Detecting Eye-Movement and Muscle Artifacts in EEG Data using Deep Neural Networks

Leon Ackermann
University of Osnabrück
lackermann@uni-osnabrueck.de

Aaron Maiwald
University of Osnabrück
amaiwald@uni-osnabrueck.de

Abstract

Objective:* EEG artifacts such as eye or muscular movements are unwanted signals which mislead EEG classification and are difficult and time-consuming to remove by hand. We, therefore, want to improve existing methods for automating artifact detection using deep learning. *Methods:* We present two models to detect EEG artifacts from eye and muscular movements: a CNN with attention and a bidirectional, convolutional LSTM. We train and evaluate our models on the TUH Artifact Corpus. *Results:* for muscle artifacts, the CNN achieved better accuracy with 89 percent while LSTM had an accuracy of only 72 percent. For eye-movement artifacts, the LSTM achieved 92 percent and the CNN 79 percent accuracy. Both models had poor precision and recall and specialized in different artifacts. For the simpler task of detecting the presence of artifacts within a segment, both models had high accuracy, precision, and recall, overall achieving performance competitive with state-of-the-art models.

1. Introduction

EEG (electroencephalography) is a non-invasive method for recording the electrical activity of the brain. Researchers use EEG for its high temporal resolution, capturing brain activity changes in milliseconds and because it is a relatively affordable, portable, and widely accessible neuroimaging technique (for an overview see Michel and Brunet, 2019). A major challenge in analysing EEG data is the presence of artifacts. Artifacts in EEG recordings are unwanted signals or interferences unrelated to brain activity (Jiang et al., 2019). Common sources include eye-movements, muscle activity, and heartbeat. Environmental noise and electrode issues can also introduce artifacts. (Hamid et al., 2020 Identifying and removing artifacts is crucial for accurate EEG data analysis; it is also time-consuming to do manually. It is therefore desirable for the relevant research community to develop reliable and fast methods for detecting (and ideally removing) such artifacts.

There exists substantial research on EEG artifact detection and removal using techniques, such as Principal Component Analysis (Berg and Scherg, 1991) and Independent Component Analysis (Jung et al., 1998) or Filtering techniques such as Adaptive Filtering (He et al., 2004) and Wiener Filtering (Somers et al., 2018).

Only very recently have researchers started to apply deep learning to this task. Deep learning is a subfield of machine learning that uses neural networks with multiple layers to automatically learn hierarchical representations of the data. Recent literature concentrated on deploying Convolutional Neural Networks (Nejedly et al., 2019) or Attention-based networks (Cisotto et al., 2020; Peh et al., 2022) or LSTM-based networks (Hasib-Al-Rashid et al., 2020) for identifying artifacts.

The majority of the research in this field has not been validated on large datasets comprising more than 100 patients. Instead, small datasets consisting of less than 100 patients or semi-simulated datasets by injecting noise into regular EEGs are often used (Abdi-Sargezeh et al., 2021,Mashhadi et al., 2020). Evaluation metrics to measure the effectiveness of artifact detection methods have been lacking in most studies (Thomas et al., 2021). As a result, comparing existing artifact detectors has been challenging.

In this paper, we present a deep learning classifier for EEG artifact detection and compare its performance to previous work. The task of our classifier is to recognise segments containing artifacts specifically from muscle or eye-movements. To achieve this, we compare various deep learning architectures such as long short-term memory (LSTM) networks and attention-based models. These architectures have shown promising results in other domains and have the potential to improve the accuracy of artifact detection in EEG data. This paper makes two important contributions:

1. While previous models have only classified segments of EEG data as containing artifacts or not, our model also

predicts precisely when an artifact occurs within a segment. Insofar as EEG artifact detection is for the purpose of cleaning EEG data, knowing precisely when artifacts occur is very advantageous: researchers can remove EEG data segments in a much more targeted way, meaning that much less data is lost. We demonstrate that by using relatively simple models, it is possible to achieve accuracy and recall that is competitive with state-of-the-art methods.
2. We investigate the performance of a bidirectional, convolutional LSTM on this task. The key motivation for this architecture is the success of Phan et al. **w**ho used deep bidirectional Recurrent Neural Networks (RNNs) for automatic sleep stage classification from EEG data. We hoped that such a model architecture might be suitable for EEG artifact detection, since this task shows some structural similarity to EEG sleep stage classification.

