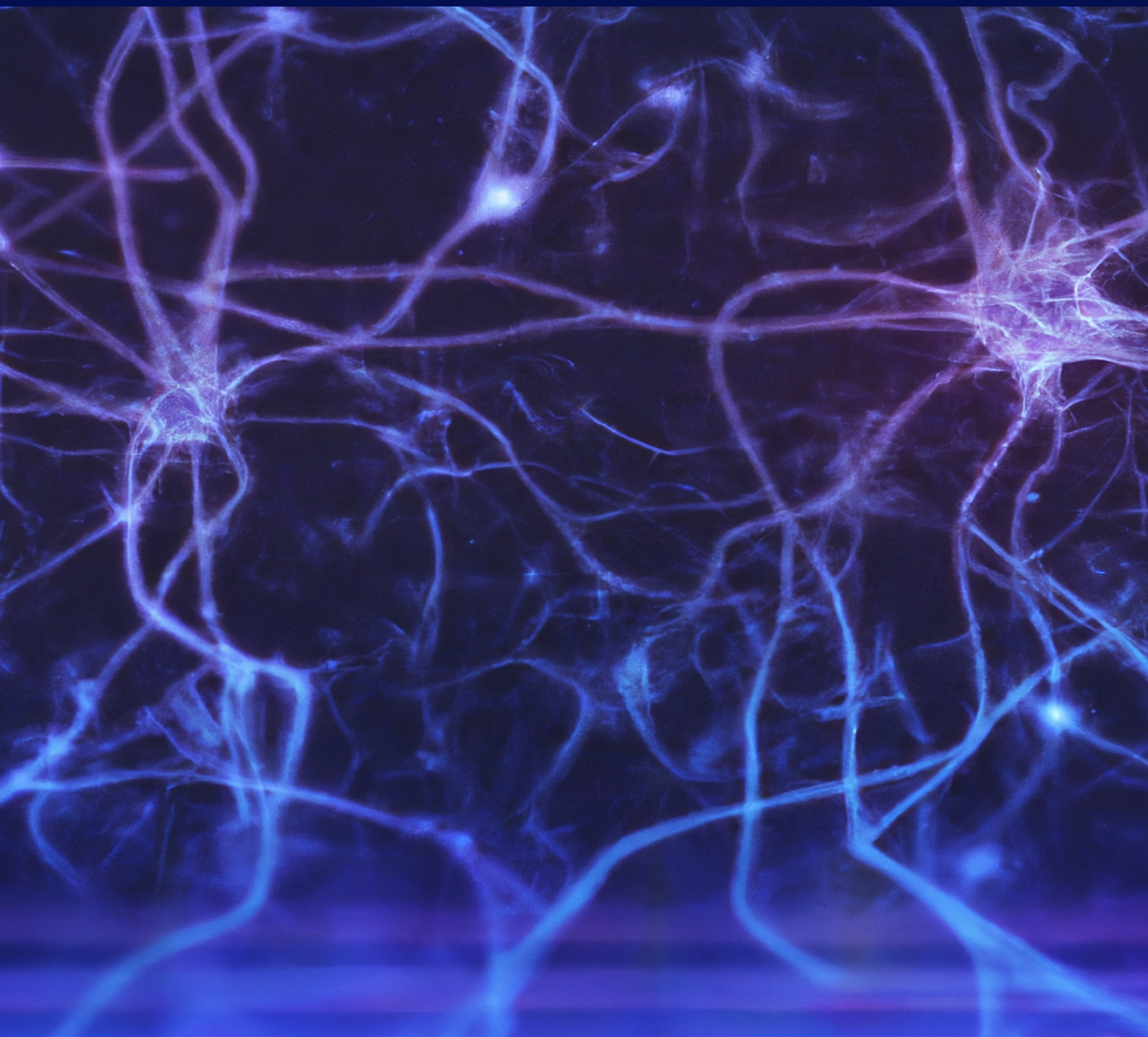
DTU Health Tech
Department of Health Technology

Automatic seizure detection
based on Machine Learning and EEG

Bachelor's Thesis

Zhensheng Chen



Automatic seizure detection
based on Machine Learning and EEG

Bachelor's Thesis
September, 2022

By: Zhensheng Chen
Supervisors: Ying Gu and Helge Bjarup Dissing Sørensen
Cover photo: Generated by Dalle 2 Open IA

”For my friend who passed away of epilepsy while I was writing this thesis.
I hope that one day humanity will be able to overcome this disease.”

Automatic seizure detection

based on Machine Learning and EEG

Zhensheng Chen

Abstract

The diagnosis and treatment of epilepsy depend on accurate seizure detection. In clinical practice, the evaluation of seizures is done by visual inspection of an electroencephalogram (EEG). it is very time-consuming and requires trained experts. Automatic seizure detection is important. Machine learning approaches are intensely being applied to this problem due to their ability to classify seizure conditions from a large amount of data, and provide pre-screened results for neurologists.

This work proposes a variety of experiments with different machine-learning architectures (support vector machine SVM, K nearest neighbour KNN, random forest RF, feef forward neural network FFNN and convolutional neural network CNN) for the detection of epileptic seizures using multichannel EEG signals from the CHBT-MIT Scalp EEG Database. The best model built in this work contains a combination of a feed-forward neural network (FFNN) and a convolutional neural network (CNN). CNN input images are constructed by applying short-time Fourier transform (STFT) to electroencephalography (EEG) signals and then merged with statistical metrics into a FFNN. The best model of this project showed an outstanding performance of 98.615% accuracy, 98.737% sensitivity and 98.425% specificity. This work also includes a discussion of other exciting ideas that could lead to future research investigations.

Contents

1 Introduction

Epilepsy is a serious disorder of the central nervous system, which affects 1% of the world's populations. Approximately 30% of epilepsy patients are not helped effectively by medication. Due to loss of control and consciousness, patients often experience serious injuries. The unpredictable occurrences and consequences of seizures deeply impact patients' quality of life. Moreover, each year there is about SUDEP (Sudden Unexpected Death in Epilepsy) for every 1,000 people with epilepsy.

The electroencephalogram (EEG) is one of the most popular methods for studying epilepsy and detecting changes in electrical brain activity that may suggest an impending seizure. An epileptic seizure is a serious clinical issue. Manual EEG inspection is required for the diagnosis of epilepsy, which takes time and is prone to inaccuracy. According to Elger and Hoppe's research, more than half of the seizures seen during long-term video EEG monitoring were unreported [1], whereas fewer than half of the epileptic seizures that patients document can record precisely. The creation of a workable and trustworthy intelligent diagnostic method for automated seizure detection is crucial.

1.1 Related work

Studies about automatic seizure detection started in the 1970s and different algorithms to solve this problem were presented. Back then, automated seizure detection methods may rely on the identification of different patterns such as increasing amplitude[2], maintaining rhythmic activity [3], or flattening the EEG [4]. Several algorithms have been developed based on spectral [5] or wavelet characteristics [6], amplitude versus background activity and spatial context [7]. There are also several approaches using chaotic features such as correlation dimension [7], Lyapunov exponents [8] and entropies [9]. These characteristics can then be used for the classification of EEG signals using statistical methods(e.g. nearest neighbour classifiers [10], decision trees [11], ANNs [6], support vector machines (SVMs) [12], random forest [13]...) Seizure detection systems must be highly sensitive, even if this results in a large number of false detections. Such systems can then be used significantly reduce the amount of data required for examination; Neurophysiologists can then easily rule out false findings.

Seizure detection systems can also be used to create warning systems since early seizure detection can alert the patient to an incoming seizure. Additionally, they notify medical personnel and inform them to conduct behaviour tests to better determine which particular tasks may be hampered by a seizure and aid them in determining the cause of the seizure activity.

Patients' seizures can be prevented by taking anti-epileptic drugs, however, the consumption of these drugs can lead to several side effects (e.g. aplastic anemia, fatal liver toxicity, irreversible visual field defects)[14]. Techniques used to forecast seizures (e.g. e time-domain analysis [15], frequency-based methods [16], nonlinear dynamics and chaos [17], and intelligent systems [18].) can change drug infusion to give on demand and might eliminate side effects in some patients by consuming less amount of anti-epileptic drugs.

1.2 Project Description

This project aims to develop an automated seizure detection system using machine learning architectures. The system will be trained on a dataset of EEG recordings that have been manually annotated by an expert with the location of seizures. The goal of the system is to accurately identify seizures in new EEG recordings, thus supporting clinicians in their evaluation of seizures and making the process more efficient.

The performance of the system will be evaluated by comparing the seizure detection times identified by the model to those identified by the expert. The project will investigate different machine learning architectures and compare their results.

The objectives of this project are to gain experience working with EEG data Figure 1.1, gain an understanding of epileptic seizures, and explore the use of machine-learning techniques for automated seizure detection. This research aims to contribute to the field by developing a reliable and efficient automated seizure detection system that can be used in a clinical setting to aid in the diagnosis of epilepsy.

This project builds upon the extensive research that has already been conducted in the field of automated seizure detection using machine learning and deep learning techniques. Many studies have achieved impressive results, however, this project aims to take a novel approach by combining different techniques and architectures to find new and improved methods for seizure detection. This research may bring fresh perspectives and insights to the field by investigating new combinations of methods and techniques that have not yet been explored. The ultimate goal of this project is to develop a different automated seizure detection system that can aid in the diagnosis and treatment of epilepsy and hopefully this work can contribute to investigations in this field and further advance the understanding and treatment of epilepsy.

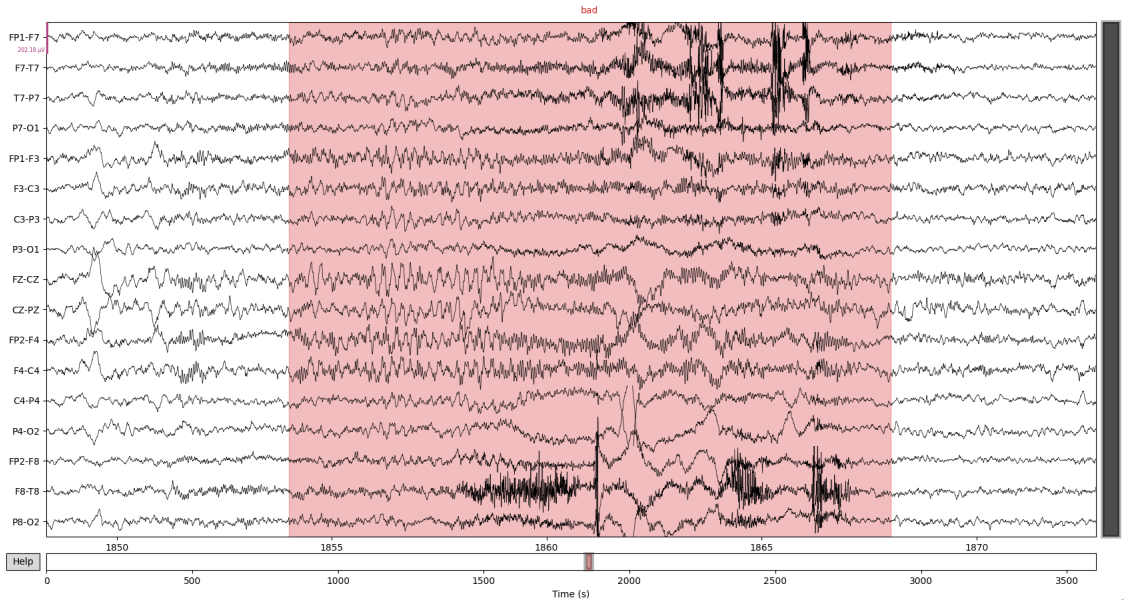


Figure 1.1: Sample of EEG displayed with Python MNE library, red section belongs to seizure

2 Background

In this section, context is explained in order to provide a better understanding of the entire project.

2.1 Epilepsy

Epilepsy is a chronic neurological disorder characterized by recurrent seizures caused by abnormal electrical activity in the brain. It is one of the most common neurological disorders, affecting around 1% of the world population[19, 20]. The seizures can manifest in different ways, from mild convulsions to severe convulsive seizures and can vary in frequency from once in a lifetime to several per day[21].

Epileptic seizures are classified into two main categories: focal seizures, which originate from a specific area of the brain, and generalized seizures, which involve the entire brain [22]. The cause of epilepsy is not always known, but it can be caused by a variety of factors such as head injury, brain infections, brain tumours, genetic factors and developmental disorders [23].

The diagnosis of epilepsy is based on the patient's history and a clinical examination, including a detailed description of the seizures and their characteristics [24]. Electroencephalography (EEG) is a valuable tool in the diagnosis and management of epilepsy, as it allows the detection of abnormal electrical activity in the brain [25].

Treatment of epilepsy includes medication, surgery, and lifestyle changes. Antiepileptic drugs are the first line of treatment for most patients with epilepsy [26]. In some cases, surgical treatment may be considered when seizures are not controlled by medication or when the seizures are caused by a lesion in a specific area of the brain [27].

2.2 Electroencephalography

Electroencephalography (EEG) is a non-invasive method for measuring the electrical activity of the brain. It involves the recording of electrical signals from the scalp using electrodes placed on the head Figure 2.1 [25]. EEG is widely used in the diagnosis and management of neurological disorders, particularly epilepsy, as it allows the detection of abnormal electrical activity in the brain [28].

EEG signals are composed of different frequency bands [25]. Each frequency band is associated with different brain states, such as sleep, wakefulness and attention [25].

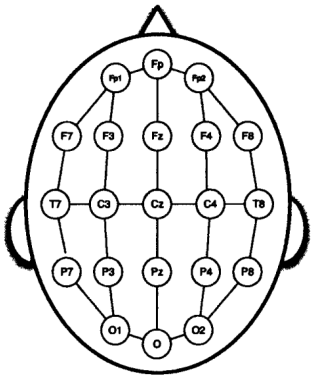


Figure 2.1: Scalp electrodes placement in human head

2.3 Short-time Fourier transform

By considering previous works as referents, it is highly recommended to work with frequency domain data since it can provide a more detailed view of the EEG signal by breaking it down into its different frequency components, which can reveal important information about brain activity related to seizures.

Previous studies mentioned that using only information from the frequency domain will lose of spatial information, to prevent this from happening, I am dividing the entire time domain data into small epochs of 2 seconds with 1 second of overlapping, this method can also be called short-time Fourier transform (STFT).

The short-time Fourier transform (STFT) is used to analyze how the frequency content of a non-stationary signal changes over time. the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment. Mathematically, this is written as: Equation (2.1)

$$\text{STFT}\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}dt \tag{2.1}$$

2.4 Standarization

Standardization eq. (2.4) is a method of feature scaling in which data values are rescaled to fit the distribution between 0 and 1 using mean eq. (2.2) and standard deviation eq. (2.3) as the base to find specific values.

$$\mu = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{2.2}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{N}} \tag{2.3}$$

$$Z = \frac{x - \mu}{\sigma} \tag{2.4}$$

Standardization of data is an important preprocessing step before training a machine learning neural network for several reasons:

- Scale invariance: Neural networks assume that the data is in a standard scale. Without standardization, it may be sensitive to the scale of the data, which can affect the performance of the model.
- Faster convergence: Standardizing the data can help speed up the convergence of the optimization algorithm used to train the model. This is because the optimization algorithm may be able to converge faster when the data is on a standard scale.
- Improved performance: Standardizing the data can also improve the performance of the model it can help the model better learn the underlying patterns in the data.

