Adversarial Training with Manifold Constraints for EEG Augmentation

Michael Corelli 1938627[1,†], Gianmarco Donnesi 2152311[1,†]

[1]Department of Computer, Control and Management Engineering. Sapienza University of Rome

Abstract

This work presents a framework for the analysis and augmentation of EEG signals, combining generative deep learning techniques with advanced temporal sequence models. The pipeline begins with a detailed preprocessing of EEG data, including band-pass filtering, noisy channel interpolation, epoch segmentation, automatic artifact removal and tangent space projection of SPD covariance matrices. To address data scarcity, we propose a data augmentation approach using a *Wasserstein Generative Adversarial Network* (WGAN-GP), designed to generate synthetic EEG signals that preserve the geometric and temporal properties of the *Riemannian* domain. The augmented dataset, consisting of both real and synthetic sequences, is then processed by a *Temporal Fusion Transformer* (TFT), enhanced with mechanisms such as Positional Encoding, Gated Residual Networks and Multi-Head Attention, to model the spatial and temporal dependencies of EEG signals. Experimental evaluations demonstrate that integrating synthetic data generated by the WGAN-GP, combined with the abstraction capabilities of the TFT, improves classification/regression performance while providing interpretability through attention weight analysis. This approach offers a robust solution for analyzing limited EEG datasets, bridging the gap between the scarcity of real data and the need for high-performance deep learning models.

Keywords

WGAN-GP, TFT, EEG, Riemannian manifold

1. Highlights

- **Advanced EEG Preprocessing**: developed an advanced pipeline integrating artifact removal, channel interpolation, and Riemannian manifold projection, ensuring high-quality and noise-free EEG data for subsequent analysis,
- **Innovative Data Augmentation**: introduced a Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) to generate realistic synthetic EEG signals, effectively mitigating the challenges of data scarcity,
- **Enhanced Dataset Diversity**: combined real and synthetic EEG data to create an enriched training dataset, leading to significant improvements in both classification and regression performance,
- **State-of-the-Art Temporal Modeling**: used the Temporal Fusion Transformer (TFT) to capture intricate spatial and temporal dependencies in EEG data, incorporating attention mechanisms for enhanced model interpretability,
- **Empirical Validation and Scalability**: showed performance gains through extensive experiments, validating the scalability and robustness of the proposed pipeline in data-constrained scenarios.

The full code repository containing all scripts developed for this study is available at this Github Link

2. Introduction

Electroencephalography (EEG) is a widely used non-invasive technique for recording brain activity, offering high temporal resolution that makes it suitable for applications in neuroscience, medicine and machine learning. However, the analysis and modeling of EEG signals present several challenges. These include the intrinsic noise in the data, variability across subjects and the high dimensionality of multichannel signals combined with relatively small datasets. These issues often hinder the performance of modern deep learning models, which require large amounts of data to generalize effectively.

2.1. Challenges in EEG Data Analysis

One of the primary challenges in EEG data analysis is the scarcity of labeled data. Acquiring high-quality, annotated EEG datasets is resource-intensive, requiring specialized equipment, controlled environments and expert knowledge for accurate labeling. This scarcity hampers the training of deep learning models, which generally thrive on large volumes of data to achieve optimal performance and generalization.

Another significant challenge is the presence of *noise* and *artifacts* in EEG recordings. Sources of noise include electrical interference, muscle movements, and eye blinks, which can obscure the underlying neural signals. Preprocessing steps such as **band-pass filtering**, **artifact removal** and **channel interpolation** are essential to enhance data quality. However, these steps must be meticulously executed to preserve the integrity of the neural information while eliminating unwanted perturbations.

Additionally, EEG data is inherently high-dimensional and temporally complex, characterized by multivariate time series with intricate spatial and temporal dependencies. Capturing these dependencies requires sophisticated modeling techniques that can effectively leverage both spatial patterns across multiple electrodes and temporal dynamics over time.

2.2. Importance of Data Augmentation and Advanced Modeling

To address the limitations posed by data scarcity, data augmentation emerges as a crucial strategy. Augmenting EEG datasets with synthetic data can enhance the diversity and volume of training samples, thereby improving model robustness and generalization. Traditional augmentation tech-

[†] These authors contributed equally.

corelli.1938627@studenti.uniroma1.it (M. C. 1938627);
donnesi.2152311@studenti.uniroma1.it (G. D. 2152311)
https://github.com/MichaelCorelli (M. C. 1938627);
https://github.com/GianmarcoDonnesi (G. D. 2152311)

niques, such as adding noise or applying time-shifting, are often simplistic and may not capture the complex statistical properties of EEG signals. Consequently, there is a growing interest in using generative deep learning models, particularly Generative Adversarial Networks (GANs), to produce high-fidelity synthetic EEG data that maintains the intricate spatial and temporal characteristics of real recordings.
Simultaneously, the complexity of EEG data necessitates the use of advanced temporal sequence models capable of capturing long-range dependencies and intricate patterns. Transformer-based architectures, renowned for their efficacy in natural language processing and other sequential data tasks, have shown promise in modeling EEG signals. Specifically, the Temporal Fusion Transformer (TFT) integrates mechanisms such as Positional Encoding, Multi-Head Attention, and Gated Residual Networks to effectively model both spatial and temporal dependencies, offering superior performance in classification and regression tasks.

2.3. Contributions of This Work

This paper presents a pipeline for the analysis and augmentation of EEG signals, integrating generative deep learning techniques with advanced temporal sequence models. This pipeline includes band-pass filtering, interpolation of noisy channels, epoch segmentation, and artifact removal using AutoReject [1]. Additionally, it employs tangent space projection of SPD covariance matrices via PyRiemann to preserve the geometric properties of EEG data while mitigating noise-related distortions.
Building on this foundation, the work integrates a novel data augmentation framework based on a Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP). By generating synthetic EEG signals that maintain both the underlying geometric structures on the Riemannian manifold and the temporal dependencies inherent to neural data, this augmentation strategy enriches the dataset without sacrificing signal integrity. To capture and model these spatial and temporal dependencies, the pipeline incorporates the Temporal Fusion Transformer (TFT). Extensive empirical evaluations confirm that the combination of synthetically generated EEG data and the abstraction power of TFT basically improves performance on classification and regression tasks. Furthermore, the capacity of TFT to highlight relevant neural patterns via attention weights underscores its potential to clarify underlying neural mechanisms. By addressing challenges related to data scarcity and complex spatiotemporal features, our approach provides a solution for analyzing limited EEG datasets. This improvement has particular promise for neuroscientific research and clinical diagnostics.

3. Related Works

The landscape of EEG research has recently benefited from the convergence of generative adversarial networks, Riemannian geometry, and transformer-based temporal modeling. This section surveys foundational and contemporary contributions in generative modeling, manifold learning and temporal sequence processing that inform our approach to synthesizing and analyzing EEG data within an interpretable framework.

3.1. Advancements in EEG Data Synthesis

Generative Adversarial Networks (GANs) have been widely employed to synthesize realistic samples in diverse domains, including EEG. While initial adversarial approaches [2] successfully demonstrated the potential of GANs in EEG data generation, they encountered well-known challenges such as training instability and mode collapse.
Enhancements such as the Wasserstein GAN with Gradient Penalty (WGAN-GP) [3] address these challenges by enforcing a Lipschitz constraint, thereby improving convergence and the fidelity of generated signals. Subsequent studies [4] adapted WGAN-GP for EEG, confirming its ability to produce synthetic data that preserve core spatiotemporal properties of brain activity. These approaches emphasize the necessity of preserving geometric properties, motivating our incorporation of Riemannian manifold constraints to enhance the fidelity and utility of synthetic EEG data.

3.2. Manifold-Based EEG Characterization

EEG signals are naturally represented as high-dimensional covariance matrices on the manifold of Symmetric Positive Definite (SPD) matrices, rendering standard Euclidean methods inadequate for capturing their intrinsic geometry. Pioneering work by Barachant et al. [5] and further developments in [6] established Riemannian geometry and introduces tangent space projections and manifold-based techniques to linearize and analyze SPD matrices. These geometric techniques allow for more accurate characterization of spatial dependencies and have proven crucial in tasks such as mental state classification. Within this paradigm, software libraries like PyRiemann facilitate implementations that preserve the manifold structure, ensuring that both real and augmented EEG data retain their geometric fidelity.

3.3. Transformer Architectures for Sequential EEG

Transformer-based models excel at capturing long-range dependencies and have shown promise in EEG tasks that involve complex temporal relationships. The Temporal Fusion Transformer (TFT) [7], in particular, combines multi-head attention with gated residual networks (GRNs) and positional encoding to capture both short-term and long-term patterns. Its capacity to learn salient features across time and channels has positioned TFT as a powerful interpretive tool in EEG analysis, enabling more transparent classification and regression by revealing which temporal segments and spatial locations drive network predictions. Our approach involves using this model on augmented EEG sequences, benefiting from its ability to integrate synthetic data and deliver interpretable predictions for advancing EEG temporal analysis.

3.4. Assessing Model Fidelity and Transparency

Sound evaluation of synthetic EEG quality often involves adapting traditional metrics such as the Fréchet Inception Distance (FID) or Inception Score (IS) to domain-specific architectures like EEG_net [8].In addition to these, precision and recall, which are complemented by Riemannian distance calculations, provide further insight into the fidelity

and diversity of generated signals. Transformer-based interpretability methods center on the visualization of attention weights and GRN activations, helping to identify influential temporal windows and EEG channels. These methods are employed in our work to validate both the generative and predictive stages, promoting trust in the system and guiding future improvements.

4. Methodology

4.1. Dataset

In order to select an appropriate dataset for our project, we have done research on the open-access platform NEMAR. This platform offers a repository of human neuroelectromagnetic recordings. NEMAR [9] hosts different datasets among which those concerning Electroencephalography (EEG), Magnetoencephalography (MEG) and Intracranial EEG (iEEG). This datasets are different in terms of participant age, number of subjects, and number of recording channels.

