Data Augmentation Techniques for Deep Learning in Motor Imagery EEG Classification

Alberto Baldrati
alberto.baldrati@stud.unifi.it

Serban Cristian Tudosie
cris.sherban.t@gmail.com

Abstract

Brain computer interfaces (BCIs) provide a new communication bridge between human minds and devices, however the ability to control such devices with our minds largely depends on the accurate classification and identification of non-invasive EEG signals. For this reason recent advances in deep learning have helped the progress in BCI field with convolutional neural networks that are becoming the new cutting-edge tools to tackle the problem of EEG recognition. In order to successfully train a convolutional neural network a large amount of data is needed and due to the strict requirements for subjects and experimental environments, it is difficult to collect large-scale and high-quality EEG data. Based on this, the aim of this work is to investigate the performance of different data augmentation methods for the classification of Motor Imagery (MI) data using a Convolutional Neural Network tailored for EEG named EEGNet [1].

1 Introduction

The Brain Computer Interfaces (BCIs) provides a new pathway between the human brain and computer by analyzing electric signals generated by the nervous systems. Traditionally, EEG is acquired through invasive ways, but in recent years it has been possible to collect EEG with relatively inexpensive noninvasive approaches (see [2] for an example). Because of that EEG-based BCI systems have gained popularity in various real-world applications from the healthcare domain to the entertainment industry. For example in the healthcare domain, EEG signals are used to detect the organic brain injury or predict epileptic seizure [3]. In our experiments we will focus on Motor Imagery (MI) EEG signals which are widely used in BCI systems. MI is a mental process that imitates motor intention without real motion output, i.e., the brain imagines the entire movement without actually contracting the muscles. In the field of neurophysiology there are many similarities between real movements and motor imagery because of the consistency of the peripheral autonomic nerves and the cortical potential [4]. Therefore, MI is a brain activity that is similar to real exercise and may cause a change in the potential of the cortex [5]. As a result, the EEG signals can be decoded into different command to control external devices when the brain imagines different movements.

Exploring an effective and reliable way to extract robust features and classify EEG signals effectively is an important stage in applications which rely on this kind of data. Traditionally, the process of EEG signal recognition consists of two stages: a feature extraction stage, where meaningful informations are extracted from EEG recordings and a classification stage, where a decision is made from the selected features. Traditional feature extraction methods mainly include frequency band analysis, multiscale radial basis function, continuous wavelets transform etc. In the classification stage, many traditional algorithms like SVMs and Bayesian classifiers have been employed. However these methods heavily rely on handcrafted features, and the feature selection steps are time-consuming even for experts in this domain. Additionally, such a two-stage model is inconvenient to train and implement. Recently, deep learning techniques have gained widespread attention and achieved remarkable success in many fields as computer vision, speech recognition and natural language processing. Applying deep learning models to complete the feature extraction task shows an appreciable performance as it does not require any handcrafting feature selection steps. Therefore, its effectiveness has encouraged researchers to adopt deep learning methods to recognize EEG recordings [6]. In recent years various architectures with different data preprocessing pipelines have been developed and tested. For instance [7] proposed to convert EEG recordings to 2-D images by using complex Morlet wavelets and classify this image representations using CNN. In [8] authors develop a system still based on CNN that uses a different preprocessing pipeline based on short-time Fourier transform (STFT) instead.

In all our experiments we have used EEGNet [1] to classify the motor imagery EEG data. EEGNet is a CNN tailored for EEG signals which takes as inputs the EEG recordings in the form of times series after a filtering process (more details about preprocessing pipeline in section 2.3 on the following page). This convNet is able to learn robust feature representations and classify the signal simultaneously. However being an end-to-end model it has more parameters to optimize than two-stage traditional methods, therefore to effectively take advantage of such expressiveness power we require a larger amount of data.

The private dataset we experimented with is made of 240 samples split in 3 classes. Such shortage of data makes it clear that data augmentation techniques might improve the robustness and the performance of our entire model. Data augmentation is a technique that generates effective training samples from origin datasets when we do not have sufficient data to train a deep learning model. This approach has been applied for years in computer vision tasks where building a new sample from an image is quite easy (i.e. it is possible to perform rotation, scaling, horizontal flips, perturbations to brightness, contrast, color etc.). However such transformations are not useful for EEG data since the latter are dynamic time series. In this work we explore and test current data augmentation techniques available in literature tailored for EEG data on our private dataset reporting which of them can bring an actual improvement.

2 Dataset

2.1 Hardware

The frame that holds the used hardware is the Ultracortex MarkIV, from OpenBCI that has been printed with PLA filament at home. Eight dry EEG electrodes (figure 1b) were used to acquire the signal. The board that holds the controller is the *OpenBCI Cyton Biosensing Board* (figure 1a) which is a 8-channel neural interface with a 32-bit processor. As its core the OpenBCI Cyton Board implements the *PIC32MX250F128B* micro-controller giving it much local memory and fast processing speeds. Data is sampled at 250Hz on each of the eight channels.

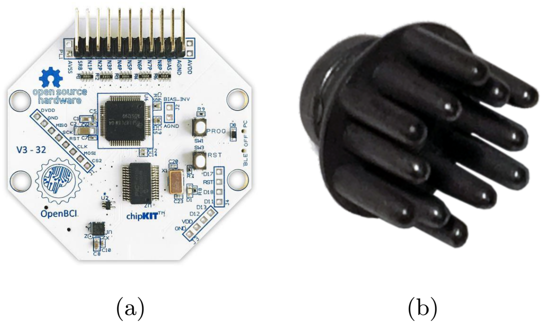
(a) (b)