Since I've all features stored in Pandas dataframe, the standardization can be applied with executing instruction Listing 1 from appendix.

2.5 Neural Network

Neural Network is a type of machine-learning model that is inspired by the structure and function of the human brain. It is composed of layers of interconnected nodes, also known as artificial neurons. These neurons are organized in layers, and they process and transmit information by means of weighted connections between them. Neural networks can be trained to perform a wide range of tasks, such as image classification, speech recognition, and natural language processing.

Activation function

An activation function is a mathematical function that is applied to the output of a neuron in an artificial neural network. The purpose of the activation function is to introduce non-linearity into the output of the neuron, allowing the network to learn complex, non-linear relationships between inputs and outputs.

There are several commonly used activation functions, such as sigmoid, ReLU, and tanh. For our model, we are choosing Leaky ReLU, which is a variant of the ReLU activation function that addresses the problem of ”dying ReLU” which occurs when a neuron’s output is zero, and the gradient of the activation function is also zero, so the weights will not be updated, and the neuron will stop working. Leaky ReLU solves this problem by introducing a small positive slope (or ”leak”) to the negative part of the ReLU function, which allows the gradients to flow through the neuron even when the output is close to zero.

$$R(z) = \left\{ \begin{array}{ll} z & z > 0 \\ \alpha z & z <= 0 \end{array} \right\} \tag{2.5}$$

The reason to choose Leaky ReLU over other activation functions is that it has been found to be more effective in training deep neural networks. In comparison to other activation functions like ReLU, it has been found to produce better results in terms of the convergence of the training process and the final accuracy of the model. The α from the formula() is the hyperparameter that controls the angle of the negative slope, since we don’t exactly need any specific behaviour to the activation function, well keep the default value of 0.01.

Loss function

A loss function, also known as a cost function, is a mathematical function that measures the difference between the predicted output(p) and the true output(y) of a neural network. The goal of training a neural network is to minimize the value of the loss function.

There are several commonly used loss functions, such as mean squared error (MSE), mean absolute error (MAE) and cross-entropy loss.

Cross-entropy loss eq. (2.6) is a popular loss function for classification problems, it is often used for problems where the goal is to predict a probability distribution over a set of discrete classes. In the case of seizure prediction, where the goal is to classify an EEG signal as either a seizure or non-seizure, cross-entropy loss is a suitable choice.

$$-\left(y \log(p) + (1-y) \log(1-p)\right) \tag{2.6}$$

Cross-entropy loss is a measure of the dissimilarity between the predicted probability distribution and the true distribution. It penalizes the model more for predictions that are further away from the true labels. It also has the advantage of providing a probabilistic interpretation of the output, which will be discussed in **??** about how to use this value as a level of confidence.

Optimizers

An optimizer is an algorithm used to adjust the parameters of a neural network during training, in order to minimize the cost function. The cost function represents the difference between the predicted output of the network and the true output. The goal of training a neural network is to find the set of parameters that minimize this cost function.

Adam eq. (2.7) is a popular optimization algorithm that combines the advantages of two other optimization algorithms, Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) [29]. It uses a combination of gradient information and moving averages of the gradient and squared gradient to adapt the learning rate on a per-parameter basis.

$$m_n = E\left[X^n\right] \tag{2.7}$$

Adam is an adaptive optimization algorithm, which means that it modifies the learning rate during training. It uses a combination of gradient information and historical gradient information to adjust the learning rate for each parameter. This allows Adam to converge faster and with more stable results, compared to traditional optimization algorithms such as stochastic gradient descent (SGD) [29].

2.5.1 Feed Forward Neural Network

Feed Forward Neural Network consists of layers of interconnected artificial neurons, or nodes, which process and transmit information through the network.

The information flows in one direction, from the input layer through one or more hidden layers and to the output layer, without looping back fig. 2.2. This architecture is called feed-forward because the information flows in one direction, from the input layer to the output layer without looping back [30].

The input layer receives the input data, and the output layer produces the predictions. The hidden layers process the information by applying mathematical operations (also called activation functions) Section 2.5 to the input data and passing the results to the next layer. The number of hidden layers, and the number of neurons in each layer, can be adjusted to optimize the performance of the network.

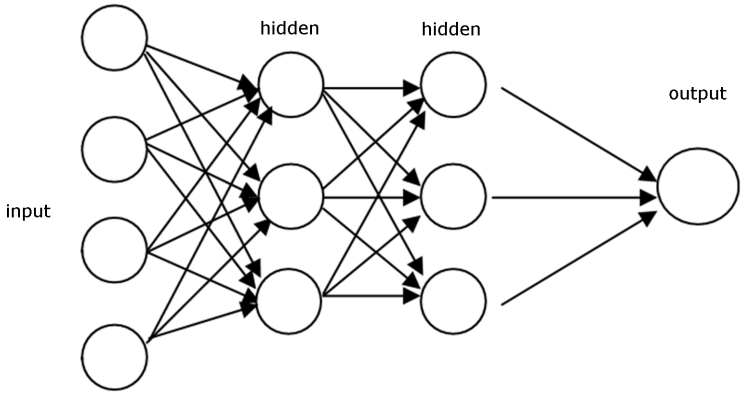


Figure 2.2: Data loaded from input goes throw several hidden layers before reaching the output

2.5.2 Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of neural network algorithm that is particularly well-suited for image and signal processing tasks. CNNs are inspired by the structure of the human visual cortex, which is composed of small, locally connected regions called receptive fields [30]. A CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layers are responsible for learning local patterns in the input data by applying a set of learnable filters to the input. These filters slide across the input, computing dot products between the filter weights and the input at each position. These dot products are then passed through a non-linear activation function, such as Leaky ReLU Section 2.5, to introduce non-linearity into the model.

Pooling layers are used to down-sample the feature maps outputted by the convolutional layers. This helps to reduce the spatial size of the feature maps, while also retaining the most important features [31].

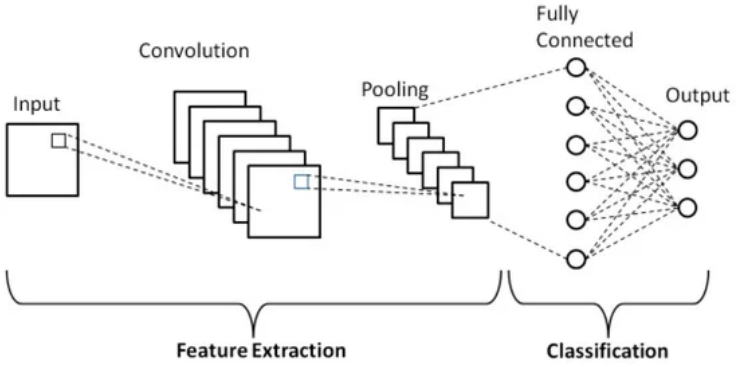


Figure 2.3: Data loaded from input goes throw convolution and pooling layers before reaching the output

2.6 Performance metrics

In the context of classification problems, three commonly used performance metrics are accuracy, specificity, and sensitivity. Variables needed to calculate each metric are true positive (TP), true negative (TN), false positive (FP) and false negative (FN).

- **Accuracy** refers to the proportion of correctly classified instances in the entire dataset. Mathematically, it is defined as the ratio of the number of true positive (TP) and true negative (TN) predictions to the total number of observations.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TN} + \text{FP} + \text{TP} + \text{FN}} \times 100 \tag{2.8}$$

- **Specificity** measures the proportion of correctly classified negative instances out of all negative instances. It is defined as the ratio of the number of true negative predictions to the total number of negative observations.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \times 100 \tag{2.9}$$

- **Sensitivity** measures the proportion of correctly classified positive instances out of all positive instances. It is defined as the ratio of the number of true positive predictions to the total number of positive observations.

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100 \tag{2.10}$$

3 Methodology

In this chapter, I will describe the steps taken to implement my seizure detection model, including data cleaning and preprocessing and the process of building and training machine learning models.

3.1 Dataset

CHB-MIT Scalp EEG Database Table 3.1, compiled by Children's Hospital Boston, contains EEG recordings from child patients suffering from intractable seizures. Subjects were observed for many days after discontinuing anti-seizure medication to define their seizures. This dataset is really clean and I am using this dataset to train and test my seizure detection model.

In this database, all EEG recordings are made at a sample rate of 256Hz, and each case has many files including separate sessions. Each recording session contains numerous channels Figure 2.1; however, not all of the channels from each instance are equal, which can lead to a significant inconsistency problem; hence, channel selection and data filtering are necessary before using this data to train each model.

Case	Gender	Age
chb01	F	11
chb02	M	11
chb03	F	14
chb04	M	22
chb05	F	7
chb06	F	1.5
chb07	F	14.5
chb08	M	3.5
chb09	F	10
chb10	M	3
chb11	F	12
chb12	F	2
chb13	F	3
chb14	F	9
chb15	M	16
chb16	F	7
chb17	F	12
chb18	F	18
chb19	F	19
chb20	F	6
chb21	F	13
chb22	F	9
chb23	F	6

Table 3.1: Summary of CHB-MIT Scalp database content CHB-MIT Scalp EEG Database

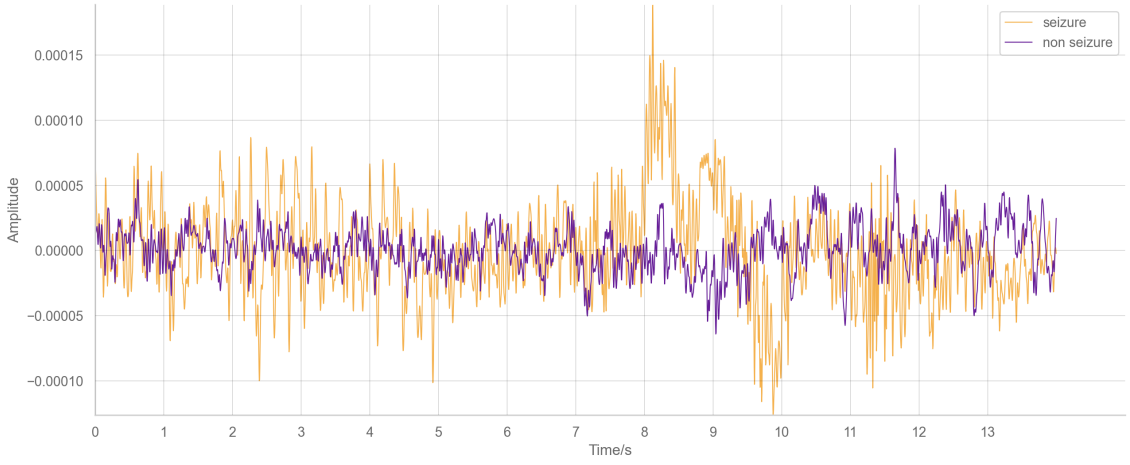


Figure 3.1: Sample of seizure and non-seizure in the time domain from CHB-MIT Scalp database

3.2 Preprocessing

Preprocessing is an important step before training a model to predict seizures because it helps to improve the quality and reliability of the data. This can help to improve the accuracy and performance of your model while also ensuring that the model is able to generalize to new, unseen cases. Additionally, preprocessing the data can also reduce the data's dimensionality, which can improve the computational efficiency of training the model. A flow chart from my source code Listing 2 in appendix, Listing 3 in the following flow chart Figure 3.2.

3.2.1 Data inspection

CHB-MIT Scalp is encoded as European Data Format (EDF) which is a standard file format designed for the exchange and storage of medical time series. In order to extract all the encoded information, MNE library support is needed in order to provide the right toolkit to read the data and integrate it into python.

MNE is an open-source Python package for exploring, visualizing, and analyzing human neurophysiology. This library is huge but only really specific methods are being used Listing 2 (in appendix) in order to extract our data.

Mne also provides support for inspecting data from different channels, however, seizure annotations aren't located in the raw data, manual seizure annotations are being assigned in the raw data in order to visualize seizures in the EEG visualization Listing 3 from appendix.

3.2.2 Features selection

Channel selection

Cases from CHB-MIT Scalp are not recorded in the same way, since some of their recordings provide more channels than others. The goal is to keep the maximum amount of patients in order to generalize better the model, so instead of discarding patients, I am discarding channels that aren't available in every case. Getting the intersection of channels from all cases, channels to be included in the prediction will be the following 17: "P8-O2", "C4-P4", "FP1-F3", "FP2-F8", "CZ-PZ", "FP1-F7", "T7-P7", "C3-P3", "FP2-F4", "P4-O2", "F8-T8", "F7-T7", "F3-C3", "FZ-CZ", "P3-O1", "P7-O1", "F4-C4". A clear display of the channel selection can be seen in Figure 1.

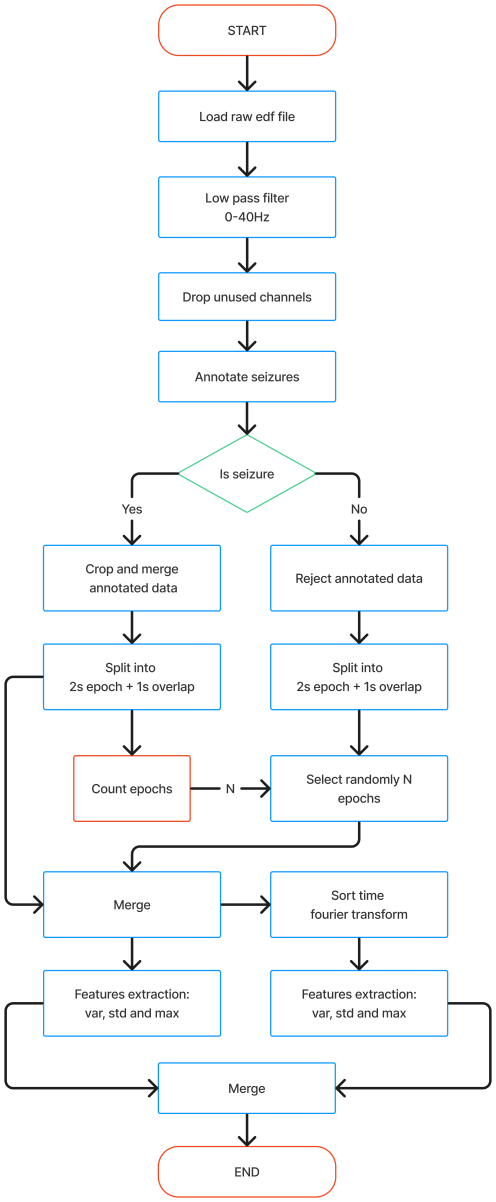


Figure 3.2: flow chart of preprocessing's code implementation

Dealing with unbalanced data

Unbalanced data is a frequent problem in machine learning, especially when working with EEG datasets. Oversampling the minority class or undersampling the majority class is a typical strategy for dealing with uneven data. In this scenario, there are many more interictal intervals than ictal intervals. The strategy taken to produce balanced data is to have the same number of interictal and ictal intervals Figure 3.2. This may be accomplished by randomly picking segments of interictal periods to match the same amount of ictal duration.