In the second section of this paper, we will introduce the dataset we use, explain our preprocessing steps and illustrate the architectures of our models: a bidirectional, convolutional LSTMs and an attention-enhanced CNN. Section 3 will show our most important results and an ablation study. Section 4 will discuss the performance of our models in comparison to similar models in the recent literature. Section 5 concludes this work and lists promising directions for future research.

2. Methods

2.1. Dataset

We train and evaluate our classifier on the TUH EEG Artifact dataset from the Temple University Hospital of Philadelphia (Pennsylvania) which is a subset of the TUH EEG Data Corpus (Hamid et al., 2020). This subset dataset consists of normal EEG signals and EEG signals affected by five types of artifacts: chewing events, eye-movements, muscular artifacts, shivering events, and instrumental artifacts (such as electrode pop, electrostatic artifacts, or lead artifacts). In total, the dataset comprises 259 recorded EEG sessions that were recorded over a span of 13 years. TUH recorded 213 patients between 10 and 90 years old. The EEGs were recorded with a sampling frequency of 250 Hz and 16-bit resolution. For the purpose of this project, we limit our analysis to the most common types of artifacts: muscular artifacts and eye-movement artifacts. Muscle artifacts appear as sharp waves in the dataset such as in the first figure in Figure 1 while eye-movement artifacts appear as a spike in the EEG signal such as the second figure in Figure 1.

In total the dataset comprises around 16 hours of muscle artifacts and 11 hours of eye-movement artifacts in our data, see Table 1.

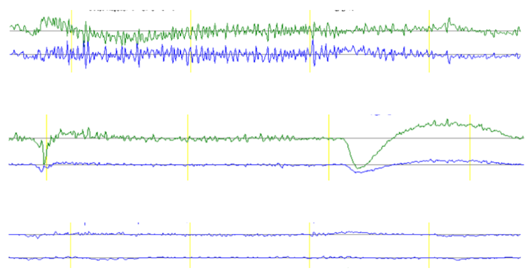
Figure 1. Example of EEG recording with muscle artifact (first), eye-movement artifact (second) and without any artifact (third) (Hamid et al., 2020). For the purpose of illustration, we selected only two channels.

Artifact	Number Events	Duration (h)
Muscle	7656268	16.62
Eye Movement	5211538	11.30

Table 1. Number of artifacts in each category and corresponding length of EEG recordings.

2.2. Data Acquisition and Preprocessing

We wrote a custom Python script to download the EDF files and convert them to PKL files. The conversion process enables us to extract the EEG data from the EDF files in a structured and standardized format for further analysis.

The EDF files contain time-stamps for when the artifacts start and when they end. We decided to create labels for each time-step, indicating with 0s and 1s whether an artifact is present or not.

We created windows of EEG data to segment the continuous time-series into smaller and more manageable parts. We chose a window length of 5 seconds based on previous research by Peh et al., 2022, and since not all recordings had the same frequency, we downsampled them to a frequency of 128Hz. Following previous research, we decided to use all and only the channels that were present in the entire dataset, which amounted to 19 channels. In line with common deep learning practice, we normalized the data with a z-transformation.

During the analysis of our dataset, we observed that only 9 percent of time-steps had a positive label. As a result, our models tended to learn to predict only negative labels. To address this issue, we removed all samples (each containing a 5s recording) where no artifacts were present following the cluster-based undersampling approach presented in Yen and Lee, 2009. The undersampling method reduced our sample size from approximately 50,000 to roughly 11,000 samples in the training set, while increasing the balance of positive (artifacts) to negative (non-artifacts) samples from 9 percent to around 35 percent. We found that this had a positive effect on the performance of our model. We ensured that evaluating our model on the test data and preventing overfitting with the validation set was done on the original, imbalanced data, see Table 2.

Dataset	Balance
Training set	35,37%
Validation set	10.14%
Test set	8.37%

Table 2. Balance of each dataset, training set was undersampled while validation and test set are kept at original balance

Hyperparameters	**Grid Search Range**	**LSTM**
Bidirectional	True, False	True
LSTM Layers	1,2,4 to 8	1
LSTM Hidden Units	32, 64, 128, 256	128
Convolutional Layers	1,2,3,4	2
Dense Layers	0,1,2,4	1
Dropout rate	0.3, 0.4, 0.5	0.5
Learning Rate	0.0001, 0.001, 0.01	0.001
Hyperparameters	**Grid Search Range**	**CNN**
Convolutional Layers	1,2,3,4,6	6
Attention	True, False	True
Attention Heads	4,8	8
Dense Layers	1,2,3	1
Optimizer	SGD, Adam, Adagrad	Adam
Learning Rate	0.0001, 0.001, 0.01	0.001

