SciPy 2023

July 10 – July 16, 2023

Proceedings of the 22nd Python in Science Conference
ISSN: 2575-9752

EEG-to-fMRI Neuroimaging Cross Modal Synthesis in Python

David Calhas![1,2]

[1] INESC-ID, [2] Instituto Superior Tecnico

Abstract

Electroencephalography (EEG) and functional magnetic resonance imaging (fMRI) are two ways of recording brain activity; the former provides good time resolution but poor spatial resolution, while the converse is true for the latter. Recently, deep neural network models have been developed that can synthesize fMRI activity from EEG signals, and vice versa. Because these generative models simulate data, they make it easier for neuroscientists to test ideas about how EEG and fMRI signals relate to each other, and what both signals tell us about how the brain controls behavior. To make it easier for researchers to access these models, and to standardize how they are used, we developed a Python package, EEG-to-fMRI, which provides cross modal neuroimaging synthesis functionalities. This is the first open source software enabling neuroimaging synthesis. Our main focus is for this package to help neuroscience, machine learning, and health care communities. This study gives an in-depth description of this package, along with the theoretical foundations and respective results.

Keywords Electroencephalography, Functional Magnetic Resonance Imaging, Synthesis, Deep Learning, Learning, Machine Learning, Computer Vision

1. Introduction

Neuronal activity, usually measured through electroencephalography (EEG), is related to haemodynamical activity, measured through functional magnetic resonance imaging (fMRI). The first captures the dynamics of the electrical field, whose source is located from the firing neurons' action potentials. In its turn, the second measures the blood supply dynamics. These two while being studied simultaneously [1]–[8] differ in many aspects such as: temporal and spatial resolution, brain functions captured, recording and hardware cost. Recently we have seen several studies that use deep neural network models [9] to learn a mapping from EEG data to and from fMRI data [10], [11]. These are a type of generative models [12], that sample/synthesize instances from a different data source (instead of a distribution). Such a model could allow health care cost reductions and discoveries of new neuroscience insights on the relationship between these two modalities. Indeed, pathologies that require MRI scans diagnostics benefit from a lower cost EEG assessment, since availability of MRI hardware is very scarce [13]. As Python [14] becomes a hub for scientific development [15]–[17] we find the need to provide open source software that provides solutions for *EEG to fMRI synthesis*, urgent, in order for third party scientific contributions coming from other laboratories to coexist and health care software integration to develop for diagnostic settings. To that end, we provide a description of the open source software EEG-to-fMRI, which originated from an academic scientific project funded by Fundação para a Ciência e Tecnologia, and make publicly available a github repository.

Published Jul 10, 2023

Correspondence to
David Calhas
david.calhas@tecnico.ulisboa.pt

Data Availability
Source code available:
https://github.com/eeg-to-fmri/eeg-to-fmri

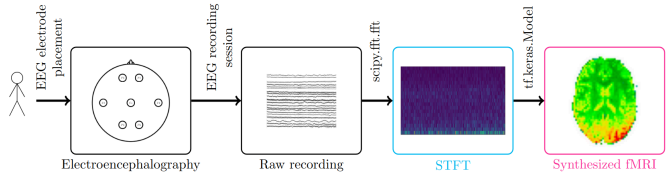

Figure 1. *The pipeline of the EEG to fMRI synthesis project consists of processing EEG recordings, that are sourced from a human subject that goes through an EEG recording sesssion, then processing the channel by time signal and taking the short time Fourier transform. The latter, gives us the time frequency representation of the EEG signal. The novelty of this software is that it provides a model that given as input the EEG signal it predicts the corresponding fMRI volume associated with that segment. A video demonstration of the whole pipeline is available on Youtube.*

2. Methods

The mapping function provided in this software is the one proposed by [11]. It consists on transforming the EEG from a channel by time representation to a channel by time-frequency one, achieved using the short time Fourier transform [18] by means of the fast Fourier transform (scipy.fft.fft) available in the SciPy package. The latter corresponds to the second step of the diagram illustrated in Figure Figure 1. In the second step, this representation is then forward through a deep neural network (implemented as a tf.keras.Model), with contributions ranging from Resnet blocks [19], an automated machine learning framework [20] and Fourier features [21]. Ultimately, this enables the prediction of an fMRI volume associated with a 26 second segment of an EEG recording.