Statistical metrics

Common statistical metrics are being used to describe the attributes of a dataset. The advantage of using statistical measures instead of raw data is that it reduces the dimensionality of the data, which can increase the computing efficiency of your model.

In order to choose the right metrics, it's important to only choose the feature with a higher correlation with the target. Table of correlation Figure 3.3 is a table showing the correlation coefficients between multiple variables. It can be used to identify relationships between variables: By looking at the correlation coefficients, variables that are positively or negatively correlated with each other can be identified. This can help understand the relationships between different variables and how they may affect each other. By looking at the correlation coefficients, highly correlated with the target variables can be identified, the main point of doing this is to identify and select the most relevant features for the model.

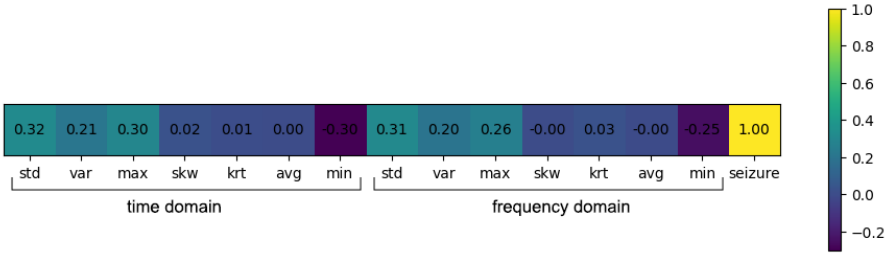


Figure 3.3: Seizure correlation with features as a heat map

A higher correlation means more representative in order to classifying each epoch. Regarding the image, features like **maximum**, **standard deviation** and **variation** seem more relevant than **skewness**, **kurtosis**, **minimum** and **average**.

To sum up, I want to obtain features from each epoch before and after applying STFT Section 2.3 (which means I am using both time and frequency domain) to each epoch of the data. By combining 3 of the most relevant statistical metrics and the previously mentioned.

3.2.3 Data cleaning

Preparing data for training

In order to prepare the dataset for training, everything is stored in Pandas dataframe. Pandas is a Python library for data manipulation and analysis. It provides data structures and functions for working with structured data, such as data tables and time series data. The main reason to use pandas for machine learning is that it is easy to clean, preprocess, and transform datasets, which is often a critical step in the machine learning pipeline.

Train test split

Training and test sets are used to evaluate the performance of a machine-learning model. The idea is to split the available data into two sets: one set is used to train the model and the other set is used to evaluate the performance of the trained model.

The training set is used to fit the model and the test set is used to evaluate the performance of the model. By using a separate test set, I can get an unbiased estimate of the model's performance on unseen data. This is important because it allows us to understand how well the model is likely to perform on new, unseen data, which is critical for real-world applications.

Since our data works with cases and each case is an independent patient. These patients are randomly selected into the train or test sets, to better generalize the result without compromising the size of the training set. 21 patients are being chosen for training the model and 3 are being chosen for the testing and validation see Figure 3.4.

Figure 3.4: Representation of train and test balance. Red means test set and black means train set

3.3 1-dimensional data model training

In this section, I will use several machine-learning approaches to train a model with the test set and evaluate its correctness. There are 102 different features and 1 binary target for each named epoch, which can be true or false depending on whether the epoch target is a seizure or not.

3.3.1 Feed forward neural network model architecture

For this model, I am using **PyTorch**. **PyTorch** is an open-source machine-learning library for Python. It contains a variety of built-in tools and functions to accelerate the process of building a neural network. The entire code of the Neural network is written with the following code Listing 4 from appendix and the following graph Figure 3.5 notice that it uses the activation function mentioned in section 2.5.

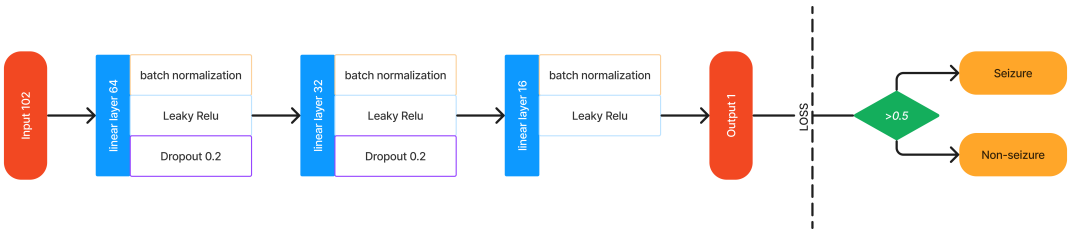


Figure 3.5: FFNN structure from Listing 4 from appendix

3.3.2 Feed forward neural network model training

In order to train my **FFNN** model, I first separated the training set into batches of 50 samples each. This is a common practice when training neural networks as it allows the model to learn from the data in smaller chunks, which can improve the stability and speed of the training process.

Next, I applied **cross-entropy** loss as the cost function Section 2.5 and used the **Adam** optimizer to adjust the weights of the network during training. Adam is an adaptive optimization algorithm that uses a combination of gradient information and historical gradient information to adjust the learning rate on a per-parameter basis. It is well suited for training neural networks and is known to converge faster and more stable than other optimization algorithms.

I set the learning rate to 0.0001 which is a common value for the **Adam** optimizer. This means that the optimizer will make small updates to the weights at each step. This value was chosen after trying different learning rates and evaluating the results.

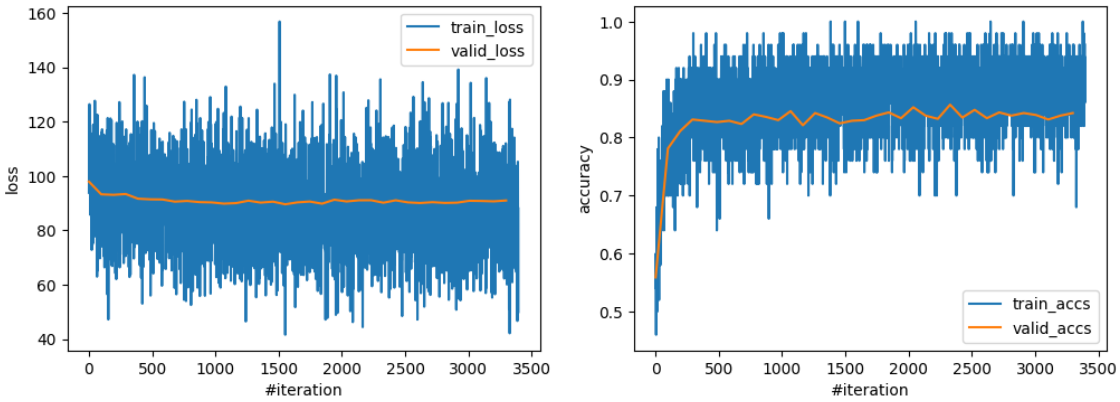


Figure 3.6: Neural network model training performance, left: reduction of the loss function from each iteration, right: increase of the accuracy from each iteration. Lower loss function and higher accuracy mean better training performance, but always following values from the valid set to avoid over-fitting.

After training the model on the training set, I evaluated its performance by measuring its accuracy on the test set. The final accuracy of the model was 83.778%, which means that the model was able to correctly classify 83.778% of the samples in the test set. This accuracy can be considered as a good result, considering the trade-off between specificity. However, we can clearly see in Figure 3.6, there is still a huge gap for the training accuracy to reach 100%. So most probably there is still room for improvement.

Some possible solution to identify the problem is by changing the model architecture by adding more layers or changing their size. However, this can also lead to an over-fitting problem, this happens because the model has learned the noise in the training data, rather than the underlying pattern, and as a result, it becomes highly sensitive to small variations in the input data. So as an alternative, it is possible to check the quality of the model by comparing it with others. If other models cannot outperform the **FFNN** means the real bottleneck is not the architecture, then we can look into improving the dataset again.

3.3.3 Comparing with other Machine learning models

I am using three different machine learning tools, Support Vector Machine (**SVM**), K Nearest Neighbours (**KNN**) and Random Forest (**RF**) to compare their performance with the **FFNN** model using the same dataset.

SVM is a linear model that finds the best boundary between different classes, **KNN** is a non-parametric method that finds the k-neighbours of each sample and assigns the class that is most common among them. **RF** is an ensemble learning method that constructs multiple decision trees and combines their predictions to make a final prediction. They are all supervised

By using these different machine learning tools, I will be able to compare their performance with the **FFNN** model and determine which one is the most accurate and suitable for the seizure detection task. By doing this, I can conclude whether the limitation of the **FFNN** model is caused by the architecture or the dataset and if there is any room for improvement.

This specific code can be found in Listing 13, Listing 5 and Listing 6 from appendix. After evaluating their performance with accuracy, specificity and sensitivity Section 2.6 the following table is generated:

Model	**Accuracy**	**Specificity**	**Sensitivity**
Support Vector Machine (**SVM**)	87.554%	87.339%	87.768%
K Nearest Neighbour (**KNN**)	87.446%	87.339%	87.554%
Random Forest (**RF**)	87.554%	91.631%	83.476%
Feed Forward Neural Network (**FFNN**)	83.778%	86.683%	81.314%

Table 3.2: Metrics table of testing set comparison from different machine learning models

According to Table 3.2, most of the models have similar scores in each metric, which means our model architecture is not particularly bad. We can keep trying other machine-learning algorithms like naive Bayes, Logistic regression or decision trees. However, considering that scores from accuracy in all of the tested models can't reach over >88%, probably is not the problem of the model. In this case, we may assume that the bottleneck of the training resides in the dataset that is not generating enough relevant features to detect seizures with precision.

3.4 2-dimensional data model training

According to Section 3.3, our main problem is to achieve higher accuracy on the model relays on which features we selected before the training. Using statistical metrics as features was a wise choice in terms of performance, however, it may be not enough. This new approach consists of generating 2D data and taking advantage of Convolutional Neural Network Section 2.5.2 to provide the extra features missing for the **FFNN** to achieve higher accuracy.

3.4.1 2-dimensional data extraction

Since the entire frequency domain is segmented, boundaries have to be set, *Journal of Clinical Neurophysiology* [32] mentioned that frequencies above 40 Hz are poorly visualized on conventional EEG scalp recordings. We are also considering this idea and applying a low pass filter in the frequency domain before generating the 2-dimensional dataset.

Low pass filtering

Low-pass filtering is a technique used to remove high-frequency components from a signal while preserving the low-frequency components. It is often used in signal processing to remove unwanted noise or to isolate specific frequency bands of interest.

In the case of EEG signals, low pass filtering is beneficial for seizure detection because seizures are generally characterized by low-frequency changes in the EEG signal. By setting a low pass filter at 40 Hz, you are removing the high-frequency components of the signal that are not relevant to the seizure detection task and focusing on the low-frequency components that are more likely to contain information about seizures. This can help to improve the performance of the seizure detection model by reducing noise and highlighting the relevant features of the signal.

Heat map generation

A heat map is a graphical representation of data where individual values are represented as colours. It is used to visualize the distribution of a dataset, highlighting areas of high and low density. The colours in a heat map are typically chosen to represent different levels of a variable, such as a temperature or density, with warm colours indicating high values and cool colours indicating low values.

Since we have several channels for each **EEG** epoch, after applying **STFT**, an image of a heat map can be generated. To simplify the amount of data from the heat map, I am separating the frequency domain into 41 chunks calculated by computing the average between segments of 1Hz (E.G. 0Hz-0.99Hz, 1Hz-1.99Hz, 2Hz-2.99Hz...) Figure 3.7. A good reason to use this technique is that it can remove any noise and make the model cleaner and smoother in each chunk.

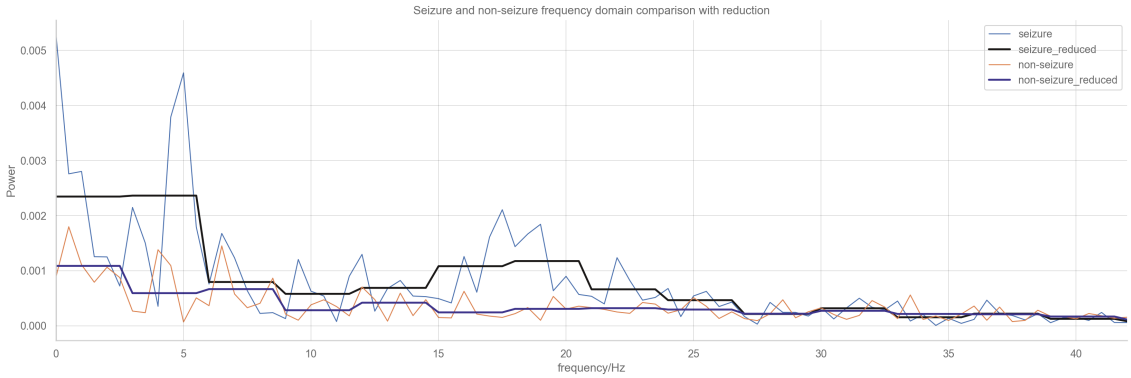


Figure 3.7: Sample of STFT to a seizure and non-seizure reduction after applying reduction

3.4.2 convolutional model architecture

Data extracted from reduced STFT and combined channels can be used as input for a CNN, neural network. The proposed model architecture Figure 3.8 is a combination of the architecture from Section 3.3.1 but adding output extracted from a CNN.

Notice that in Figure 3.8 we have a flatten layer, this does the job of transforming 2D data back into a vector of 1D data. Activation function Section 2.5, loss function Section 2.5 remains the same as in Section 3.3.1.

3.4.3 convolutional model training

Since this model is deeper than the previous one, training this model will definitely be harder. I am using also the technique of separating the training set into batches of 50, the learning rate of 0.001 and Adam optimizer like Section 3.3.2 since the purpose of this model is the same as the previous one.

After looking into the performance of this model, an outstanding learning performance is being shown. Considering the valid set (orange line) is not inside the training set, it is surprisingly good to see how it is still reaching almost the 100%of accuracy.

