Generating EEG signals using Generative Adversarial Networks

Hamlin Liu
Computer Science
hamlinliu250@g.ucla.edu

Arjun Kallapur
Computer Science
arjun.kallapur@gmail.com

Daniel Smith
Computer Science
couthelloworld@g.ucla.edu

Utsav Munendra
Computer Science
utsavm9@g.ucla.edu

Abstract

In this project, we aim to create and test different generative adversarial network architectures. We will test the multiple different architectures used to create the generator and discriminator networks by applying them to electroencephalogram (EEG) data. The data consists of 22 channels taken at 1000 time steps, and have 4 task labels. We investigate different types of generative adversarial network (GAN) architectures as well as network architectures to determine the performance of our various generators. Given the very difficult process of training GANs, we were only able to generate data that could be classified accurately $\approx 25\%$ *of the time by a classifier that could classify real data accurately 67% of the time.*

1. Introduction

EEG data can be used to provide insight into the inner workings of neural systems. Due to the non-invasive nature of the EEG procedure, it allows for easy access of measuring signals but a lot of noise also gets added in the process [1]. Ideally, we can create models to decipher EEG signals and extract useful features which can aid in understanding more of the human nervous system and brain. Being able to generate these EEG signals from a generative network would prove very valuable to the field, as generative models will be able to augment current classification by providing new unseen data to certain classifiers. Also, by being able to produce EEG data of certain classes, the generative models might also provide certain insight on the original data distribution that was used to train it.

A framework that focuses on optimizing generative networks are generative adversarial networks was first proposed by Goodfellow et al. [6]. This framework proved to be very effective at generating images and other forms of sequential data such as audio signals [3]. GANs have also been shown to be effective at generating EEG signals utilizing regular convolutional neural networks (CNNs) [7]. This project will focus on the feasibility of using student resources to apply the GAN frameworks for creating classes of EEG signals. We are also trying to see if the application of different network architectures other than CNNs, such as recurrent neural networks (RNNs), can take advantage of the temporal relation of the signal.

2. Methods

We discuss the different methods and architectures used to increase the performance of the GAN.

2.1. Dataset and Data Preprocessing

The EEG dataset used for training consisted of 2115 trials. Each trial had 22 channels and 1000 timestamps at which values were measured. The test dataset contained 443 trials [2]. Two main preprocessing methods we used were trimming and subsampling. From visual inspection, roughly the final 400 timesteps were noise and were hence trimmed. The remaining data was subsampled at every 6th timestep in order to reduce noise, for a final sequence length of 100 time stamps. This number was chosen empirically after our initial generative networks failed to produce meaningful samples of length 1000 and 600. This preprocessing is visualized in Appendix B, Figure 9.

2.2. Architectures

There are multiple frameworks to train GANs. In this project, we focused on a framework that followed the general layout of GANs but allowed us to control the class type of the generated data. In a regular vanilla GAN, there are two neural networks, a generator and discriminator, that are working as adversaries, competing in a minimax game. The generator network takes in a random noise vector and outputs new instances of data by trying to learn the original

data distribution. The discriminator network will try to classify real training samples as real and generated instances as fake. The objective of the generator is to generate samples that cannot be distinguished by the discriminator. Thus these two networks will be competing in a minimax game until they reach a Nash equilibrium [5].

However, because vanilla GANs cannot control the type of data that is being generated, we modified the framework to be similar to a conditional generational adversarial networks (CGANs) [9]. In our framework, the generator is conditioned on the label or class of the data instance that needs to be generated. Using this class label, we are then able to train the generator to focus on generating certain classes. A layout of the two architectures can be seen in Figure 10 in Appendix B. To determine what architecture works best, we also varied the types of layers that composed the generators and discriminators as well as the input noise dimensions. All the model architectures tested in our framework are described in Appendix A. By default, most of the networks used a rectified linear unit (ReLU) ($f(x) = x$ when $x > 0$ and $f(x) = 0$ when $x < 0$) as their activations.

