EEG Data Classification using Various Neural Networks

Amulya Shruthi
UID: 505626283
C247
ashruthi1797@g.ucla.edu

Ashmita Deb
UID: 105472409
C247
ashmitadeb@g.ucla.edu

Shweta Katti
UID: 505604846
C247
shwetakatti@g.ucla.edu

Yash Trivedi
UID: 905730032
C247
yashtrvd@g.ucla.edu

Abstract

The aim of this project is to classify the electroencephalography (EEG) data collected from Brain Computer Interaction (BCI) Competition IV [2] using neural networks such as Recurrent Neural Network (RNN), Convolutional Recurrent Neural Network (CRNN), Generative Adversarial Network (GAN) and Variational Autoencoder (VAE). We first find a base model that performs best on the data, and then augment it with artificial GAN generated images to improve accuracy.

1. Introduction

The EEG dataset has 9 subjects which are recorded with the help of 22 electrodes and consists of four different tasks that need to be classified. The BCI dataset contains a total of 2115 trials collected over 1000 time stamps recorded periodically. Since these EEG signals are collected using a non-invasive scalp electrode, they potentially have relatively poor spatiotemporal resolution compared to other neural recording techniques and thus contain a lot of noise. For the project, we apply several deep learning models such as CNN, RNN, CRNN and also generate new artificial data using GANs and use it on the model that gives us best performance in order to improve the classification accuracy. The motivation behind each model's choice is explained in the following subsections.

1.1. Convolutional Neural Network (CNN)

CNNs are the most dominant form of deep learning model used for classification. We can build complex models using just CNNs by training large datasets.[6] It consists of multiple layers such as convolution, pooling and fully connected layers. In this architecture, we use two convolutional layers, which are used for feature extraction. We use max pooling operations to downsample the data. To avoid vanishing gradients, we used squared ReLU and softmax activation functions. We also used Batch Normalization and dropouts to avoid any risk of overfitting.

1.2. Recurrent Neural Network (RNN)

A Recurrent Neural Network recognizes sequential data and patterns, hence predicting the most likely scenario next. It consists of a looping mechanism that allows information to re enter the hidden layer. However, due to the vanishing gradient issue, these suffer from short term memory. In order address the issue, we use a specialized RNN i.e. Long Short Term Memory (LSTM). [9]

The RNN implemented has two LSTM layers in our case. By adding dropout, we reduced the model's sensitivity to noise. By adding high dropout to both layers however, we observed that performance dropped yet again. We had to add a large dropout only in the second layer to maximize performance. The model in total consisted of two Batch normalization, Dropout and two LSTM layers.

1.3. Convolutional Recurrent Neural Network (CRNN)

We can note that the BCI dataset consists of temporal information. Hence, it is intuitive to use a model that uses both feed-forward and feedback connections to include information from a given time period by considering present and past data. It is also very common to build a hybrid CNN model which consists of RNN layers for improved accuracy. Hence, we test CRNN on the training dataset for improved classification results. The Convolutional Recurrent Neural Network is a combination of both Convolutional Neural Network (CNN) and Recurrent

Neural Nework (RNN).

The network is built by first using four 2-dimensional Convolution layers and batch normalization. Exponential Linear Unit (ELU) is used as the activation function as it tends to converge cost to zero faster and produce more accurate results. These layers are then followed by two Bidirectional Gated Recurrent Units (GRU). We use a bidirectional RNN to enable past and future input traversals. This connects two hidden layers of opposite directions to the same output. We use GRUs to avoid the problem of vanishing gradients which typically occurs while using a Simple RNN. Throughout this whole architecture, we use dropouts of 0.3 which is a method used for regularization that helps to avoid overfitting.

1.4. Generative Adversarial Network (GAN)

Generative Adversial Networks aim to learn the data distribution via an adversarial process, which involve a minimax game between two networks - the Generator G and the Discriminator D. The objective for the Generator is to come up with extremely believable "fake" examples of data that can fool the Discriminator into classifying it as coming from the real data distribution. In other words, the training process for G is to maximize the probability of D making a mistake. On the other hand, D needs to maximize the probability of assigning the correct label to both - training examples and artificial samples generated by G.

