Learning long-term dependencies for action recognition with a biologically-inspired deep network

Yemin Shi[1,2], Yonghong Tian[1,2]*, Yaowei Wang[3], Wei Zeng[1,2], Tiejun Huang[1,2]
[1] National Engineering Laboratory for Video Technology, School of EE&CS,
Peking University, Beijing, China
[2] Cooperative Medianet Innovation Center, China
[3] School of Information and Electronics, Beijing Institute of Technology, Beijing, China

Abstract

Despite a lot of research efforts devoted in recent years, how to efficiently learn long-term dependencies from sequences still remains a pretty challenging task. As one of the key models for sequence learning, recurrent neural network (RNN) and its variants such as long short term memory (LSTM) and gated recurrent unit (GRU) are still not powerful enough in practice. One possible reason is that they have only feedforward connections, which is different from the biological neural system that is typically composed of both feedforward and feedback connections. To address this problem, this paper proposes a biologically-inspired deep network, called shuttleNet[1]*. Technologically, the shuttleNet consists of several processors, each of which is a GRU while associated with multiple groups of hidden states. Unlike traditional RNNs, all processors inside shuttleNet are loop connected to mimic the brain's feedforward and feedback connections, in which they are shared across multiple pathways in the loop connection. Attention mechanism is then employed to select the best information flow pathway. Extensive experiments conducted on two benchmark datasets (i.e UCF101 and HMDB51) show that we can beat state-of-the-art methods by simply embedding shuttleNet into a CNN-RNN framework.*

1. Introduction

Deep neural networks (DNNs) have achieved great success in recent years as they are able to learn complex features and patterns from data. A typical DNN has multiple nonlinear layers which are connected with feedforward connections. In order to gain better performance, very deep structure must be considered to construct a DNN, which will then leads to massive amount of parameters and high risk of over-fitting.

In biological neural system, the visual areas of the brain are interconnected in a complex pattern of feedforward, lateral, and feedback pathways [12, 25]. Feedback connections are ubiquitous throughout the cortex, and subcortical regions in ascending hierarchical pathways also receive a large amount of feedback from cortical areas [9, 29]. This fact motivates us that DNNs may benefit a lot from imitating the biological neural system by introducing lateral or feedback connections.

Following the similar idea, recurrent neural networks (RNNs) introduce lateral connections to the temporal domain to condition their present state on the entire history of inputs. Because of the temporal lateral connection mechanism, RNNs are able to capture long-term dependencies in sequential data over an extended period of time. Moreover, RNNs have been theoretically proved to be a Turing-complete machine, indicating that they can be used to approximate any functions [32]. As one variant of RNNs, long short term memory (LSTM) [17] is proposed to solve the gradient vanishing and exploding problems. When unfolded in time, LSTMs are equivalent to very deep neural networks that share model parameters and receive the input at each time step. The parameter-sharing mechanism guarantees that there are not too many parameters and consequently the network is trainable. Many works [7, 30, 47] have proved the effectiveness of LSTM on the action recognition task from video sequences.

In this paper, we propose a new kind of biologically-inspired deep neural network, called shuttleNet, which is composed of both feedforward and feedback connections. Technologically, the shuttleNet consists of N processors and $N \times D$ groups of hidden states. In our model, each processor is a simple Gated Recurrent Unit [5], typically containing a group of weight matrices and associated with D groups of hidden states. All the N processors are orga-

[*] Corresponding author: Yonghong Tian (email: yhtian@pku.edu.cn).
[1] The code is available at `https://github.com/shiyemin/shuttleNet`

nized as a ring, each of which clockwise connects to the next processor with a stride of K. If an input x is fed into the n^{th} processor, it will return to the original processor after $\frac{N}{K}$ steps (typically $D \geq \frac{N}{K}$). This process forms a loop connection which can be viewed as a combination of both feedforward and feedback connections. In the network, x will be fed into all the N processors and be passed through D steps, consequently leading to N pathways. Moreover, these processors can be shared in different pathways while having standalone states at each step. After that, the attention mechanism [3] is employed to select the best pathway, suppose that each pathway corresponds to one potential prediction. In short, the network works in a shuttle way, thus the name shuttleNet.

We evaluate the proposed shuttleNet for action recognition task on two benchmark datasets, namely UCF101 and HMDB51. Experimental results show that the shuttleNet outperforms LSTMs and GRUs remarkably. By simply embedding our shuttleNet into a CNN-RNN network, we can beat most of the state-of-the-art action recognition methods on both datasets.

The rest of the paper is organized as follows: In section 2, we review the related work. We will briefly introduce the biological background and our motivation in section 3. The proposed shuttleNet is presented in section 4. Experimental results are discussed in section 5. Finally, section 6 concludes this paper.