Figure 1: (a) Cyton Board, (b) dry electrode

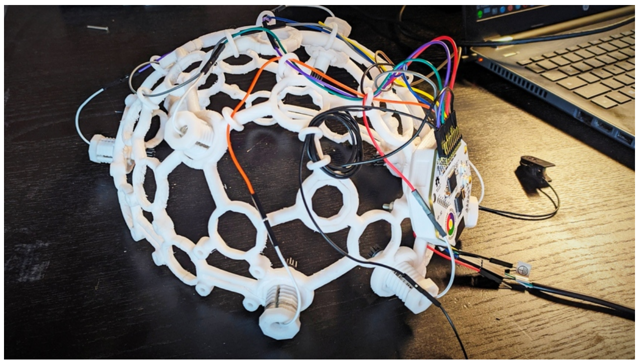

Figure 2: Headset

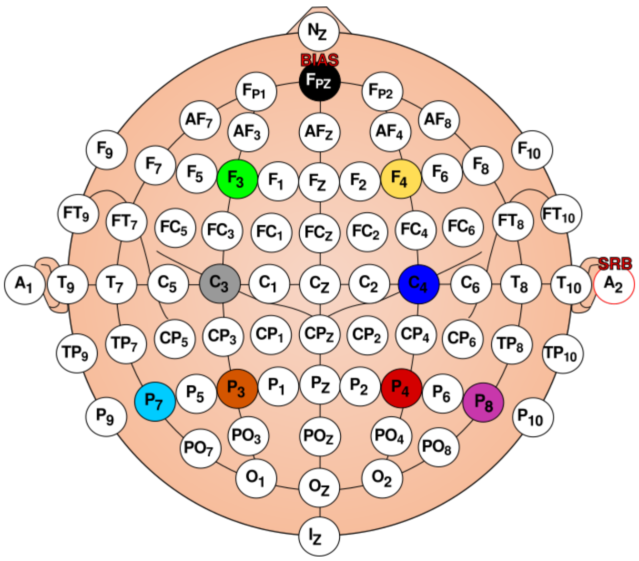

Figure 3: Electrodes positions

2.2 Data Acquisition

Figure 3 shows the position of the eight dry electrodes during data acquisition according to the 10-20 system for scalp electrode positioning. In the figure each electrode placement site has a letter to identify the area of the brain it is reading from: pre-frontal (Fp), frontal (F), temporal (T), parietal (P), occipital (O) and central (C). The colored positions show the ones used for this study. The bias dry electrode is placed frontally.

In the acquisition procedure the subject has to sit down in a comfortable position on a chair, with hands in a resting position and feet touching the ground. The acquisition procedure has been taken underground, with no electronic devices connected to the wall sockets in proximity of the subject in order to reduce the 50Hz (power line frequency in Italy) noise. This physical reduction of noise is taken in consideration because of the lack of isolation of the components of the acquisition system. During the acquisition the subject sees on a screen which motor action he should imagine and starts to think about that. This procedure lasts about eight seconds, but actually we start recording after about six seconds and such recording lasts one second. The reason for this is that we try to acquire the most stable samples possible. In fact the human capacity of switching among thoughts is not instantaneous and we want the motor imagery thought to be stable before acquisition.
Each acquisition sessions saves a total of 50 samples, the subject has to do multiple and different sessions of acquisition to make the network generalize as much as possible since every tiny movement or misalignment of one electrode may produce different data.

2.3 Data

The dataset we collected with the hardware described in section 2.1 following the procedure outlined in section 2.2 on the preceding page is made of 240 samples equally split in three classes. The three classes correspond to three different motor imagery actions which are: (thinking of) moving the *feet*, (thinking of) moving the *hands* and a *none* action where the subject does not think at any muscular movement.

2.3.1 Data Format

The data acquisition protocol described above saves each sample as a raw EEG signal embedded in a matrix. Each row of such matrix is the EEG signal taken from a different electrode and, since the Cyton Board has a sampling rate of 250Hz and the actual acquisition time is one second, in a sample for each channel there are 250 timestamps. This means that each sample is a 8×250 matrix. The data is not preprocessed during acquisition to keep the possibility of tuning parameters like lowpass and highpass frequency filters at a later stage. Figure 4 shows an example of a sample as it is before any kind of processing. We can notice that the board saves each channel on a different scale

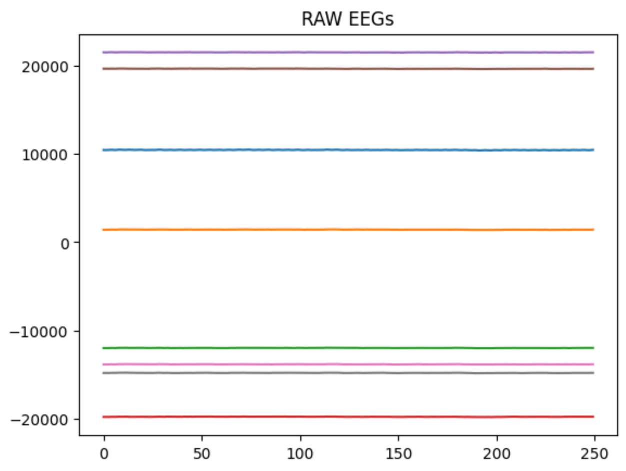

Figure 4: Raw EEG data

2.3.2 Data Preprocessing

Since the raw EEG data are stored at a different scale the first thing that should be done is the standardization of the dataset in order to have all the channels on the same scale. The standardization is performed channel-wise, which means that after such step each channel has zero mean and an unitary standard deviation. We can notice that in the standardization process each channel is not affected by other channels in the same sample or by other samples in the dataset. Figure 5 shows an example of a sample after the mentioned above channel-wise standardization.

