Details in 1D model in 4th place solution (bilzard's part)

1. Outline

Initially, my solution was an ensemble of 1D and 2D models. However, after merging teams and recognizing that my teammates had already developed strong 2D models, I shifted my focus primarily towards 1D models (i.e. the models process raw EEG signals directly).

2. Basic Concept

The approach to our 1D modeling is twofold:

1. **L/R Symmetric Modeling**: We aimed to maintain symmetry in the model to facilitate the detection of Laterality.
2. **Channel Quality Factor (CQF)**: This is used to evaluate the quality of EEG channels and identify any that are suboptimal.

We will discuss these on the following sections.

3. L/R Symmetric modeling

In this section, we'll cover the basics of *L/R Symmetric Modeling*.

Our inspiration came from textbooks[2], where we learned that the differences in signals from the left and right channels are key to telling General seizures apart from Lateral ones. That led us to design a model that treats left and right brain signals symmetrically, as illustrated in Fig. 1.

Here's how it works: The left and right brain signals are fed into a 1D CNN model separately, but they share the same parameters. The features we get from these are then processed in two ways: 1) through L/R invariant mapping, and 2) with a similarity encoder. We'll dive into these parts more in the sections that follow.

An important detail is our use of a late fusion approach. This means we input each channel separately into the 1D encoder to get a more abstract representation of the EEG channels.

We also discovered that gradually blending the EEG channels partway through the 1D encoder improved our results. We call this the Channel Mixer. While we didn't explore every possible architecture, adding a simple 1D convolution layer in each block was enough to see significant benefits.

3.1. L/R invariant mapping

This part of our model pulls out features that don't change whether we swap left and right EEG channels. We do this using two mathematical functions, f

and g, defined as follows:

$$u = f(x, y)v = g(x, y) \text{ where } f(x, y) = f(y, x) \text{ and } g(x, y) = g(y, x)$$

We considered several options for these functions, like:

1. diff, mean
2. prod, sum
3. min, max

In our initial tests, we found that the choice of function didn't drastically change the outcome. So, we settled on min/max as our default functions.

3.2. Similarity Encoder

To pinpoint the differences between the left and right sides, we measure the cosine similarity between paired L/R channels (for example, (Fp1, Fp2), (O1, O2), etc.). After calculating the similarity, we project it onto a d-dimensional vector to form a feature vector. This approach proved to be more effective than using the one-dimensional cosine similarity directly.