2. Related work

Basically, action recognition aims at categorizing the actions or behaviors of one or more persons in a video sequence. Two-stream ConvNets [33] is widely recognized as the first successful deep learning framework for action recognition. It extracts the spatial and temporal characteristics in one framework, and trains the standalone CNNs for two streams separately. Wang *et al.* [44] also successfully trained very deep two-stream ConvNets on the UCF101 dataset. Similarly, trajectory-pooled deep-convolutional descriptor (TDD) was proposed by Wang *et al.* [42], which shares the merits of both hand-crafted features such as dense trajectories [40, 41] and deeply-learnt features. However, two-stream ConvNets did not capture and utlize the long-term dependence in the network.

Hierarchical recurrent neural network [8] is one of the earliest works which attempted to improve the efficiency of capturing long term dependency. Long short term memory (LSTM) [17], the most successful approach to deal with vanishing gradients, was proposed by Hochreiter and Schmidhuber. Basically, LSTM relies on a fantastic structure made of gates to control the flow of information to the hidden neurons. Peephole LSTM [13] adds peepholes to some gates so as to allow them look at the cell state. Gated recurrent unit (GRU), introduced by Cho *et al* [5], is a slightly more dramatic variation on the LSTM, which combines the forget and input gates into a single update gate and merges the cell state and hidden state. Mikolov *et al.* [26] proposed to add a hidden layer to RNNs and make the weight matrix close to identity.

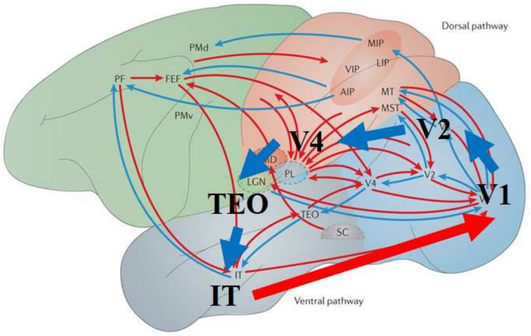

Figure 1. Illustration of the visual cortical pathways [20]. Feedforward connections are represented by blue arrows and feedback connections are represented by red arrows. A pathway is enlarged for convenience. The feedforward and feedback connections together generate a loop connection.

It should be noted that LSTMs were introduced to model long-term actions for action recognition recently. Yue-Hei *et al.* [47] and Donahue *et al.* [7] proposed their own recurrent networks respectively by connecting LSTMs to CNNs. Wu *et al.* [46] achieved the state-of-the-art performance by connecting CNNs and LSTMs under the hybrid deep learning framework. Shi *et al.* [31, 30] also introduced their DTD and sDTD to model the dependence on the temporal domain. Nevertheless, LSTMs and GRUs are still not powerful enough for action recogntion in practice. One possible reason is that they have only feedforward connections, which is different from biological neural network that is typically composed of both feedforward and feedback connections. Therefore, this paper proposes a biologically-inspired deep network, called shuttleNet, by introducing loop connections in the network. We will show the shuttleNet outperforms LSTMs and GRUs remarkably for action recognition.

3. Background

Hierarchical processing is the key to understand vision system. It consists of hierarchically organized distinct anatomical areas functionally specialized for processing different aspects of a visual object [12]. These visual areas are interconnected through ascending feedforward connections, descending feedback connections and connections from neural structures at the same hierarchical level [22]. The lateral geniculate nucleus (LGN) is a relay center in the thalamus for the visual pathway. There is evidence that only 10% of inputs to LGN come from the retina and 90% are feedback modulatory inputs from cortex and the brain-

Table 1. Some popular deep networks and their depths.

LeNet-5	VGG-16	GoogLeNet	ResNet-152
5	16	22	152

Please refer to [24, 34, 38, 15].

stem. As shown in Figure 1, the feedback connections play an important role in visual cortical pathways.

On the contrary, DNN researchers concentrate on designing deeper network with only feedforward connections, as shown in Table 1. As the model becoming more and more deeper, it will consume more time and computing resource to train. However, LGN and V1[2] have only 6 layers but still are powerful enough for understanding vision information. A reasonable way to re-design the networks is to imitate the feedback connections of visual cortical pathways.

A feedback connection is a connection along which the information can go back to the previous layer after a few feedforward steps. Together with the feedforward connections, the feedback connections can always result in circular paths. This means that it is not necessary to explicitly generate the feedback connections and we can introduce feedback connections to neural network by generating loop connections.

4. shuttleNet

In this section, we will first present the overall framework of the proposed shuttleNet. After that, we will describe the details about its key components one by one.

4.1. The overall framework

Basically, shuttleNet is a biologically-inspired deep neural network, which introduce loop connection and processor-sharing mechanism to the network. As shown in Figure 2, the input x_t at time t is projected by a fully-connected layer so that the inputs to processors have the same length as the states. The main module of shuttleNet consists of N processors. The projected input is fed into all the N processors and pass through D steps with a stride of K, consequently resulting in N pathways. At every step, all processors work simultaneously and generate their outputs. During these steps, all processors are shared while having standalone hidden states. Finally, attention mechanism is applied to select the best pathway based on outputs at the last step and input.

