Temporal Lift Pooling for Continuous Sign Language Recognition

Lianyu Hu[1], Liqing Gao[1], Zekang Liu[1], and Wei Feng[✉1]

Tianjin University, Tianjin 300350, China
{hly2021,lqgao,lzk100953}@tju.edu.cn;wfeng@ieee.org

Abstract. Pooling methods are necessities for modern neural networks for increasing receptive fields and lowering down computational costs. However, commonly used hand-crafted pooling approaches, e.g., max pooling and average pooling, may not well preserve discriminative features. While many researchers have elaborately designed various pooling variants in spatial domain to handle these limitations with much progress, the temporal aspect is rarely visited where directly applying hand-crafted methods or these specialized spatial variants may not be optimal. In this paper, we derive temporal lift pooling (TLP) from the Lifting Scheme in signal processing to intelligently downsample features of different temporal hierarchies. The Lifting Scheme factorizes input signals into various sub-bands with different frequency, which can be viewed as different temporal movement patterns. Our TLP is a three-stage procedure, which performs signal decomposition, component weighting and information fusion to generate a refined downsized feature map. We select a typical temporal task with long sequences, i.e. continuous sign language recognition (CSLR), as our testbed to verify the effectiveness of TLP. Experiments on two large-scale datasets show TLP outperforms hand-crafted methods and specialized spatial variants by a large margin (1.5%) with similar computational overhead. As a robust feature extractor, TLP exhibits great generalizability upon multiple backbones on various datasets and achieves new state-of-the-art results on two large-scale CSLR datasets. Visualizations further demonstrate the mechanism of TLP in correcting gloss borders. Code is released[1].

Keywords: Lifting Scheme, Continuous Sign Language Recognition, Temporal Lift Pooling

1 Introduction

Sign language is one of the most commonly used communication tools for the deaf people. However, mastering this language is difficult for the hearing people, which forms an obstacle for communication between two groups. To handle this problem, continuous sign language recognition (CSLR) aims to translate sign videos into corresponding gloss sentences, which is feasible to bridge this gap.

[1] https://github.com/hulianyuyy/Temporal-Lift-Pooling

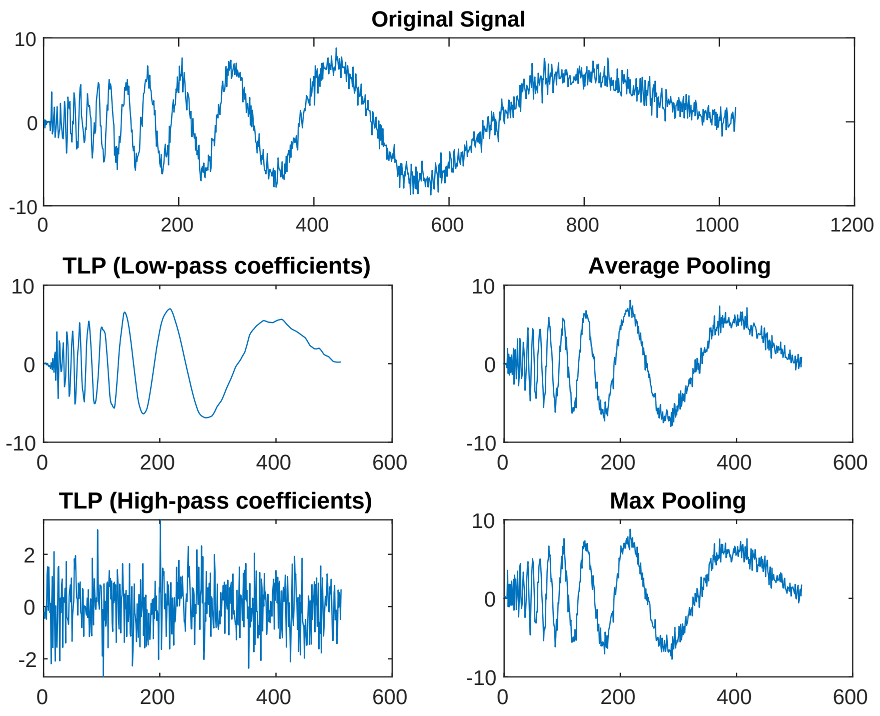

Fig. 1: Effects of temporal lift pooling (TLP) and hand-crafted pooling methods. TLP clearly decomposes input signal into various temporal patterns while hand-crafted pooling methods can't well distinguish noise from body movements.

Pooling methods are necessities for modern neural networks (NNs) for increasing receptive fields and generating discriminative representations. Several simple pooling methods, like max pooling and average pooling, are broadly employed in various domains [1,24,38] for their remarkable generalizability. While effective and efficient, simply using these hand-crafted methods may not fully consider local structures and optimally preserve features of different hierarchies. For the spatial domain, many researchers [38,13,40,31,32,39,11] have realized the limitations of hand-crafted pooling and elaborately designed many downsampling approaches for better preserving details. However, the temporal aspect is rarely explored where directly applying hand-crafted methods or these spatially specialized variants may not fit the temporal pattern well.

Our method is inspired by the Lifting Scheme [33] from signal processing, which is commonly used in information compression [27], reconstruction [7] and denoising [37]. The Lifting Scheme decomposes an input signal into various sub-bands with downscaled sizes of different frequencies, which is ideal for joint time-frequency analysis. Applying the idea of Lifting Scheme, we present temporal lift pooling (TLP) to factorize inputs into major and adjunctive movements and integrate them into a downsized refined representation. As shown in Fig. 1, the low-pass coefficients generated by TLP smoothly restore original low-frequency signals, which can be viewed as body movement patterns. The high-pass coefficients extract high-frequency components from inputs that represent detailed dynamics. In contrast, hand-crafted max pooling and average pooling fail to deal with mixed inputs and even amplify extremes or lose details sometimes.