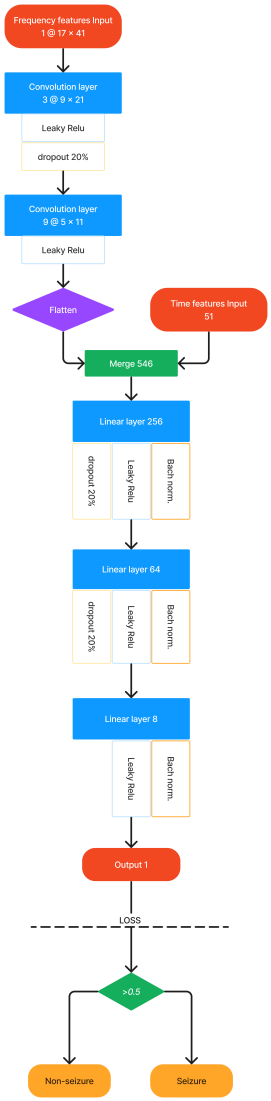


Figure 3.8: FFNN structure from Listing 8 from appendix

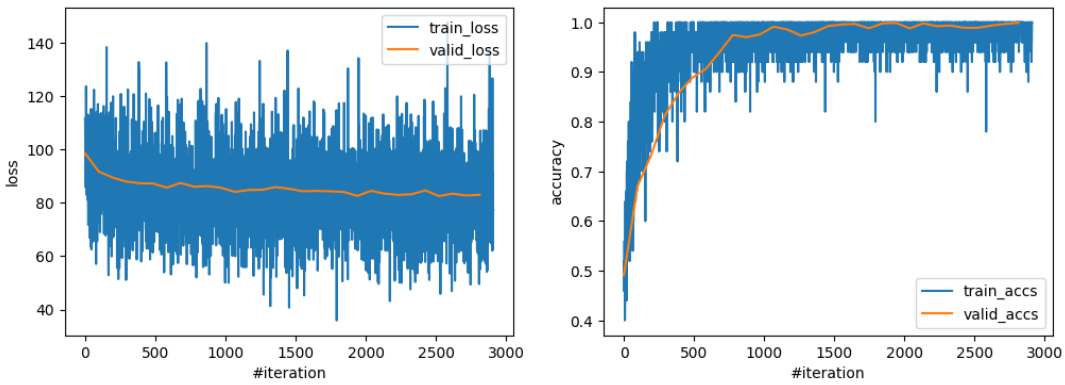


Figure 3.9: Neural network model training performance from CNN+FFNN

4 Results

I this chapter I will be discussing how I evaluated the performance of the model and analyzed the results. The evaluation process should also consider the trade-off between specificity and sensitivity and how I can balance it to provide the best results.

4.1 2-dimensional data visulization

Image Figure 4.1 is generated by combining 17 channels after applying **STFT** to each epoch. This is an example of input loaded in the convolutional neural network where each epoch can generate matrices of 17 channels x 41 chunks.

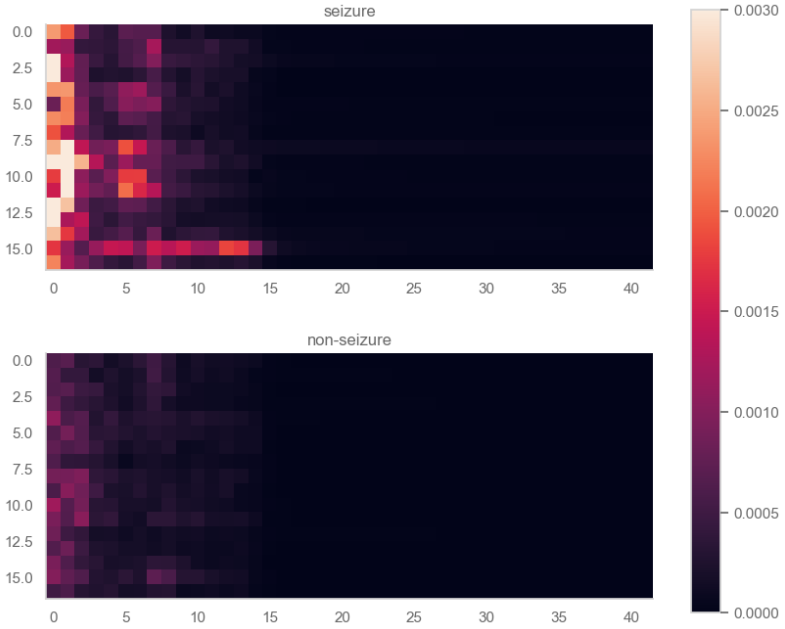


Figure 4.1: Heat map generated after STFT and reduction

We can clearly differentiate seizure and non-seizure from Figure 4.1. Since these images can be converted into the matrix and loaded into convolutions layers.

4.2 Models comparison

Each model was evaluated with the testing set, with is completely isolated from the training phase to avoid any kind of overfitting. The performance of each model was evaluated by comparing the seizure detection times to the reference times obtained by an expert with annotated data from CHB-MIT Scalp EEG Database, and the evaluation metrics used were sensitivity, specificity, and accuracy section 2.6. It is important to mention that the final model should not only have high accuracy but also a good balance between sensitivity and specificity, as it guarantees better and more accurate results.

Model	Accuracy	Specificity	Sensitivity
Support Vector Machine (**SVM**)	87.554%	87.339%	87.768%
K Nearest Neighbour (**KNN**)	87.446%	87.339%	87.554%
Random Forest (**RF**)	87.554%	91.631%	83.476%
Feed Forward Neural Network (**FFNN**)	83.778%	86.683%	81.314%
Convolutional + FFNN (**CNN+FFNN**)	**98.615%**	**98.737%**	**98.425%**

Table 4.1: Metrics table of testing set comparison from different machine learning models

From Table 4.1 we can see how the CNN+FNN model outperformed all the others, mainly due to its larger amount of data and deeper architecture of the neural network.

4.3 Testing with unbalanced data

Testing the model with unbalanced data can be used in order to simulate real-world scenarios. In practice, the distribution of seizure and non-seizure cases in EEG recordings is often unbalanced, with many more non-seizure cases than seizure cases. This is because seizures are relatively rare events, and it is difficult to collect a large number of seizure cases for training and testing the model.

In this case, MNE library from python is being used to visualize the results. In order to annotate seizures detected from the CNN+FNN model, an entire session is loaded into the model without any kind of processing as a real case experiment. Code can be found in Listing 16 from the appendix.

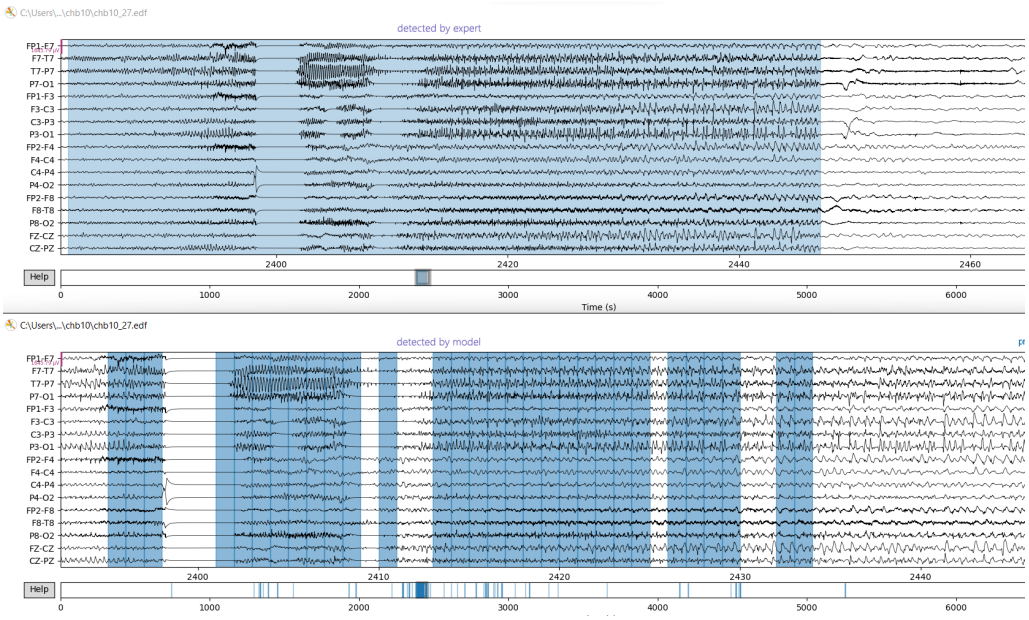


Figure 4.2: Comparison of seizures detected by an expert (above) and ML (below)

By looking into Figure 4.2, It's true that machine learning is detecting somehow the seizures, however, there are still a lot of noise. The model is detecting seizures but since it is trained with balanced data it is not working properly with unbalanced data. Figure 4.2 is detected by the CNN+FNN model, which means there is still a gap for this model to work with unbalanced data.

5 Discussion

In this project different machine-learning models have been built in order to detect seizures. In the near future is more likely to combine different methods and techniques to improve the seizure detection process. In this chapter some general ideas and comparisons.

It's known in the medical field that sensitivity is more important than specificity since we want to avoid missing any true positives and not relevant getting some false positives. Experts can do the minimum effort to reject false positives. Since we have been working with Cross Entropy Loss Section 2.5, a threshold can be used to evaluate the confidence, which means how confident is the model with its prediction. We can modify the threshold from 0 to 1 in order to assign seizures upon a boundary of its confidence. As a result in Figure 5.1, it is possible to see that specificity increases when decreasing the threshold and sensitivity increases when increases the threshold is. In future works, this could be an interesting approach to increase precision in medical diagnosis by combining experts' knowledge with artificial intelligence.

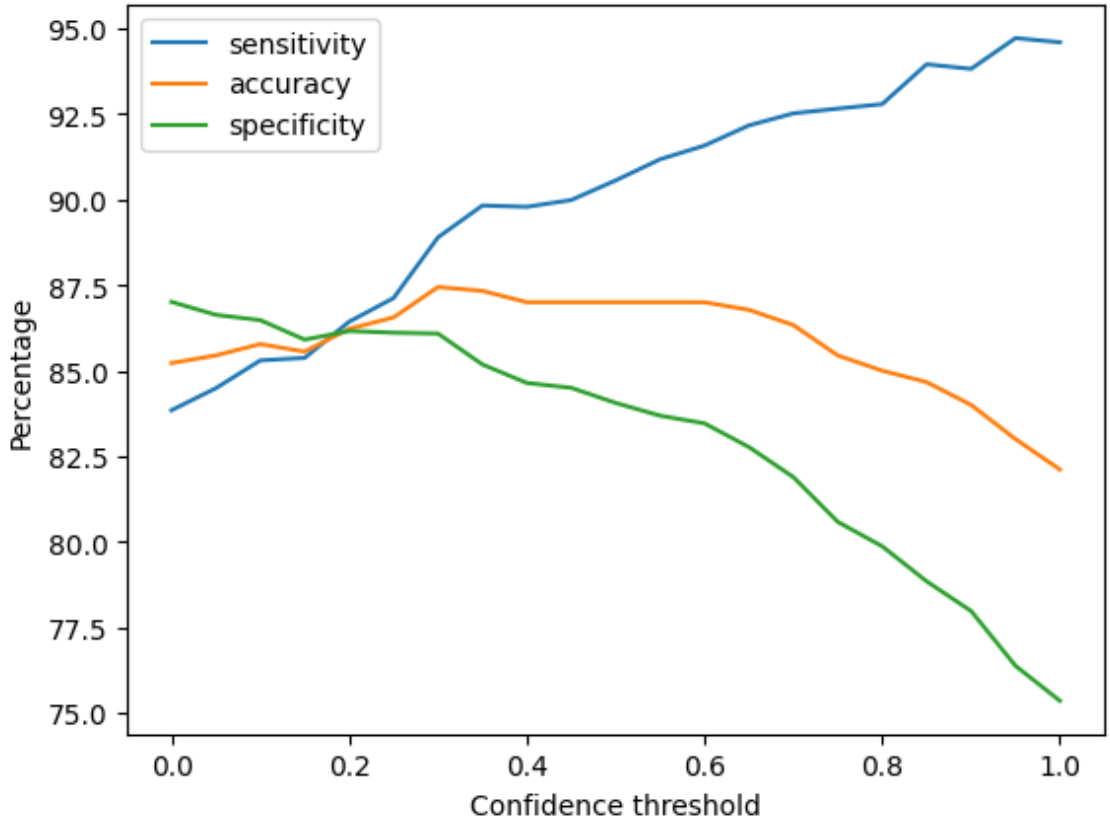


Figure 5.1: Plot generated calculating sensitivity, accuracy and specificity with a different threshold value.

While testing with unbalanced data this new dataset from Section 3.4 have been treated wrongly by computing the standardization of the seizures and non-seizures separately. This means the model is clearly over-fitted to the standardization, which shouldn't be done since new and unknown data should be standardized with the same values of standard deviation and average. Therefore the accuracy shown in Table 4.1 about the CNN+FNN model might be biased.

It is imperative to address this issue in future work and ensure that the data is properly standardized before training the model. This highlights the importance of proper data preprocessing and handling in machine learning, as even small errors in this step can lead to significant inaccuracies in the final model. Additionally, this issue also highlights the importance of testing models with new and unseen data to ensure that they generalize well to real-world scenarios.

One limitation of this project is that the EEG recordings used in this study were collected from a single hospital and may not represent the diverse range of EEG recordings from different environments. This can limit the generalizability of the results to other populations and settings. Additionally, the dataset used in this study only includes EEG recordings from children patients from 2 to 22 years Table 3.1, which means the model may not be suitable for adults.

Another limitation of this project is the size of the dataset. The dataset used in this study is relatively small, which can limit the ability of the model to generalize to new and unseen data. Furthermore, the dataset used in this study is composed of recordings from a single type of EEG device, and the results may not generalize to recordings from other devices.

A limitation of this study is the use of a single expert's annotation of seizures, which may limit the generalization of the results to other experts. Furthermore, the model is not able to predict seizures, it can only detect them in the provided dataset. Therefore, in future work, it would be beneficial to expand the dataset to include a more diverse range of patients, ages, and EEG recording environments, as well as to explore the possibility of using forecasting models for seizure prediction.

Additionally, this study uses a binary classification approach, where the goal is to classify a given EEG segment as either seizure or non-seizure. However, the reality of seizures is more complex and there are other types of seizures and seizure-like events that are not captured by this approach. Therefore, in future work, it would be beneficial to explore multi-class classification approaches that could capture this complexity.

6 Conclusion

In conclusion, this thesis presented a study on the use of machine learning techniques for the automated detection of seizures in EEG recordings. The goal of this research was to develop a model that could accurately detect seizures in EEG data and to evaluate its performance using a dataset of EEG recordings with seizures highlighted by an expert. The proposed model was a neural network that combined frequency features with a CNN and time features with an FFNN. The results of the evaluation showed that the proposed model was able to accurately detect seizures with an accuracy of 98.615%.

Furthermore, the study also compared the performance of different machine learning algorithms, such as Support Vector Machine (SVM), K-Nearest Neighbors (KNN) and Random Forest (RF) with the proposed model, and found that the proposed neural network model performed better. The study also explored the use of unbalanced data in the dataset, where the major problem resides in those models.

This study contributes to the ongoing research in the field of automated seizure detection by demonstrating the potential of using machine learning techniques for this task. The results of this research have the potential to support clinicians in the evaluation of seizures and to make the process more accurate and less time-consuming. However, there are still some limitations that need to be addressed in future work, such as increasing the sample size of the dataset and incorporating more advanced techniques to improve the performance of the model.

Bibliography

[1] Christian E Elger and Christian Hoppe. “Diagnostic challenges in epilepsy: seizure under-reporting and seizure detection”. In: *The Lancet Neurology* 17.3 (2018), pp. 279–288.

[2] PF Prior, RSM Virden, and DE Maynard. “An EEG device for monitoring seizure discharges”. In: *Epilepsia* 14.4 (1973), pp. 367–372.