We designed a set of generators first around the idea that the EEG signals are temporally related. Thus, some generators and discriminators that were tested used only recurrent neural networks (RNN) layers to take advantage of this temporal relation. In the case of the generators, following Hyland et al., we inputted a noise vector that had the same sequence length that was conditioned to a class label at each step of the sequence [4]. Given this input sequence, we varied the hidden dimension of the one RNN layer of the generator in order to experiment switching between Long-Short Term Memory layers (LSTM) and gated recurrent units (GRU). The output of this generator had be the same dimensions as the preprocessed data so that any discriminator used would be able to compare it to real data instances.

To take advantage of temporal relation on the discriminator side, we also experimented with an RNN style discriminator using LSTM layers. This discriminator, consisting of one LSTM layer, would take in a sequence of the EEG signal and output a sequence of labels determining if step is real or fake. This also follows a similar structure to the discriminators to those in Hyland et al., which were used to discriminate other types of generated sequential data [4].

Given their performance in image classification and their small amount of trainable parameters, CNNs are advantageous when used in GANs, especially if the problem entails some sort of relation to images. Thus, we based our CNN generators and discriminators off of deep convolutional GANs (DCGANs) [10]. Our generators consisted of fractionally-strided convolutional layers (also referred to as deconvolutional layers) followed by batch normalization and rectified linear units (ReLU) for activation. For the output layer, instead of using a ReLU activation, we used a Tanh activation, following the DCGAN framework. The discriminator was built in a similar fashion to normal convolutional classifiers but removed max pooling layers and replaced their activation functions with Leaky ReLU.

2.3. Evaluation

Our limited experience with EEG data meant that manual validation of the data would not be possible. Instead, we first trained several classifiers on the real dataset, and calculated a validation accuracy. We then tested the pre-trained classifiers on the generated dataset, using the obtained accuracy as a metric of the validity of the generated data, noting that a classifier taking random guesses would achieve an average accuracy of 25% since there are four classes. The results for each of the following classifiers are noted in Appendix A.

2.3.1 Fully-connected Net

The Fully Connected classifier used had 3 hidden layers with 1500, 1000, 500 activations respectively. This network also had batch normalization layers, and used ReLU as the activation function.

2.3.2 Shallow Convolutional Net

The next architecture we implemented was Convolutional Neural Networks (CNNs). As was explored by Schirrmiester et al. [12], CNNs can be used to capture spatial patterns and thus may be useful in the current application. The architecture involved 2 [CNN + ReLU + Maxpool] layers and 1 [CNN - ReLU] layer. The final layer was [Linear - Softmax]. In this architecture, the Maxpool layers play a vital role in downsampling the convolutional layers. This is helpful to get a larger receptive field and preventing a large number of parameters in the final linear layer.

2.3.3 Convolutional and Recurrent Hybrid Nets

In order to further improve classification accuracy, we used hybrid architectures. The motivation behind using a hybrid architecture was to fully capture temporal patterns in the EEG data (since it is a time-series) along with the spatial patterns which CNNs manage to capture. This intuition was backed by research from Lin et al. [8] This network had a [Conv - ReLU - MaxPool] layer, followed by a [Conv - ReLU] layer, an LSTM layer and then a linear layer.

3. Discussion

3.1. Results from Classifiers

As is noted in Appendix A, CNN-RNN hybrid architectures had the best classification accuracies. This result backs up our intuition that hybrid architectures would perform the best since they capture both the spatially-local as

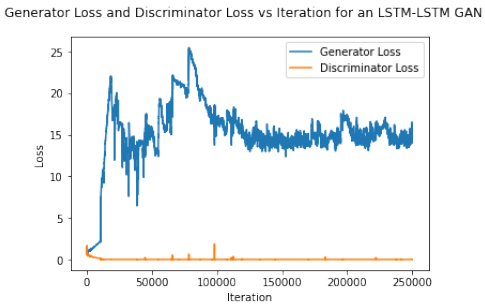


Figure 1: Convergence failure in an LSTM-LSTM GAN

well as global temporal patterns. Another interesting result is that dowsampling the data with CNNs before extracting temporal patterns with RNNs led to better classification accuracy than having RNNs before CNNs. This result is in line with those found by Zhu et al. [13], and could be due to the fact that spatial correspondences are more pronounced than temporal correspondences in the EEG dataset. It could also be due to the fact that downsampling before finding temporal correspondences means that global temporal patterns are found, as opposed to local temporal patterns.