TLP is consisted of three stages: lifting process, component weighting and fusion, which step by step decomposes input signal and reweights its components for a unified output. As a plug-and-play tool, TLP is implemented with tiny convolutional neural networks with only additional 0.4% computational costs. As an effective downsampling unit, it exhibits excellent generalizability upon multiple backbones on various datasets. By only replacing two pooling locations with TLP, a significant 1.5% performance boost is witnessed, which largely surpasses the hand-crafted methods and spatial pooling variants on CSLR. Besides, TLP achieves new state-of-the-art results on two large-scale CSLR datasets. Visualizations present the effects of TLP to correct gloss borders on accurate recognition.

2 Related Work

2.1 Continuous Sign Language Recognition

Earlier methods [10,14,8,21] in CSLR always employ hand-crafted features or HMM-based systems [22,23] to perform temporal modeling and translate sentences step by step. HMM-hybrid methods [22,23] typically first employ a feature extractor for representative features and then adopt an HMM for long-term temporal modeling. The success of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) bring huge progress for CSLR. CTC loss [12] provides a new perspective to align target sentences with input frames which is broadly used by recent CSLR approaches [29,28,3,6,26,25]. They first rely on a feature extractor, i.e. 3D or 2D&1D CNN hybrids, to extract frame-wise features, and then adopt a LSTM module for capturing long-term temporal dependencies. However, several methods [29,6] found in such conditions the feature extractor is not well trained. Some recent approaches present the iterative training strategy to relieve this problem, but consume much more computations and multiple training stages. More recent studies [25,3,28] try to directly enhance the feature extractor by adding alignment losses [25] or adopt pseudo labels for supervision to tackle this issue [3].

2.2 Pooling Methods

Pooling has been commonly used in modern NNs for discriminative representations and reducing computational costs since Neocognitron [9]. Previous studies mainly focus on pooling in the spatial domain but rarely explore the temporal side. Max pooling and average pooling are two commonly used flexible hand-crafted methods in various tasks which could be dated back to LeNet [24] periods. Boureau et al. [1] prove max pooling can preserve more discriminative features than average pooling in terms of probability. Apart from these simple hand-crafted methods, various pooling variants have been proposed to better preserve details while maintaining efficiency. L_p pooling [13] introduces L_p norm to activate and normalize outputs, which can be viewed as a continuum between max and average pooling. Mixed pooling [38] tries to integrate the characteristics of max pooling and average pooling by a learned coefficient for better performance. Stochastic pooling [39] presents a multinomial sampling algorithm to select output values in the sampling window. S3Pool [40] attempts to introduce regulation in rows and columns, which can be viewed as some kind of data augmentation. Detail-preserving pooling (DPP) [31] aims to preserve details in 2D grids by selecting discriminative responses. LIP [11] formulates existing pooling methods under a general framework and designs a tiny convolutional network to generate local importance for values in a sampling window. Softpool [32] employs the softmax function to measure the contribution of values and adopts the normalized outputs as downscaled contents. LiftPool [41] introduces Lifting Scheme to design both downsampling and upsampling variants. However, it mainly tackles spatial tasks and doesn't consider temporal patterns. Besides, it fails to further measure the contribution of different components in sub-bands and doesn't deal with hierarchical features. Especially, all these approaches focus on the spatial aspect but don't explore the temporal side, while not all of them (e.g., DPP [31] and S3Pool [40]) are directly applicable for temporal tasks. Besides, directly applying hand-crafted pooling methods or these specialized spatial variants may not be optimal for temporal modeling. As shown in the experiments, our TLP surpasses all these counterparts by a large margin.

3 Methods

3.1 Overview

As shown in Fig. 2, recent CSLR methods [29,28,3,6,26,25] usually first employ a common 2D CNN to extract frame-wise features, and then deploy a 1D CNN consisted of a sequence of 1D Conv and pooling methods to model short-term temporal dependencies, followed by a two-layer BiLSTM and classifier for sentence prediction. Especially, two pooling layers are adopted in the 1D CNN to squeeze input length for downsampled discriminative representations to predict sentences. Practically, max pooling with kernel size of 2 and stride of 2 is used as default. As the downsampling process is intrinsically lossy, it's necessary to consider which information to be kept for subsequent sentence prediction. Inappropriate downsampling may lead to beneficial information loss and movement pattern deformation, thus affecting recognition performance. In this paper, we refer to Lifting Scheme [33] originated from signal processing to handle this issue and derive an efficient and effective pooling method.

3.2 Temporal Lift Pooling

Pooling methods are necessities for reducing computational costs and obtaining discriminative representations for temporal tasks with long input sequences,