A noticeable advantage of shuttleNet is that, even though having complex connections, no extra parameters are needed except for the attention module. Considering that the number of parameters does not increase obviously, we can effectively lower the risk of over-fitting.

Our model works like recurrent neural network, and consists of three modules: an input projector, a group of loop-connected processors and an output selector. The entire model (all three components) is trained via backpropagation through time, receiving gradients from every time step, which are then propagated through the unrolled network.

4.2. Input projector

The input projector is crucial when the inputs and hidden states have different lengths. It makes sure the inputs of processors and hidden states have the same number of dimensions. However, when the input feature length is equal to the length of hidden state, the input projector is not necessary. Typically, the input at time t is a fixed length feature, e.g. the output of CNNs or a bag-of-the-words (BoW) representation. In this work, the inputs are outputs of the last convolution or pooling layer of CNNs. We use a simple fully-connected layer with batch normalization [18] to project the input. More precisely, let the input be x_t. The projected representation of this input is computed as follows:

$$x'_t = w_p x_t \tag{1}$$

$$x^o_t = max(\frac{x'_t - E[x'_t]}{\sqrt{Var[x'_t]}} + b, 0) \tag{2}$$

where w_p and b are the learnable parameters, $E[x] = \frac{1}{N}\sum_{i=1}^{N} x_i$, $Var[x] = \frac{1}{N}\sum_{i=1}^{N}(x_i - E[x])^2$, N is batch size and $max(x, 0)$ is the ReLU [27] activation function.

4.3. Loop-connected processors

As discussed in section 3, the visual pathways have more feedback connections than feedforward connections. Our first intuition is to design a network with feedforward and feedback connections like the visual pathways. However, to make the model computable and stable, we choose to implement the connections as a simplified version, loop connections. We will then describe it with mathematical linguistic forms.

For most known network structure, layers are stacked one by one and the last layer produces the final output. The layers of a network can be considered as some nonlinear functions. For input x, a four-layer network generate the output as follows:

$$y = f_4(f_3(f_2(f_1(x)))) \tag{3}$$

where f_1, f_2, f_3 and f_4 are the four layers and y is the output. In this setting, the network projects input from its original space, where it is hard to classify the samples, to another space, where it is easier to classify the samples. Basically, a network can be decomposed into several nonlinear functions. There are a lot of redundancy in network and it

[2] The primary visual area (V1) of the cerebral cortex is the first stage of cortical processing of visual information.

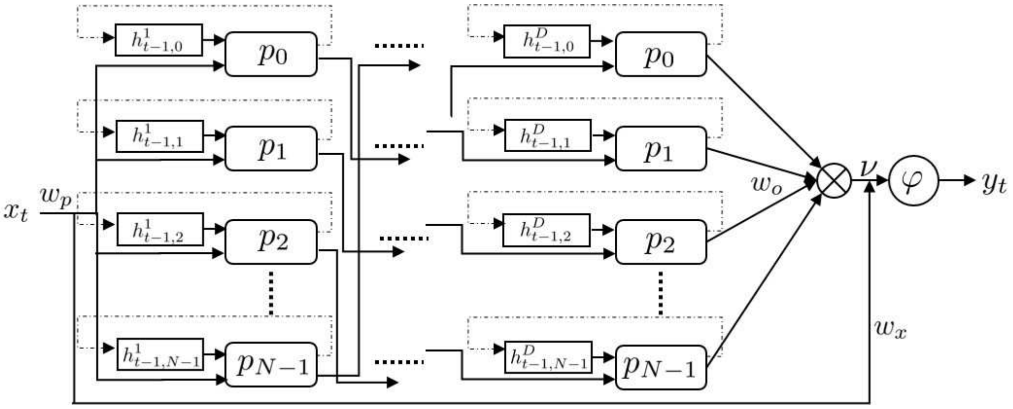

Figure 2. Diagram of the shuttleNet. The processors are represented by p_n and the hidden states are represented by $h^d_{t,n}$. The information flow pathways are represented by the arrow lines. Every column is called one step. The input x_t is first projected by w_p. The projected input is fed into all the N processors and pass through D steps with a stride of K, consequently resulting in N pathways. At every step, all processors work simultaneously and generate their outputs. During the steps, all processors are shared while keeping standalone hidden states. Finally, attention mechanism is applied to select the best pathway based on outputs at the last step and input.