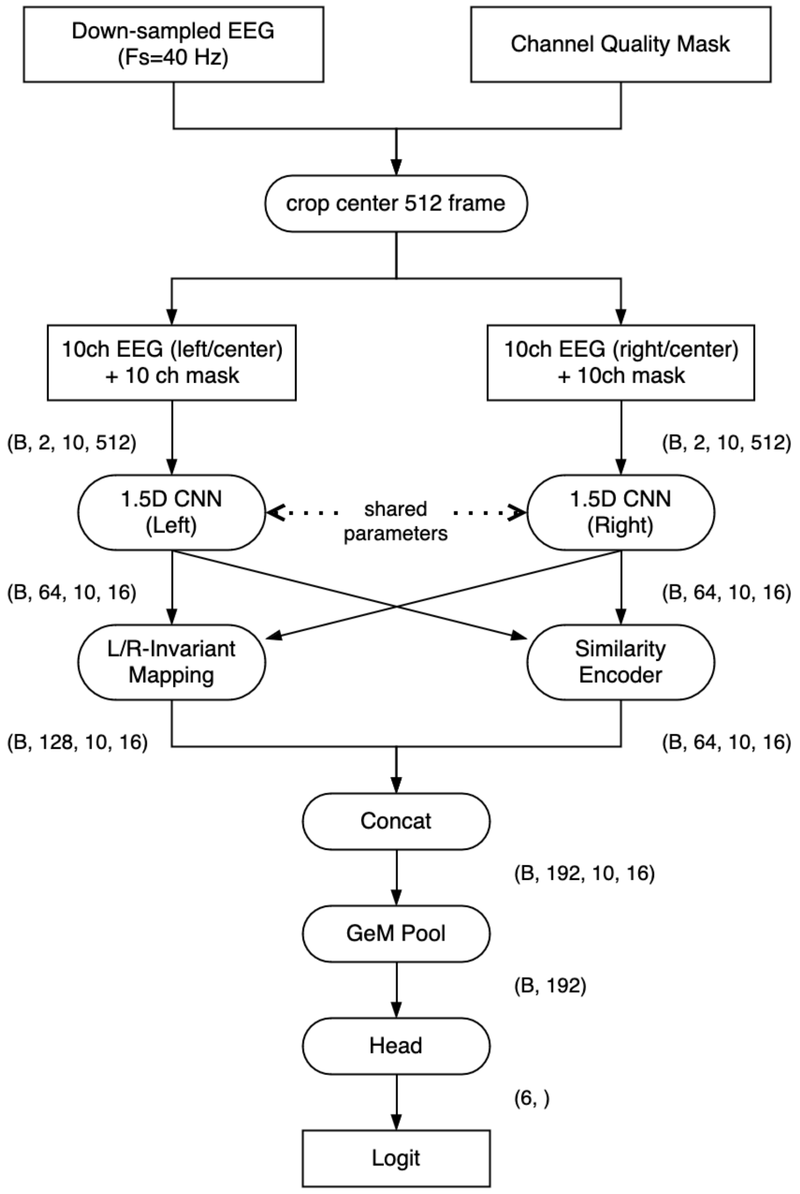

Fig. 1 Architecture of 1D Model Proposed

4. Channel Quality Factor (CQF)

During our initial EDA, we noticed a significant presence of bad channels in the EEG signals (see Fig. 2). This observation led us to devote effort to assessing channel quality, aiming to reduce potential confusion for our model.

The challenge of identifying corrupted channels is a well-known issue in EEG research. We discovered a study [1] that addresses this problem by introducing the *Locality Factor (LOF)*. LOF assesses the quality of a channel based on its discrepancy from other (usually normal) channels, with discrepancy measured by distance metrics like the Euclidean distance.

While [1] considers channels in a static manner, we adapted LOF for use with time-series data, allowing us to capture nuanced aspects of channel quality over time. We refer to this adapted feature as the Channel Quality Factor (CQF), illustrated in Fig. 3.

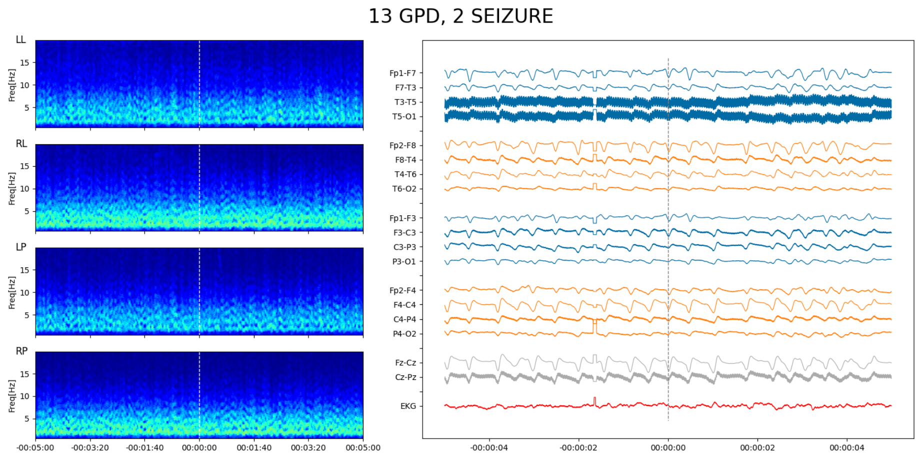

Fig. 2: Example of Bad Channels

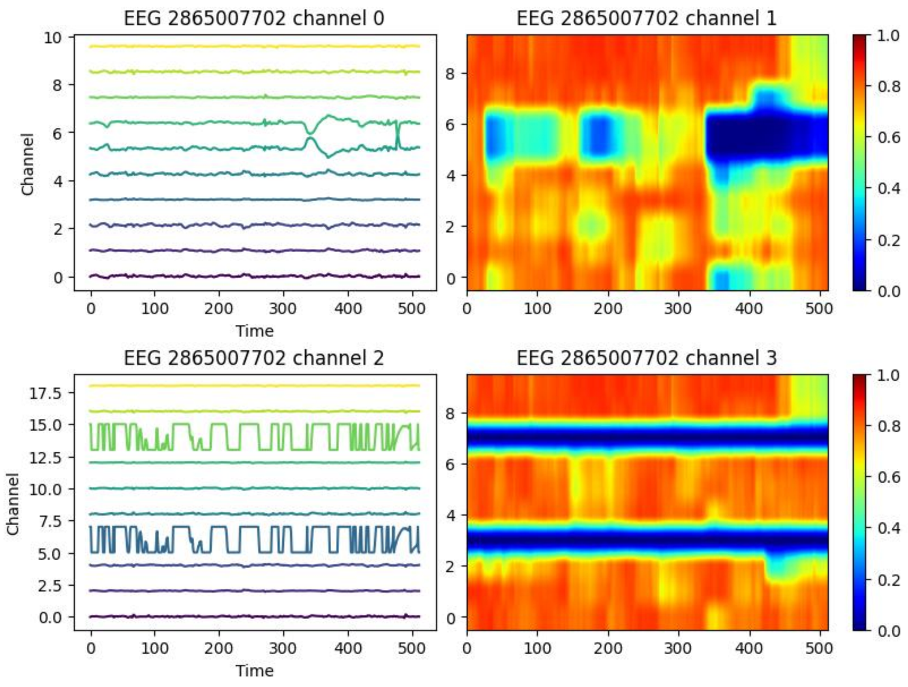

Fig. 3: CQF Illustration: Channels of lower quality are indicated in cooler colors.