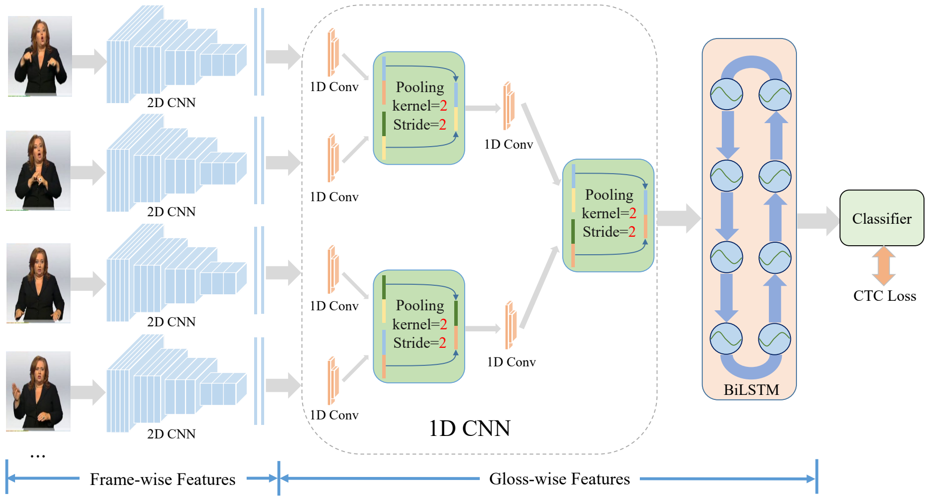

Fig. 2: Overview of recent CSLR methods. They first employ a common 2D CNN to extract frame-wise features, and then adopt a 1D CNN to perform short-term temporal modeling. A two-layer BiLSTM is used to capture long-term temporal dependencies, followed by a classifier to predict sentences. Especially, two pooling layers are adopted in 1D CNN for shortened discriminative representation. Practically, max pooling with kernel size of 2 and stride of 2 is used as default. We replace them with TLP to intelligently preserve discriminative features.

e.g., CSLR. Commonly used hand-crafted pooling methods may not well consider local patterns and don't optimally preserve critical representations. We derive temporal lift pooing (TLP) from the Lifting Scheme to exploit temporal correlations in signals to build a downsized approximation.

As shown in Fig. 3(a), our TLP is composed of three stages, i.e., lifting process, component weighting and fusion. We will detail them one by one.

Lifting process. Given a 1D temporal signal $x = [x_1, x_2, x_3, \ldots, x_t]$ ($x \in \mathcal{R}^{C\times T}, t \in \mathcal{N}+$) where C denotes channel and T represents the sequence length, lifting process decomposes x into a downsized approximation s and a difference signal d as :

$$s, d = \mathcal{F}(x). \tag{1}$$

Here, $\mathcal{F}(\cdot) = f_{update} \circ f_{predict} \circ f_{split}$ is consisted of three functions: split, predict and update as shown in Fig. 3, where $\circ$ is the function composition operator.

Split f_{split}: $x \mapsto (x_e, x_o)$. It partitions input signal x into two disjoint sets x_e, x_o for downsized signal generation. Practically, x_e and x_o are generated with even and odd indices, respectively, where x_e= $[x_2, x_4, \ldots, x_{2k}]$ and x_o= $[x_1, x_3, \ldots, x_{2k-1}]$ ($k \in \mathcal{N}+$) are temporally closely correlated.

Predict $f_{predict}$: $(x_e, x_o) \mapsto d$. Given a selected set, e.g., x_e, $f_{predict}$ predicts another set x_o by a predictor $\mathcal{P}(\cdot)$. As only one basis x_e is given, the prediction

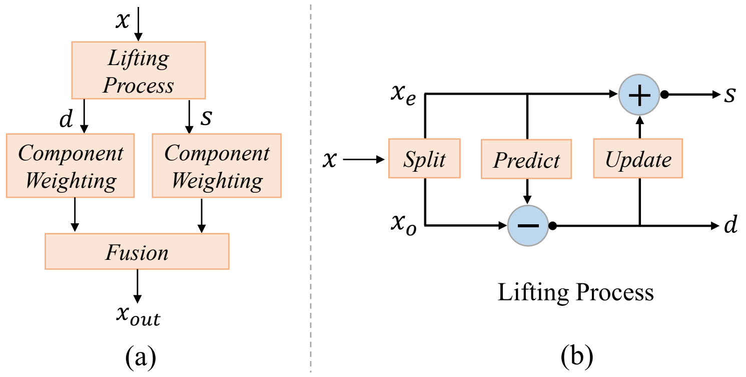

Fig. 3: (a) Overview for temporal lift pooling of three stages: lifting process, component weighting and fusion. (b) Lifting process. x is split into x_e and x_o, where the predictor and updater generate an approximation s and a difference signal d.

is not required to be precise, which is expected to be collaborated with following f_{update}. So the difference signal d with high-pass coefficients is obtained as :

$$d = x_o - \mathcal{P}(x_e). \tag{2}$$

Update f_{update}: $(x_e, d) \mapsto s$. As the prediction process is undoubtedly aliasing and simply taking alternately sampled x_e as the approximation of x will cause inevitably information loss, an update function $\mathcal{U}(\cdot)$ takes the difference d as input for compensation and generates the smoothed downsized representations s as :

$$s = x_e + \mathcal{U}(d). \tag{3}$$

This update procedure can be viewed as applying a low-pass filter for x and thus s is the downsized approximation of original signal with low-pass coefficients. It's worth noting that the prediction and update procedure are intrinsically adversarial. If $f_{predict}$ precisely predicts x_o based on x_e, the difference signal d will be minor and thus the approximation s will be extremely biased towards x_e, resulting in aliasing downsampling. If $f_{predict}$ can't well perform prediction, the difference signal d will be huge and make the approximation s deformed from original signal x. Thus, $f_{predict}$ and $f_{predict}$ works in an antagonistic way, expecting to generate discriminative and detailed signals, s and d, respectively.

The classic Lifting Scheme methods apply predefined low-pass filters and high-pass filters to decompose x into different sub-bands. However, manually designing filters for $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ is difficult [42] and can't fit various conditions. Previously, [42] proposed to optimize filters in $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ by backward propagated gradients for signal processing. We design $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ with tiny fully convolutional networks which are optimized in an end-to-end manner as:

$$\mathcal{P}(\cdot) = \mathrm{Tanh}() \circ \mathrm{Conv}(k=1) \circ \mathrm{ReLU}() \circ \mathrm{Conv}(k=K, g=C_{in}), \tag{4}$$

$$\mathcal{U}(\cdot) = \mathrm{Tanh}() \circ \mathrm{Conv}(k = 1) \circ \mathrm{ReLU}() \circ \mathrm{Conv}(k = K, g = C_{in}). \tag{5}$$

Here k denotes the kernel size and g represent the group number for convolution. We prefer to first deploy a 1D depth-wise convolution with kernel size K to aggregate local temporal patterns, followed by a ReLU activation. And then we use a normal 1×1 convolution to enable channel-wise aggregation, followed by a Tanh activation for feature prediction.