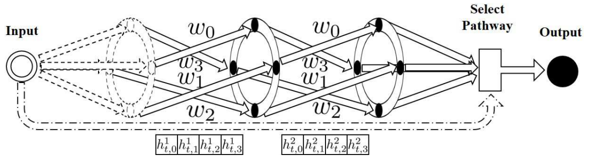
Input
w_0
w_3
w_1
w_2
w_0
w_3
w_1
w_2
Select Pathway
Output
$h^1_{t,0}$ $h^1_{t,1}$ $h^1_{t,2}$ $h^1_{t,3}$
$h^2_{t,0}$ $h^2_{t,1}$ $h^2_{t,2}$ $h^2_{t,3}$

Figure 3. Illustration of a 4-processor-2-step shuttleNet.

is possible that we share some functions while achieving comparable or better performance as follows:

$$y = f_3(f_1(f_2(f_1(x)))) \tag{4}$$

There is an obvious advantage in this kind of layer-sharing mechanism that we can decrease number of parameters while keeping the network depth consequently reducing over-fitting.

In order to tap into the potentials of layer sharing, we further develop this idea. A group of layers are organized into a ring. The input x is first fed into all these layers. Then the outputs of the layers are clockwise fed into the next layer with a stride of K.

Formally, for N layers $\{f_0, f_1, ..., f_{N-1}\}$ and a given input x, we compute N outputs as follows:

$$y_j = f_{(j+D*K)\%N}(...(f_{(j+K)\%N}(f_j(x)))) \tag{5}$$

where $j \in \{0, 1, \ldots, N-1\}$, % is the modulus operator, D is the user defined maximum computing steps and y_j is the output of the j^{th} pathway. Therefore, there are totally N outputs. Because that each layer can be reused multiple times in each pathway consequently resulting in multiple information flow loops and layers are organized into a circle, we call it loop connection.

For a specific application like action recognition, we will implement the layers with recurrent neural networks (RNNs), e.g. LSTMs or GRUs. For the convenience of expression, we will also use "processor“ to denote the RNN layer.

For a RNN layer, the hidden state h is used to remember history information. The n^{th} RNN processor p_n at d^{th} step and time t in the loop connection works as follows:

$$o^d_{t,n}, h_{t,n} = p_n(o^{d-1}_{t,n-K}, h_{t,n}) \tag{6}$$

$$o^{d+1}_{t,n}, h_{t,n} = p_n(o^d_{t,n-K}, h_{t,n}) \tag{7}$$

where $o^d_{t,n}$ is the output of the n^{th} processor at d^{th} step and time t and $h_{t,n}$ is the hidden state of the n^{th} processor. The equation 6 means that $h_{t,n}$ is used by p_n at the d^{th} step while equation 7 means that $h_{t,n}$ is also used by p_n at the $(d+1)^{th}$ step. However in the loop connection, the two equations are happening in two different information pathways. The chaotic dependencies will end up with unstable training process.

Our solution of the chaotic dependencies is to keep an individual hidden state for each processor at every step while sharing the processor itself between steps. Formally, the processor p_n at d^{th} step and time t in the modified loop connection works as follows:

$$o^d_{t,n}, h^d_{t,n} = p_n(o^{d-1}_{t,n-K}, h^d_{t-1,n}) \tag{8}$$

$$o^{d+1}_{t,n}, h^{d+1}_{t,n} = p_n(o^d_{t,n-K}, h^{d+1}_{t-1,n}) \tag{9}$$

where $h^d_{t,n}$ is hidden state of p_n at d^{th} step and time t. In these equations, each group of hidden state is only used in one pathway and transmits along time.

4.4. Output selector

According to section 4.2 and 4.3, given input x_t at time t, after going through the input projector and N pathways, there are N outputs $\{o^D_{t,0}, o^D_{t,1}, \cdots, o^D_{t,N-1}\}$. We need to choose the output which will produce the best prediction. In this paper, attention mechanism [3] is employed to help select the proper output. Specifically, the output is computed using the following equations:

$$e_{t,n} = \nu^T tanh(w_x x^o_t + w_o o^D_{t,n}) \tag{10}$$

$$\alpha_{t,n} = \frac{exp(e_{t,n})}{\sum_{l=0}^{N-1} exp(e_{t,l})} \tag{11}$$

$$y_t = \sum_{l=0}^{N-1} \alpha_{t,l} o^D_{t,l} \tag{12}$$

where vector ν and the weight matrixes w_x and w_o are the learnable parameters and the y_t is the output of shuttleNet. The vector e_t assigns a weight for each output $o^D_{t,n}$, which means how much attention should be put on $o^D_{t,n}$. These attention weights are normalized by softmax to create the attention mask α_t over the outputs.

The 3D structure of a 4-processor-2-step shuttleNet with $K = 1$ is shown in Figure 3. The input is replicated so that each processor has its own input. And processors are formed as a circle and work at every step simultaneously. The processors have standalone hidden state at every step. The information flow is propagated in a wheel way. Finally, attention mechanism is utilized to select the best pathway.

It should be noted that when N and D are 1, shuttleNet is completely equivalent to a RNN layer. When N is larger than 1 and D is 1, shuttleNet can be seen as a bank of RNN layers. When N is 1 and D is larger than 1, shuttleNet is a weight-shared stacked RNN.

