# Adaptive Detrending to Accelerate Convolutional Gated Recurrent Unit Training for Contextual Video Recognition

Minju Jung[a,c], Haanvid Lee[b], Jun Tani[c,*]

[a]*School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea*
[b]*School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, Korea*
[c]*Cognitive Neurorobotics Research Unit, Okinawa Institute of Science and Technology Graduate University, Okinawa, Japan*

**Abstract**

Video image recognition has been extensively studied with rapid progress recently. However, most methods focus on short-term rather than long-term (contextual) video recognition. Convolutional recurrent neural networks (ConvRNNs) provide robust spatio-temporal information processing capabilities for contextual video recognition, but require extensive computation that slows down training. Inspired by normalization and detrending methods, in this paper we propose "adaptive detrending" (AD) for temporal normalization in order to accelerate the training of ConvRNNs, especially of convolutional gated recurrent unit (ConvGRU). For each neuron in a recurrent neural network (RNN), AD identifies the trending change within a sequence and subtracts it, removing the internal covariate shift. In experiments testing for contextual video recognition with ConvGRU, results show that (1) ConvGRU clearly outperforms feed-forward neural networks, (2) AD consistently and significantly accelerates training and improves generalization, (3) performance is further improved when AD is coupled with other normalization methods, and most importantly, (4) the more long-term contextual information is required, the more AD outperforms existing methods.

*Corresponding author
*Email address:* `tani1216jp@gmail.com` (Jun Tani)

*Keywords:* Detrending, normalization, internal covariate shift, convolutional neural networks (CNNs), recurrent neural networks (RNNs), convolutional recurrent neural networks (ConvRNNs)

## 1. Introduction

Convolutional neural networks (CNNs) [1] show remarkable performance on the ImageNet challenge dataset, consisting of 1000 classes and 1.2 million training images [2]. Encouraged by this success, several approaches exploit the spatial processing capability of CNNs in video recognition tasks [3, 4]. Two-stream CNNs [3] and convolutional 3D (C3D) networks [4] are the most commonly used networks. Two-stream CNNs combine classification abilities of spatial- and temporal-stream networks, being composed of a spatial-stream network that processes individual RGB frames and a temporal-stream network that processes stacked optical flow over several frames. C3D networks extend 2D convolution to 3D convolution by adding time as a third dimension, processing stacked consecutive RGB frames. However, both networks employ a stacking strategy that utilizes only a limited number of temporal correlations between stacked frames in order to recognize videos. Once the temporal window advances to the next position, information from the previous stack is completely dropped. This creates a problem of contextual recognition that requires the extraction of long-range temporal correlations [5].

In this paper, we attempt to overcome this limitation using recently introduced convolutional recurrent neural networks (ConvRNNs) that replace the weight multiplication of RNNs with convolution in order to exploit spatial and temporal information processing capabilities of CNNs and recurrent neural networks (RNNs), respectively [6, 7, 8]. By extracting spatio-temporal features hierarchically, ConvRNNs handle complex problems in the space-time domain, such as precipitation nowcasting [6], video recognition [7], and video prediction [8]. Also, problems restricted to the spatial domain can be handled by ConvRNNs in an iterative manner [9]. For example, in instance segmentation, ConvRNNs sequentially segment one instance of an image at a time [9]. However, training ConvRNNs is painfully slower than training feed-forward CNNs, which receive a single frame or stacked multiple frames for video recognition, because recurrent connections require additional computation. Moreover, it is hard to parallelize computation of ConvRNNs due to the sequential nature of RNNs, which require computations from previous time steps in advance for computing the current time step. Thus, finding a way to achieve faster learning convergence has been a barrier to practical development of ConvRNNs.

Loffe and Szegedy [10] argue that internal covariate shift is responsible for the increased training time in feed-forward neural networks, including multi-layer perceptrons (MLPs) and CNNs, and they suggest batch normalization (BN) to normalize the input distribution of a neuron for each mini-batch, as a way to reduce training time. BN successfully removes internal covariate shift, thereby significantly accelerating training with improved generalization, and this technique has become standard for training feed-forward neural networks. Some studies use BN with RNNs because unrolled RNNs over time can be seen as deep neural networks in terms of time as well as depth [11, 12]. However, BN is incompatible with RNNs, regardless of computing global statistics along the time domain [11] or local statistics at each time step [12]. Use of global statistics ignores statistics at each time step, but uses of local statistics does not accommodate training sequences of variable lengths. As an alternative, layer normalization (LN) [13] eliminates dependencies between mini-batch samples that obviate the use of BN with RNNs. LN computes statistics over all neurons in each layer and accelerates training of RNNs and MLPs, but not CNNs. Neither BN nor LN is generally applied to ConvRNNs.

The current paper focuses on the time domain in order to accelerate training of ConvRNNs. Much of time series analysis and many forecasting methods can be applied only to stationary time series. Detrending transforms non-stationary time series to stationary series by identifying the change as a trend and removing it. This method is straightforward, and is illustrated in the context of the

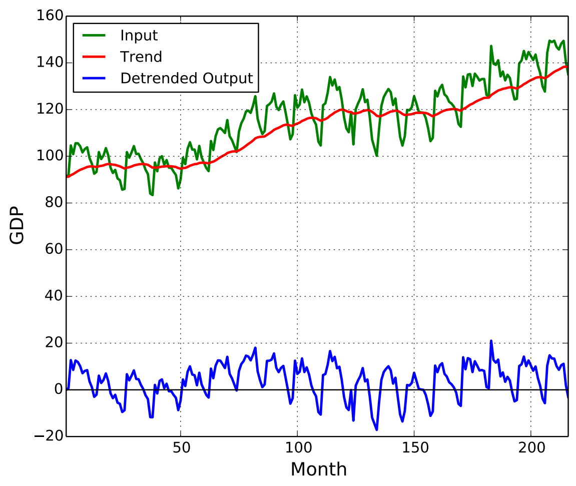


Figure 1: Example of conventional detrending with Brazilian GDP. The detrended output is obtained by subtracting the trend from the original input. In this example, we use an exponential moving average (EMA) with a fixed decay factor of 0.95 to define the trend.

Brazilian gross domestic product[1] in Fig. 1. The current research applies this method to normalize sequences of neurons in RNNs. Our key insight here is that the hidden state of a gated recurrent unit (GRU) [14] can be considered as a trend that can be approximated by the form of an exponential moving average with an adaptively changing decay factor. Based on this insight, we propose a novel temporal normalization method, "adaptive detrending" (AD), for use with GRU and convolutional gated recurrent unit (ConvGRU), which is a variant of ConvRNNs extended from GRU. The implications of AD are fourfold:

- AD is easy to implement, reducing computational cost and consuming less memory than competing methods.
- AD eliminates temporal internal covariate shift.
- AD controls the degree of detrending (or normalization) through decay factor adaptability.

[1]http://www2.stat.duke.edu/~mw/data-sets/ts_data/brazil_econ

- AD is fully compatible with existing normalization methods.

## 2. Background

### 2.1. Batch Normalization

Internal covariate shift slows training of deep neural networks, because the distribution of layer inputs changes continuously as lower layer parameters are updated. Batch normalization (BN) [10] has recently been proposed to reduce internal covariate shift by normalizing network activation as follows:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{1}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu)^2 \tag{2}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{4}$$

where $x$ is the activations of a neuron in a mini-batch of size $m$, $\mu$ and $\sigma^2$ are the mean and variance of a mini-batch, respectively, $\hat{x}$ is normalized input, $\epsilon$ is an infinitesimal constant for numerical stability, and $y$ is an affine transformation of normalized inputs $\hat{x}$. During training, the input distribution to a layer is transformed to a fixed distribution with a zero mean and unit variance, regardless of the change in parameters of lower layers. Additionally, an affine transformation with two learnable parameters $\gamma$ and $\beta$ follows normalization in order to recover the original activation when required. BN accelerates training and improves generalization of CNNs on ImageNet classification tasks.

Due to its success in feed-forward neural networks, BN has been applied to RNNs to speed training and improve generalization [11, 12]. In [11], BN is applied only to vertical (input-to-hidden) and not to horizontal (hidden-to-hidden) connections because the repeated rescaling of horizontal connections induces vanishing and exploding gradient problems. Also, the mean and variance for BN are computed by averaging along not only the mini-batch axis but also the time axis, which is called "sequence-wise normalization." On the other hand, Cooijmans et al. [12] develop "step-wise normalization" and show that (1) applying BN to horizontal as well as vertical connections is possible by properly initializing $\gamma$ of an affine transformation and beneficial for reducing temporal internal covarite shift, and (2) using statistics for each time step separately preserves initial transient phase information. However, with this method, estimation of statistics at each time step degrades along the time axis due to variation in length of training and test sequences. During training, mini-batch configuration involves the use of zero, or last frame padding for shorter sequences. Furthermore, statistics for each time step are estimated only up to the length of the longest training sequence $T_{max}$. After training, accurate statistics for test sequences longer than the longest training sequence $T_{max}$ cannot be generated. Due to these factors, performance suffers.

### *2.2. Layer Normalization*

Ba et al. [13] introduce "layer normalization" (LN) to overcome the limitations of BN when applied to RNNs. LN has the same form as that of Cooijmans et al.'s [12] step-wise normalization, with the difference that LN normalizes over the spatial axis rather than by mini-batch. The assumption underlying LN is that changes in output from one layer correlate highly with changes in summed inputs of the next layer. Hence, LN estimates statistics for data from a single training session using all activations in each layer. By estimating statistics over layers instead of mini-batches, LN properly estimates statistics at each time step, regardless of mini-batch sequence length variability. In experiments with RNNs, LN achieves faster convergence and better generalization than baseline and other normalization methods, especially for long sequences and small mini-batches.

However, LN does not perform well with CNNs. The authors report that LN is better than the baseline without normalization, but not better than BN.

They hypothesize that neurons in a layer have different statistics due to the spatial topology of feature maps, so that the central assumption of LN cannot be supported for CNNs. We agree that normalizing all neurons in a layer with the same statistics is not the best method for normalizing CNNs. However, because BN works successfully for CNNs by estimating statistics of each feature map, LN's shortcomings with CNNs might reflect different statistics between feature maps, and not within a feature map.

## 3. Model

### *3.1. Gated Recurrent Unit*

Standard recurrent neural networks (RNNs) have greater utility than feed-forward networks because they add a recurrent connection to handle sequential data. RNNs consist of three layers: an input layer $\mathbf{x}$, a hidden layer $\mathbf{h}$, and an output layer $\mathbf{y}$. RNNs are able to handle sequential data because the hidden layer receives both current input from the input layer as well as information about its own previous state through a recurrent connection as follows:

$$\mathbf{h}_t = g(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{5}$$

$$\mathbf{y}_t = f(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{6}$$

where $g(\cdot)$ and $f(\cdot)$ are element-wise non-linear activation functions for the hidden and output layers, respectively, and $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{b}$ represent the learnable parameters of RNNs: forward connection weights, recurrent connection weights, and biases, respectively.

