Classifying Movement-Related EEG Data using Neural Networks

Chiao Lu*
Computer Science, UCLA
josephlu85@engineering.ucla.edu

Alex Wang*
Computer Science, UCLA
alexhw@g.ucla.edu

Josh Vendrow*
Computer Science, UCLA
jvendrow@ucla.edu

Abstract

We aim to classify 2,115 training/validation samples and 443 test samples of 1,000-timestep Electroencephalography (EEG) signals gathered from 22 electrodes using convolutional and recurrent neural network architectures. We explored many deep learning architectures along with many data preprocessing techniques, such as trimming, subsampling, and averaging to augment the dataset for better training. With our data preprocessing methods, our ConvMixGRU is able to achieve an accuracy of 70.60%.

1. Introduction

EEG analysis has been used in a variety of fields including biology, electrical engineering, and neural science. Because EEG measurements are noninvasive, the data gathered contains significant noise. Being able to differentiate important signal information encoded in EEG signals from background noise is crucial in identifying the characteristics of brain activities. Deep learning has recently become a popular approach to handling this kind of data.

Moreover, given the temporal properties of EEG data, it is natural to associate this classification task with Recurrent Neural Networks (RNN), which have been widely employed in the Natural Language Processing (NLP) domain. RNNs are very powerful in encoding temporal dependencies and properties in time series data. We look to incorporate RNNs in the classification of EEG data.

Another popular architecture that is studied extensively in EEG classification is Convolutional Neural Networks (CNN) [4, 6, 8, 11]. In CNNs, the EEG data can be regarded as a 2D pattern of channels signals over rows and time over columns. As a result, we are motivated to perform convolution both temporally and spatially in this task. By doing so, the bivariate features of EEG data can be effectively utilized.

In this report, we classify EEG data from the BCI Competition 2008 - Graz dataset A [2], which consists of EEG data from 9 subjects performing four different motor imagery tasks (left hand, right hand, both feet, and tongue). We investigate and analyze the aforementioned deep neural networks and the combinations of them, along with different data preprocessing techniques.

* These author contributed equally.

2. Methodologies

2.1. Data Preprocessing

2.1.1 Trimming

We first cluster samples into their corresponding labels and examine their EEG signals by plotting the averaged signals in their respective channel. As shown in Figure 1, the EEG signals reveal distinguishable patterns roughly in the first 500 time steps. The signals in the later 500 time steps are noisier and do not exhibit clear patterns. Therefore, we trim away the later 500 time steps and perform further preprocessing methods on the first 500 time steps.

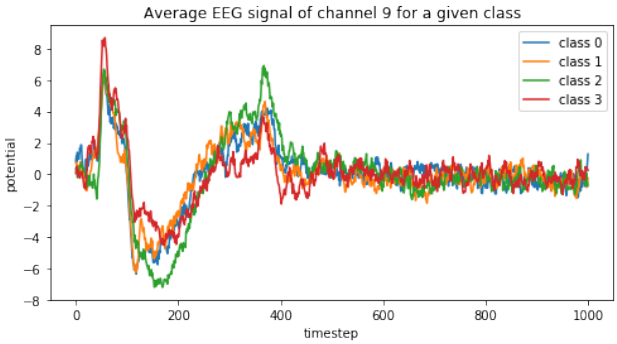

Figure 1. Selected 1 channel out of 22 for demonstration. All other channels show similar trends

2.1.2 Augmentation

Prior to data augmentation, the model overfits the training data, with training accuracies attaining over 80% but low validation accuracies around 50%. To address this, we augment our data using subsampling, window averaging, max-pooling, and Gaussian noise addition. For subsampling, we

subsample at intervals of 5 timesteps starting at timestep 0 to 4 individually on the trimmed data samples to make five 100-timestep data samples. We then add Gaussian noise $n \sim \mathcal{N}(\mu = 0, \sigma = 0.5)$ to these 100-timestep data samples. We also average and maxpool the signals at intervals of 5 timesteps. This gives us a total of 2,115 $\times$ 7 training and 443 $\times$ 7 testing samples. We determined the specifics of strategy experimentally.

We perform the same augmenting procedures on test data to get 7 samples from each datapoint and implement a majority voting system over these samples for classification.

3. Models

As defaults, we use rectified linear units (RELU, $f(x) = x$ if $x > 0$ and $f(x) = 0$ if $x \leq 0$) as activation functions and we use adam optimizer for all models. Following the results in [8], which looked at the application of CNNs to EEG Data, we use exponential linear units (ELU, $f(x) = x$ if $x > 0$ and $f(x) = e^x - 1$ if $x \leq 0$) for convolution layers. For regularization, we use Dropout layers and L2 regularization.