3.2. Results from Generated Data

The best performing GAN achitecture was a GRU generator and an LSTM discriminator, which achieved an average of 29% accuracy, a 16% improvement over random chance. Overall, the classifiers achieved a much lower accuracy on the generated data than the real data (see Appendix A), and the accuracy achieved was only a slight improvement on random chance. Further, the amplitude range of the generated data is an order of magnitude smaller than that of the real data. Thus, it can be concluded that the GANs were not able to generate realistic data. Here are some reasons as to why our results were poor:

3.2.1 Convergence Failure

In order to converge, the generator and discriminator networks must reach an equilibrium, but if one network rapidly converges before the other, it will impair the learning of the other network. In our experiments, we noted that the discriminator loss rapidly converged to zero, while the generator loss remained non-zero and did not appreciably decrease. This means that the discriminator was able to easily classify every generated sample and hence the generator network could not learn. An example of this is visualized in Figure 1.

3.2.2 Difficulty in Tuning Parameters

As is noted by Salimans et al., [11] it is difficult to train GANs to converge since finding Nash equilibria is a hard problem, particularly when the cost functions being optimized are non-convex, as is the case in this application. Further, the space of parameters is a very high dimensional space, and thus settling on choices of parameters that lead to model convergence can be difficult.

3.2.3 Other Problems

Some specific difficulties that were reached in our case were the distribution of the data as well as difficulties with compute power. Due to our inexperience with EEG signals and the dataset, there were not a lot of features that we knew that we could take advantage of. Thus the data we generated might have been too difficult of a distribution for the generator to emulate. Lacking enough computing resources also proved to be a difficulty. Our main access to GPU compute time was through Google Colab which has time limits on user's access to GPUs. Thus we could not train our networks for extended periods of time.

3.3. Future work

While the results from our analysis were not too promising, the issue of GANs collapsing during training is not uncommon [7]. An improvement that can be made is by using WGANs as opposed to GANs. While the original GAN framework tries to minimize the Jensen-Shannon (JS) divergence between the real data distribution $\mathbb{P}_r$ and fake data distribution $\mathbb{P}_\theta$, the WGAN tries to minimize the Wasserstein distance between the two distributions $\mathbb{P}_\theta$ and $\mathbb{P}_r$[7] [6]. This leads to the discriminator, now called a critic, to maximize the value

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

while the generator maximizes the value $E_{x_f \sim \mathbb{P}_\theta}[D(x_f)]$. Although we did not implement any WGANs, Hartmann et al shows it is possible to create realistic data by modifying the training for the WGAN [7]. In addition to changing our model architecture, different metrics such as the Sliced Wasserstein Distance are shown to create the most realistic looking EEGs. However Hartmann et al recommends using several other metrics such as the Frechet Inception Distance and the Euclidean Distance in order to leverage the advantages and disadvantages of each of them.

4. Conclusion

With the amount of noise that is inherent in the data, replicating the EEG data using GANs did not appear to be quite effective. The best model we were able to create was a CNN+GRU architecture with an accuracy score of 67%. Although our generator was able to create data from noise, considering it could only be classified around 25% of the time, the model was ineffective at generating artificial data that resembled real EEG samples. Due to the constraints on our time and resources, we were unable to obtain useful results, however the knowledge and experienced gained will be useful in our future projects and endeavors.

References

[1] Alois Schlögl and Mel Slater and Gert Pfurtscheller. Presence research and EEG. http://www0.cs.ucl.ac.uk/research/equator/papers/Documents2002/Mel_presence_2002.pdf, 2002.

[2] C. Brunner, R. Leeb, G. R. Müller-Putz, A. Schlögl, and G.Pfurtscheller. BCI Competition 2008 – Graz Data Set A. http://www.bbci.de/competition/iv/desc_2a.pdf.

[3] C. Donahue, J. J. McAuley, and M. S. Puckette. Synthesizing audio with generative adversarial networks. *CoRR*, abs/1802.04208, 2018.

[4] C. Esteban, S. L. Hyland, and G. Rätsch. Real-valued (medical) time series generation with recurrent conditional gans, 2017.

[5] I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[7] K. G. Hartmann, R. T. Schirrmeister, and T. Ball. Eeg-gan: Generative adversarial networks for electroencephalograhic (eeg) brain signals, 2018.