4.1 Modification from LOF to CQF

In this subsection, we describe how CQF are modified from original LOF.

In the original definition of LOF[1],

$$\mathrm{LOF}_k(p) := \frac{1}{N} \sum_{o \in N_k(p)} \frac{\frac{1}{N} \sum_{o' \in N_k(p)} d(p, o')}{\frac{1}{N} \sum_{q' \in N_k(o)} d(o, q)} \tag{1}$$

Here,

- $N_k(p)$ is the set of channels within k-nearest neighbor from p.
- $d(p, q)$ is the distance between two EEG voltage sequence p, q.

For distance metrics, we chose Euclidian distance.

Intuitively, LOF indicates how the channel of interest is distant from other (usually non-outlier) channels. The numerator of equation (1) means *how far the channel from neighbor channels*, and the denominator means *average channel distance of nearest neighbors.* Thus, LOF means the relative metric of outlierness from neighbor channels. LOF gets larger if channel quality deteriorates.

Now, we will show step-by-step modification from LOF to CQF.

First, we replaced channel vector of full time series p to t-th sub-segment p_t to capture changing quality in time-domain.

$$\mathrm{LOF}_k(p) := \frac{1}{N} \sum_{o \in N_k(p_t)} \frac{\frac{1}{N} \sum_{o' \in N_k(p_t)} d(p_t, o'_t)}{\frac{1}{N} \sum_{q'_t \in N_k(o_t)} d(o_t, q_t)} \tag{2}$$

Next, we mapped LOF to indicates quality of channels:

$$\mathrm{CQF}_k(p) := \frac{1}{\mathrm{LOF}_k(p)/d_{\text{threshold}} + 1} \tag{3}$$

Here, $d_{\text{threshold}}$ is a hyper-parameter.

Note that $\mathrm{CQF}_k(p) \in [0, 1]$ since $\mathrm{LOF}_k(p) \in [0, \infty)$. CQF gets smaller if the channel quality deteriorates.

In some preliminary experiments, we visually examined whether LOF successfully detect outlier channels, and we found it fails to detect outlier channel if multiple outlier channels gather in a cluster: which means the numerator of equation (2) becomes small, and the LOF becomes smaller.

To solve the above issue, we replaced LOF to *GOF (Global Outlier Factor)* which is defined by:

$$\mathrm{GOF}_k(p) := \frac{1}{N} \sum_{o \in N_k(p_t)} \frac{\frac{1}{N} \sum_{o' \in N_k(p_t)} d(p_t, o'_t)}{d_{\text{top-k}}} \tag{4}$$

Here,

- $d_{\text{top-k}}$ is average top-k (closest) distance of all channel pairs.

Intuitively, GOF indicates relative channel qualify from finest channel groups.

Finally, we replaced LOF to GOF and obtain CQF we used in our experiments:

$$\widetilde{\mathrm{CQF}}_k(p) := \frac{1}{\mathrm{GOF}_k(p)/d_{\text{threshold}} + 1} \tag{5}$$

6. Result

The cross-validation scores (CVs) of our 1D models in the final submissions are listed below. These CVs were calculated using samples with num_votes >= 8.4.

exp_name	CV(num_votes>=8.4)	LB(Private)	LB(Public)
v5_eeg_24ep_cutmix	0.2477	0.327657	0.256772

We should emphasize our 1D model is really light weight. It only has 863K parameters and took 1-2 minutes for infer with test set with 15 models (3 seed x 5 fold ensemble).

7. Reference

- [1] Velu et. al., Adaptable and Robust EEG Bad Channel Detection Using Local Outlier Factor (LOF)
- [2] American Clinical Neurophysiology Society's Standardized Critical Care EEG Terminology: 2021 Version
- [3] https://github.com/huggingface/pytorch-image-models

Appendix

A1. Training Configuration

Most participants 2-stage training scheme, however, we schedule min_vote during 1-stage training schedule.