In our experiment, we tested a variety of architectures. We list all the models we implemented in Appendix B.

3.1. AvgPoolCNN

We base one of our models on the shallow convolution architecture presented in [8]. This model has a convolution across time, followed by a convolution across all electrodes, and then an average pool over time, directly to a dense layer with 4 outputs to a softmax loss. We use ELUs as activation functions for the convolution layers. Our input size is smaller than the input in this paper following the trimming and subsampling process, so we decrease convolution sizes accordingly and choose the specific parameters using hyperparameter selection.

3.2. ConvMixGRU and SimpleConvGRU

In a recent survey paper, Craik et el. [3] discussed many practical models applied to EEG classification tasks. Specifically, many effective models leveraged a combination of CNNs and RNNs [7]. Motivated by this survey paper, we created two models that consist of a convolutional layer, GRU layer, and a Fully Connected layer with different orders of convolution and GRU layer. We have also tested with batchnormalization and dropout layers between them. The details can be found in Appendix B.

3.3. SimpleGrid and Fourier model Model

3.3.1 SimpleGrid

Thus far, our models have ignored the positional configuration of probes on the scalp. In this model, we transform the original 22 channels into a 6 $\times$ 7 grid. This gives us data samples of 6 $\times$ 7 $\times$ 100, where 6 $\times$ 7 refers to the 2D positional configuration of the 22 probes on the scalp. We pass the input to a convolutional layer followed by a GRU layer and then a FCN to perform the final classification.

3.3.2 SingleFourier

One popular method for EEG classification is the Fourier method proposed by Bashivan [1], which takes a multichannel EEG time series and performs a Fourier transformation. Each probe, after a FT (over whole timesteps), gives a frequency domain signal. Now, with domain knowledge of the brain waves [5], we know that there are three (frequency) regions of interest - theta range (4-8 Hz), alpha range (8-12 Hz), and beta range (12 - 35 Hz). The frequency domain signal is divided into these three ranges, which can be regarded as RGB values. With the spatial location of probes in 2D and RGB values obtained from each probe, we can obtain an image from one sample. The structure of the model in illustrated in Figure 2.

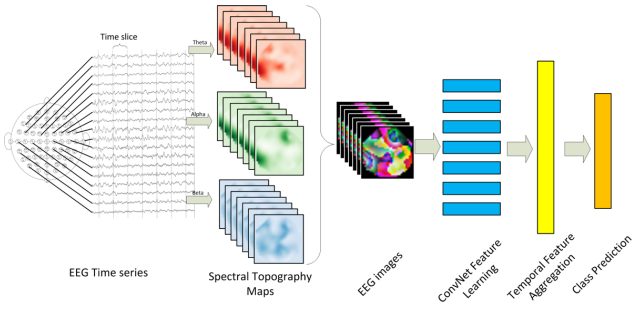

Figure 2. Fourier network architecture

The image is then fed to a CNN (and optionally an RNN) and then a softmax layer is used to perform classification. A variant of the model transforms a multichannel EEG time series into "movies" instead of images by chopping the time series into several chunks (e.g. by splitting a 10 second series to 10 one-second series) and performs the same transformation steps on each chunk.

4. Results and Discussion

4.1. Overall Results

The best testing accuracy is achieved by the ConvMixGRU model with 68.76% val. accuracy and **70.6%** test accuracy. The best result of a CNN only architecture on the preprocessed EEG data is 59.35% val. accuracy and 62.76% test accuracy. The SimpleGRUConv achieved a val. accuracy of 60.37% and test accuracy of 62.50%. Note that the test accuracies are higher than validation accuracies because we use a majority classifier on the test data. On the contrary, the SimpleGrid and SingleFourier models performed much worse, both with val. and test accuracies of around 40%.

The possible reasons for bad performance using the popular model will be discussed in the Discussion section.

4.2. Ablation Study

We also conduct an ablation study of different processing techniques on our ConvMixGRU. Without any preprocessing, ConvMixGRU only achieves 43.14% test accuracy. Trimming improves ConvMixGRU's accuracy to 52.78%. Subsampling and noise provides the most improvement on test performance, attaining to 64.76% and 61.54% when applied individually, an increase of 21.62% and 18.40% respectively. Combining three methods renders the most improvement of 27.46% on test accuracy. Using averaging and maxpooling only increases marginally around 2% compared to no preprocessing. The details of the ablation results are in Appendix A.

5. Discussion

5.1. ConvMixGRU Architecture

For convolution layers, we looked at models with a variety of filter sizes based on the success of networks such as VGGNet [10] and ResNet [10] that take advantage of large stacks of small (3×3) filters. Our results for these small convolutions were fairly poor, which is likely because electrodes that are numbered closer may not actually have a more important relationship. For this reason, we generally replaced most convolutions across electrodes with a single 22×1 convolution, which avoids such an assumption.