However, standard RNNs do not capture long-term dependencies well because of vanishing and exploding gradient problems [15, 16]. The gated recurrent unit (GRU) was proposed by Cho et al. [14] to overcome the vanishing gradient problem. It employs the same gating mechanism as long short-term memory (LSTM) [17], but employs a simpler architecture by eliminating the output gate and modifying some other parts of LSTM. Specifically, GRU has two gating units, called a reset gate $\mathbf{r}$ and an update gate $\mathbf{z}$. The hidden state $\mathbf{h}_t$ at each time step $t$ is calculated using a leaky integrator with an adaptive time constant determined by the update gate $\mathbf{z}$. In other words, the hidden state $\mathbf{h}_t$ is a linear interpolation between the previous hidden state $\mathbf{h}_{t-1}$ and the candidate hidden state $\tilde{\mathbf{h}}_t$ as weighted by the update gate $\mathbf{z}$, and is defined as follows:

$$\mathbf{h}_t = \mathbf{z}_t \odot \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} \tag{7}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{8}$$

where $\sigma(\cdot)$ is a sigmoid function and $\odot$ is an element-wise multiplication.

The candidate hidden state $\tilde{\mathbf{h}}_t$ at each time step $t$ is calculated similarly to that of the hidden layer in standard RNNs (5). However, unlike standard RNNs, the reset gate $\mathbf{r}$ determines how much the previous hidden state $\mathbf{h}_{t-1}$ affects the candidate hidden state $\tilde{\mathbf{h}}_t$ as follows:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{r}_t \odot \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{9}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{10}$$

### *3.2. Gated Recurrent Unit Normalization in the Spatial Domain*

Following Ba et al. [13], in this paper we apply recurrent batch normalization (recurrent BN) [12] and layer normalization (LN) [13] to GRU. We refer to recurrent BN and LN as "spatial" normalization methods to differentiate the present approach from normalization in the time domain reviewed above. The following equations represent GRU normalization in the spatial domain:

$$\mathbf{r}_t = \sigma(\mathrm{N}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\mathbf{W}_r \mathbf{x}_t) + \mathrm{N}_{\boldsymbol{\gamma}}(\mathbf{U}_r \mathbf{h}_{t-1})) \tag{11}$$

$$\mathbf{z}_t = \sigma(\mathrm{N}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\mathbf{W}_z \mathbf{x}_t) + \mathrm{N}_{\boldsymbol{\gamma}}(\mathbf{U}_z \mathbf{h}_{t-1})) \tag{12}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathrm{N}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\mathbf{W}_h \mathbf{x}_t) + \mathbf{r}_t \odot \mathrm{N}_{\boldsymbol{\gamma}}(\mathbf{U}_h \mathbf{h}_{t-1})) \tag{13}$$

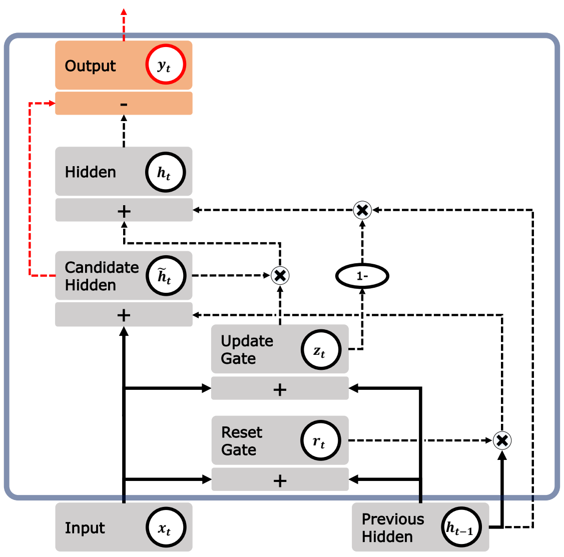


Figure 2: Schematic of gated recurrent unit (GRU) with adaptive detrending (AD). Gray modules and black lines are standard for GRU, and light red modules and red lines are newly added for AD. Solid lines represent weight multiplication operations and dashed lines represent element-wise operations.

$$\mathbf{h}_t = \mathbf{z}_t \odot \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} \tag{14}$$

where $\mathrm{N}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\cdot)$ represents the normalization followed by an affine transformation with two learnable parameters (gain $\boldsymbol{\gamma}$ and bias $\boldsymbol{\beta}$) for recurrent BN and LN, and $\mathrm{N}_{\boldsymbol{\gamma}}(\cdot)$ is the same as $\mathrm{N}_{\boldsymbol{\gamma},\boldsymbol{\beta}}(\cdot)$ except for an affine transformation with only the gain $\boldsymbol{\gamma}$ to remove the bias redundancy within an equation. For the same reason, the biases of the original GRU equations are removed. Note that the gain $\boldsymbol{\gamma}$ and bias $\boldsymbol{\beta}$ are shared over time as mentioned in [12, 13].

## *3.3. Adaptive Detrending*

By normalizing in a step-wise manner [12, 13], spatial methods accelerate training and improve performance on complex sequential tasks such as language modeling. However, spatial normalization methods have a limitation. Although statistics are estimated at each time separately in order to capture the initial transient, each estimation is based only on current neural activations. This is not optimal for RNNs, because the true statistics of RNNs at a current step inherently depend on those of previous steps. Hence, statistics estimation for RNNs must take into account how RNNs generate statistics over time. Specifically, each current estimation must influence subsequent estimation(s). This reminds us of a moving average (MA).

In statistics, a MA is widely used to extract long-term trends from noisy time series by filtering out fluctuations. There are many variants of MA, including a simple moving average (SMA) [18] and an exponential moving average (EMA) [19]. Among these variants, an EMA is preferred when the MA needs to quickly respond to recent data because past data decay exponentially over time. In addition, unlike SMA, EMA does not require redundant computation caused by window shifting and contains the full past history of a time series due to its recursive formulation. The value of the EMA $\mu_t$ at time step $t$ is calculated by

$$\mu_t = \alpha \cdot x_t + (1 - \alpha) \cdot \mu_{t-1} \tag{15}$$

where $x_t$ is the current input value and $\alpha$ is a constant decay factor or smoothing factor between 0 and 1.

Detrending is a method that removes a slowly changing component, called a "trend", in order to render time series stationary. We think that detrending can be applied to RNNs to eliminate temporal internal covariate shift. Notice that the definition of EMA in (15) is the same as that for the hidden state $\mathbf{h}$ of GRU in (7) when, rather than being fixed, the decay factor $\alpha$ is continuously changing at each time step as shown in (8). By considering the hidden state $\mathbf{h}$ as a trend of the candidate hidden state $\tilde{\mathbf{h}}$, we can apply detrending to GRU for temporal normalization, as follows:

$$\mathbf{y}_t = \tilde{\mathbf{h}}_t - \mathbf{h}_t \tag{16}$$

where $\mathbf{y}_t$ is the detrended output at time step $t$, and is input into the next layer (schematic in Fig. 2).

Before going any further, one critical point must be addressed. Some may doubt the proposed detrending method because it normalizes the candidate hidden state $\tilde{\mathbf{h}}$ not the hidden state $\mathbf{h}$, which is the output of GRU that we want to normalize. To directly normalize or detrend the hidden state $\mathbf{h}$, we need to estimate a trend $\boldsymbol{\mu}^h$ of the hidden state $\mathbf{h}$ with a decay factor $\boldsymbol{\alpha}^h$ as follows:

$$\boldsymbol{\mu}_t^h = \boldsymbol{\alpha}_t^h \cdot \mathbf{h}_t + (1 - \boldsymbol{\alpha}_t^h) \cdot \boldsymbol{\mu}_{t-1}^h \tag{17}$$

$$\boldsymbol{\alpha}_t^h = \sigma(\mathbf{W}_\alpha \mathbf{x}_t + \mathbf{U}_\alpha \mathbf{h}_{t-1} + \mathbf{b}_\alpha) \tag{18}$$

However, unlike the case in which (16) utilizes the hidden state $\mathbf{h}$ and the update gate $\mathbf{z}$ of GRU as a trend and a decay factor, respectively, (17) and (18) are not parts of the GRU computation. Therefore, this approach requires additional computation and memory, which is not desirable because the main objective is to accelerate training. Going back to the definition of detrending, the detrended output is converted to stationary input by removing the trend, because the trend contains a low-frequency component that makes the input non-stationary. In the same manner, the trend can be rendered stationary by removing the input, because the input contains all frequency components in the trend, including a low-frequency component in the trend of the trend, which is necessary for detrending the trend. Hence, the proposed detrending method in (16) can be considered as a normalization of the hidden state $\mathbf{h}$ with sign switching as follows:

$$-\mathbf{y}_t = \mathbf{h}_t - \tilde{\mathbf{h}}_t \tag{19}$$

Compared with direct detrending of the hidden state $\mathbf{h}$, which leaves a high-frequency component in the hidden state $\mathbf{h}$, a stationary hidden state $\mathbf{h}$ by (19) contains a high-frequency component in the candidate hidden state $\tilde{\mathbf{h}}$. However,
155 we think that this difference might not be significant for GRU because it can control the degree of detrending or extract a required frequency component by modulating the update gate $\mathbf{z}$. A sign switched input is considered the same as the original one in neural networks, so that hereafter we use (16) rather than (19) for the proposed detrending method.

As mentioned above, the proposed detrending method uses the update gate $\mathbf{z}$ in (8) as the decay factor $\alpha$ in (15), but it is adaptively changed over time rather than fixed. Hence, we call this method "adaptive detrending" (AD) to differentiate it from conventional detrending methods that employ a pre-defined or fixed setting to estimate a trend. AD presents several benefits. First, it requires negligible additional computation and memory because statistics estimation is already included as a part of the GRU computation. Second, AD is fully differentiable. Differentiability is necessary to normalize activations of neural networks because a statistics estimation and normalization must be included in gradient descent optimization to prevent model explosion [10]. Third, AD automatically estimates trend shape by adapting a decay factor at each time step for each sample. Hence, we do not need to worry about defining a trend fitting function (linear, polynomial, or moving average), or about setting parameters of a trend fitting function (such as the window size of a moving average). Furthermore, the trend estimated by AD works as a control for the degree of detrending, neurons, and samples. This is crucial because a fixed degree of detrending might cause loss of some informative frequency components that change over time, neurons, and samples. Finally, AD achieves both sample-wise and neuron-wise normalization using the time domain for normalization. Unlike BN, sample-wise normalization of AD removes dependencies between samples in a mini-batch, so that AD can be applied to RNNs without constraints. Unlike LN, neuron-wise normalization of AD allows it to be applied to a network regardless of whether neurons have similar statistics (e.g., MLPs) (e.g., CNNs).