3. Description

The dependencies of each component are described in an UML diagram. Overall to install the package, the user is required to install the following dependencies: `tensorflow 2.9.0`, `matplotlib 3.5.3`, `mne 0.23.4`, `nilearn 0.7.0`, `tensorflow_probability 0.12.2`, `tensorflow_determinism 0.3.0`, and `tensorflow_addons 0.19.0`. As seen there is a high dependency in tensorflow related packages. This is due to the whole system provided being built in tensorflow, a library that enables automatic differentiation and is widely used for deep learning model development.

3.1. Package modules

This package, as it is provided, has eight main modules:

- *models*: here you will find the code that implements the models for synthesis and classification. The synthesis models are located in the *synthesizers.py*, where the class *EEG_to_fMRI* is implemented. This class defines a `tf.keras.Model`, composed of two encoders, one for the EEG and another for the fMRI. Additionally, there is a decoder that maps the latent EEG representation, that is the output of the EEG encoder, to the estimate of the fMRI volume associated. The fMRI encoder is defined in the *fmri_ae.py* file. The task would not be complete without the extrapolation of the synthesis model to a classification task. To that end, two linear classifiers are provided, the *ViewLatentContrastiveClassifier* and the *ViewLatentLikelihoodClassifier*, corresponding to a contrastive latent loss and a cross entropy error driven classification, respectively. The extrapolation is made by taking the output of the neural flow that comes from the EEG, that is the EEG encoder and the decoder.
- *layers*: this module contains the layers which compose the synthesizer models. It provides five main types of layers. First, the *TopographicalAttention* computes a self attention mechanism in the channels dimension of the EEG [11] that allows the representation to be correctly processed by the convolutional blocks. Second, the *ResnetBlock* [19] implements a residual block composed of convolutional layers. This block allows efficient gradient propagation, by tackling the vanishing gradient phenomena. Third, the *FourierFeatures* layer projects *cosines* functions of different shifts and biases to build a latent spectral basis for the prediction of the fMRI volume. Fourth, the *DenseVariational* is an implementation of the `tf.keras.layers.Dense` layer for two-dimensional inputs, whose weights are drawn from gaussian distributions. Last, but not least, we provide the *DCT* based layers, which implement the discrete cosine transform [22]. These layers are useful for alternative ways of decoding the latent representation of the EEG to produce the desired fMRI volume.
- *regularizers*: here the implementation of different regularizers for the synthesizer model is provided. Most importantly in the *activity_regularizers.py* file is found the *OrganizeChannels* implementation which can be added in the *TopographicalAttention* layer, so that the layer does not surpress any channel;
- *learning*: here are implemented the routines for the optimization of the models, namely the *losses* and the *train* procedures. The *train.py* is a generalizable training routine that fits any type of `tf.keras.Model` instance. Following, the *losses.py* contain a set of losses that are implemented to train the models provided in the *models* module. Most importantly, here you will find the *mae_cosine* for the deterministic versions of the *EEG_to_fMRI* model, the *LaplacianLoss* for variational versions, and the *ContrastiveClassificationLoss* which serves as the cost function for the *ViewLatentContrastiveClassifier*;
- *explainability*: this module contains explainability methods that were employed for the models developed. The implementation for the layer wise relevance propagation [23] can be found in *lrp.py*. In particular, since this type of algorithm is not model agnostic, we

have the implementation for all the permutations' of operations that the *EEG_to_fMRI* model can have;

- *data*: this is maybe the most important module for researchers wanting to try out their collected data with this software. Here all of the functions, that read data and manipulate it as such to allow the efficient training, are implemented. Starting with the *eeg_utils.py*, where the *get_eeg_instance_< DS>* function is implemented for a limited set of datasets that participated in the experiments of the associated studies. The < DS> stands for the identifier of the dataset. Currently there are a set of functions implemented for publicly available datasets. Please check the description of the code in the github repository for more details. In the *fmri_utils.py* file one finds functions of the form *get_individuals_paths_< DS>* , that have the exact same function as the functions to read EEG recordings, but in this case they read nifti file format. These type of files are the standard format for fMRI recordings. Finally, the *data_utils.py* and *preprocessed_data.py* are responsible for concatenating the EEG and fMRI instance pairs, with all the alignments and event synchronization events taken into account, as well as the processing to build `tf.data.Dataset` classes.
- *metrics*: in this module are implemented the metrics usually used to evaluate synthesis of generated images, such as the root mean squared error, mean absolute error, structural similarity index measure [24], among others;
- *utils*: Last but not least, is the utilities module, which provides the user with print functions, configuration of the tensorflow environments and several visualizations that were used in the original studies that developed the package.

