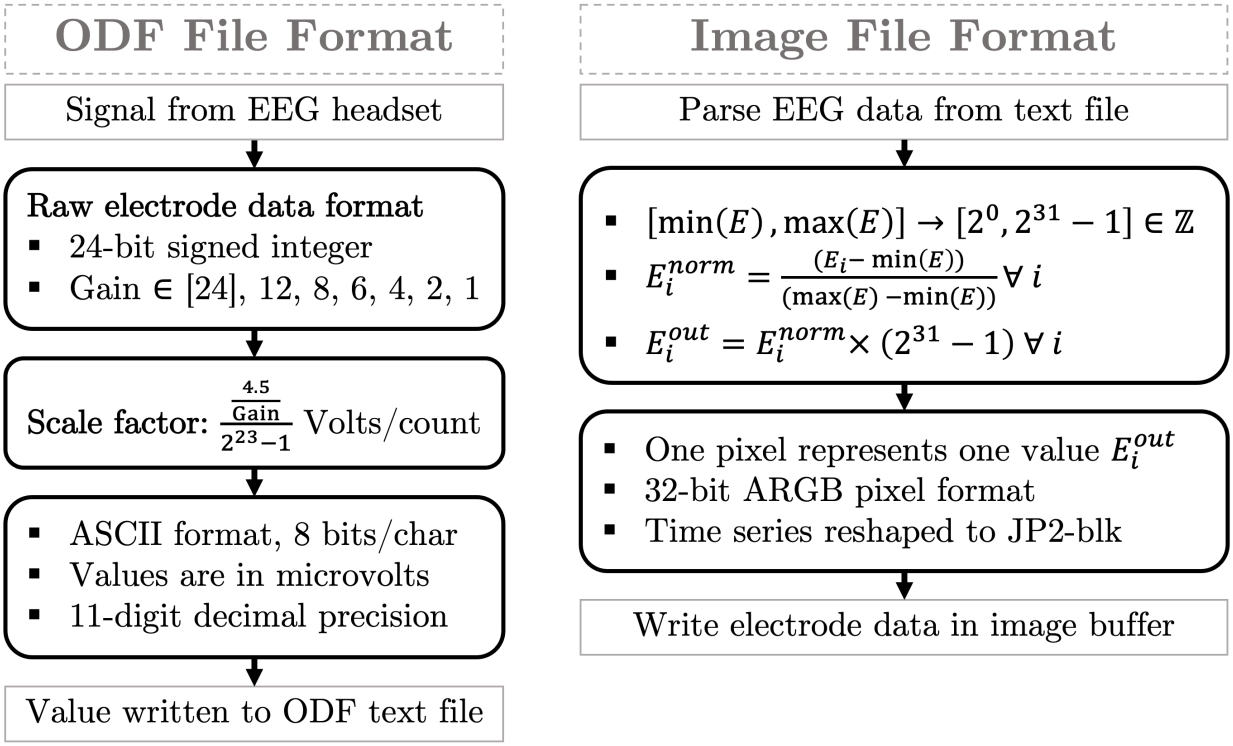

Figure 5: End-to-end file formatting diagram.

3.3 OpenBCI Encoding Process

In the left column of Figure 5, we see a flowchart detailing the process for reading and storing raw EEG data from a headset source in OpenBCI GUI. The EEG headset chip first samples each electrode and renders the sample values as 24-bit signed integers. These integer-valued samples are then converted into scientific units (μV) using a scale factor determined by the chipset specifications, which are provided by the manufacturer. In our case, the Cyton headset uses the ADS1299 chipset, which has the scale factor listed in Figure 5. These microvolt values are finally converted to strings and stored in a UTF-8-encoded, comma-delimited text file, where each line in the text file contains all of the data representing one sample. In Figure 7, we see a depiction of the order and description of values contained in each line. We can see the efficiency in data representation here is incredibly low. For each EEG channel sample, A 24-bit value is subsequently represented by an average of 18 8-bit text characters, totaling 144 bits, or 6 times as many bits relative to the original integer value from the headset. For all other types of data, a similar problem persists. In Figure 7, we note that there are many values labeled as "Other" and "Analog". These are auxiliary channels, which can be used for sampling values from peripheral equipment such as EMG and ECG, which capture muscle response data and electrical signal data from the heart, respectively. If any auxiliary channels are left unused, the recorded text file simply places values of "0.0" in these columns for every sample. This adds an additional 24 bits of overhead to every unused auxiliary channel for every sample.