Note that, unlike other spatial normalization methods, we do not use an affine transformation after AD. AD itself acts as a temporal normalizer following the affine transformation of each neuron with gain $\gamma$ and bias $\beta$ that change over time and between samples.

### *3.4. Convolutional Gated Recurrent Unit*

The convolutional gated recurrent unit (ConvGRU) is a natural extension of GRU via the convolutional property of CNNs and is defined as follows:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r * \mathbf{x}_t + \mathbf{U}_r * \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{20}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z * \mathbf{x}_t + \mathbf{U}_z * \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{21}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h * \mathbf{x}_t + \mathbf{r}_t \odot \mathbf{U}_h * \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{22}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} \tag{23}$$

where $*$ is a convolution operation. The key difference between ConvGRU and GRU is that ConvGRU preserves spatial topology because of its convolution operation, using 2D weight kernels on 2D feature maps. Furthermore, ConvGRU drastically reduces the number of parameters compared with GRU when directly applied to the spatial domain. Both spatial normalization methods and AD can be applied to ConvGRU in the same manner as GRU.

## 4. Experiments

We conducted three experiments to verify the effectiveness of the proposed method. The first two experiments focus on contextual video recognition to show that (1) ConvRNNs, especially ConvGRU, can successfully recognize a video by extracting spatio-temporal features at multiple scales, as emphasized by several authors [5, 20, 21], and (2) the proposed method significantly speeds ConvGRU training. We do not use popular action recognition datasets including UCF-101 [22] and HMDB-51 [23] because feed-forward networks have already performed well on these datasets with only short-term information processing [3, 4]. For example, it would be very easy to categorize a video showing a person playing a guitar without extracting a temporal profile of the video, but by simply categorizing an object, i.e. guitar. Rather, we use two video datasets[2][3] proposed by Lee et al. [20] for contextual recognition that require both temporal as well as spatial information. The first dataset is for object-related action (OA) recognition and the second one is for object-related action with modifier (OA-M) recognition. In these two experiments, we compare ConvGRU with (1) a spatial CNN receiving individual video frames as inputs and (2) convolutional 3D (C3D) network [4] receiving video clips, each has 16 frames, as inputs for short-term information processing. We do this to show that long-term information is crucial for contextual recognition, which cannot be demonstrated with UCF-101 and HMDB-51. At the same time, we explain how AD works more effectively as a task becomes contextually more complex by changing complexity of contextual information in a dataset from OA to OA-M recognition.

As the final experiment, we performed 3D skeleton-based action recognition task on the NTU RGB+D dataset [24], which is the largest dataset for 3D human action recognition. The goals of the final experiment are (1) to investigate whether AD is restricted only to ConvRNNs or whether it extends generally to RNNs, including GRU and (2) to examine the performance of AD when long-term contextual information becomes more crucial, as the extension of the previous two experiments. To do that, we compare the convergence speed and recognition accuracy of GRU with or without AD and spatial normalization. Details about network configuration, training, and evaluation protocol for spatial CNN, C3D network, and GRU are provided in Appendix A, Appendix B, and Section 4.5, respectively.

## *4.1. Implementation Details*

### *4.1.1. Architecture*

ConvGRU networks used for the following experiments consist of one convolutional (Conv) layer, two convolutional gated recurrent unit (ConvGRU) layers,

[2]https://github.com/haanvid/CL1AD/releases
[3]https://github.com/haanvid/CL2AD/releases

Table 1: Convolutional Gated Recurrent Unit Network configuration. The filter has four dimensions (height×width×input channels×output channels) for convolutional and fully-connected layers, two dimensions (height×width) for pooling layers, and both the stride and pad have two dimensions (height×width).

| Layer | Type | Forward | | | Recurrent | | |
|---|---|---|---|---|---|---|---|
| | | Filter | Stride | Pad | Filter | Stride | Pad |
| 1 | Conv (ReLU) | 7×7×3×32 | 3×3 | 0×0 | - | - | - |
| 2 | Max | 3×3 | 3×3 | 0×0 | - | - | - |
| 3 | ConvGRU | 3×3×32×64 | 1×1 | 1×1 | 3×3×64×64 | 1×1 | 1×1 |
| 4 | Max | 2×2 | 2×2 | 0×0 | - | - | - |
| 5 | ConvGRU | 3×3×64×128 | 1×1 | 1×1 | 3×3×128×128 | 1×1 | 1×1 |
| 6 | Global Avg | 6×6 | - | - | - | - | - |
| 7 | FC (Softmax) | 1×1×128×$C_1$ | - | - | - | - | - |
| | ⋮ | | | | | | |
| | FC (Softmax) | 1×1×128×$C_N$ | - | - | - | - | - |

two max pooling (Max) layers, one global average pooling (Global Avg) layer, and one fully-connected (FC) layer. The bottom layer is a convolutional layer for spatial dimension reduction having 32 feature maps with 7×7 kernels and 3×3 stride, and is followed by a rectified linear unit (ReLU) activation function. Then, two ConvGRU layers are stacked, the first having 64 and the second 128 feature maps, with 3×3 kernels and 1×1 padding in both the forward $\mathbf{W}$ and recurrent $\mathbf{U}$ paths. The two max pooling layers are located in between the three layers (one convolutional and two ConvGRU layers) with subsampling factors of 3×3 for the first and 2×2 for the second. A global average pooling layer [25] follows the last ConvGRU layer to vectorize all feature maps. The vectorized feature maps are fed into a fully-connected layer with a softmax activation function for each category. When there are $N$ categories in a dataset, the $n$-th category has $C_n$ classes. The network configuration is summarized in Table 1.

### *4.1.2. Training*

ConvGRU networks were trained by mini-batch stochastic gradient descent (SGD) with Nesterov's accelerated gradient (NAG) [26], with the momentum coefficient $\mu$ set to 0.9, and implemented in Torch7 [27] and PyTorch [28]. Mini-batch size was set to 8. The gradient was calculated using a back-propagation through time (BPTT) algorithm. For each category, we used the negative log likelihood loss function $\ell$ as follows:

$$\ell = -\sum_{c=1}^{C} p_c \log(\hat{p}_c) \tag{24}$$

where $C$ is the number of classes, and $p_c$ and $\hat{p}_c$ are the true and predicted probability of class $c$, respectively. As in Jung et al. [5], error was only generated at the end of a training sequence to utilize accumulated information through space and time for recognition. As in Krizhevsky et al. [2], an L2-norm weight decay of 0.0005 was applied while updating network parameters in order to prevent over-fitting. Because of the exploding gradient problem, we employed a gradient clipping method [29] which rescales the L2-norm of the gradient to a threshold whenever the L2-norm exceeds that threshold. Here, the threshold was set to 10. The initial hidden state $\mathbf{h}_0$ in ConvGRU was set to 0.

All weights were initialized using randomly selected values from a zero-mean Gaussian distribution, with the standard deviation $\sigma$ set differently depending on the experiment. Similar to the bias initialization trick used to solve the gradient vanishing problem of LSTM [30, 31], update gate biases were initialized to -2, and the remaining biases were initialized to 0 by default (unless otherwise noted). For both recurrent batch normalization (recurrent BN) [12] and layer normalization (LN) [13], the gain $\gamma$ and bias $\beta$ of each affine transformation were initialized to 1 and 0, respectively. However, when recurrent BN and LN were applied to the update gate, the bias $\beta$ of each affine transformation was initialized to -2 according to the above mentioned bias initialization trick. Note that we did not initialize the gain $\gamma$ to 0.1 as did Cooijmans et al. [12] because initializing the gain $\gamma$ to 1 produced better results in the following experiments.

#### *4.1.3. Data Pre-processing and Augmentation*

All images in both datasets were rescaled to 128×170 pixels and integer values 0 to 255 were normalized to real values -1 to 1. By rescaling the size of an image, the degree of coverage of an image by a cropped image for data-augmentation can be controlled to the desired level, and can be fixed among all images, even though each might have different resolution. To reduce over-fitting, we applied data-augmentation methods, which artificially increase the size of training samples In detail, all images in the same video were randomly sampled or cropped from a 112×112 region, and then the images were randomly flipped horizontally with a 50% probability. These augmented images were given to the network.

### *4.1.4. Testing*

From each test video, we obtained 10 inputs for the network by consistently cropping 1 center and 4 corners of all images in the video, and then horizontally flipping them. The output at the end of a test video was used for classification scoring in the same manner as for the training phase. The final classification accuracy was obtained by averaging all 10 classification scores.

## *4.2. Evaluation Protocol*

For the following two experiments, each dataset was randomly divided into three splits for cross-validation. Each split contained eight subjects for training and two subjects for testing. After measuring recognition accuracy for each of three splits, the average recognition accuracy over all three splits was calculated.

## *4.3. Object-Related Action Recognition*

The dataset for object-related action (OA) recognition contained 900 videos in 15 object-action combination classes, which are a partial combination of four objects ("Book", "Laptop", "Bottle", and "Cup") and nine actions ("Change page", "Sweep", "Open", "Close", "Type", "Shake", "Drink", "Stir", and "Blow") rather than a full combination of them. To avoid action inference by object identification, each video in the dataset included two objects. One was the target object on which a subject performs an action, and the other was a distractor unrelated to the current action. Each object-action combination class was demonstrated by 10 subjects, two times each, with three possible distractors present

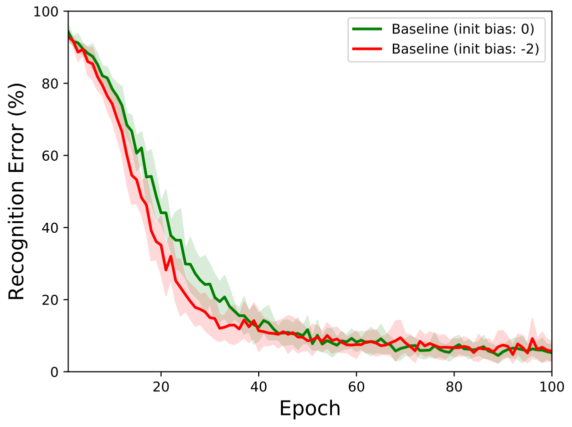
Table 2: Comparison of accuracy (mean $\pm$ standard deviation) on the OA recognition dataset.