3.2. New data integration

In order to use the software provided for new data, we recommend that the dataset is structured as shown in Figure Figure 3.

If the data is provided as illustrated then the user only has to name the directory of the data as *01*. This should suffice for the correct loading of the data. Any type of issue that is

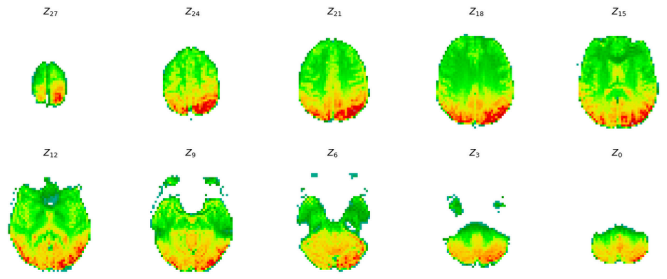

Figure 2. *The synthesized signal of fMRI. This is the output visualization when running the code for the classification notebook.*

```
NEW
|       .
|       ..
|       README.md
└──────EEG
|       └──────sub-001
|       |       |       FILE.eeg
|       |       |       FILE.vhdr
|       |       |       FILE.vmrk
|       └──────sub-002
|       ...
└──────BOLD
        └──────sub-001
        |       |       FILE.anat
        |       |       FILE.nii.gz
        └──────sub-002
        ...
```

Figure 3. *Recommended structure of the dataset directory.*

encountered for this package should be published as an issue in the official github repository.

3.3. Building an EEG to fMRI model

For the user to build an EEG to fMRI model, they have to first import the correct library module and *tensorflow*.

```
from tensorflow as tf
from eeg_to_fmri.models.synthesizers
        import EEG_to_fMRI
from eeg_to_fmri.models.synthesizers
        import parameters
from eeg_to_fmri.models.synthesizers
        import na_specification_eeg
from eeg_to_fmri.models.fmri_ae
        import na_specification_fmri
```

Let us define the size of the EEG representation $\vec{x} \in \mathbb{R}^{64\times134\times10\times1}$, the fMRI representation $\vec{y} \in \mathbb{R}^{64\times64\times30\times1}$, and the latent representation, for both the EEG and fMRI, that is $\vec{z}_x, \vec{z}_y \in \mathbb{R}^{7\times7\times7}$.

```
fmri_dim=(64,134,10,1)
fmri_dim=(64,64,30,1)
latent_dim=(7,7,7)
```

Then, we have to define the parameters for the model, these are provided as a variable in the *synthesizers.py*.

```
learning_rate,weight_decay,kernel_size,
stride_size,batch_size,latent_dimension,
n_channels,max_pool,batch_norm,
skip_connections,dropout,
n_stacks,outfilter,local=parameters
```

Some of these parameters will also be used to define the fMRI encoder. Since the fMRI encoder is a different class, we need to define the initialization parameters.

```
fmri_parameters=(parameters, latent_dim,
    fmri_dim, kernel_size, stride_size,
    n_channels, max_pool, batch_norm,
    weight_decay, skip_connections,
    n_stacks, True, False,
    outfilter, dropout, None,
    False, na_specification_fmri)
```

Next, we have all of the parameters necessary to build a simple deterministic version of the EEG to fMRI model.

```
with tf.device('/CPU:0'):
    model = EEG_to_fMRI(latent_dim, eeg_dim,
        na_specification_eeg, n_channels,
        weight_decay=weight_decay,
        skip_connections=True, batch_norm=True,
        fourier_features=True,
        random_fourier=True,
        topographical_attention=True,
        conditional_attention_style=True,
        conditional_attention_style_prior=False,
        local=True, seed=None,
        fmri_args = fmri_parameters)
```

