Spatiotemporal Pooling on Appropriate Topological Maps Represented as Two-Dimensional Images for EEG Classification

Takuto Fukushima and Ryusuke Miyamoto

[1] Department of Computer Science, Graduate School of Science and Technology
[2] Department of Computer Science, School of Science and Technology
Meiji University, Kawasaki, Japan
{taku,miya}@cs.meiji.ac.jp

1 Details of our model

This section describes the details of our model, including the hyper-parameters, #params.

1.1 Detailed Hyper-parameters

Table 1: Hyper-parameters of our model.

Hyper-parameter	Value
Number of layers in a MLP	2
mlp ratio (InternImage)	4.0
mlp ratio (ST-pooling)	4.0
drop rate (InternImage)	0.0
drop rate (ST-pooling)	0.1
drop path rate (InternImage)	0.4
drop path rate (ST-pooling)	0.0
topological map size	74
Data augmentation	noise, CutMix, MixUp
CutMix α	1.0
MixUp α	0.8
CutMix-MixUp switch prob	0.5
label smoothing	0
Epochs	50
learning rate	$1e-4$
batch size	8
Adam ϵ	$1e-8$
Adam (β_1, β_2)	(0.9, 0.999)
Weight decay	0
t-SNE perplexity	30.0

In this study, our model was trained using the hyper-parameters shown in Tab. 1.

1.2 #params of our model

The #params of our model is 55M when N, H, and W are 60, 74, and 74, respectively.

2 Details of comparison methods

This section describes the details of the methods used for comparison.

2.1 Details of HCANN

HCANN [4] is the model that combines CNN and multi-head attention for EEG classification and represents state-of-the-art model for BCI Competition IV 2a [1]. Furthermore, HCANN is the only model with open-source code available among the previous models compared in Tab. 2 of the main manuscript.

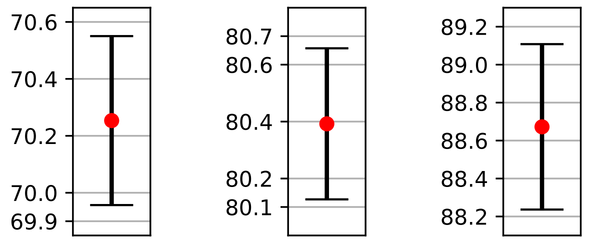
Data preprocessing for HCANN We filtered the data using a causal third-order Butterworth filter with a cut-off frequency ranging from 4 to 38Hz, following the same method applied to BCI Competition IV 2a as described in [4]. Additionally, we used Z-score normalization to normalize the EEG signals, as it was essential for ensuring effective learning. Without Z-score normalization, the model did not perform well during training.

2.2 Details of hyper-parameters for models used in comparison

The hyper-parameters of the models, which were used to verify the effect of ST-pooling, are shown in Tab. 2.

Table 2: Hyper-parameters of models used in comparison.

Method	Hyper-parameter	Value
PoolFormer	model	PoolFormerV2-S24 [8]
	pretrained	No
Multi-Head Attention	mlp ratio	4.0
	Number of layers	2



(a) L/R/O/F(%). **(b)** L/R/O(%). **(c)** L/R(%).

Fig. 1: 95% confidence interval error bars.

3 Details of PhysioNet EEG Motor Movement/Imagery Dataset

This section describes the detail method of data acquisition from the PhysioNet EEG Motor Movement/Imagery Dataset [3, 6].

For each subject, 21 trials were selected per class. The annotation time for the L, R, and F tasks is 4.1 seconds per trial. To acquire one second of rest before and after motor imagery and four seconds of motor imagery, similar to previous studies [2, 7], we first collected data for one second of rest before motor imagery and the first four seconds of motor imagery. Subsequently, we acquired data for one second of rest after the end of motor imagery. For the O tasks, similar to previous study [2], we randomly acquired continuous data for six seconds.

The chance levels for four-class, three-class and two-class classifications are 1/4, 1/3, and 1/2, respectively, because of the equal number of samples in each class.

4 Additional research

This section describes additional research that strengthens our conclusions.

4.1 Multiple random subject splits

First, the subjects were divided using four different seed values, and the results are shown in Tabs. 3, 4 and 5. In these tables, 0, 1, 2, 3, and 4 in the leftmost column show the indices of cross-validation. The column names 7, 42, 79, and 3407 represent seed values used for random splits. The column labeled "orig" corresponds to the same splits used in the main manuscript. We couldn't know the seed values of "orig" splits because we use GroupKFold from scikit-learn [5] when we use "orig" splits. The 95% confidence intervals are represented as error-bars in Figs. 1a, 1b and 1c. These confidence intervals were calculated based on the Student's t-distribution owing to the small sample size.

The results indicate that the improvement in accuracy over existing studies is not owing to the randomness of subject split.