| Model | Accuracy (%) | | |
|---|---|---|---|
| | Object | Action | Joint |
| Spatial CNN | $86.4 \pm 6.9$ | $63.5 \pm 9.9$ | $59.6 \pm 9.9$ |
| C3D | $99.0 \pm 0.7$ | $98.1 \pm 0.7$ | $97.4 \pm 1.0$ |
| Baseline (init bias: 0) | $98.2 \pm 0.6$ | $97.8 \pm 1.0$ | $97.0 \pm 1.1$ |
| Baseline (init bias: -2) | $98.3 \pm 0.5$ | $98.0 \pm 1.4$ | $97.0 \pm 1.4$ |
| AD (init bias: 0) | $99.1 \pm 0.7$ | $98.8 \pm 0.9$ | $98.4 \pm 0.8$ |
| AD (init bias: -2) | $99.1 \pm 0.8$ | $99.1 \pm 0.7$ | $98.7 \pm 0.8$ |
| BN (all) | $99.1 \pm 0.6$ | $98.6 \pm 1.1$ | $98.0 \pm 1.4$ |
| BN (hidden) | $98.8 \pm 1.0$ | $98.4 \pm 1.0$ | $97.9 \pm 1.2$ |
| BN (gates) | $98.5 \pm 1.0$ | $97.7 \pm 1.3$ | $97.0 \pm 1.5$ |
| LN (all) | $98.8 \pm 0.6$ | $98.2 \pm 0.7$ | $97.5 \pm 0.8$ |
| LN (hidden) | $98.5 \pm 0.6$ | $98.2 \pm 1.0$ | $97.4 \pm 0.9$ |
| LN (gates) | $98.0 \pm 0.8$ | $97.3 \pm 1.6$ | $96.4 \pm 1.5$ |
| BN+AD | $99.3 \pm 0.5$ | $98.9 \pm 0.8$ | $98.8 \pm 0.7$ |
| LN+AD | $99.1 \pm 0.5$ | $99.4 \pm 0.7$ | $98.7 \pm 0.6$ |

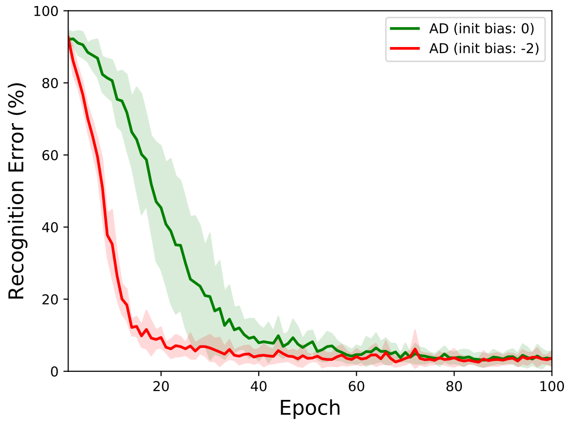
each time. The viewpoint and background were static. We pre-processed all videos to have a fixed number of frames (50 frames) because batch normalization has difficulties handling variable length sequences in a mini-batch. All networks were initialized with a standard deviation of 0.07 and trained with a learning rate of 0.01 over 100 epochs. In this experiment, we repeated the evaluation protocol three times with different initialization seeds for each split, which allowed us to gain statistically robust results from a small OA recognition dataset.

### *4.3.1. Feed-Forward Networks*

Table 2 shows the test accuracy of the networks. The accuracy gap between the spatial CNN and C3D indicates that at least short-term information needs



(a)



(b)

Figure 3: Importance of initializing the update gate bias on the object-related action (OA) recognition dataset. The graphs compare two initialization strategies for the update gate bias of (a) the baseline and (b) AD. Solid lines and shaded regions represent average test recognition errors and standard deviations over three splits with three runs, respectively.

to be processed for OA recognition. In the case of the spatial CNN without using any temporal information, OA recognition is made difficult by both distractor objects as well as action similarity.

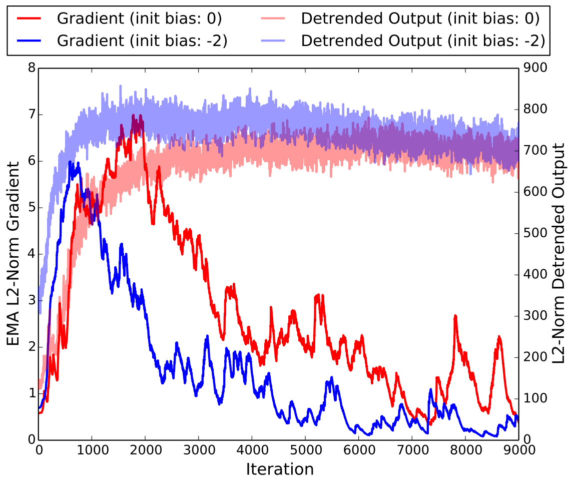


Figure 4: L2-norm of the gradient (solid lines) and detrended output (semi-transparent lines) of AD versus iteration on the OA recognition dataset (split 1). Each L2-norm of the gradient is smoothed by EMA with a decay factor of 0.99 for better visualization.

### *4.3.2. Initialization of Update Gate Bias*

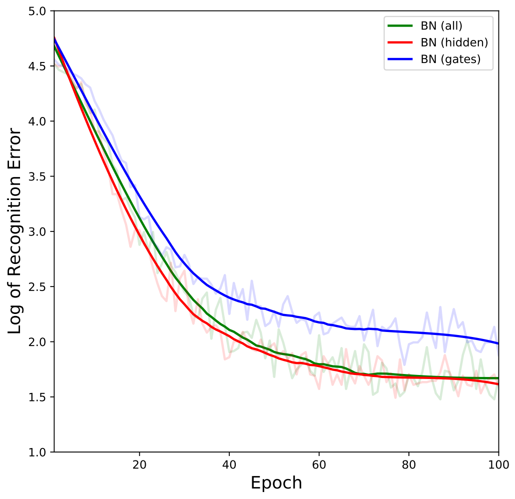
Initializing the forget gate bias to a large positive value (usually 1 or 2) is a trick widely used with LSTM to prevent the vanishing gradient problem when the weights and biases of LSTM are initialized with small random numbers [30, 31]. On the other hand, when the forget gate is set at 0.5 by initializing LSTM with small random numbers, initial information decays exponentially over time. With this bias initialization trick, performance and convergence speed of LSTM are improved, especially when long-term dependencies are crucial. In the case of GRU, initializing the update gate bias with a large negative value provides the same effect as the bias initialization trick of LSTM.

In order to examine the effect of the bias initialization trick, we compared convergence speeds using different initial update gate biases (between 0 and -2) in the baseline model (ConvGRU without normalization) and adaptive detrending (AD) (Fig. 3). Both the baseline and AD demonstrate convergence speed improvement with the update gate bias initialized to -2 rather than 0, but the improvement in AD is much more significant than that of the baseline. Furthermore, when the update gate bias of AD is initialized to 0, the variance of convergence graphs trained on the three different splits is larger than the baseline with the zero and negative initial biases, and AD with the negative initial bias. These results indicate that (1) a random initialization causes extremely slow, unstable learning with AD, and (2) initialization of the update gate bias is more crucial for AD performance than for the baseline.

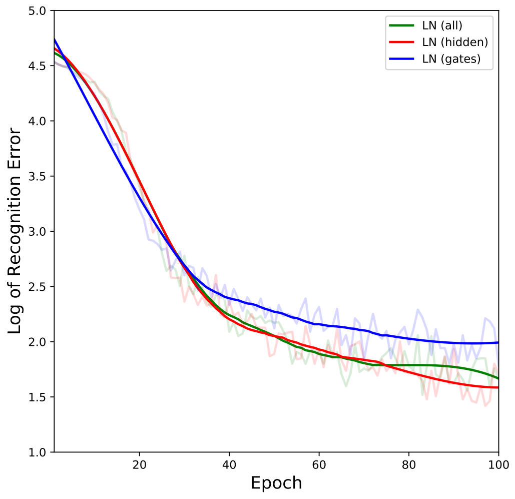
Because the hidden state (or trend) of AD closely follows input sequences, we hypothesized that the detrended output of AD is initially too small to create enough of a gradient for effective training when the update gate bias is initialized to 0. To verify our hypothesis, we analyzed the L2-norm of the detrended output and that of the gradient for both zero and negative bias initializations during AD training as shown in Fig. 4. Because the L2-norm of the gradient shows a high fluctuation, we smoothed it using an exponential moving average (EMA) with a decay factor of 0.99. As we expected, the L2-norm of the detrended output with zero bias initialization is considerably smaller than that of the negative bias initialization during the initial phase of learning. As a result, BPTT cannot generate enough gradient for training, with the net result that zero bias initialization slows network training. In other words, negative bias initialization converges to the local minima faster than zero bias one by boosting the gradient scale in the initial phase of learning. Due to faster convergence, the L2-norm of the gradient of the negative bias initialization is diminished earlier than that of the zero bias in the later phase of learning. These results suggest that the large gradient in the initial phase of learning is crucial for learning acceleration. Hence, from this point forward, the bias of the update gate was initialized to -2.

### *4.3.3. Overhead Reduction for Spatial Normalization*

Although spatial normalization methods can accelerate deep neural network training, "overhead," including computational cost and memory consumption are required to estimate statistics and to normalize increases. Given the benefits of spatial normalization methods, overhead might be tolerable for most neural



(a)



(b)

Figure 5: Effect of spatial normalization according to location on the OA recognition dataset. (a) Batch normalization (BN). (b) Layer normalization (LN). For “hidden”, the bias of the update gate is initialized to -2. Otherwise, the bias of an affine transformation for the update gate is initialized to -2. Semi-transparent lines represent the log of recognition errors averaged over three splits with three runs and solid lines represent the smoothed log of recognition errors. Each log of the recognition error is smoothed by Savitzky-Golay filter with a window length of 51 and polynomial order of 3 for better visualization.

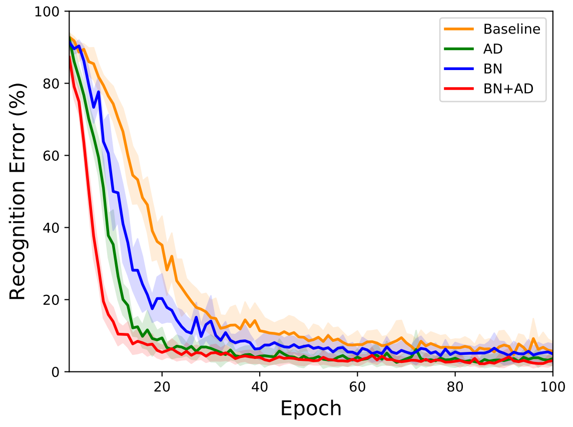
networks. However, when the same spatial normalization methods are applied to ConvRNNs, required overhead increases dramatically because normalization needs to be performed on extremely high spatial-temporal dimensions in ConvRNNs.