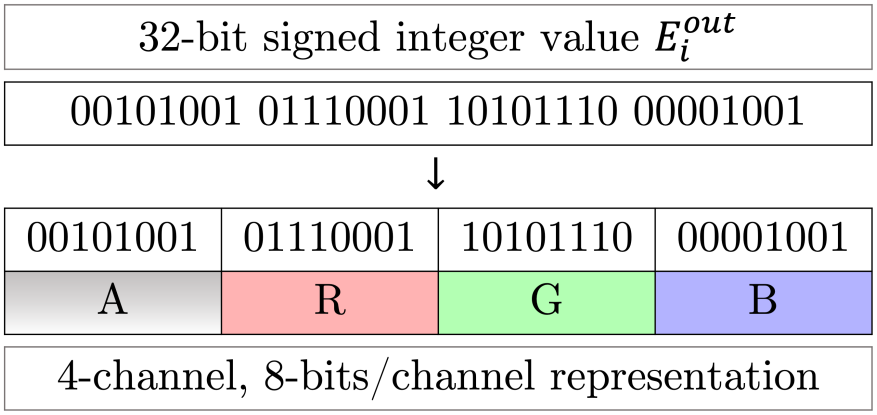

Figure 6: Bit-allocation diagram.

3.4 JP2-EEG Encoding Process

In our scheme, we parse the EEG channel sample values and convert them back into integers. The mathematical details of this process are listed in the right column of Figure 5, wherein we normalize the EEG matrix E and scale the normalized matrix E^{norm} to span the range of positive 32-bit signed integer values, thus creating E^{out}. To normalize, we first identify the minimum and maximum electrode sample values contained in the selected text file. We then operate over every electrode sample value E_i and convert them to a normalized range $0 \leq E_i^{norm} \leq 1$ per the depicted formula. Java uses signed integers as the native primitive integer type, so we use only the dynamic range available for positive integers, which is $[0, 2^{31} - 1] \in \mathbb{Z}$. Therefore, we have only 31 bits of dynamic range per sample to work with, which still exceeds the actual dynamic range of the input data (24-bit). This means we are able to retain full precision in the reconstructed data. It is important to note that there is potentially a more efficient way to do this. Since the raw samples from the EEG headset are 24-bit signed integers, one could theoretically store these integer samples directly in a matrix before the conversion to microvolts occurs. This could save both computational resources and bits. Unfortunately, this would require making some drastic changes to the OpenBCI GUI source code, since scaling is performed very early in the recording process. So, time permitting, this improvement could be worth exploring thoroughly in future work.

3.4.1 Bit Allocation

To our knowledge, we are unable to directly store E^{out} in an image with a 32-bit grayscale pixel format within the JPEG 2000 standards framework. Instead, the 32-bit integer values are separated into four 8-bit chunks and packed into the four respective image channels of an ARGB pixel (per Figure 6). Therefore, one pixel represents one electrode sample value. Once these pixels are processed, we arrange them into the *jp2-blk* format (per Figure 3) and encode the resulting image. An example encoded EEG image can be seen in Figure 8 and is further discussed in the beginning of section 3.6.

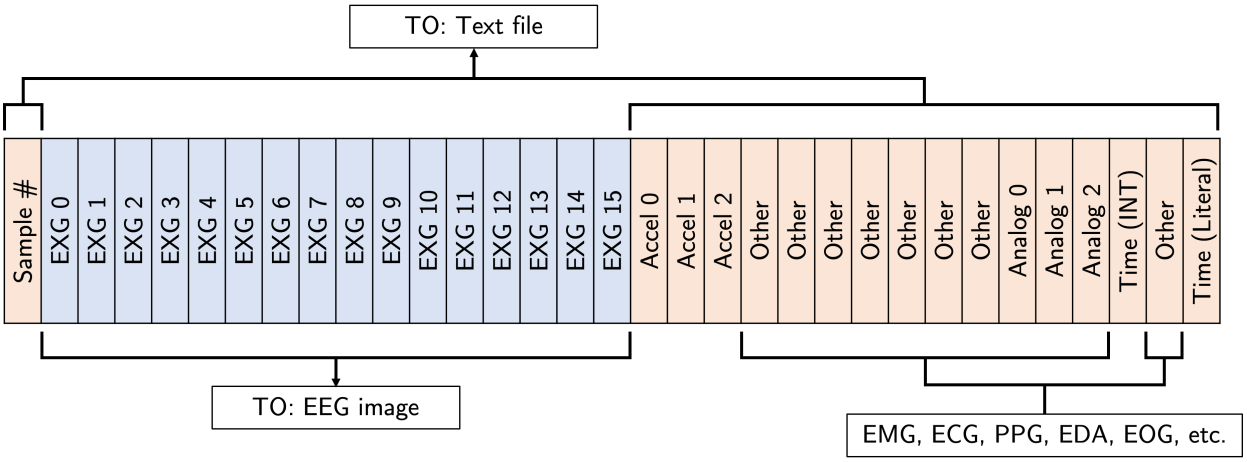

Figure 7: Format of a single line in a raw OpenBCI text file.