Table 3. Results of hyperparameter optimization

2.3. Models

After we decided on using CNNs and LSTMs, we performed a grid search on the hyperparameters in Table 3. The first round focused on architectural parameters: for the CNN this was the number of hidden layers, the number of attention heads, whether to use attention at all and the number of convolutional layers. For the LSTM this included the number of LSTM layers, the number of LSTM hidden units in each layer, the number of hidden dense layers, and whether to use bidirectionality or not. Each configuration was trained for 50 epochs. We then selected the five models with the lowest validation loss and performed a second round of grid-search on non-architectural hyperparameters: optimizer, learning rate and dropout rate. To train our models we used cloud GPUs, specifically NVIDIA RTX 4090 and NVIDIA A100 SXM4 X1. In total, we trained for approximately 80 hours. As a loss function, we used binary cross-entropy for parameter optimisation for all models.

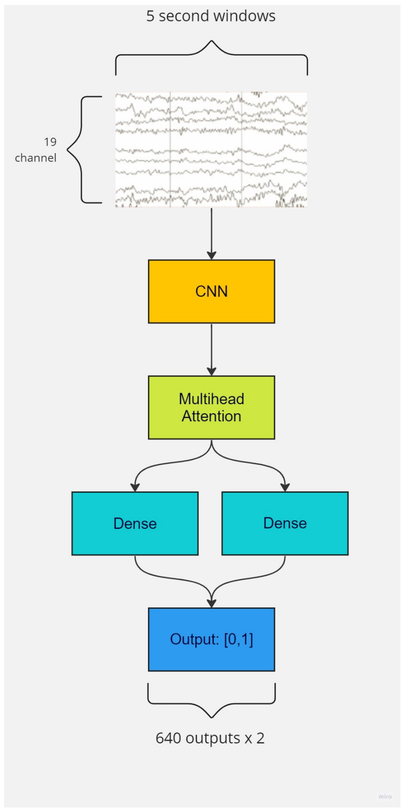

Figure 2. Convolutional Attention-based network

2.4. CNN with Attention

Our first model is a CNN with self-attention, see Figure 2. The model receives an input matrix consisting of 19 channels of 5-second windows at a frequency of 128hz, resulting in a 640 x 19-dimensional input matrix. The input matrix is processed by six consecutive 1-D convolutional layers, each with three filters, with the filter size increasing by 8 in each layer. After each convolutional layer, we apply Max Pooling with a pool size of two. We then apply a multi-head attention layer with 8 attention heads, apply dropout with probability 0.3 and connect this to a sigmoid layer with 640 units. This layer predicts, for each time step, the probability of a muscle artifact occurring in each one of 19 channels. The output thus has the same shape as the input. We aggregate these probabilities into one for each time-step by taking the arithmetic mean after experimenting with taking the maximum or product and finding that this maximized model performance.

2.5. Bidirectional, convolutional LSTM

There have been several approaches to classify EEG data with LSTMs. We took inspiration from Nagabushanam et al., 2020 Hasib-Al-Rashid et al., 2020, Singh and Malhotra, 2022, Ni et al., 2017, Tuncer and Doğru Bolat, 2022 to design a convolutional, bidirectional LSTM (see Figure 3). Our LSTM model receives an input batch with samples consisting of 640 timesteps each containing 19 EEG channels. We then extend the last axis of the training set to match the dimensions required by the Convolutional Block unfolded over time. The convolutional block consists of two 1-d convolutional layers with 8 and 16 filters respectively each having a kernel size of 3*3 followed by a 1-d Max Pool Layer and Flattening Layer. The output of the convolutional block gets processed by a one-layer Bidirectional LSTM with 128 Hidden Units. The Bidirectional LSTM is unfolded over time into 640 timesteps. At each timestep, the output of the bidirectional LSTM is passed to the output dense layer with two nodes.

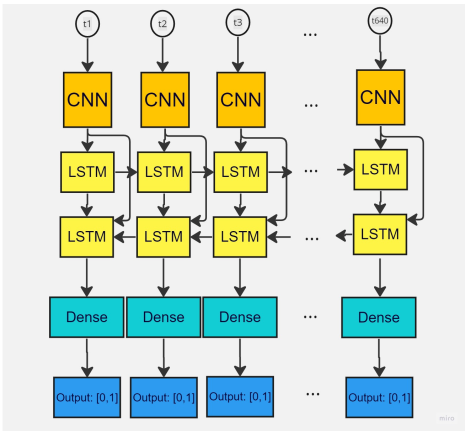

Figure 3. Architecture of our bidirectional, convolutional LSTM. Inputs are time-slices of the EEG windows. We slightly simplified the architecture for the purpose of illustration.