For generating discriminative representations, two loss constraints are employed apart from original task loss. Recall that the downsized approximation s is originated from x_e by Eq. 3, it's essentially close to x_e. We employ loss C_u to encourage x_e to approximate x_o as well, by minimizing the L2-Norm distance between s and x_o as:

$$C_u = \|s - x_o\|^2 = \|\mathcal{U}(d) + x_e - x_o\|^2. \tag{6}$$

Another loss C_p is used to minimizing the L2-Norm of difference signal d as:

$$C_p = \|d\|^2 = \|x_o - \mathcal{P}(x_e)\|^2. \tag{7}$$

Thus, for a certain task, the final loss functions can be written as:

$$\mathcal{L}_{total} = \mathcal{L}_{task} + \alpha_u C_u + \alpha_p C_p \tag{8}$$

where $\mathcal{L}_{task}$ is the loss for a certain temporal task, e.g., CSLR in this paper, and α_u and α_p are coefficients for C_u and C_p, respectively.

Component weighting. The approximation s and difference signal d represent low-pass coefficients and high-pass coefficients for original signal x, respectively, which can be viewed as dominating movement patterns and adhering action details for input sequences. As not all frequency components in s or d play an important role in depicting human dynamics, we design a component weighting module f_{weight} to dynamically emphasize or suppress certain components in s or d for robust temporal representations.

Specially, for each timestamp t, f_{weight} aims to generate a specific coefficient for each channel, resulting in a total weight matrix $W \in \mathcal{R}^{C\times T}$. We instantiate f_{weight} with a tiny fully convolutional network followed by a Sigmoid function at the end, which is optimized in an end-to-end manner to dynamically decide the weights. Each value ranging from (0,1) in W represents the importance of a certain component generated by the lifting process. Instead of directly multiplying W with inputs for weighting, which we found badly hurts original representations, we perform component weighting in a residual way as:

$$X^{out} = (W - \frac{1}{2}\mathbb{1}) \times X_{in} + X_{in} \tag{9}$$

where $\mathbb{1} \in \mathcal{R}^{C\times T}$ is a full-one matrix. We first change values in W from (0,1) into $(-\frac{1}{2}, \frac{1}{2})$ and multiple W with X_{in} for obtaining biased components. Adding the biased components with X_{in} can effectively strengthen or weaken X_{in}, without hurting its original expressions.

Fusion. Given the low-pass and high-pass coefficients s^* and d^* after component weighting, we devise three simple strategies to fuse them into a single and robust representation as the temporal downsized output.

Sum. s^* and d^* are simply summed to combine their components as:

$$y = s^* + d^* \tag{10}$$

Concatenation. s^* and d^* are concatenated along channel dimension as:

$$y = \text{Concat}(s^*, d^*), \tag{11}$$

resulting in $y \in \mathcal{R}^{2C \times T}$ with double capacity.

Convolutional bottleneck. We employ a tiny convolutional network consisted of sequences of convolution with kernel size 1, BatchNorm and ReLU to combine s^* and d^* as:

$$y = \text{ReLU}(\text{BN}(\text{Conv}(\text{Concat}(s^*, d^*)))). \tag{12}$$

Discussion. Max pooling and average pooling are two widely used pool methods in various tasks. However, they follow predefined mechanisms to select values, which may not be optimal. For example, max pooling typically puts all attention on the element with the largest activation. However, the assumption that the maximum activation stands for the most discriminative element, may not always be true. Besides, the max operator hinders gradient-based optimization where only the largest activation in a sampling region is assigned back-propagated gradients, which may slow down convergence. Although average pooling ensures all elements can contribute to outputs, it treats them equally which usually results in smoothed outputs and hurts small but discriminative responses. In this paper, we refer to Lifting Scheme from signal processing to decompose signals into different sub-bands with major movements or discriminative details, which naturally fit the problem. Our method dynamically generates the downsized output for each sample, which jumps out of the hand-crafted scope like max or average pooling. Besides, our method can smoothly behave like max or average pooling, which falls between them but keeps more representative features than both due to hierarchical signal decomposition.

4 Experiments

4.1 Datasets

We evaluate our method on two commonly used large-scale datasets: RWTH-PHOENIX-Weather-2014 (PHOENIX14) and RWTH-PHOENIX-2014-Weather-T (PHOENIX14-T).

PHOENIX14 [21] is recorded from the German TV weather by nine signers wearing dark clothes in front of a clean background. It contains 6841 sentences with a vocabulary of 1295 signs, divided into 5672 training instances, 540 development (Dev) instances and 629 testing (Test) instances. All videos are shot by 25 fps with resolution 210×260.

PHOENIX14-T [2] is available for both CSLR and Sign Language Translation (SLT) tasks which can be viewed as an extension of PHOENIX14. It contains 8247 sentences with a vocabulary of 1085 signs, split into 7096 training samples, 519 development (Dev) samples and 642 testing (Test) samples.

4.2 Training Details

ResNet18 [15] is adopted as the 2D CNN backbone for fair comparison with recent methods. The 1D CNN is consisted of a sequence of {K5, P2, K5, P2} layers where Kσ and Pσ denotes a 1D convolutional layer and a pooling layer with kernel size of σ, respectively. A two-layer BiLSTM with hidden size 1024 is adopted for long-term temporal modeling, followed by a fully connected layer for sentence prediction. We train our models for 80 epochs with initial learning rate 0.001 which is divided by 5 at epoch 40 and 60. Adam [19] optimizer is adopted as default with weight decay 0.001 and batch size 2. All input frames are first resized to 256×256, and then randomly cropped to 224×224 with 50% horizontal flipping and 20% temporal rescaling during training. During inference, a 224×224 center crop is simply adopted. Following VAC [25], we employ the visual enhancement loss and visual alignment loss for additional visual supervision, with weights of 1.0 and 25.0, respectively. We only substitute two pooling layers in the 1D CNN with our TLP, as shown in Fig. 2. The coefficients α_u and α_p for loss C_u and C_p are set as 0.001.