3.5 Text File Formatting

The remaining data in the text file, sans the EEG data we have encoded in the image, is copied to a new text file with a "_JP2" suffix concatenated to the file name, so we may differentiate the two. The items contained in this modified text file are pictured as orange blocks in Figure 7. We add header data to the new text file containing the minimum and maximum electrode sample values so we may later reconstruct the original EEG data when selecting the EEG image file using the "Select JP2 File" button. The EEG image and the modified text file are grouped in a corresponding folder, and the modified text file may be compressed as a ZIP archive, depending on the intended application. The compression results for both cases are discussed in section 3.6. The benefits and drawbacks of additional ZIP compression are detailed in section 3.6.2.

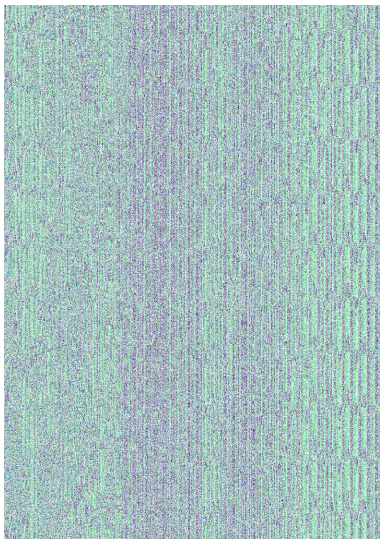

Figure 8: EEG image sample.

3.6 Testing and Analysis

Once the compression algorithm was programmed, we proceeded to test the widget. We arbitrarily selected a few previously-collected EEG data files of varying sizes and fed them to the encoder using the "Convert TXT File" button. Figure 8 depicts one such image yielded by the encoder. We note that the yielded image does not present useful visual information to the human eye. This is because the bits used to represent each EEG channel sample are arbitrarily separated into color channels to enable compatibility with the codec we are employing, which nullifies the previously valid linear relationship between the recorded integer value and the sampled voltage. We next present the compression results and discussion for this experiment.

#	**Board**	**SRate** (Hz)	**RAW** (MB)	**JP2** (MB)	**CR** (X:1)	**ZIP** (MB)	**CR** (X:1)	**JP2+ ZIP** (MB)	**CR** (X:1)
1	Synth	250	4.584	2.718	1.686	2.068	2.216	1.392	3.291
2	Synth	250	17.816	9.123	1.953	8.032	2.218	5.389	3.306
3	Cyton	1000	20.856	10.547	1.977	7.614	2.739	3.241	6.434
4	Cyton	1000	102.643	40.207	2.553	25.999	3.948	11.759	8.729
5	Cyton	1000	114.930	49.385	2.327	36.204	3.174	15.570	7.381
6	Cyton	1000	144.392	57.192	2.525	34.795	4.150	16.301	8.858

Table 1: Compression results for select OpenBCI ODF files.

3.6.1 General Applications

In the table above, we see six items representing EEG data files of varying sizes. The left column contains the item index, board identifier, and sampling rate used by the board. The *Synth* board is simply an algorithmic data stream from a synthetic source. This source produces a random signal that is difficult to compress effectively, but it proves insightful for testing the "worst case scenario" from a data efficiency standpoint. The 16-channel *Cyton* board is the one we've been exemplifying in discussion thus far, as it is the current flagship EEG headset compatible with OpenBCI GUI and will consistently create the largest files due to its high electrode channel count and high sample rate. The second column lists the uncompressed file size for each item, and the third column lists the respective compressed file group size and corresponding compression ratio. The file group consists of the EEG image file (e.g. pictured in Figure 8), and the modified text file which contains all other ancillary data as outlined in section 3.5. We are not so much concerned with the specifications of the hardware used in each case, but rather how the compression algorithm performs in a variety of cases. We can see that, for increasing file size, the compression ratio generally increases to a point and plateaus around 2.5:1, which is quite similar to the results yielded in the Matlab-based experiment in which we compressed EEG data from the BCICIV dataset [9]. Compressing the files in this fashion is ideal for use cases where users need to access file data quickly and frequently, as the electrode trace data can be parsed from the image and coupled with the corresponding text file data with little computational expense.