5. Experiments

In this section, we will first introduce the detail of datasets and their corresponding evaluation schemes. Then, we describe the implementation details of our model. To find out the effect of each parameter, we explore experiments with multiple parameter settings and prove the effectiveness of shuttleNet. We finally report the experimental results and compare shuttleNet with the state-of-the-art methods to demonstrate its superior performance.

5.1. Datasets

To verify the effectiveness of shuttleNet, we conduct experiments on two benchmark datasets: HMDB51 [21] and UCF101 [35].

The HMDB51 dataset is a large collection of realistic videos from various sources, including movies and web videos. It is composed of 6,766 video clips from 51 action categories, with each category containing at least 100 clips. The action categories include simple facial actions, general body movements and human interactions. Our experiments follow the original evaluation scheme, and average accuracy over the three train-test splits is reported.

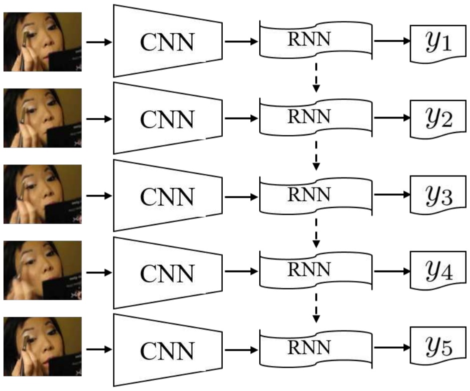

Figure 4. General action recognition network structure. The video frames are fed into CNNs to learn representations. The CNN representations are then fed into RNNs to learn temporal features. Finally, the output of RNNs is used to predict action label.

The UCF101 dataset is one of the most popular action recognition benchmarks. It contains 13,320 video clips (27 hours in total) from 101 action classes and there are at least 100 video clips for each class. The 101 classes are divided into five groups: Body-Motion, Human-Human Interactions, Human-Object Interactions, Playing Musical Instruments and Sports. Following [19], we conduct evaluations using 3 train/test splits, which is currently the most popular setting in using this dataset. Results are measured by classification accuracy on each split and we report the mean accuracy over the three splits.

Compared with very large datasets used for image classification, the datasets for action recognition is relatively smaller. Therefore, we pre-train our model on the ImageNet dataset [6]. Unlike pure CNN models [33, 45], which are fully pre-trained on ImageNet, our network has new layers which can not be trained on image datasets. In order to apply shuttleNet to smaller dataset like HMDB51, we transfer the learnt model from UCF101 to HMDB51.

5.2. Implementation details

As shown in Figure 4, we use a CNN-RNN network structure as in [7, 30] and use two-stream framework [33] to get our final prediction. In order to conduct as many exploration experiments as we can, we use a relative small network, GoogLeNet [38], as our CNN implementation to test these parameter combinations. After choosing the best hyper-parameters, we switch to Inception-ResNet-v2 [37] to get the final performance. To highlight effectiveness of shuttleNet and ensure fairness, the RNN part is tested with several options: 2-layer LSTM [17], 2-layer GRU [5], 3-layer GRU and 1-layer shuttleNet. The shuttleNet may have 1–3 processors and 1–3 steps.

We implement shuttleNet with TensorFlow [1] and use

GRU as the processor because of less parameter number and good performance compared with other RNNs.

To extract optical flow, we choose the TVL1 optical flow algorithm [48] and use the OpenCV GPU implementation. We discretize the optical flow fields into interval of $[0, 255]$ by a linear transformation and save them as images. Unlike the two-stream ConvNets in [33], whose temporal stream input is volumes of stacking optical flow fields ($224 \times 224 \times 2F$, where F is the number of stacking flows and is set to 10), our temporal stream input is single optical flow. An optical flow field is computed from two consecutive frames and composed of vertical and horizontal flows. To make use of the optical flow, the flow-x, flow-y and their quadratic mean is used to form a three channel image.

After initializing with the pre-trained ImageNet model, the network is trained with mini-batch stochastic gradient descent with momentum (set to 0.9). We read 16 frames/optical flows with a stride of 5 from each video as one sample for the RNN. We resize all input images to 256×256, and then use the fixed-crop strategy [44] to crop a 224×224 region from images or their horizontal flip. Because the 16 consecutive samples are needed in the RNN, we also force images from the same video to crop the same region. In the test phase, we sample 4 corners and the center from each image and its horizontal flip, and 5 samples are extracted from each video. The batch size is 16 (videos) and the learning rate starts from 0.01. For spatial stream, the learning rate is divided by 10 at iteration $10K$, $20K$ and $30K$, and training is stopped at $40K$ iteration. For temporal stream, the learning rate is divided by 10 at $20K$, $30K$ and $40K$, and training is stopped at $45K$ iteration.