Table 3: Accuracy for L/R/O/F.

	orig	7	42	79	3407
0	69.44	71.83	69.44	71.43	67.23
1	74.09	68.42	71.37	72.39	71.60
2	66.61	71.71	70.63	70.01	67.97
3	69.35	66.01	71.37	69.82	72.56
4	71.49	72.74	68.15	66.67	73.99
Avg	70.20	70.14	70.20	70.06	70.67
σ	2.49	2.53	1.24	1.94	2.63

Table 4: Accuracy for L/R/O.

	orig	7	42	79	3407
0	80.73	82.01	80.42	81.71	77.70
1	82.99	77.85	79.29	82.01	80.57
2	78.31	81.93	80.73	80.27	78.99
3	78.02	77.78	82.70	80.32	82.38
4	83.41	81.83	79.13	76.27	82.46
Avg	80.69	80.28	80.45	80.12	80.42
σ	2.26	2.01	1.28	2.05	1.87

Table 5: Accuracy for L/R.

	orig	7	42	79	3407
0	89.57	89.68	89.34	91.50	88.32
1	90.25	85.49	88.55	91.16	89.34
2	84.92	88.10	87.53	87.19	87.19
3	88.57	87.98	90.95	90.12	88.45
4	89.52	90.60	86.31	86.43	89.76
Avg	88.57	88.37	88.54	89.28	88.61
σ	1.90	1.74	1.58	2.08	0.89

4.2 Random coordinate transformation

Table 6: Accuracy for L/R/O/F.

	7	31	42	79	3407
0	67.06	67.29	67.40	63.32	67.01
1	72.17	68.99	71.32	71.77	71.03
2	63.95	62.64	63.89	62.87	61.28
3	67.50	66.90	68.10	67.08	65.83
4	69.17	70.42	69.05	68.51	67.98
Avg	67.97	67.25	67.95	66.71	66.63
σ	2.69	2.62	2.42	3.32	3.18

Next, we performed random coordinate transformations. Accuracy for the L/R/O/F task was measured five times using random projection with five different seeds. Additionally, the subject splits were “orig” splits. The results are shown in Tab. 6, which has the same structure as described in Tabs. 3, 4 and 5. The 95% confidence interval is represented as error-bar in Fig. 2. This confidence

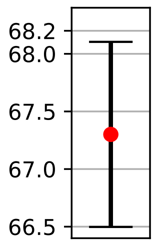


Fig. 2: 95% confidence interval error bar.

interval was calculated using the Student's t-distribution because of the small sample size.

The result shows the effectiveness of t-SNE and demonstrates that the coordinate transformation of electrodes significantly affects the accuracy.

5 Code in PyTorch

This section describes the code of our model in a format similar to PyTorch code. Algorithm 1 and 2 show the code of the ST-pooling and all of our model, respectively.

Algorithm 1 ST-pooling, PyTorch-like Code.

```
from torch import nn
import torch
from timm.models.layers import DropPath
import copy

class Pooling(nn.Module):
    '''
    Pooling layer for PoolFormer basic block
    '''
    def __init__(self, pool_size=3):
        super().__init__()
        self.pool = nn.AvgPool2d(
            pool_size, stride=1, padding=pool_size//2, count_include_pad=False)

    def forward(self, x):
        return self.pool(x) - x

class PoolFormerBlock(nn.Module):
    '''
    Implementation of one PoolFormer block.
    Modified from:
    https://github.com/sail-sg/poolformer/blob/main/models/poolformer.py
    '''
    def __init__(self, L, dropout=0.1, drop_path_rate=0, mlp_ratio=4, layer_scale=None):
        super().__init__()
        self.pooling = Pooling()
        self.norm1 = nn.LayerNorm(L, eps=1e-5)
        self.norm2 = nn.LayerNorm(L, eps=1e-5)
        hidden_dim = int(L * mlp_ratio)
        self.linear1 = nn.Linear(L,hidden_dim)
        self.linear2 = nn.Linear(hidden_dim,L)

        self.layer_scale_1 = nn.Parameter(layer_scale * torch.ones((L)), requires_grad=True)
        self.layer_scale_2 = nn.Parameter(layer_scale * torch.ones((L)), requires_grad=True)
        self.droppath = DropPath(drop_path_rate)
        self.act = nn.ReLU()
        self.dropout = nn.Dropout(dropout)

    def forward(self,x):
        x = x + self.droppath(self.layer_scale_1 * self.pooling(self.norm1(x)))
        x = x + self.droppath(self.layer_scale_2 * self.linear2(self.dropout(self.act(self.
             linear1(self.norm2(x))))))
        return x

class STpooling(nn.Module):
    '''
    Implementation of ST-pooling
    --L: feature vector length
    --dropout: dropout rate
    --drop_path_rate: drop path rate
    --mlp_ratio: mlp ratio
    --layer_scale: layer scale
    '''
    def __init__(self, L, dropout=0.1, drop_path_rate=0, mlp_ratio=4, layer_scale=None):
        super().__init__()
        self.pool_former_blocks = _get_clones(PoolFormerBlock(L, dropout, drop_path_rate,
             mlp_ratio, layer_scale), 2)

    def forward(self, x):
        for block in self.pool_former_blocks:
            x = block(x)
        return x

def _get_clones(module, N):
    return nn.ModuleList([copy.deepcopy(module) for i in range(N)])
```