The model is built once it is specified the input dimension, this is done through the *tf.keras.Layer#build*. This will initialize all of the weights of the network.

```
model.build((None,)+eeg_dim, (None,)+fmri_dim)
```

3.4. Cost function and optimization

Regarding the cost function, which is provided in the *learning* module at the *losses.py* file, we can specify different metrics that are provided. Take, for instance, the example of using an approximation of the mean absolute error at the output for the fMRI volume prediction $\hat{\vec{y}}$, and approximation of the latent representations of the fMRI as proposed by [11]. This is reduced to the mathematical formula

$$\mathcal{L}(\vec{x},\vec{y},\vec{z}_x,\vec{z}_y) = ||\vec{y}-\hat{\vec{y}}||_1^1 + 1 - \frac{\vec{z}_x \cdot \vec{z}_y}{||\vec{z}_x||_2^2 \, ||\vec{z}_y||_2^2}. \quad (1)$$

In terms of code, this is already implemented and can be loaded directly.

```
from eeg_to_fmri.learning import losses
loss_fn=losses.mae_cosine
```

The optimizer is already provided in the `tensorflow` library and its corresponding learning rate is given in the parameters variable.

```
optimizer=
        tf.keras.optimizers.Adam(learning_rate)
```

Training the model requires an input, the EEG $\vec{x}$, and an output, the fMRI $\vec{y}$. The architecture processes both the EEG and fMRI, producing the latent representation for both. Proceeding the latent EEG representation, $\vec{z}_x$, is fed to the decoder which estimates the fMRI $\hat{\vec{y}}$.

```
def apply_gradient(model, optimizer, loss_fn,
    x, y, return_logits=False, call_fn=None):
    with tf.GradientTape(persistent=True) as
                                        tape:
        logits=model(x, y, training=True)
        regularization=0.
        if(len(model.losses)):
            regularization=
                tf.math.add_n(model.losses)
        loss=loss_fn(y, logits)+regularization
    gradients=tape.gradient(loss,
            model.trainable_variables)
    optimizer.apply_gradients(zip(gradients,
        model.trainable_variables))
    return tf.reduce_mean(loss)
```

The training routine functions are found in the *learning* module. The function *apply_gradient* computes the forward and backward pass of the neural network. Note that the input of the *EEG_to_fMRI* model is composed of two tensors, the EEG and fMRI, which is the reason why one gives the model both *x* and *y*. The loss computes both the estimation of the fMRI according to the mean absolute error (MAE) and the cosine distance between the latent representations. Regularization terms, such as weight decay and other activity regularizers that may or may not participate in the model, are also added to the loss. The loss function used for the synthesis task, the one used in [11], is the MAE and the cosine distance. This loss function is defined in the *learning* module, found at the *losses.py* file. This loss receives a *tf.Tensor* object, *y_true*, and a list of *tf.Tensor*. The list of tensors contains the outputs of the neural network that should be approximate the ground truth. The first element of the list is the estimated fMRI volume and the second and third items are the latent EEG and fMRI representations, respectively.

```
def mae_cosine(y_true, y_pred):
    return tf.reduce_mean(
        tf.math.abs(y_pred[0] - y_true),
        axis=(1,2,3)) +
        cosine(y_pred[1], y_pred[2])
```

At test time, the *EEG_to_fMRI* model can discard the fMRI input. To that end, only the decoder attribute is called, which is composed of the EEG encoder and the decoder.

```
def call(self, x1, x2):
    if(self.training):
        return [self.decoder(x1),
            self.eeg_encoder(x1),
            self.fmri_encoder(x2)]

    return self.decoder(x1)
```

Note that, the *pretrained_EEG_to_fMRI* class processes a pretrained model of the type *EEG_to_fMRI* and builds a new model that only processes EEG and outputs an fMRI prediction given the representation learned. This class is built to be then appended with a classifier, that can be either a

ViewLatentContrastiveClassifier or a *ViewLatentLikelihoodClassifier*.

4. Examples

In this section, we walk through the examples given in jupyter notebooks.