For the RNN layer, we tried GRUs and LSTMs, and in general GRUs performed slightly better than LSTMs in classification. We tried out different sizes in multiples of 22, and 44 seems to performed the best.

The results listed in Appendix A show the capabilities of both CNN and RNN architecture with both the AvgPoolCNN and SimpleGRU accomplishing above 60+% test accuracy. However, integrating both architectures boosted the final performance a bit more. This emphasizes the benefits of combining CNN and RNN in EEG data classification. We also tried training our model with a single subject. Although the validation accuracy on that subject was comparable, the test accuracy across all subjects was much lower than val accuracy. This indicates that the network overfits from an individual and cannot generalize to other subjects.

5.2. Why Do SimpleGrid and Fourier Method Fail?

In general, the Fourier method [1] has had success in practice, however it performs badly on our EEG dataset. There are potentially three major reasons: not enough sampling frequency, bad spatial resolution, and a coarse Fourier transform.

5.2.1 Not Enough Frequency/Spatial Resolution

Recall that the beta range of a human brain wave is around 12 to 35 Hz. By Nyquist Theorem [9], we would need at least 70 Hz sampling frequency. However we realized that our sampling strategy sampled the data at 50 Hz. Therefore, we lose information in the beta range. In addition, the Fourier method in [1] consumes $32 \times 32 \times 3$ images for CNNs. Our inputs, on the other hand, only have $6 \times 7 \times 3$ dimension. After a couple convolutional layers, the inputs become too shallow. This leads to bad performance because there is not sufficient data to learn from this smaller representation of the data.

5.2.2 Coarse Fourier

Our Fourier strategy takes the entire time series and applies a Fourier transform to get the frequency domain representation. Bashivan proposed another approach which is a more refined method of applying the FT. As mentioned earlier, they chop the time series into several one-second frames and apply a FT onto each of the one-second frames. So, for example, a 60-second time series would be chopped into 60 1-second time series, and we get 60 Fourier maps. Since these chopped frames are related in time, we can additionally use RNNs to leverage the temporal relation.

6. Conclusion

Our results suggest that the optimal architecture is one that leverages the benefits from both CNN and RNN. The intrinsic EEG data property of signals over different channels forms a spatial pattern where CNNs can be applied. On the other hand, the fact that the data was collected as time series prompts the use of an RNN to capture the temporal features. Combined with data preprocessing, our ConvMixGRU is able to achieve a test accuracy of 70.60%. This demonstrates that a deep learning approach requires a large dataset to train to its full potential.

Due to the time constraint and the scope of this project, we only focus on the cores of optimizing the model performance. For possible future work, we would like to try ensembling different models, as well as more in-depth analysis of thr Fourier method. There are also other preprocessing methods, in which we seek to automatically generate data using Generative Adversarial Networks (GANs) and Variational Autoencoder (VAR).

References

[1] P. Bashivan, I. Rish, M. Yeasin, and N. Codella. Learning representations from eeg with deep recurrent-convolutional neural networks. *arXiv preprint arXiv:1511.06448*, 2015.

[2] C. Brunner, R. Leeb, G. Muller-Putz, A. Schlogl, and B. Competition. Graz data set a. *Institute for Knowledge Discovery, and Institute for HumanComputer Interfaces Graz University of Technology, Austria*, 2008.

[3] A. Craik, Y. He, and J. L. Contreras-Vidal. Deep learning for electroencephalogram (eeg) classification tasks: a review. *Journal of neural engineering*, 16(3):031001, 2019.

[4] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of neural engineering*, 15(5):056013, 2018.

[5] E. K. S. Louis, L. C. Frey, J. W. Britton, J. L. Hopp, P. Korb, M. Z. Koubeissi, W. E. Lievens, and E. M. Pestana-Knight. The normal eeg. In *Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants [Internet]*. American Epilepsy Society, 2016.

[6] P. Mirowski, D. Madhavan, Y. LeCun, and R. Kuzniecky. Classification of patterns of eeg synchronization for seizure prediction. *Clinical neurophysiology*, 120(11):1927–1940, 2009.

[7] P. Nagabushanam, S. T. George, and S. Radha. Eeg signal classification using lstm and improved neural network algorithms. *Soft Computing*, pages 1–23, 2019.

[8] 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. arxiv, 2017. *arXiv preprint arXiv:1703.05051*.

[9] C. E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[11] Y. R. Tabar and U. Halici. A novel deep learning approach for classification of eeg motor imagery signals. *Journal of neural engineering*, 14(1):016003, 2016.