Algorithm 2 ProposedModel, PyTorch-like Code.

```
import InternImage
import STpooling
import torch
from torch import nn
import math

class ProposedModel(nn.Module):
    '''
    Implementation of our proposed model
    --core_op: core operation of InternImage
    --num_classes: number of classes
    --N: number of frames
    --intern_drop: dropout rate of InternImage
    --intern_drop_path: drop path rate of InternImage
    --pool_drop: dropout rate of ST-pooling
    --pool_drop_path: drop path rate of ST-pooling
    --pool_mlp_ratio: mlp ratio of ST-pooling
    --pool_layer_scale: layer scale of ST-pooling
    '''
    def __init__(self, core_op, num_classes, N, intern_drop=0.0, intern_drop_path=0.4,
          pool_drop=0.1, pool_drop_path=0, pool_mlp_ratio=4.0, pool_layer_scale=1e-5):
        super().__init__()
        # InternImage-S
        intern_model = InternImage(core_op=core_op, channels=80, in_chans=1, depths=[4, 4, 21,
               4], groups=[5, 10, 20, 40], layer_scale=1e-5, offset_scale=1.0, mlp_ratio=4.0,
              post_norm=True, drop_path_rate=intern_drop_path, drop_rate=intern_drop)
        self.intern_model = intern_model

        # feature vector length
        L = 640

        # remove InternImage head
        self.intern_model.conv_head = nn.Identity()
        self.intern_model.head = nn.Identity()

        # calculate positional encoding
        device = torch.cuda.current_device()
        position = torch.arange(0, N).unsqueeze(1).float()
        div_term = torch.exp(torch.arange(0, L, 2).float() * -(math.log(10000.0) / L))
        self.positional_enc = torch.zeros(N, L, device=device)
        self.positional_enc[:, 0::2] = torch.sin(position * div_term)
        self.positional_enc[:, 1::2] = torch.cos(position * div_term)

        self.st_pooling = STpooling(L, dropout=pool_drop, drop_path_rate=pool_drop_path,
              mlp_ratio=pool_mlp_ratio, layer_scale=pool_layer_scale)
        self.norm = nn.LayerNorm(L, eps=1e-5)
        self.fc = nn.Linear(N * L, num_classes)
        self.N = N
        self.final_features = N * L

    def forward(self,x):
        # InternImage
        x = x.reshape(-1, 1, 74, 74)
        x = self.intern_model(x)

        # ST-pooling
        # (B * N, L) -> (B, N, L)
        x = x.reshape(-1, self.N, x.shape[1])
        x += self.positional_enc
        x = self.st_pooling(x)
        x = self.norm(x)

        # head
        # (B, N, L) -> (B, N * L)
        x = x.reshape(-1, self.final_features)
        x = self.fc(x)
        return x
```

References

1. Brunner, C., Leeb, R., Müller-Putz, G., Schlögl, A., Pfurtscheller, G.: BCI Competition 2008–Graz data set A. Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology **16**, 1–6 (2008)
2. Dose, H., Møller, J.S., Iversen, H.K., Puthusserypady, S.: An end-to-end deep learning approach to MI-EEG signal classification for BCIs. Expert Syst. Appl. **114**, 532–542 (2018). https://doi.org/10.1016/j.eswa.2018.08.031
3. Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: PhysioBank, PhysioToolkit, and PhysioNet Components of a New Research Resource for Complex Physiologic Signals. Circulation **101**(23), e215–e220 (2000). https://doi.org/10.1161/01.CIR.101.23.e215
4. Ji, Y., Li, F., Fu, B., Zhou, Y., Wu, H., Li, Y., Li, X., Shi, G.: A novel hybrid decoding neural network for EEG signal representation. Pattern Recognition p. 110726 (2024). https://doi.org/https://doi.org/10.1016/j.patcog.2024.110726
5. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. JMLR **12**, 2825–2830 (2011)
6. Schalk, G., McFarland, D., Hinterberger, T., Birbaumer, N., Wolpaw, J.: BCI2000: a general-purpose brain-computer interface (BCI) system. IEEE Trans. Biomed. Eng. **51**(6), 1034–1043 (2004). https://doi.org/10.1109/TBME.2004.827072
7. Xie, J., Zhang, J., Sun, J., Ma, Z., Qin, L., Li, G., Zhou, H., Zhan, Y.: A Transformer-Based Approach Combining Deep Learning Network and Spatial-Temporal Information for Raw EEG Classification. IEEE Trans. Neural Syst. Rehabil. Eng. **30**, 2126–2136 (2022). https://doi.org/10.1109/TNSRE.2022.3194600
8. Yu, W., Si, C., Zhou, P., Luo, M., Zhou, Y., Feng, J., Yan, S., Wang, X.: MetaFormer Baselines for Vision. PAMI **46**(2), 896–912 (2024). https://doi.org/10.1109/TPAMI.2023.3329173