4.1.1. General information

The dataset selected for this study is "*200 Objects Infants EEG*" [10] and on the OpenNeuro/NEMAR, identified by the code `ds005106` [11]. The dataset has a single session containing three scans, has 32 EEG channels and both participants and event files are 42.

4.1.2. Motivations of the choice

Accurate and rapid visual recognition of different objects is a fundamental cognitive ability. The initial infant neuroimaging studies have yielded promising results, but there are factors that impede progress in the field, for example it is known that they often suffer from limited stimulus sets, small sample sizes and restricted public access to neural data. Therefore it is necessary to work with high-quality and open-access neuroimaging datasets for infant object recognition. Also, the infant EEG data are often noisier than adult recordings so it is required to use advanced preprocessing pipelines and their development depends critically on the availability of open data.

4.1.3. Specific information

The dataset used in this study is based on electroencephalographic responses from 42 human infants in reaction to visual presentations of various objects. Infants between 2 and 12 months of age were exposed to 200 images depicting 50 distinct objects, presented repeatedly under a rapid serial visual presentation scheme, which is adapted to the infant's comfort in the way that ensures efficient capture of electrophysiological responses to a wide range of visual stimuli. Each image was shown at least three times for each participant. The Figure 1 summarizes key demographic information, including participants'ages in months, the number of experimental blocks completed and a scatterplot represent the relationship between age and blocks completed. The Figure 2 illustrates the rapid serial visual presentation design employed to deliver these stimuli.
The Figure 3 presents the set of stimuli used in the experiment. In the reference cited, a technical validation was performed that can ensure the robustness and utility of the dataset for exploring the neural basis of visual object recognition in early infancy. This demonstrate recognizable neural response paths for both single objects and categorical difference. [10]

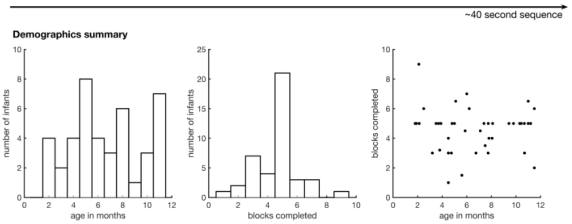

Figure 1: Participant demographics.

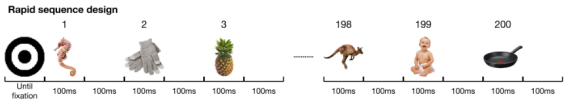

Figure 2: Rapid serial visual presentation design.

Figure 3: Stimulus set.

4.2. Preprocessing Pipeline

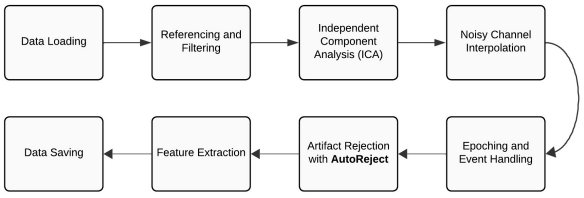

Figure 4: EEG preprocessing pipeline flowchart

The preprocessing of EEG data is a critical step to ensure the integrity and reliability of subsequent analyses. In this study, an advanced preprocessing pipeline was implemented, integrating multiple techniques to enhance signal quality and prepare the data for subsequent modeling stages. This pipeline addresses both spatial and temporal aspects of EEG signals, ensuring a sound approach to data

cleaning and feature extraction.

1. Data Loading
Raw EEG data were acquired in '`.set`' and '`.fdt`' formats, compatible with the EEGLAB software, alongside event files in '`events.tsv`' format that delineate experimental conditions. Using the MNE-Python library, the EEG data were loaded with the '`preload=True`' parameter to facilitate immediate in-memory manipulation.

2. Referencing and Filtering
To minimize background noise and improve the signal-to-noise ratio, a common average reference was applied using the '`set_eeg_reference`' function. Subsequently, the EEG signals underwent band-pass filtering between 0.5 Hz and 40 Hz using a Finite Impulse Response (FIR) filter with a Hamming window (`fir_design='firwin'`). This filtering approach preserves the frequency components of interest while attenuating low-frequency drifts and high-frequency artifacts.

3. Independent Component Analysis (ICA)
To eliminate ocular and muscular artifacts, a rapid ICA was performed with a reduced number of components ('`n_components=0.99`'), balancing computational efficiency with artifact removal efficacy. The ICA was executed without preliminary rejection criteria and the identified independent components were subsequently applied to the EEG data to mitigate artifacts.

4. Noisy Channel Interpolation
Channel variability was assessed by computing the variance of each EEG channel. Channels with variances falling below the 1st percentile or above the 99th percentile were flagged as noisy. Identified noisy channels were interpolated using data from surrounding channels to maintain spatial consistency across the EEG scalp.

5. Epoching and Event Handling
EEG data were segmented into epochs ranging from 0 to 0.5 seconds relative to valid events ('`stimnum`' ranging from 1 to 200). Epoching was conducted without baseline correction to preserve the initial dynamic characteristics of the EEG signals. Events lacking valid 'stimnum' entries or falling outside the specified range were excluded to ensure dataset integrity.

6. Artifact Rejection with AutoReject
To further enhance data quality, the AutoReject algorithm was employed. This method utilizes cross-validation to automatically detect and correct epochs containing artifacts. AutoReject effectively preserves a substantial number of clean epochs, thereby minimizing the loss of valuable data while ensuring robustness against noisy segments.

7. Feature Extraction via Covariance Matrices and Tangent Space Projection
Feature extraction involved calculating Symmetric Positive Definite (SPD) covariance matrices for each clean epoch using the Oracle Approximating Shrinkage (OAS) estimator. These covariance matrices were then projected into the tangent space using the Tangent Space Projection technique provided by PyRiemann. This projection reduces dimensionality and facilitates geometric analysis of the EEG data.

8. Riemannian Manifold
$\mathbb{M}$ is differentiable manifold of dimension G and $T_C\mathbb{M}$ denote the tangent space of $\mathbb{M}$ at a point $C \in \mathbb{M}$. Consider two tangent vectors $T_1, T_2 \in T_C\mathbb{M}$, where the inner product at C is defined by:

$$\langle T_1, T_2 \rangle_C = \mathrm{Tr}(T_1 C^{-1} T_2 C^{-1}),$$

and $\mathrm{Tr}(\cdot)$ indicates the trace of a matrix. The inner product lead to a norm for whatever tangent vector $T \in T_C\mathbb{M}$ given by:

$$\|T\|_C = \sqrt{\langle T, T \rangle_C} = \sqrt{\mathrm{Tr}(TC^{-1}TC^{-1})}.$$

The logarithmic map project a point $C' \in \mathbb{M}$ on the tangent space $T_C\mathbb{M}$, in this way it produces a tangent vector T'. That is:

$$T' = \log_C(C') = C^{\frac{1}{2}} \log\left(C^{-\frac{1}{2}} C' C^{-\frac{1}{2}}\right) C^{\frac{1}{2}}.$$

While the exponential map projects a tangent vector $T' \in T_C\mathbb{M}$ on the manifold, so it result in a point $C' \in \mathbb{M}$:

$$C' = \exp_C(T') = C^{\frac{1}{2}} \exp\left(C^{-\frac{1}{2}} T' C^{-\frac{1}{2}}\right) C^{\frac{1}{2}},$$

$\log(\cdot)$ and $\exp(\cdot)$ indicates the matrix logarithm and exponential functions.
The Riemannian distance $\delta_R(C, C')$, between two points $C, C' \in \mathbb{M}$, is defined as the length of the shortest path that connect the two points on the manifold. Defining the distance in terms of the tangent vector got through the logarithmic map:

$$\delta_R(C, C') = \|\log_C(C')\|_C = \|T'\|_C.$$

Demonstrates the evaluation of the distance between two SPD matrices C and C' on a Riemannian manifold, through the use of the tangent space learning technique:

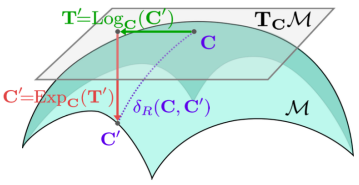

Figure 5: The Concept of Riemannian Manifold

For our project have considered two ways in order to obtain more information: the PyRiemann library and the implementation present in the `riemannian_manifold.py` file. The first choice allowed us to get the following information:

Listing 1: Data loading output using functions from the PyRiemann library

```
[INFO] Loaded ts_features shape:
    (4724, 528)
[INFO] Loaded labels shape: (4724,)
[INFO] Loaded subjects shape:
    (4724,)
ts_features shape = (4724, 528)
labels shape = (4724,)
subjects shape = (4724,)
```

```
channels = ['Fp1' 'Fz' 'F3' 'F7' '
    F9' 'FC5' 'FC1' 'C3' 'T7' 'CP5'
    'CP1' 'Pz' 'P3'
 'P7' 'P9' 'O1' 'Oz' 'O2' 'P10' 'P8
     ' 'P4' 'CP2' 'CP6' 'T8' 'C4' '
     Cz' 'FC2'
 'FC6' 'F10' 'F8' 'F4' 'Fp2']
```

while through the use of the `riemannian_manifold.py` file it was obtained:

Listing 2: Data loading output using riemannian_manifold script

```
[INFO] Loaded ts_features shape:
    (4715, 300)
[INFO] Loaded labels shape: (4715,)
[INFO] Loaded subjects shape:
    (4715,)
ts_features shape = (4715, 300)
labels shape = (4715,)
subjects shape = (4715,)
channels = ['Fp1' 'Fz' 'F3' 'F7' '
    F9' 'FC5' 'FC1' 'C3' 'T7' 'CP5'
    'CP1' 'Pz' 'P3'
 'P7' 'P9' 'O1' 'Oz' 'O2' 'P10' 'P8
     ' 'P4' 'CP2' 'CP6' 'T8' 'C4' '
     Cz' 'FC2'
 'FC6' 'F10' 'F8' 'F4' 'Fp2']
```

Then was decided to adapt the code in order to use the library, because is possible to get more information with it. [12, 13]

9. Data Saving

Preprocessed data, including tangent space-projected features, event labels, channel information, and temporal metadata, were saved in '`.npz`' format. This format enables efficient storage and rapid retrieval of data for subsequent modeling phases.