Since the time GANs were first introduced in [3], we have seen a lot of variations - mainly with respect to the loss functions. One such variation is the Wasserstein GAN (WGAN) of [1] that we implement in this research. WGANs provide better stability and get rid of the vanishing gradients problem with the traditional minimax GAN. Further more, adding a *gradient penalty (GP)* for the discriminator, as proposed in [4], performs better than the regular WGANs.

In this project, we use WGAN with GP to generate artificial data to use with the best performing base architecture (CNN / RNN / CRNN) and expect the test accuracy to increase over what is achieved with just the base architecture. The individual networks of our WGAN consist of convolutional layers along with Batchnorm regularization. We use a ReLU activation for the generator, and LeakyReLU for the discriminator [8].

2. Results

The Convolutional Neural Network gave us a test accuracy of 65%. The LSTM Recurrent Neural Network gave us a test accuracy of 55%. The combination of CNN and RNN with GRU gave us a test accuracy of 71%. Thus, out of the three architectures, CRNN performed the best and hence we used this model and appended the artificially generated data points from our GAN for subject 0 and evaluated the test accuracy for subject 0. CRNN with original dataset resulted in a test accuracy of 26% for subject 0 and CRNN with GAN generated images appended into the training data gave us a test accuracy of 35%.

3. Discussions

The three architectures (CNN, RNN and CRNN) were tested for the completed dataset and the test accuracies were compared to help select the best performing model. But we compared the CRNN with and without GAN generated images for only one particular subject due to time and computational limits. But since we see an increase in the test accuracy for subject 0 when trained on CRNN with GAN generated data, it is safe to assume that the model accuracy on the entire dataset would also increase.

We also ran the RNN architecture with different preprocessing techniques to improve accuracy. We initially used the up-sampling, re-sampling and concatenation technique and this gave a validation accuracy of 55%. Eventually on using max-pooling and averaging (to remove noise) before the RNN architecture, we could see an improvement in the validation accuracy. This jumped up to 83%.

CNNs are slightly advantaged and have performed better than RNNs with a better test accuracy on the EEG dataset. RNNs cannot look for high level patterns, but CNNs can because of the convolutions in their architecture.

Out of three architectures tried on the entire EEG dataset, CRNN performs the best. This is expected as RNNs work particularly well with time series data. We then used this architecture to prove the efficacy of GANs - i.e., train one model using some basic data augmentation techniques (described in the Appendix) and the other with an additional 500 samples generated by the GAN. We see an increase in test accuracy from 26% to 35%. This is also expected, since if the GAN is trained well, this is essentially another data augmentation technique and thus, increasing the size of the training set leads to an increase in performance.

We also tried to build a generative model using Variational Autoencoder (VAE) as it is supposed to suppress the noise present and therefore improve the performance of the CRNN model. But the VAE model constructed collapsed and could not generate the artificial images for the particular subject.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. 2017.

[2] C. Brunner, R. Leeb, G. Müller-Putz, A. Schlögl, and G. Pfurtscheller. Bci competition 2008 – graz data set a. 2008.

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

[4] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. 2017.

[5] X. Lai, X. Bai, and Y. Hao. Unsupervised generative adversarial networks with cross-model weight transfer mechanism for image-to-image translation. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 1814–1822, 2021.

[6] X. Lun, Z. Yu, T. Chen, F. Wang, and Y. Hou. A simplified cnn classification method for mi-eeg via the electrode pairs signals. *Frontiers in Human Neuroscience*, 14, 2020.

[7] J. L. Mahendra Kumar, M. Rashid, R. M. Musa, M. A. Mohd Razman, N. Sulaiman, R. Jailani, and A. P.P. Abdul Majeed. The classification of eeg-based wink signals: A cwt-transfer learning pipeline. *ICT Express*, 7(4):421–425, 2021.

[8] A. Radford, S. Chintala, and L. Metz. Unsupervised representation learning with deep convolutional generative adversial networks. 2016.

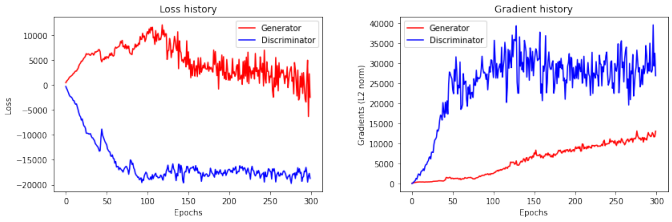
[9] M. A. Zain. Predicting emotions using eeg data with recurrent neural networks. 2021.