Unlike GoogLeNet, Inception-ResNet-v2's default input size is 299×299. However, such a big input size will result in massive GPU memory requirements. Therefore, we random initialize the auxiliary tower and the classification layer, and then force the Inception-ResNet-v2 to train with 224×224 images. And we use RMSProp to optimize the model. Another issue of training Inception-ResNet-v2 with RNNs is that the batch normalization is highly relied on the mean and variance of each batch. When training Inception-ResNet-v2 with RNNs, each GPU (with 12G memory e.g. one core of Tesla K80) can only hold 32 images, which are two videos. In order to get reasonable mean and variance, we train the model with 8 GPUs, which is capable of 16 videos. Then we select 2 images from each video for each GPU and get 32 images. The hand-selected batches, which have 32 images, are respectively fed into 8 GPUs. After getting features from the last layer of CNNs (or the input projector), we rearrange the images to their original order to train the RNNs (or shuttleNet). Even though it is complicated to train RNNs (or shuttleNet) with Inception-ResNet-v2, the test stage is very easy and we are able to test with a batch size of 160 images without any modification.

Table 2. Accuracy and parameter number of shuttleNet with different settings on UCF101 split 1.

# processors	# steps	Spatial stream	Params
2	1	82.37%	10.49M
	2	83.03%	10.49M
	3	80.89%	10.49M
1	2	82.9%	6.29M
2		83.03%	10.49M
3		83.19%	14.68M
3	3	82.13%	14.68M

In the remainder of the paper, we use spatial stream and temporal stream to indicate the streams in two-stream framework and use "two streams" to indicate the late fusion of two models.

5.3. Exploration experiments

In this section, we will first test shuttleNet with different setting by adjusting the number of processors and the number of steps. Then, we compare our model with several existing RNN models in section 5.3.2 to prove the effectiveness of shuttleNet. We find that our shuttleNet can achieve much better performance with similar number of parameters. We also select base network by train shuttleNet with GoogLeNet and Inception-ResNet-v2 and compare their accuracy.

5.3.1 Parameter study

There are three hyper parameters in shuttleNet: the stride K, the number of processors N and the number of steps D. We will fix K as 1 in all of our experiments and explore the effect of other parameter by running experiments under multiple settings.

The performance of shuttleNet under different settings are shown in table 2. We find that as the number of steps growing, the performance of spatial stream first increases then decreases. As the number of processors growing, the spatial stream accuracy keeps increasing and parameter number grows quickly. Actually, the conclusions are easy to understand. When increasing the number of steps, each processor will have to take care of more situations. The more steps used, the harder for processors to learn. However, with 2 steps, it works like regularization approach preventing overfitting hence improving the performance. When increasing the number of processors, all processors work like model ensemble. The more processors used, the better the ensemble will be. However, more processors will result in more parameters hence increasing complexity.

Based on the above discussion, we will use 2 steps in the following sections. Although our shuttleNet can be easily extended as a multi-layer model, we use only one layer

Table 3. The accuracies of shuttleNet on two datasets.

Dataset	split	Spatial	Temporal	Two streams	Two streams + MIFS
UCF101	1	87.9%	86.3%	94.1%	95.1%
	2	87.1%	88.6%	94.6%	95.8%
	3	86.8%	87.4%	94.3%	95.2%
	Mean	87.3%	87.4%	94.4%	95.4%
HMDB51	1	55.6%	61.9%	69.0%	74.2%
	2	53.0%	58.9%	66.0%	71.2%
	3	53.9%	59.5%	64.8%	69.8%
	Mean	54.2%	60.1%	66.6%	71.7%

Table 4. Comparison of shuttleNet to several popular recurrent neural networks on UCF101 split 1. The second column is the number of the processors. The shuttleNets use 2 steps.

Model	#	Spatial	Temporal	Two Streams
GoogLeNet[†]	-	77.74%	79.75%	85.91%
LSTM	2	78.8%	80.15%	88.58%
GRU	2	80.49%	82.77%	90.27%
GRU	3	82.42%	81.21%	90.64%
shuttleNet	1	82.9%	82.92%	91.65%
shuttleNet[‡]	2	83.03%	82.18%	91.94%

[†] There is no RNN in GoogLeNet [38].

[‡] We use 1 group of optical flow fields as one sample for GoogLeNet instead of 10 as in [33, 45], which result in relatively worse performance than others'. But we can still get state-of-the-art result based on this implementation.

Table 5. Number of parameters in 2-layer RNN models with state size of 1024 and 1-layer shuttleNet without the input projector.

Model	LSTM	GRU	shuttleNet
Params	16.78M	8.39M	10.49M

Table 6. The mean accuracy of shuttleNet with GoogLeNet and Inception-ResNet-v2 on the three splits of UCF101.