To optimize preprocessing time and handle large-scale EEG datasets efficiently, the pipeline was parallelized using the **Joblib** library. By enabling simultaneous processing of multiple subjects across multi-core CPUs, this approach significantly reduced computational time and ensured scalability. Computationally intensive tasks, such as artifact removal with AutoReject and the calculation of SPD covariance matrices, greatly benefited from this strategy.

The framework is robust and fault-tolerant, logging errors and handling scenarios where certain subjects may have incomplete or corrupted data. This design is ideal for large datasets, such as those from multi-site or longitudinal studies, allowing for efficient integration of complex preprocessing techniques while maintaining high throughput and reliability.

4.3. Data Augmentation with WGAN-GP

Generative Adversarial Networks are used in generative applications, in particular for image synthesis, but their applications have been expanded to include time series data. In this section explore the application of Wasserstein GAN with Gradient Penalty in the domain of signal generation, demonstrating their effectiveness in augmenting temporal data.

4.3.1. GANs for signals

EEG has a fundamental role in recording brain activity and is an important part of the development of brain-computer interface technologies. The problem is that have limited availability and high variability of EEG signals, so take on different challenges in the creation of reliable BCIs. The most machine learning is about decoding information from real world data, but our goal is to generate such data. Artificial data generation can be used to augment training data through the production of natural looking samples that were not included in the original dataset.

4.3.2. Choice of GAN

To identify a type of GAN that is suitable for our case, were considered different choices. The first was to use vanilla GANs, but suffer from instability during training and are limited to low resolution images, even if has been made progress for the stability and quality of the generated images through regularization methods and by progressively increasing the resolution of the images during training.
Another tipe considered was WaveGAN that is used with time series data generation, the problem is that this type of GAN was not correct to generate naturalistic samples of EEG data. Finally, the chosen architecture for this study is the WGAN, in particular was selected Wasserstein GANs GP that introduce a modification to the improved training of Wasserstein GANs to stabilize the training using the gradient penalty.
It is possible to observe the difference of our choice from the classic GAN's architecture [Figure 6], which is composed of two opposing networks: generator and discriminator. The generator network is designed to produce synthetic samples that mimic the distribution of real data, while the discriminator network is trained to accurately distinguish between real data and previously generated synthetic samples.

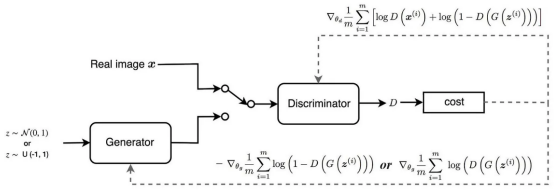

Figure 6: Standard GAN architecture [14].

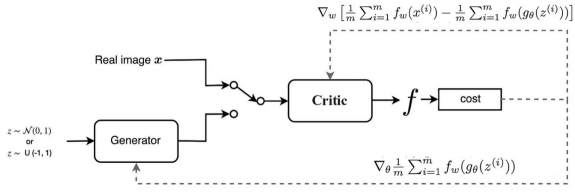

Figure 7: Wasserstein GAN architecture [14].

Using GANs [6] must be careful about the instability of the discriminator during training, because the discriminator

may fail by identifying only some narrow modes of the input distribution as real, which would lead the generator to create few different outputs.
Respect to GANs can achieve stable GAN training using Wasserstein GANs [7].
However, WGANs generate bad samples or fail to converge, which are due to the use of weight clipping in WGANs to impose a Lipschitz constraint on the critic.
The original GANs tried to minimize the JS, Jensen-Shannon, divergence between the real data distribution $\mathbb{P}_r$ and the fake data distribution $\mathbb{P}_\theta$.
If the discriminator is trained to optimality, this can lead to the problem of vanishing gradients for the generator.
Another proposal is to minimize the Wasserstein distance between the distributions instead of the JS divergence, this leads the discriminator, called critical, to maximize the difference:

$$\tilde{W}(\mathbb{P}_r, \mathbb{P}_\theta) = E_{x_r \sim \mathbb{P}_r}[D(x_r)] - E_{x_f \sim \mathbb{P}_\theta}[D(x_f)]$$

and the generator to maximize $E_{x_f \sim \mathbb{P}_\theta}[D(x_f)]$.
The critic provides a useful gradient for the generator everywhere if $D(x)$ is K-Lipschitz. The solution has as an additional term, the gradient penalty:

$$\mathbb{P}_2(\mathbb{P}_{\hat{x}}) = \lambda \cdot E_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2]$$

with $\mathbb{P}_{\hat{x}}$ containing the points lying on a straight line between the real and generated samples, at the critical loss.
When training WGAN-GP the choice of λ is crucial, because if choose it too high then the penalty term can control the distance term, while if choose it too small the Lipschitz continuity is not sufficiently enhanced.
This model performs better than standard WGANs allowing stable training of GAN architectures.

4.4. WGAN-GP Architecture

The Wasserstein Generative Adversarial Network with Gradient Penalty [3, 2, 4, 15] is structured in two principal components a Generator and a Discriminator, also called the Critic. Both networks use the LeakyReLU activation function with a negative slope coefficient of $\alpha = 0.2$, which helps mitigate the issue of vanishing neurons, common in standard ReLU activations, in order to ensure more stable gradient flow during training.
The Discriminator network is built using 1D convolutional layers, fully connected layers and dropout for regularization. It contains 64 hidden channels with each convolutional layer configured to use a kernel size of 2, stride of 1 and zero padding. To prevent overfitting we utilized Dropout layers with a probability of $p = 0.2$. Moreover the Discriminator does not utilize batch normalization, as this can sometimes destabilize GAN training dynamics.
Instead the Generator network features 100 hidden channels in order to capture the complexity required for synthesizing high-fidelity EEG signals. Similar to the Discriminator, it employs 1D convolutional layers and linear layers with dropout $p = 0.2$, but it applies batch normalization via 'BatchNorm1d' to stabilize training. The convolutional layers in the Generator also use a kernel size of 2, stride of 1 and no padding. Furthermore the Generator incorporates shortcut, residual, connections using convolutional layers with a kernel size of 1, stride of 1, and no padding, that facilitate improved gradient flow and information conservation through layers.
To manage resolution changes without introducing unwanted frequency artifacts, the network uses linear interpolation for upsampling and average pooling for downsampling. The Generator takes noise vectors from a latent space of dimension 300, with each element of the vector sampled from a Gaussian distribution $\mathcal{N}(0, 0.02)$. Both the Generator and Discriminator weights are initialized using a Gaussian distribution with mean 0 and standard deviation 0.02, which are fundamental to achieve stable convergence. The critic generally learns more rapidly than the generator, so want that the training regimen updates the critic five times for every single update of the generator. This way helps to keep the equilibrium between the two networks, in order to ensure the critic provides meaningful feedback. A gradient penalty coefficient $\lambda = 50$ is used to enforce the 1-Lipschitz constraint on the critic by penalizing deviations from a unit gradient norm. For optimization is utilized the Adam optimizer with a learning rate of 1×10^{-4} and momentum parameters $\beta_1 = 0.5$ and $\beta_2 = 0.9$, which enhance stability in GAN training scenarios.

4.5. Temporal Fusion Transformer (TFT) for EEG Modeling

The Temporal Fusion Transformer (TFT) is an advanced deep learning model designed to capture complex temporal dependencies within sequential data.

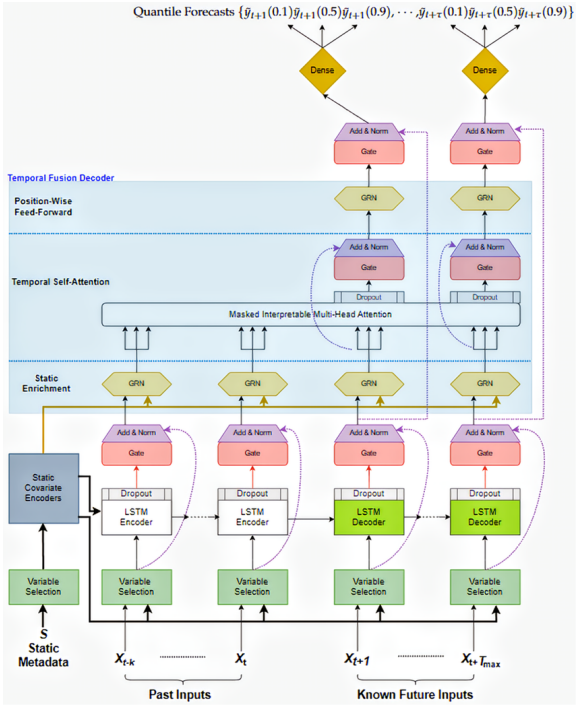

Figure 8: Architecture of the Temporal Fusion Transformer, illustrating its key components and mechanisms for modeling complex temporal dependencies in EEG data [16].

In the context of EEG analysis, this specific architecture was selected for its proficiency in modeling long-range temporal relationships and its adaptability in integrating various attention mechanisms and gated residual networks.