Appendix

As highlighted in the project report, we test three base architectures on the entire dataset, namely - CNN, RNN and CRNN, and find that CRNN performs the best. We then augment our dataset with GAN generated images, and train it using the CRNN architecture and evaluate its performance on subject 0. The results are presented in the table below.

Model	**Accuracy**
CNN	65%
LSTM RNN	55%
CRNN + GRU	**71%**
CRNN (subject 0)	26%
CRNN + GAN (subject 0)	**35%**

Fig. 1 shows the Accuracy and Loss trajectories for the generator and discriminator during training.



(a) Loss v/s epochs (b) Grad v/s epochs

Figure 1: GAN training

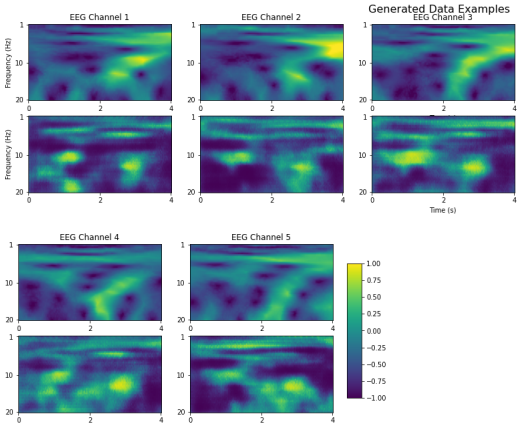
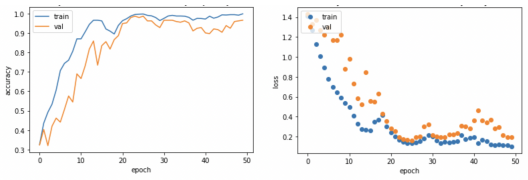


Figure 2: Output of GAN (Row 2 and Row 4) v/s actual data (Row 1 and Row 3)

Fig. 3 shows the Accuracy and Loss trajectories for the CRNN model with appended GANs generated images.

We now explain the various architectures used, and the data augmentation techniques.



(a) Accuracy Trajectory (b) Loss Trajectory

Figure 3: CRNN with GANs model

Data Augmentation

For every architecture tested, we employ a set of basic pre-processing techniques to augment the data. First, we trim the data to include only the first 500 timesteps, as the EEG data just had random noise for the last 500. Next, we use two pooling operations - max pooling and averaging, to double the dataset. We also subsample the training data, and concatenate it with the pooling output to obtain a dataset which is four times the original size.

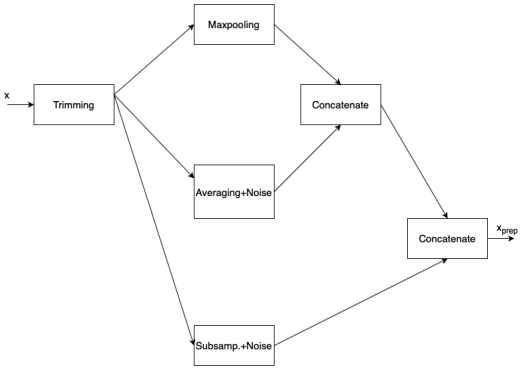


Figure 4: Data Augmentation

For training GANs, we specifically use a morlet wavelet transform (MWT) [7] as brain signals are known to operate in frequency bands. Also, GANs are notorious networks to train and we expected MWT will help learn the features of the dataset better compared to conventional data processing techniques. [5] Then, while combining CRNN with the GAN generated data, we perform the same wavelet transform on the augmented data (Fig. 4) to match its shape with the GAN generated data and pass them together into the CRNN model.

CNN Architecture

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 250, 1, 25)        5525

 max_pooling2d_4 (MaxPooling  (None, 84, 1, 25)        0
 2D)

 batch_normalization_8 (Batc  (None, 84, 1, 25)        100
 hNormalization)

 elu_6 (ELU)                 (None, 84, 1, 25)         0

 dropout_6 (Dropout)         (None, 84, 1, 25)         0

 conv2d_9 (Conv2D)           (None, 84, 1, 50)         12550

 batch_normalization_9 (Batc  (None, 84, 1, 50)        200
 hNormalization)