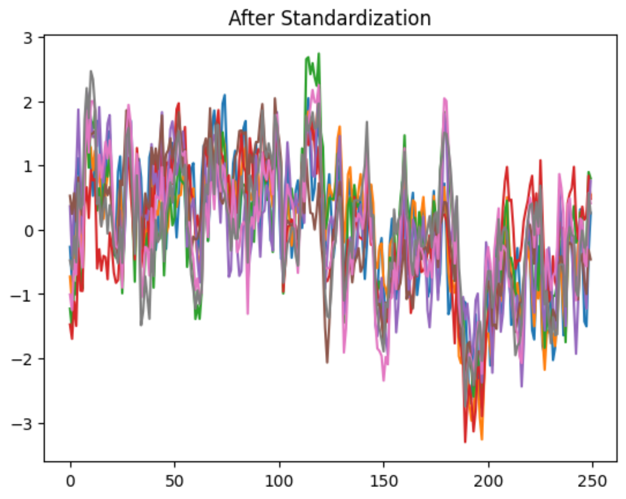

Figure 5: EEG data after standardization

After this first crucial step, the next data manipulation step to perform is to apply a notch filter in order to filter out the 50Hz or 60Hz depending on the local grid power system. The chosen approach is the usage of a Butterworth bandstop filter from BrainFlow [9]. In figure 6 are shown the effects of band stop filtering at 50Hz on the sample displayed in figure 5 on the preceding page
We can notice that this sample is not very different from the previous one, this is due to the fact that the data were acquired underground without any electronics nearby. In less ideal conditions the bandstop pass is crucial to obtain meaningful EEG data.

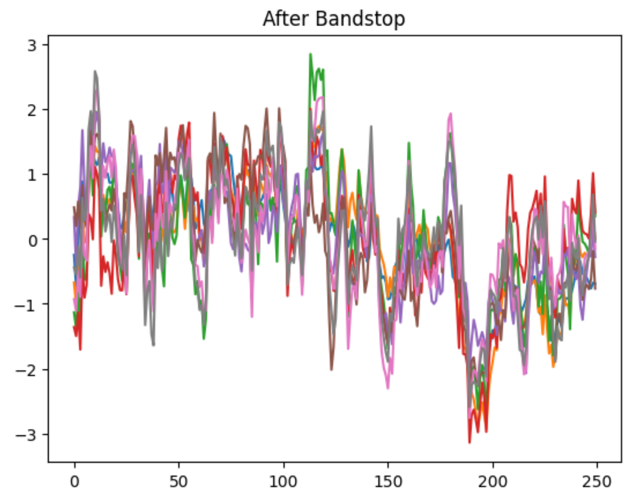

Figure 6: EEG data after band stop at 50Hz

As a final step, we carried out bandpass filtering. Grid searches allowed to find 8Hz and 45Hz as the best thresholds. This filter is computed by a Butterworth bandpass filter, again using the BrainFlow library. In figure 7 are shown the effect of the above mentioned bandpass filtering.

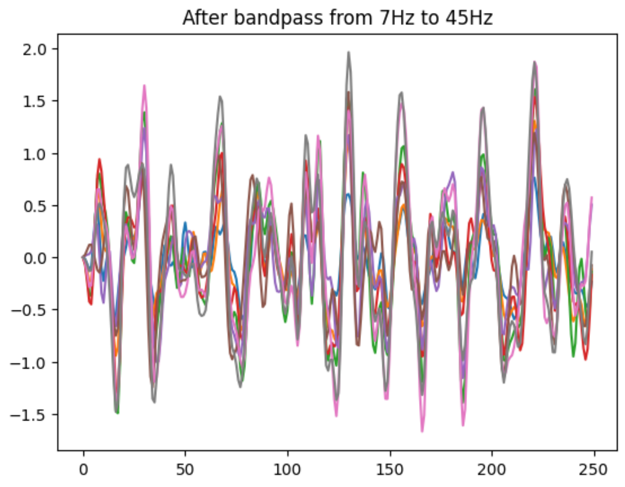

Figure 7: EEG data after bandpass between 8Hz and 45Hz

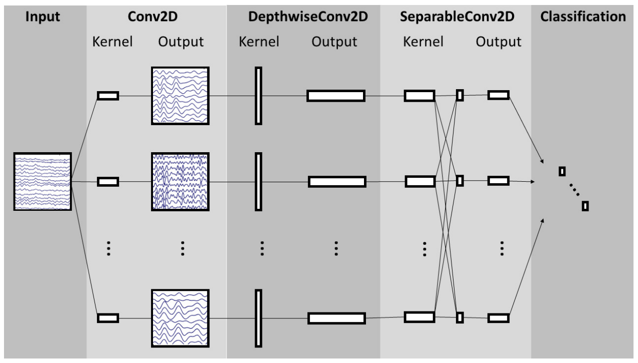

Figure 8: Overall visualization of the EEGNet architecture

3 EEGNet

In all our experiments we have used EEGNet [1] as end-to-end convolutional neural network. It is a compact CNN architecture for EEG-based BCIs that can be applied across several different BCI paradigms and can be trained without a very large dataset.

From figure 8 where the overall structure of the net is displayed we can understand that the net is made of 4 blocks: the first Conv2D block, the DepthwiseConv2D block, the SeparableConv2D block and the final classifications block.

In the **Conv2D** block are fitted F_1 2D convolutional filters of size (1,125), with the filter length chosen to be half the sampling rate of the data (250Hz in our case), outputting F_1 feature maps containing the EEG signal at different band-pass frequencies. Setting the length of the temporal kernel at half the sampling rate allows for capturing frequency information at 2Hz and above. In other words this block performs a temporal convolution where informations are taken only along the time dimension as it is shown in figure 9 on the next page. After such temporal convolution a Batch Normalization [10] is applied along the feature map dimension.

In the **DepthwiseConv2D** block a *Depthwise Convolution* with kernel size of (Channel, 1) is then performed to learn a spatial filter. In CNN applications for computer vision the main benefit of a depthwise convolution is reducing the number of trainable parameters to fit, as these convolutions are not fully-connected

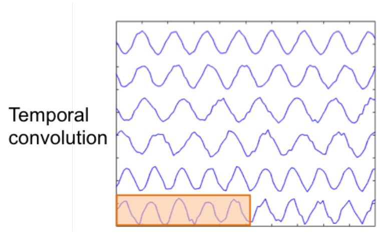

Figure 9: Temporal Convolution

to all previous feature maps (as shown in figure 8 on the preceding page). Moreover, when used in EEG-specific applications, this operation provides a direct way to learn spatial filter for each temporal filter, thus enabling the efficient extraction of frequency-specific spatial filters (see figure 10). A depth parameter D controls the number of spatial filters to learn for each feature map ($D = 1$ is shown in figure 8 on the preceding page for illustration purposes). Each spatial filter is also regularized by using a maximum norm constraint of 1 on its weights; $\|w\|^2 < 1$.

