

Appendix

Algorithm 1: Pseudo code of attention layer in a PyTorch-like style.

```

1 /* B: batch size, H: height, W:
   width, C: channel number, M: head
   number, ak: Attention kernel
   size, s: stride */
   Input : query(Q), key (K), and value (V)
2 // ##### Initialization #####
3 Unfold = nn.Unfold(ak, s);
4 Fold = nn.Fold(output_size = (H, W), ak, s);
5 AvgPooling = nn.AdaptiveAvgPool3d((M, H//2,
   W//2))
6 // ##### Forward pass #####
7 Q = Q.view(B*M, C, H, W);
8 K = K.view(B*M, C, H, W);
9 V = V.view(B*M, C, H, W);
10 /*  $\mathcal{T}(\bullet)$  in Eq. (3) :
   ( $K, \prod(\text{Patch Size}), B$ ) */
11 unfold_q = Unfold(Q).view(B*M, C, ak*ak, -1);
12 unfold_k = Unfold(K).view(B*M, C, ak*ak, -1);
13 unfold_v = Unfold(V).view(B*M, C, ak*ak, -1);
14 //  $Q*K: A(q_i^p, K_j^p)^b$  in Eq. (4)
15 attention = unfold_q*unfold_k;
16 attention = Softmax(attention);
17 //  $\text{Attention}(Q^p, K^p, V^p)^b$  in Eq. (5)
18 out_att = (attention*unfold_v);
19 out_att = out_att.view(B*M, C*ak*ak, -1);
20 //  $\mathcal{T}(\bullet)^{-1}$  in Eq. (6)
21 out_att = Fold(out_att);
22 // Adaptive average pooling in
   Eq. (7)
23 out_att = AvgPooling(out_att)
24 Return out_att

```

A. Setting of Spectrogram Parameters

In Table I, we show the parameters used to create spectrograms from SHL 2018, WISDM, and SLEEP-EDF-20 in our paper.

TABLE I
THE SETTINGS OF SPECTROGRAM PARAMETERS.

Dataset	Sampling Frequency	Size of FFT	Length of hop	Padding
SHL 2018	100Hz	500	10	Yes
WISDM	20Hz	40	10	Yes
SLEEP-EDF-20	100Hz	500	10	Yes

-1,-1	-1,-0.3	-1,0.3	-1,1
-0.3,-1	-0.3,-0.3	-0.3,0.3	-0.3,1
0.3,-1	0.3,-0.3	0.3,0.3	0.3,1
1,-1	1,-0.3	1,0.3	1,1

Fig. 1. An example of relative distance computation (a 4×4 spectrogram). The relative distances are computed with respect to the position of the highlighted pixel. The format of distances is **row offset**, **column offset** ($\mathbb{R}^{2 \times 4 \times 4}$).

B. Positional Encoding

Positional encoding is widely adopted in self-attention modules. In this paper, we adopt positional encoding in the adaptive convolution module. Specifically, the popular relative positional encoding [1] is adopted when computing the attention weights. The row and column offsets are associated with an pixel embedding ($pixel_embedding \in [-1, 1]$), as shown in Fig. 1. The relative distances are computed using a 1×1 convolution, where the embedding is convert from $\mathbb{R}^{2 \times H \times W}$ to $\mathbb{R}^{1 \times H \times W}$ (position encoding).

C. Pseudo code of Attention Layer

As the core operations \mathcal{T} and \mathcal{T}^{-1} in the attention layer are implemented using the PyTorch functions, to facilitate understanding, a pseudo code of the attention layer is illustrated in Algorithm 1 using a PyTorch-like style.

REFERENCES

- [1] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in Neural Information Processing Systems*, 32, 2019.