4.6. Model Architecture

4.6.1. Encoder

In order to capture both short-range and long-range dependencies characteristic of EEG signals, the encoder integrates a multi-layer Long Short-Term Memory (LSTM) network with positional encoding, multi-head self-attention, and Gated Residual Networks (GRNs). Specifically, the encoder consists of two stacked LSTM layers, each containing 64 hidden units, to represent hierarchical temporal structures in the data.
To retain the sequential ordering, a sinusoidal positional encoding is applied to each input timestep:

$$\begin{aligned} \mathrm{PE}_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d}}\right), \\ \mathrm{PE}_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d}}\right) \end{aligned} \tag{1}$$

where pos is the position and i is the dimension index in the embedding space of size d. This ensures that the sequential nature of the EEG data is preserved and effectively used during processing.
After positional encoding, the encoder employs multi-head self-attention with eight parallel attention heads to simultaneously focus on multiple aspects of the input sequence. Given input $X \in \mathbb{R}^{T\times d}$ (where T is the sequence length and d the embedding dimension), the model computes linear projections $Q = X\,W^Q$, $K = X\,W^K$ and $V = X\,W^V$. In particular $W^Q, W^K, W^V \in \mathbb{R}^{d\times d_k}$ are learnable weights and d_k is the dimension of the queries and keys. The attention output is then computed as:

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\Big(\frac{Q\,K^T}{\sqrt{d_k}}\Big)\,V. \tag{2}$$

For the multi-head version, h attention heads are computed in parallel and concatenated to form the final representation:

$$\mathrm{MHA}(Q,K,V) = \mathrm{concat}\big(\mathrm{head}_1,\ldots,\mathrm{head}_h\big)\,W^O, \tag{3}$$

where each $head_i = Attention(Q\,W_i^Q,\ K\,W_i^K,\ V\,W_i^V)$ and W^O is a learnable projection weight.
At this point GRNs are employed to enhance gradient flow and and mitigate vanishing or exploding gradients.The output of a GRN is computed as:

$$\mathrm{GRN}(x) = \mathrm{Gate}(x)\odot f(x) + (1-\mathrm{Gate}(x))\odot x \tag{4}$$

where $Gate(x) = \sigma(W_g x + b_g)$ and

$$f(x) = \mathrm{ELU}\left(W_2\left(\mathrm{Dropout}(\mathrm{ELU}(W_1 x + b_1))\right) + b_2\right) \tag{5}$$

By combining these layers, the encoder is able to learn complex non-linear transformations while preserving stable gradient behavior.

4.6.2. Decoder

The decoder translates the temporal representations learned by the encoder into task-specific outputs. It consists of a single LSTM layer that initializes its hidden state with the final hidden state produced by the encoder, thus taking advantage of the temporal context extracted upstream. A subsequent Gated Residual Network (GRN) feedforward layer refines the decoder's output, enhancing the capacity to capture higher-order relationships in the data. Finally, a fully connected projection layer maps these refined representations into the desired output space, ensuring adaptability to diverse downstream objectives like classification or regression tasks.

4.6.3. Training and Optimization

Training follows a supervised learning paradigm with task-specific objective functions: **cross-entropy loss** for classification or **mean squared error** (MSE) for regression. The Adam optimizer is used with an initial learning rate of 1×10^{-4} and $\beta = (0.5, 0.9)$. A batch size of 32 balances computational considerations with stable gradient updates. Over 20 training epochs, performance is evaluated by task-relevant metrics, such as accuracy and F1-score for classification or MSE for regression. Training runs on GPU-accelerated hardware within the PyTorch framework, offering efficient experimentation and hyperparameter tuning.

4.6.4. Integration of Synthetic Data

A distinctive aspect of this study is the augmentation of the training dataset with synthetic EEG data generated using a Wasserstein GAN with Gradient Penalty (WGAN-GP). This augmentation strategy not only increases the diversity of the training samples but also ensures the preservation of the geometric and temporal characteristics inherent in real EEG signals. By enriching the dataset with high-quality synthetic data, the model's generalization capabilities were significantly improved, particularly in scenarios with limited real-world data.

4.6.5. Performance Evaluation

The TFT's performance was evaluated using metrics aligned with the specific objectives of the study. For **classification** tasks, were used metrics like F1-score to assess the model's ability to correctly identify EEG events across different conditions. Instead, for **regression** tasks were employed metrics such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) to evaluate the precision of continuous predictions.
To conclude, the Temporal Fusion Transformer in joint with WGAN-GP based synthetic data underscores the efficacy of advanced neural architectures combined with generative augmentation for EEG analysis, particularly in data-constrained scenarios. This integrated framework demonstrates adaptability and predictive performance, highlighting its potential for wide-ranging applications in neuroscientific research and clinical diagnostics.

5. Experimental Setup

5.1. Data Preparation

The foundation of our experimental framework lies in the preparation of EEG data, ensuring that the subsequent modeling stages with both the Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) and the Temporal Fusion Transformer (TFT) operate on high-quality, well-structured inputs. This process encompasses several

key steps, ranging from raw data acquisition and preprocessing to dataset consolidation and transformation tailored for specific modeling task.

5.1.1. Raw Data Acquisition and Initial Preprocessing

Our dataset comprises EEG recordings from 42 human infants, sourced from the open-access NEMAR [9] repository (dataset ID: ds005106 [11]). Each subject's data includes recordings from 32 EEG channels, captured in response to visual stimuli presented in a rapid serial visual presentation paradigm. The raw data are stored in EEGLAB-compatible formats (`.set` and `.fdt`) alongside event files in `.tsv` format, delineating experimental conditions.
The preprocessing pipeline, implemented in `preprocessing_data.py`, begins with loading the raw EEG data using the MNE-Python library. For each subject, the raw data undergo a series of transformations to enhance signal quality and extract meaningful features. Initially, a common average reference is applied to reduce background noise, followed by band-pass filtering within the 0.5 Hz to 40 Hz range using a Finite Impulse Response (FIR) filter with a Hamming window. This filtering step retains the frequency components pertinent to neural activity while attenuating low-frequency drifts and high-frequency artifacts.

5.1.2. Artifact Removal and Channel Interpolation

To mitigate ocular and muscular artifacts inherent in EEG recordings, a fast Independent Component Analysis (ICA) is performed with a reduced number of components (`n_components=0.99`). This approach balances computational efficiency with effective artifact removal. Subsequent to ICA, channels exhibiting anomalously low or high variance specifically those falling below the 1st percentile or above the 99th percentile of variance distribution are identified as noisy. These bad channels are then interpolated using data from neighboring channels to maintain spatial consistency across the EEG scalp.

5.1.3. Epoching and Automatic Artifact Rejection

The cleaned continuous EEG data are segmented into epochs ranging from 0 to 0.5 seconds relative to valid stimulus events (`stimnum` ranging from 1 to 200). This epoching process is crucial for isolating event-related potentials and ensuring that the data fed into subsequent models are temporally aligned with experimental conditions. To further enhance data quality, the `AutoReject` algorithm is employed to automatically detect and correct or discard epochs containing residual artifacts. This automated approach ensures the retention of a maximal number of clean epochs, thereby preserving valuable neural information while eliminating corrupted segments.

5.1.4. Feature Extraction: Covariance Matrices and Tangent Space Projection

Each clean epoch is transformed into a Symmetric Positive Definite (SPD) covariance matrix using the Oracle Approximating Shrinkage (OAS) estimator, implemented via the `Covariances` class from the PyRiemann library. These covariance matrices encapsulate the spatial dependencies across EEG channels, providing a robust representation of neural connectivity patterns. To facilitate geometric analysis and compatibility with standard machine learning techniques, the SPD matrices are projected into the tangent space using the `TangentSpace` transform. This projection linearizes the manifold of SPD matrices, enabling the application of conventional Euclidean-based models.

5.1.5. Consolidation of Preprocessed Data

The `load_preprocessed_data.py` script consolidates the tangent-space features, labels, and associated metadata from all subjects into a unified structure. Specifically, it aggregates the feature arrays, labels, subject identifiers, channel labels, and temporal information into a single dictionary. The consolidated dataset includes `ts_features`, which is stored as a two-dimensional array of shape (N, D), where N denotes the total number of epochs and D represents the dimensionality of the tangent-space embeddings; `labels`, containing either integer or categorical labels for each epoch; `subjects`, indicating the subject identifier aligned with each epoch; `chan_labels`, listing the EEG channel names used; `times`, providing the time points defined during epoching; and `onsets_sec`, recording the onset times for each epoch relative to the raw EEG data.

5.1.6. Preparation of Data for WGAN-GP

The `prepare_wgan_data.py` script transforms the consolidated dataset into a format optimized for WGAN-GP training. It begins by partitioning the tangent-space features and labels into training and test subsets in an 80% to 20% ratio through stratified splitting, ensuring balanced class distributions. From each feature vector, the first 512 elements are then selected and reshaped into a three-dimensional array of size $(n\_channels, n\_frames)$. A z-score normalization is subsequently applied according to the mean and standard deviation derived from the training set. This normalization enforces stable statistical properties, which is particularly crucial for adversarial training. Following this transformation, the normalized data are wrapped in an `EEGDatasetWGAN` class and loaded into a dataloader to enable efficient batch-wise iteration, with a standard batch size of 64 and shuffling enabled. To facilitate future reference or validation, both training and test data can be saved in `.npz` format within the `derivatives/preprocessing` directory.

5.1.7. Preparation of Data for Temporal Fusion Transformer (TFT)

In parallel to the WGAN-GP preparation, the `prepare_tft_data.py` script organizes the unified dataset for training the TFT. The process starts with extracting a subset of features corresponding to a fixed number of channels and frames (e.g., 32 channels by 16 frames). To ensure statistical consistency, the selected data undergo z-score normalization, standardizing the features to have zero mean and unit variance. The normalized arrays are then reshaped into three-dimensional structures $(n\_samples, n\_channels, n\_frames)$ to facilitate subsequent segmentation into temporal sequences.

Before segmentation, multiple data augmentation strategies are applied to broaden the variety of training samples. *Gaussian noise* is incorporated to replicate the sensor variability encountered in realistic EEG recordings, whereas

time warping is employed to simulate variations in signal speed and latency. In addition, *channel-wise amplitude scaling* accounts for differences in electrode impedance or recording conditions, and *random channel permutations* enforce spatial invariance by diversifying the relative positions of electrodes. Each augmented version of the original data is combined with the baseline dataset, and all corresponding labels are replicated to match the expanded pool of samples.