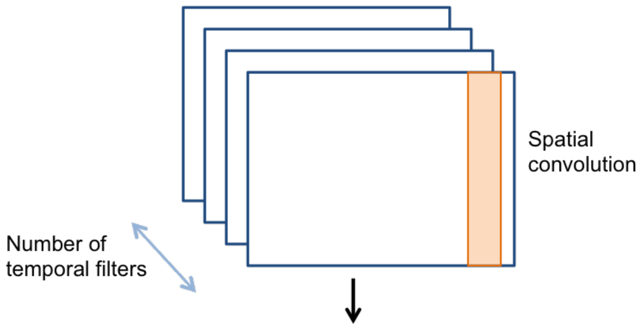

Figure 10: Spatial Convolution

After such depthwise convolution a Batch Normalization layer is applied along the feature map dimension before applying the exponential linear unit (ELU) non linearity. In order to reduce the sampling rate of the signal and further regularize the model we apply an average pooling layer with kernel size (1,4) and a dropout layer with dropout probability set to 0.5.

In the **SeparableConv2D** block is used a *Separable Convolution*, which is a Depthwise Convolution (in our case with kernel size of (1,16)) followed by F_2 (1,1) point-wise convolution. The main benefits of separable convolution are reducing the number of parameters to fit and explicitly decoupling the relationship within and across feature maps by first learning a kernel which summarize each feature map individually, then optimally merging the output afterwards. When used for EEG-specific applications this operation separates learning how to summarize individual feature maps in time (the depthwise convolution) with how to optimally combine the feature maps (the point-wise convolutions). This operation is also particularly useful for EEG signals as different feature maps may represent data at different time-scales of information. Batch Normalization, ELU activation function and use an average pooling layer of size (1,8) are then used.

In the **classification** block, the features are passed directly to a softmax classification with N units, N being the number of classes in the data. A dense layer for feature aggregation is omitted in order to reduce the number of free parameters in the model.

We investigate several different configuration of the EEGNet architecture by varying the number of filter F_1 and the number of spatial filters per temporal filters D to learn. As suggested in [1] in all our experiments $F_2 = D * F_1$ is set (where F_2 is the number of temporal filters along with their associated spatial filters) . Although in principle F_2 can take any value; $F_2 < D * F_1$ denotes a compressed representation, learning fewer feature maps than inputs, whereas $F_2 > D * F_1$ denotes an overcomplete representation, learning more feature maps than the inputs. After experimenting a while with our dataset we found that $F_1 = 12$ and $D = 2$ brought the best overall performance.

4 EEG Data Augmentation

As briefly introduced above the main focus of this work is to explore and experiment various data augmentation techniques in order to mitigate the negative effects of overfitting and stabilize the training process. Although the data format of our dataset is almost the same of the images, their semantics is very different and therefore we cannot use standard computer vision data augmentation pipelines. Most of the techniques we are going to present are taken from scientific literature, in these works they have proven to be effective improving performances on fixed benchmark datasets.

4.1 Channel Swapping

The first data augmentation technique we experimented with is very simple and is not directly taken from literature. It consists of randomly swapping a channel of a sample with the same channel of another sample having the same label. For instance if we have a sample a with the label $feet$ we can swap one of its channel, let's say for instance channel 2, with channel 2 of a sample b which still has the label $feet$. This channel swapping is made possible by the channel-wise standardization which makes each channel independent. If we had a standardization which does not treat each channel independently obviously such swapping among channels would not make sense. The whole channel swapping process in only controlled by an hyperparameter, which is the probability of swapping each single channel in a sample, i.e. if we set such probability to 0.1 the probability of swapping channel 1 is 0.1, the probability of swapping channel 2 is 0.1 and so on. This augmentation process is done dynamically through a python generator which takes down the probability that the network see the same sample twice

4.2 Channel Mirroring

Also this technique is very simple and we have not found application of it within the studies about EEG signals, however it can be seen as an adaptation of standard computer vision x-flip augmentation. As the name suggests this technique consists of randomly flipping (along the temporal axis) a channel. Like the channel swapping augmentation such process is controlled by only an hyperparameter which is the probability of mirroring each single channel in a sample. Also this data augmentation is performed dynamically.

4.3 Gaussian Noise

In [11] multiple ways of augmentation based on random noise addiction on EEG data are explored. However EEG signal has a strong randomness and non-stationarity and they have shown that some local noises, such as Poisson noise, Salt noise or Pepper noise change the features of EEG data locally. Based on these considerations in their work they focus on adding Gaussian noise to each feature sample of the original training data to obtain new training samples. In order to ensure that overall amplitude value of the sample will not be changed with the addition of noise, the noise added has zero mean. In their studies they have applied this data augmentation technique to EEG based emotion recognition slightly improving the performances in SEED dataset [12] and in MAHNOB-HCI dataset [13]. In our experiments we have explored various standard deviation value ranging from 1e-3 to 0.1. Like channel mirroring and channel swapping this data augmentation is performed dynamically.

4.4 Gaussian Noise on STFT

In [6] is pointed out that the previous data augmentation based on adding Gaussian noise directly in time domain is not very effective when considering EEG data. In fact considering the characteristic of EEG such as low signal-to-noise ratio, non-stationarity and insufficient spatial resolution, this method may destroy EEG signal's feature in time domain. Therefore they propose a new method which first transforms the EEG recordings to the frequency domain via short-time Fourier transform (STFT), then adds perturbation to amplitudes in frequency domain and finally reconstruct the time-series by inverse STFT.