As detailed in Section 3.2, recurrent BN (hereafter, dropping "recurrent" abbreviated simply as BN) and LN normalize the three input distributions of the candidate hidden state, reset gate, and update gate of GRU. However, all three normalizations might not contribute equally to the improvement achieved by BN and LN. If this assumption is correct, we can eliminate less important normalizations to alleviate overhead while minimizing performance and training speed degradation. So, we investigated effects of normalization depending on where BN and LN are employed, as follows: candidate hidden state ("hidden"), reset and update gates ("gates"), and candidate hidden state along with reset and update gates ("all").

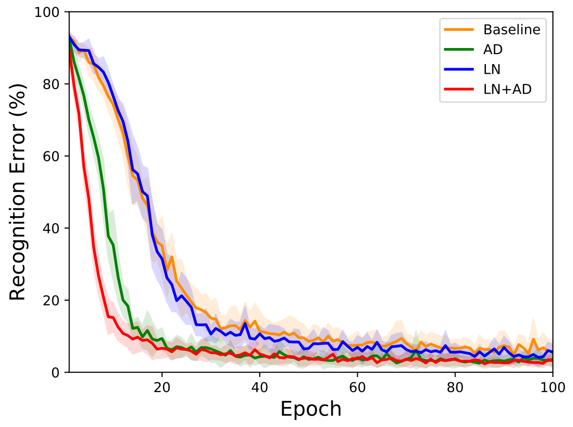
Compared with "all", "gates" marginally slows the convergence speed of BN and converges to worse local optima with both BN and LN, matching the performance degradation of "gates" (around 1.0%) (Fig. 5 and Table 2). In the case of "hidden" using both BN and LN, although it requires only one third the overhead of "all", the convergence speed and performance are similar to those of "all". These results indicate that normalization of the candidate hidden state plays the most important role in convergence speed and performance increase. Hence, from this point forward in the experiment, BN and LN were applied only to the candidate hidden state.

### *4.3.4. Adaptive Detrending versus Spatial Normalization*

Since spatial normalization methods demonstrably sped-up training and improved performance on many different tasks [12, 13], we expected the same benefits from BN and LN on the OA recognition task. As expected, both BN and LN demonstrate increased recognition accuracy over the baseline, and BN accelerates convergence speed (Fig. 6 and Table 2). However, LN does not offer any acceleration in ConvGRU. LN offers greater speed over baseline, but under-



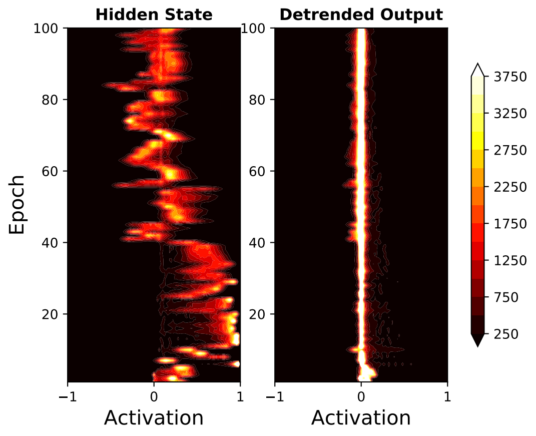
(a)



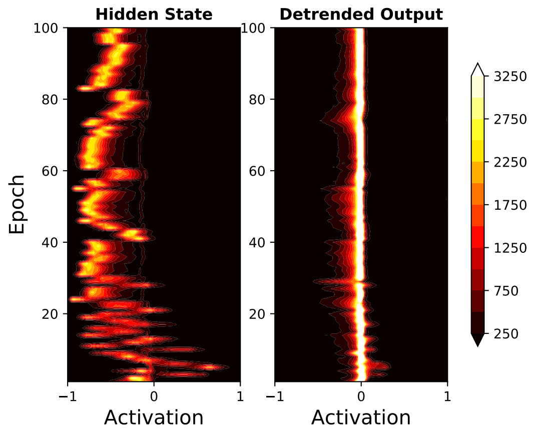
(b)

Figure 6: Graph of test recognition error averaged over three splits with three runs versus training epochs on the OA recognition dataset. (a) Comparison of the baseline, AD, BN, and BN+AD. (b) Comparison of the baseline, AD, LN, and LN+AD. Spatial normalization methods (BN and LN) are only applied to the candidate hidden state. The bias of the update gate is initialized to -2. Solid lines and shaded regions indicate the same things as those in Fig. 3.

performs BN when applied to CNNs because of the statistics estimation error caused by incorrect initial assumptions (as explained in Section 2.2). Surprisingly, the convergence speed of LN was worse in ConvGRU than in CNNs. We hypothesize that the statistics estimation error plaguing LN when implemented

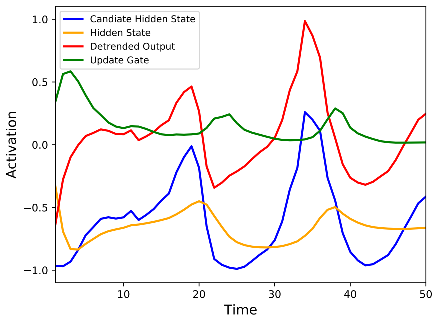


(a)

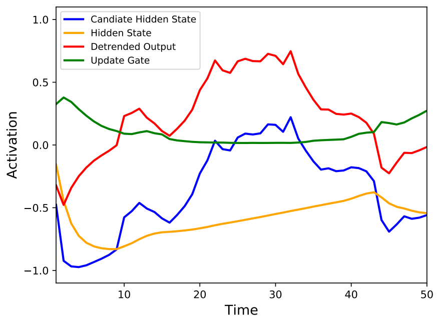


(b)

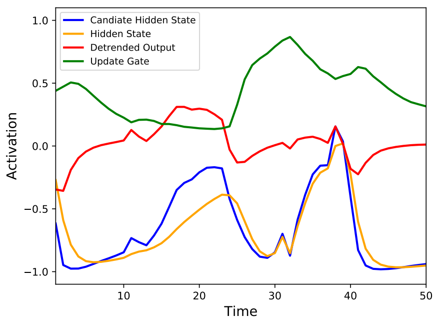
Figure 7: Visualization of change in distribution for a single neuron from the first ConvGRU layer over 100 epochs. Each distribution represents a single neuron's activations for 720 training videos, each having 50 frames, from the OA recognition dataset (split 1). Each distribution is represented by the histogram, which is of activation from -1 to 1 with bins of 0.02. The contour level indicates how many samples (total number of samples: $50 \times 720 = 36000$) are located in each bin of the histogram. The maximum contour level of (a) is set differently than those of (b) in order to aid visualization. Two selected neurons are shown in (a) and (b). Left panel: The distribution change of the hidden state. Right panel: The distribution change of the detrended output.



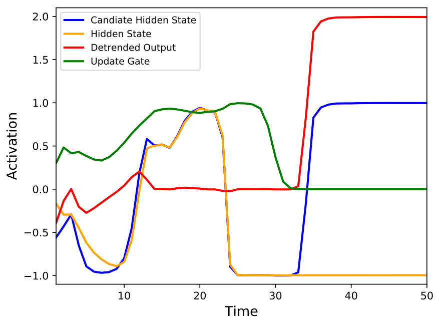
(a)



(b)



(c)



(d)

Figure 8: Visualization of AD for a neuron. Time series are obtained from a single neuron in the second ConvGRU layer while receiving a training video, after training is finished on the OA recognition dataset (split 1). Three selected neurons are shown in (a)-(d). Note that (b)-(c) are the same neuron, but receive different videos.

in CNNs is accumulated through time in ConvRNNs, leading to significant decrease in convergence speed.

AD improves convergence speed significantly as well as increasing recognition accuracy over those of the baseline, and over both BN and LN (Fig. 6 and Table 2). These results imply that the time domain is more critical than the spatial domain when normalizing RNNs.

Now, in order to clearly resolve mechanisms of AD in a qualitative manner, consider internal activations of the network. Hidden state activations drastically change over epochs in terms of mean and variance, indicating internal covariate shift (Fig. 7). However, detrended output is much more stable over epochs, indicating that AD successfully reduces internal covariate shift. These results support the notion that accelerated convergence depends on eliminating internal covariate shift [10].

At the level of a single neuron, Fig. 8 plots four time series: the candidate hidden state, hidden state, update gate, and detrended output neurons in the second ConvGRU layer after training. From the perspective of detrending, one may think of the candidate hidden state, hidden state, and update gate in terms of input, trend, and decay factor, respectively. Trends that contain the low-frequency component are successfully estimated and removed from inputs to generate detrended outputs of each neuron (Fig. 8(a) and (b)). These results are similar to the example of detrending (Fig. 1), implying that AD really works as a detrending method. However, unlike conventional detrending methods, AD automatically controls the degree of detrending by changing the decay factor over time. AD basically removes a low-frequency component from input, but a high-frequency component can also be removed by increasing the decay factor as required (Fig. 8(c) and (d)). Furthermore, although the time series in Fig. 8(b) and (c) come from the same neuron, AD works very differently depending on input. These results suggest that AD adaptively provides the proper degree of detrending to suit each time, neuron, and sample. Note that BN and LN control the degree of normalization by using an affine transformation, but it is static rather than dynamic because the parameters of an affine transformation are fixed after training.

Interestingly, in Fig. 8(d), the detrended output is almost flat at zero until around 30 time steps because of the high decay factor, but then suddenly the decay factor converges to zero. Then for the remaining time steps, the trend is fixed to -1 while the input increases rapidly from -1 to 1, and the detrended output (trend subtracted from input) becomes 2 (1-(-1)=2). If this sudden increase in the input after 30 time steps is important for correct classification, it is clear that AD improves discrimination between classes by enhancing true class-related information while removing irrelevant information. Note that the detrended output of the second ConvGRU layer is given directly to the fully-connected layer for classification. Therefore, we argue that control over the degree of detrending by AD not only reduces temporal internal covariate shift, but also improves classification performance or generalization capability.

### *4.3.5. Synergy between Adaptive Detrending and Spatial Normalization*

Because (1) normalization in the spatial or temporal domain is beneficial for training ConvGRU (see Section 4.3.4) and (2) two domains do not overlap, each of these improvements may be combined by applying AD together with spatial normalization methods. When used with BN or with LN, AD speeds convergence more than BN, LN, or AD used alone (abbreviated as BN+AD and LN+AD, respectively)(Fig. 6). These results empirically verify our hypothesis that utilizing the time domain as well as the spatial domain for normalization generates beneficial synergy.