 elu_7 (ELU)                 (None, 84, 1, 50)         0

 dropout_7 (Dropout)         (None, 84, 1, 50)         0

 conv2d_10 (Conv2D)          (None, 84, 1, 100)        50100

 batch_normalization_10 (Bat  (None, 84, 1, 100)       400
 chNormalization)

 elu_8 (ELU)                 (None, 84, 1, 100)        0

 dropout_8 (Dropout)         (None, 84, 1, 100)        0

 flatten_2 (Flatten)         (None, 8400)              0

 dense_4 (Dense)             (None, 100)               840100

 dense_5 (Dense)             (None, 4)                 404

=================================================================
Total params: 909,379
Trainable params: 909,029
Non-trainable params: 350
_________________________________________________________________
```

Figure 5: Model Architecture for CNN training

RNN Architecture

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 250, 30)           6360

 batch_normalization_6 (Batc  (None, 250, 30)          120
 hNormalization)

 lstm_7 (LSTM)               (None, 16)                3008

 batch_normalization_7 (Batc  (None, 16)               64
 hNormalization)

 dropout_3 (Dropout)         (None, 16)                0

 dense_3 (Dense)             (None, 4)                 68

=================================================================
Total params: 9,620
Trainable params: 9,528
Non-trainable params: 92
_________________________________________________________________
```

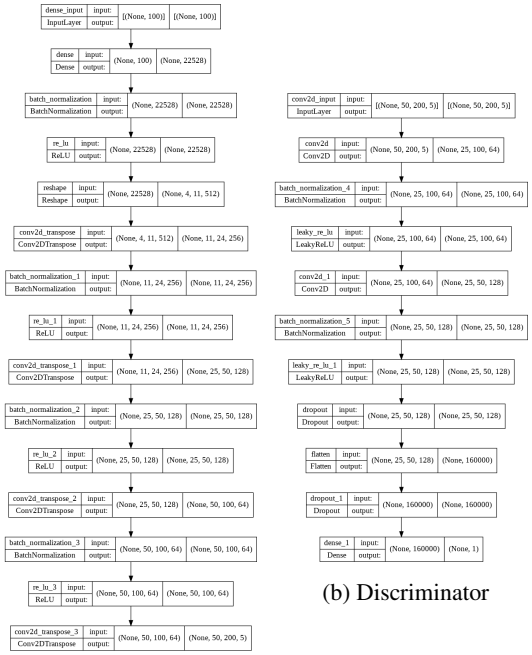
Figure 6: Model Summary for RNN training

CRNN Architecture

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 50, 200, 25)       1275

max_pooling2d_1 (MaxPooling  (None, 17, 200, 25)       0
2D)

batch_normalization_3 (Batc  (None, 17, 200, 25)       100
hNormalization)

elu_3 (ELU)                  (None, 17, 200, 25)       0

dropout_5 (Dropout)          (None, 17, 200, 25)       0

conv2d_4 (Conv2D)            (None, 17, 200, 50)       12550

batch_normalization_4 (Batc  (None, 17, 200, 50)       200
hNormalization)

elu_4 (ELU)                  (None, 17, 200, 50)       0

dropout_6 (Dropout)          (None, 17, 200, 50)       0

conv2d_5 (Conv2D)            (None, 17, 200, 100)      50100

batch_normalization_5 (Batc  (None, 17, 200, 100)      400
hNormalization)

elu_5 (ELU)                  (None, 17, 200, 100)      0

dropout_7 (Dropout)          (None, 17, 200, 100)      0

flatten_2 (Flatten)          (None, 340000)            0

dense_3 (Dense)              (None, 100)               34000100

reshape_2 (Reshape)          (None, 100, 1)            0

bidirectional_2 (Bidirectio  (None, 200)               61800
nal)

dropout_8 (Dropout)          (None, 200)               0

flatten_3 (Flatten)          (None, 200)               0

dense_4 (Dense)              (None, 100)               20100

reshape_3 (Reshape)          (None, 100, 1)            0

bidirectional_3 (Bidirectio  (None, 20)                780
nal)

dropout_9 (Dropout)          (None, 20)                0

dense_5 (Dense)              (None, 4)                 84

=================================================================
Total params: 34,147,489
Trainable params: 34,147,139
Non-trainable params: 350
_________________________________________________________________
```

Figure 7: Model Architecture for CRNN training

GAN Architecture



(a) Generator

(b) Discriminator

Figure 8: Model Architectures for GAN training