Let's assume that we are given one training sample X_i and its corresponding label y_i. First we denote the time-series of one channel as $x(t)$. Then we transform it to frequency domain by short time Fourier transform:

$$Z(\tau, f) = \int_{-\infty}^{+\infty} x(t)g(t-\tau)e^{-j2\pi ft}dt \qquad (1)$$

where $Z(\tau, f)$ is a two-dimensional complex matrix representing the phase and amplitude of the signal over time τ and frequency f, and $g(t-\tau)$ is the Hann window function centered around zero. $Z(\tau, f)$ is a complex matrix which can be also represented as $A\cos(\phi) + jA\sin(\phi)$, where $A \in \mathbb{R}^{f\times\tau}$ is the amplitude and $\phi \in \mathbb{R}^{f\times\tau}$ is the phase in every frequency f and time index τ. Then we add Gaussian-distributed noise $p \sim \mathcal{N}(\mu, \sigma^2)$ to the amplitude randomly, where μ is the mean value and σ is the standard deviation of Gaussian noise. Next, we combine the disturbed amplitude and original phase to generate new complex matrix:

$$Z'(\tau, f) = (A+p)\cos\phi + j(A+p)\sin\phi \qquad (2)$$

where $Z'(\tau, f)$ is the new complex matrix, and $A+p$ is the new amplitude after adding Gaussian noise. After that, the new time-series $x'(t)$ is reconstructed by inverse STFT as

$$x'(t) = STFT^{-1}(Z'(\tau, f)) \qquad (3)$$

where $STFT^{-1}$ represents the process of inverse short time Fourier transform. At the end this procedure is applied to each channel in X_i in order to create a new perturbed training sample X'_i. Obviously the new training sample has same label y_i as the origin sample. In their studies they shown that such technique improves performances on BCI Competition IV dataset 2a from 74% accuracy to 76.3% sampling the noise from $p \sim \mathcal{N}(0, 0.001)$. In our experiments we add the noise dynamically to take down the probability that the network see the same sample twice.

4.5 Empirical Mode Decomposition

The empirical mode decomposition algorithm [14] allows users to conduct a data-driven analysis for nonlinear and non-stationary signals. The algorithm decomposes the original signals into a finite number of functions called intrinsic mode functions (IMFs), each of which represents a non-linear oscillation of the signal. For each intrinsic mode function obtained by empirical mode decomposition, each signal fulfills two conditions: (1) The number of maxima is the same as the number of zero-crossing, or differs by at most one. (2) The mean value between the envelope of the local maxima and the envelope of the local minima is zero. The process to obtain the IMFs from a signal $x(t)$ is shown in Algorithm 1.

Once $r_i(t)$ fulfills the terminating condition, the signal can be recovered by adding its IMFs and the final residue $r_n(t)$, where n denotes the number of IMFs of empirical mode decomposition.

$$x(t) = \sum_{k=1}^{n} IMF_k(t) + r_n(t) \tag{4}$$

The structure of the decomposed signal decides the number of IMFs. For our EEG signals, we found that the decomposed signals has at most 6 IMFs. The decomposition process of this method is depicted in figure 11

Algorithm 1: The Algorithmic process of Empirical Mode Decomposition

```
Input: $x(t)$
Output: imf(t)
1  initial $r_0(t) = x(t)$
2  repeat(i=1,2...)
3      repeat(j=1,2...)
4          $h_{j-1}(t) = r_{i-1}(t)$
5          Detect the local maxima and the local
             minima of $h_{j-1}(t)$
6          Interpolate all local maxima to generate
             the upper envelope; and interpolate all
             local minima to generate the lower
             envelope, respectively
7          Obtain the local mean $m_{j-1}(t)$ by
             averaging the upper and lower
             envelopes
8          $h_j(t) = h_{j-1}(t) - m_{j-1}(t)$
9      until $h_j(t)$ satisfies the IMF's conditions
10     $imf_i(t) = h_j(t)$
11     $r_i(t) = r_{i-1}(t) - imf_i(t)$
12 until $r_i(t)$ is a monotonic function or does not
     have enough extrema to calculate the upper
     and lower envelopes
13 return imf(t)
```

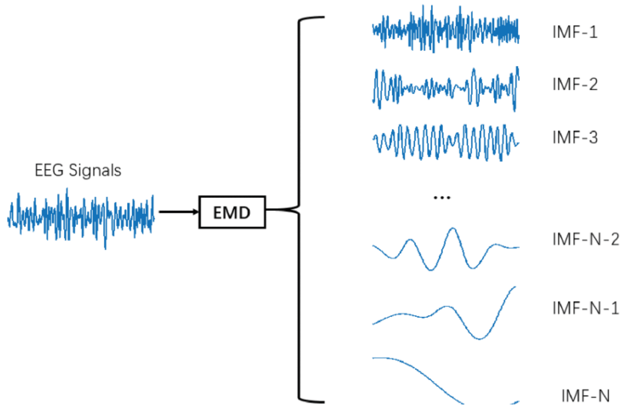

Figure 11: Empirical mode decomposition (EMD) of and EEG signal

In [7] such technique is used to augment motor imagery signals improving performance on their model in BCI competition II dataset III up to a maximum of 10% achieving a new state-of-the-art model. The main idea is to use empirical mode decomposition to generate new frames by swapping IMFs of the decompositions. This will create new EEG frames similar to the original ones but not equal. Moreover, the intrinsic characteristics of each class will be preserved because we will only mix IMFs of the same class when generating new frames of this class. We thereby generated the appropriate ratio of the artificial EEG frames, with the aim of decreasing overfitting problem in a deep learning system, and eventually improved classification results. The generation process can be summarized as follows:

1. Define the number of artificial frames to be create for deep learning approach, meeting the requirements that each class has the same number of artificial EEG frames.