	Spatial	Temporal	Merge
GoogLeNet	81.4%	84.9%	92.3%
Inception-ResNet-v2	87.3%	87.4%	94.4%

shuttleNet with 2 processors to compare with other models, for example, 2-layer LSTM or GRU, to emphasize the good performance and ensure fairness.

5.3.2 Comparison with baselines

As shown in Table 4, we compare the shuttleNet with several popular recurrent neural network models. The shuttleNet achieves a good performance of 91.94% on UCF101 split 1 while none of other models close to 91%. On the spatial stream, shuttleNet outperforms other RNN models remarkably. Even the three layer GRU network can not beat our 2-processor shuttleNet.

Unlike most of existing methods, who use a stack of 10 optical flow fields as one sample, all of our experiments use 1 group of optical flow fields as one sample to reduce complexity. Therefore we get relatively worse performance on the temporal stream and two-stream model. But we can still achieve the state-of-the-art performance after using shuttleNet. We believe we can boost the performance by using stack of 10 optical flow fields as input regardless of the complexity.

To prove the effectiveness of output selector, we conduct an experiment with a 2-processor-1-step shuttleNet. The accuracy of spatial stream is 82.37% and temporal stream accuracy is 82.76%. The two-stream shuttleNet achieves an accuracy of 91.7%. This means that output selector is very effective and can improve the performance significantly.

It's amazing that our 1-processor-2-step shuttleNet also achieve an accuracy of 91.65%, which is still much better than all other RNN models. The great performance of 1-processor shuttleNet proves the effectiveness of our processor-sharing mechanism.

The processor-sharing mechanism also makes sure that our model won't have too many parameters. As shown in Table 5, our model has slightly more parameters than GRU while having much less parameters than LSTM. The addition parameters are used to compute the attention value for each path. This means that no matter how many processors we have, there won't be any more parameters needed.

The overall potential of loop connection is fully demonstrated by the 2-processor-2-step shuttleNet. It outperforms all of these baselines and achieves a high accuracy of 91.94%. The good performance proves the advantage of shuttleNet.

Parallel computing is another potential advantage of our shuttleNet. Because that each processor in shuttleNet at every step works independently, it's easy to accelerate the computing by parallel computing all processors.

5.3.3 Selection of base network

The shuttleNet gets inputs from previous CNNs. It is important to select a better network. We test shuttleNet with GoogLeNet and Inception-ResNet-v2 and the results are shown in Table 6. The Inception-ResNet-v2 gets a 2% higher accuracy than GoogLeNet. The shuttleNet achieves state-of-the-art performance even using a relatively simpler network like GoogLeNet. And it achieves much better performance by using Inception-ResNet-v2. In the rest of the paper, we will use Inception-ResNet-v2 as our base network and report the accuracies.

5.4. Evaluation

In this section, we will first evaluate shuttleNet on more datasets and report their accuracy. Then we compare shuttleNet with several state-of-the-art methods in section 5.4.2.

5.4.1 Evaluation of shuttleNet

From section 5.3.2, we prove the effectiveness of shuttleNet on UCF101 split 1. We now report the performance of shuttleNet on more splits and dataset with Inception-ResNet-v2 in Table 3.

For the HMDB51 dataset, shuttleNet performs good on the first split while relatively worse on the other two splits. And the overall performance is not as good as in UCF101. We believe this is because of the small training set size (3570 videos). But shuttleNet is still outstanding on HMDB51 when comparing to other methods. On UCF101 dataset, the shuttleNet performs good on all three splits.

In most cases, the temporal stream achieves better performance than the spatial stream. This proves that the motion information plays an important role in action recognition.

Compared to pure CNN models, our method adds new layers to the network, which can not be pre-trained on ImageNet. Training the new layers on small dataset, e.g. HMDB51, is easy to be over-fitting. In order to apply shuttleNet to small dataset, we merge shuttleNet with MIFS [23], which is a hand-crafted feature, using late fusion. The improvement proves the complementation between deep features and traditional features.

5.4.2 Comparison with the state-of-the-art methods

Finally, we compare against the state-of-the-art over all three splits of HMDB51 and UCF101. The results are summarized in Table 7, where we compare our method with both traditional approaches such as iDT and deep learning methods such as TSN [45].

Compared to the two-stream ConvNets [33], which is the most famous baseline method, we get around 12.3% and 7.4% improvements on HMDB51 and UCF101 datasets, respectively. These results are much better than traditional iDT [41] and MIFS [23]. Compared to TSN [45], the shuttleNet achieves better performance on both HMDB51 and UCF101.

Table 7. Comparison of shuttleNet to the state-of-the-art methods.