Appendix

A. Experiment Results

Model Performance (all preprocessing applied)

Model	Val Acc	Test Acc
`ConvMixGRU`	68.76%	70.60%
`ConvMixGRU` (1 Sub.)	61.57%	52.29%
`SimpleGRUConv`	60.37%	62.50%
`AvgPoolCNN`	59.35%	62.76%
`VanillaGRU`	62.20%	64.79%
`VanillaLSTM`	59.82%	61.17%
`SimpleGrid`	41.28%	40.20%
`SingleFourier`	36.73%	42.89%
`FCN`	34.92%	35.21%

Ablation Study on the best model

ConvMixGRU's Performance on preprocessing methods

Model	Val Acc	Test Acc
All methods	68.76%	70.60%
Subsampling	63.52%	64.76%
Gaussian Noise	60.33%	61.54%
Trimming	56.34%	52.78%
Window Avg.	50.43%	45.49%
MaxPool (5)	50.14%	45.27%
No preprocess	42.52%	43.14%

B. Architectures

Summaries of models tested in this report:

ConvMixGRU

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 91, 22)            4862
_________________________________________________________________
batch_normalization_1 (Batch (None, 91, 22)            364
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 45, 22)            0
_________________________________________________________________
gru_1 (GRU)                  (None, 45, 44)            8844
_________________________________________________________________
dropout_1 (Dropout)          (None, 45, 44)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1980)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                126784
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 260
=================================================================
Total params: 141,114
Trainable params: 140,932
Non-trainable params: 182
```

SimpleGRUConv

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 100, 22)	2970
conv1d_1 (Conv1D)	(None, 91, 22)	4862
flatten_1 (Flatten)	(None, 2002)	0
dropout_1 (Dropout)	(None, 2002)	0
dense_1 (Dense)	(None, 4)	8012

Total params: 15,844
Trainable params: 15,844
Non-trainable params: 0

AvgPoolCNN

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 22, 100)	0
reshape_1 (Reshape)	(None, 22, 100, 1)	0
conv2d_1 (Conv2D)	(None, 22, 91, 48)	528
batch_normalization_1 (Batch	(None, 22, 91, 48)	192
dropout_1 (Dropout)	(None, 22, 91, 48)	0
conv2d_2 (Conv2D)	(None, 1, 91, 40)	42280
batch_normalization_2 (Batch	(None, 1, 91, 40)	160
average_pooling2d_1 (Average	(None, 1, 17, 40)	0
flatten_1 (Flatten)	(None, 680)	0
dense_1 (Dense)	(None, 4)	2724
activation_1 (Activation)	(None, 4)	0

Total params: 45,884
Trainable params: 45,708
Non-trainable params: 176

SimpleGRU

Layer (type)	Output Shape	Param #
gru_1 (GRU)	(None, 100, 22)	2970
flatten_1 (Flatten)	(None, 2200)	0
dropout_1 (Dropout)	(None, 2200)	0
dense_1 (Dense)	(None, 4)	8804

Total params: 11,774
Trainable params: 11,774
Non-trainable params: 0

SimpleLSTM

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 100, 22)	3960
flatten_1 (Flatten)	(None, 2200)	0
dropout_1 (Dropout)	(None, 2200)	0
dense_1 (Dense)	(None, 4)	8804

Total params: 12,764
Trainable params: 12,764
Non-trainable params: 0

SimpleGrid

Layer (type)	Output Shape	Param #
conv3d_1 (Conv3D)	(None, 4, 6, 49, 32)	800
batch_normalization_1 (Batch	(None, 4, 6, 49, 32)	24
max_pooling3d_1 (MaxPooling3	(None, 2, 3, 49, 32)	0
conv3d_2 (Conv3D)	(None, 2, 3, 49, 1)	33
reshape_1 (Reshape)	(None, 6, 49)	0
gru_1 (GRU)	(None, 20)	4200
dense_1 (Dense)	(None, 32)	672
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 4)	132

Total params: 5,861
Trainable params: 5,849
Non-trainable params: 12

SingleFourier

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 6, 6, 128)	1664
reshape_1 (Reshape)	(None, 36, 128)	0
batch_normalization_1 (Batch	(None, 36, 128)	512
reshape_2 (Reshape)	(None, 6, 6, 128)	0
conv2d_2 (Conv2D)	(None, 5, 5, 64)	32832
max_pooling2d_1 (MaxPooling2	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 32)	8224
dropout_2 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 4)	132

Total params: 43,364
Trainable params: 43,108
Non-trainable params: 256

FCN

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 2200)	0
dense_1 (Dense)	(None, 100)	220100
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 100)	10100
dropout_2 (Dropout)	(None, 100)	0
dense_3 (Dense)	(None, 4)	404

Total params: 230,604
Trainable params: 230,604
Non-trainable params: 0