Once the labels are numerically encoded using scikit-learn label encoder, the preprocessed and augmented data are converted into tensors. An instance of a custom dataset class, `EEGSequenceDatasetTFT`, is instantiated to produce the overlapping temporal sequences required by the TFT. Within this class, each continuous EEG sequence is segmented over a defined number of frames, with the label mapped to the final segment in that sequence. All sequences are finally wrapped into a dataloader configured with an appropriate batch size, random shuffling, and parallel data loading based on the number of available worker processes. To manage the extensive volume of EEG data and the computational complexity inherent in this pipeline, parallel execution is engineered via the *joblib* library.

5.2. Models Training

5.2.1. WGAN-GP Training

To approximate and minimize the Earth Mover Distance between the real and fake feature distributions, is possible to apply the Wasserstein loss to the model loss function.
If want utilize the distance of the two distributions as the loss score does not need that the critic produce a bounded score between `[-1,1]`, which helps to reduce the vanishing gradient and mode collapse problem.
The gradient penalty is an important regularization term that enforces the critic to be 1-Lipschitz continuous. To get more stable results constrain the loss function to be Lipschitz. The latter constrains the loss function to be Lipschitz, so that get more stable results. The loss function is as follows:

$$\begin{aligned} L =& E_{x\sim\mathbb{P}_g} \sim [C(G(z))] - E_{x\sim\mathbb{P}_r}[C(x)] + \\ & \lambda E_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla C(\hat{x})||_2 - 1)^2] \\ \min_G \max_C & E_{x\sim\mathbb{P}_g} \sim [C(G(z))] - E_{x\sim\mathbb{P}_r}[C(x)] \end{aligned} \tag{6}$$

The critic wants to maximize the average cost E(C(x)) for distributing real values, while the generator wants to minimize the average cost E(G(x)) for giving false values.
Progressively the critic learn to evaluate real and false samples and it aims is return a smaller Wasserstein distance for the real samples x and a larger Wasserstein distance for the generated samples $\tilde{x}$. Through the subtraction of the two loss scores can minimize the loss function L.
When the gradient norm of the critic is greater than 1, then the loss function is penalized by the gradient penalty term, in this way is applied the Lipschitz constraint.
The interpolated samples $\hat{x}$ were used to compute the regularization term, moreover the value $\hat{x}$ represents an intermediate sample between real and fake samples and is represented as:

$$\hat{x} = \epsilon x + (1-\epsilon)G(z)$$

where ϵ indicates a random number used to choose a random point in the space between real and fake data. Sample the noise vector z from a Gaussian distribution, in a similar way to traditional WGAN training, then insert it into the generator and transform it into fake samples that resemble to the real data. Now the critic evaluates the fake sample and the generator utilizes the critic's feedback to produce more realistic fake samples. These steps are repeated until the generator output is no longer distinguishable from the real data.
In the following pseudocode is illustrated the WGAN training algorithm with gradient penalty:

Algorithm 1 WGAN-GP training algorithm [3]

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: Initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, \dots, n_{\text{critic}}$ **do**
    for $i = 1, \dots, m$ **do**
      Sample real data $x \sim \mathbb{P}_r$, latent variable $z \sim p(z)$, a random number $\epsilon \sim \mathcal{U}[0,1]$.
      $\tilde{x} \leftarrow G_\theta(z)$
      $\hat{x} \leftarrow \epsilon x + (1-\epsilon)\tilde{x}$
      $L^{(i)} \leftarrow D_w(\tilde{x}) - D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{z^{(i)}\}_{i=1}^m \sim p(z)$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2)$
end while

Our ultimate aim is to trick the critic into believing that the generator's output is real data. [4, 15]

5.2.2. TFT Training

The training phase of the Temporal Fusion Transformer (TFT), implemented in the `tft_model.py` script, is explained in details below in this section.

The network architecture features a Gated Residual Network (GRN) for handling non-linear transformations with skip connections, a simple feed-forward component inspired by Transformer blocks, and a multi-head attention module that captures both local and global dependencies in the data. A positional encoding mechanism augments the input representations in the encoder, introducing explicit information on the relative ordering of time steps.

The encoder, constructed in `TFTEncoder`, processes sequences with LSTM layers followed by multi-head attention and feed-forward transformations. A sequence of `TFTEncoderLayer` blocks refines these representations, while gated residual connections improve gradient flow and stability. Subsequently, the `TFTDecoder` transforms the encoded features into predictions. It uses a single-layer LSTM initialized by the final hidden state of the encoder, followed by a GRN feed-forward layer for additional non-linear processing and a linear projection to the desired output space.

Model training and validation is carried out by the `train_tft` function, which initializes the loss function according to the designated task (classification or regression).

The Adam optimizer, configured with a specific learning rate and weight decay, updates model parameters. Regularization methods such as dropout (with a probability of 0.2) are also applied to mitigate overfitting. Then the use of `OneCycleLR` scheduler moderates the learning rate over training epochs. Mixed-precision scaling, provided by `torch.amp.GradScaler`, speeds up training and stabilizes gradient calculations, while gradient clipping avoids exploding gradients. Parameter updates occur after gradient scaling and progress is tracked using `tqdm` for real-time feedback on loss and accuracy.

Validation data are supplied to monitor generalization after each epoch. If validation loss declines, the current model state is saved; if no improvement is observed for a predefined patience window, early stopping is invoked. Finally, the `train_tft` function logs each training epoch to facilitate post-training analysis and debugging.

5.3. Evaluation Metrics

In this work, evaluation metrics address two key objectives: assessing the quality of synthetic data generated by the Wasserstein GAN with Gradient Penalty (WGAN-GP) and evaluating the Temporal Fusion Transformer (TFT) in modeling EEG signal dependencies.
For WGAN-GP, the metrics were adapted to measure the fidelity and diversity of synthetic EEG data relative to real data, while regarding the metrics used to evaluate the Transformer, they were chosen based on the evaluation of the classification or regression performance of the model.

5.3.1. WGAN-GP Performance Evaluation

Overview of metrics
GANs can be complex to evaluate and the most popular metrics are in the computer vision domain. There is still some work in progress to understand how to properly evaluate time series GANs, anyway the evaluation metrics can be divided into *qualitative* and *quantitative*.
The first one refers to human visual evaluation by inspecting samples generated by GANs, but cannot be considered a evaluation of the performance of GANs due to the lack of a suitable objective evaluation metric. While the second one also uses metrics associated with statistical measures utilized for time series analysis and similarity measures. These last kind of metrics are commonly used for time series evaluation, since they are able to reflect the stability between the training data and the generated synthetic data. So they are suitable as performance metrics for GANs.
Several metrics for evaluating image-based GANs are well established and among them some have also been introduced in sequential and time-series GANs. We can remember the Fréchet Inception Distance and Inception Score as metrics.

Metrics used
In our project were used the following metrics: Riemannian Distance [17] to measure geometric quantities such as lengths of curves, angles between vectors, and distances between points on the manifold, Real vs Fake Accuracy to evaluate how well a classifier can distinguish between real data samples and synthetic data samples generated, the Fréchet Inception Distance indicates how closely the generated images resemble real images based on their statistical properties:

$$FID = \|\mu_{real} - \mu_{generated}\|^2 + Tr(\sum_{real} + \sum_{generated} - 2(\sum_{real}\sum_{generated})^{\frac{1}{2}}), \quad (7)$$

Inception Score compares the generated to real images analyzing only the first, Kernel Inception Distance is more robust for smaller datasets, it uses Maximum Mean Discrepancy with a polynomial kernel making it less biased for data limited scenarios, Precision tells us how well the generated images align with the distribution of real images and Recall measures the diversity within the generated images indicating how much of the distribution of real images they cover.
In this case are dealing with signals, not with images, so for some of the metrics employed, instead of using a pre-trained model, need to use a neural network, named EEG_net, which allows us to work with signals. [8, 18, 19]

5.3.2. Temporal Fusion Transformer Performance Evaluation

The performance evaluation of the Temporal Fusion Transformer (TFT) was structured to assess its predictive capabilities across classification and regression tasks, emphasizing generalization to unseen data and robustness under varying conditions. The evaluation methodology begins loading the pre-trained TFT model into evaluation mode to deactivate training-specific layers such as dropout to ensure deterministic inference. The test dataset was subsequently segmented into overlapping temporal sequences using a sliding window approach, configured with a default sequence length of 10 time steps and batched for parallelized processing. Predictions were generated for each batch, with classification tasks using argmax over logits and regression tasks directly outputting continuous values. Performance metrics, including accuracy and F1-scores were computed using scikit-learn utilities to quantify overall correctness, class imbalance handling and per-class prediction dynamics. The evaluation was executed on GPU-accelerated hardware (different NVIDIA GPUs via Google Colab Pro) to optimize computational efficiency, particularly for large-scale EEG data. The validate_tft function orchestrated the entire workflow, from model loading and data preparation to prediction generation and metric computation, ensuring a streamlined and reproducible assessment pipeline.
The evaluation process of the TFT performance is designed to assess the model's predictive capabilities in both classification and regression tasks, ensuring that it generalizes well to unseen data and maintains robustness across various scenarios.

5.4. Implementation Details

This subsection highlights the most relevant technical aspects of our EEG analysis pipeline, referencing specific code snippets to illustrate key implementation choices. The overall architecture integrates data preprocessing, generative model training (WGAN-GP), temporal modeling (TFT) and metric-based evaluation.