[3] WRS Webber et al. “An approach to seizure detection using an artificial neural network (ANN)”. In: *Electroencephalography and clinical Neurophysiology* 98.4 (1996), pp. 250–272.

[4] GW Harding. “An automated seizure monitoring system for patients with indwelling recording electrodes”. In: *Electroencephalography and Clinical Neurophysiology* 86.6 (1993), pp. 428–437.

[5] Anthony M Murro et al. “Computerized seizure detection of complex partial seizures”. In: *Electroencephalography and Clinical Neurophysiology* 79.4 (1991), pp. 330–333.

[6] Abdulhamit Subasi et al. “Wavelet neural network classification of EEG signals by using AR model with MLE preprocessing”. In: *Neural Networks* 18.7 (2005), pp. 985–997.

[7] Alison A Dingle et al. “A multistage system to detect epileptiform activity in the EEG”. In: *IEEE Transactions on Biomedical Engineering* 40.12 (1993), pp. 1260–1268.

[8] Nihal Fatma Güler, Elif Derya Übeyli, and Inan Güler. “Recurrent neural networks employing Lyapunov exponents for EEG signals classification”. In: *Expert systems with applications* 29.3 (2005), pp. 506–514.

[9] N Kannathal et al. “Entropies for detection of epilepsy in EEG”. In: *Computer methods and programs in biomedicine* 80.3 (2005), pp. 187–194.

[10] Hao Qu and Jean Gotman. “A patient-specific algorithm for the detection of seizure onset in long-term EEG monitoring: possible use as a warning device”. In: *IEEE transactions on biomedical engineering* 44.2 (1997), pp. 115–122.

[11] Kemal Polat and Salih Güneş. “Classification of epileptiform EEG using a hybrid system based on decision tree classifier and fast Fourier transform”. In: *Applied Mathematics and Computation* 187.2 (2007), pp. 1017–1026.

[12] Andrew B Gardner et al. “One-Class Novelty Detection for Seizure Analysis from Intracranial EEG.” In: *Journal of Machine Learning Research* 7.6 (2006).

[13] Md Mursalin et al. “Automated epileptic seizure detection using improved correlation-based feature selection with random forest classifier”. In: *Neurocomputing* 241 (2017), pp. 204–214.

[14] Emilio Perucca. “An introduction to antiepileptic drugs”. In: *Epilepsia* 46 (2005), pp. 31–37.

[15] Heino H Lange et al. “Temporo-spatial patterns of pre-ictal spike activity in human temporal lobe epilepsy”. In: *Electroencephalography and clinical neurophysiology* 56.6 (1983), pp. 543–555.

[16] Steven J Schiff et al. “Brain chirps: spectrographic signatures of epileptic seizures”. In: *Clinical Neurophysiology* 111.6 (2000), pp. 953–958.

[17] Klaus Lehnertz et al. “Its possible use for interictal focus localization, seizure anticipation, and prevention: Nonlinear EEG analysis in epilepsy”. In: *Journal of Clinical Neurophysiology* 18.3 (2001), pp. 209–222.

[18] Amir B Geva and Dan H Kerem. “Forecasting generalized epileptic seizures from the EEG signal by wavelet analysis and dynamic unsupervised fuzzy clustering”. In: *IEEE Transactions on Biomedical Engineering* 45.10 (1998), pp. 1205–1216.

[19] W Allen Hauser, John F Annegers, and Leonard T Kurland. “Incidence of epilepsy and unprovoked seizures in Rochester, Minnesota: 1935–1984”. In: *Epilepsia* 34.3 (1993), pp. 453–458.

[20] Patrick Kwan and Martin J Brodie. “Early identification of refractory epilepsy”. In: *New England Journal of Medicine* 342.5 (2000), pp. 314–319.

[21] World Health Organization. *Epilepsy*. 2017. URL: https://www.who.int/news-room/fact-sheets/detail/epilepsy.

[22] Jerome Engel Jr. “A proposed diagnostic scheme for people with epileptic seizures and with epilepsy: report of the ILAE Task Force on Classification and Terminology”. In: *Epilepsia* 42.6 (2001), pp. 796–803.

[23] P Kwan and JW Sander. “The natural history of epilepsy: an epidemiological view”. In: *Journal of Neurology, Neurosurgery & Psychiatry* 75.10 (2004), pp. 1376–1381.

[24] Robert S Fisher et al. “ILAE official report: a practical clinical definition of epilepsy”. In: *Epilepsia* 55.4 (2014), pp. 475–482.

[25] Ernst Niedermeyer and FH Lopes da Silva. *Electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins, 2005.

[26] Matthew C Walker and Simon Shorvon. “Emergency treatment of seizures and status epilepticus”. In: *The treatment of epilepsy* (2015), pp. 221–244.

[27] Samuel Wiebe et al. “A randomized, controlled trial of surgery for temporal-lobe epilepsy”. In: *New England Journal of Medicine* 345.5 (2001), pp. 311–318.

[28] Manouchehr Javidan. “Electroencephalography in mesial temporal lobe epilepsy: a review”. In: *Epilepsy research and treatment* 2012 (2012).

[29] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[30] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[31] Dominik Scherer, Andreas Müller, and Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *International conference on artificial neural networks*. Springer. 2010, pp. 92–101.

[32] Qirui Ren et al. “A 13 μW Analog Front-End with RRAM-Based Lowpass FIR Filter for EEG Signal Detection”. In: *Sensors* 22.16 (2022), p. 6096.

Appendix

Codes

```
# df is Pandas.dataframe with all stored features
df = (df - df.mean()) / df.std()
```

Listing 1: Standardization of dataframe

```
import mne
#it is important to activate the flag preload, otherwise, time series data won
    't be stored in the environment
edf_data = mne.io.read_raw_edf("filePath.edf", preload=True)
```

Listing 2: mne library support to extract the data

```
# seizure_start is a list of seizure starting time
# duration is a list of their duration
ictal_information = mne.Annotations(
        onset=seizure_start,
        duration=seizure_duration,
        description="bad"
        )
edf_data.set_annotations(ictal_information)
# reject_by_annotation will remove any segment with an annotation of "bad"
non_seizure = mne.make_fixed_length_epochs(
        edf_data,
        seizure_duration,
        reject_by_annotation=True)
seizure = mne.concatenate_raws(edf_data.crop_by_annotations())
```

Listing 3: annotating and splitting seizure from non-seizure

```
    class FFNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(FFNN, self).__init__()
        # define layers
        self.lin1 = nn.Linear(input_dim, 64)
        self.norm1 = nn.BatchNorm1d(64)
        self.lrelu = nn.LeakyReLU()
        self.drop = nn.Dropout(0.2)

        self.lin2 = nn.Linear(64, 32)
        self.norm2 = nn.BatchNorm1d(32)
        self.lrelu = nn.LeakyReLU()

        self.lin3 = nn.Linear(32, 16)
        self.norm3 = nn.BatchNorm1d(16)
        self.lout = nn.Linear(16, output_dim)

    def forward(self, x):
        y_pred = self.lin1(x)
        y_pred = self.norm1(y_pred)
        y_pred = self.lrelu(y_pred)
        y_pred = self.drop(y_pred)

        y_pred = self.lin2(y_pred)
        y_pred = self.norm2(y_pred)
        y_pred = self.lrelu(y_pred)
        y_pred = self.drop(y_pred)

        y_pred = self.lin3(y_pred)
        y_pred = self.norm3(y_pred)
```

```
        y_pred = self.lrelu(y_pred)

        y_pred = self.lout(y_pred)
        return y_pred.squeeze()
```

Listing 4: FFNN model architecture code with Pytorch

```
from sklearn.svm import SVC
svm=SVC(kernel='linear')
svm.fit(X_train.values, y_train)
y_pred = svm.predict(X_test.values)
```

Listing 5: SVM training and prediction with sklearn

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100)
rf.fit(X_train.values, y_train)
y_pred = rf.predict(X_test.values)
```

Listing 6: RF training and prediction with sklearn

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=60, n_jobs=-1)
knn.fit(X_train.values, y_train)
y_pred = knn.predict(X_test.values)
```

Listing 7: KNN training and prediction with sklearn

```
input_size = X_train.shape[1] - 1 + 495
output_size = 1

class FFNN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(FFNN, self).__init__()
        # define layers
        self.conv1 = nn.Conv2d(1, 3, 3, padding=1, stride=2)
        self.relu = nn.ReLU()
        # 3x9x5
        self.conv2 = nn.Conv2d(3, 9, 3, padding=1, stride=2)
        # 9x5x3
        self.lin1 = nn.Linear(input_dim, 256)
        self.norm1 = nn.BatchNorm1d(256)
        self.lrelu = nn.LeakyReLU()
        self.drop = nn.Dropout(0.2)

        self.lin2 = nn.Linear(256, 64)
        self.norm2 = nn.BatchNorm1d(64)
        self.lrelu = nn.LeakyReLU()

        self.lin3 = nn.Linear(64, 8)
        self.norm3 = nn.BatchNorm1d(8)
        self.lout = nn.Linear(8, output_dim)

    def forward(self, time, freq):
        freq = self.conv1(freq)
        freq = self.relu(freq)
        freq = self.drop(freq)

        freq = self.conv2(freq)
        freq = self.relu(freq)

```

```
            freq = freq.contiguous().view(freq.shape[0],freq.shape[1]*freq.
                shape[2]*freq.shape[3])
            features = torch.cat((freq, time), dim=1)

            y_pred = self.lin1(features)
            y_pred = self.norm1(y_pred)
            y_pred = self.lrelu(y_pred)
            y_pred = self.drop(y_pred)

            y_pred = self.lin2(y_pred)
            y_pred = self.norm2(y_pred)
            y_pred = self.lrelu(y_pred)
            y_pred = self.drop(y_pred)

            y_pred = self.lin3(y_pred)
            y_pred = self.norm3(y_pred)
            y_pred = self.lrelu(y_pred)

            y_pred = self.lout(y_pred)
            return y_pred.squeeze()

    model = FFNN(input_size, output_size)
```

Listing 8: CNN + FFNN model architecture code with Pytorch

```
import numpy as np
import pandas as pd
from scipy.fft import rfft, rfftfreq
from scipy.stats import skew, kurtosis
import csv, mne, math, pickle, json
from IPython.display import clear_output

# constants
EPOCH_DURATION = 2
OVERLAP_DURATION = 1

# randomly selected patients to be train or test
train = [
    "chb01",
    "chb02",
    "chb03",
    "chb04",
    "chb05",
    "chb06",
    "chb10",
    "chb11",
    "chb13",
    "chb14",
    "chb15",
    "chb16",
    "chb17",
    "chb18",
    "chb19",
    "chb20",
    "chb21",
    "chb22",
    "chb23",
    "chb24",
]

test = ["chb07", "chb08", "chb09"]

```

```python
# drive path to the dataset (should change it if you want to try the code)
drive_path = "C:/Users/Eugene Chen/Desktop/UNI/Project/Data/"
seizure_pointers = pd.read_excel(drive_path + "seizure data.xlsx", index_col
    =0)
seizure_pointers["index"] = (
    seizure_pointers["seizure_file"]
    + " "
    + seizure_pointers["seizure_number"].astype(str)
)

# channels to be selected
seizure_pointers = seizure_pointers.set_index("index")
channels = [
    "P8-O2",
    "C4-P4",
    "FP1-F3",
    "FP2-F8",
    "CZ-PZ",
    "FP1-F7",
    "T7-P7",
    "C3-P3",
    "FP2-F4",
    "P4-O2",
    "F8-T8",
    "F7-T7",
    "F3-C3",
    "FZ-CZ",
    "P3-O1",
    "P7-O1",
    "F4-C4",
]

# labels to identify the features of output dataset
label_std = [n + "-std" for n in channels]
label_var = [n + "-var" for n in channels]
label_max = [n + "-max" for n in channels]
label_std_rfft = [n + "-std_rfft" for n in channels]
label_var_rfft = [n + "-var_rfft" for n in channels]
label_max_rfft = [n + "-max_rfft" for n in channels]


# only for a loading bar can be ignored
DONE = 1
TOTAL = 0
for t in train + test:
    TOTAL += seizure_pointers[seizure_pointers.case == t].shape[0]

# loading bar
def progress_bar():
    percent = 100 * (DONE / float(TOTAL))
    bar = " " * int(percent) + "-" * (100 - int(percent))
    print(f"\r|{bar}| {percent: .2f}%", end="\r")

def diff(lst1, lst2):
    '''
    input:
    lst1 -> list
    lst2 -> list
    output:
    return intersection of lst1 with lst2
    '''
    return list(set(lst1) - set(lst2))
```

```
99
100 def get_reduced_freq(target, batch_size, sampling_rate):
101     '''
102     input:
103     target -> sample to apply the tranformation
104     batch_size -> lenght of the signal reduction (int)
105     sampling_rate -> sampling rate of the signal (int)
106     output:
107     return target applied reduction and fourier tranformation
108     '''
109     result = []
110     for channel in target:
111         layer = []
112         batch=[]
113         target_ft = abs(rfft(channel))
114         target_ft = [ x.real for x in target_ft]
115         for i in target_ft[0:(sampling_rate*40)+2]:
116             batch.append(i)
117             if len(batch)==batch_size:
118                 batch_mean = sum(batch)/batch_size
119                 layer.append(batch_mean)
120                 batch=[]
121         result.append(layer)
122     return result
123
124 def get_packed_stft(target, batch_size, sampling_rate, i):
125     '''
126     input:
127     target -> sample to apply the tranformation
128     batch_size -> lenght of the signal reduction (int)
129     sampling_rate -> sampling rate of the signal (int)
130     i -> index starting number
131     output:
132     return pandas series with the label 'stft'
133     starting with index i ready to be concatenated to a pandas dataframe
134     '''
135     fft_X = {}
136     for x in [get_reduced_freq(x, batch_size, sampling_rate) for x in target]:
137         fft_X[i] = x
138         i+=1
139     return pd.Series(fft_X).rename('stft')
140
141 def standarize(key,vector):
142     '''
143     input:
144     key -> feature category to standarize (string)
145     vector -> numpy array to apply standarization
146     output:
147     return standarized numpy array
148     '''
149     # there need to be a json file with standarization values
150     with open('standarization_values.json') as json_file:
151         standarization = json.load(json_file)
152     metrics = standarization[key]
153     return (vector - metrics["mean"]) / metrics["std"]
154
155 def create_df_file_from_patient(patient, file):
156     '''
157     input:
158     patient -> dictionary with patient information extracted from a csv
159     file -> path oto store the output dataset
160     output:
```

```
    void but generated a .pikle file of the pandas dataframe
    '''
    global DONE
    first = True
    record = pd.DataFrame()
    for i, sesion in patient.groupby("seizure_file"):
        seizure_start = list(sesion.seizure_start.values)
        seizure_duration = list(sesion.seizure_duration.values)
        edf_data = mne.io.read_raw_edf(
            drive_path
            + "chb-mit-scalp-eeg-database-1.0.0/"
            + sesion["case"][0]
            + "/"
            + sesion["seizure_file"][0]
            + ".edf/",
            verbose=50,
        )
        if len(diff(channels, edf_data.ch_names)) == 0:
            clear_output(wait=True)
            progress_bar()
            edf_data.drop_channels(diff(edf_data.ch_names, channels))

            # locate and anotate seizures
            seizures = mne.Annotations(
                onset=seizure_start, duration=seizure_duration, description="
                    bad"
            )
            edf_data.set_annotations(seizures)

            # get seizures and split into epochs
            raw_seizures = mne.concatenate_raws(
                edf_data.crop_by_annotations(), verbose=50
            )
            seizures = mne.make_fixed_length_epochs(
                raw_seizures,
                EPOCH_DURATION,
                overlap=OVERLAP_DURATION,
                reject_by_annotation=False,
                verbose=50,
            )

            # get non-seizures and split into epochs
            non_seizures = mne.make_fixed_length_epochs(
                edf_data, EPOCH_DURATION, reject_by_annotation=True, verbose
                    =50
            )

            # computing features for non-seizure
            X = non_seizures._get_data(verbose=50)
            Y = seizures._get_data(verbose=50)
            std_X = standarize("time-std",np.std(X, axis=2))
            var_X = standarize("time-var",np.var(X, axis=2))
            max_X = standarize("time-max",np.max(X, axis=2))
            X_rfft = np.real(rfft(X, axis=2))
            std_X_rfft = standarize("freq-std",np.std(X_rfft, axis=2))
            var_X_rfft = standarize("freq-var",np.var(X_rfft, axis=2))
            max_X_rfft = standarize("freq-max",np.max(X_rfft, axis=2))
            index_X = list(range(len(X)))

            # computing convolutions for non-seizure
            fft_X = get_packed_stft(X, 6, 14, 0)