3.6.2 Archival Applications

For long-term storage, a user may choose to trade higher computational expense and increased encoding time for further reduced file size via the ZIP compression method. ZIP is a file archiving tool that employs a variety of lossless compression algorithms like DEFLATE, which itself uses a combination of Huffman coding and LZ77 to reduce data redundancy [5]. Upon *zipping* a file or group of files, a new file with a *.zip* extension is yielded, containing all of the specified data within. To access this data after archiving, a user must *unzip* the file, which takes a non-trivial amount of time depending on how large and varied the contents are. ZIP works quite well with text file data, so we choose to implement it here.

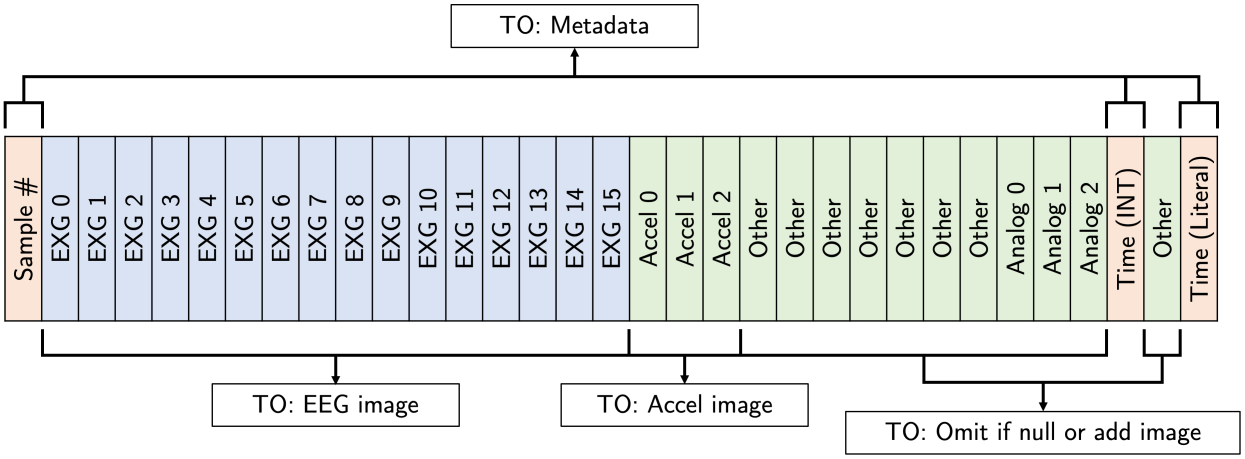

Figure 9: Potential compression algorithm optimization.

The fourth column in Table 1 lists the respective file sizes and corresponding compression ratios yielded by the ZIP method, which involves simply zipping the raw text file. The final column lists the same metrics yielded by the JP2+ZIP method, which involves zipping the modified text file and grouping this file with the EEG image file in a folder. Comparing the two methods, we can see that the JP2+ZIP method is notably superior across the board, yielding over double the compression ratio in four cases here relative to the ZIP method.

The main caveat in employing ZIP compression is the overhead encoding and decoding time added to the workflow. Considering the large increase in compression, the JP2+ZIP method may be worth employing in cases where a user intends to put files in long-term storage, upload files to a cloud storage, or transmit files over low-bandwidth networks. If we take steps to further optimize the compression algorithm, we have the potential to yield compression performance similar to that realized with the JP2+ZIP method while also negating the decoding overhead that comes with utilizing ZIP.

3.6.3 Potential Optimizations

In section 3.3 we state that the efficiency in data representation for OpenBCI text files is generally very low. This is not only true for the electrode channel data but for the rest of the data contained in the file (per Figure 7). Encoding large numerical values as strings of UTF-8 text characters will drastically increase the amount of bits required to store that value. One could extend the JP2-EEG encoding process to produce images for each category of data contained in the text file that accommodates for the specific data characteristics per the depiction in Figure 9. For example, the accelerometer on the EEG headset has 3 channels, each sampled with 16-bit resolution. This accelerometer data can be stored in a JPEG-2000 encoded image using a 16-bit grayscale pixel format with the *jp2-blk* matrix reshaping scheme applied. If any of the auxiliary channels are left unused, one could delete the columns in the text file for these channels and program a flag to recognize the absence instead of filling the column with zeros. As mentioned in section 3.4, the current implementation of the JP2-EEG encoder uses a 32-bit ARGB pixel format to represent electrode samples that are captured with 24-bit resolution. This inefficiency is certainly resolvable, but it is out of scope for the current implementation due to time constraints.