Furthermore, AD solves the difficulty of applying LN to CNNs and their variants, including ConvGRU (as described in Section 4.3.4). More specifically, neuron-wise normalization of AD, which is naturally acquired using the time domain, overcomes the limitation of LN. Once activations of ConvGRU that have different statistics over feature maps are normalized (or detrended) by AD in a neuron-wise manner, detrended activations having similar statistics satisfy the assumption of LN. That is why the improvement from LN to LN+AD achieved by temporal and neuron-wise normalization exceeds the improvement from BN to BN+AD achieved by temporal normalization alone.

## *4.4. Object-Related Action with Modifier Recognition*

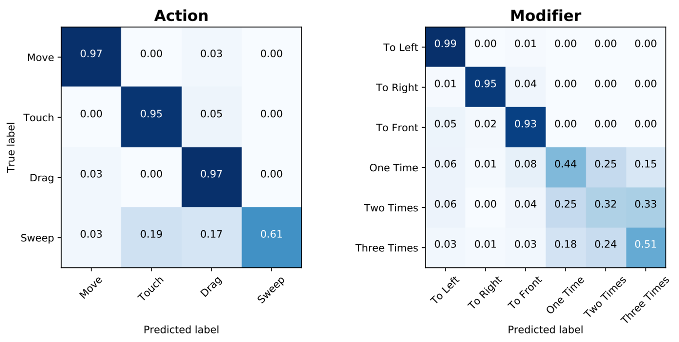
Extending the OA recognition experiment, we tested AD on the object-related action with modifier (OA-M) recognition dataset. The dataset for OA-M recognition consisted of 840 videos in 42 object-action-modifier combination classes created by non-exhaustively and non-redundantly combining four objects ("Box", "Book", "Cup", and "Spray"), four actions ("Move", "Touch", "Drag", and "Sweep"), and six modifiers ("To Left", "To Right", "To Front",

Table 3: Comparison of accuracy on the OA-M recognition dataset.

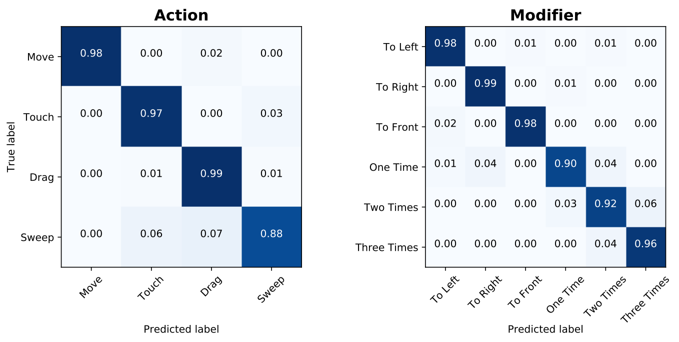
| Model | Accuracy (%) | | | |
|---|---|---|---|---|
| | Object | Action | Modifier | Joint |
| Spatial CNN | 87.9 | 57.3 | 36.7 | 26.4 |
| C3D | 97.8 | 91.5 | 72.8 | 68.8 |
| Baseline | 99.0 | 96.4 | 95.8 | 92.9 |
| AD | 98.2 | 98.4 | 98.4 | 95.4 |
| LN | 97.6 | 96.4 | 95.0 | 90.5 |
| LN+AD | 99.4 | 98.0 | 99.0 | 97.2 |

"One Time", "Two Times", and "Three Times"). Each object-action-modifier combination class was performed by 10 subjects two times each with a randomly selected distractor present in each video. The viewpoint and background were static. Compared with OA recognition, OA-M recognition is more complex because adding the modifier category provides a large number of combination classes, and modifier recognition requires long-term (or contextual) information. For example, a network should wait until a video is finished to discriminate between the modifiers "One Time" and "Two Times". Unlike the OA recognition experiment, we directly used raw videos sampled at a frame rate of 15 *fps* without frame length normalization, because this processing might cause the loss of temporal information. The maximum length of the sampled sequences was 117, which is more than two times longer than the 50 frame normal used during OA recognition. Longer sequences require more capability to capture long-term dependencies [32]. All networks were initialized with a standard deviation of 0.05 and trained with a learning rate of 0.005 over 200 epochs. Because several sequences exist in a mini-batch, each gradient for each sequence was linearly weighted depending on its own sequence length by dividing the maximum sequence length by the specific training sequence length.

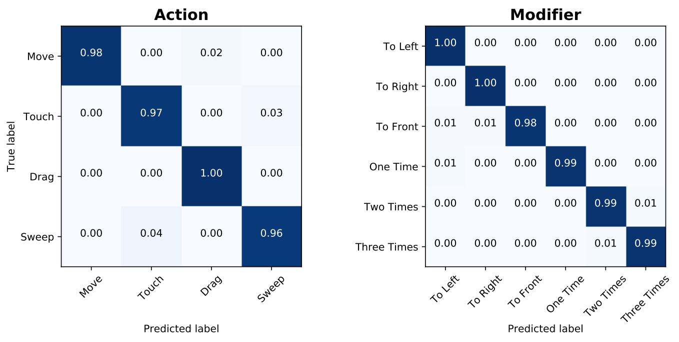
Recognition accuracies clearly distinguish ConvGRU, spatial CNN, and C3D in terms of temporal processing capability (Table 3). Although the short-term



(a) C3D



(b) ConvGRU



(c) LN+AD

Figure 9: Confusion matrices of (a) C3D, (b) ConvGRU, and (c) LN+AD on the OA-M recognition dataset.

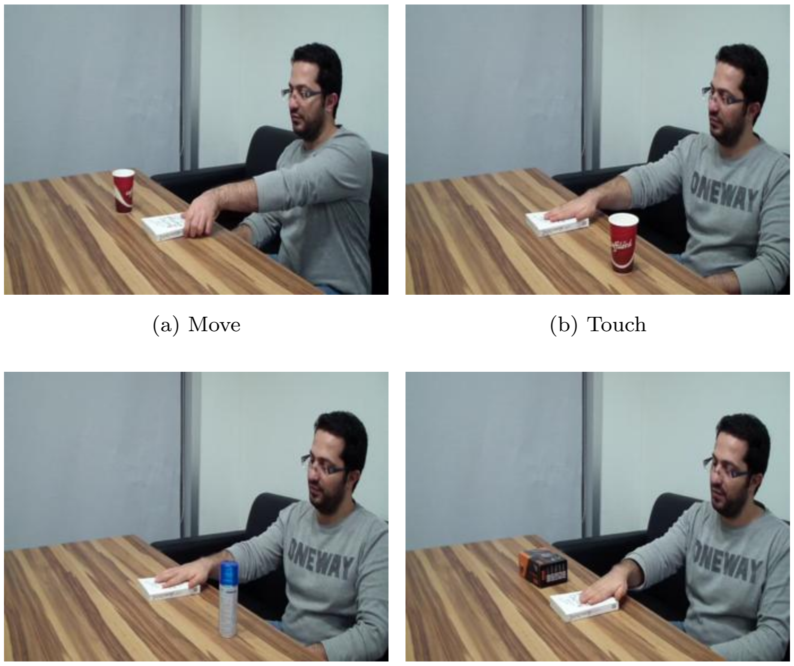

(a) Move (b) Touch (c) Drag (d) Sweep

Figure 10: Sample frame for each action class from the OA-M recognition dataset.

processing capability of C3D is adequate for object and action recognition (97.8% and 91.5%), this is not true for modifier recognition (72.8%) requiring long-term information. To analyze more details of long-term processing capabilities between the networks, we reported the confusion matrices of C3D, ConvGRU, and LN+AD (Fig. 9). C3D shows low recognition rates for "Sweep" class in the action category and counting related classes ("One Time", "Two Times", and "Three Times") in the modifier category compared with those of ConvGRU and LN+AD (Fig. 9). In the case of the modifier category, misclassifications are mostly located within counting classes. These results indicate that C3D correctly recognizes that an input video is related to the counting modifiers, but fails to match the exact numbers because the window-based temporal processing of C3D lacks long-term processing. In the action category, C3D confuses "Sweep" with "Touch" and "Drag" because (1) the shapes of the hand

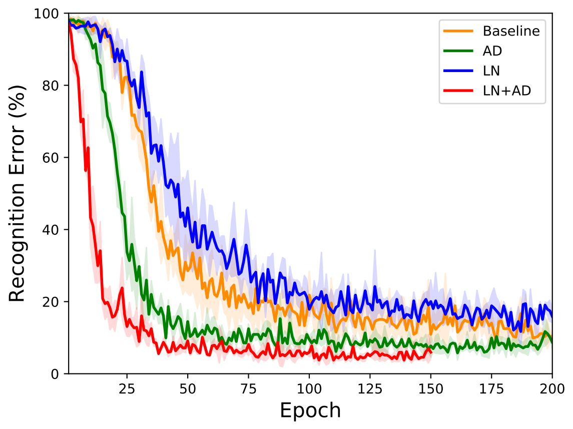


Figure 11: Graph of test recognition error averaged over three splits versus training epochs on the object-related with modifier (OA-M) recognition dataset. Other conditions are the same as those in Fig. 6.

Table 4: Comparison of convergence speed on the OA-M recognition dataset.

| Model | Epochs to Baseline's max accuracy | Degree of acceleration |
|---|---|---|
| Baseline | 200 | ×1.0 |
| AD | 63 | ×3.2 |
| LN+AD | 28 | ×7.1 |

during action are similar (Fig. 10) and (2) the number of samples for "Sweep" is half those of "Touch" and "Drag". To resolve ambiguity between actions, the network should accumulate information over long periods of time.

On the other hand, thanks to the rich spatio-temporal processing capability of ConvGRU, ConvGRU is capable of contextual video recognition, including the failure classes of C3D (Fig. 9(b) and Table 3). This is why ConvGRU must be used, despite the computational burden of ConvGRU. Hence, the proposed method is crucial because it reduces the computational burden of ConvGRU by significantly accelerating training (Fig. 11). Also, we measured the number of epochs required to attain maximum baseline accuracy by the networks. Note that we smoothed out fluctuations in Fig. 11 by using the Savitzky-Golay filter with a window length of 51 and a polynomial order of 3 before measuring the degree of acceleration by each model. AD needs 3.2 times fewer epochs than the baseline (Table 4). Furthermore, AD improves the generalization capability of ConvGRU. Recognition accuracy is improved 2.5% (Table 3).