```

```
            # computing features for seizure
            std_Y = standarize("time-std",np.std(Y, axis=2))
            var_Y = standarize("time-var",np.var(Y, axis=2))
            max_Y = standarize("time-max",np.max(Y, axis=2))
            Y_rfft = np.real(rfft(Y, axis=2))
            std_Y_rfft = standarize("freq-std",np.std(Y_rfft, axis=2))
            var_Y_rfft = standarize("freq-var",np.var(Y_rfft, axis=2))
            max_Y_rfft = standarize("freq-max",np.max(Y_rfft, axis=2))
            fft_Y = get_packed_stft(Y, 6, 14, len(X))
            index_Y = list(range(len(X), len(X) + len(Y)))

            # computing features for seizure
            df_X = pd.DataFrame(data=std_X, index=index_X, columns=label_std)
            df_X = df_X.join(pd.DataFrame(data=var_X, index=index_X, columns=
                label_var))
            df_X = df_X.join(pd.DataFrame(data=max_X, index=index_X, columns=
                label_max))
            df_X = df_X.join(pd.DataFrame(data=std_X_rfft, index=index_X,
                columns=label_std_rfft))
            df_X = df_X.join(pd.DataFrame(data=var_X_rfft, index=index_X,
                columns=label_var_rfft))
            df_X = df_X.join(pd.DataFrame(data=max_X_rfft, index=index_X,
                columns=label_max_rfft))
            df_X = df_X.join(fft_X)

            df_Y = pd.DataFrame(data=std_Y, index=index_Y, columns=label_std)
            df_Y = df_Y.join(pd.DataFrame(data=var_Y, index=index_Y, columns=
                label_var))
            df_Y = df_Y.join(pd.DataFrame(data=max_Y, index=index_Y, columns=
                label_max))
            df_Y = df_Y.join(pd.DataFrame(data=std_Y_rfft, index=index_Y,
                columns=label_std_rfft))
            df_Y = df_Y.join(pd.DataFrame(data=var_Y_rfft, index=index_Y,
                columns=label_var_rfft))
            df_Y = df_Y.join(pd.DataFrame(data=max_Y_rfft, index=index_Y,
                columns=label_max_rfft))
            df_Y = df_Y.join(fft_Y)
            df = pd.concat([df_X, df_Y])

            # merge seizure and non-seizure
            aux = pd.concat(
                [
                    pd.DataFrame(data=0, index=index_X, columns=["seizure"]),
                    pd.DataFrame(data=1, index=index_Y, columns=["seizure"]),
                ]
            )
            df = df.join(aux)
            if record.empty:
                record = df
            else:
                record = pd.concat([record, df])
        else:
            print(diff(channels, edf_data.ch_names))
            print(sesion["seizure_file"], "no channels")
        DONE += 1

    # save file
    pickle.dump(df, file)
    file.close()


# loop per patient
```

```
for target in train:
    patient = seizure_pointers[seizure_pointers["case"] == target]
    create_df_file_from_patient(
        patient, open("data/train/" + target + ".pickle", "wb")
    )

for target in test:
    patient = seizure_pointers[seizure_pointers["case"] == target]
    create_df_file_from_patient(
        patient, open("data/test/" + target + ".pickle", "wb")
    )
print("compleated")
```

Listing 9: Complete code to extract features

```
import csv
import pandas as pd
import os, pickle

#train
dataframes = []
directory = 'data\\train'
for filename in os.listdir(directory):
    with open(os.path.join(directory, filename), 'rb') as f:
        data = pickle.load(f)
        seizure = data[data.seizure == 1]
        non_seizure = data[data.seizure == 0]
        non_seizure = non_seizure.sample(seizure.shape[0])
        data = pd.concat([seizure, non_seizure]).sample(frac=1)
        dataframes.append(data)

dataframes = pd.concat(dataframes).reset_index().drop(columns=["index"])
with open('data\\train.pickle', 'wb') as f:
    pickle.dump(dataframes, f)

#test
dataframes = []
directory = 'data\\test'
for filename in os.listdir(directory):
    with open(os.path.join(directory, filename), 'rb') as f:
        data = pickle.load(f)
        seizure = data[data.seizure == 1]
        non_seizure = data[data.seizure == 0]
        non_seizure = non_seizure.sample(seizure.shape[0])
        data = pd.concat([seizure, non_seizure]).sample(frac=1)
        dataframes.append(data)

dataframes = pd.concat(dataframes).reset_index().drop(columns=["index"])
with open('data\\test.pickle', 'wb') as f:
    pickle.dump(dataframes, f)
```

Listing 10: merge datasets from files

```
class SeizureData(torch.utils.data.Dataset):
    def __init__(self, x_val, y_val):
        self.freq_features = torch.tensor([[x] for x in x_val.stft],
            requires_grad=True, dtype=torch.float32)
        x_val = x_val.drop(columns=['stft'])
        self.n_samples = x_val.shape[0]
        self.y_data = torch.tensor(y_val.values.astype(np.float32),
            requires_grad=False)
```

```
        self.time_features = torch.tensor(x_val.values.astype(np.float32),
            requires_grad=True)

    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.time_features[index], self.freq_features[index], self.
            y_data[index]

    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples

dataset = SeizureData(X_train, y_train)
train_loader = torch.utils.data.DataLoader(dataset=dataset,
                           batch_size=50,
                           shuffle=True,
                           drop_last=True)
dataset = SeizureData(X_test, y_test)
test_loader = torch.utils.data.DataLoader(dataset=dataset,
                           batch_size=50,
                           shuffle=True,
                           drop_last=True)
```

Listing 11: batch loader of the dataset

```
learning_rate = 0.0001
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Listing 12: training loop

```
num_epochs = 20
num_total_steps = len(train_loader)

# Train the model
model.train()
for epoch in range(num_epochs):

    with torch.no_grad():
        val_losses, val_accs, val_lengths = 0, 0, 0
        for valid_time,valid_freq,valid_label in test_loader:
            outputs = model(valid_time,valid_freq)
            predicted = (outputs>0.5).float()
            val_losses += criterion(outputs, valid_label)
            val_accs += ((predicted == valid_label).sum().item())/valid_label.
                size(0)
            val_lengths += 1
        valid_accs.append(val_accs / val_lengths)
        valid_loss.append(val_losses / val_lengths)
        valid_iter.append(iter_num)

    for i, (time,freq,label) in enumerate(train_loader):

        #forward
        output = model(time, freq)
        loss = criterion(output, label)
        #plot
        train_accs.append(float(accuracy(output, label).data.numpy()))
        train_loss.append(float(loss.data.numpy()))
        train_iter.append(iter_num)
        iter_num += 1
        #backward
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if best is None:
            best = model
            best_loss = loss
        elif loss < best_loss:
            best = model
            best_loss = loss


    fig = plt.figure(figsize=(12,4))
    plt.subplot(1, 2, 1)
    plt.plot(train_iter, train_loss, label='train_loss')
    plt.plot(valid_iter, valid_loss, label='valid_loss')
    plt.xlabel("#iteration")
    plt.ylabel("loss")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(train_iter, train_accs, label='train_accs')
    plt.plot(valid_iter, valid_accs, label='valid_accs')
    plt.xlabel("#iteration")
    plt.ylabel("accuracy")
    plt.legend()
    plt.show()
    print(f"Train, it: {iter_num} loss: {train_loss[-1]:.2f} accuracy: {
        train_accs[-1]:.2f}")
    print(f"Valid, it: {iter_num} loss: {valid_loss[-1]:.2f} accuracy: {
        valid_accs[-1]:.2f}")
    clear_output(wait=True)
model = best
```

Listing 13: CNN + FFNN training loop

```
# Test the model
with torch.no_grad():
    all_predicted = torch.tensor([])
    all_label = torch.tensor([])
    for time, freq, labels in test_loader:
        outputs = model(time, freq)
        # outputs>0.5 means output set threshold to 0.5
        all_predicted = torch.cat([all_predicted, (outputs>0.5).float()])
        all_label = torch.cat([all_label, labels])
metrics(all_predicted, all_label).style.hide(axis="index")
```

Listing 14: detect seizures from the testing set and compute the metrics

```
# distance of between each threshold to be tested
step = 0.05
# list of threshold to test
threshold = np.arange(0, 1+step, step)

#dicctionary to store the values of the metrics
values = {}
for i in threshold:
    values[i] = {'snst': [], 'spcf': [], 'accr': []}

with torch.no_grad():
    for time, freq, labels in test_loader:
        outputs = model(time, freq)
        for t in threshold:
```

```
            predicted = (outputs>t).float()
            tn, fp, fn, tp = confusion_matrix(predicted, labels, labels=[0, 1]
                 ).ravel()
            if tp+tn == 0:
                values[t]['snst'].append(0)
            else:
                values[t]['snst'].append((tp/(tp+fn))*100)
            if tp+tn == 0:
                values[t]['spcf'].append(0)
            else:
                values[t]['spcf'].append((tn/(fp+tn))*100)
            values[t]['accr'].append(((tp+tn)/(tn+fp+fn+tp))*100)

    sensitivity = []
    specificity = []
    accuracy = []
    for _,v in values.items():
        sensitivity.append(sum(v['snst'])/len(v['snst']))
        specificity.append(sum(v['spcf'])/len(v['spcf']))
        accuracy.append(sum(v['accr'])/len(v['accr']))

#plot values from the dictionary of results
plt.plot(threshold, sensitivity, label='sensitivity')
plt.plot(threshold, accuracy, label='accuracy')
plt.plot(threshold, specificity, label='specificity')
plt.ylabel('Percentage')
plt.xlabel('Confidence threshold')
plt.legend()
plt.show()
```

Listing 15: Modifying the threshold variable to see the variation of the metrics

```
import mne
from scipy.fft import rfft
import json

# using the line below can open a separated window to see the results
%matplotlib qt

def standarize(vector):
    return vector - vector.mean() / vector.std()
def get_reduced_freq(target, batch_size, sampling_rate):
    result = []
    for channel in target:
        layer = []
        batch=[]
        target_ft = abs(rfft(channel))
        target_ft = [ x.real for x in target_ft]
        for i in target_ft[0:(sampling_rate*40)+2]:
            batch.append(i)
            if len(batch)==batch_size:
                batch_mean = sum(batch)/batch_size
                layer.append(batch_mean)
                batch=[]
        result.append(layer)
    return result

def get_packed_stft(target, batch_size, sampling_rate, i):
    fft_X = {}
    for x in [get_reduced_freq(x, batch_size, sampling_rate) for x in target]:
        fft_X[i] = x
        i+=1
```

```
    return pd.Series(fft_X).rename('stft')

target = "chb10"
file = "chb10_27.edf"
drive_path = "C:/Users/Eugene Chen/Desktop/UNI/Project/Data/"
seizure_pointers = pd.read_excel(drive_path + "seizure data.xlsx", index_col
    =0)
seizure_pointers["index"] = (
    seizure_pointers["seizure_file"]
    + " "
    + seizure_pointers["seizure_number"].astype(str)
)
record_path = drive_path+"/"+target+file
channels = [
    "P8-O2",
    "C4-P4",
    "FP1-F3",
    "FP2-F8",
    "CZ-PZ",
    "FP1-F7",
    "T7-P7",
    "C3-P3",
    "FP2-F4",
    "P4-O2",
    "F8-T8",
    "F7-T7",
    "F3-C3",
    "FZ-CZ",
    "P3-O1",
    "P7-O1",
    "F4-C4",
]
label_std = [n + "-std" for n in channels]
label_var = [n + "-var" for n in channels]
label_max = [n + "-max" for n in channels]
label_std_rfft = [n + "-std_rfft" for n in channels]
label_var_rfft = [n + "-var_rfft" for n in channels]
label_max_rfft = [n + "-max_rfft" for n in channels]
patient = seizure_pointers[seizure_pointers["index"] == "chb10_27 1"]
seizure_start = patient.seizure_start.values[0]
seizure_duration = patient.seizure_duration.values[0]
edf_data = mne.io.read_raw_edf(
            drive_path
            + "chb-mit-scalp-eeg-database-1.0.0/"
            + patient.case.values[0]
            + "/"
            + patient.seizure_file.values[0]
            + ".edf/",
            preload=True,
            verbose=50
        )
edf_data.drop_channels(list(set(edf_data.ch_names) - set(channels)))
seizures = mne.Annotations(
                onset=seizure_start, duration=seizure_duration, description="
                    ground true"
            )
edf_data.set_annotations(seizures)
EPOCH_DURATION = 2
OVERLAP_DURATION = 1
samples = mne.make_fixed_length_epochs(edf_data, EPOCH_DURATION, overlap=
    OVERLAP_DURATION, reject_by_annotation=False, verbose=50)
samples = samples._get_data(verbose=50)
```