[8] T. Lin, T. Guo, and K. Aberer. Hybrid neural networks for learning the trend in time series. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2273–2279, 2017.

[9] M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[10] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[11] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[12] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG. *CoRR*, abs/1703.05051, 2017.

[13] J. Zhu, H. Chen, and W. Ye. A hybrid cnn–lstm network for the classification of human activities based on micro-doppler radar. *IEEE Access*, 8:24713–24720, 2020.

Appendix

A. Performance

Please note our notation. Each framework has a generator and discriminator and denoted `G - D` where `G` is the generator architecture and `D` is the discriminator architecture. So for example `LSTM-CNN` has a generator that has a LSTM architecture and a discriminator with CNN architectures. Please note that the + represents that the network is hybrid of both architectures. The rows represent the model of GAN generating the data while the columns represent the classifier network.

These table report the accuracy of classifiers on data coming from different sources.

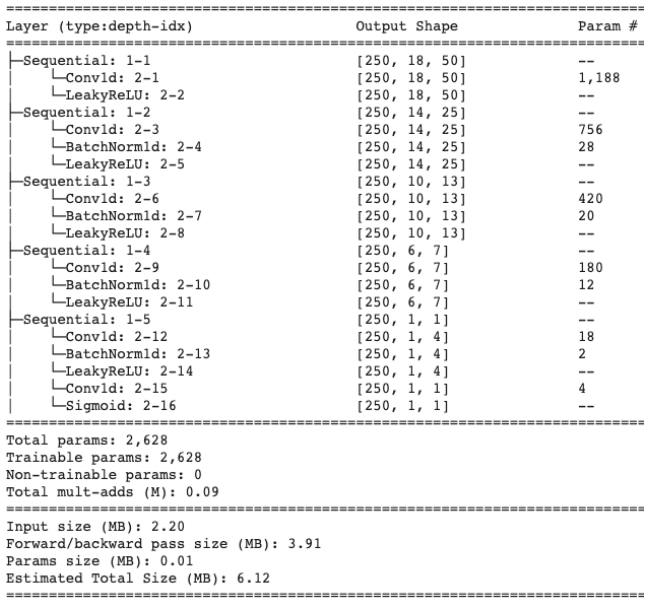
Data source	**Classifier**		
	FC Net	CNN	CNN + LSTM
Real Data	50.7%	60.8%	65.0%
GRU - LSTM	25.6%	36.2%	24.6%
GRU - CNN	23.3%	34.0%	21.8%
LSTM - CNN	26.3%	29.1%	22.1%
LSTM - LSTM	28.6%	22.2%	20.4%
CNN - CNN	22.9%	22.9%	22.9%
CNN + LSTM - CNN	24.7%	24.7%	23.6%

Data source	**Classifier**	
	CNN + GRU	LSTM + CNN
Real Data	67.3%	54.9%
GRU - LSTM	25.6%	33.5%
GRU - CNN	23.3%	36.7%
LSTM - CNN	26.3%	24.4%
LSTM - LSTM	23.1%	23.4%
CNN - CNN	22.9%	27.0%
CNN + LSTM - CNN	24.7%	26.8%

B. Architectures and Figures

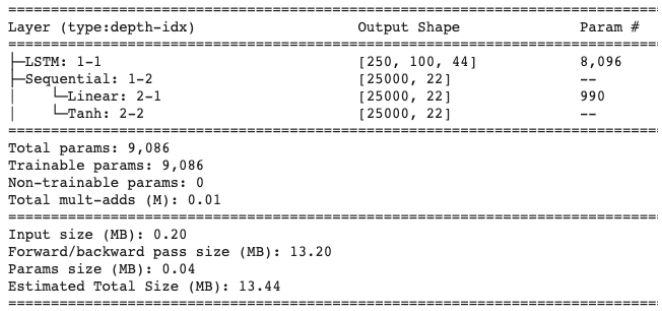
```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─LSTM: 1-1                              [250, 100, 44]            8,448
├─Sequential: 1-2                        [25000, 22]               --
│    └─Linear: 2-1                       [25000, 22]               990
│    └─Tanh: 2-2                         [25000, 22]               --
==========================================================================
Total params: 9,438
Trainable params: 9,438
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================
Input size (MB): 0.40
Forward/backward pass size (MB): 13.20
Params size (MB): 0.04
Estimated Total Size (MB): 13.64
==========================================================================
```