BN is not used in this experiment because of mini-batch sequence length variability. LN performs worse than the baseline in terms of training speed and recognition accuracy (Fig. 11 and Table 3). As hypothesized in the previous experiment, we think that LN statistical estimation error accumulates over time, leading to poorer results. However, by solving the limitation of LN with neuron-wise normalization of AD, LN+AD shows the most significant improvements in both training speed and generalization over the baseline, as well as over LN or AD, alone. Specifically, LN+AD requires 7.1 and 2.2 times fewer epochs, and improves recognition accuracy by 4.3% and 1.8% compared with those of the baseline and AD, respectively (Tables 3 and 4). The confusion matrix of LN+AD in Fig. 9(c) shows that LN+AD further improves the recognition accuracy of ConvGRU, especially for the classes ("Sweep" in the action category; and "One Time", "Two Times", and "Three Times" in the modifier category) requiring to process long-term information. This result implies that the proposed method helps to deal with long-term processing.

### *4.5. 3D Skeleton-Based Action Recognition*

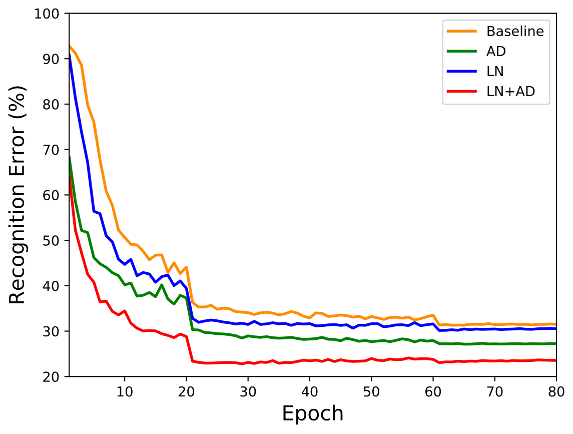
We performed one more experiment for 3D skeleton-based action recognition that needs to extract long-term contextual information for better action recognition [24, 33, 34], which is consistent with the previous two experiments. The goal of the experiment was to show that (1) AD is not restricted to ConvGRU for video recognition, but is more generally applicable up to GRU and (2) AD provides even better training speed, generalization, and synergy with spatial normalization methods when longer temporal processing is required. For this experiment, we used the NTU RGB+D dataset [24] because it is currently the largest dataset for 3D human action recognition with high intra-class and view point variants. The NTU RGB+D dataset consists of 56,880 action samples in

Table 5: Comparison of accuracy on the NTU RGB+D datasets

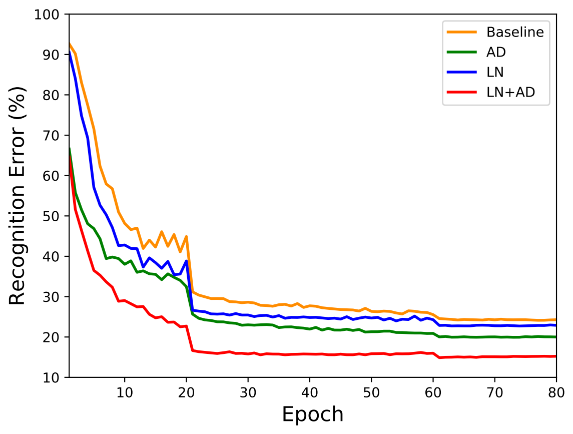
| Model | Accuracy (%) | |
|---|---|---|
| | Cross Subject | Cross View |
| Baseline | 68.8 | 76.0 |
| AD | 73.0 | 80.2 |
| LN | 70.0 | 77.4 |
| LN+AD | 77.3 | 85.2 |
| ST-LSTM + Trust Gate [33] | 69.2 | 77.7 |
| Temporal Conv [35] | 74.3 | 83.1 |
| Clips + CNN + MTLN [34] | 79.6 | 84.8 |
| ST-GCN [36] | 81.5 | 88.3 |

60 action classes. Each sample was collected from 40 subjects with three different camera views and contains four different modalities: RGB videos, depth map sequences, 3D skeleton data, and infrared videos. Among the four modalities, we only used 3D skeleton data for evaluating GRU. Each 3D skeleton data contains at most two subjects and each subject has 25 3D joints (X, Y, and Z), so the dimension of each data is $2\times3\times25=150$. The longest sample in the dataset has 300 time steps, which is 2.6 times longer than that of the OA-M dataset.

All networks consisted of three GRU layers, each with 300 neurons, followed by a fully-connected layer for classification. Output activations of stacked GRU were averaged over time and then passed to the fully-connected layer to generate classification results. The initial hidden state $\mathbf{h}_0$ in GRU was set to 0. The update gate bias was initialized to -5 in order to capture longer temporal dependencies. The gain $\gamma$ and bias $\beta$ of each affine transformation for LN were intialized to 1 and 0, respectively. Note that LN was applied only to the candidate hidden state. All networks were trained by stochastic gradient descent (SGD) with Nesterov momentum 0.9 [26]; the size of mini-batch, L2-norm weight decay, and threshold of gradient clipping were set to 64, 0.0002, and 1,



(a)



(b)

Figure 12: Graph of test recognition error averaged over three runs versus training epochs in the cross view evaluation on the NTU RGB+D dataset. (a) Cross-subject evaluation. (b) Cross-view evaluation.

respectively. All networks were trained with an initial learning rate of 0.1 over 80 epochs. During training, we dropped the learning rate 10 times at 20 and 60 epochs.

We followed two standard evaluation protocols defined in [24]. One is a cross-subject evaluation, in which 20 subjects were used for training (40,320 samples) and the remaining subjects were for testing (16,560 samples). The other is a cross-view evaluation, in which two camera views were used for training (37,920 samples) and the remaining one was for testing (18,960 samples). We repeated the evaluation protocols three times to get robust results.

All normalization methods (AD, LN, and LN+AD) improve the recognition accuracy in both cross-subject and cross-view evaluations (Table 5). Furthermore, AD, LN, and LN+AD achieve a training speed-up (Fig. 12). Unlike previous experiments for contextual video recognition, LN successfully improves both training speed and generalization [13] because the limitation of LN does not occur in GRU. However, even in this case, AD consistently outperforms LN, reflecting the effectiveness of temporal normalization. Surprisingly, although we used a simple GRU without any prior knowledge of the task, recognition accuracies of AD and LN+AD are competitive with recent stat-of-the-art results [33, 35, 34, 36], which indicates that AD and LN+AD provide strong generalization capability. We found that the degree of improvement in learning acceleration and generalization by AD is better in the 3D skeleton-based action recognition experiment than in either the OA or OA-M recognition experiment. This implies that temporal normalization by AD becomes more effective as the maximum sequence length increases because unrolled RNNs over long time scales can be considered as deep feed-forward networks, which suffer from severe internal covariate shift.

## 5. Discussion and Conclusion

This paper proposes a novel temporal normalization method, “adaptive detrending” (AD), to accelerate training of recurrent neural networks (RNNs) by removing the temporal internal covariate shift. Although several normalization methods employing batch normalization (BN) have been proposed to accelerate training of RNNs, these methods utilize only the spatial domain and neglect the time domain for statistical estimation. The key insight of this paper is to view the hidden state of the gated recurrent unit (GRU) as a trend with an exponential moving average. With this in mind, and with simple modifications, we were able to implement AD in GRU. AD has several advantages over other normalization methods: It is highly efficient in terms of computational and memory requirements. Unlike conventional detrending methods that require manual parameter setting, AD learns and estimates trends automatically. AD is generally applicable to both GRU and ConvGRU, which is not the case for either BN or for layer normalization (LN).

We conducted three experiments as follows: (1) object-related action (OA) recognition, (2) object-related action with modifier (OA-M) recognition, and (3) 3D skeleton-based action recognition. These three experiments were organized in terms of the complexity of contextual information in each dataset for examining the effectiveness of AD compared with existing normalization methods, depending on contextual complexity. By adding a modifier category requiring accumulation of long-term contextual information, the OA-M recognition dataset for the second experiment is more contextually complex than the OA dataset for the first experiment. Also, the NTU RGB+D dataset for the third experiment has high contextual complexity due to a very long sequences in the dataset. The present set of experiments demonstrates that (1) convolutional GRU (ConvGRU) has much richer temporal processing capability required for contextual recognition than feed-forward neural networks, (2) AD consistently provides faster convergence and better generalization than baseline and spatial normalization methods, and most importantly, (3) the benefits of AD written in (2) are further enhanced in the case of longer temporal processing by removing internal covariate shift over long time scales. Also, we rediscovered that the negative bias initialization trick for the update gate of GRU, helps to address problems with slow and unstable learning using AD. Qualitative analysis reveals that AD controls the degree of detrending over time, neurons, and samples, accounting for the performance improvement. Furthermore, AD works well with spatial normalization methods. Especially in the case of CNNs, neuron-wise normalization by AD overcomes the main limitation of LN. In conclusion, with little additional overhead, AD substantially alleviates the computational burden of GRU and ConvGRU by reducing the number of epochs required to converge and it converges to better local minima, at the same time demonstrating strong synergy with existing normalization methods. Looking ahead, AD should prove helpful in studies utilizing the rich spatio-temporal processing capability of ConvGRU and its variants.

The detrending mechanisms of AD reminds us of background subtraction that detects foreground by subtracting background. Among many variants of background subtraction to deal with temporal background change in a video, Wren et al. [37] proposed a running Gaussian average method that updates the background estimation over time using the exponential moving average (EMA) in (15). Considering that AD estimates the trend by EMA, the trend and detrended output of AD can be considered as background and foreground, respectively. In the OA and OA-M recognition datasets, the background will be a static or slow component of a video (such as a table, the torsos of subjects, and a distractor) and the foreground will be a fast component of a video (such as the arms of the subjects and target object). Therefore, AD might work as a foreground detector that focuses on motion information crucial for video recognition, leading to better convergence speed and generalization.

For future work, we will extend the current research for contextual video recognition by using a large set of contextually complex videos to achieve deep understanding of video recognition at a higher semantic level. For the purpose of enhancing and promoting research in this direction, we will create a publicly available video dataset with high-level concept labels through crowdsourcing platforms, such as Amazon Mechanical Turk (AMT). In addition, we will apply AD to speech recognition. Step-wise BN and LN [12, 13] approaches are difficult to apply to speech recognition because these tend to lose dynamics of speech signals. The sequence-wise BN [11] approach has demonstrated accelerated training and performance improvement in speech recognition [38] by preserving dynamics of speech signals. However, it cannot provide different degrees of normalization over time as might be required for further elimination of internal covariate shift. Automatic control over the degree of detrending by AD is expected to balance preservation of speech signal dynamics with the reduction

Table A.1: Convolutional neural network configuration. The format of the table is the same as that in Table 1.

| Layer | Type | Filter | Stride | Pad |
|---|---|---|---|---|
| 1 | Conv (ReLU) | 7×7×3×96 | 2×2 | 0×0 |
| 2 | Max | 2×2 | 2×2 | 0×0 |
| 3 | Conv (ReLU) | 5×5×96×256 | 2×2 | 1×1 |
| 4 | Max | 2×2 | 2×2 | 0×0 |
| 5 | Conv (ReLU) | 3×3×256×512 | 1×1 | 1×1 |
| 6 | Conv (ReLU) | 3×3×512×512 | 1×1 | 1×1 |
| 7 | Conv (ReLU) | 3×3×512×512 | 1×1 | 1×1 |
| 8 | Global Avg | 6×6 | - | - |
| 9 | FC (Softmax) | 1×1×512×$C_1$ | - | - |
|  | ⋮ |  |  |  |
|  | FC (Softmax) | 1×1×512×$C_N$ | - | - |

of internal covariate shift.