WGAN-GP for Data Augmentation
To generate high-fidelity synthetic EEG signals, a WGAN-GP was implemented, focusing on temporal coherence

and geometric fidelity. The gradient penalty enforces a 1-Lipschitz constraint on the critic, improving training stability. A core function in the discriminatore's training loop demonstrates how the penalty is computed:

Listing 3: Gradient penalty snippet in the discriminator

```
def gp_loss(self, gp_scale,
    real_signals, fake_signals):

        N, _, L = real_signals.shape
        device = real_signals.device

        a = torch.rand(N, 1)
        a = a.expand(N, int(
            real_signals.nelement() / N)
            ).contiguous()
        a = a.view(N, 32, L)
        a = a.to(device)

        i = a * real_signals.detach() +
             ((1 - a) * fake_signals.
            detach())
        i = i.to(device)
        i.requires_grad_(True)

        disc_i = self.
            forward_discriminator(i)
        grad = autograd.grad(outputs=
            disc_i, inputs=i,
            grad_outputs=torch.ones(
            disc_i.size()).to(device),
            create_graph = True,
            retain_graph = True,
            only_inputs = True)[0]
        grad = grad.view(grad.size(0),
            -1)

        gp = ((grad.norm(2, dim=1) - 1)
            **2).mean() * gp_scale

        return gp
```

By enforcing the Lipschitz constraint in this manner, mode collapse is mitigated and the generator learns to produce diverse synthetic signals that align with the real data distribution.

Here, gradient clipping prevents gradient explosion in complex architectures, while adaptive optimizers (e.g., Adam) foster stable convergence. This routine also supports dynamic learning rate scheduling (OneCycleLR) to further optimize training efficiency.

5.4.1. Data Augmentation Techniques

Gaussian Noise Addition

A fundamental data augmentation strategy involves introducing Gaussian noise to EEG signals, thereby simulating real-world noise conditions and enhancing model robustness. Given an EEG signal $\mathbf{x}$, Gaussian noise $\boldsymbol{\eta}$ with mean $\mu = 0.0$ and standard deviation $\sigma = 0.01$ is generated and added to the signal to obtain the noisy signal $\mathbf{x}_{\text{noisy}}$:

$$\mathbf{x}_{\text{noisy}} = \mathbf{x} + \boldsymbol{\eta}, \quad \boldsymbol{\eta} \sim \mathcal{N}(\mu, \sigma^2)$$

The corresponding Python function encapsulates this idea by using NumPy's random number generation capabilities to create noise with the specified parameters and adding it

Listing 4: Gaussian noise function

```
def add_gaussian_noise(data, mean=0.0,
    std=0.01):
    noise = np.random.normal(mean, std,
         data.shape)
    return data + noise
```

This process ensures that the model learns to be resilient to such perturbations.

Time Warping

Temporal variability in EEG signals, arising from neural processing speed differences or experimental conditions, is addressed via time warping. This technique non-linearly stretches or compresses the time axis by sampling a random warping factor f from $\mathcal{U}(0.8, 1.2)$. Time indices are recalculated as $indices = clip(round\left(\frac{\text{range}(0,T)}{f}\right), 0, T-1)$, where T is the original length. The signal is resampled using these indices and the output is adjusted to match the original length through truncation or zero-padding. The implementation in the listing below ensures consistent dimensionality while simulating temporal distortions.

Listing 5: Time warping function

```
def time_warp(data, factor_range=(0.8,
    1.2)):
    factor = np.random.uniform(*
        factor_range)
    indices = np.round(np.arange(0,
        data.shape[1], factor)).astype(
        int)
    indices = np.clip(indices, 0, data.
        shape[1] - 1)
    warped_data = data[:, indices]

    M = warped_data.shape[1]
    target_frames = data.shape[1]

    if M > target_frames:
        start = np.random.randint(0, M
            - target_frames + 1)
        warped_data = warped_data[:,
            start : start +
            target_frames]
    elif M < target_frames:
        pad_width = target_frames - M
        warped_data = np.pad(
            warped_data,
            pad_width=((0, 0), (0,
                pad_width)),
            mode='constant'
        )
    return warped_data
```

Scaling

Amplitude variability across EEG recordings, due to factors like electrode contact quality, is mimicked via channel-wise scaling. Each channel is multiplied by a random factor s_i sampled from an uniform distribution $\mathcal{U}(0.9, 1.1)$, resulting in a scaled signal $\mathbf{x}_{i,\text{scaled}} = s_i \cdot \mathbf{x}_i$. This encourages the model to focus on underlying patterns rather than absolute amplitudes.

Listing 6: Scaling function

```
def scale_data(data, scale_range=(0.9,
```

```
    1.1)):
    scale = np.random.uniform(*
        scale_range, size=(data.shape
        [0], 1))
    return data * scale
```

Channel Permutation
To mitigate over-reliance on specific channel configurations, channels are randomly permuted during training. This technique challenges the model to extract spatially invariant features, enhancing robustness to electrode placement variations. The implementation in the snippet of code below generates a random permutation of channel indices and reorders the data accordingly, ensuring that each channel contributes equally to the learning process.

Listing 7: Channel Permutation

```
def permute_channels(data):
    perm = np.random.permutation(data.
        shape[0])
    return data[perm, :]
```

6. Results

6.1. WGAN-GP Performance

In this section we analyze the performance of WGAN-GP, in order to understand if the choices made have led to significant results.
The first analysis we address concerns the Generator and Discriminator architecture during the training phase.
The Table 1 shows the results obtained from the training using Dropout.

Ep	Disc. Err	Disc. Err GP	Gen. Err	*lr* D	*lr* G
1	-0.0022	0.3099	0.0009	0.0001	0.0001
2	-0.0013	0.3124	-0.0006	0.0001	0.0001
3	-0.0044	0.2953	-0.0036	0.0001	0.0001
4	-0.0055	0.3040	-0.0082	0.0001	0.0001
5	-0.0101	0.2769	-0.0118	0.0001	0.0001
6	-0.0106	0.3421	-0.0167	0.0001	0.0001
7	-0.0102	0.2552	-0.0198	0.0001	0.0001
8	-0.0146	0.3108	-0.0239	0.0000	0.0000
9	-0.0243	0.4447	-0.0250	0.0000	0.0000
10	-0.0232	0.7029	-0.0267	0.0000	0.0000
11	-0.0254	0.6438	-0.0265	0.0000	0.0000
12	-0.0194	0.7530	-0.0297	0.0000	0.0000
13	-0.0279	0.7120	-0.0294	0.0000	0.0000
14	-0.0304	0.6356	-0.0312	0.0000	0.0000
15	-0.0292	0.8103	-0.0327	0.0000	0.0000
16	-0.0351	0.9206	-0.0329	0.0000	0.0000

Table 1
Training Results for WGAN-GP using `Dropout(0,2)`

As the epochs increase, we can observe a continuous decrease in the errors of the Discriminator and Generator. The GP error of the Discriminator oscillates between increase and decrease, but generally tends to grows. While for the two Learning rates we observe a transition from 0.0001 to 0 at epoch 8. The Table 2 shows the results obtained from the training without using Dropout. In this second case, we can always observe that as the epochs increase, there is a continuous decrease in the errors of the Discriminator and the Generator, better than the first model. Unlike the previous case, in this one the GP error of the Discriminator oscillates more than the previous model between increase and decrease, but also in this case it tends to increase. While for the two Learning rates we always observe a passage from 0.0001 to 0 at epoch 8.

Ep	Disc. Err	Disc. Err GP	Gen. Err	*lr* D	*lr* G
1	0.0005	0.1017	-0.0017	0.0001	0.0001
2	-0.0015	0.3401	-0.0082	0.0001	0.0001
3	-0.0084	0.2552	-0.0044	0.0001	0.0001
4	-0.0097	0.1459	-0.0356	0.0001	0.0001
5	-0.0138	0.5341	-0.1387	0.0001	0.0001
6	-0.0133	0.3314	-0.2246	0.0001	0.0001
7	-0.0257	0.6019	-0.2830	0.0001	0.0001
8	-0.0451	0.3785	-0.3457	0.0000	0.0000
9	-0.0559	0.5781	-0.4128	0.0000	0.0000
10	-0.0741	0.8696	-0.4993	0.0000	0.0000
11	-0.0928	0.1581	-0.5680	0.0000	0.0000
12	-0.1117	0.2509	-0.6426	0.0000	0.0000
13	-0.1322	0.9290	-0.6949	0.0000	0.0000
14	-0.1582	0.2276	-0.7346	0.0000	0.0000
15	-0.1646	0.1692	-0.7595	0.0000	0.0000
16	-0.1703	0.4189	-0.7682	0.0000	0.0000

Table 2
Training Results for WGAN-GP without using `Dropout`

The second analysis shows the results of the evaluation metrics used. The outcomes listed in Tables 3 and 4 indicate that the model trained without Dropout attains a lower Riemannian Distance, suggesting that it yields synthetic EEG signals more closely resembling the statistical structure of the real recordings. The reduced Real vs Fake Accuracy in the Dropout-trained variant, on the other hand, implies a heightened similarity to the original data, thereby making accurate classification more challenging and hinting at improved generative fidelity under dropout. Examining the Fréchet Inception Distance shows a near-zero score for the non-dropout model, underscoring minimal discrepancy between the synthetic and real datasets in feature space. The Inception Score in the model without dropout is marginally higher, though the gap indicates only a slight edge in the diversity or quality of the generated samples relative to the dropout model. In line with expectations for a comparatively small dataset, the Kernel Inception Distance also confirms a strong agreement with real data in the non-dropout case, validating the generator's ability to capture the underlying data distribution. Both models achieve Precision and Recall scores of 1.0, reflecting that the generated data effectively spans and matches the full distribution of the real samples.

Metric	Value
Riemannian Distance (RD)	3.57×10^{-15}
Real vs Fake Accuracy	0.3
Frèchet Inception Distance (FID)	0.1927
Inception Score (Mean)	1.0515
Inception Score (Std)	1.28×10^{-4}
Kernel Inception Distance (KID)	0.0508
Precision	1.0
Recall	1.0

Table 3
Validation Results for WGAN-GP with Dropout

Metric	Value
Riemannian Distance (RD)	2.85×10^{-15}
Real vs Fake Accuracy	0.4
Frèchet Inception Distance (FID)	0.1528
Inception Score (Mean)	1.1284
Inception Score (Std)	8.12×10^{-5}
Kernel Inception Distance (KID)	0.0325
Precision	1.0
Recall	1.0

Table 4
Validation Results for WGAN-GP without Dropout

6.2. TFT Performance

The training progress of the Temporal Fusion Transformer (TFT) is shown in Figure 10 for accuracy and in Figure 9 for loss.