Model	HMDB51	UCF101
STIP+BoF [21]	23.0%	43.9%
DT+MVSV [4]	55.9%	83.5%
iDT+FV [41]	57.2%	85.9%
iDT+HSV [28]	61.1%	88.0%
MIFS [23]	65.1%	89.1%
Two-stream ConvNets [33]	59.4%	88.0%
F$_{ST}$CN [36]	59.1%	88.1%
TDD+FV [42]	63.2%	90.3%
TDD+iDT+FV [42]	65.9%	91.5%
MoFAP [43]	61.7%	88.3%
LTC [39]	64.8%	91.7%
sDTD [30]	65.2%	92.2%
Conv Fusion [11]	65.4%	92.5%
Three-stream TSN [45]	69.4%	94.2%
Ours	**71.7%**	**95.4%**

Even though we use a very simple framework and only replace RNNs with our shuttleNet, our model still outperforms these methods on UCF101 and HMDB51. The superior performance of our method demonstrates the effectiveness of shuttleNet. It's possible that we can still improve the performance by using 10 groups of optical flow fields as input or pre-train the new layers of shuttleNet on large dataset like YouTube-8M [2].

6. Conclusion

In this paper, we present a biologically-inspired deep network, shuttleNet, for action recognition. As demonstrated on two benchmark datasets, shuttleNet can outperform LSTMs and GRUs remarkably while having roughly equal number of parameters. This is largely ascribed to the loop connections as well as processor-sharing mechanism. In the future, there are still a lot of research directions that will be addressed, for example, to check the performance of the stacked shuttleNet and whether CNNs will benefit from loop connection. For more comprehensive evaluation of our approach, we will conduct experiments on the THUMOS challenge dataset [14] and ActivityNet dataset [16, 10].

Acknowledgement. This work is partially supported by grants from the National Basic Research Program of China under grant 2015CB351806, the National Natural Science Foundation of China under contract No. 61650202, No. U1611461, and No. 61425025, also supported by grants from NVIDIA and the NVIDIA DGX-1 AI Supercomputer.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al.

Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467*, 2016.

[2] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv:1609.08675*, 2016.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[4] Z. Cai, L. Wang, X. Peng, and Y. Qiao. Multi-view super vector for action recognition. In *CVPR*, 2014.

[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[7] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[8] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, volume 400, page 409. Citeseer, 1995.

[9] A. Erişir, S. C. Van Horn, and S. M. Sherman. Relative numbers of cortical and brainstem inputs to the lateral geniculate nucleus. *PNAS*, 94(4), 1997.

[10] B. G. Fabian Caba Heilbron, Victor Escorcia and J. C. Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[11] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[12] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1), 1991.

[13] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *IJCNN*, volume 3. IEEE, 2000.

[14] A. Gorban, H. Idrees, Y.-G. Jiang, A. Roshan Zamir, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. `http://www.thumos.info/`, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[16] F. C. Heilbron and J. C. Niebles. Collecting and annotating human activities in web videos. In *Proceedings of International Conference on Multimedia Retrieval*, page 377. ACM, 2014.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[19] Y. Jiang, J. Liu, A. R. Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. Thumos challenge: Action recognition with a large number of classes. In *ECCV Workshop*, 2014.

[20] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. Hudspeth. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[21] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *ICCV*, 2011.

[22] V. A. Lamme, H. Super, and H. Spekreijse. Feedforward, horizontal, and feedback processing in the visual cortex. *Current opinion in neurobiology*, 8(4), 1998.

[23] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. In *CVPR*, 2015.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[25] S. L. Macknik and S. Martinez-Conde. The role of feedback in visual attention and awareness. *Cognitive Neurosciences*, 2009.

[26] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*, 2014.

[27] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[28] X. Peng, L. Wang, X. Wang, and Y. Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice. *CVIU*, 2016.

[29] S. M. Sherman and R. Guillery. The role of the thalamus in the flow of information to the cortex. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 357(1428), 2002.

[30] Y. Shi, Y. Tian, Y. Wang, and T. Huang. Sequential deep trajectory descriptor for action recognition with three-stream cnn. *TMM*, 2017.

[31] Y. Shi, W. Zeng, T. Huang, and Y. Wang. Learning deep trajectory descriptor for action recognition in videos using deep neural networks. In *ICME*, 2015.

[32] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *COLT*. ACM, 1992.

[33] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[35] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv:1212.0402*, 2012.

[36] L. Sun, K. Jia, D.-Y. Yeung, and B. E. Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *ICCV*, 2015.

[37] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv:1602.07261*, 2016.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[39] G. Varol, I. Laptev, and C. Schmid. Long-term temporal convolutions for action recognition. *arXiv:1604.04494*, 2016.

[40] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *CVPR*, 2011.

[41] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[42] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *CVPR*, 2015.

[43] L. Wang, Y. Qiao, and X. Tang. Mofap: A multi-level representation for action recognition. *IJCV*, 2015.

[44] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao. Towards good practices for very deep two-stream convnets. *arXiv:1507.02159*, 2015.

[45] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: towards good practices for deep action recognition. In *ECCV*. Springer, 2016.

[46] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *ACM MM*, 2015.

[47] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[48] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l 1 optical flow. In *PR*. Springer, 2007.