Word Error Rate (WER) is used as the metric of measuring similarity between predicted sentence and reference sentence. It's defined as the minimal number of substitution, insertion and deletion operations to convert the predicted sentence to the reference sentence as:

$$\text{WER} = \frac{\#\text{substitutions} + \#\text{insertions} + \#\text{deletions}}{\#\text{reference}}. \tag{13}$$

Note that the lower WER, the better.

4.3 Ablation Study

Configurations for $\boldsymbol{\mathcal{P}(\cdot)}$ & $\boldsymbol{\mathcal{U}(\cdot)}$. Tab. 1a ablates the performance when varying the kernel size K for $\mathcal{P}(\cdot)$ & $\mathcal{U}(\cdot)$. Notably, a larger kernel with more local aggregation ability consistently brings better performance. When K reaches 7, it brings no more performance gain. We thus set K as 5 by default. We then test other instantiations for $\mathcal{P}(\cdot)$ & $\mathcal{U}(\cdot)$, e.g., a simple combination of Conv and Tanh, and found it obtains lower performance than current design.

Configurations for component weighting. In the upper part of Tab. 1b, we first test different implementations for f_{weight} to dynamically strengthen or

Configurations for $\mathcal{P}(\cdot)$ & $\mathcal{U}(\cdot)$	Dev(%)	Test(%)
K=3	20.0	21.1
K=4	19.9	21.1
K=5	**19.7**	**20.8**
K=7	19.9	21.0
$Tanh \circ Conv(k=5)$	20.2	21.3

(a) Effects of different configurations for $\mathcal{P}(\cdot)$ & $\mathcal{U}(\cdot)$

Component weighting	Dev(%)	Test(%)
$Sigmoid \circ IN \circ Conv(k=1) \circ Conv(k=5, g=C_{in})$	20.1	21.2
$Sigmoid \circ IN \circ Conv(k=\mathbf{3})$	20.0	21.2
$Sigmoid \circ IN \circ Conv(k=\mathbf{5})$	**19.7**	**20.8**
$Sigmoid \circ IN \circ Conv(k=\mathbf{7})$	19.9	20.9
$Sigmoid \circ \mathbf{BN} \circ Conv(k=5)$	21.1	22.3
$X^{out} = W \times X_{in}$	20.9	21.9
$X^{out} = (W - \frac{1}{2}\mathbb{1}) \times X_{in} + X_{in}$	**19.7**	**20.8**
-	20.2	21.4
Shared for s & d	19.9	20.9
Independent for s & d	**19.7**	**20.8**

(b) Effects of different configurations for component weighting.

Fusion	Dev(%)	Test(%)
Only s^*	20.2	21.2
Sum	**19.7**	**20.8**
Concatenation	19.9	21.1
Convolutional bottleneck	20.1	21.4

(c) Effects of different fusion methods.

Locations for TLP	Dev(%)	Test(%)
-	21.2	22.3
First location	20.1	21.3
Second location	20.3	21.1
Two locations	**19.7**	**20.8**

(d) Effects of locations for TLP

Table 1: Ablation study for different modules of TLP on PHOENIX14 dataset.

weaken various components. Compared to the two-Conv counterpart in the top, we observe a simpler design, i.e. $Sigmoid \circ IN \circ Conv$, achieves better performance which we employ as default. When varying the kernel size for f_{weight}, k=5 performs best among all candidates. We further compare the effect of normalization methods in f_{weight} which are typically employed to accelerate convergence and promote performance. InstanceNorm(IN) [36] and commonly used BatchNorm(BN) [18] are compared. We find that IN achieves more stable and superior performance than BN, which results from cross-batch normalization hurting the feature distribution for each sample. We then compare the choice of directly weighting with our residual architecture for component weighting. Seen from the middle in Tab. 1b, the residual architecture outperforms directly weighting by a large margin, where the latter inevitably hurts the original representation and leads to unstable expressions. We finally test the effects of component weighting under different configurations. Compared to w/o component weighting, deploying component weighting in an either shared or independent way for s & d achieves better performance. Furthermore, independent weighting for s & d brings more performance boost by considering the specific characteristics of two pathways.

Fusion methods. Tab. 1c ablates different configurations for fusion of s^* and d^*. s^* and d^* generated by lifting process correspond to different hierarchical features, while our TLP allows to flexibly choose which sub-band to be kept as downsized outputs. We note that only preserving s^* obtains lower performance than other variants, which demonstrates the effectiveness of combining low-pass coefficients s^* and high-pass coefficients d^* for effective recognition. Tab. 1c shows that simply summing s^* and d^* gives the best performance among all candidates, which we employ as default in the following experiments.

Locations for TLP. We incrementally add one or more TLPs in different locations to verify its effectiveness in Tab. 1d. Compared to our baseline w/o TLP, adding one TLP in either the first or second location brings considerable 1.1% and 0.9% performance boost, respectively. Replacing total two pooling instances with TLP gives notable 1.5% promotion without any other architecture change, demonstrating the key role of temporal pooling and the effect of TLP for robust discriminative representations.