2. Randomly select the frames that contribute with their IMFs to generate the artificial EEG frames. To generate an artificial frame of a specific class, selected EEG frames are split into three sets of EEG frames according to their class.

3. Randomly select EEG frames from the set of frames belonging to the same class. The first selected EEG frame contributes with all its first IMFs (8 IMFs, one IMF for channel), the second one with its second IMFs, and successively until n-th frame, which contributes with its n-th IMFs.

4. Generate a new arifical 8-channel EEG frame by adding up all the IMFs corresponding to the same channel

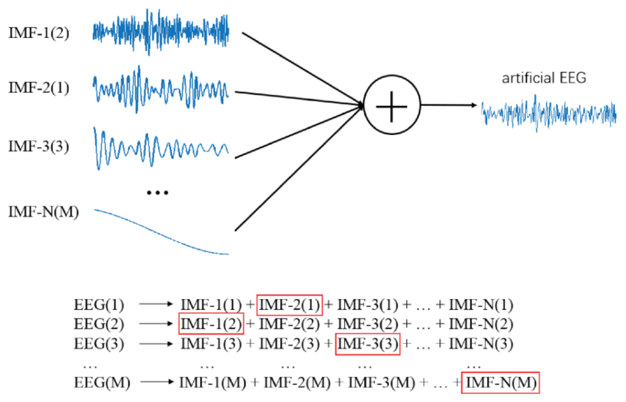
IMF-1(2)
IMF-2(1)
IMF-3(3)
...
IMF-N(M)
artificial EEG
EEG(1) → IMF-1(1) + IMF-2(1) + IMF-3(1) + ... + IMF-N(1)
EEG(2) → IMF-1(2) + IMF-2(2) + IMF-3(2) + ... + IMF-N(2)
EEG(3) → IMF-1(3) + IMF-2(3) + IMF-3(3) + ... + IMF-N(3)
EEG(M) → IMF-1(M) + IMF-2(M) + IMF-3(M) + ... + IMF-N(M)

Figure 12: Mixing modes strategy for creating artificial EEG frames.

As stated above, different EEG signals might generate different numbers of IMFs, so we need to predefine the number of artificial EEG samples which contribute with their IMFs. We set the number of artificial EEG frames as 6, and additional zero value IMFs need to be added in order to reach 6 IMFs for every decomposed signal. A graphical example of this procedure is depicted in figure 12

4.6 Generative Adversarial Networks

The last data augmentation technique we experimented with is based on GAN (Generative Adversarial Networks [15]). This kind of networks has been used for generating EEG synthetic motor-imagery data in [8] and for generating differential-entropy data from EEG in [16]. GANs consist of two competing components which are both parameterized as deep neural networks. Given real data distribution X_r and generated data distribution X_g, the generator G produces realistic-like X_g to confuse the discriminator D, while the discriminator D tries to distinguish whether a sample comes from X_r or X_g. The adversarial training procedure can be formulated as a min-max problem:

$$\begin{aligned}\min_{\theta_G}\max_{\theta_D} L(X_r, X_g) = \mathbb{E}_{x_r\sim X_r}[log(D(x_r))] \\ + \mathbb{E}_{x_g\sim X_g}[log(1-D(x_g))]\end{aligned} \tag{5}$$

where θ_g and θ_d represent the parameters of the generator and discriminator, respectively. And X_g is implicitly defined by $x_g = G(x_z)$, where x_z is sampled from uniform or Gaussian noise distribution.

The adversarial training of traditional GANs can be formalized by minimizing the Jensen-Shannon divergence between the probability distribution of real data and generated data. However, the discontinuity of Jensen-Shannon divergence makes it hard to provide useful gradient for optimizing generator, which is one of the main causes of GANs instability. To eliminate this issue, Jensen-Shannon divergence is replaced with the Earth-Mover distance in Wasserstein GANs [16]:

$$W(X_r, X_g) = \inf_{\gamma\sim\Pi(X_r,X_g)} \mathbb{E}_{(x_r,x_g)\sim\gamma}[||x_r - x_g||] \tag{6}$$

where $\Pi(X_r, X_g)$ denotes all possible joint distributions of real distribution X_r and generated distribution X_g defined in traditional GANs.

The earth-mover distance is continuous and differentiable almost everywhere, and thus can provide meaningful gradients for optimizing generator, which ensures the convergence of the GANs. Since it is difficult to implement the infimum of equation (6) in reality, an alternative approach is to apply Kantorovich-Rubinstein duality of earth-mover distance:

$$\begin{aligned}W(X_r, X_g) = \frac{1}{K}\sup_{||f||_L\le K} \mathbb{E}_{x_r\sim X_r}[f(x_r)] \\ -\mathbb{E}_{x_g\sim X_g}[f(x_g)]\end{aligned} \tag{7}$$

where f denotes the set of 1-Lipschitz functions.

In realistic implementation, f is replaced by discriminator D and $||f||_L \le K$ is replaced by $||D||_L \le 1$. In order to make the training procedure more stable and make convergence faster, [17] enforced Lipschitz constraint with gradient penalty instead of weight clipping to directly constrain the gradient norm. In their approach, an extra penalty term is appended to the loss function:

$$\begin{aligned}\min_{\theta_G}\max_{\theta_D} L(X_r, X_g) = \mathbb{E}_{x_r\sim X_r}[D(x_r)] \\ - \mathbb{E}_{x_g\sim X_g}[D(x_g)] \\ - \lambda\mathbb{E}_{\hat{x}\sim\hat{X}}[(||\nabla_{\hat{x}}D(\hat{x}||_2 - 1)^2]\end{aligned} \tag{8}$$

where λ is a hyperparameter controlling the trade-off between original objective and gradient penalty, and $\hat{x}$ denotes the data points sampled from the straight line between real distribution X_r and generator distribution X_g:

$$\hat{x} = \alpha x_r + (1-\alpha)x_g, \alpha \sim \mathbb{U}[0,1], x_r \sim X_r, x_g \sim X_g \tag{9}$$