3. Results

Our study compared the performance of a CNN and an LSTM network on the task of detecting muscle and eye-movement artifacts in EEG data. We summarise the performance of our best models in each category in 4

First the task of *localizing* artifacts within segments. For detecting artifacts from muscle activity, the LSTM had an accuracy of 72 percent, while the CNN achieved a much better 89 percent accuracy. We found the reverse for detecting eye-movement artifacts: while the LSTM had an impressive 92 percent accuracy, the CNN only classified 79 percent of time-steps correctly. We note that the test data was heavily imbalanced towards negative labels, and thus a high accuracy on its own is not strong evidence that the model learned its task. Both models had poor precision and again, the two models seemed to specialize in different artifacts: while the CNN achieved 60 percent precision for muscle artifacts, the LSTM only acieved 20 percent precision, but 45 percent precision for eye-movements, while the CNN only achieved 14 percent.

Unsurprisingly, our models performed generally much better at the simpler task of detecting only whether (not where) artifacts occur within a segment. For detecting eye-movement artifacts, the CNN classified 80 percent of segments correctly. For muscle artifacts, the CNN achieved 79 percent accuracy. In this case, both models had high precision and recall: the CNN with 80 percent for eye-movements and 90 per cent for muscle artifacts[add other...].

3.1. Ablation

In the following, we will analyse the contribution that the various components of our models make towards their overall performance.

3.1.1 CNN with Attention

First, for the CNN we compared the (i) full model to an otherwise identical (ii) model without attention and (iii) without convolutional layers (i.e. just a multi-head attention layer with a dense layer). We found that the effect of these two components on the models' performance varied between the two artifacts (muscle and eye) we tried to detect.

In the case of muscle artifacts, both ablated models had significantly *higher* overall validation loss after 100 epochs, i.e. performed markedly *worse*. Adding convolutional layers reduced validation loss by 0.2, while adding a multi-head attention layer reduced validation loss only by 0.04.

In the case of eye-movement artifacts, we found the reverse: the full model had significantly *higher* validation loss than both ablated models. Taking away convolutional layers reduced validation loss roughly twice as much as using a model without attention.

It thus seems like the convolutional layers *improved* the models' ability to detect *muscle* artifacts much more than the attention layer. On the other hand, the convolutional layers *decreased* the models' ability to detect *eye* artifacts much more than the attention layer.

3.1.2 LSTM

The best-performing LSTM model uses the hyperparameters depicted in Table 3. We compare that best (iv) LSTM model which is a Convolutional Bidirectional-LSTM to (v) an identical Bidirectional LSTM model without Convolutions and (vi) to an identical Convolutional Multilayer LSTM model where instead of a Bidirectional LSTM, we take two LSTM layers. Table 4 depicts the performance of the different models for predicting eye-movement and muscle artifacts.

In the case of the muscle artifacts, the accuracy for model (vi) decreases while precision and recall stay similarly low.

Architectures	Muscle Artifacts			Eye Movement Artifacts		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
(i) Full: CNN with Attention	0.79	0.57	0.55	0.9	0.15	0.3
(ii) CNN without Attention	0.81	0.43	0.63	0.86	0.14	0.22
(iii) Attention with Dense	0.91	0.31	0.04	0.87	0	0
(iv) Full: Bidirectional LSTM with CNN	0.72	0.2	0.6	0.92	0.45	0.54
(v) Multi Layer LSTM with CNN	0.77	0.22	0.49	0.91	0.37	0.45
(vi) Bidirectional LSTM without CNN	0.66	0.17	0.66	0.93	0.5	0.53

Table 4. Performance of our best models in bold ('Full') and results of our ablation studies, showing, for each model, the effect of two model components on accuracy, precision and recall. On the left for muscle artifacts, on the right for eye movement artifacts.

Model (v) performs slightly better in terms of accuracy while precision stays the same but recall is reduced by almost 0.1.

When comparing the performance of the models for eye-movement artifacts we observe the same patterns. Model (iv) performs the worst out of all three since precision and recall decrease noticeably. Model (v) performs slightly better than Model (vi) with higher precision.