Figure 2: architecture of the generator for the LSTM-CNN GAN



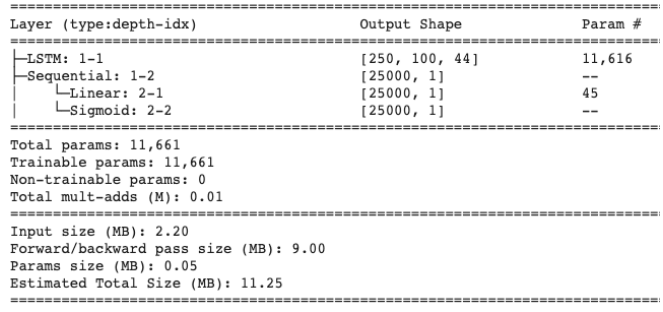
```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Sequential: 1-1                        [250, 18, 50]             --
│    └─Conv1d: 2-1                       [250, 18, 50]             1,188
│    └─LeakyReLU: 2-2                    [250, 18, 50]             --
├─Sequential: 1-2                        [250, 14, 25]             --
│    └─Conv1d: 2-3                       [250, 14, 25]             756
│    └─BatchNorm1d: 2-4                  [250, 14, 25]             28
│    └─LeakyReLU: 2-5                    [250, 14, 25]             --
├─Sequential: 1-3                        [250, 10, 13]             --
│    └─Conv1d: 2-6                       [250, 10, 13]             420
│    └─BatchNorm1d: 2-7                  [250, 10, 13]             20
│    └─LeakyReLU: 2-8                    [250, 10, 13]             --
├─Sequential: 1-4                        [250, 6, 7]               --
│    └─Conv1d: 2-9                       [250, 6, 7]               180
│    └─BatchNorm1d: 2-10                 [250, 6, 7]               12
│    └─LeakyReLU: 2-11                   [250, 6, 7]               --
├─Sequential: 1-5                        [250, 1, 1]               --
│    └─Conv1d: 2-12                      [250, 1, 4]               18
│    └─BatchNorm1d: 2-13                 [250, 1, 4]               2
│    └─LeakyReLU: 2-14                   [250, 1, 4]               --
│    └─Conv1d: 2-15                      [250, 1, 1]               4
│    └─Sigmoid: 2-16                     [250, 1, 1]               --
==========================================================================
Total params: 2,628
Trainable params: 2,628
Non-trainable params: 0
Total mult-adds (M): 0.09
==========================================================================
Input size (MB): 2.20
Forward/backward pass size (MB): 3.91
Params size (MB): 0.01
Estimated Total Size (MB): 6.12
==========================================================================
```

Figure 3: architecture of the discriminator for the LSTM-CNN, CNN-CNN, CNN+LSTM, and GRU-CNN GAN



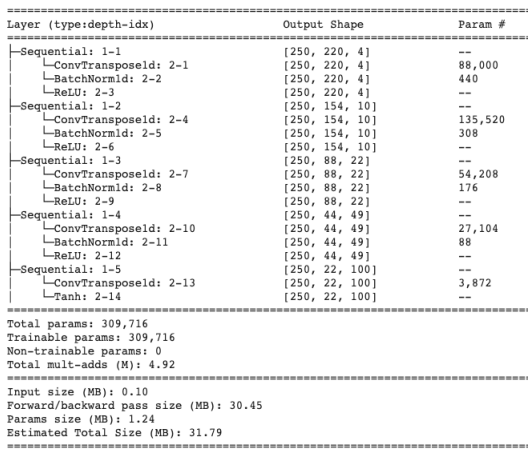
```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─LSTM: 1-1                              [250, 100, 44]            8,096
├─Sequential: 1-2                        [25000, 22]               --
│    └─Linear: 2-1                       [25000, 22]               990
│    └─Tanh: 2-2                         [25000, 22]               --
==========================================================================
Total params: 9,086
Trainable params: 9,086
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 13.20
Params size (MB): 0.04
Estimated Total Size (MB): 13.44
==========================================================================
```