Methods	GFLOPs	Throughout(Vids/s)	Memory(M)	Dev(%)	Test(%)
Max pooling	3.671	12.22	8827	21.2	22.3
Average pooling	3.671	12.40	8827	21.1	22.1
TLP(Ours)	3.686	12.12	8846	19.7 (+**1.5**)	20.8 (+**1.5**)

Table 2: Computational efficiency of TLP against commonly used max pooling and average pooling on PHOENIX14 dataset.

4.4 Computational Efficiency

Our TLP is an efficient plug-and-play tool with little computational overhead. Under the formula of Eq. 4, Eq. 5 and Eq. 9, two TLPs totally consumes 7.5M$\times$2 = 15.0M FLOPs[2], which is negligible (0.4%) compared to our 2D backbone ResNet18 (3.64G FLOPs). Considering our TLP is composed of highly specialized operators like convolution and activation functions, it enjoys high computational efficiency on GPUs with little delay. As shown in Tab. 2, compared to commonly used max pooling and average pooling, our TLP exhibits similar GFLOPs, throughout and memory usage with significantly higher performance, which is a both effective and efficient plug-and-play tool for temporal tasks.

[2] FLOPs denote floating point operations, which measure the computational costs of models.

Pooling methods	PHOENIX14		PHOENIX14-T	
	Dev(%)	Test(%)	Dev(%)	Test(%)
Max pooling (**Baseline**)	21.2	22.3	21.1	22.8
Stochastic pooling	22.2 (-1.0)	23.4 (-1.1)	22.3 (-1.2)	23.7 (-0.9)
Mixed pooling	21.5 (-0.3)	22.6 (-0.3)	21.3 (-0.2)	23.0 (-0.2)
L_p pooling (p=3)	21.5 (-0.3)	22.5 (-0.2)	21.3 (-0.2)	23.1 (-0.3)
SoftPool	21.3 (-0.1)	22.5 (-0.2)	21.1 (+0.0)	22.9 (-0.1)
L_p pooling (p=2)	21.1 (+0.1)	22.3 (+0.0)	21.2 (-0.1)	22.7 (+0.1)
Average pooling	21.1 (+0.1)	22.1 (+0.2)	20.9 (+0.2)	22.6 (+0.2)
TLP(Ours)	19.7 (**+1.5**)	20.8 (**+1.5**)	19.4 (**+1.7**)	21.2 (**+1.6**)

Table 3: Comparison of our TSP with other pooling variants on the Dev and Test set of PHOENIX14 and PHOENIX14-T dataset.

4.5 Comparison with other pooling methods

We compare our TSP with other pooling variants to demonstrate its effectiveness in Tab. 3. Most of these counterpart pooling methods are elaborately designed for preserving critical spatial features. As shown in Tab. 3, except L_p pooling (p=2) and average pooling, most pooling methods cause performance decline on both PHOENIX14 dataset and PHOENIX14-T dataset. L_p pooling (p=2) and average pooling bring marginal performance boost ($\leq$ 0.2%). Although most of these variants are elaborately designed for spatial tasks with superior performance, they don't exhibit much superiority on temporal tasks, e.g., CSLR. In contrast, some of them even lead to lower performance. Hand-crafted pooling methods, like max pooling and average pooling, show robust performance on both datasets, demonstrating their excellent generalization ability. Compared to these pooling variants, our TLP consistently exhibits superior performance ($\geq$ 1.5%) upon both datasets and largely surpasses all of them by a large margin. These results verify the effectiveness of our TLP by combining different sub-bands for discriminative hierarchical representations.

Methods	PHOENIX14		PHOENIX14-T	
	Dev(%)	Test(%)	Dev(%)	Test(%)
ResNet18 [15]	21.2	22.3	21.1	22.8
ResNet18 w/ TLP	19.7 (**+1.5**)	20.8 (**+1.5**)	19.4 (**+1.7**)	21.2 (**+1.6**)
SqueezeNet [17]	23.2	23.5	21.7	23.1
SqueezeNet w/ TLP	22.2 (**+1.0**)	22.3 (**+1.2**)	20.6 (**+1.1**)	22.0 (**+1.1**)
RegNetX-800Mf [30]	21.4	22.5	21.0	22.3
RegNetX-800Mf w/ TLP	20.0 (**+1.4**)	21.1 (**+1.4**)	19.5 (**+1.5**)	20.9 (**+1.4**)
RegNetY-800Mf [30]	21.3	22.2	20.7	21.8
RegNetY-800Mf w/ TLP	19.7 (**+1.6**)	20.5 (**+1.7**)	19.1 (**+1.6**)	20.1 (**+1.7**)

Table 4: Generalizability of TLP upon various backbones on both PHOENIX14 dataset and PHOENIX14-T dataset.

Methods	Backbone	PHOENIX14				PHOENIX14-T	
		Dev(%)		Test(%)		Dev(%)	Test(%)
		del/ins	WER	del/ins	WER		
SubUNet [4]	CaffeNet	14.6/4.0	40.8	4.3/4.0	40.7	-	-
Staged-Opt [5]	VGG-S	13.7/7.3	39.4	12.2/7.5	38.7	-	-
Align-iOpt [29]	3D-ResNet	12.6/2	37.1	13.0/2.5	36.7	-	-
Re-Sign [23]	GoogLeNet	-	27.1	-	26.8	-	-
SFL [26]	ResNet18	7.9/6.5	26.2	7.5/6.3	26.8	25.1	26.1
STMC [43]	VGG11	-	25.0	-	-	-	-
DNF [6]	GoogLeNet	7.8/3.5	23.8	7.8/3.4	24.4	-	-
FCN [3]	Custom	-	23.7	-	23.9	23.3	25.1
CMA [28]	GoogLeNet	7.3/2.7	**21.3**	7.3/2.4	**21.9**	-	-
VAC [25]	ResNet18	7.9/2.5	**21.2**	8.4/2.6	**22.3**	-	-
SLT* [2]	GoogLeNet	-	-	-	-	24.5	24.6
C+L+H* [20]	GoogLeNet	-	26.0	-	26.0	22.1	24.1
DNF* [6]	GoogLeNet	7.3/3.3	23.1	6.7/3.3	22.9	-	-
STMC* [43]	VGG11	7.7/3.4	**21.1**	7.4/2.6	**20.7**	**19.6**	**21.0**
Baseline	ResNet18	7.9/2.5	21.2	8.4/2.6	22.3	21.1	22.8
TLP(Ours)	ResNet18	6.3/2.8	**19.7**	6.1/2.9	**20.8**	**19.4**	**21.2**