In order to generate data with multiple categories, and auxiliary label Y_r is fed into both discriminator and generator. In the generator, we concatenate X_z with Y_r. And in the discriminator, we concatenate both X_r and X_g with Y_r to construct a hidden representation, which controls the categories of generated data. Then the proposed CWGAN can be formulated by:

$$\begin{aligned}\min_{\theta_G}\max_{\theta_D} L(X_r, X_g, Y_r) = \\ \mathbb{E}_{x_r\sim X_r, y_r\sim Y_r}[D(x_r|y_r)] - \mathbb{E}_{x_g\sim X_g, y_r\sim Y_r}[D(x_g|y_r)] \\ - \lambda\mathbb{E}_{\hat{x}\sim\hat{X}, y_r\sim Y_r}[(||\nabla_{\hat{x}|y_r}D(\hat{x}|y_r)||_2 - 1)^2]\end{aligned} \tag{10}$$

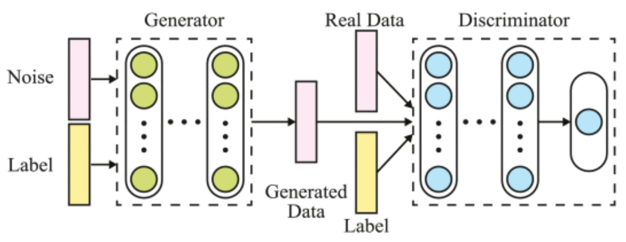

Figure 13: Illustration of the proposed CWGAN

The losses of discriminator (maximum term) and generator (minimum term) are optimized in an alternating procedure. In practice the generator loss contains the second term of equation (10) on the preceding page. And the discriminator loss is updated for *critic* (we set this value to five) times in each adversarial training iteration. The structure of CWGAN is shown in figure 13

In our experiments we notice that a standard conditional GAN had difficulty in converging, more specifically it suffered from mode collapse, which means that the generator model was only capable of generating one or a small subset of different outcomes or modes. In other words they learnt to map several different input X_z values to the same output point. As mentioned above in our experiments we use a conditional Wasserstein GAN with gradient penalty (CWGAN-GP) which is proved to stabilize training and be less sensitive to problems of mode collapse.

Both discriminator and generator structure are inspired to the above mentioned EEGNet. The discriminator is basically identical to EEGNet with minor differences like the usage of Leaky-ReLU as activation function and the output activation function which is linear due to the functioning of WGAN. The generator obviously, having to perform an upsampling operation from a latent space, has different characteristics.

In figure 13 is outlined the complete frameworks of the data augmentation pipeline. It consists of two parts, the Conditional Wasserstein GAN and a quality evaluation of the data. Once trained the conditional WGAN we have to generate high-quality samples discarding the worst samples. Initially the selection criteria we employ during our experiments was based on the discriminator: we generated multiple samples and we selected the ones which have the greater discriminator output (i.e the ones who can trick the discriminator). However this criteria made the generated samples too similar to one another, in fact keeping only the samples that maximally fool the discriminator implies they will tend to be similar. To overcome such problem we used multiple discriminator models, which had been saved

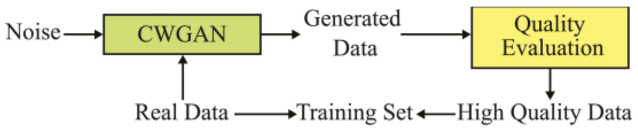

Figure 14: The framework for EEG data augmentation

at different epochs during training and therefore their output will be maximized by different samples.

Our conditional Wasserstein GAN has been trained for 30k epochs, starting from a 50 dimensional latent space with a batch size of 32. As suggested in [17] in each iteration the discriminator is trained five times and the generator only one. Both generator and discriminator have been trained with Adam optimizer with learning rate set to 2e-5, $\beta_1 = 0.5$ and $\beta_2 = 0.9$.

5 Experiments

We conducted exhaustive experiments on all the augmentation techniques above mentioned. As deep learning framework we used Keras [18] with TensorFlow backend [19]. Due to the lack of data, in order to have robust results, a 10-fold cross validation is performed, during each step of the process 70% of the data is used as training set (168 samples), 20% as validation set (48 samples) and 10% as test set (24 samples). The splitting process takes into account the label of the samples, ensuring that each set is perfectly balanced among the three classes, moreover in order to ensure a fair comparison between techniques such process is controlled by a fixed random state which is the same among all the data augmentation tests. The model which has been tested at each k-fold step is the one with the smaller loss on the validation set. As final result of the 10-fold process we take as a result the average of accuracy among the ten test set results. We then repeated the above mentioned pipeline for five times and we took as results the average.

All the models have been trained for a maximum of 10k epochs, Adam optimizer with a learning rate equal to 5e-5 was used and batch size is set to 32. In order to save time in already overfitted models with the help of a Keras callback we stop the training if the model does not improve for 4k consecutive epochs.

In table 1 on the next page are displayed all the already averaged results. From the table we can infer that the majority of data augmentation techniques we experimented with do not bring any performance improvement, on the contrary they contribute to lower the overall accuracy. For channel mirroring and channel swapping augmentation we can notice that as the prob-

	Channel Mirroring					Channel Swapping				
Probability	0.01	0.02	0.05	0.10	0.15	0.01	0.02	0.05	0.10	0.15
Accuracy	67.58	66.33	65.75	63.75	64.08	66.08	67.53	66.33	63.15	62.83
	-0.25	-1.50	-2.08	-4.08	-3.75	-1.75	-0.30	-1.50	-4.68	-5.00

	Gaussian Noise					Gaussian Noise on STFT				
σ	1e-3	5e-3	1e-2	5e-2	1e-1	1e-3	5e-3	1e-2	5e-2	1e-1
Accuracy	67.25	67.83	68.12	66.66	67.25	67.50	67.43	68.67	66.91	65.99
	-0.58	0	+0.29	-1.17	-0.58	-0.33	-0.40	+0.84	-0.92	-1.84