These results were produced on the test set. We found that due to the size and balance of our test set, we often scored slightly different results of the above-described models. However, one observation continued to be in the foreground. Model (v) performs worse than (iv) and (vi) on any random test set.

We, therefore, assume that, from the components we ablated, the bidirectional processing had the largest positive impact on the performance of our model. This finding is further strengthened by the results of our hyperparameter optimization where we found that bidirectional LSTMs performed 10% better on average than their identical Multi-Layer LSTMs. We further assume that the data fits a bidirectional processing structure as patterns of the EEG signal before an artifact appearance might play a similarly important role as the EEG signal patterns after the artifact. Moreover, our results show that the LSTM based models show stronger performance in predicting eye movement artifacts.

4. Discussion

The following will compare our results to state-of-the-art models. Several studies have reported results on artifact detection and rejection in EEG data. Y. Roy et al., 2019 performed multi-class artifact classification and achieved a sensitivity of 72.39 percent for the background class. S. Roy, 2019 and Abdi-Sargezeh et al., 2021 used wavelet transform for artifact detection and measured the effectiveness of their artifact rejection module indirectly by performing brain-computer interface classification (Abdi-Sargezeh et al., 2021). They reported an accuracy improvement from 63 percent to 72.5 percent. Mashhadi et al., 2020 employed a U-NET model to reject ocular artifacts and reported a mean square error of 0.00712. Dhindsa, 2017 achieved an accuracy of 93.3 percent and an AUC of 0.923 in artifact classification on an EEG dataset with four channels. Finally, Pion-Tonachini et al. (2019) used independent component analysis and CNN/GAN to classify EEG independent components and achieved a balanced accuracy of 0.855, 0.623, and 0.597 for 2, 5, and 7-class classification, respectively. They reported a sensitivity of 73 percent for the background class in all scenarios (Pion-Tonachini et al., 2019).

In this study, we evaluated the performance of our models on two related but distinct tasks: (i) detecting the presence of artifacts in a window of EEG data and (ii) localizing where within that window the artifact occurs. For task (i), our models achieved competitive results compared to state-of-the-art models in terms of accuracy, precision, and recall. Given the greater difficulty of (ii) compared (i) it is unsurprising that our models performed markedly worse on this task. It is important to note that for task (ii), our models performed considerably worse than for task (i), and we were unable to make direct comparisons with previous literature, as existing studies focused primarily on task (i).

Our study has two main limitations. First, our models classify multi-channel segments, which may limit their adaptability to EEG data with a different number of channels compared to previous studies that focused on single-channel models. Second, we only focused on the two most common types of artifacts, namely those from eye-movements and muscle activity, neglecting other potential artifacts, such as those from shivering or sweating. Nonetheless, given that the artifacts we focused on constitute the majority of artifacts in typical EEG recordings, we believe that our system could be highly beneficial to neuroscientists.

5. Conclusion

In this paper, we presented two multi-channel neural systems for the automated detection of EEG artifacts caused by eye-movements and muscle activity. The first system is CNN with attention and the second system is a bidirectional, convolutional LSTM, both optimized using binary cross-entropy loss. We trained the models to detect the presence of artifacts and localize them within the EEG data. Our proposed system has shown to be effective in rejecting artifacts while keeping the clean EEG data intact, which could potentially enhance the interpretability of EEG recordings.

Future research could address these limitations by creating models that more accurately localize precisely where artifacts occur in EEG data. As a result, researchers using such software could remove artifacts in a more targeted way. Furthermore, it would be advantageous if such models were trained to detect artifacts in a way that is not tailored to a certain number of channels, e.g. by instead processing single channels.

6. Data and code availability

For the purpose of transparency, we have shared our data and code in this GitHub repository, alongside instructions for how to reproduce our findings.

References

Abdi-Sargezeh, B., Foodeh, R., Shalchyan, V., & Daliri, M. R. (2021). EEG artifact rejection by extracting spatial and spatio-spectral common components. *Journal of Neuroscience Methods*, *358*, 109182. https://doi.org/10.1016/j.jneumeth.2021.109182

Berg, P., & Scherg, M. (1991). Dipole modelling of eye activity and its application to the removal of eye artefacts from the EEG and MEG. *Clinical Physics and Physiological Measurement*, *12*(A), 49–54. https://doi.org/10.1088/0143-0815/12/A/010