```
90
91  # plot seizures annotated by an expert
92  edf_data.plot()
93
94  def standarize(key,vector):
95      with open('standarization_values.json') as json_file:
96          standarization = json.load(json_file)
97      metrics = standarization[key]
98      return (vector - metrics["mean"]) / metrics["std"]
99
100 #extract features from the unbalances dataset
101 std_X = standarize("time-std",np.std(samples, axis=2))
102 var_X = standarize("time-var",np.var(samples, axis=2))
103 max_X = standarize("time-max",np.max(samples, axis=2))
104 X_rfft = np.real(rfft(samples, axis=2))
105 std_X_rfft = standarize("freq-std",np.std(X_rfft, axis=2))
106 var_X_rfft = standarize("freq-var",np.var(X_rfft, axis=2))
107 max_X_rfft = standarize("freq-max",np.max(X_rfft, axis=2))
108 fft_X = get_packed_stft(samples, 6, 14, 0)
109 index_X = list(range(len(samples)))
110 df = pd.DataFrame(data=std_X, index=index_X, columns=label_std)
111 df = df.join(pd.DataFrame(data=var_X, index=index_X, columns=label_var))
112 df = df.join(pd.DataFrame(data=max_X, index=index_X, columns=label_max))
113 df = df.join(pd.DataFrame(data=std_X_rfft, index=index_X, columns=
        label_std_rfft))
114 df = df.join(pd.DataFrame(data=var_X_rfft, index=index_X, columns=
        label_var_rfft))
115 df = df.join(pd.DataFrame(data=max_X_rfft, index=index_X, columns=
        label_max_rfft))
116 df = df.join(fft_X)
117
118 model.eval()
119 detected = []
120 with torch.no_grad():
121     for index, item in df.iterrows():
122         freq = item.pop("stft")
123         freq = torch.tensor([[[x] for x in freq]], requires_grad=False, dtype=
                torch.float32).permute(0,2,1,3)
124         time = torch.tensor([item.values.astype(np.float32)], requires_grad=
                False)
125         outputs = model(time, freq)
126         predicted = (outputs>1).float()
127         if predicted == 1:
128             detected.append(index)
129
130 # this method is to reduce the amount of noice by ignoring seizures detected
        of 1s lenght
131 def remove_no_neighbor_numbers(numbers):
132     result = []
133     for i in range(len(numbers)):
134         if (i > 0 and i < len(numbers) - 1) or (len(numbers) == 1):
135             result.append(numbers[i])
136     return result
137 detected = remove_no_neighbor_numbers(detected))
138
139 # Anotate detected seizures and plot in a separated window
140 seizures_pred = mne.Annotations(
141                 onset=detected, duration=[1]*len(detected), description="
                    predicted"
142             )
143 edf_data.set_annotations(seizures_pred)
144 edf_data.plot()
```

Listing 16: Testing the model with unbalanced data

```
import mne
import matplotlib
import numpy as np
import pandas as pd
from sklearn import preprocessing
import seaborn
import matplotlib.pyplot as plt
import scipy
from scipy.fft import rfft, rfftfreq
import random
import pickle
import csv

def random_sample(arr: np.array, size: int) -> np.array:
    return arr[np.random.choice(len(arr), size=size, replace=False)]
def diff(lst1, lst2):
    return list(set(lst1) - set(lst2))

# path to the dataset
drive_path = "C:/Users/Eugene Chen/Desktop/UNI/Project/Data/"

seizure_pointers = pd.read_excel(drive_path+'seizure data.xlsx', index_col=0)
seizure_pointers['index'] = seizure_pointers['seizure_file'] +" "+
    seizure_pointers['seizure_number'].astype(str)
seizure_pointers=seizure_pointers.set_index('index')
channels = ['P8-O2', 'C4-P4', 'FP1-F3', 'FP2-F8', 'CZ-PZ', 'FP1-F7', 'T7-P7',
    'C3-P3', 'FP2-F4', 'P4-O2', 'F8-T8', 'F7-T7', 'F3-C3', 'FZ-CZ', 'P3-O1', '
    P7-O1', 'F4-C4']

# constants
EPOCH_DURATION = 2
OVERLAP_DURATION = 1

edf_data = mne.io.read_raw_edf(drive_path+'chb-mit-scalp-eeg-database-1.0.0/'+
    sesion['case'][0]+'/'+sesion['seizure_file'][0]+'.edf/', preload=True,
    verbose=40)
if len(diff(channels, edf_data.ch_names))==0:
    edf_data.filter(h_freq=40, l_freq=1, verbose=False)
    edf_data.drop_channels(diff(edf_data.ch_names, channels))
    seizures = mne.Annotations(onset=seizure_start, duration=seizure_duration,
         description='bad')
    edf_data.set_annotations(seizures)
    raw_seizures = mne.concatenate_raws(edf_data.crop_by_annotations(),
        verbose=False)
    seizures = mne.make_fixed_length_epochs(raw_seizures,EPOCH_DURATION,
        overlap=OVERLAP_DURATION,reject_by_annotation=False, verbose=False)
    non_seizures = mne.make_fixed_length_epochs(edf_data,EPOCH_DURATION,
        reject_by_annotation=True, preload = 'fast', verbose=False)

ictal_ft = rfft(seizure)
N = len(ictal_ft)
n = np.arange(N)
T = 2
freq = n/T

last_read = ""
seizures_array = np.array([]).reshape(0,17,512)
num = 0
```

```
epoch_list = []

for i, row in seizure_pointers.iterrows():
    clear_output(wait=True)
    print(str(num)+"/"+str(seizure_pointers.shape[0]-1))
    print(i)
    num+=1

    if last_read != row['seizure_file']:
        file_name = row['seizure_file']+'.edf/'
        folder_name = row['case']+'/'
        edf_data = mne.io.read_raw_edf(drive_path+'chb-mit-scalp-eeg-database
            -1.0.0/'+folder_name+file_name, verbose=40)
        print('current channels:', edf_data.ch_names)
        print('dropping channels:', diff(edf_data.ch_names, channels))
        print('not contains channels:', diff(channels, edf_data.ch_names))
        if len(diff(channels, edf_data.ch_names))!=0:
            continue
        edf_data.drop_channels(diff(edf_data.ch_names, channels))

    seizures = mne.Annotations(onset=row['seizure_start'], duration=row['
        seizure_duration'], description='bad')
    edf_data.set_annotations(seizures)

    if last_read != row['seizure_file'] or i == seizure_pointers.shape[0]-1:
        last_read = row['seizure_file']
        raw_seizures = mne.concatenate_raws(edf_data.crop_by_annotations(),
            verbose=False)
        seizures = mne.make_fixed_length_epochs(raw_seizures,EPOCH_DURATION,
            reject_by_annotation=False, verbose=False)
        seizures_array = np.concatenate((seizures_array,seizures.get_data()),
            axis=0)
        non_seizures = mne.make_fixed_length_epochs(edf_data,EPOCH_DURATION,
            reject_by_annotation=True, preload = 'fast', verbose=False)
        epoch_list.append(non_seizures)


label_std = [n + '-std' for n in channels]
label_var = [n + '-var' for n in channels]
label_max = [n + '-max' for n in channels]
label_skw = [n + '-skw' for n in channels]
label_krt = [n + '-krt' for n in channels]
label_min = [n + '-min' for n in channels]
label_avg = [n + '-avg' for n in channels]
label_std_rfft = [n + '-std_rfft' for n in channels]
label_var_rfft = [n + '-var_rfft' for n in channels]
label_max_rfft = [n + '-max_rfft' for n in channels]
label_skw_rfft = [n + '-skw_rfft' for n in channels]
label_krt_rfft = [n + '-krt_rfft' for n in channels]
label_min_rfft = [n + '-min_rfft' for n in channels]
label_avg_rfft = [n + '-avg_rfft' for n in channels]

X = non_seizures_array
Y = seizures_array

label=lst:correlation_matrix]
std_X = np.std(X, axis=2)
var_X = np.var(X, axis=2)
max_X = np.max(X, axis=2)
avg_X = np.average(X, axis=2)
min_X = np.min(X, axis=2)
skw_X = scipy.stats.skew(X, axis=2)
```

```
krt_X = scipy.stats.kurtosis(X, axis=2)
X_rfft = np.real(rfft(X, axis=2))
std_X_rfft = np.std(X_rfft, axis=2)
var_X_rfft = np.var(X_rfft, axis=2)
max_X_rfft = np.max(X_rfft, axis=2)
avg_X_rfft = np.average(X_rfft, axis=2)
min_X_rfft = np.min(X_rfft, axis=2)
skw_X_rfft = scipy.stats.skew(X_rfft, axis=2)
krt_X_rfft = scipy.stats.kurtosis(X_rfft, axis=2)
index_X = list(range(len(X)))

std_Y = np.std(Y, axis=2)
var_Y = np.var(Y, axis=2)
max_Y = np.max(Y, axis=2)
avg_Y = np.average(Y, axis=2)
min_Y = np.min(Y, axis=2)
skw_Y = scipy.stats.skew(Y, axis=2)
krt_Y = scipy.stats.kurtosis(Y, axis=2)
Y_rfft = np.real(rfft(Y, axis=2))
std_Y_rfft = np.std(Y_rfft, axis=2)
var_Y_rfft = np.var(Y_rfft, axis=2)
max_Y_rfft = np.max(Y_rfft, axis=2)
avg_Y_rfft = np.average(Y_rfft, axis=2)
min_Y_rfft = np.min(Y_rfft, axis=2)
skw_Y_rfft = scipy.stats.skew(Y_rfft, axis=2)
krt_Y_rfft = scipy.stats.kurtosis(Y_rfft, axis=2)
index_Y = list(range(len(X),len(X)+len(Y)))

df_X = pd.DataFrame(data=std_X, index=index_X, columns=label_std)
df_X = df_X.join(pd.DataFrame(data=var_X, index=index_X, columns=label_var))
df_X = df_X.join(pd.DataFrame(data=max_X, index=index_X, columns=label_max))
df_X = df_X.join(pd.DataFrame(data=avg_X, index=index_X, columns=label_avg))
df_X = df_X.join(pd.DataFrame(data=min_X, index=index_X, columns=label_min))
df_X = df_X.join(pd.DataFrame(data=skw_X, index=index_X, columns=label_skw))
df_X = df_X.join(pd.DataFrame(data=krt_X, index=index_X, columns=label_krt))
df_X = df_X.join(pd.DataFrame(data=std_X_rfft, index=index_X, columns=
    label_std_rfft))
df_X = df_X.join(pd.DataFrame(data=var_X_rfft, index=index_X, columns=
    label_var_rfft))
df_X = df_X.join(pd.DataFrame(data=max_X_rfft, index=index_X, columns=
    label_max_rfft))
df_X = df_X.join(pd.DataFrame(data=avg_X_rfft, index=index_X, columns=
    label_avg_rfft))
df_X = df_X.join(pd.DataFrame(data=min_X_rfft, index=index_X, columns=
    label_min_rfft))
df_X = df_X.join(pd.DataFrame(data=skw_X_rfft, index=index_X, columns=
    label_skw_rfft))
df_X = df_X.join(pd.DataFrame(data=krt_X_rfft, index=index_X, columns=
    label_krt_rfft))


df_Y = pd.DataFrame(data=std_Y, index=index_Y, columns=label_std)
df_Y = df_Y.join(pd.DataFrame(data=var_Y, index=index_Y, columns=label_var))
df_Y = df_Y.join(pd.DataFrame(data=max_Y, index=index_Y, columns=label_max))
df_Y = df_Y.join(pd.DataFrame(data=avg_Y, index=index_Y, columns=label_avg))
df_Y = df_Y.join(pd.DataFrame(data=min_Y, index=index_Y, columns=label_min))
df_Y = df_Y.join(pd.DataFrame(data=skw_Y, index=index_Y, columns=label_skw))
df_Y = df_Y.join(pd.DataFrame(data=krt_Y, index=index_Y, columns=label_krt))
df_Y = df_Y.join(pd.DataFrame(data=std_Y_rfft, index=index_Y, columns=
    label_std_rfft))
df_Y = df_Y.join(pd.DataFrame(data=var_Y_rfft, index=index_Y, columns=
    label_var_rfft))
```

```
df_Y = df_Y.join(pd.DataFrame(data=max_Y_rfft, index=index_Y, columns=
    label_max_rfft))
df_Y = df_Y.join(pd.DataFrame(data=skw_Y_rfft, index=index_Y, columns=
    label_skw_rfft))
df_Y = df_Y.join(pd.DataFrame(data=krt_Y_rfft, index=index_Y, columns=
    label_krt_rfft))
df_Y = df_Y.join(pd.DataFrame(data=avg_Y_rfft, index=index_Y, columns=
    label_avg_rfft))
df_Y = df_Y.join(pd.DataFrame(data=min_Y_rfft, index=index_Y, columns=
    label_min_rfft))

df = pd.concat([df_X, df_Y])
df=(df-df.mean())/df.std()
aux = pd.concat([pd.DataFrame(data=0, index=index_X, columns=['seizure']),pd.
    DataFrame(data=1, index=index_Y, columns=['seizure'])])
df = df.join(aux)

# select a random channels with extracted features and seizure
ch = "F8-T8"
onset = df[[ch+"-std",ch+"-var",ch+"-max",ch+"-skw",ch+"-krt",ch+"-avg",ch+"-
    min",ch+"-std_rfft",ch+"-var_rfft",ch+"-max_rfft",ch+"-skw_rfft",ch+"-
    krt_rfft",ch+"-avg_rfft",ch+"-min_rfft","seizure"]]
# reject null values
onset = onset.dropna()

# plot correlation matrix
fig, ax = plt.subplots(figsize=(12, 3))
fig.canvas.draw()
ax.set_xticks(np.arange(15), minor=False)
ax.set_xticklabels(['std', 'var', 'max', 'skw', 'krt',
       'avg', 'min', 'std', 'var',
       'max', 'skw', 'krt', 'avg',
       'min', 'seizure'], minor=False)
ax.invert_yaxis()
heat = onset.corr()["seizure"].to_numpy()
heat = np.expand_dims(heat, axis=0)
im = ax.imshow(heat)
ax.get_yaxis().set_visible(False)
for i,label in np.ndenumerate(heat):
       ax.text(i[1],0,"{:.2f}".format(label),ha='center',va='center')
fig.colorbar(im)
```

Listing 17: get correlation matrix of different features

Images

Figure 1: Channels selected from the dataset

EEG data analisis

```
import numpy as np
import pandas as pd
from numpy.fft import fft, rfft
from scipy.stats import skew, kurtosis
import csv, mne, math
import matplotlib.pyplot as plt
import seaborn as sns
```

Import a saple from the dataset CHB-MIT Scalp EEG Database

```
target = "chb16"
file = "chb16_14.edf"