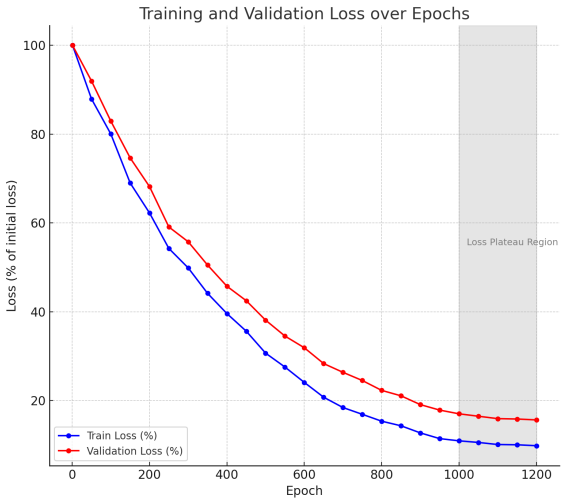

Figure 9: Training and validation loss over epochs for the TFT model. The loss is normalized as a percentage of the initial value.

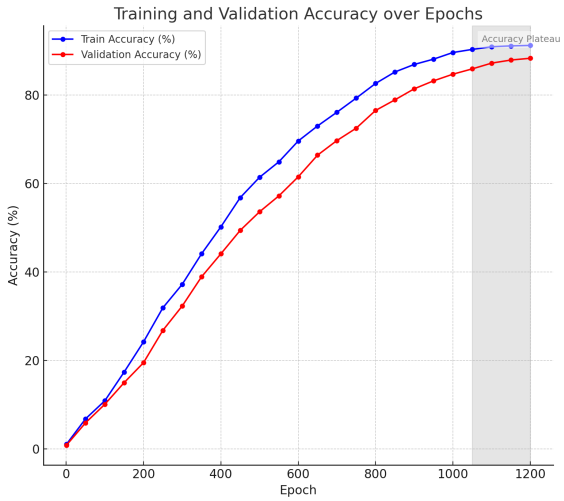

Figure 10: Training and validation accuracy over epochs for the TFT model.

Over the course of 1200 epochs, the training loss steadily decreases and eventually approaches a near-constant minimum, reflecting a stable optimization procedure. This smooth convergence is attributed to careful hyperparameter tuning, including the use of a moderate initial learning rate, combined with a `OneCycleLR` scheduler that stabilizes learning-rate transitions, a momentum that prevent abrupt shifts in parameter updates and the use of dropout to mitigate overfitting.
The validation loss follows a similar downward trend, indicating that the learned representations generalize satisfactorily to unseen samples. After approximately 1000 epochs, the improvement rate begins to slow, and the training accuracy converges into a *plateau* around 88-89%. Despite this plateau, the validation accuracy progresses in parallel and this overlap in the later stages of training is an indication of consistent generalization.
To improve the training process of the model, it was implemented an early stopping mechanism to abort the process when a plateau is reached. In particular, in our case this mechanism wasn't triggered because the validation loss continued to improve until it reached a plateau in a range of epochs under the patience threshold, in a way that the training process ended without being interrupted. Overall, the loss and accuracy curves confirm that the TFT successfully models the complex temporal patterns in the EEG dataset and maintains a strong balance between efficient optimization on the training set and sound generalization to new samples.

6.3. Interpretability Analysis

In this section we analyze the interpretability of both models, a critical aspect of our pipeline to provide insight of the underlying neural processes.

6.3.1. WGAN-GP Interpretability

While generative models like WGAN-GP are often considered "black boxes," understanding how they generate synthetic EEG data can lend credibility to their outputs and ensure that the synthetic data maintains the essential characteristics of real EEG signals.

Latent Space Exploration
One approach to interpret the WGAN-GP model is to analyze its latent space. The generator maps noise vectors z from a latent space to synthetic EEG samples. By interpolating between different points in the latent space and observing the corresponding generated signals, we can assess how smoothly the generator transitions between different EEG patterns. This interpolation can reveal whether the model has learned a meaningful structure that captures variations in EEG data, such as changes in amplitude, frequency components and temporal dynamics.
The latent space interpolation Figure 11 demonstrates the model's ability to smoothly transition between two synthetic EEG signals and to produce continuous, realistic EEG patterns that retain essential features like rhythmic oscillations and amplitude variations. The consistent quality of these transitions highlights the model's robustness and potential for applications such as studying gradual changes in brain states or generating realistic intermediate signals for various scenarios.

Feature Distribution Comparison
Another interpretability tool involves comparing the statistical properties of real and synthetic EEG data. By visualizing distributions of key features it is possible to validate whether

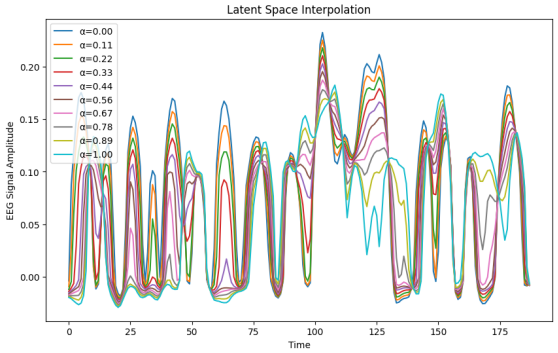

Figure 11: Latent space interpolation demonstrating the generation of smoothly varying synthetic EEG signals by interpolating between two distinct latent vectors.

the generator preserves critical characteristics of the EEG signals.

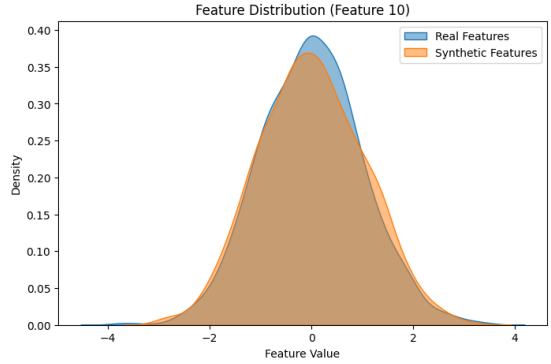

Figure 12: Comparison of feature distributions between real and synthetic EEG signals.

From the Figure 12 we can notice that the synthetic feature distribution aligns closely with the real data, showcasing the model's ability to learn the underlying data distribution effectively. This significant overlap between the two distributions indicates that the generative model generalizes well to the real data space, that is critical for ensuring that synthetic data reflects realistic characteristics.

6.4. Visualization of Synthetic Data

To provide a clearer understanding of the data used and processed by the TFT, we now present visual comparisons between real EEG data from our dataset and synthetic data generated by the WGAN-GP in the frequency domain.
In Figure 13, we see that the real EEG signal exhibits slightly higher amplitudes at certain frequency bands (e.g., around 0.2 Hz), indicating a more pronounced variability in those ranges. In contrast, the synthetic signal is generally smoother and stays below the real EEG amplitude at most frequencies. This behavior suggests that the generative model succeeds in capturing the broad spectral structure of the EEG but tends to underestimate the strongest peaks. The overall shape of the synthetic spectrum is consistent with the real one, implying that the WGAN-GP is effectively learning key features of the underlying distribution.

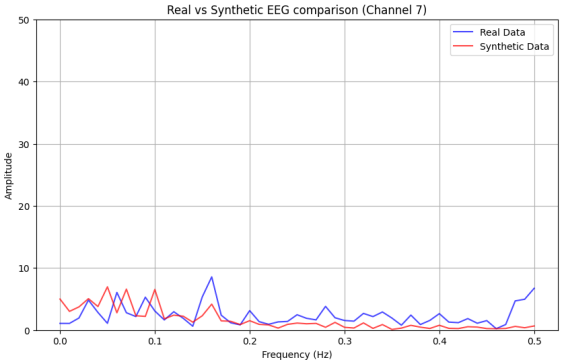

Figure 13: Comparison between Real and Synthetic data in the frequency domain

7. Conclusions

This study presents a processing flow for electroencephalogram (EEG) analysis that integrates advanced preprocessing techniques, generative models incorporating Riemannian geometric constraints and time series modeling. The process employs the Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) to generate synthetic EEG signals that preserve the spatial and geometric properties of the real data. Synthetic data can reflect the true record structure while reducing noise instability through the use of Riemannian geometry. These signals are processed through a Time Fusion Transformer (TFT) that models spatio-temporal dependencies through multi-head attention, position encoding, and gated residual networks. The results show that this process enhances model robustness, reduces overfitting, and improves accuracy, especially in the presence of data scarcity. Comparative analysis of the WGAN-GP configurations (with and without Dropout) shows that the Dropout-free model performs better in terms of Riemann Distance, FID, and KID, despite greater oscillations in the gradient penalty error of the discriminator. Both models achieve perfect precision and recall, indicating strong agreement between the synthetic and real data distributions. Combining WGAN-GP with TFT addresses the problem of scarce EEG data and provides opportunities for interpretability through the attention mechanism. This process provides a sound framework for advancing EEG analysis, bridging the gap between data enhancement, temporal modeling and practical applications.