4.1. Synthesis

We provide a compressed version of the dataset of [25]. Users can directly execute the code and have both the python package, as well as the dataset, setup in a google colab environment. The flow of execution has been already described. In the end a synthesized fMRI is shown, as illustrated in Figure Figure 2. This image is built using the *viz_utils.py*. The user can find metrics for synthesis evaluation in *eeg_to_fmri.metrics.quantitative_metrics*. We report results from the [11] study on the NODDI dataset [25]. An example with a reduced dataset is available in this synthesis notebook. The best model, which used the configuration of the *eeg_to_fmri.models.synthesizers.EEG_to_fMRI* achieved 0.3972 RMSE and 0.4613 SSIM. This constitutes the state-of-the-art for this task and provides a view that can be applied in EEG only datasets for classification task.

4.2. Classification

We also provide a compressed version of the dataset of [26]. This example, available in this classification notebook, is based on a publicly available dataset that contains individuals diagnosed with schizophrenia and healthy controls. The whole goal of the project is to be applied in an health care setting and to this end we employ an end to end software solution. The whole software package is able to synthesize fMRI and adapt to a classification setting, that given EEG recordings outputs a set of probabilities for each group of people considered in the dataset.

5. Collaboration

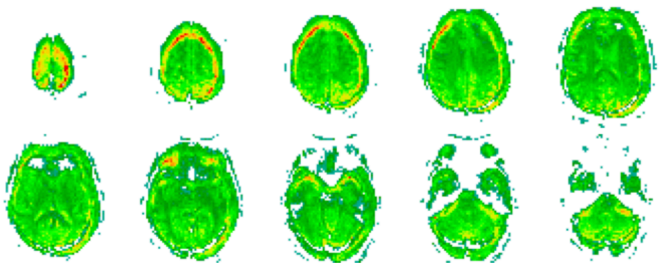

Figure 4. *Output of the predicted fMRI when given an EEG representation. Note that, due to the EEG encoder being optimized towards classifying the data according to the groups of individuals defined, e.g. schizophrenic and healthy controls, the decoder (that has the parameters frozen) gives a slightly altered representation. This change is seen in the produced fMRI, where activity beyond the limit of the human scalp is reported. Please recall Figures Figure 1 and Figure 2 to directly compare with an fMRI representation without these flaws.*

Ultimately, the goal of this package is to collect, in one package, methods for EEG to fMRI synthesis. We welcome contributions from authors of related work such as [10]. In the future, we plan to add a module or example folder with implementations of these approaches, so that other research groups can easily access them and reproduce key results.

On a higher level, this software is encouraged for testing its applicability in health care settings. The impact, that such mappings from EEG to fMRI, would have on society is enormous, given that the diagnostic is faithful. Take for instance the example of the MRI machine density across the African continent. In the worst case scenario, Nigeria has a density of 0.33 MRI machines per million people, according to [13]. To let that sink in, imagine having to wait in a line of 3 million people to get a diagnostic exam. This type of waiting bottleneck impacts greatly the development of diseases for the worse. Countries in such conditions would greatly benefit from contributions that further advance this scientific field. Even fortunate countries, whose economy thrives, that are able to provide their populations with a good ratio of MRI machines, they still have small portions of the population who live in remote areas. These people find it hard to get quality health care, without having to travel significant distances.

6. Conclusion

This is the first package, to the best of our knowledge, that provides a machine learning oriented synthesis between functional neuroimaging modalities (EEG and fMRI). It is targeted to help the neuroscience community, in tasks such as modality augmentation, resolution enhancement, neuroimaging explainability techniques, among others. We hope to motivate researchers, scientists, and software developers to contribute to this package which we have been so passionate about throughout the last years.

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e Tecnologia under the PhD Grant SFRH/BD/5762/2020 to David Calhas.

References

[1] H. Shibasaki, “Human brain mapping: hemodynamic response and electrophysiology”, *Clinical Neurophysiology*, vol. 119, no. 4, pp. 731–743, 2008, doi: 10.1016/j.clinph.2007.10.026.

[2] Q. Yu *et al.*, “Building an EEG-fMRI multi-modal brain graph: a concurrent EEG-fMRI study”, *Frontiers in human neuroscience*, vol. 10, p. 476, 2016, doi: 10.3389/fnhum.2016.00476.