Cisotto, G., Zanga, A., Chlebus, J., Zoppis, I., Manzoni, S., & Markowska-Kaczmar, U. (2020). Comparison of Attention-based Deep Learning Models for EEG Classification [arXiv:2012.01074 [cs, eess, q-bio]]. Retrieved March 1, 2023, from http://arxiv.org/abs/2012.01074

Dhindsa, K. (2017). Filter-Bank Artifact Rejection: High performance real-time single-channel artifact detection for EEG. *Biomedical Signal Processing and Control*, *38*, 224–235. https://doi.org/10.1016/j.bspc.2017.06.012

Hamid, A., Gagliano, K., Rahman, S., Tulin, N., Tchiong, V., Obeid, I., & Picone, J. (2020). The Temple University Artifact Corpus: An Annotated Corpus of EEG Artifacts [ISSN: 2473-716X]. *2020 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 1–4. https://doi.org/10.1109/SPMB50085.2020.9353647

Hasib-Al-Rashid, Manjunath, N. K., Paneliya, H., Hosseini, M., Hairston, W. D., & Mohsenin, T. (2020). A Low-Power LSTM Processor for Multi-Channel Brain EEG Artifact Detection [ISSN: 1948-3287]. *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 105–110. https://doi.org/10.1109/ISQED48828.2020.9137056

He, P., Wilson, G., & Russell, C. (2004). Removal of ocular artifacts from electro-encephalogram by adaptive filtering. *Medical & Biological Engineering & Computing*, *42*(3), 407–412. https://doi.org/10.1007/BF02344717

Jiang, X., Bian, G.-B., & Tian, Z. (2019). Removal of Artifacts from EEG Signals: A Review. *Sensors (Basel, Switzerland)*, *19*(5), 987. https://doi.org/10.3390/s19050987

Jung, T.-P., Makeig, S., Bell, A. J., & Sejnowski, T. J. (1998). Independent Component Analysis of Electroencephalographic and Event-Related Potential Data. In P. W. F. Poon & J. F. Brugge (Eds.), *Central Auditory Processing and Neural Modeling* (pp. 189–197). Springer US. https://doi.org/10.1007/978-1-4615-5351-9_17

Mashhadi, N., Khuzani, A. Z., Heidari, M., & Khaledyan, D. (2020). Deep learning denoising for EOG artifacts removal from EEG signals. *2020 IEEE Global Humanitarian Technology Conference (GHTC)*, 1–6. https://doi.org/10.1109/GHTC46280.2020.9342884

Michel, C. M., & Brunet, D. (2019). EEG Source Imaging: A Practical Review of the Analysis Steps. *Frontiers in Neurology*, *10*. Retrieved April 1, 2023, from https://www.frontiersin.org/articles/10.3389/fneur.2019.00325

Nagabushanam, P., Thomas George, S., & Radha, S. (2020). EEG signal classification using LSTM and improved neural network algorithms. *Soft Computing*, *24*(13), 9981–10003. https://doi.org/10.1007/s00500-019-04515-0

Nejedly, P., Cimbalnik, J., Klimes, P., Plesinger, F., Halamek, J., Kremen, V., Viscor, I., Brinkmann, B. H., Pail, M., Brazdil, M., Worrell, G., & Jurak, P. (2019). Intracerebral EEG Artifact Identification Using Convolutional Neural Networks. *Neuroinformatics*, *17*(2), 225–234. https://doi.org/10.1007/s12021-018-9397-6

Ni, Z., Yuksel, A. C., Ni, X., Mandel, M. I., & Xie, L. (2017). Confused or not Confused? Disentangling Brain Activity from EEG Data Using Bidirectional LSTM Recurrent Neural Networks. *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health In-*

formatics, 241–246. https://doi.org/10.1145/3107411.3107513

Obeid, I., & Picone, J. (2016). The Temple University Hospital EEG Data Corpus. *Frontiers in Neuroscience*, *10*. https://doi.org/10.3389/fnins.2016.00196

Peh, W. Y., Yao, Y., & Dauwels, J. (2022). Transformer Convolutional Neural Networks for Automated Artifact Detection in Scalp EEG [arXiv:2208.02405 [eess]]. Retrieved March 1, 2023, from http://arxiv.org/abs/2208.02405

Pion-Tonachini, L., Kreutz-Delgado, K., & Makeig, S. (2019). ICLabel: An automated electroencephalographic independent component classifier, dataset, and website. *NeuroImage*, *198*, 181–197. https://doi.org/10.1016/j.neuroimage.2019.05.026