7.1. Advantages of the Proposed Pipeline

The proposed pipeline brings several key advantages to EEG analysis. First, it enables **robust data augmentation**, using a WGAN-GP with Riemannian constraints to generate high-quality synthetic data that enriches the training set, effectively addressing the common issue of limited labeled EEG data. In addition to augmentation, the pipeline features comprehensive preprocessing techniques to ensure that EEG data fed into subsequent models are cleaned up, noise-reduced, and geometrically informative, which is critical for reliable analysis.
Furthermore, the use of a **Temporal Fusion Transformer** allowed to capture complex temporal dependencies in EEG signals through the use of attention mechanisms and gated residual networks, leading to improved performance and

enhanced interpretability. The incorporation of gradient penalties within the GAN framework, along with carefully designed training strategies for the TFT, contributes to stable training processes that reduces the risk of overfitting but also enhances the generalization capabilities of the models. Finally, the exploration of the generative model's latent space, combined with the attention mechanisms in the TFT, opens pathways to understand the underlying patterns and decision-making processes within the data.

7.2. Limitations

The proposed pipeline has some limitations, in particular it may face computational and storage challenges when applied to substantially larger datasets or real-time applications. Moreover, the complexity of combined WGAN-GP and TFT architectures demands significant computational resources and expertise for hyperparameter tuning, which may limit broader adoption. Finally, the approach has been validated on a specific EEG dataset (infant visual responses). Its generalizability to other EEG paradigms or neurological conditions needs further exploration.

7.3. Future Work

Building on the current work, future research can focus on several key areas. One promising direction is **enhanced interpretability**: by implementing and visualizing detailed interpretability techniques for the TFT such as extracting and analyzing attention weights and gated residual network activations researchers can further explicate the model's decision process. This deeper insight into how the model operates would not only build trust but also guide refinements in design and application.

Another area for exploration is the generation of **synthetic data applicable in real-life situations**. Incorporating controlled noise into the synthetic data generation process could better simulate various recording conditions, thereby enhancing the model's robustness to real-world artifacts. By accurately modeling these conditions, the synthetic data would provide more realistic training scenarios and improve the overall resilience of the system.

Research can also aim at **scalability testing**. Evaluating the pipeline on larger, more diverse EEG datasets will test its scalability, stability and performance across different domains and patient populations.

Further, streamlining the pipeline for real-time applications or integrating it into brain-computer interface (BCI) systems could extend its utility beyond offline analysis, making the technology more accessible and applicable in practical scenarios where immediate feedback is crucial.

Finally, adapting and validating the models on related time-series data such as MEG or fMRI, or even datasets outside neuroscience could greatly expand the applicability of the techniques developed in this study across various research fields.

References

[1] M. Jas, D. Engemann, Y. Bekhti, F. Raimondo, A. Gramfort, Autoreject: Automated artifact rejection for MEG and EEG data, NeuroImage 159 (2017) 417–429. URL: https://autoreject.github.io/stable/index.html. doi:10.1016/j.neuroimage.2017.07.064.

[2] K. G. Hartmann, R. T. Schirrmeister, T. Ball, EEG-GAN: Generative adversarial networks for electroencephalograhic (EEG) brain signals, https://arxiv.org/abs/1806.01875, 2018. arXiv:1806.01875.

[3] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville, Improved Training of Wasserstein GANs, https://arxiv.org/abs/1704.00028, 2017. arXiv:1704.00028.

[4] J. Park, P. Mahey, O. Adeniyi, Improving EEG Signal Classification Accuracy Using Wasserstein Generative Adversarial Networks, https://arxiv.org/abs/2402.09453, 2024. arXiv:2402.09453.

[5] A. Barachant, S. Bonnet, M. Congedo, C. Jutten, Multiclass Brain-Computer Interface Classification by Riemannian Geometry, IEEE transactions on biomedical engineering 59 (2011) 920–8. doi:10.1109/TBME.2011.2172210.

[6] F. Yger, M. Berar, F. Lotte, Riemannian Approaches in Brain-Computer Interfaces: A Review, IEEE Transactions on Neural System and Rehabilitation Engineering PP (2017). doi:10.1109/TNSRE.2016.2627016.

[7] B. Lim, S. O. Arik, N. Loeff, T. Pfister, Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting, https://arxiv.org/abs/1912.09363, 2020. arXiv:1912.09363.

[8] E. Brophy, Z. Wang, Q. She, T. Ward, Generative adversarial networks in time series: A survey and taxonomy, https://arxiv.org/abs/2107.11098, 2021. arXiv:2107.11098.

[9] NEMAR: NeuroElectroMagnetic Data Archive and Tools Resource, https://nemar.org, 2022.

[10] Grootswagers, Tijl, G. Quek, Z. Zeng, M. Varlet, Human Infant EEG Recordings for 200 Object Images Presented in Rapid Visual Streams, PsyArXiv. April 26. https://doi.org/10.31234/osf.io/dzrkq, 2024.

[11] T. Grootswagers, G. Quek, Z. Zeng, M. Varlet, ds005106, https://nemar.org/dataexplorer/detail?dataset_id=ds005106, 2024-04-26 05:35:07.

[12] G. Zhang, A. Etemad, Spatio-Temporal EEG Representation Learning on Riemannian Manifold and Euclidean Space, IEEE Transactions on Emerging Topics in Computational Intelligence 8 (2024) 1469–1483. doi:10.1109/tetci.2023.3332549.

[13] Spatio-Temporal EEG Representation Learning on Riemannian Manifold and Euclidean Space (TF v1.14.0), https://github.com/guangyizhangbci/EEG_Riemannian, 2022.

[14] J. Hui, GAN — Wasserstein GAN and WGAN-GP, site, 2018.

[15] Improving EEG Signal Availability using Deep Learning and Generative Adversarial Networks, https://github.com/JoshParkSJ/eeg-wgan, 2023.

[16] A. Metin, A. Kaşif, C. Catal, Temporal fusion transformer-based prediction in aquaponics, The Journal of Supercomputing 79 (2023) 1–25. doi:10.1007/s11227-023-05389-8.

[17] Encyclopedia of Mathematics, Riemannian Metric, https://encyclopediaofmath.org/wiki/Riemannian_metric, 2023. Accessed: 2025-01-10.

[18] CNN-EEG: Applying Convolutional Neural Networks to EEG signal Analysis, https://github.com/CNN-for-EEG-classification/CNN-EEG, 2024.

[19] H. Amit, PyTorch Implementation of Common GAN

Metrics, site, 2024. Accessed: 23/12/2024.

Contents

1 Highlights **1**

2 Introduction **1**
2.1 Challenges in EEG Data Analysis 1
2.2 Importance of Data Augmentation and Advanced Modeling 1
2.3 Contributions of This Work 2

3 Related Works **2**
3.1 Advancements in EEG Data Synthesis . . . 2
3.2 Manifold-Based EEG Characterization . . . 2
3.3 Transformer Architectures for Sequential EEG 2
3.4 Assessing Model Fidelity and Transparency 2

4 Methodology **3**
4.1 Dataset 3
4.1.1 General information 3
4.1.2 Motivations of the choice 3
4.1.3 Specific information 3
4.2 Preprocessing Pipeline 3
4.3 Data Augmentation with WGAN-GP 5
4.3.1 GANs for signals 5
4.3.2 Choice of GAN 5
4.4 WGAN-GP Architecture 6
4.5 Temporal Fusion Transformer (TFT) for EEG Modeling 6
4.6 Model Architecture 7
4.6.1 Encoder 7
4.6.2 Decoder 7
4.6.3 Training and Optimization 7
4.6.4 Integration of Synthetic Data 7
4.6.5 Performance Evaluation 7

5 Experimental Setup **7**
5.1 Data Preparation 7
5.1.1 Raw Data Acquisition and Initial Preprocessing 8
5.1.2 Artifact Removal and Channel Interpolation 8
5.1.3 Epoching and Automatic Artifact Rejection 8
5.1.4 Feature Extraction: Covariance Matrices and Tangent Space Projection 8
5.1.5 Consolidation of Preprocessed Data 8
5.1.6 Preparation of Data for WGAN-GP . 8
5.1.7 Preparation of Data for Temporal Fusion Transformer (TFT) 8
5.2 Models Training 9
5.2.1 WGAN-GP Training 9
5.2.2 TFT Training 9
5.3 Evaluation Metrics 10
5.3.1 WGAN-GP Performance Evaluation 10
5.3.2 Temporal Fusion Transformer Performance Evaluation 10
5.4 Implementation Details 10
5.4.1 Data Augmentation Techniques . . 11

6 Results **12**
6.1 WGAN-GP Performance 12
6.2 TFT Performance 13
6.3 Interpretability Analysis 13
6.3.1 WGAN-GP Interpretability 13
6.4 Visualization of Synthetic Data 14

7 Conclusions **14**
7.1 Advantages of the Proposed Pipeline 14
7.2 Limitations 15
7.3 Future Work 15

List of Figures

1 Participant demographics. 3
2 Rapid serial visual presentation design. . . . 3
3 Stimulus set. 3
4 EEG preprocessing pipeline flowchart . . . 3
5 The Concept of Riemannian Manifold . . . 4
6 Standard GAN architecture [14]. 5
7 Wasserstein GAN architecture [14]. 5
8 Architecture of the Temporal Fusion Transformer, illustrating its key components and mechanisms for modeling complex temporal dependencies in EEG data [16]. 6
9 Training and validation loss over epochs for the TFT model. The loss is normalized as a percentage of the initial value. 13
10 Training and validation accuracy over epochs for the TFT model. 13
11 Latent space interpolation demonstrating the generation of smoothly varying synthetic EEG signals by interpolating between two distinct latent vectors. 14
12 Comparison of feature distributions between real and synthetic EEG signals. . . . 14
13 Comparison between Real and Synthetic data in the frequency domain 14

List of Tables

1 Training Results for WGAN-GP using `Dropout(0,2)` 12
2 Training Results for WGAN-GP without using `Dropout` 12
3 Validation Results for WGAN-GP with Dropout 12
4 Validation Results for WGAN-GP without Dropout 13

Listings

1 Data loading output using functions from the PyRiemann library 4
2 Data loading output using riemannian_manifold script 5
3 Gradient penalty snippet in the discriminator 11
4 Gaussian noise function 11
5 Time warping function 11
6 Scaling function 11
7 Channel Permutation 12