4 CONCLUSIONS

In this project we implemented an image compression scheme to compress EEG signal data in OpenBCI. Our ultimate goal with this project was twofold. First, we wanted to meaningfully reduce the size of EEG data files. Second, we wanted to reduce the computational expense and transmission bandwidth requirements when analyzing these files as a means for creating a more efficient user experience. Loading and analyzing a relatively small EEG data file in OpenBCI GUI on modern hardware is a trivial task in isolation. However, once these files reach the gigabyte range, it is taxing for the computer to parse them, difficult to store large collections of them, and frustrating for the user waiting on the computer to download the data and render it. We were successful in fully realizing the first goal, as shown in section 3.6. Unfortunately, as discussed in section 3.1.1, we were unable to completely develop the EEG time series browser interface within the JP2-EEG widget due to time constraints and programming complexity, thus not fully achieving the second goal. In the current version of the JP2-EEG widget, the backend operates as intended, and the widget interface contains the following graphical elements: text fields for the in- and out-times, render button, scroll bar, scroll bar reference time, and encode/decode buttons. The widget interface is lacking the time series plot and the peripheral elements for modifying plot parameters. The viability of methods and noteworthy results discussed in our experiments motivate the push for universal data compatibility in EEG and, more broadly, other types of medical data. A considerable benefit of open-source development is that this implementation can be improved over time by other developers willing to and interested in picking up the code and contributing to it. The code for this widget is available on our Github page [8].

REFERENCES

[1] George K Anastassopoulos and A Skodras. Jpeg2000 roi coding in medical imaging applications. In *Proc. 2nd IASTED Int. Conf. on Visualisation, Imaging and Image Processing (VIIP2002)*, pages 783–788. Citeseer, 2002.

[2] Giuliano Antoniol and Paolo Tonella. Eeg data compression techniques. *IEEE Transactions on Biomedical engineering*, 44(2):105–114, 1997.

[3] Alexander J Casson and Esther Rodriguez-Villegas. Toward online data reduction for portable electroencephalography systems in epilepsy. *IEEE transactions on biomedical engineering*, 56(12):2816–2825, 2009.

[4] Justin Dauwels, K Srinivasan, M Ramasubba Reddy, and Andrzej Cichocki. Near-lossless multi-channel eeg compression based on matrix and tensor decomposition. *IEEE journal of biomedical and health informatics*, 17(3):708–714, 2012.

[5] Peter Deutsch. Deflate compressed data format specification version 1.3. Technical report, Network Working Group, 1996.

[6] Behzad Hejrati, Abdolhossein Fathi, and Fardin Abdali-Mohammadi. Efficient lossless multi-channel eeg compression based on channel clustering. *Biomedical Signal Processing and Control*, 31:295–300, 2017.

[7] Garry Higgins, Stephen Faul, Robert P McEvoy, Brian McGinley, Martin Glavin, William P Marnane, and Edward Jones. Eeg compression using jpeg-2000: How much loss is too much? In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 614–617. IEEE, 2010.

[8] Joshua Tellez. JP2-EEG. `https://github.com/joshtell/jp2-eeg`.

[9] Joshua Tellez. Evaluation of jpeg-2000 image compression as applied to electroencephalography. In *International Foundation for Telemetering Proceedings, 57*. International Foundation for Telemetering, 2022.

[10] A Temko, R McEvoy, D Dwyer, S Faul, G Lightbody, and W Marnane. React: Real-time eeg analysis for event detection. In *Proceedings of the AMA-IEEE Medical Technology Conference on Individualized Healthcare, Washington, DC, USA*, volume 2123, 2010.

[11] Elizabeth Waterhouse. New horizons in ambulatory electroencephalography. *IEEE Engineering in Medicine and Biology Magazine*, 22(3):74–80, 2003.

[12] Yaniv Zigel, Arnon Cohen, and Amos Katz. The weighted diagnostic distortion (wdd) measure for ecg signal compression. *IEEE transactions on biomedical engineering*, 47(11):1422–1430, 2000.