Roy, S. (2019). Machine Learning for removing EEG artifacts: Setting the benchmark [arXiv:1903.07825 [eess, stat]]. https://doi.org/10.48550/arXiv.1903.07825

Roy, Y., Banville, H., Albuquerque, I., Gramfort, A., Falk, T. H., & Faubert, J. (2019). Deep learning-based electroencephalography analysis: A systematic review [Publisher: IOP Publishing]. *Journal of Neural Engineering*, *16*(5), 051001. https://doi.org/10.1088/1741-2552/ab260c

Singh, K., & Malhotra, J. (2022). Two-layer LSTM network-based prediction of epileptic seizures using EEG spectral features. *Complex & Intelligent Systems*, *8*(3), 2405–2418. https://doi.org/10.1007/s40747-021-00627-z

Somers, B., Francart, T., & Bertrand, A. (2018). A generic EEG artifact removal algorithm based on the multi-channel Wiener filter. *Journal of Neural Engineering*, *15*(3), 036007. https://doi.org/10.1088/1741-2552/aaac92

Thomas, J., Thangavel, P., Peh, W. Y., Jing, J., Yuvaraj, R., Cash, S. S., Chaudhari, R., Karia, S., Rathakrishnan, R., Saini, V., Shah, N., Srivastava, R., Tan, Y.-L., Westover, B., & Dauwels, J. (2021). Automated Adult Epilepsy Diagnostic Tool Based on Interictal Scalp Electroencephalogram Characteristics: A Six-Center Study. *International Journal of Neural Systems*, *31*(5), 2050074. https://doi.org/10.1142/S0129065720500744

Tuncer, E., & Doğru Bolat, E. (2022). Classification of epileptic seizures from electroencephalogram (EEG) data using bidirectional short-term memory (Bi-LSTM) network architecture. *Biomedical Signal Processing and Control*, *73*, 103462. https://doi.org/10.1016/j.bspc.2021.103462

Yen, S.-J., & Lee, Y.-S. (2009). Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications*, *36*(3), 5718–5727. https://doi.org/10.1016/j.eswa.2008.06.108

7. Appendix: Implementation

7.0.1 Data Source and Preprocessing

The data was sourced from the Temple University Hospital (TUH) Artifact Corpus (Obeid and Picone, 2016). We obtained the annotated artifact data upon request to the authors of the TUH Artifact Corpus. The data was then saved and preprocessed through a series of functions, which can be found in the `data/utils/preprocessing.py` file. The preprocessing pipeline involved the following steps:

1. Using the `mark_artifact(filename, ...)` function to mark artifacts and create the target label(s) y_{arr}.
2. Reading the EEG signals from the file, focusing on the 19 relevant channels.
3. Normalizing the data to achieve a mean of 0 by subtracting the mean and dividing by the standard deviation using the `normalize(data)` function.
4. Applying z-scoring to the data and smoothing out outliers using a standard deviation with the `z_score(data, ...)` function.
5. Stacking the artifact information with the EEG data.
6. Converting the raw data into chunks and resampling based on specified parameters, such as seconds per sample and target frequency with the `convert_to_chunks(data, ...)` function.
7. Returning the preprocessed data.

After preprocessing, the data was organized in the format `(n_samples, n_channels, n_timepoints)` and saved as a Pickle file (.pkl) in our Google Drive folder for access.

7.0.2 Data Preparation

We implemented the Python module `data.py` with the class `Data` where all necessary functions for preparing the data and creating useful statistics can be found. Follow this link to get to the code. With the function `load(self, data_path)` you can extract the downloaded pickled file into a numpy array. Next, the `split(self, ...)` function splits the dataset into a train, validation and test dataset. You can adjust the split size as you like. The `prepare_data(self, ...)` function is capable of preparing the data for our two architecture types indicated by passing suitable argument values for the arguments `tranformer`, `lstm` and `ccn` in a boolean format. Moreover, the arguments `balanced` decides over the use of undersampling of the data and the argument `dataset` determines the return type of the function being either a Tensorflow dataset or numpy arrays. The function `plot(self, ...)` plots one sample with 19 feature channels and 2 channels for the presence of the two artifacts. Lastly, the two functions `get_balance(self, ...)` and `get_artifact_duration(self, ...)` provide useful information about the data. A plot of a data sample and other statistics as the duration and balance of the datasets can be found in the notebook data_statistics_visualisations.ipynb.