[3] Y. He *et al.*, “Spatial–temporal dynamics of gesture–speech integration: a simultaneous EEG-fMRI study”, *Brain Structure and Function*, vol. 223, pp. 3073–3089, 2018, doi: 10.1007/s00429-018-1674-5.

[4] G. M. Rojas, C. Alvarez, C. E. Montoya, M. de la Iglesia-Vayá, J. E. Cisternas, and M. Gálvez, “Study of resting-state functional connectivity net-

works using EEG electrodes position as seed", *Frontiers in neuroscience*, vol. 12, p. 235, 2018, doi: 10.3389/fnins.2018.00235.

[5] L. Bréchet, D. Brunet, G. Birot, R. Gruetter, C. M. Michel, and J. Jorge, "Capturing the spatiotemporal dynamics of self-generated, task-initiated thoughts with EEG and fMRI", *Neuroimage*, vol. 194, pp. 82–92, 2019.

[6] I. Daly, D. Williams, F. Hwang, A. Kirke, E. R. Miranda, and S. J. Nasuto, "Electroencephalography reflects the activity of sub-cortical brain regions during approach-withdrawal behaviour while listening to music", *Scientific reports*, vol. 9, no. 1, pp. 1–22, 2019, doi: 10.1038/s41598-019-45105-2.

[7] C. Cury, P. Maurel, R. Gribonval, and C. Barillot, "A sparse EEG-informed fMRI model for hybrid EEG-fMRI neurofeedback prediction", *Frontiers in neuroscience*, vol. 13, p. 1451, 2020, doi: 10.3389/fnins.2019.01451.

[8] R. Abreu, J. Jorge, A. Leal, T. Koenig, and P. Figueiredo, "EEG microstates predict concurrent fMRI dynamic functional connectivity states", *Brain topography*, vol. 34, no. 1, pp. 41–55, 2021, doi: 10.1007/s10548-020-00805-1.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[10] X. Liu and others, "A convolutional neural network for transcoding simultaneously acquired EEG-fMRI data", in *NER*, 2019. doi: 10.1109/ner.2019.8716994.

[11] D. Calhas and R. Henriques, "EEG to fMRI Synthesis Benefits from Attentional Graphs of Electrode Relationships", *arXiv preprint arXiv: 2203.03481*, 2022.

[12] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[13] G. I. Ogbole, A. O. Adeyomoye, A. Badu-Peprah, Y. Mensah, and D. A. Nzeh, "Survey of magnetic resonance imaging availability in West Africa", *Pan African Medical Journal*, vol. 30, no. 1, 2018, doi: 10.11604/pamj.2018.30.240.14000.

[14] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[15] C. R. Harris *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[16] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python", *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[17] M. Abadi *et al.*, "Tensorflow: a system for large-scale machine learning.", in *Osdi*, 2016, pp. 265–283.

[18] J. Allen, "Short term spectral analysis, synthesis, and modification by discrete Fourier transform", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 3, pp. 235–238, 1977, doi: 10.1109/tassp.1977.1162950.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. doi: 10.1109/cvpr.2016.90.

[20] D. Calhas, V. M. Manquinho, and I. Lynce, "Automatic Generation of Neural Architecture Search Spaces", in *Combining Learning and Reasoning: Programming Languages, Formalisms, and Representations*, 2022.

[21] M. Tancik *et al.*, "Fourier features let networks learn high frequency functions in low dimensional domains", *arXiv preprint arXiv: 2006.10739*, 2020.

[22] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform", *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974, doi: 10.1016/b978-0-08-092534-9.50007-2.

[23] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation", *PloS one*, vol. 10, no. 7, p. e130140, 2015, doi: 10.1371/journal.pone.0130140.

[24] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity", *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004, doi: 10.1109/tip.2003.819861.

[25] F. Deligianni, M. Centeno, D. W. Carmichael, and J. D. Clayden, "Relating resting-state fMRI and EEG whole-brain connectomes across frequency bands", *Frontiers in Neuroscience*, vol. 8, p. 258, 2014, doi: 10.3389/fnins.2014.00258.

[26] A. Padée and others, ""Fribourg Ultimatum Game in Schizophrenia Study"". OpenNeuro, 2022. doi: 10.18112/openneuro.ds004000.v1.0.0.