# this drive path can be modified
drive_path = "C:/Users/Eugene Chen/Desktop/UNI/Project/Data/"
seizure_pointers = pd.read_excel(drive_path + "seizure data.xlsx", index_col=0)
seizure_pointers["index"] = (
    seizure_pointers["seizure_file"]
    + " "
    + seizure_pointers["seizure_number"].astype(str)
)
record_path = drive_path+"/"+target+file
channels = [
    "P8-O2",
    "C4-P4",
    "FP1-F3",
    "FP2-F8",
    "CZ-PZ",
    "FP1-F7",
    "T7-P7",
    "C3-P3",
    "FP2-F4",
    "P4-O2",
    "F8-T8",
    "F7-T7",
    "F3-C3",
    "FZ-CZ",
    "P3-O1",
    "P7-O1",
    "F4-C4",
]
patient = seizure_pointers[seizure_pointers["index"] == "chb16_14 1"]


seizure_start = patient.seizure_start.values[0]
seizure_duration = patient.seizure_duration.values[0]
edf_data = mne.io.read_raw_edf(
            drive_path
            + "chb-mit-scalp-eeg-database-1.0.0/"
            + patient.case.values[0]
            + "/"
            + patient.seizure_file.values[0]
            + ".edf/",
            preload=True,
            verbose=50
        )
# appliying low pass filter
edf_data.filter(h_freq=40, l_freq=0, verbose=False)
edf_data.drop_channels(list(set(edf_data.ch_names) - set(channels)))
seizures = mne.Annotations(
                onset=seizure_start, duration=seizure_duration, description="bad"
            )
edf_data.set_annotations(seizures)
seizure_sample = mne.concatenate_raws(
                edf_data.crop_by_annotations(), verbose=50
            )
non_seizures = mne.make_fixed_length_epochs(
                edf_data, seizure_duration, reject_by_annotation=True, verbose=50
            )


sample_id = 30
channel = 4

no_seizure = non_seizures[sample_id]._get_data(verbose=50)[0][channel]
sample_len = no_seizure.shape[0]
seizure = seizure_sample.get_data()[channel][:sample_len]
step = seizure_duration/sample_len
x = np.arange(0, seizure_duration, step)
```

Selecting a random 2s epoch of the dataset

2 seconds means 2 times the sampling rate witch is `2 * 256` = 512

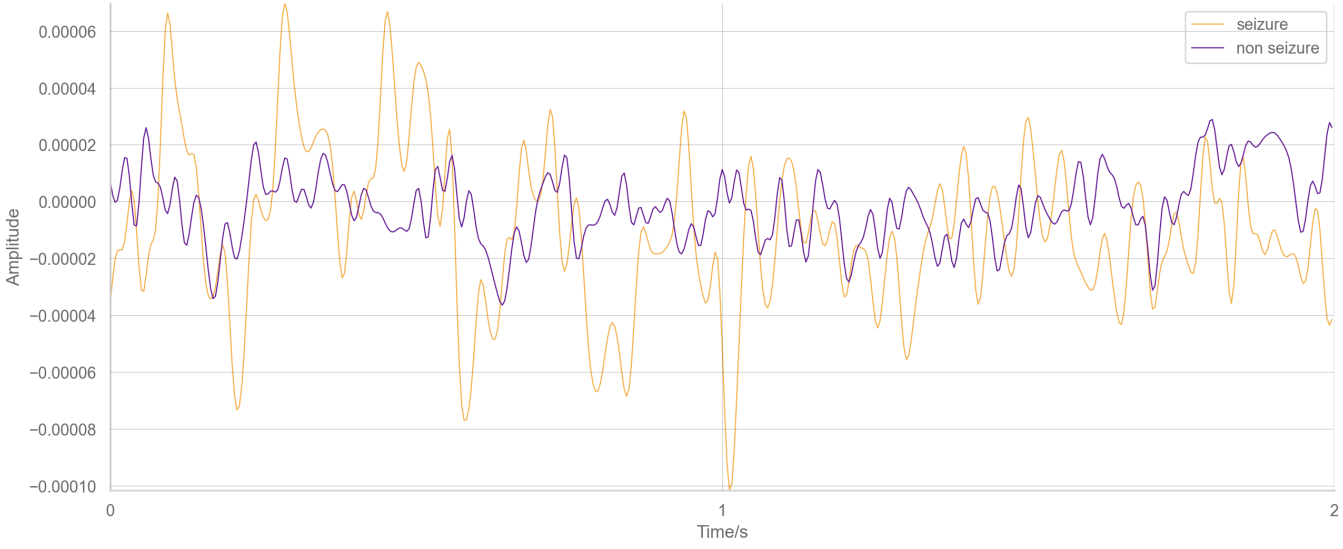
```
#Selecting 2s of non-seizure
no_seizure = no_seizure[1000:1512]
#Selecting 2s of seizure
seizure = seizure[1000:1512]
sample_len = no_seizure.shape[0]
step = 2/sample_len
x = np.arange(0, 2, step)
```

```
# Some desing declarations, can be ignored
sns.set(font='arial',
        rc={
            'axes.edgecolor': 'lightgrey',
            'axes.facecolor': 'None',
            'axes.grid': False,
            'axes.labelcolor': 'dimgrey',
            'axes.spines.right': False,
            'axes.spines.top': False,
            'figure.facecolor': 'white',
            'lines.solid_capstyle': 'round',
            'patch.edgecolor': 'w',
            'patch.force_edgecolor': True,
            'text.color': 'dimgrey',
            'xtick.bottom': False,
            'xtick.color': 'dimgrey',
            'xtick.direction': 'out',
            'xtick.top': False,
            'ytick.color': 'dimgrey',
            'ytick.direction': 'out',
            'ytick.left': False,
            'axes.axisbelow': False,
            'ytick.right': False})
```

Plotting to see the difference between seizure and non-seizure

```
fig, ax = plt.subplots(figsize=(15, 6), dpi=200)
ax.grid(color='black', linestyle='-', linewidth=0.1)
ax.set_xticks(np.arange(0, 14, 1))
ax.axis(xmin=0,xmax=2,ymax=seizure.max(),ymin=seizure.min())
ax.plot(x, seizure, color ='#F5B14C', linewidth=0.8, label="seizure")
ax.plot(x, no_seizure, color ='#661D98', linewidth=0.8, label="non seizure")
# ax.set_title("Seizure and non seizure in time domain comparison")
ax.set_xlabel("Time/s")
ax.set_ylabel("Amplitude")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1e128319970>
```



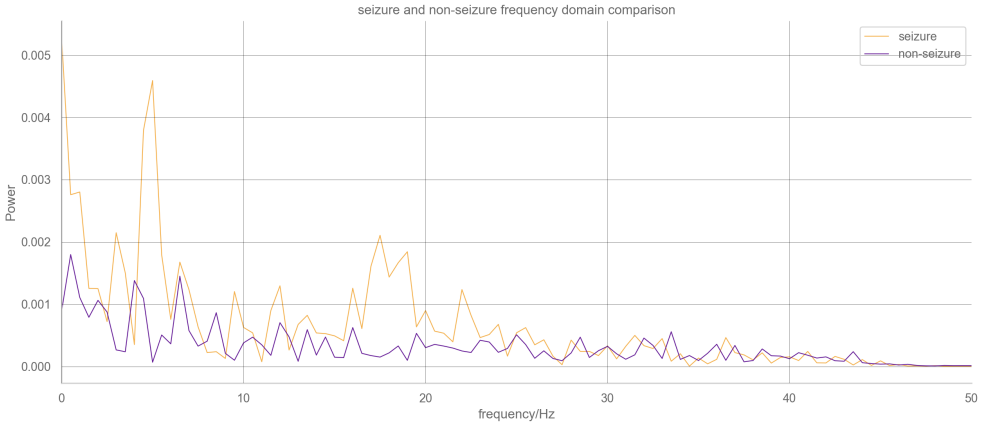
Computing fourier transformation ang getting only the real part

```
ictal_ft = abs(rfft(seizure))
ictal_ft = [ x.real for x in ictal_ft]
interictal_ft = abs(rfft(no_seizure))
interictal_ft = [ x.real for x in interictal_ft]
N = len(ictal_ft)
n = np.arange(N)
T = 2
freq = n/T
freq_sampling_rate = (freq<1).sum()
```

Plot the frequency domain from an epoch of seizure and non-seizure

```
fig, ax = plt.subplots(figsize=(15, 6), dpi=200)
ax.grid(color='black', linestyle='-', linewidth=0.2)
# ax.axis(ymax=ictal_ft.max(),ymin=0)
ax.axis(xmin=0,xmax=50)
ax.plot(freq, ictal_ft, color ='#F5B14C', linewidth=0.8, label="seizure")
ax.plot(freq, interictal_ft, color ='#661D98', linewidth=0.8, label="non-seizure")
ax.set_title("seizure and non-seizure frequency domain comparison")
ax.set_xlabel("frequency/Hz")
ax.set_ylabel("Power")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1e11ae14250>
```




```
def get_reduced_freq(target, batch_size, sampling_rate):
    '''
    input:
    target -> sample to apply the tranformation
    batch_size -> lenght of the signal reduction (int)
    sampling_rate -> sampling rate of the signal (int)
    output:
    return target applied reduction and fourier tranformation
    '''
    result = []
    batch=[]
    for i in target[0:(sampling_rate*40)+1]:
        batch.append(i)
        if len(batch)==batch_size:
            batch_mean = sum(batch)/batch_size
            result.append(batch_mean)
            batch=[]
    return result


def get_reduced_freq_repeated(target, batch_size, sampling_rate):
    '''
    input:
    target -> sample to apply the tranformation
    batch_size -> lenght of the signal reduction (int)
```

```
    sampling_rate -> sampling rate of the signal (int)
    output:
    return target applied reduction and fourier tranformation
    but returning an array with the same lenght as the target
    '''
    result = []
    batch=[]
    for i in target[0:(sampling_rate*40)+1]:
        batch.append(i)
        if len(batch)==batch_size:
            batch_mean = sum(batch)/batch_size
            result+=[batch_mean]*batch_size
            batch=[]
    result+=[sum(batch)/len(batch)]*len(batch)
    return result
```

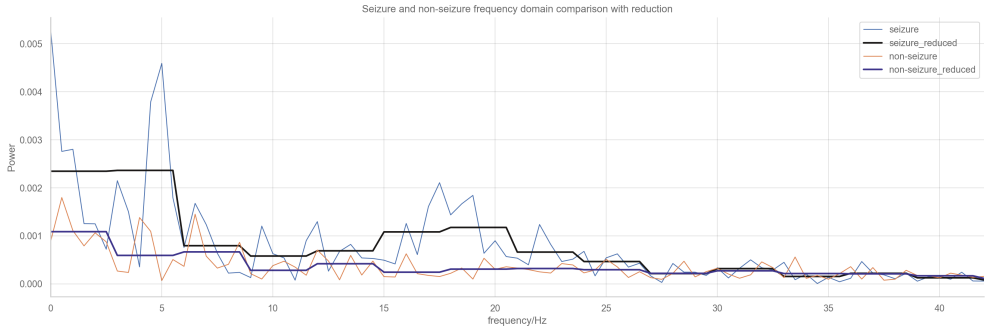
Calculating the results of the reduction aaplied

```
num_baches = 20
bach_size=6
ictal_ft_features = get_reduced_freq(ictal_ft, bach_size, 14)
interictal_ft_features = get_reduced_freq(interictal_ft, bach_size, 14)
reduced_ictal_ft  = get_reduced_freq_repeated(ictal_ft, bach_size, 14)
reduced_interictal_ft = get_reduced_freq_repeated(interictal_ft, bach_size, 14)
```

Plot the each epoch after fourier transformation and

```
fig, ax = plt.subplots(figsize=(20, 6), dpi=200)
ax.grid(color='black', linestyle='-', linewidth=0.1)
ax.axis(xmin=0,xmax=42)
ax.plot(freq, ictal_ft, linewidth=1, label="seizure")
ax.plot(freq, reduced_ictal_ft, color ='#191716', linewidth=2, label="seizure_reduced")
ax.plot(freq, interictal_ft, linewidth=1, label="non-seizure")
ax.plot(freq, reduced_interictal_ft, color ='#3d348b', linewidth=2, label="non-seizure_reduced")
ax.set_title("Seizure and non-seizure frequency domain comparison with reduction")
ax.set_xlabel("frequency/Hz")
ax.set_ylabel("Power")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1e11b1f2fd0>
```

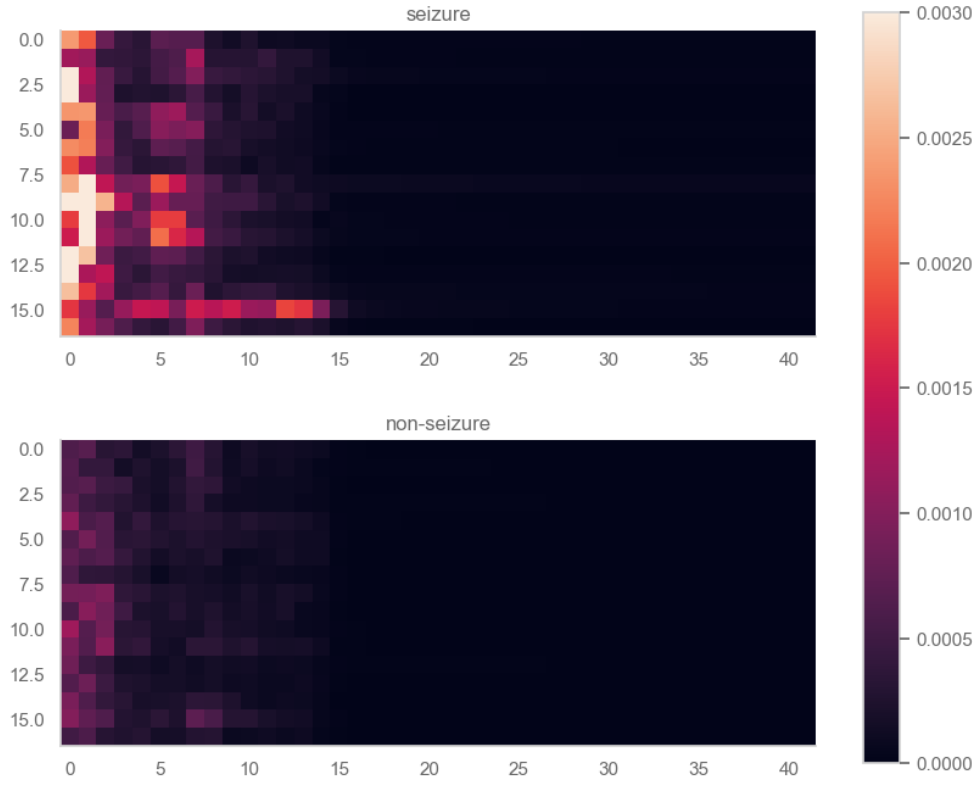


matrix from all layers frequencies

```
bach_size=6
convolution_features_ictal = []
convolution_features_interictal = []
for channel in range(seizure_sample.get_data().shape[0]):
    seizure = seizure_sample.get_data()[channel][1000:1512]
    no_seizure = non_seizures[sample_id]._get_data(verbose=50)[0][channel][1000:1512]
    ictal_ft = abs(rfft(seizure))
    ictal_ft = [ x.real for x in ictal_ft]
    interictal_ft = abs(rfft(no_seizure))
    interictal_ft = [ x.real for x in interictal_ft]
    ictal_ft_features = get_reduced_freq(ictal_ft, bach_size, 14)
    interictal_ft_features = get_reduced_freq(interictal_ft, bach_size, 14)
    convolution_features_ictal.append(ictal_ft_features)
    convolution_features_interictal.append(interictal_ft_features)
```

```
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10,8))
im =axes[0].imshow(convolution_features_ictal, vmin = 0, vmax = 0.003)
axes[0].set_title("seizure")
axes[1].imshow(convolution_features_interictal, vmin = 0, vmax = 0.003)
axes[1].set_title("non-seizure")
fig.colorbar(im, ax=axes.ravel().tolist())
```

```
<matplotlib.colorbar.Colorbar at 0x1e11f8406a0>
```



Technical
University of
Denmark

Ørsteds Plads, Building 345B
2800 Kgs. Lyngby
Tlf. 4525 1700

www.byg.dtu.dk