7.1. Hyperparameter Tuning

The heart of our project was finding the most suitable architecture parameters for the task. We provide two jupyter notebooks with `transformer_architecture_tuning.ipynb` and `lstm_architecture_tuning.ipynb`. Both notebooks make use of the `Data` class to properly preprocess the data with all the suitable arguments. Afterward both notebooks use the custom Python module `experiment.py`. The module consists of the functions `create_model(self, ...)`, `run_model(self, ...)` and `run_experiment(self, ...)` to provide all the functionalities to run a hyperparameter optimization search. To search a predefined search space, all we do is to create a new `Experiment` object and call the function `run_experiment(self, ...)`. This function gets the following inputs:

- `hparams_dict`: Dict, a dictionary with all hyperparameters as `hp.HParam` object
- `logdir`: String, the path to the log directory
- `metrics`: List, a list with all metrics to be logged
- `train`: Tf Dataset, the training dataset
- `val`: Tf Dataset, the validation dataset
- `test`: Tf Dataset, the test dataset
- `epochs`: Int, number of epochs to train

We used the tensorboard plugin `hparams` to make the hyperparameters and metrics visible on our tensorboard.

7.2. Testing

To properly evaluate our models on the test set, we provided the custom Python module `tester.py` to load a saved model with the function `load_model(self, ...)` with its checkpoints and calculate accuracy, precision, recall and f1-score of the model evaluated on the test dataset with the function `get_metrics(self, ...)`.

7.3. Architectures

We provide two architecture classes with an Attention-based model and LSTM-based model. The optimized architecture classes can be found in the folder `architectures` in our GitHub repository. Both classes consist of a constructor for specifying the architecture with the given arguments when creating an object of the classes. This implementation allows us to dynamically search for a wide variety of models during hyperparameter optimization. Moreover, both classes have a `call(self, ...)` function to calculate forward propagation of the input and backward propagation of the gradients.

7.3.1 Attention-based architecture

The Convolutional Attention-based model is implemented in `CNNAttentionNetwork.py`. When creating an instance of the class, multiple arguments have to be provided:

- `num_units`: the number of units in each dense layer.
- `num_layers`: the number of units in each dense layer.
- `num_heads`: the number of attention heads in the MultiHeadAttention layer
- `num_conv_layers`: the number of convolutional and max-pooling layer pairs in the convolutional block
- `attention`: this parameter controls whether multihead-attention is used

7.3.2 LSTM-based architecture

The LSTM based architectures can be found in the Python module `LSTM.py` inside our repository. To create an object of the class, multiple arguments have to be provided to specify the exact structure of the model. The following arguments have to be provided:

- `num_hidden_units`: Integer, the number of hidden units in lstm layer
- `num_lstm_layers`: Integer, the number of LSTM layers
- `num_dense_units`: Integer, the number of units in each dense layer
- `num_dense_layers`: Integer, the number of dense layers in the model
- `num_conv_layers`: Integer, the number of convolutional layers in the convolutional block
- `increase`: bool, whether hidden units increase with in higher lstm layers
- `bidirectional`: Boolean, whether bidirectional or multilayer lstm used

7.3.3 Regularization

We implemented two regularization methods to prevent overfitting. Firstly, each of our architectures features a dropout layer before the block of dense layers. To find the optimal probability for the dropout layer, we searched the range of 0.3 to 0.5 and settled on 0.3 as the optimal value. Secondly, we applied Early Stopping during the training phase which resulted in the stop of training if the validation did not improve for 5 consecutive epochs. The code for Early Stopping can be found in our module `experiment.py`

7.4. Logging

7.4.1 Metrics

We logged the accuracy, precision and recall for each model architecture type through a Keras callback. Our results for the LSTM-based models can be found here and for the Attention-based model here.

7.4.2 Checkpoints

We saved the checkpoints of the best performing models through a Keras callback. They can be found in our repository when following this link.

7.4.3 Model saving

After our models were trained for the set amount of epochs, we saved it for later purposes. If we stopped the training early due to Early Stopping as explained above, we saved the model with the lowest loss on the training data set. Our models can be found in our GitHub repository.

7.4.4 Data transfer from server

In order to transfer data from the cloud gpu to our local machines we set up a SSH connection between the two parties. To achieve this, we generated a pair of public and private keys. After creating the keys, we stored the public key on the Vast AI platform, allowing for secure authentication without requiring a password. Finally, we utilized the 'scp' (secure copy) command to transfer the log files from the remote server to the local machine, ensuring a safe and efficient data transfer while maintaining the confidentiality of our research data. A documentation of this process can be found in our GitHub repository.