**Acknowledgements**

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No.2014R1A2A2A010 05491) and Okinawa Institute of Science and Technology Graduate University, Japan. Portions of the research in this paper used the NTU RGB+D Action Recognition Dataset made available by the ROSE Lab at the Nanyang Technological University, Singapore.

## Appendix A. Spatial Convolutional Neural Network

We followed the spatial stream CNN architecture used in [3]. However, we replaced the last max pooling and two fully connected layers with a global average pooling layer. The networks consisted of five convolutional (Conv) layers, two max pooling (Max) layers, one global average pooling (Global Avg) layer, and one fully-connected (FC) layer (Table A.1).

Following Section 4.1.2 for training, stochastic gradient descent (SGD) with Nesterov momentum 0.9 [26] was applied to train the networks. The size of mini-batch, L2-norm weight decay, and threshold of gradient clipping were set to 8, 0.0005, and 10, respectively. All weights and biases of the networks were initialized from a zero-mean Gaussian distribution with a standard deviation of 0.03. Also, data pre-processing and augmentation methods are exactly the same as those in Section 4.1.3. All networks were trained with a learning rate of 0.01 over 200 epochs on the object-related action dataset, and over 300 epochs on the object-related action with modifier dataset.

We followed the evaluation protocol used in the spatial stream CNN [3]. Specifically, 25 frames were sampled with equal spacing from a video, and 10 crops (1 center, 4 corners, and their horizontal flipping) were obtained from each sampled frame. Then, scores were averaged across sampled frames and crops of each sampled frame to obtain the final classification accuracy for a video.

## Appendix B. Convolutional 3D Network

Because convolutional 3D (C3D) networks [4] receive $L$ consecutive frames as input to process short-term information, convolution and pooling operations are extended from 2D (spatial) to 3D (spatio-temporal). Networks consisted of four 3D convolution (3D Conv) layers, three 3D max pooling (3D Max) layers, one 3D global average pooling (3D Global Avg) layer, and one fully-connected (FC) layer (Table B.1).

Details of (1) training and (2) data pre-processing and augmentation are the same as those in Appendix A, except that the number of stacked frames $L$ was set to 16. All networks were trained with a learning rate of 0.02 over 200 epochs on the object-related action dataset, and with a learning rate of 0.01 over 300 epochs on the object-related action with modifier dataset.

We followed the evaluation protocol used in the temporal stream CNN [3].

Table B.1: Convolutional 3D network configuration. The format of the table is the same as that in Table 1 except that time is added as the first dimension of the filter, stride, and pad.

| Layer | Type | Filter | Stride | Pad |
|---|---|---|---|---|
| 1 | 3D Conv (ReLU) | 3×7×7×3×32 | 1×2×2 | 1×0×0 |
| 2 | 3D Max | 2×2×2 | 2×2×2 | 0×0×0 |
| 3 | 3D Conv (ReLU) | 3×5×5×32×64 | 1×2×2 | 1×1×1 |
| 4 | 3D Max | 2×2×2 | 2×2×2 | 0×0×0 |
| 5 | 3D Conv (ReLU) | 3×3×3×64×128 | 1×1×1 | 1×1×1 |
| 6 | 3D Max | 2×2×2 | 2×2×2 | 0×0×0 |
| 7 | 3D Conv (ReLU) | 3×3×3×128×256 | 1×1×1 | 1×1×1 |
| 8 | 3D Global Avg | 2×3×3 | - | - |
| 9 | FC (Softmax) | 1×1×1×256×$C_1$ | - | - |
|  | ⋮ |  |  |  |
|  | FC (Softmax) | 1×1×1×256×$C_N$ | - | - |

We sampled five video clips equally spaced, each with 16 consecutive frames. From each clip, we obtained 10 sample crops: 1 from the center, 4 from each corner, and then by horizontally flipping these 5 crops, a total of 10 was obtained. Final classification accuracy for each video was obtained by averaging the scores across the sampled clips and crops of each sampled clip.

## References

[1] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. `doi:10.1109/5.726791`.

[2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[3] K. Simonyan, A. Zisserman, Two-stream convolutional networks for action recognition in videos, in: Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 568–576.

[4] D. Tran, L. Bourdev, R. Fergus, L. Torresani, M. Paluri, Learning spatiotemporal features with 3d convolutional networks, in: IEEE International Conference on Computer Vision (ICCV), 2015.

[5] M. Jung, J. Hwang, J. Tani, Self-organization of spatio-temporal hierarchy via learning of dynamic visual image patterns on action sequences, PLOS ONE 10 (7) (2015) 1–16. `doi:10.1371/journal.pone.0131214`.

[6] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, W.-c. Woo, Convolutional lstm network: A machine learning approach for precipitation nowcasting, in: C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 802–810.

[7] N. Ballas, L. Yao, C. Pal, A. C. Courville, Delving deeper into convolutional networks for learning video representations, CoRR abs/1511.06432.

[8] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, K. Kavukcuoglu, Video pixel networks, CoRR abs/1610.00527.

[9] B. Romera-Paredes, P. H. S. Torr, Recurrent instance segmentation, in: B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI, Springer International Publishing, Cham, 2016, pp. 312–329. `doi:10.1007/978-3-319-46466-4_19`.

[10] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, Vol. 37

of Proceedings of Machine Learning Research, PMLR, Lille, France, 2015, pp. 448–456.

[11] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, Y. Bengio, Batch normalized recurrent neural networks, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016, pp. 2657–2661. doi:10.1109/ICASSP.2016.7472159.

[12] T. Cooijmans, N. Ballas, C. Laurent, A. Courville, Recurrent batch normalization, in: International Conference on Learning Representations (ICLR), 2017.

[13] L. J. Ba, R. Kiros, G. E. Hinton, Layer normalization, CoRR abs/1607.06450.

[14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724–1734.

[15] S. Hochreiter, Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München (1991).

[16] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166. doi:10.1109/72.279181.

[17] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[18] J. Kenney, E. Keeping, Moving averages, in: Mathematics of Statistics, Part 1, 3rd Edition, Van Nostrand, 1962, Ch. 14.2, pp. 221–223.

[19] NIST/SEMATECH, Single exponential smoothing, in: NIST/SEMATECH e-Handbook of Statistical Methods, Ch. 6.4.3.1.

[20] H. Lee, M. Jung, J. Tani, Recognition of visually perceived compositional human actions by multiple spatio-temporal scales recurrent neural networks, CoRR abs/1602.01921.

[21] G. I. Parisi, J. Tani, C. Weber, S. Wermter, Lifelong learning of human actions with deep neural network self-organization, Neural Networks 96 (Supplement C) (2017) 137 – 149. `doi:https://doi.org/10.1016/j.neunet.2017.09.001`.

[22] K. Soomro, A. R. Zamir, M. Shah, UCF101: A dataset of 101 human actions classes from videos in the wild, CoRR abs/1212.0402.

[23] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, T. Serre, HMDB: a large video database for human motion recognition, in: IEEE International Conference on Computer Vision (ICCV), 2011.

[24] A. Shahroudy, J. Liu, T.-T. Ng, G. Wang, Ntu rgb+d: A large scale dataset for 3d human activity analysis, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[25] M. Lin, Q. Chen, S. Yan, Network in network, CoRR abs/1312.4400.

[26] Y. Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, Soviet Mathematics Doklady 27 (1983) 372–376.

[27] R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: A Matlab-like Environment for Machine Learning, in: BigLearn, NIPS Workshop, 2011.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, in: NIPS 2017 Autodiff Workshop, 2017.

[29] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: S. Dasgupta, D. McAllester (Eds.), Proceedings of the

30th International Conference on Machine Learning, Vol. 28 of Proceedings of Machine Learning Research, PMLR, Atlanta, Georgia, USA, 2013, pp. 1310–1318.

[30] F. A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with LSTM, Neural Computation 12 (10) (2000) 2451–2471. `doi:10.1162/089976600300015015`.

[31] R. Jozefowicz, W. Zaremba, I. Sutskever, An empirical exploration of recurrent network architectures, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, Vol. 37 of Proceedings of Machine Learning Research, PMLR, Lille, France, 2015, pp. 2342–2350.

[32] Z. Yu, D. S. Moirangthem, M. Lee, Continuous timescale long-short term memory neural network for human intent understanding, Frontiers in Neurorobotics 11 (2017) 42. `doi:10.3389/fnbot.2017.00042`.

[33] J. Liu, A. Shahroudy, D. Xu, G. Wang, Spatio-temporal LSTM with trust gates for 3d human action recognition, in: European Conference on Computer Vision (ECCV), 2016, pp. 816–833.

[34] Q. Ke, M. Bennamoun, S. An, F. A. Sohel, F. Boussaïd, A new representation of skeleton sequences for 3d action recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4570–4579.

[35] T. S. Kim, A. Reiter, Interpretable 3d human action analysis with temporal convolutional networks, in: CVPR 2017 Workshop: Brave new ideas for motion representations in videos II, 2017.

[36] S. Yan, Y. Xiong, D. Lin, Spatial temporal graph convolutional networks for skeleton-based action recognition, in: Association for the Advancement of Artificial Intelligence (AAAI), 2018.

[37] C. R. Wren, A. Azarbayejani, T. Darrell, A. P. Pentland, Pfinder: real-time tracking of the human body, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (7) (1997) 780–785.

[38] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, A. Hannun, B. Jun, T. Han, P. LeGresley, X. Li, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, S. Qian, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, C. Wang, Y. Wang, Z. Wang, B. Xiao, Y. Xie, D. Yogatama, J. Zhan, Z. Zhu, Deep speech 2 : End-to-end speech recognition in english and mandarin, in: M. F. Balcan, K. Q. Weinberger (Eds.), Proceedings of The 33rd International Conference on Machine Learning, Vol. 48 of Proceedings of Machine Learning Research, PMLR, New York, New York, USA, 2016, pp. 173–182.