1. 0-15 epoch: train with all data
2. 16-23 epoch: train with num_votes>=8.4

Other training configurations are:

- Optimizer: Adam
- lr: 1e-3
- batch_size: 32
- weight_decay: 0.01
- Scheduler: cosine

A2. Augmentations

- channel shuffling: shuffling channels while keeping L/R symmetry
- L/R swap
- dropout (<=128 frames x 4)
- cutmix

A3. Backbone 1D CNN Architecture

We designed 1D-CNN based on timm[3]'s EfficientNet2d's building blocks. We called this architecture as *EfficientNet1d*.

```
================================================================
Layer (type:depth-idx)                                  Output Shape
```

```
===========================================================================================
EfficientNet1d                                                         [40, 64, 16]
 ConvBnAct2d: 1-1                                                     [4, 64, 10, 512]
     Sequential: 2-1                                                  [4, 64, 10, 512]
          Conv2d: 3-1                                                [4, 64, 10, 512]
          BatchNorm2d: 3-2                                           [4, 64, 10, 512]
          ELU: 3-3                                                   [4, 64, 10, 512]
 Sequential: 1-2                                                      --
     ResBlock2d: 2-2                                                  [4, 64, 10, 256]
          MaxPool2d: 3-4                                             [4, 64, 10, 256]
          Sequential: 3-5                                            [4, 64, 10, 256]
              Sequential: 4-1                                        [4, 64, 10, 512]
                   InvertedResidual: 5-1                            [4, 64, 10, 512]
                   InvertedResidual: 5-2                            [4, 64, 10, 512]
                   InvertedResidual: 5-3                            [4, 64, 10, 512]
                   DepthWiseSeparableConv: 5-4                      [4, 64, 10, 512]
              MaxPool2d: 4-2                                         [4, 64, 10, 256]
     ResBlock2d: 2-3                                                  [4, 64, 10, 128]
          MaxPool2d: 3-6                                             [4, 64, 10, 128]
          Sequential: 3-7                                            [4, 64, 10, 128]
              Sequential: 4-3                                        [4, 64, 10, 256]
                   InvertedResidual: 5-5                            [4, 64, 10, 256]
                   InvertedResidual: 5-6                            [4, 64, 10, 256]
                   InvertedResidual: 5-7                            [4, 64, 10, 256]
                   DepthWiseSeparableConv: 5-8                      [4, 64, 10, 256]
              MaxPool2d: 4-4                                         [4, 64, 10, 128]
     ResBlock2d: 2-4                                                  [4, 64, 10, 64]
          MaxPool2d: 3-8                                             [4, 64, 10, 64]
          Sequential: 3-9                                            [4, 64, 10, 64]
              Sequential: 4-5                                        [4, 64, 10, 128]
                   InvertedResidual: 5-9                            [4, 64, 10, 128]
                   InvertedResidual: 5-10                           [4, 64, 10, 128]
                   InvertedResidual: 5-11                           [4, 64, 10, 128]
                   DepthWiseSeparableConv: 5-12                     [4, 64, 10, 128]
              MaxPool2d: 4-6                                         [4, 64, 10, 64]
     ResBlock2d: 2-5                                                  [4, 64, 10, 32]
          MaxPool2d: 3-10                                            [4, 64, 10, 32]
          Sequential: 3-11                                           [4, 64, 10, 32]
              Sequential: 4-7                                        [4, 64, 10, 64]
                   InvertedResidual: 5-13                           [4, 64, 10, 64]
                   InvertedResidual: 5-14                           [4, 64, 10, 64]
                   InvertedResidual: 5-15                           [4, 64, 10, 64]
                   DepthWiseSeparableConv: 5-16                     [4, 64, 10, 64]
              MaxPool2d: 4-8                                         [4, 64, 10, 32]
     ResBlock2d: 2-6                                                  [4, 64, 10, 16]
          MaxPool2d: 3-12                                            [4, 64, 10, 16]
          Sequential: 3-13                                            [4, 64, 10, 16]
               Sequential: 4-9                                       [4, 64, 10, 32]
                   InvertedResidual: 5-17                           [4, 64, 10, 32]
                   InvertedResidual: 5-18                           [4, 64, 10, 32]
                   InvertedResidual: 5-19                           [4, 64, 10, 32]
                   DepthWiseSeparableConv: 5-20                     [4, 64, 10, 32]
               MaxPool2d: 4-10                                       [4, 64, 10, 16]
===========================================================================================
Total params: 863,504
Trainable params: 863,504
Non-trainable params: 0
Total mult-adds (G): 2.72
===========================================================================================
Input size (MB): 0.16
Forward/backward pass size (MB): 833.78
Params size (MB): 3.45
Estimated Total Size (MB): 837.40
===========================================================================================
```