Figure 4: architecture of the generator for the LSTM-LSTM GAN



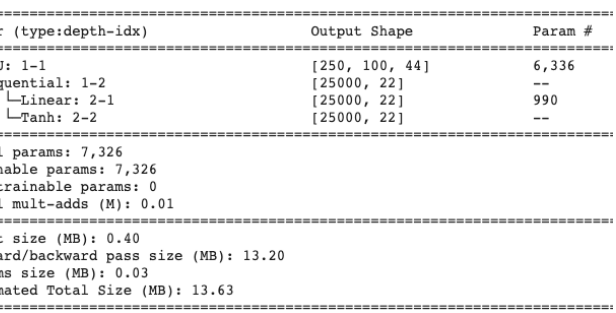
```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─LSTM: 1-1                              [250, 100, 44]            11,616
├─Sequential: 1-2                        [25000, 1]                --
│    └─Linear: 2-1                       [25000, 1]                45
│    └─Sigmoid: 2-2                      [25000, 1]                --
==========================================================================
Total params: 11,661
Trainable params: 11,661
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================
Input size (MB): 2.20
Forward/backward pass size (MB): 9.00
Params size (MB): 0.05
Estimated Total Size (MB): 11.25
==========================================================================
```

Figure 5: architecture of the discriminator for the LSTM-LSTM, GRU-LSTM GAN



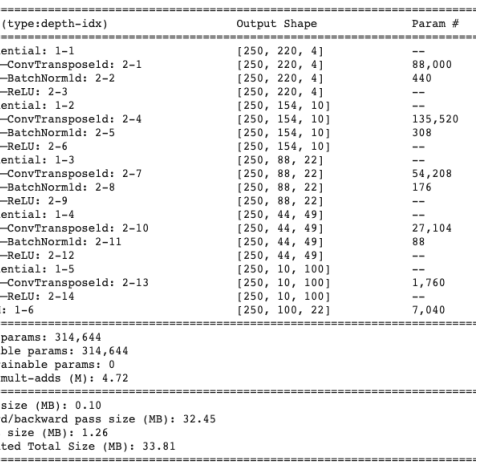
```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Sequential: 1-1                        [250, 220, 4]             --
│    └─ConvTranspose1d: 2-1              [250, 220, 4]             88,000
│    └─BatchNorm1d: 2-2                  [250, 220, 4]             440
│    └─ReLU: 2-3                         [250, 220, 4]             --
├─Sequential: 1-2                        [250, 154, 10]            --
│    └─ConvTranspose1d: 2-4              [250, 154, 10]            135,520
│    └─BatchNorm1d: 2-5                  [250, 154, 10]            308
│    └─ReLU: 2-6                         [250, 154, 10]            --
├─Sequential: 1-3                        [250, 88, 22]             --
│    └─ConvTranspose1d: 2-7              [250, 88, 22]             54,208
│    └─BatchNorm1d: 2-8                  [250, 88, 22]             176
│    └─ReLU: 2-9                         [250, 88, 22]             --
├─Sequential: 1-4                        [250, 44, 49]             --
│    └─ConvTranspose1d: 2-10             [250, 44, 49]             27,104
│    └─BatchNorm1d: 2-11                 [250, 44, 49]             88
│    └─ReLU: 2-12                        [250, 44, 49]             --
├─Sequential: 1-5                        [250, 22, 100]            --
│    └─ConvTranspose1d: 2-13             [250, 22, 100]            3,872
│    └─Tanh: 2-14                        [250, 22, 100]            --
==========================================================================
Total params: 309,716
Trainable params: 309,716
Non-trainable params: 0
Total mult-adds (M): 4.92
==========================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 30.45
Params size (MB): 1.24
Estimated Total Size (MB): 31.79
==========================================================================
```