	Empirical Mode Decomposition					GAN[1]				
Ratio	1:0.25	1:0.5	1:1	1:2	1:3	1:0.25	1:0.5	1:1	1:2	1:3
Accuracy	67.08	66.50	63.50	65.75	63.83	68.83*	68.00*	68.41*	68.75*	67.89*
	-0.75	-1.33	-4.33	-2.08	-4.00	+1.00	+0.17	0.58	+0.92	+0.06

Table 1: Classification Accuracy of data augmentation methods in our private dataset (baseline: **67.83**), the red numbers indicate a negative offset with respect to baseline, the green ones a positive offset

ability increases the accuracy decreases, this obviously means that such techniques are not robust enough. From another point of view this can mean that in a sample each channel is not independent and therefore the correlation between the channels are important, these informations are obviously lost if a channel swapping augmentation is performed. We can say the same thing for channel mirroring and the importance of temporal causality. For the noise based augmentation we see that in general they also degrade performance, however the results show that specific σ values can bring a performance improvements. Unlike in [7] the empirical mode decomposition technique does not bring any performance improvement in all the ratio we experimented, but rather we see a consistent drop in accuracy. The only data augmentation approach which brings constant and significant improvement is based on Generative Adversarial Network used to create new synthetic data, such data helps to boost the accuracy up to a percentage point. However, as stated in the footnote, the GAN has been trained on the full amount of data and not only on the training data, thus making the comparison a little bit unfair (we use * in table 1 to underline this fact). This choice was made due to our limited hardware, in fact because of our testing routine consisting of five 10-fold cross validation, in order to use only the training data for each split we should have trained 50 GANs (which would have taken more than three months on our hardware).

6 Conclusions and Future Work

In this work we show that the motor imagery classification problem can be tackled with a convolutional neural network with good results. In order to improve the baseline results we experimented several data augmentation techniques tailored for EEG data. From our experiments we can infer that it is possible to boost classification accuracy using data augmentation but it is not so easy as in computer vision. We show that the channel mirroring and channel swapping techniques degrade the performance, also the augmentation based on empirical mode decomposition [7] has proved to be not effective on our small dataset. The noise based techniques [11] [6] may improve performance but they heavily depend on the σ hyperparameter, the most promising results have been obtained through the generation of synthetic data using GAN. However as formerly said these results obtained using GANs cannot state definitely the goodness of such approach due to the training on the whole dataset.

Firstly as future work we suggest to perform all the experiments linked to the GAN augmentation training this net only on the training data. We also believe that using the above described data augmentation methods can lead to a greater stability in GAN training. Lastly we encourage to experiment with combinations of techniques since in all our experiments they have been used one at a time.

6.1 Resources

Code, complete models and further details available at `https://gitlab.com/ABaldrati/AugmentBrain`

References

[1] Vernon J. Lawhern, Amelia J. Solon, Nicholas R. Waytowich, Stephen M. Gordon, Chou P. Hung, and Brent J. Lance. Eegnet: A compact convolutional network for eeg-based brain-computer interfaces. *CoRR*, abs/1611.08024, 2016.

[1]GAN trained on all available data

[2] Open source brain-computer interfaces. `https://shop.openbci.com/collections/frontpage`. Accessed: 2021-03-31.

[3] Y. Li, X. D. Wang, M. L. Luo, K. Li, X. F. Yang, and Q. Guo. Epileptic seizure classification of eegs using time–frequency analysis based multiscale radial basis functions. *IEEE Journal of Biomedical and Health Informatics*, 22(2):386–397, 2018.

[4] Jörn Munzert, Britta Lorey, and Karen Zentgraf. Cognitive motor processes: The role of motor imagery in the study of motor representations. *Brain Research Reviews*, 60(2):306–326, 2009.

[5] W. S. Anderson and F. A. Lenz. Review of motor and phantom-related imagery. *Neuroreport*, 22(17):939–942, Dec 2011.

[6] Xian-Rui Zhang, Meng-Ying Lei, and Yang Li. An amplitudes-perturbation data augmentation method in convolutional neural networks for eeg decoding, 2018.

[7] Zhiwen Zhang, Feng Duan, Jordi Solé-Casals, Josep Dinarès-Ferran, Andrzej Cichocki, Zhenglu Yang, and Zhe Sun. A novel deep learning approach with data augmentation to classify motor imagery signals. *IEEE Access*, PP:1–1, 01 2019.

[8] K. Zhang, G. Xu, Z. Han, K. Ma, X. Zheng, L. Chen, N. Duan, and S. Zhang. Data Augmentation for Motor Imagery Signal Classification Based on a Hybrid Neural Network. *Sensors (Basel)*, 20(16), Aug 2020.

[9] Brainflow, how bio-senses work. `https://brainflow.org/`. Accessed: 2021-04-12.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[11] Fang Wang, Sheng-hua Zhong, Jianfeng Peng, Jianmin Jiang, and Yan Liu. *Data Augmentation for EEG-Based Emotion Recognition with Deep Convolutional Neural Networks*, pages 82–93. 01 2018.

[12] W. Zheng, W. Liu, Y. Lu, B. Lu, and A. Cichocki. Emotionmeter: A multimodal framework for recognizing human emotions. *IEEE Transactions on Cybernetics*, pages 1–13, 2018.

[13] M. Soleymani, J. Lichtenauer, T. Pun, and M. Pantic. A multimodal database for affect recognition and implicit tagging. *IEEE Transactions on Affective Computing*, 3(1):42–55, 2012.

[14] Norden Huang, Zheng Shen, Steven Long, M.L.C. Wu, Hsing Shih, Quanan Zheng, Nai-Chyuan Yen, Chi-Chao Tung, and Henry Liu. The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454:903–995, 03 1998.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[16] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[18] François Chollet et al. Keras. `https://keras.io`, 2015.

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.