Table 5: Comparison with other state-of-the-art methods on the PHOENIX14 and PHOENIX14-T datasets. $*$ indicate extra clues such as face or hand features are included. 'C+L+H' denotes the abbreviation of 'CNN+HMM+LSTM [20]'

4.6 Generalizability

We apply TLP to several backbones including ResNet18 [15], SqueezeNet [17], RegNetX-800Mf [30] and RegNetY-800Mf [30] on both datasets to demonstrate its generalizability. As shown in Tab. 4, TLP consistently brings significant performance boost ($\geq$ 1.0%) across different backbones. We observe an interesting phenomenon where the effect of TLP seems to be proportional to the strength of backbones. For example, the boost by TLP is relatively smaller (1.0%) for less powerful SqueezeNet [17] (23.2%). In contrast, TLP brings much more performance boost ($\geq$ 1.7%) for more powerful backbones, e.g., RegNetY-800Mf [30] (21.3%). Other backbones like MobileNet-V2 [16], EfficientNet-B0 [35] and MNASNet [34] are found out of memory upon current devices.

4.7 Comparison with the state-of-the-art

We compare our model against other state-of-the-art methods on the PHOENIX14 and PHOENIX14-T dataset in Tab. 5. The entries notated with $*$ indicate these methods such as SLT [2], CNN+LSTM+HMM [20], DNF [6] and STMC [43] utilize additional factors like face or hand features for better performance. We observe that our method outperforms all previous approaches with video information only. We also notice that our method even surpasses those approaches equipped with additional factors when only using video information, which demonstrates the effectiveness of our TLP.

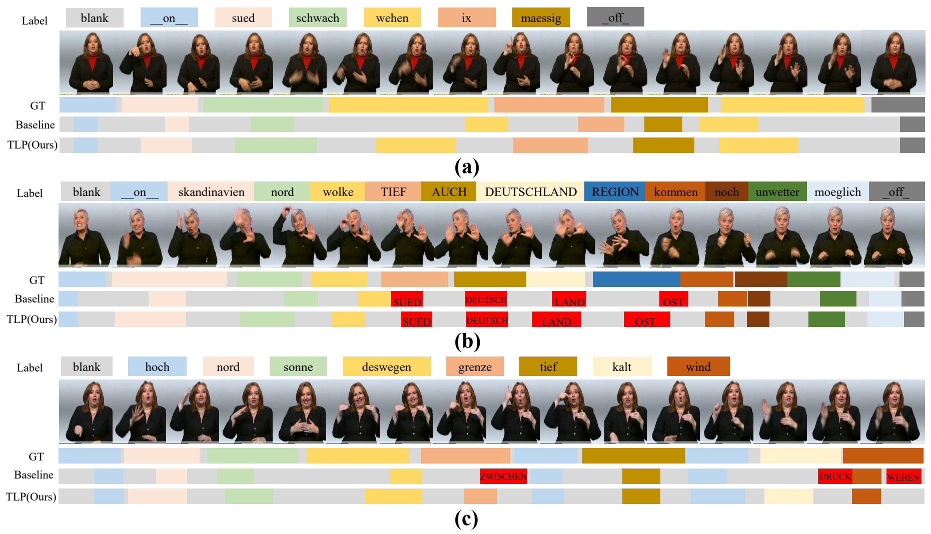

Fig. 4: Predictions of our baseline and TLP for several example videos. (a) All glosses are correctly recognized by both our baseline and TLP. (b) The same four glosses are wrongly recognized by both our baseline and TLP. (c) Our baseline wrongly recognizes several glosses while TLP makes correct predictions. Wrong recognized glosses (except del) are marked in red.

5 Visualizations

To better understand the effects of TLP, we visualize several videos with their predictions of our baseline and TLP from the Dev set of PHOENIX14 dataset in Fig. 4. Wrong recognized glosses (except del) are marked in red. We show three different cases to help demonstrate the effect of TLP in various conditions. All glosses in Fig. 4(a) are correctly recognized by both our baseline and TLP. The same four glosses in Fig. 4(b) are wrongly recognized by both our baseline and TLP. Our baseline wrongly recognizes several glosses in Fig. 4(c) while TLP makes correct predictions. We notice in all cases, TLP predicts more centralized gloss borders than our baseline which helps accurate recognition.

6 Conclusion

In this paper, we derive temporal lift pooling (TLP) from the Lifting Scheme in signal processing to decompose input signals into various sub-bands, each corresponding to a specific movement pattern. Combining different components of TLP results in a refined downsized feature map, well preserving discriminative representations. TLP exhibits excellent generalizability upon multiple backbones upon two large-scale CSLR datasets with significant performance boost. Visualizations verify the effects of TLP for correcting gloss borders.

Acknowledgment. This work is supported by NSFC 62072334 Project.