Figure 6: architecture of the generator for the CNN-CNN GAN



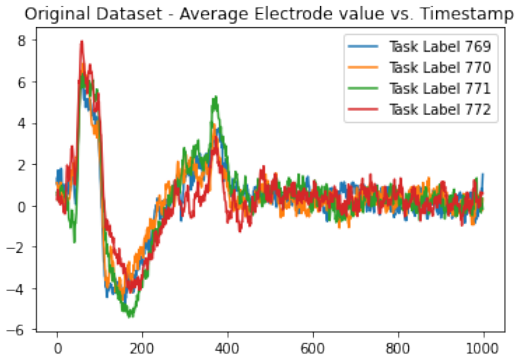
```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─GRU: 1-1                               [250, 100, 44]            6,336
├─Sequential: 1-2                        [25000, 22]               --
│    └─Linear: 2-1                       [25000, 22]               990
│    └─Tanh: 2-2                         [25000, 22]               --
==========================================================================
Total params: 7,326
Trainable params: 7,326
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================
Input size (MB): 0.40
Forward/backward pass size (MB): 13.20
Params size (MB): 0.03
Estimated Total Size (MB): 13.63
==========================================================================
```

Figure 7: architecture of the generator from the GRU-LSTM and GRU-CNN GAN

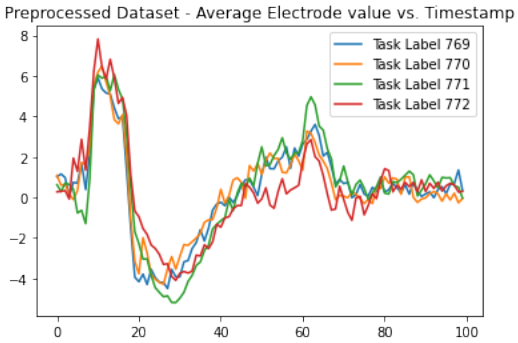


```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Sequential: 1-1                        [250, 220, 4]             --
│    └─ConvTranspose1d: 2-1              [250, 220, 4]             88,000
│    └─BatchNorm1d: 2-2                  [250, 220, 4]             440
│    └─ReLU: 2-3                         [250, 220, 4]             --
├─Sequential: 1-2                        [250, 154, 10]            --
│    └─ConvTranspose1d: 2-4              [250, 154, 10]            135,520
│    └─BatchNorm1d: 2-5                  [250, 154, 10]            308
│    └─ReLU: 2-6                         [250, 154, 10]            --
├─Sequential: 1-3                        [250, 88, 22]             --
│    └─ConvTranspose1d: 2-7              [250, 88, 22]             54,208
│    └─BatchNorm1d: 2-8                  [250, 88, 22]             176
│    └─ReLU: 2-9                         [250, 88, 22]             --
├─Sequential: 1-4                        [250, 44, 49]             --
│    └─ConvTranspose1d: 2-10             [250, 44, 49]             27,104
│    └─BatchNorm1d: 2-11                 [250, 44, 49]             88
│    └─ReLU: 2-12                        [250, 44, 49]             --
├─Sequential: 1-5                        [250, 10, 100]            --
│    └─ConvTranspose1d: 2-13             [250, 10, 100]            1,760
│    └─ReLU: 2-14                        [250, 10, 100]            --
├─LSTM: 1-6                              [250, 100, 22]            7,040
==========================================================================
Total params: 314,644
Trainable params: 314,644
Non-trainable params: 0
Total mult-adds (M): 4.72
==========================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 32.45
Params size (MB): 1.26
Estimated Total Size (MB): 33.81
==========================================================================
```

Figure 8: architecture of the generator from the CNN+LSTM-CNN GAN



(a) Original Dataset



(b) Preprocessed Dataset

Figure 9: Original and Preprocessed Datasets

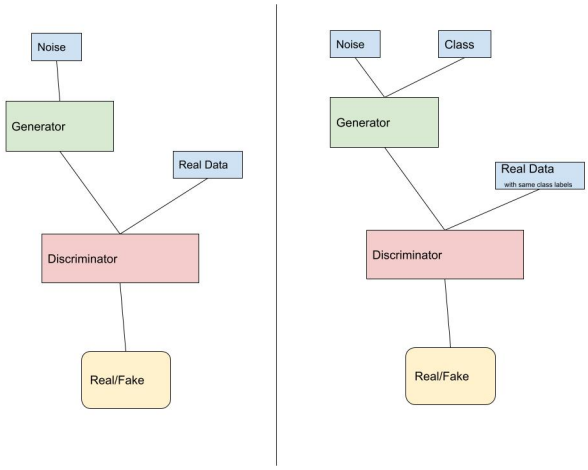


Figure 10: **Left**: the framework for vanilla GANs. **Right**: the framework for our conditional GANs