References

1. Boureau, Y.L., Ponce, J., LeCun, Y.: A theoretical analysis of feature pooling in visual recognition. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 111–118 (2010)
2. Camgoz, N.C., Hadfield, S., Koller, O., Ney, H., Bowden, R.: Neural sign language translation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7784–7793 (2018)
3. Cheng, K.L., Yang, Z., Chen, Q., Tai, Y.W.: Fully convolutional networks for continuous sign language recognition. In: ECCV (2020)
4. Cihan Camgoz, N., Hadfield, S., Koller, O., Bowden, R.: Subunets: End-to-end hand shape and continuous sign language recognition. In: ICCV (2017)
5. Cui, R., Liu, H., Zhang, C.: Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. In: CVPR (2017)
6. Cui, R., Liu, H., Zhang, C.: A deep neural framework for continuous sign language recognition by iterative training. TMM **21**(7), 1880–1891 (2019)
7. Dogiwal, S.R., Shishodia, Y., Upadhyaya, A.: Efficient lifting scheme based super resolution image reconstruction using low resolution images. In: Advanced Computing, Networking and Informatics-Volume 1, pp. 259–266. Springer (2014)
8. Freeman, W.T., Roth, M.: Orientation histograms for hand gesture recognition. In: International workshop on automatic face and gesture recognition. vol. 12, pp. 296–301. IEEE Computer Society, Washington, DC (1995)
9. Fukushima, K., Miyake, S.: Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and cooperation in neural nets, pp. 267–285. Springer (1982)
10. Gao, W., Fang, G., Zhao, D., Chen, Y.: A chinese sign language recognition system based on sofm/srn/hmm. Pattern Recognition **37**(12), 2389–2402 (2004)
11. Gao, Z., Wang, L., Wu, G.: Lip: Local importance-based pooling. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3355–3364 (2019)
12. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd international conference on Machine learning. pp. 369–376 (2006)
13. Gulcehre, C., Cho, K., Pascanu, R., Bengio, Y.: Learned-norm pooling for deep feedforward and recurrent neural networks. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 530–546. Springer (2014)
14. Han, J., Awad, G., Sutherland, A.: Modelling and segmenting subunits for sign language recognition based on hand motion analysis. Pattern Recognition Letters **30**(6), 623–633 (2009)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
16. Howard, A., Zhmoginov, A., Chen, L.C., Sandler, M., Zhu, M.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation (2018)
17. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)

18. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. Koller, O., Camgoz, N.C., Ney, H., Bowden, R.: Weakly supervised learning with multi-stream cnn-lstm-hmms to discover sequential parallelism in sign language videos. PAMI **42**(9), 2306–2320 (2019)
21. Koller, O., Forster, J., Ney, H.: Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers. Computer Vision and Image Understanding **141**, 108–125 (2015)
22. Koller, O., Zargaran, O., Ney, H., Bowden, R.: Deep sign: Hybrid cnn-hmm for continuous sign language recognition. In: Proceedings of the British Machine Vision Conference 2016 (2016)
23. Koller, O., Zargaran, S., Ney, H.: Re-sign: Re-aligned end-to-end sequence modelling with deep recurrent cnn-hmms. In: CVPR (2017)
24. LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L.: Handwritten digit recognition with a back-propagation network. Advances in neural information processing systems **2** (1989)
25. Min, Y., Hao, A., Chai, X., Chen, X.: Visual alignment constraint for continuous sign language recognition. In: ICCV (2021)
26. Niu, Z., Mak, B.: Stochastic fine-grained labeling of multi-state sign glosses for continuous sign language recognition. In: ECCV (2020)
27. Pesquet-Popescu, B., Bottreau, V.: Three-dimensional lifting schemes for motion compensated video compression. In: 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221). vol. 3, pp. 1793–1796. IEEE (2001)
28. Pu, J., Zhou, W., Hu, H., Li, H.: Boosting continuous sign language recognition via cross modality augmentation. In: ACM MM (2020)
29. Pu, J., Zhou, W., Li, H.: Iterative alignment network for continuous sign language recognition. In: CVPR (2019)
30. Radosavovic, I., Kosaraju, R.P., Girshick, R., He, K., Dollár, P.: Designing network design spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10428–10436 (2020)
31. Saeedan, F., Weber, N., Goesele, M., Roth, S.: Detail-preserving pooling in deep networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9108–9116 (2018)
32. Stergiou, A., Poppe, R., Kalliatakis, G.: Refining activation downsampling with softpool. arXiv preprint arXiv:2101.00440 (2021)
33. Sweldens, W.: The lifting scheme: A construction of second generation wavelets. SIAM journal on mathematical analysis **29**(2), 511–546 (1998)
34. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
35. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning. pp. 6105–6114. PMLR (2019)
36. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022 (2016)

37. Wu, Y., Pan, Q., Zhang, H., Zhang, S.: Adaptive denoising based on lifting scheme. In: Proceedings 7th International Conference on Signal Processing, 2004. Proceedings. ICSP'04. 2004. vol. 1, pp. 352–355. IEEE (2004)
38. Yu, D., Wang, H., Chen, P., Wei, Z.: Mixed pooling for convolutional neural networks. In: International conference on rough sets and knowledge technology. pp. 364–375. Springer (2014)
39. Zeiler, M.D., Fergus, R.: Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557 (2013)
40. Zhai, S., Wu, H., Kumar, A., Cheng, Y., Lu, Y., Zhang, Z., Feris, R.: S3pool: Pooling with stochastic spatial sampling. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4970–4978 (2017)
41. Zhao, J., Snoek, C.G.: Liftpool: Bidirectional convnet pooling. arXiv preprint arXiv:2104.00996 (2021)
42. Zheng, Y., Wang, R., Li, J.: Nonlinear wavelets and bp neural networks adaptive lifting scheme. In: The 2010 International Conference on Apperceiving Computing and Intelligence Analysis Proceeding. pp. 316–319. IEEE (2010)
43. Zhou, H., Zhou, W., Zhou, Y., Li, H.: Spatial-temporal multi-cue network for continuous sign language recognition. In: AAAI (2020)